CS297 Report

Schemes to make Aries and XML work in harmony

Advisor: Dr Chris Pollett

Reporter: Thien An Nguyen
(thien_an9@yahoo.com)

Contents

1. Introduction

XML documents are semi-structured. They are stored as a tree with labeled edges and leaves. Storing such structured data in a relational database might result in either a large number of columns with null values (which wastes space) or a large number of tables (which is inefficient for searching and retrieving data). The goal of this project is to build a stripped-down toy XML database database and test out various ARIES-based recovery algorithms for XML data. This semester we defined the syntax for our DML and built a parser for our DML syntax in Deliverable 1 and 2. This DML is used as the communication bridge between a user application and our database. Having built the communication bridge, we designed the structure for our database and used JUnit to build test cases to test for the code we've implemented so far.

Storing XML natively offers better performance in terms of retrieving and accessing XML document, because native XML database understands the structure of XML documents and so preserves their data hierarchy and meaning. Native XML databases store an entire document together physically or use physical pointers between parts of the document. This allows the document to be retrieved either without joins or with physical joins, both of which are faster than the logical joins used by relational databases. [Bourret03]

Consider the XML document in Example 1 below. In a relational database, the document might be stored in three tables: Billing, Customers, and Parts. Retrieving the document would require joins across these tables. In a native XML database, the entire document might be stored in a single place on the disk. Thus it requires a single lookup and a read to retrieve the data.

Example 1: Following is an XML example:

```
<Billing BNumber="12345">
<Customer CustId="0021">
  <CustName>Surgical Optics ltd</CustName>
  <Street>947 Mission St.</Street>
```

```
    <City>San Jose</City>
    <State>CA</State>
    <PostCode>95054</PostCode>
  </Customer>
  <OrderBy>May 01, 2004</OrderBy>
  <ShippedBy>May 02, 2004</ShippedBy>
  <Part PNumber="000-675-2311">
      <Description>
          Objective Lens System
      </Description>
      <Price>119.95</Price>
      <Quantity>5</Quantity>
  </Part>
  <Part PNumber="000-232-1000">
      <Description>
         Ocular Lens
      </Description>
      <Price>111.00</Price>
      <Quantity>9</Quantity>
   </Part>
</Billing>
```

Algorithms for Recovery and Isolation Exploiting Semantics (ARIES) is a family of locking, logging and recovery algorithms for persistent data management. All update transactions are logged, including those performed during rollback. Through appropriate chaining of the log records written during a rollback to those written during forward progress, a bounded amount of logging is ensured during the rollback even in the case of repeated failures during restart or of nested rollbacks. [Mohan90] ARIES is designed around redo history paradigm, i.e., during the redo phase, database sub-system performs all redo updates even for loser transactions.

NATIX, a native XML database management system has been recently developed (from scratch), introduces the idea of subsidiary logging and annihilator undo. Suppose a record is updated multiple times by the same transaction, causing a sub-tree to be added. In such case, the log size would be reduced if we log only the composite update operation as one. It is called subsidiary logging. Undo operations that imply undo of other operations following them in the log is called annihilators. To improve the performance in XML context, annihilator undo skips undo operations implied by the annihilators. For example,

update operation(s) to a record that has been created by the same transaction doesn't need to be undone when the transaction is aborted because the record will be deleted anyway [Kan03]. NATIX, on the other hand, supports selective redo and selective undo to accelerate restart recovery.

The goal that I am aiming to accomplish in this CS297 is to build a stripped-down educational test version of a native XML database with features that allow a user to insert, load, update, delete from the XML tree structure. This goal is slightly different from that presented in our original CS297 proposal due to the fact that Predator and Shore seem to now require incompatible and obsolete versions of gcc. Our toy database will be used to test the Recovery Manager techniques implemented in NATIX. For my CS298, I will implement ARIES and the NATIX extensions given in the research paper "The Core Technologies for Native XML Database Management System" [Kan03] in my toy database.

We now outline the rest of this report: Section 2 briefly introduces NATIX's subsidiary logging and annihilator undo. Section 3 gives details for the SHORE installation steps. Section 4 gives detail of our DML. Section 5 gives details of our database structure. Section 5 gives the brief explanation of the source code. Appendix A has the listing of our source code.

2 NATIX, A Native XML Database

To see the reasoning behind subsidiary logging and annihilator undo, this section briefly describes NATIX's storage manager and it's recovery manager.

2.1 Overall Structure

Below is a diagram from the NATIX whitepaper. It includes Storage Manager (responsible for storing and retrieving data pages to and from physical storage), Index Manager (for indexing XML tree), Tree Storage Manager (for mapping tree structured

into records), Schema Manager (maintain Document Type Definitions and physical schema information and statistics), External Storage Managers, Document Manger (control access between applications to a document's node), and Query Engine. For details on NATIX storage and other components, please see NATIX [Kan03].

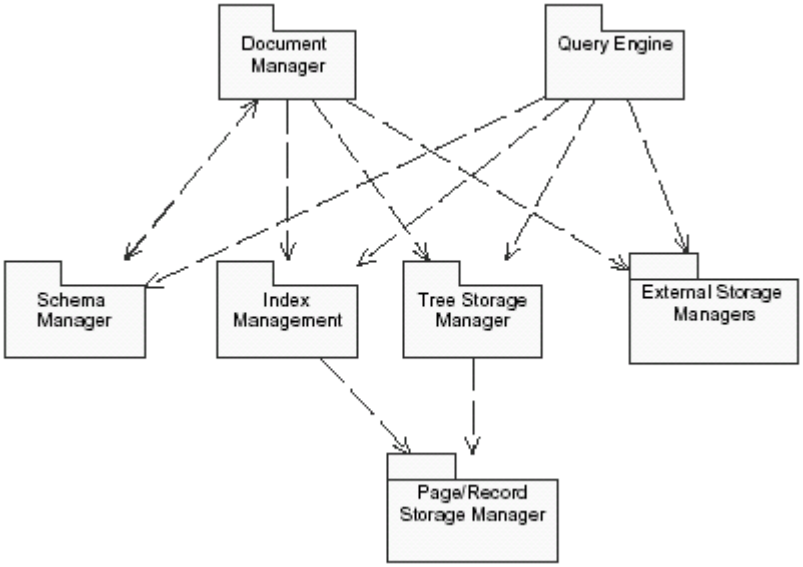

Figure 1 – NATIX Overall Architecture [Kan03]

2.2 Recovery Manager

The recovery algorithm in NATIX is based on ARIES [Mohan90] but a couple changes are required to fit it in with XML tree like structure. The following are the three phases the Recovery Manager performs during restart:

a. Analysis Phase: in this phase, the Recovery Manager scans through the log records, from the last check point to the point of system crash, to re-construct the transaction table and the dirty pages table at the point of system crash. It also builds a list of actions for rollbacks during undo phase.

b. Redo Phase: the Recovery Manager gets the last system Redo LSN from the Transaction Manager. If it sees that the log LSN is less than the page LSN (this implies

that the page has not been flushed to disk yet), it re-applies (redo) the log record to the page.

c. Undo Phase: during this phase, if un-committed transactions (loser transactions) are found in transaction table during analysis phase, the Recovery Manager will undo the changes.

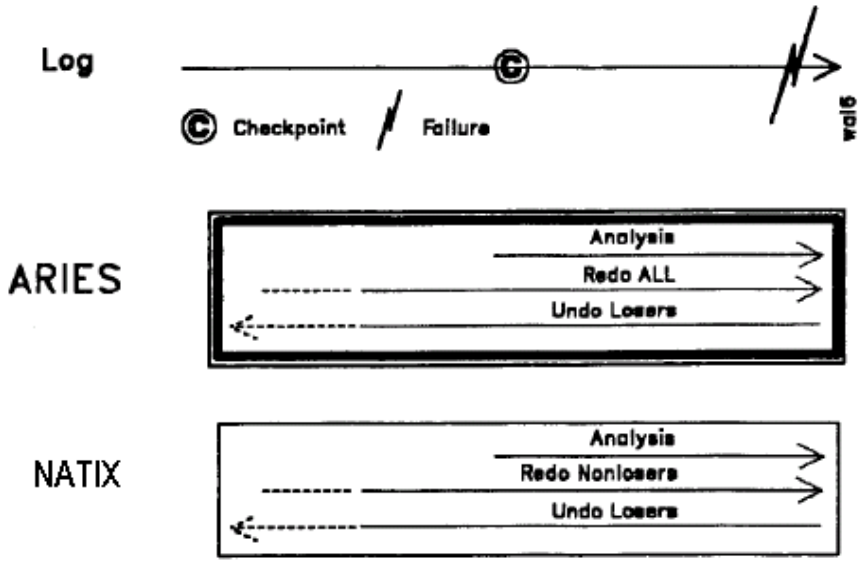

Figure 2 – Phases during Restart Recovery

Subsidiary Logging:

Database systems that support ARIES write log records to the Log Manager before the update operation is done on the page. The designers of NATIX believe that writing log records for every update by the same transaction, onto the same record in an XML page is redundant. For example, when nodes are added node by node to a sub-tree, these updates can be logged as one big composite operation (i.e. creates only one log record for the complete sub-tree insertion). By doing so, not only is the size of the logs and the time to process them are reduced but also the concurrency of the system is increased.

To implement this idea, NATIX's Page Interpreters keep private log records that are performed on the page, called subsidiary log. These log records then can be modified, re-ordered or pruned before they are published to the Log Manager.

The following examples and figures from NATIX's paper are presented to verify this idea. Consider operations performed by one transaction on the same sub-tree in a record as in Figure 3. Instead of writing 5 log records for 1, 2, 3, 4, and 5, NATIX can write 1 log record that describes the sub-tree R1 at the 5th step.

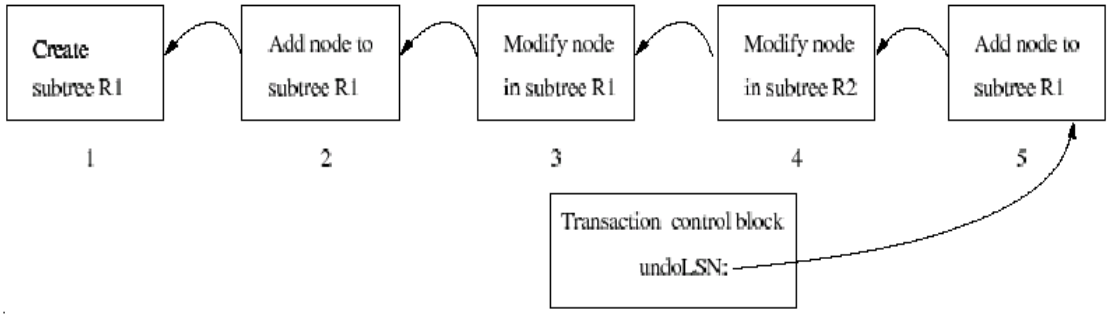

Figure 3 – Subsidiary Logging [Kan 03]

Annihilator Undo:

By definition, annihilators mean undo of operations that imply undo of other operations. For a rollback operation, updates to a record that were created by the same transaction need not be undone as reverse of update operations. Instead, it can be done by a single delete operation.

For example, consider the update operations in Figure 4. During a recovery, we perform the undo's of 5, 4, 3, 2, and 1 in the reverse order of the forward phase updates. On the other hand, if we undo the modification 4 and deletion 1, we would receive the same result as performed in the previous case, but with fewer operations which is an improvement in performance in terms of few operations to performs and increase concurrency.

