Simple DBMS

16 / 11 / 2014

A simulation of SQL database management system , that allows user to create databases , tables , insert values and delete values from it

Team members

Ahmed Khaled
Khaled Mohamed El-Tahan

Hend Almorsi
Mohamed Nabil

Islam Salah
Omar Mohamed Eissa

Contents

Project Description

Introduction

A Computer Database is a structured collection of records or data that is stored in a computer system. On the other hand, a Database Management System (DBMS) is a complex set of software programs that controls the organization, storage, management, and retrieval of data in a database. DBMS are categorized according to their data structures or types. The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data. On the other hand, Extensible Markup Language (XML) is a set of rules for encoding documents in machine readable form. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards.

Requirements

- Implement a simple DBMS that handles data stored in XML files.
- You should support the following SQL Statements:
  - Create database
  - Create table
  - Insert into table
  - Select from table
  - Delete from table
  - Update table
- For statements which contain conditions, support only the simple conditions: =, >, and <. You are NOT required to support: AND, OR, or NOT.

- You should create a DBMS Java interface and implement it. The DBMS interface should contain the functions that any program may need to use, for each of the supported statements. For example, to support database creation, you will add a method to the DBMS interface with signature similar to: *public void createDatabase(String databaseName);* The DBMS interface does NOT handle SQL queries directly. SQL queries should be sent first to another class, say *Parser* class. The *Parser* parses them, and then calls the appropriate DBMS function. The *Parser* should use only the functions supported by the DBMS interface to access the database. The files should be updated, whenever any DBMS function (that requires updating) is called.
- The *Parser* should validate the SQL statements and reject bad ones.
- The DBMS should validate the sent parameters.
- You are required to create a UML class diagram.
- Each table in the database should be stored in a separate XML file. Files should be placed in a directory. You may use an environment variable to get the directory path during execution.
- Schema files that contain data about tables and columns should exist. Tables should be validated across their schema files. (DTDs can be used for this purpose).
- The DBMS you will develop should control the management and retrieval of data from data files. The DBMS accepts requests for data from application programs and retrieves and transfers the appropriate data from files that are stored physically on disk.
- You should use SAX or DOM, or StAX parsers to parse and validate the XML database files.

Project Design Description

Main Design

- Parser Package : responsible for receiving the input command as a string and returns an object that represents the command
- Schema Package : responsible for creating and handling the schema files by which the application checks whether the required table exists or not
- DBMS : contains the engine of the application which receives the object command and connects it to the XML operation classes
- Core Package : contains the main class that implements dbms interface , a Runnable user interface and the junit test

Parser Design

- The Parser consists of two classes
    1. The Parser class : a class that receives the command as an input and uses regex to validate whether the sent string is a valid SQL command or not , this class provides many SQL simulation abilities on command validation such as :
        - Checks the selected operation
        - Checks whether the database , table & cols names are reserved SQL words or not “e.g. “update” is not accepted as a table name”
        - Every test on SQL uses ignore case because SQL is case insensitive
        - Sends the result to the Entry class
    2. The Entry Class : a class that represents the command after the validation , this class provides you to check whether the command was one of those
        - Error
        - Operation type
        - Inserted values
        - Inserted variables names
        - Inserted data types
        - Conditions

Basic Operations Design

Create Database

- it takes the name of the data base, search if it exists. If no, it makes a directory with the given name.

Create table

- it first check if the user has created a data base to save tables in it. If yes, it takes the table names and list of the column names and create a table with these data then save this new table in XML file .

Insert into table

- Insert has two case
    1. INSERT INTO table_name VALUES(v1,...);
        - Reads the table XML, saves it in a table and then puts each value of the given array of values in a column. It puts them in the same order they were given then it write the modified table again in the same XML file.
    2. INSERT INTO table_name (c1,...) VALUES (v1,...);
        - Reads the table XML, saves it in a table and then puts each value of the given array of values in the corresponding column as given in the array of columns (each value to be put in a specific column). Finally, it write the modified table again in the same XML file.