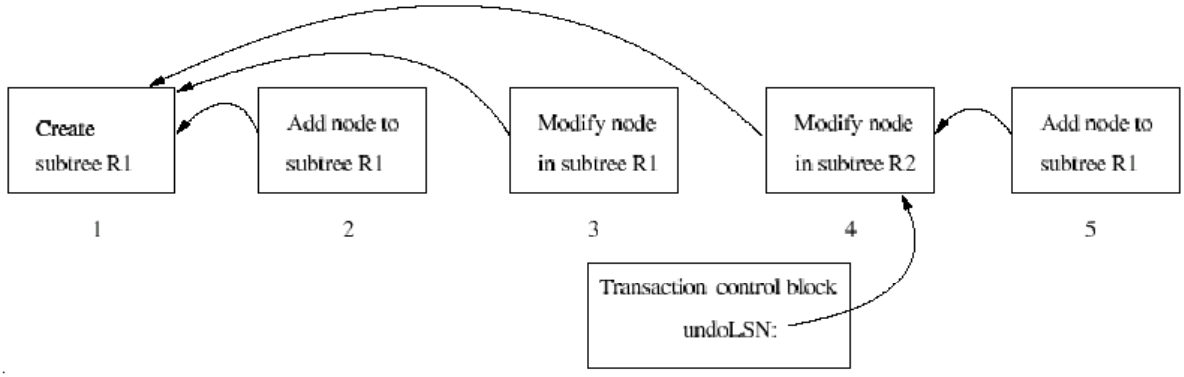

Figure 4 – Annihilator Undo [Kan03]

3 Deliverable 1 – Install Shore on Windows 2000

3.1 Description:

The advantages of the Recovery Manager in NATIX are very appealing. It does not only boost up the performance and concurrency for a database system but also simplifies management of log records. We want to implement these ideas in an open source object database as an experiment and for education purpose. We chose Predator, a relational-object database as our experimental tool. Storage manager in Predator is based on Scalable Heterogeneous Object Repository (SHORE). Thus, our first step is to install SHORE.

SHORE is an object storage engine that supports creation of persistent files and records. Each object in SHORE can be accessed using its globally unique OID. The two-phase commit protocol is used among servers. If a transaction updates data from it's own server's database, the transaction can be committed locally. Transaction rollback/abort or restart recovery algorithms are based on ARIES with variation extended for client/server architecture. Servers that own an object also create log records for the object and perform rollback/recovery. A client that caches the object will also generate log records for the object but will send log records back to server that owns the object. SHORE supports log-based recovery based on ARIES mechanism.

NATIX - native XML database management system - introduces the idea of subsidiary logging and annihilator undo. The motivation behind these ideas is to reduce the log size by:

1. logging only the composite update operations as one operation, and
2. annihilator undo skipping undo operations implied by the annihilators.

These ideas, if successful, will improve the performance on recovery process.

3.2 Installation Steps:

The following are the steps to install SHORE:

1. Download SHORE source code from http://www.cs.wisc.edu/shore/
2. Download and install cygwin from http://cygwin.com
3. Download and install perl from http://www.perl.com or ActivePerl
4. Download and install tcl from http://www.scriptics.com
5. Install Visual C++ 6.0

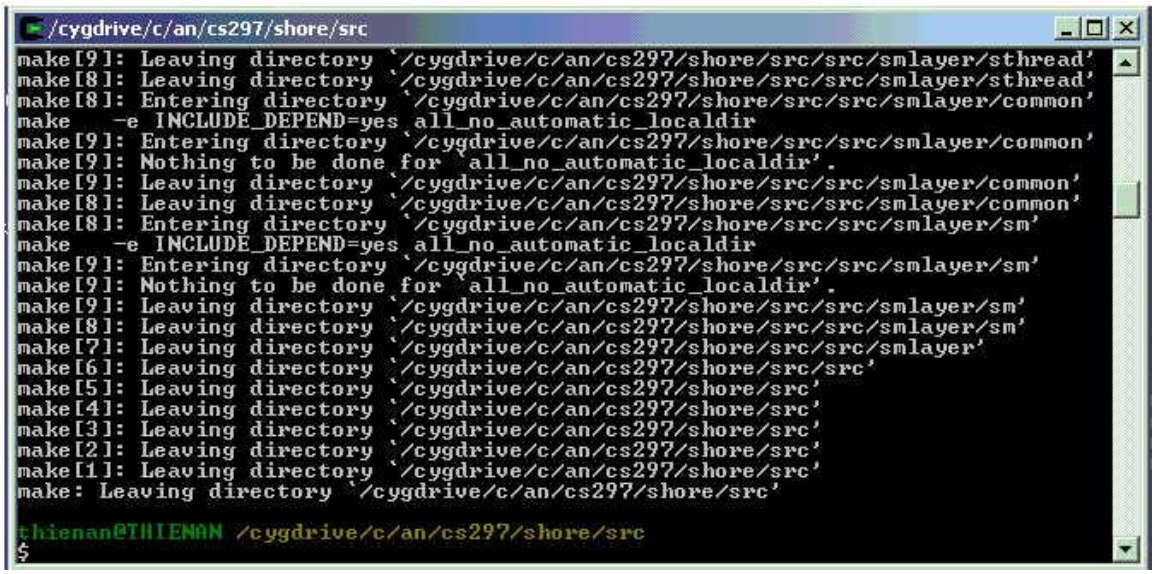

Figure 5 – Deliverable 1

6. Set PATH environment variables:

6a. Right click on My Computer -> properties -> Advanced -> Environment variables. Select PATH, click Edit. On the popup dialog, enter in the value field with:

C:\Tcl\bin;C:\Program Files\Microsoft Visual Studio\VC98\bin;
c:\cygwin\bin;c:\cygwin\lib;c:\perl\bin;c:\ActivePerl\Perl\bin; c:\Tcl\include;
6b. Add C:\Program Files\Microsoft Visual Studio\VC98\include to INCLUDE variable
6c. Add C:\Program Files\Microsoft Visual Studio\VC98\lib to LIB variable
6d. Set MAKE_MODE system variable to UNIX
7. Use winzip to unzip src-2.0.tar.gz into ./cs297/shore/src
8. Edit shore.def to set up Tcl path
9. Open cygwin window, cd to c:/an/cs297/shore/src. At the command line, type make
10. After the build finish, type make intall at the commandline.

4 Deliverable 2 - Data Manipulation Language (DML)

4.1 Description:

At the time of the Deliverable 2, we experienced problems in compiling the latest SHORE and Predator's source code as they required obsolete versions of gcc. In addition, the current version of Predator is based on an older version of SHORE which makes it even harder to get something working. As explained by a system administrator at SHORE, new versions of the source code that works for gcc 3.2 will not be available before summer. Our decision then was to build our own, simple native XML database from scratch for the remainder of the semester and implement the Recovery Manager in CS298.

In this deliverable, we introduce a simple version of a query language to manipulate data in our database.

The Data Manipulation Language (DML) enables users to access and manipulate data in a database. A query language is used to retrieve data from a database. In our project, as in SQL, the DML and query language are combined into one language. Figure 6 is an example of our DML.

4.2 Syntax and Description:

The following is a description of available statements in the DML:

1. CREATE DATABASE database-name;

This statement creates a database called database-name. database-name must start with a character or an underscore. The statement has to end with a semicolon.

2. CREATE XMLDOC xml-document-name IN database-name;

This statement creates an xml document with xml-document-name in database-name. xml-doc-name must start with a character or an underscore. database-name must be already exist in the system.

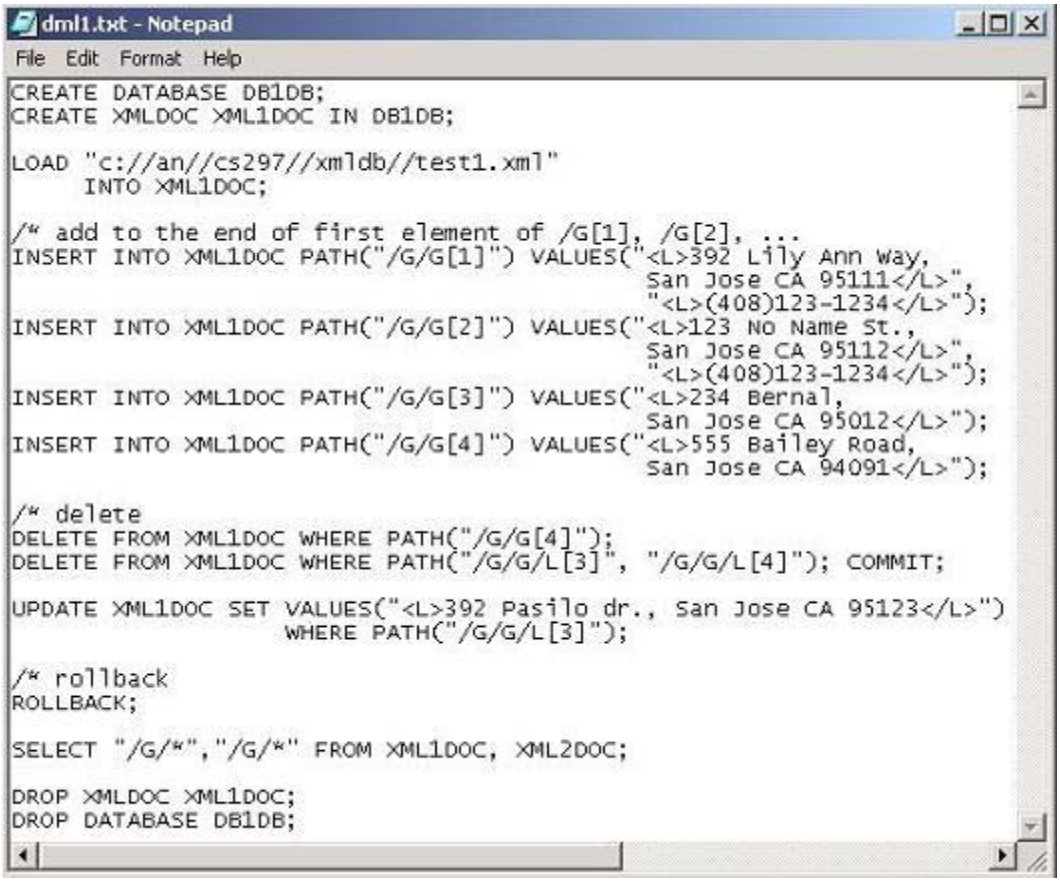

Figure 6 – DML Example

3. LOAD "file-name" INTO xml-document-name;

This statement loads data from a flat file file-name into create xml document xml-document-name. Format of the flat file could be as below, in which there are 3 types of tags:

a. <XML> </XML>: these tags indicate the beginning and the ending of an XML document. <XML can be followed by an attribute and attribute value, separated by an equal sign.

b. <G> </G>: these tags are inside XML tags. <G> </G> tags can be nested with <G></G> and/or <L></L> tags. <G> can have attribute and attribute value separate by an equal sign (like XML tag).

c. <L> </L>: these tags are the lowest level in our xml document. These tags start and end text data.

4. INSERT INTO xml-document-name PATH("location-path") VALUES("<L>leave-text-data-1</L>", ..., "<L>leave-text-data-n</L>");

This statement inserts n leave text nodes into specified location-path.

At this time, we support only absolute location-path. An absolute location-path starts with a '/' and followed by one or more location steps separated by a '/'. A location path can be used as an expression. Evaluation of the expression returns a set of nodes selected by the path.

Each location step can contain a filter predicate. A filter predicate is put inside open and close brackets ([ ]). It filters the set of nodes into a new set of nodes.

5. DELETE FROM xml-document-name WHERE PATH("location-path");

This statement will delete the node returned from evaluation of the location-path.

6. UPDATE xml-document-name SET VALUE("<L>text-data-1</L>") WHERE PATH("location-path");

This statement updates the node returned by the location-path with value inside VALUES("...,...")

7. SELECT "expression-1", ..., "expression-n" FROM xml-document-1, ..., xml-document-n, ...;

This statement queries the xml document

8. DROP XMLDOC xml-document-name;

This statement drops xml-document-name from current database system.

9. DROP DATABASE database-name;

This statement drops the current database-name from the database system.

10. COMMIT;

This statement commits the changes by the current transaction. The more often the user commit, the faster for the system to recover during restart recovery.

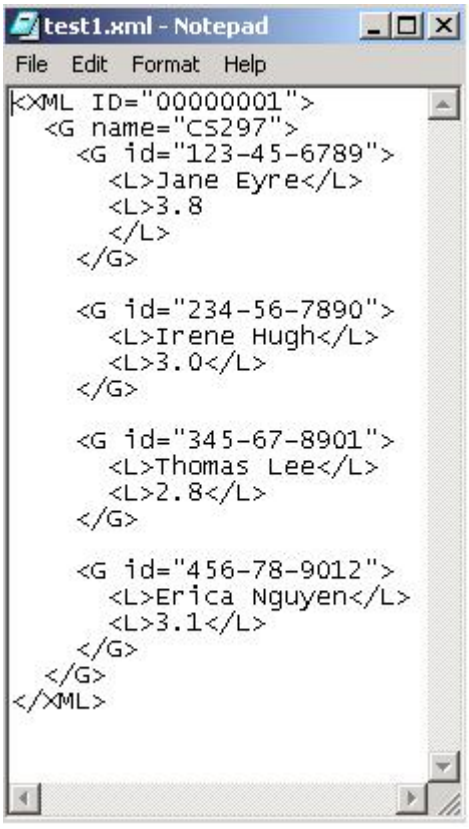

Figure 7 - Our XML Document Example

11. ROLLBACK;

This statement is used to undo the changes since the last commit or changes made by current transaction.

12. /*

This is used for comments. Despite its similarity to a C start comment, our comment only comments out the rest of the current line. For a comment that spans several lines, "/*" has to be placed at the beginning of every line to be commented.

5 Deliverable 3 - Our Native XML Database Overview

5.1 Description:

In this deliverable, we introduce our database architecture with concentration on logical and physical storage representations. Our database consists of Document Manager, Query Engine, Tree Storage Manager, Storage Manager, and Recovery Manager.

The Storage Manager is responsible for storing document trees on disks or files and retrieving data back into memory. For simplicity, a page is our storage unit. When a specific page is accessed, our sub-system will first look for the page in the buffer pool in main memory. If the page is not found in memory, the Storage Manager performs fetch operation and loads the requested page and the two pages that follow into memory. Below are brief explanations for our components:

Tree Storage Manager: Tree Storage Manager is as in NATIX, it maps trees used to model documents into pages.

Document Manager: Document Manger provides API for applications to access XML documents.

Query Engine: Query Engine analyzes queries (based on our DML) and returns appropriate result(s) or action(s).

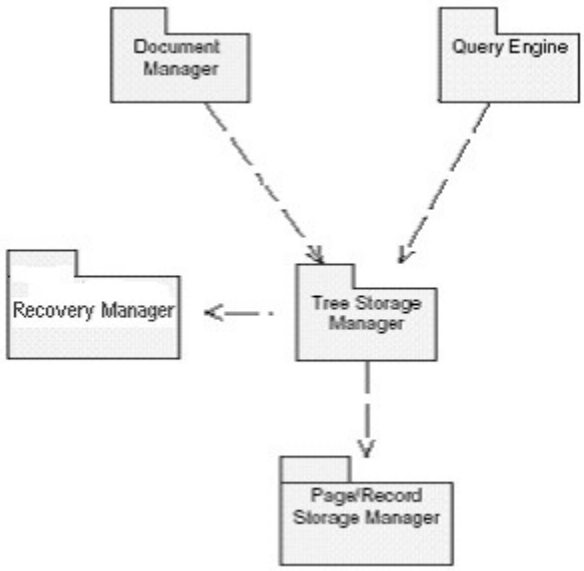

Figure 8 – Our Overall Structure

5.2 Storage Manager

The Storage Manager is composed of Logical and Physical parts as describe below:

5.2.1. Logical Model

Like NATIX, we map XML Elements tree nodes one-to-one onto our logical data model. Attributes are mapped to child nodes. However, for simplicity sake, our XML document contains only 3 types of tags and will be discussed in more details in deliverable 2.

Example 4: Following is our possible XML document:

```
<XML ID="00000001">
  <G name="CS297">
    <G id="123-45-6789">
      <L>Jane Eyre</L>
    </G>
    <G id="234-56-7890">
      <L>Irene Hugh</L>
      <L>3.0</L>
    </G>
</G>
</XML>
```

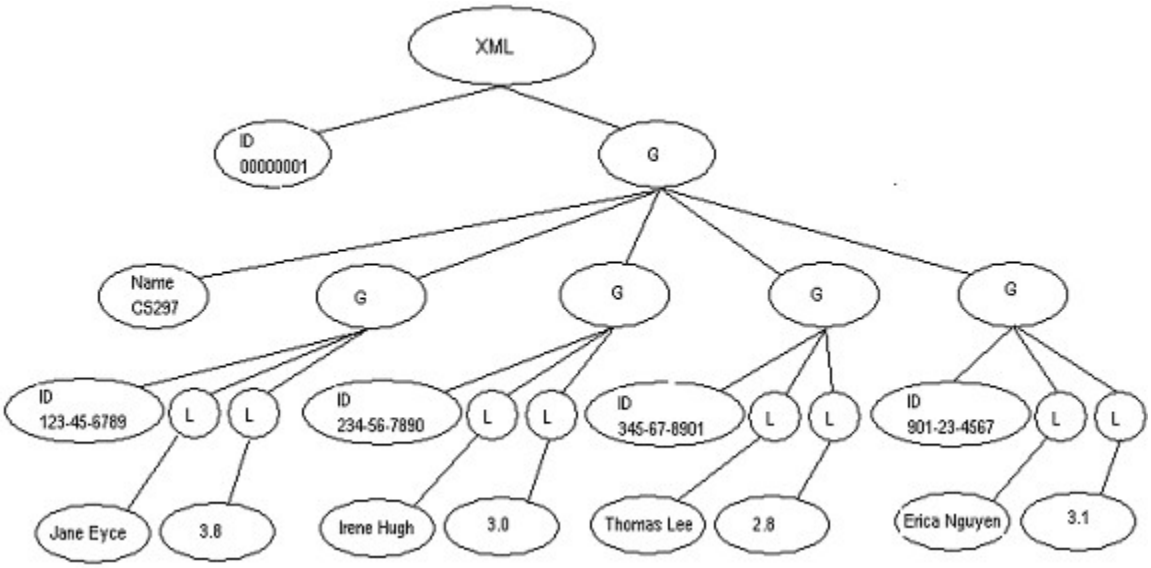

Figure 9 – Logical Tree Associate with Its XML Document

The logical data tree is materialized as a physical data tree and stored on disk(s). A detail structure of a physical page is shown on Figure 14. The page contains a page number, a page header followed by the root node of the sub-tree, and the list of its children. At the end of the page is a list of pointers to reusable nodes in the page. When inserting nodes into a tree results in a page overflow, we split the data tree into sub-trees. This results in the root page of the physical tree containing the Left page number and the Right page number instead of proxy nodes as in NATIX.

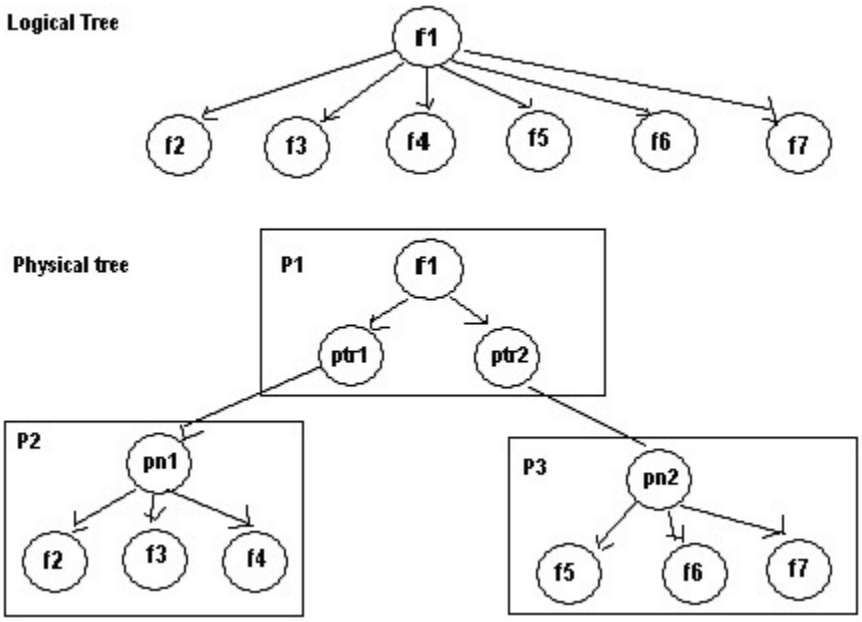

Figure 10 – Our Physical tree associate with Logical tree

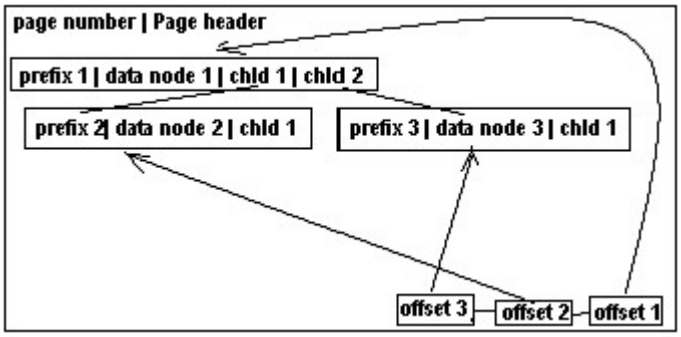

Figure 11 - Location map to a node in a record

5.3. Recovery Manager: we plan to implement our Recovery Manager based on NATIX in the Fall, 2004.

Here is the general flow of our database system:

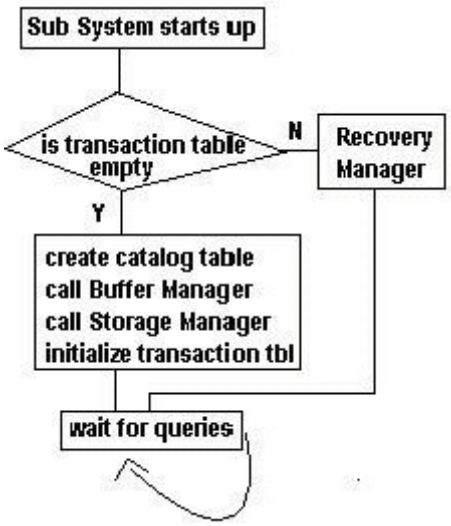

Figure 12 – Overall Control Flow

The following is a brief description of our components:

Buffer Manager: we implement Steal/No force for page management and simplified version of Least Recently Use (LRU) page swapping for buffer management. For Steal/No force, the page will be written to disk when it is needed by other transactions or when the Buffer Manager performs page swap without checking the commit status of updates on the page.

Transaction Manager: this component contains a transaction table of active transactions. When a transaction commits, its entry in the transaction table will be removed.

Page management: In the header of every page, there is a flag to indicate if all the updates on this page have been committed. There is also a flag in every data field's prefix that indicate if the individual data field is committed and ready for reuse. The prefix also has a 4 byte field containing the length of the data field. At the bottom of the page is an ArrayList page map.

When a DML command is submitted, it is scanned, parsed, and then analyzed. Then, it is passed to an agent to execute.

JUnit test case: At each stage of coding, we also provide corresponding unit test cases so that when a new code is added, we can make sure the new code path does not regress the already working version. It also helps in debugging and diagnosing a specific error.