Update Design

- Update has two types , you can update all the elements in specific columns of a table with some entered values , or you can update some elements of a columns which satisfy a condition .
- it must be provided with table name , target columns names , the values used in updating and a condition in the second version .
- Types of this function :
    - Without where condition
        - access the file of the table
        - search for the entered columns to be updated
        - update columns values with the new values
    - With where condition
        - access the file of the table
        - search for the rows which satisfy the condition and select these rows
        - search for the entered columns to be updated
        - update the elements of the columns which belongs to the selected rows

Select Design

- In selecting operation we have four cases
    - the command used : "Select * From table_name”
        - we do this task by selecting every row in the table.
        - The row is formed by taking the cell in each column which corresponds that row as our table is composed of some columns.
    - the command used : “Select * From table_name where…”
        - we do this task by selecting every row in the table which satisfies the condition.
        - The row is formed by taking the cell in each column which corresponds that row as our table is composed of some columns.
        - This happens after firstly check if the row has the cell(s) which validates this condition.

  - the command used : “Select c1,c2 From table”
    - we do this task by selecting every row in the table.The row is formed by taking the cell in the specified columns , in the order they given to the program, which corresponds that row as our table is composed of some columns.
  - the command used : "Select c1,c2 From table where…"
    - we do this task by selecting every row in the table which satisfies the condition . The row is formed by taking the cell in the specified columns , in the order they given to the program, which corresponds that row as our table is composed of some columns.
    - The is happens after firstly check if the row has the cell(s) which validates this condition.

Delete Design

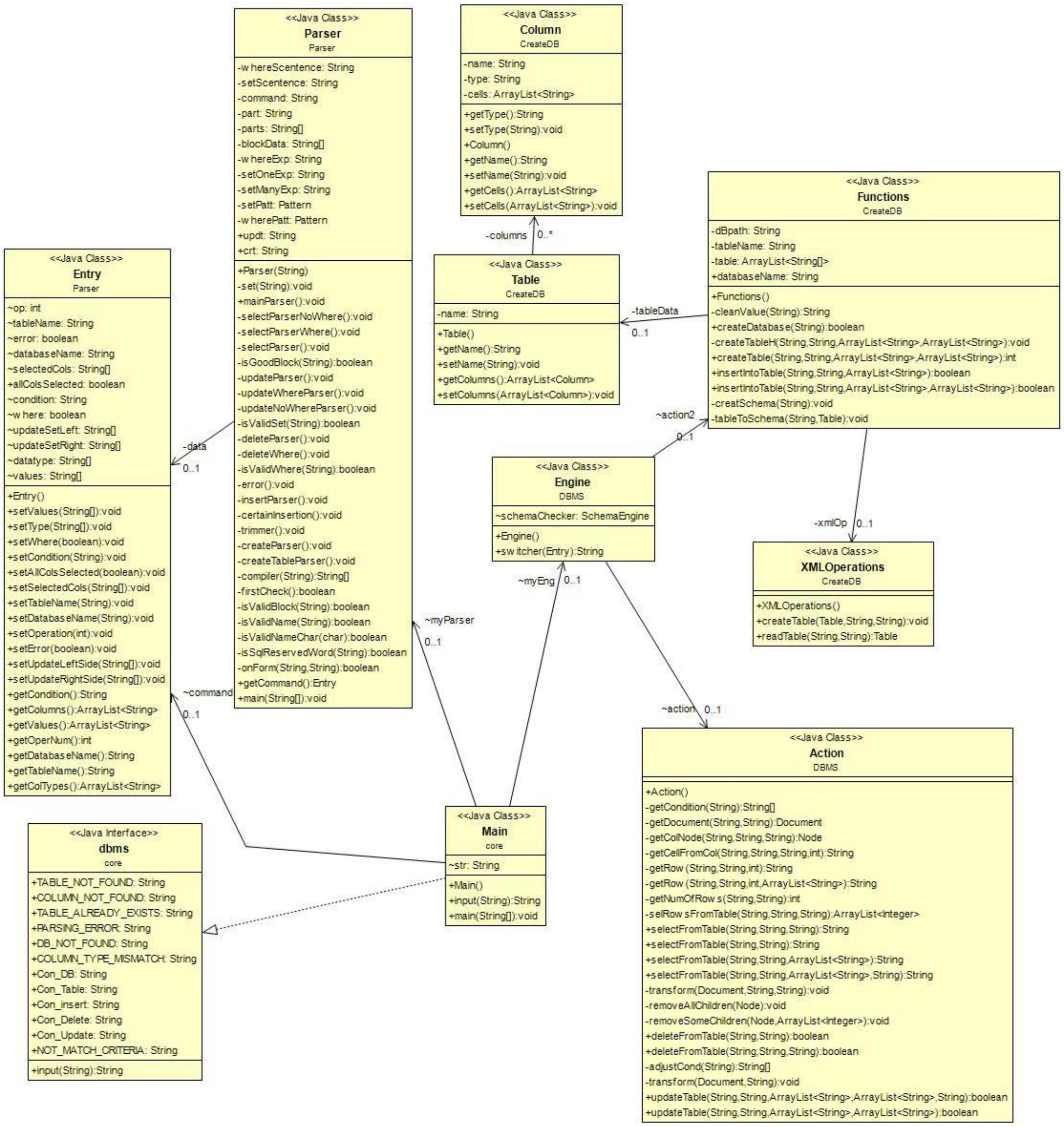
- In deleting operation we have two cases
  - the command used : "DELETE (*) FROM table_name”
    - we do this task by deleting every row in the table.
    - The row is deleted by deleting each cell in each column which corresponds that row as our table is composed of some columns.
  - the command used : "DELETE FROM table_name where..."
    - we do this task by deleting every row in the table which satisfies the condition.
    - The row is deleted by deleting each cell in each column which corresponds that row as our table is composed of some columns.
    - The is happens after firstly check if the row has the cell(s) which validates this condition.

Schema file Design

- Basic Operation :
  - creating schema file for each created Data base .
  - saving for each created table , name of the table , names and types for its columns in the schema file
- Schema file validation
  - before creating table , make sure that there isn't any existing table with the same name of the new table
  - before doning any operation , make sure that the target table is exist and each of the columns used in the operations is exist in the target table and the entered values are suitable with columns type
  - if there is a condition , make sure that the column in the condition is exist and the entered value for it is suitable with the column type

The UML Design

The Class Diagram


<<Java Class>>
Parser
Parser
-w hereScentence: String
-setScentence: String
-command: String
-part: String
-parts: String[]
-blockData: String[]
-w hereExp: String
-setOneExp: String
-setManyExp: String
-setPatt: Pattern
-w herePatt: Pattern
+updt: String
+crt: String
+Parser(String)
-set(String):void
+mainParser():void
-selectParserNoWhere():void
-selectParserWhere():void
-selectParser():void
-isGoodBlock(String):boolean
-updateParser():void
-updateWhereParser():void
-updateNoWhereParser():void
-isValidSet(String):boolean
-deleteParser():void
-deleteWhere():void
-isValidWhere(String):boolean
-error():void
-insertParser():void
-certainInsertion():void
-trimmer():void
-createParser():void
-createTableParser():void
-compiler(String):String[]
-firstCheck():boolean
-isValidBlock(String):boolean
-isValidName(String):boolean
-isValidNameChar(char):boolean
-isSqlReservedWord(String):boolean
-onForm(String,String):boolean
+getCommand():Entry
+main(String[]):void
<<Java Class>>
Column
CreateDB
-name: String
-type: String
-cells: ArrayList<String>
+getType():String
+setType(String):void
+Column()
+getName():String
+setName(String):void
+getCells():ArrayList<String>
+setCells(ArrayList<String>):void
-columns
0..*
<<Java Class>>
Functions
CreateDB
-dBpath: String
-tableName: String
-table: ArrayList<String[]>
+databaseName: String
+Functions()
-cleanValue(String):String
+createDatabase(String):boolean
-createTableH(String,String,ArrayList<String>,ArrayList<String>):void
+createTable(String,String,ArrayList<String>,ArrayList<String>):int
+insertIntoTable(String,String,ArrayList<String>):boolean
+insertIntoTable(String,String,ArrayList<String>,ArrayList<String>):boolean
-creatSchema(String):void
-tableToSchema(String,Table):void
<<Java Class>>
Entry
Parser
~op: int
~tableName: String
~error: boolean
~databaseName: String
~selectedCols: String[]
+allColsSelected: boolean
~condition: String
~w here: boolean
~updateSetLeft: String[]
~updateSetRight: String[]
~datatype: String[]
~values: String[]
+Entry()
+setValues(String[]):void
+setType(String[]):void
+setWhere(boolean):void
+setCondition(String):void
+setAllColsSelected(boolean):void
+setSelectedCols(String[]):void