Unit test case source code

```
/**
 *Title:       The following class is a unit testcase that will start our database sub-system,
 *              read dml6.txt and execute statements in dml6.txt.
 *Date:       Spring 2004
 *Author:   Thien An Nguyen
 * Advisor: Dr. Chris Pollett
 */

package cs297.xmldb.unit.server;

import junit.framework.*;
import java.util.*;
import java.lang.*;
import java.io.*;
import cs297.xmldb.src.server.*;
import cs297.xmldb.src.cm.*;
import cs297.xmldb.src.sm.*;

public class ThTestExecuteDML extends TestCase
{
 public ThTestExecuteDML(String name)
 {
   super(name);
 }

 protected void setUp()
 {
 }

 public void testExecuteDML6()
 {
  System.out.println("\ntestExecuteDML6()");
  try{
/*  Initial database sub-system */
   ThSystem subSys = new ThSystem();

/*  start the database sub-system */
   subSys.start();

/* pass sub-system object to an agent so that it can access global objects */
   ThAgent agent = new ThAgent(subSys);
```

```
/*  Test Query Engine. Parse the DML file for DML syntax error */
   ThPlan plan = ThParser.compileDML("c://an//cs297//xmldb//dml6.txt");

/* Execute statements in the DML file*/
   int rc = agent.executeDMLPlan(plan);

/* If there turn code from executing the dml file is not 0, signal error*/

   if (rc > 0)
   {
    fail("ExecuteDML Error!");
   }
  }catch (Exception e){
   fail(e.toString());
  }
  }

  public static void main(String args[])
 {
   junit.textui.TestRunner.run(ThTestExecuteDML.class);
  }
}
```

Following is the DML6.TXT

```
CREATE DATABASE DB1DB;
CREATE XMLDOC XML1DOC IN DB1DB;

LOAD "c://an//cs297//xmldb//test1.xml"
    INTO XML1DOC;

printTree(XML1DOC);

/* add to the end of first element of /G[1], /G[2], ...
INSERT INTO XML1DOC PATH("/G/G[1]") VALUES("<L>392 Lily Ann Way,
                    San Jose CA 95111</L>",
                    "<L>(408)123-1234</L>");
INSERT INTO XML1DOC PATH("/G/G[2]") VALUES("<L>123 No Name St.,
                    San Jose CA 95112</L>",
                    "<L>(408)123-1234</L>"); COMMIT;
INSERT INTO XML1DOC PATH("/G/G[3]") VALUES("<L>234 Bernal,
                    San Jose CA 95012</L>");
INSERT INTO XML1DOC PATH("/G/G[4]") VALUES("<L>555 Bailey Road,
                    San Jose CA 94091</L>");
INSERT INTO XML1DOC PATH("/G/G[5]") VALUES("<L>1234 1st street,
                    San Francisco CA 95012</L>",
                         <L>(510)344-1010</L>");
INSERT INTO XML1DOC PATH("/G/G[6]") VALUES("<L>2234 2st street,
                    Santa Rosa CA 94323</L>",
                    <L>(808)463-1203</L>");
INSERT INTO XML1DOC PATH("/G/G[7]") VALUES("<L>3542 3st street,
                    Santa monica CA 95002</L>",
```

```
                    <L>(808)463-1203</L>");
/* comments
printTree(XML1DOC);

DELETE FROM XML1DOC WHERE PATH("/G/G[4]");
DELETE FROM XML1DOC WHERE PATH("/G/G/L[4]", "/G/G/L[3]");
printTree(XML1DOC);

UPDATE XML1DOC SET VALUE("<L>392 Pasilo dr., San Jose CA 95123</L>")
          WHERE PATH("/G/G/L[2]");
printTree(XML1DOC);

DROP XMLDOC XML1DOC;
DROP DATABASE DB1DB;
```

Here is the result from running the above testcase:

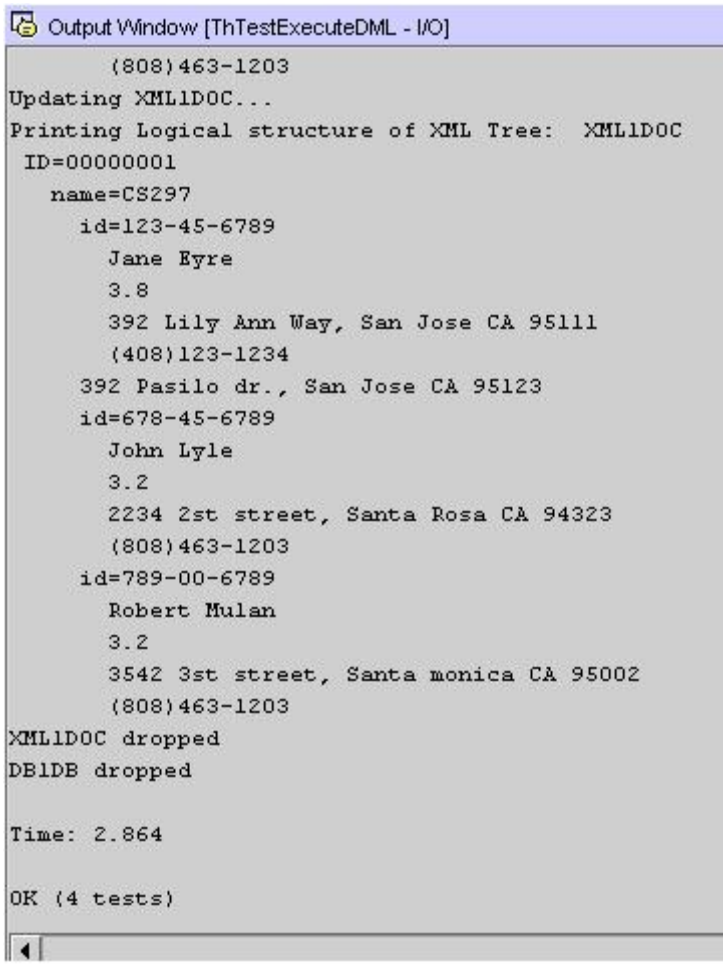

Figure 13 – Output Result

6 Deliverable 4 – Our XML Database Source code

6.1 Description:

In this deliverable, we implement our database design using JAVA. Reader can refer to Deliverable 3 for the architectural design.

The following is the source code for a logical leaf node. It contains a leaf id, page number in which it resides, and the actual data:

```
public ThLeave(long id, int pagenum, String dat)
 {
   iD = id;
   pageNum = pagenum;
   data = dat;
 }
```

Below is the logical Group node. It contain a node id, an attribute which can be null, and a list of children in the ArrayList:

```
public class ThGroup implements IthNode
{
  public long nodeId;
  public ThAttributeNode attribute;
  public ArrayList children;
```

Here is the structure of a document. It contains a reference to the database subsystem where global states of the database is kept, the document's name, the logical pointer to the root of the xml document's tree, and the physical page that will be flushed to disk when the page is full:

```
public class ThXMLDoc
{
….
  ThSystem subsys;
  public String name = null;
  public ThGroup root; //logical tree root
  public ThPage rootPage;
…
}
```

A database can contain 0 to n documents (if storage is enough). Following is the structure of a database. Its name is created from the DML file. The hash table contains the XML documents create in this database:

```
  public ThDB(String dbName)
  {
    name = new String(dbName);
```

```
    docNames = new Hashtable();
    dbUp = true;
  }
```

The following is the structure for the Buffer Manager. It consists of a hash table of pages in memory and it manages a free pages list. Buffer Manager also maintains the least recently used ArrayList of page number.

```
public class ThBufferMgr implements Serializable{
  public static int pSn = 0;   //page serial number
  public Hashtable bufferPool = new Hashtable();  //pages in buffer pool
  public ArrayList freeList = new ArrayList();
  public ArrayList leastRecentlyUse = new ArrayList();
  public int totalPage = 20; //20 pages in the pool
}
```

Please refer to the Appendix for code details.

7 Conclusion:

As XML continues to gain popularity over others as the way of sharing information over the Internet, the need for a database that stores XML document natively is also increased. For relational and hierarchy databases, they build a separate layer on top of their storage structure to store xml documents. For native xml databases, there are research papers in the area of storage manager, specifically in the memory and physical structure for storage manager, but very few papers study into the recovery manager. Storage structure is an important component of a database. Recovering data from a crash and/or abort transactions is also a critical feature of any database. The goal that I want to accomplish in CS298 is to build a recovery manager that has ARIES [Mohan4] combines with the ideas of subsidiary logging and annihilator undo from NATIX [Kan 03] on our native XML database.

In this semester, we are built stripped-down, toy native xml database to help us accomplish our goal in CS298. In Deliverable 1 and 2, we defined the syntax for our data manipulation language (DML). This is the communication bridge between user application and our database system. We built a parser parses queries from user application based on our DML syntax. In CS298, our parser will be a component on our client side. In Deliverable 3 and 4, we defined the overall architecture for our database system. This structure will be reused in CS298 as we will build the Recovery Manager as part of the Storage Manager. This structure is the core structure and will be on our server side.

References:

[Reichardt] Mark E. Reichardt, “XML's Role in the GeoSpatial information Revolution”.

[Bourret03] Ronald Bourret, “XML and Databases”, 2003.

[Kan03] Core Technologies for Native XML Database Management System. Carl-Christian Kanne. Mannheim. 2003.

[W3C] “Namespaces in XML”, World Wide Web Consortium 14-January-1999

[Mohan90] ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, P. Schwarz. ACM Transactions on Database Systems, 1990.

[Seshadri97] PREDATOR: An OR-DBMS with Enhanced Data Types. P. Seshadri, M. Paskin. SIGMOD, 1997, p. 568-571.

[Venkatarman96] Global Memory Management for Multi-server Database Systems. S. Venkatarman. University of Wisconsin - Madison, 1996.

[Mohan92] ARIES/KVL: A Key-Value Locking Method for Concurrency Control of Multiaction Transactions Operating on B-Tree Indexes. C. Mohan. In Proc. of the 12th International Conference on Distributed Computing Systems, Yokohama, 1992.

[Mohan99] Repeating History Beyond ARIES. C. Mohan. Proc. 25th International Conference on Very Large Data Bases, Edinburgh, 1999.

[Molesky95] Recovery Protocols for Shared Memory Database Systems. L. Molesky, K. ramamritham. ACM, 1995.

[Lahiri01] Fast-Start: Quick Fault Recovery in Oracle. T. Lahiri, A. Ganesh, R. Weiss, A. Joshi. ACM SIGMOD, 2001.

[Mohan93] ARIES/LHS: A Concurrency Control and Recovery Method Using Write-Ahead Logging for Linear Hashing with Separators. C. Mohan. Proceedings of the 9th IEEE International Conference on Data Engineering, 1993.

Shore SM source code available at: ftp://ftp.cs.wisc.edu/shore/current/

Predator source code available at http://www.distlab.dk/predator/download.htm

Appendix: Source code for the project

----------interface IThNode----------

```
package cs297.xmldb.src.cm;

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

/**
 * This is interface for ThGroup and ThLeave
 * Date: Spring 2004
 * @author: Thien An Nguyen,
 */
public interface IthNode
{
  public IThNode getChild(int idx);
  public void insert(IThNode node);
}
```

----------class ThGroup----------

```
package cs297.xmldb.src.cm;

import java.io.*;
import java.util.*;
import cs297.xmldb.src.cm.*;

public class ThGroup implements IthNode
{
  public long nodeId;
  public ThAttributeNode attribute;
  public ArrayList children;

   public ThGroup(String attr, String dat)
   {
     attribute = new ThAttributeNode(attr,dat);
     children = new ArrayList();
   }

  public ThGroup(ThAttributeNode attrnode)
  {
    attribute = attrnode;
    children = new ArrayList();
  }

  public IThNode getChild(int idx)
  {
    return (ThGroup) children.get(idx);
  }

  public Object getChildren()
  {
    return children;
  }
```

```
  public void insert(IThNode node)
  {
  }
}
```

----------class ThLeave----------

```
package cs297.xmldb.src.cm;

public class ThLeave implements IThNode{
 public long iD;
 public int pageNum;
 public int offSet;
 public String data;

 public ThLeave()
 {
 }

 public ThLeave(long id, int pagenum, String dat)
 {
   iD = id;
   pageNum = pagenum;
   data = dat;
 }

  public IThNode getChild(int idx)
  {
    return null;   // no more child
  }

  public void insert(IThNode node)
  {
  }
}
```

----------class ThParser----------

```
package cs297.xmldb.src.cm;

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

/**
 * Date: Spring 2004
 * @author: Thien An Nguyen,
 */
public class ThParser {
 /* parse the DML plan to break the plan into statements
  * Each statement is terminated by ';'
  * @params: name of the flat file
  * @returns: ArrayList of statements
  */
 public static ArrayList scanDML(String fileName){
  ArrayList cmdList = new ArrayList();
  try{
```

```
   DataInputStream in = new DataInputStream(
                        new BufferedInputStream(
                          new FileInputStream(fileName)));
   /*read in line by line*/
   String inLine = new String();
   StringBuffer cmd = new StringBuffer();

   while((inLine = in.readLine())!=null){
    if (inLine.indexOf("//*") >= 0){ // this is a comment
      ;                              // do nothing
    }else if (inLine.indexOf(';') < 0){
      cmd.append(inLine);
    }else{                           //one or more ';' found on the same line
      int idx = inLine.indexOf(';');
      while (idx > 0){               //ignore if ';' is at the beginning of the stmt
        cmd.append(inLine.substring(0,idx));
        cmdList.add(cmd);            //append the first stmt
        cmd = new StringBuffer();

        inLine = new String(inLine.substring(idx+1, inLine.length()));
        idx = inLine.indexOf(';');
      }
    }
   }
   in.close();
 }catch(Exception e){
   e.printStackTrace();
 }
 return cmdList;
}

/**
 *  This function checks to see if a word is exist in an ArrayList
 *  @params: the word (String) to be checked, the ArrayList of words
 *  @returns: true if the word is exist inside the list
 *                false if it is not exist
 */
public static boolean isExist(String word, ArrayList list){
  for (int i=0; i<list.size();i++){
    if (word.equals(list.get(i))){
      return true;
    }
  }
  return false;
}

/**
 * This function creates an error statement
 * @receive: the error message, and list of original parameters
 * @return: The ThStatement object that contains the error message and original parameters
 */
public static ThStatement errStatement(String errmessage, ArrayList params)
{
  return new ThStatement(errmessage, params, false);
}
```

```
/**
 *
 * @receive:
 * @return:
 */
public static ThStatement parseSelect(StringTokenizer line)
{
 ArrayList stmts = new ArrayList();
 ArrayList params;
 String word;
 ThStatement stmt = null;
 params = new ArrayList();

   if (line.hasMoreTokens()){
     word = line.nextToken();
     ArrayList paths = new ArrayList();
     paths.add(word.substring(word.indexOf("")+1,
                        word.lastIndexOf("")));
     params.add(paths);
     if (line.hasMoreTokens()){
       word = line.nextToken();
       if (word.equals("FROM")){
         if (line.hasMoreTokens()){
           word = line.nextToken();
           ArrayList docs = new ArrayList();
           docs.add(word);
           params.add(docs);
           stmt = new ThStatement("SELECT", params, true);
         }else {
           stmt = errStatement("Missing Document Name!", params);
         }
       }
     }else{
       stmt = errStatement("In-complete SELECT Statement!", params);
     }
   }else{
     stmt = errStatement("In-complete SELECT Statement!", params);
   }

   return stmt;
}

/**
 *
 * @receive:
 * @return:
 */
public static ThStatement parseInsert(StringTokenizer line,
                                      ArrayList databases,
                                      ArrayList xmldocs)
{
 ArrayList stmts = new ArrayList();
 ArrayList params;
 String word;
 ThStatement stmt = null;
 params = new ArrayList();
```

```
if (line.hasMoreTokens()){
 word = line.nextToken();
 if(word.equals("INTO")){
  if (line.hasMoreTokens()){     //docname
    word = line.nextToken();
    if(isExist(word, xmldocs)){  //make sure docname already exist
      params.add(word);          //otherwise, flag an error
      if (line.hasMoreTokens()){ //path()
        word = line.nextToken();
        if(word.indexOf("PATH")>=0){
          params.add(word.substring(word.indexOf("")+1,
                              word.lastIndexOf("")));
          if (line.hasMoreTokens()){  //VALUES(,)
            word = line.nextToken();
            if (word.indexOf("VALUES")>=0){
              ArrayList values = new ArrayList();
              while(line.hasMoreTokens()){
                word += (" "+line.nextToken());
              }
              if(word.indexOf("\", ")<0){   //1 value added into specified path
                word = word.substring(word.indexOf("<L")+3,
                                word.lastIndexOf("</L"));
                values.add(word);
                params.add(values);
              }else{    //more than 1 values added to the specified path
                while (word.indexOf("<L") >= 0){
                  int idx = word.indexOf("</L");
                  String tmp = word.substring(word.indexOf("<L")+3,idx);
                  word = word.substring(idx+4);
                  values.add(tmp);
                }
                params.add(values);
              }
              stmt = new ThStatement("INSERT", params, true);
            }else{
              stmt = errStatement("INSERT", params);
            }
          }else{
            stmt = errStatement("Missing VALUES Clause!", params);
          }
        }else{
          stmt = errStatement("Missing PATH", params);
        }
      }else{
      }
    }else{
      stmt = errStatement("MXLDOC "+word+" NOT FOUND!", params);
    }
  }else{
    stmt = errStatement("In-complete statement!", params);
  }
 }else{
  stmt = errStatement("Syntax Error! Missing INTO keyword.", params);
 }
}else{
```

```
      stmt = errStatement("In-complete statement!", params);
    }

  return stmt;
}

/**
 *
 * @receive:
 * @return:
 */
 public static ThStatement parseUpdate(StringTokenizer line)
 {
  ArrayList stmts = new ArrayList();
  boolean succ = false;
  ArrayList databases = new ArrayList();
  ArrayList xmldocs = new ArrayList();
  ArrayList params;
  String word;
  ThStatement stmt = null;
  params = new ArrayList();

    if (line.hasMoreTokens()){
      word = line.nextToken();
      params.add(word);
      if (line.hasMoreTokens()){
        word = line.nextToken();
        if (word.equals("SET")){
          if (line.hasMoreTokens()){
            word = line.nextToken();
            if (word.indexOf("VALUE")>=0){
              while(word.indexOf(')')<0){
                word += (" "+line.nextToken());
              }
              word = word.substring(word.indexOf("<L")+3,
                                    word.lastIndexOf("</L"));
              params.add(word);

              if (line.hasMoreTokens()){
                word = line.nextToken();
                if (word.indexOf("WHERE") >=0 ){
                  word = line.nextToken();
                  if (word.indexOf("PATH") <0 ){
                    stmt = errStatement("Update Missing path(s)!", params);
                  }else{
                    ArrayList values = new ArrayList();

                    while (true){
                      word = word.substring(word.indexOf("")+1,
                                            word.lastIndexOf(""));
                      values.add(word);
                      params.add(values);
                      if (line.hasMoreTokens())
                        word = line.nextToken();
                      else
                        break;
```

```
                }

                succ = true;
                stmt = new ThStatement("UPDATE", params, true);
               }
              }else{ //endif "where"
              }
             }else{
              stmt = errStatement("VALUES ERROR!", params);
             }
            }else{
             stmt = errStatement("UPDATE", params);
            }
           }else{
            stmt = errStatement("Incomplete statement!", params);
           }
          }else{
           stmt = errStatement("Incomplete statement!", params);
          }
         }else{
          stmt = errStatement("Incomplete statement!", params);
         }
        }else{
         stmt = new ThStatement("Incomplete statement!", params, false);
        }

 return stmt;
}

/**
 *
 * @receive:
 * @return:
 */
 public static ThStatement parseDelete(StringTokenizer line)
 {
  ArrayList stmts = new ArrayList();
  ArrayList params;
  String word;
  ThStatement stmt = null;
  params = new ArrayList();

    if (line.hasMoreTokens()){
      word = line.nextToken();
      if(word.equals("FROM")){
        if (line.hasMoreTokens()){
          word = line.nextToken();
          params.add(word);
          if (line.hasMoreTokens()){
            word = line.nextToken();
            if (word.equals("WHERE")){
              if (line.hasMoreTokens()){
                word = line.nextToken();
                if (word.indexOf("PATH")<0){
                  stmt = errStatement("Missing path(s)!", params);
                }else{
```

```
            ArrayList values = new ArrayList();

            while (true){
              word = word.substring(word.indexOf("")+1,
                                word.lastIndexOf(""));
              values.add(word);
              params.add(values);
              if (line.hasMoreTokens())
                word = line.nextToken();
              else
                break;
            }

            stmt = new ThStatement("DELETE", params, true);
          }
        }
       }
      }else {
        stmt = errStatement("Incomplete DELETE statement!", params);
      }
     }else {
      stmt = errStatement("Missing XmlDoc's name!", params);
     }
    }else{
     stmt = errStatement("Incomplete DELETE statement!", params);
    }
   }else {
    stmt = errStatement("Incomplete DELETE statement!", params);
   }
  return stmt;
 }

/**
 *
 * @receive:
 * @return:
 */
 public static ThStatement parseCreate(StringTokenizer line,
                    ArrayList databases, ArrayList xmldocs)
 {
  ArrayList stmts = new ArrayList();
  ArrayList params;
  String word;
  ThStatement stmt = null;
  params = new ArrayList();

   if (line.hasMoreTokens()){
    word = line.nextToken();
    if(word.equals("XMLDOC")){     //xmldoc
     if(line.hasMoreTokens()){
      word = line.nextToken();   //doc name
      params.add(word);
      xmldocs.add(word);
      if(line.hasMoreTokens()){
       word = line.nextToken();
       if(word.equals("IN")){
```

```
            if(line.hasMoreTokens()){
              word = line.nextToken();
              if (isExist(word, databases)){
                params.add(word);
                stmt = new ThStatement("CREATE XMLDOC", params, true);
              }else{
                stmt = errStatement("DATABASE "+word+" NOT FOUND!", params);
              }
            }else{
              stmt = errStatement("Missing Database's name", params);
            }
          }else{
            stmt = errStatement("Missing Database's name", params);
          }
        }else{
        }
      }else{
        stmt = errStatement("Missing the rest of the statement", params);
      }
    }else if(word.equals("DATABASE")){ //if database
      if(line.hasMoreTokens()){
        word = line.nextToken();
        params.add(word);
        databases.add(word);
        stmt = new ThStatement("CREATE DATABASE", params, true);
      }else{
        stmt = errStatement("Missing Database's name", params);
      }
    }else{
      stmt = errStatement("Unknown Object to create", params);
    }
  }else{  //if CREATE
    stmt = errStatement("CREATE DATABASE/XMLDOC?", params);
  }
  return stmt;
}

/**
 *
 * @receive:
 * @return:
 */
public static ThStatement parseLoad(StringTokenizer line,
                                    ArrayList databases,
                                    ArrayList xmldocs)
{
 ArrayList stmts = new ArrayList();
 ArrayList params;
 String word;
 ThStatement stmt = null;
 params = new ArrayList();

  if(line.hasMoreTokens()){
    word = line.nextToken();      // filename followed
    word = word.substring(word.indexOf("")+1,
                          word.lastIndexOf(""));
```

```
      params.add(word);
      if(line.hasMoreTokens()){
        word = line.nextToken();
        if(word.equals("INTO")){
          if(line.hasMoreTokens()){ //doc name
            word = line.nextToken();
            if(isExist(word, xmldocs)){
              params.add(word);
              stmt = new ThStatement("LOAD", params, true);
            }else{
              stmt = errStatement("MXLDOC "+word+" NOT FOUND!", params);
            }
          }else{
            stmt = errStatement("Missing document name!", params);
          }
        }else{
          stmt = errStatement("Missing INTO", params);
        }
      }else{
        stmt = errStatement("In-complete statement!", params);
      }
    }else {
      stmt = new ThStatement("In-complete statement!", params, false);
    }

    return stmt;
  }

  /**
   *
   * @receive:
   * @return:
   */
  public static ThStatement parseDrop(StringTokenizer line,
                                      ArrayList databases,
                                      ArrayList xmldocs)
  {
    ArrayList stmts = new ArrayList();
    ArrayList params;
    String word;
    ThStatement stmt = null;
    params = new ArrayList();

     if (line.hasMoreTokens()){
      word = line.nextToken();
      if(word.equals("XMLDOC")){
        if(line.hasMoreTokens()){
          word = line.nextToken();   //doc name
          params.add(word);
          xmldocs.remove(word);
          stmt = new ThStatement("DROP XMLDOC", params, true);
        }else{
          stmt = errStatement("Missing xmldoc's name!", params);
        }
      }else if (word.equals("DATABASE")){
        if(line.hasMoreTokens()){
```

```
          word = line.nextToken();   //database name
          params.add(word);
          databases.remove(word);
          stmt = new ThStatement("DROP DATABASE", params, true);
        }else{
          stmt = errStatement("Missing database's name!", params);
        }
      }else{
        stmt = errStatement("DROP command Incomplete!", params);
      }
    }else{
      stmt = errStatement("DROP command Incomplete!", params);
    }

   return stmt;
  }

/**
  *  This function will call function scan to scan through the input DML file
  * The return of the scan function is an arraylist of statements.
  * This function will go through each element in the ArrayList and parse for keywords
  * and check for syntax error.
  *  @params: name of the flat file
  *  @returns: ArrayList of statements
  */
  public static ThPlan compileDML(String testcase){  //scan
   System.out.println("Parsing DML...");
   ArrayList lst = scanDML(testcase);   //break DML plan into stmts

   //Now, check for syntax & encode the DML
   ArrayList stmts = new ArrayList();
   boolean succ = false;
   ArrayList databases = new ArrayList();
   ArrayList xmldocs = new ArrayList();
   ArrayList params;
   String word;
   ThStatement stmt = null;
   String filename = null;

   //for each stmt, go through each word in the stmt
   for (int i=0; i<lst.size(); i++){
     params = new ArrayList();
     String temp = ((StringBuffer)lst.get(i)).toString();
     StringTokenizer line = new StringTokenizer(temp);
     //parse through each statement from the lst
     while (line.hasMoreTokens()){
       word = line.nextToken();
       if (word.equals("SELECT")){         //SELECT
         stmt = parseSelect(line);
       }else if (word.equals("INSERT")){   //INSERT
         stmt = parseInsert(line, databases, xmldocs);
       }else if (word.equals("UPDATE")){   //UPDATE
         stmt = parseUpdate(line);
       }else if (word.equals("DELETE")){   //DELETE
         stmt = parseDelete(line);
       }else if (word.equals("CREATE")){   //CREATE
```

```
        stmt = parseCreate(line, databases, xmldocs);
      }else if (word.equals("LOAD")){
        stmt = parseLoad(line, databases, xmldocs);
      }else if (word.equals("COMMIT")){     //COMMIT
        succ = true;
        stmt = new ThStatement("COMMIT", params, true);
      }else if (word.equals("ROLLBACK")){   //ROLLBACK
        succ = true;
        stmt = new ThStatement("ROLLBACK", params, true);
      }else if (word.equals("DROP")){
        stmt = parseDrop(line, databases, xmldocs);
      }else if(word.indexOf("print") >= 0){
        params.add(temp.substring(temp.indexOf('(')+1,temp.indexOf(')')));
        stmt = new ThStatement("PRINT", params, true);
      }else{            // Syntax Error
        ThStatement obj = new ThStatement();
          //obj.param = -1;
      }//end if
    }//end while
    stmts.add(stmt);
  }//end for
  return new ThPlan(lst, stmts, succ);
 }
}
```