+setTableName(String):void
+setDatabaseName(String):void
+setOperation(int):void
+setError(boolean):void
+setUpdateLeftSide(String[]):void
+setUpdateRightSide(String[]):void
+getCondition():String
+getColumns():ArrayList<String>
+getValues():ArrayList<String>
+getOperNum():int
+getDatabaseName():String
+getTableName():String
+getColTypes():ArrayList<String>
<<Java Class>>
Table
CreateDB
-name: String
+Table()
+getName():String
+setName(String):void
+getColumns():ArrayList<Column>
+setColumns(ArrayList<Column>):void
-tableData
0..1
~action2
0..1
-data
0..1
<<Java Class>>
Engine
DBMS
~schemaChecker: SchemaEngine
+Engine()
+sw itcher(Entry):String
-xmlOp
0..1
<<Java Class>>
XMLOperations
CreateDB
+XMLOperations()
+createTable(Table,String,String):void
+readTable(String,String):Table
~myEng
0..1
~myParser
0..1
~command
0..1
~action
0..1
<<Java Class>>
Action
DBMS
+Action()
-getCondition(String):String[]
-getDocument(String,String):Document
-getColNode(String,String,String):Node
-getCellFromCol(String,String,String,int):String
-getRow (String,String,int):String
-getRow (String,String,int,ArrayList<String>):String
-getNumOfRow s(String,String):int
-selRow sFromTable(String,String,String):ArrayList<Integer>
+selectFromTable(String,String,String):String
+selectFromTable(String,String):String
+selectFromTable(String,String,ArrayList<String>):String
+selectFromTable(String,String,ArrayList<String>,String):String
-transform(Document,String,String):void
-removeAllChildren(Node):void
-removeSomeChildren(Node,ArrayList<Integer>):void
+deleteFromTable(String,String):boolean
+deleteFromTable(String,String,String):boolean
-adjustCond(String):String[]
-transform(Document,String):void
+updateTable(String,String,ArrayList<String>,ArrayList<String>,String):boolean
+updateTable(String,String,ArrayList<String>,ArrayList<String>):boolean
<<Java Class>>
Main
core
~str: String
+Main()
+input(String):String
+main(String[]):void
<<Java Interface>>
dbms
core
+TABLE_NOT_FOUND: String
+COLUMN_NOT_FOUND: String
+TABLE_ALREADY_EXISTS: String
+PARSING_ERROR: String
+DB_NOT_FOUND: String
+COLUMN_TYPE_MISMATCH: String
+Con_DB: String
+Con_Table: String
+Con_insert: String
+Con_Delete: String
+Con_Update: String
+NOT_MATCH_CRITERIA: String
+input(String):String

Notes to user

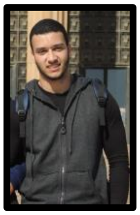
The Runnable User interface

To run the runnable user interface go to core package then use the Runnable class

The Junit Test

The junit test works only once very good , next time it gives another behavior as the database is already created so it won’t accept to create a new database with the same name , to rerun the junit you can simply refresh the project and delete the existent database file from the Junit

Contact Information



Ahmed Khaled
ID : 9

Hend Almorsi
ID : 76

Islam Salah
ID : 16

Khaled El-Tahan
ID : 26

Omar Mohamed
ID : 46

Mohamed Nabil
ID : 58