----------class ThStatement----------

```
package cs297.xmldb.src.cm;
import java.util.*;

/**
 * This class contains the statements that scanned
 * from a DML flat file and the statements after
 * being parsed.
 * Date: Spring 2004
 * @author: Thien An Nguyen,
 */
public class ThStatement
{
 public String sql;
 public ArrayList params; //i.e. filename, path, ...
 public boolean success = false;

 public ThStatement(String s, ArrayList p, boolean succ)
{
  sql = new String(s);
  params = p;
  success = succ;
 }

 public ThStatement()
{
 }
}
```

----------class ThXMLDoc----------

```
package cs297.xmldb.src.cm;
```

```
import java.io.*;
import java.util.*;
import cs297.xmldb.src.server.*;
import cs297.xmldb.src.bm.*;

/**
 * This class handle functions that relate to given xml document
 * Date: Spring 2004
 * @author: Thien An Nguyen,
 */
public class ThXMLDoc
{
  //Error return codes
  public static int MISS_MATCH_LEAVE = -4; // Missmatch <L>,</L>
  public static int MISS_MATCH_GROUP = -5; // Missmatch <G,</G>
  public static int MISS_MATCH_XML = -6;   // Missmatch <XML,</XML>
  public static int UNKNOWN_TAG = -7;      // Unknown tag
  public static int UNKNOWN_PATH = -8;     // Unknown PATH
  String spaces = " "; //for print Tree alignment
  ThSystem subsys;
  public String name = null;
  public ThGroup root; //logical tree root
  public ThPage rootPage;

  public ThXMLDoc(String xmlname, ThSystem subs)
  {
    name = new String(xmlname);
    root = new ThGroup("ROOT", "ROOT");
    subsys = subs;
    rootPage = subsys.BM.getPage();
  }

  public ThXMLDoc(ThXMLDoc subtree, ThSystem subs)
  {
    root = new ThGroup("ROOT", "ROOT");
    subsys = subs;
  }

 /**
 * return:  a list of objects selected
 * receive: expression path
 */
  public Object select(String path)
  {
    System.out.println("Selecting from XMLDoc: "+name);
    ArrayList resultSet = new ArrayList();
    //current position
    IThNode currPtr = (IThNode)((ThGroup)root).children.get(0);
    int idx;

    while (true)
    {
      if (path.startsWith("/G"))
      {
        path = path.substring(2);
```

```
      if (path.startsWith("[")){ //indication selection of children
        if (path.indexOf("*")>=0){  //all of the children
          resultSet = (ArrayList)((ThGroup)currPtr).getChildren();
        }
        else{              //indicate which child to be selected
          idx = Integer.parseInt(path.substring(path.indexOf("[")+1,
                                      path.indexOf("]")));
          resultSet.add(currPtr.getChild(idx-1));
          path = path.substring(path.indexOf("]")+1);
        }

        return resultSet;
      }
      else{ //by default, first child in the group
        currPtr = (IThNode)currPtr.getChild(0);
      }
    }
    else{
      System.out.println("Unknown Path!");
      break;
    }
  }

  return resultSet;
}

/*
 * receive:  the expression path
 * return:   return code indicate of execution success for fail
 */
public synchronized int delete(Object pat){
  System.out.println("Deleting from XMLDoc...");
  int rc = 0;
  ArrayList arr = (ArrayList) pat;
  for (int k=0; k<arr.size();k++)
  {
   //current position
   IThNode currPtr = (IThNode)((ThGroup)root).children.get(0);
   Stack stack = new Stack();
   int idx = 0;
   String path = (String) arr.get(k);

   while (true)
   {   //travese down the tree until the condition is matched
     if (path.startsWith("/G"))
     {
       path = path.substring(2);
       stack.push(currPtr);
       if (path.startsWith("["))
       { //indication of which child
         idx = Integer.parseInt(path.substring(path.indexOf("[")+1,
                        path.indexOf("]")));
         idx -= 1; // array idx starts from 0
         currPtr = (IThNode)currPtr.getChild(idx);
         path = path.substring(path.indexOf("]")+1);
```

```
      }
      else{ //by default, first child in the group
        currPtr = (IThNode)currPtr.getChild(0);
      }

      if (path.equals(""))
      {
        currPtr = (ThGroup)stack.pop();
        ((ThGroup)currPtr).children.remove(idx);
        break;
      }
    }
    else if (path.startsWith("/L")){
      path = path.substring(2);
      currPtr = (ThGroup)stack.pop();
      if (path.startsWith("["))
      { //indication of which child
        idx = Integer.parseInt(path.substring(path.indexOf("[")+1,
                        path.indexOf("]")));
        idx -= 1; // array idx starts from 0
        ((ThGroup)currPtr).children.remove(idx);
      }
      else{ //by default, first child in the group
        ((ThGroup)currPtr).children.remove(0);
      }

       break;
    }
    else{
      System.out.println("something wrong!");
      break;
    }
  }
  }

  return rc;
}

public synchronized int insert(String path, Object info)
{
  System.out.println("Inserting into XMLDoc...");
  int rc = 0;
  IThNode currPtr = (ThGroup)root.children.get(0);  //current position

  while (true)
  {
    if (path.startsWith("/G")){
      //traversing the path
      path = path.substring(2);
      if (path.startsWith("["))
      { //indication of which child
        int idx = Integer.parseInt(path.substring(path.indexOf("[")+1,
                          path.indexOf("]")));
      currPtr = (IThNode)currPtr.getChild(idx-1);
      path = path.substring(path.indexOf("]")+1);
      }
```

```
      else{ //by default, first child in the group
        currPtr = (IThNode)currPtr.getChild(0);
      }
    }
    else if(path.startsWith("/L")){

      path = path.substring(2);
    }

    if (path.equals(""))
      break;
  }

  //prepare node to add into this path
  ArrayList values = (ArrayList)info;
  int offset = 0; //for now
  for(int i=0;i<values.size();i++)
  {
    String data = (String)values.get(i);
    ThPage page = (ThPage)subsys.BM.getPage();
    ThLeave leave = new ThLeave(ThSystem.getId(),page.pageNum, data);
    ((ThGroup)currPtr).children.add(leave);
  }

  return rc;
}

public synchronized int load(String filename, String xmlname)
{
  ArrayList groups = new ArrayList();
  Stack stack = new Stack();
  Stack tmpStk = new Stack();
  ThXMLDoc xmldoc;
  int rc  = 0; //return code
  stack.push(root);

  try{
   System.out.println("openning file:  "+filename);
   DataInputStream in = new DataInputStream(
                        new BufferedInputStream(
                            new FileInputStream(filename)));
   /*read in line by line*/
   String inLine;
   String data;
   String attr;
   long id;
   int offset=0;

   System.out.println("Loading into XMLDoc...");
   while((inLine = in.readLine())!=null)
   {
     /* If it reads in form <L> ...data...</L>, parse the data
      *       between the 2 tags and creates leave node.
      * else if reads in only <L> ... data, parse data, push on stack
      * else if reads in only </L>, pop tage, <L>, pop data
      *                                and create leave node
```

```
 * else if reads in only </L>, pop tag and it's not <L>, ERROR
 */
if (inLine.indexOf("<L") >= 0 && inLine.indexOf("</L>") >= 0){ //<L> ... </L>
  data = new String(inLine.substring(inLine.indexOf("<L")+3,
                                inLine.indexOf("</L>")));
  id = ThSystem.getId();   //unique logical id for this leave node
  ThLeave leave = new ThLeave(id,rootPage.pageNum,data);
  ThGroup group = (ThGroup)stack.pop();
  group.children.add(leave);
  stack.push(group);
}
else if(inLine.indexOf("<L") >= 0 && inLine.indexOf("</L>") < 0){ //<L> ...
  tmpStk.push(inLine);
}
else if(inLine.indexOf("</L>") >= 0 && inLine.indexOf("<L") < 0){ //</L>
  data = (String)tmpStk.pop();
  if (data.indexOf("<L")<0)
  {  //<L> not found
    return MISS_MATCH_LEAVE;
  }
  data = new String(data.substring(data.indexOf("<L")+3));
  id = ThSystem.getId();   //unique logical id for this leave node
  ThLeave leave = new ThLeave(id,rootPage.pageNum,data);
  ThGroup group = (ThGroup)stack.pop();
  group.children.add(leave);
  stack.push(group);
}
else if (inLine.indexOf("<G") >= 0)
{
/* If found <G, push on stack
 * else if found </G>, pop stack, construct G node
 */
  attr = new String(inLine.substring(inLine.indexOf("<G")+3,
                                inLine.indexOf("=")));
  data = new String(inLine.substring(inLine.indexOf("=")+2,
                                inLine.indexOf(">")-1));
  ThGroup group = new ThGroup(attr,data);
  stack.push(group);
}
else if (inLine.indexOf("<XML") >= 0)
{
/* If found <XML, push on stack
 * else if found </XML>, pop stack, construct G node
 */
  attr = new String(inLine.substring(inLine.indexOf(" ")+1,
                                inLine.indexOf("=")));
  data = new String(inLine.substring(inLine.indexOf("=")+2,
                                inLine.indexOf(">")-1));
  ThGroup group = new ThGroup(attr,data);
  stack.push(group);
}
else if(inLine.indexOf("</G>")>=0 || inLine.indexOf("</XML>") >= 0)
{
  ThGroup group = (ThGroup)stack.pop();
  ThGroup parent = (ThGroup)stack.pop();
  parent.children.add(group);     //add this Group node into tree
```

```
      stack.push(parent);
    }else{
      ;
    }
   }
   System.out.println("Closing file:  "+filename);
   in.close();
   rc = 0;
  }catch(Exception e){
   e.printStackTrace();
  }

  return rc;
 }

 public int update(String xmlname,  String path,
                String newVal, ThSystem subsys)
 {
   System.out.println("Updating "+xmlname+"...");
   int rc = 0;
   IThNode currPtr = (IThNode)((ThGroup)root).children.get(0);  //current position
   Stack stack = new Stack();
   int idx = 0;

   while (true)
   {
     if (path.startsWith("/G"))
     {
       path = path.substring(2);
       stack.push(currPtr);
       if (path.startsWith("["))
       { //indication of which child
         idx = Integer.parseInt(path.substring(path.indexOf("[")+1,
                          path.indexOf("]")));
         idx -= 1; // array idx starts from 0
         currPtr = (IThNode)currPtr.getChild(idx);
         path = path.substring(path.indexOf("]")+1);
       }
       else{ //by default, first child in the group
         currPtr = (IThNode)currPtr.getChild(0);
       }

       if (path.equals(""))
       {
         currPtr = (ThGroup)stack.pop();
         ((ThGroup)currPtr).children.remove(idx);
         ((ThGroup)currPtr).children.add(idx,newVal);
         break;
       }
     }
     else if (path.startsWith("/L"))
     {
       path = path.substring(2);
       currPtr = (ThGroup)stack.pop();
       if (path.startsWith("[")){ //indication of which child
         idx = Integer.parseInt(path.substring(
```

```
                    path.indexOf("[")+1, path.indexOf("]")));
          idx -= 1; // array idx starts from 0
          ((ThGroup)currPtr).children.remove(idx);
          ThPage page = (ThPage)subsys.BM.getPage();
          ThLeave leave = new ThLeave(ThSystem.getId(),page.pageNum, newVal);
          ((ThGroup)currPtr).children.add(idx, leave);
        }
        else{ //by default, first child in the group
          ((ThGroup)currPtr).children.remove(0);
          ThPage page = (ThPage)subsys.BM.getPage();
          ThLeave leave = new ThLeave(ThSystem.getId(),page.pageNum, newVal);
          ((ThGroup)currPtr).children.add(0, leave);
        }
        break;
      }else{
        System.out.println("something wrong!");
        break;
      }
    }


    return rc;
  }

/**
  *  This function will call printNode to perform the actually print for document tree
  *  @params: none
  *  @returns: none
  */
  public void printTree()
  {
    System.out.println("Printing Logical structure of XML Tree:  "+name);

    printNode((ThGroup)root.children.get(0));
  }

 /**
  *  This function print document tree recursively
  *  @params: root node
  *  @returns: none
  */
  public void printNode(ThGroup node)
  {
    System.out.println(spaces+node.attribute.toString());
    spaces += "  ";
    for (int i=0;i<node.children.size();i++)
    {
      Object obj = node.children.get(i);

      if (((obj.getClass()).getName()).equals("cs297.xmldb.src.cm.ThGroup"))
      {
        printNode((ThGroup)obj);
      }
      else{     //it's a leave node
        System.out.println(spaces + ((ThLeave)obj).data);
      }
```

```
    }
    spaces = spaces.substring(2);
  }

  public void printNodeToResultSet(ThGroup node, ArrayList arr)
  {
    arr.add(spaces+node.attribute.toString());
    spaces += "  ";
    for (int i=0;i<node.children.size();i++)
    {
      Object obj = node.children.get(i);

      if (((obj.getClass()).getName()).equals("cs297.xmldb.src.cm.ThGroup"))
      {
        printNode((ThGroup)obj);
      }
      else{     //it's a leave node
        arr.add(spaces + ((ThLeave)obj).data);
      }
    }
    spaces = spaces.substring(2);
  }
}
```

----------Class ThAgent----------

```
package cs297.xmldb.src.server;
import java.io.*;
import java.util.*;
import java.net.*;
import cs297.xmldb.src.cm.*;

/**
 * This class handle execution of DML statements
 * Date: Spring 2004
 * @author: Thien An Nguyen,
 */
public class ThAgent
{
 ThSystem subSys;
  public ThAgent(ThSystem sub)
 {
   subSys = sub;
 }

/**
 * This function handle execution of DML statements
 * Receive: the parsed statements in a structure called ThPlan
 * Return: return code
 */
 public int executeDMLPlan(ThPlan plan, ArrayList resultSet)
{
   if (!plan.success)
     return 12;  //parsing error

   int rc = 0;
   System.out.println("Executing DML Plan");
```

```
ArrayList stmts = plan.stmts;
for (int i=0; i<stmts.size(); i++)
{  //execute individual stmt
 ThStatement stmt = (ThStatement) stmts.get(i);

 if (stmt.sql.equals("SELECT"))
 {
   String path = (String)((ArrayList)stmt.params.get(0)).get(0);
   String xmlname = (String)((ArrayList)stmt.params.get(1)).get(0);
   int numDoc = ((ArrayList)stmt.params.get(1)).size();

   String dbname = subSys.currentDb;
   ThDB db = (ThDB)subSys.cataLog.get(dbname);
   ThXMLDoc xmldoc = (ThXMLDoc)db.docNames.get(xmlname);

   for (int j=0;j<numDoc;j++)
  { //loop through number of documents
     if (xmldoc == null){
       System.out.println(xmlname+" not found");
      return 8;
     }
     else{
       Object result = xmldoc.select(path);

       ArrayList res = (ArrayList) result;
       for (int k=0;k<res.size();k++)
       {
        Object obj = res.get(k);
        xmldoc.printNodeToResultSet((ThGroup)obj, resultSet);
       }
     }
   }

   return 0;
 }
 else if(stmt.sql.equals("INSERT"))
 {
   String xmlname = (String)stmt.params.get(0);
   //look for current db in catalog
   String dbname = subSys.currentDb;
   ThDB db = (ThDB)subSys.cataLog.get(dbname);
   //look for xmlname in current db
   ThXMLDoc xmldoc = (ThXMLDoc)db.docNames.get(xmlname);
   if (xmldoc == null)
   {
     System.out.println(xmlname+" not found");
     return 8;
   }else{
     String path = (String)stmt.params.get(1);
     Object content = stmt.params.get(2);
     rc = xmldoc.insert(path, content);
     if (rc > 0)
       return rc;
   }
 }
 else if(stmt.sql.equals("DELETE")){
```

```
  String xmlname = (String)stmt.params.get(0);
  //look for current db in catalog
  String dbname = subSys.currentDb;
  ThDB db = (ThDB)subSys.cataLog.get(dbname);
  //look for xmlname in current db
  ThXMLDoc xmldoc = (ThXMLDoc)db.docNames.get(xmlname);
  if (xmldoc == null){
    System.out.println(xmlname+" not found");
    return 8;
  }
  else{
    Object path = stmt.params.get(1);
    rc = xmldoc.delete(path);
    if (rc > 0)
      return rc;
  }
}
else if(stmt.sql.equals("UPDATE"))
{
  String xmlname = (String)stmt.params.get(0);
  String newVal = (String)stmt.params.get(1);
  String path = (String)((ArrayList)stmt.params.get(2)).get(0);
  //look for current db in catalog
  String dbname = subSys.currentDb;
  ThDB db = (ThDB)subSys.cataLog.get(dbname);
  //look for xmlname in current db
  ThXMLDoc xmldoc = (ThXMLDoc)db.docNames.get(xmlname);
  if (xmldoc == null){
    System.out.println(xmlname+" not found");
    return 8;
  }
  else{
    rc = xmldoc.update(xmlname, path, newVal, subSys);
    if (rc > 0)
      return rc;
  }
}
else if(stmt.sql.equals("ROLLBACK"))
{

}
else if(stmt.sql.equals("COMMIT"))
{
   //will be implements
}
else if(stmt.sql.equals("LOAD"))
{
  String xmlname = (String)stmt.params.get(1);
  String filename = (String)stmt.params.get(0);
  //look for current db in catalog
  String dbname = subSys.currentDb;
  ThDB db = (ThDB)subSys.cataLog.get(dbname);
  //look for xmlname in current db
  ThXMLDoc xmldoc = (ThXMLDoc)db.docNames.get(xmlname);
  if (xmldoc == null){
    System.out.println(xmlname+" not found");
```

```
        return 8;
      }
      else
      {
        rc = xmldoc.load(filename,xmlname);
        if (rc > 0)
          return rc;
      }
    }
    else if(stmt.sql.equals("CREATE XMLDOC"))
    {
      String xmlname = (String)stmt.params.get(0);
      String dbname = (String)stmt.params.get(1);

      ThDB db = (ThDB) subSys.cataLog.get(dbname);
      subSys.currentDb = dbname;
      rc = db.createXmlDocEntry(xmlname, subSys);
      if (rc != 0) //if fail, rc=8
        return rc;
    }
    else if(stmt.sql.equals("DROP XMLDOC"))
    {
      String dbname = subSys.currentDb;
      ThDB db = (ThDB) subSys.cataLog.get(dbname);
      db.dropXmlDoc((String)stmt.params.get(0));
    }
    else if(stmt.sql.equals("CREATE DATABASE"))
    {
      String dbname = (String)stmt.params.get(0);
      rc = subSys.createDB(dbname);
      if (rc == -9){
        System.out.println("Database name "+dbname);
      }
    }else if(stmt.sql.equals("DROP DATABASE")){
      subSys.dropDB((String)stmt.params.get(0));
    }else if(stmt.sql.equals("PRINT")){
      String xmlname = (String)stmt.params.get(0);
      ThDB db = (ThDB)subSys.cataLog.get(subSys.currentDb);
      ThXMLDoc doc = (ThXMLDoc)db.docNames.get(xmlname);
      doc.printTree();
    }else{
      System.out.println("Unknown statement:   "+stmt.sql);
      return 12;
    }
  }
  return 0;
 }
}
```

----------Class ThDB----------

```
package cs297.xmldb.src.server;

import java.io.*;
import java.util.*;
import cs297.xmldb.src.cm.*;
```

```
/*
 * This class creates an instans of a specific database
 * when it is called
 * Date: Spring 2004
 * @author: Thien An Nguyen,
 */
public class ThDB
{
 public boolean dbUp = false; //no access to this db
 public Hashtable docNames;
 public String name = null;

 public static int DOC_NOT_FOUND = -3;

 public ThDB(String dbName)
 {
  name = new String(dbName);
  docNames = new Hashtable();
  dbUp = true;
 }

 public void startDB()
 {
  dbUp = true;
 }

 public void stopDB()
 {
  dbUp = false;
 }

 public boolean insertDoc(String docName)
 {
  return false;
 }

 public boolean deleteDoc(String docName)
 {
  return false;
 }



/**
 *  This function creates an xmldocument object in the database
 * @params: name of the xml document, and the sub system
 * @returns: return code indicate the state of the execution.
 * RC=0 indicate of successful execution
 */
 public int createXmlDocEntry(String xmlName, ThSystem subsys)
 {
  if(dbUp){
   docNames.put(xmlName, new ThXMLDoc(xmlName, subsys));   // this DB
   System.out.println(xmlName+" added in "+name);
   return 0;
  }else{
```

```
    return ThSystem.SYS_DOWN;  //-1
   }
  }

  //remove this db from catalog table
  public int dropXmlDoc(String xmlName)
  {
   if(dbUp){
    if ((docNames.remove(xmlName)) != null)
     System.out.println(xmlName + " dropped");
    else
     return DOC_NOT_FOUND;   //-3
    return 0;
   }else{
    return ThSystem.SYS_DOWN;  //-1
   }
  }
}
```

----------Class ThSystem----------

```
package cs297.xmldb.src.server;

import java.io.*;
import java.util.*;
import cs297.xmldb.src.cm.*;
import cs297.xmldb.src.bm.*;
import cs297.xmldb.src.sm.*;

/**
 * This class contains global variables such as system
 * catalog (where to search for created databases,...
 * Date: Spring 2004
 * @author: Thien An Nguyen,
 */
public class ThSystem implements Serializable
{
  public static int SYS_DOWN = -1;
  public static int DB_NOT_FOUND = -2;
  public static int DB_EXISTED = -9;

  public boolean sysUp = false; // system is down
  public Hashtable cataLog;
  public String currentDb;
// public StorageMgr SM;
  public ThBufferMgr BM;
  public ThTransactionMgr TM;
  static long id = 1;
  ObjectOutputStream outStream;

  /*
   * construct database subsystem
   */
  public ThSystem()
  {
   System.out.println("System starting up...");
   try{
```

```
    FileInputStream infile = new FileInputStream("c://an//cs297//xmldb//system.out");
    ObjectInputStream inStream = new ObjectInputStream(infile);

    cataLog = (Hashtable)inStream.readObject();
    if (cataLog == null){ //empty
      cataLog = new Hashtable();
    } else{
      //currentDb = ?
    }

    outStream = new ObjectOutputStream(
              new FileOutputStream("c://an//cs297//xmldb//system.out"));
  }catch(Exception e){
    cataLog = new Hashtable();
  }
  BM = new ThBufferMgr();
  TM = new ThTransactionMgr("c://an//cs297//xmldb//transaction.out");
  //check to see if transaction table is not empty
  if (TM.transactionTable.size() > 0){
    //go through logs & apply
  }
 }

public int createDB(String dbName)
{ //create an entry in catalog table for
  if(sysUp)
  {
    if (cataLog!=null && cataLog.get(dbName)!=null){
      return DB_EXISTED;
    }

    cataLog.put(dbName, new ThDB(dbName));   // this DB
    currentDb = dbName;
    System.out.println(dbName + " created");
    return 0;
  }else{
    return SYS_DOWN;  //-1
  }
}

public int dropDB(String dbName)
{  //remove this db from catalog table
  if(sysUp){
    if ((cataLog.remove(dbName)) != null)
      System.out.println(dbName + " dropped");
    else
      return DB_NOT_FOUND; //-2
    return 0;
  }else{
    return SYS_DOWN; //-1
  }
}

public static long getId()
{
  return id++;
```

```
 }
  public boolean start()
  {
    sysUp = true;
    System.out.println("Sub-system is up");
    return true;
  }

  public boolean stop()
  {
    sysUp = false;       //so that no body can access
    System.out.println("Sub-system is stopped");
    return true;
  }
}
```

----------class ThPage----------

```
package cs297.xmldb.src.bm;

import java.io.*;
import java.util.*;
import cs297.xmldb.src.cm.*;

public class ThPage implements Serializable
{
  public int pageNum;
  public int pSize = 1000; //page size = 1000 bytes
  public StringBuffer pageBuff;
  public boolean commitedAll = false;

  public ThPage(int pnum)
  {
    pageNum = pnum;
    pageBuff = new StringBuffer();
  }

  public ThPage(int pnum, int size)
  {
    pSize = size;
    pageBuff = new StringBuffer();
  }

  public int write()
  {
  ObjectOutputStream outStream;

  try{
      outStream = new ObjectOutputStream(
        new FileOutputStream("c://an//cs297//xmldb//src//bm//"+pageNum+".out"));
      outStream.writeObject(this);
  }catch(Exception e){
      e.toString();
  }
    return 0;  //success
  }
}
```

----------class ThPage----------

```
package cs297.xmldb.src.bm;

import java.io.*;
import java.util.*;
import cs297.xmldb.src.cm.*;

public class ThBufferMgr implements Serializable
{
 public static int pSn = 0;   //page serial number
 public Hashtable bufferPool = new Hashtable();  //pages in buffer pool
 public ArrayList freeList = new ArrayList();
 public ArrayList leastRecentlyUse = new ArrayList();
 public int totalPage = 20; //20 pages in the pool
 public int capacity = 0;
 public int maxCapacity = (2*totalPage)/3;

 /*Buffer Manager will be construct at db sub-system starts up
  */
 public ThBufferMgr()
 {
   for (int pSn=1;pSn<=totalPage;pSn++)
  {
    freeList.add(new ThPage(pSn));
   }
 }

 public ThPage getPage()
 {
  return new ThPage(1);
 }

 public ThPage getPage(int num)
 {
   ThPage page = (ThPage)bufferPool.get(Integer.toString(num));
   if ((page) != null){ //if found page in bufferpool, return it
     return page;
   }else{
     ;//bring the page in from disk
     return null;
   }
 }

 public ThPage getNewPage(
 {
   if (freeList.size() < 1){
     //force the LRU to disk
     //find the lease recently use block(page)
     String key = (String)leastRecentlyUse.remove(0);
     ThPage page = (ThPage)bufferPool.remove(key);
     //write this page to disk
     page.write();
     pSn++;
     page = new ThPage(pSn);
     bufferPool.put(Integer.toString(pSn), page);
```

```
      return page;
    }
    else{
      //remove the 1st page in freelist
      ThPage newpage = (ThPage)freeList.remove(0);
      //add page number to recently use list
      leastRecentlyUse.add(Integer.toString(newpage.pageNum));
      //put page in buffer pool
      bufferPool.put(Integer.toString(capacity), newpage);

      return newpage;
    }
  }
}
```

Following is the content of DML7.TXT:

```
CREATE DATABASE DB1DB;
CREATE XMLDOC XML1DOC IN DB1DB;

LOAD "c://an//cs297//xmldb//test1.xml"
    INTO XML1DOC;

printTree(XML1DOC);

//* add to the end of first element of /G[1], /G[2], ...
INSERT INTO XML1DOC PATH("/G/G[1]") VALUES("<L>392 Lily Ann Way,
                    San Jose CA 95111</L>",
                    "<L>(408)123-1234</L>");
INSERT INTO XML1DOC PATH("/G/G[2]") VALUES("<L>123 No Name St.,
                    San Jose CA 95112</L>",
                    "<L>(408)123-1234</L>"); COMMIT;
INSERT INTO XML1DOC PATH("/G/G[3]") VALUES("<L>234 Bernal,
                    San Jose CA 95012</L>");
INSERT INTO XML1DOC PATH("/G/G[4]") VALUES("<L>555 Bailey Road,
                    San Jose CA 94091</L>");

INSERT INTO XML1DOC PATH("/G/G[5]") VALUES("<L>1234 1st street,
                    San Francisco CA 95012</L>",
                        <L>(510)344-1010</L>");

INSERT INTO XML1DOC PATH("/G/G[6]") VALUES("<L>2234 2st street,
                    Santa Rosa CA 94323</L>",
                    <L>(808)463-1203</L>");

INSERT INTO XML1DOC PATH("/G/G[7]") VALUES("<L>3542 3st street,
                    Santa monica CA 95002</L>",
                        <L>(808)463-1203</L>");


//* comments

printTree(XML1DOC);
```

```
DELETE FROM XML1DOC WHERE PATH("/G/G[4]");
DELETE FROM XML1DOC WHERE PATH("/G/G/L[4]", "/G/G/L[3]");

printTree(XML1DOC);

UPDATE XML1DOC SET VALUE("<L>392 Pasilo dr., San Jose CA 95123</L>")
            WHERE PATH("/G/G/L[2]");

printTree(XML1DOC);

SELECT "/G[*]" FROM XML1DOC;

DROP XMLDOC XML1DOC;
DROP DATABASE DB1DB;
```

Following is the snapshot of running the above testcase:

```
testExecuteDML7()
System starting up...
Constructing transactiontable...
Sub-system is up
Parsing DML...
Executing DML Plan
DB1DB created
XML1DOC added in DB1DB
openning file:  c://an//cs297//xmldb//test1.xml
Loading into XMLDoc...
Closing file:  c://an//cs297//xmldb//test1.xml
Printing Logical structure of XML Tree:  XML1DOC
 ID=00000001
   name=CS297
     id=123-45-6789
       Jane Eyre
       3.8
     id=234-56-7890
       Irene Hugh
       3.0
     id=345-67-8901
       Thomas Lee
       2.8
     id=456-78-9012
       Erica Nguyen
       3.1
     id=567-45-6789
       Rochester Thornfield
       2.8
     id=678-45-6789
       John Lyle
       3.2
     id=789-00-6789
       Robert Mulan
       3.2
Inserting into XMLDoc...
Inserting into XMLDoc...
```

```
Inserting into XMLDoc...
Inserting into XMLDoc...
Inserting into XMLDoc...
Inserting into XMLDoc...
Inserting into XMLDoc...
Printing Logical structure of XML Tree:  XML1DOC
 ID=00000001
  name=CS297
    id=123-45-6789
      Jane Eyre
      3.8
      392 Lily Ann Way, San Jose CA 95111
      (408)123-1234
    id=234-56-7890
      Irene Hugh
      3.0
      123 No Name St., San Jose CA 95112
      (408)123-1234
    id=345-67-8901
      Thomas Lee
      2.8
      234 Bernal, San Jose CA 95012
    id=456-78-9012
      Erica Nguyen
      3.1
      555 Bailey Road, San Jose CA 94091
    id=567-45-6789
      Rochester Thornfield
      2.8
      1234 1st street, San Francisco CA 95012
      (510)344-1010
    id=678-45-6789
      John Lyle
      3.2
      2234 2st street, Santa Rosa CA 94323
      (808)463-1203
    id=789-00-6789
      Robert Mulan
      3.2
      3542 3st street, Santa monica CA 95002
      (808)463-1203
Deleting from XMLDoc...
Deleting from XMLDoc...
Printing Logical structure of XML Tree:  XML1DOC
 ID=00000001
  name=CS297
    id=123-45-6789
      Jane Eyre
      3.8
      392 Lily Ann Way, San Jose CA 95111
      (408)123-1234
    id=234-56-7890
      Irene Hugh
      3.0
      123 No Name St., San Jose CA 95112
      (408)123-1234
```

```
    id=678-45-6789
      John Lyle
      3.2
      2234 2st street, Santa Rosa CA 94323
      (808)463-1203
    id=789-00-6789
      Robert Mulan
      3.2
      3542 3st street, Santa monica CA 95002
      (808)463-1203
Updating XML1DOC...
Printing Logical structure of XML Tree:  XML1DOC
 ID=00000001
  name=CS297
    id=123-45-6789
      Jane Eyre
      3.8
      392 Lily Ann Way, San Jose CA 95111
      (408)123-1234
    392 Pasilo dr., San Jose CA 95123
    id=678-45-6789
      John Lyle
      3.2
      2234 2st street, Santa Rosa CA 94323
      (808)463-1203
    id=789-00-6789
      Robert Mulan
      3.2
      3542 3st street, Santa monica CA 95002
      (808)463-1203
Selecting from XMLDoc: XML1DOC
  id=123-45-6789
    Jane Eyre
    3.8
    392 Lily Ann Way, San Jose CA 95111
    (408)123-1234
  id=678-45-6789
    John Lyle
    3.2
    2234 2st street, Santa Rosa CA 94323
    (808)463-1203
  id=789-00-6789
    Robert Mulan
    3.2
    3542 3st street, Santa monica CA 95002
    (808)463-1203
 name=CS297
  392 Pasilo dr., San Jose CA 95123
Time: 1.632
OK (5 tests)
```