Alexandria University

Faculty of Engineering

Simple XML DBMS

Team Members:

Salma Yehia	35
Moamen Raafat	52
Mohamed Ayman	61
Nada Ayman	79

Computer and Systems Engineering Dept.

1) Introduction:

a. Project Description:

A Database Management System (DBMS) that controls the organization, storage, management, and retrieval of data in a database, Accepting and validating SQL commands (e.g. Create database, Drop database, Create table, Drop Table, Update Table, Insert into Table, Select From Table, Delete from table, etc.) in addition to handling multiple conditions supporting Boolean complete set using ‘AND’, ‘OR’ and ‘NOT’(WHERE statement), saving tables in Extensible Markup Language (XML) form, and validating them across their Schema files - Document Type Definition (DTD), also printing well formatted tables to the user.

The project in developed in java language, using the several OOP Principles and design patterns to speed up the development process, satisfy certain condition which allow testing functionalities such as DBManagerSingleton and improve the code readability in case it’s later maintained.

DOM and SAX parsers were used for parsing and validating XML files.

b. Report Overview:

i. Software Design:

Illustrates the design and how the code is organized between different packages and classes, also explains the role of each class and how it interacts with the rest, all of this is supported by the UML design.

ii. Design Patterns:

Shows the design patterns used in the project and why each one of them is the most suitable where it is used.

iii. User Guide:

Shows the user how to use the project and take advantage of its features, by providing all the supported commands and their usage, supported by illustrative examples for more clarity.

iv. Design Decisions:

1. Illustrates the reason why we have chosen that design for project.
2. Illustrates the benefits of the current design of the project.
3. Difficult decisions and trade-offs while planning for the design of the project

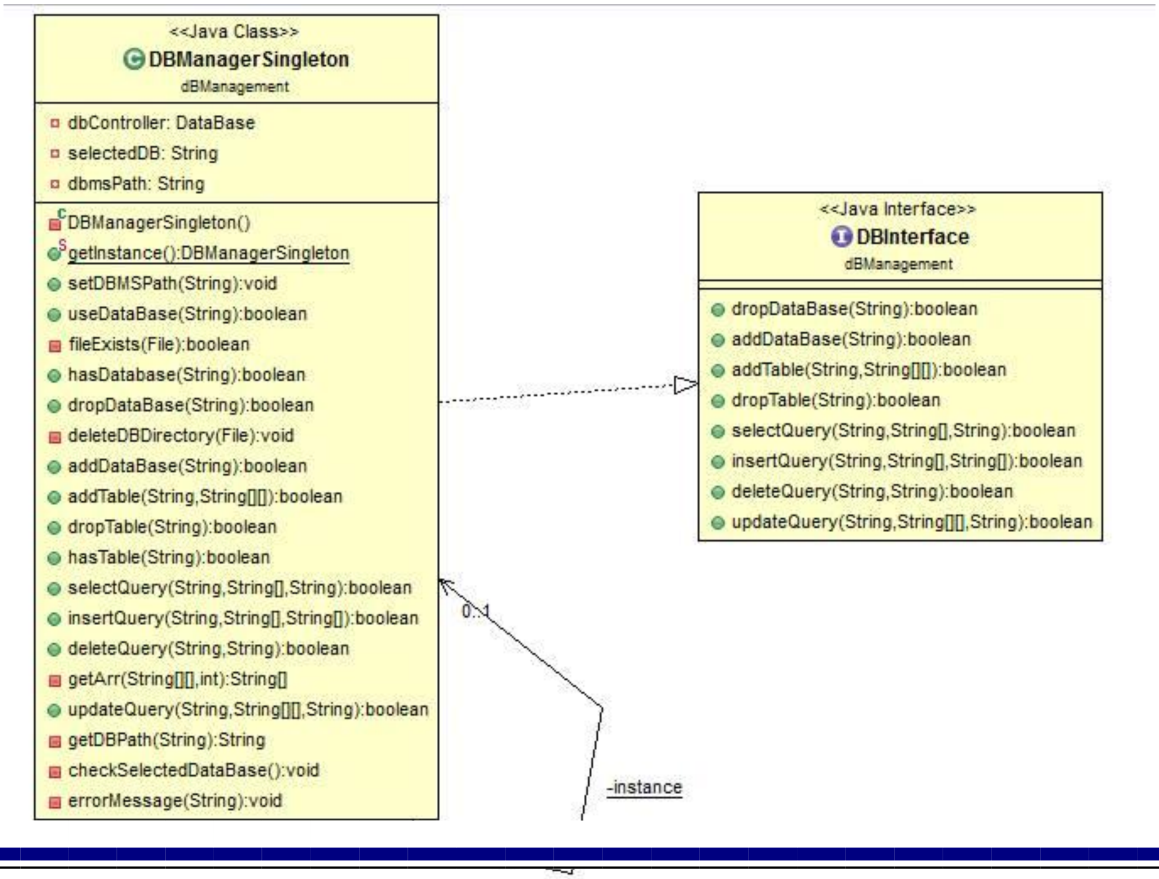
2) Software Design:

The project is divided into 5 main packages:

I. dB Management:

a) DBManagerSingleton:

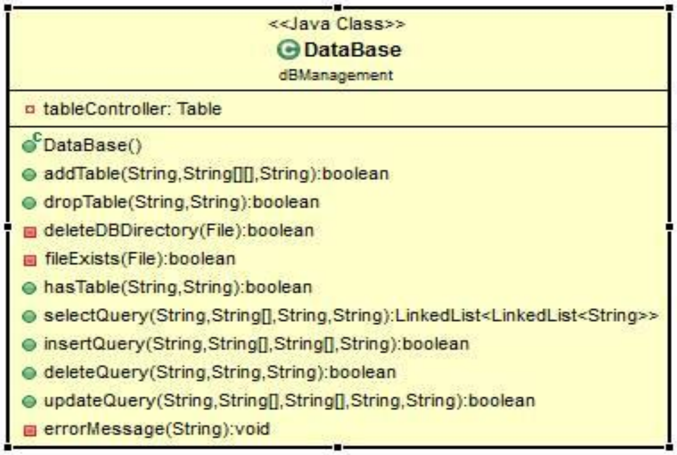
It implements the “DBInterface” interface that defines the main functionalities in the database from creating or dropping databases to the four basic commands in the SQL Language , (‘Update’, ‘Delete’, ‘Select’, ‘Insert’) where the command is already parsed before reaching the dB Management specifying which method to use. The Db Manager uses a database controller to retrieve data from the disk and use the database controller for editing the database or table inside.



<<Java Class>>
DBManagerSingleton
dBManagement
dbController: DataBase
selectedDB: String
dbmsPath: String
DBManagerSingleton()
getInstance():DBManagerSingleton
setDBMSPath(String):void
useDataBase(String):boolean
fileExists(File):boolean
hasDatabase(String):boolean
dropDataBase(String):boolean
deleteDBDirectory(File):void
addDataBase(String):boolean
addTable(String,String[][]):boolean
dropTable(String):boolean
hasTable(String):boolean
selectQuery(String,String[],String):boolean
insertQuery(String,String[],String[]):boolean
deleteQuery(String,String):boolean
getArr(String[][],int):String[]
updateQuery(String,String[][],String):boolean
getDBPath(String):String
checkSelectedDataBase():void
errorMessage(String):void
<<Java Interface>>
DBInterface
dBManagement
dropDataBase(String):boolean
addDataBase(String):boolean
addTable(String,String[][]):boolean
dropTable(String):boolean
selectQuery(String,String[],String):boolean
insertQuery(String,String[],String[]):boolean
deleteQuery(String,String):boolean
updateQuery(String,String[][],String):boolean
0..1
-instance

b) <u>DataBase:</u>

The database Controller is the linkage between the Db manager and the tables where upon specifying certain database to be used, where upon the call of the database controller the process is shifted to be a responsibility of the controller specifying whether the file was removed in case of throwing exception or executing the query on the chosen table.



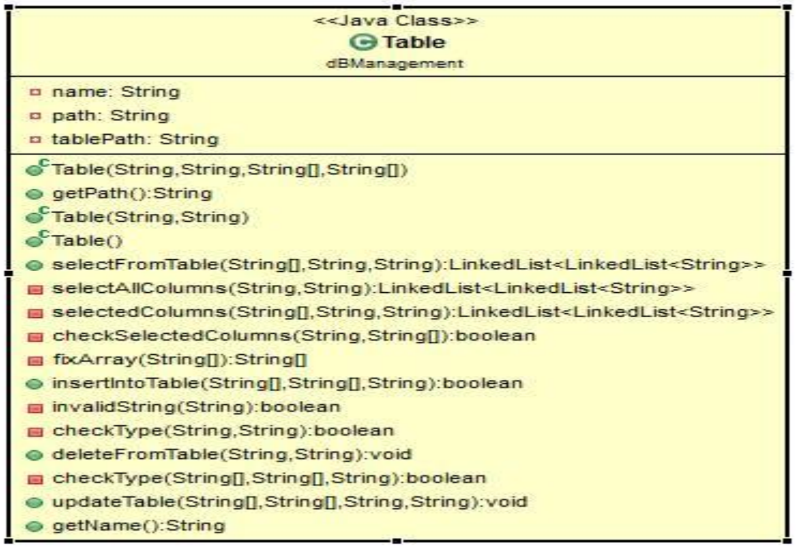
```
<<Java Class>>
DataBase
dBManagement

tableController: Table

DataBase()
addTable(String,String[][],String):boolean
dropTable(String,String):boolean
deleteDBDirectory(File):boolean
fileExists(File):boolean
hasTable(String,String):boolean
selectQuery(String,String[],String,String):LinkedList<LinkedList<String>>
insertQuery(String,String[],String[],String):boolean
deleteQuery(String,String,String):boolean
updateQuery(String,String[],String[],String,String):boolean
errorMessage(String):void
```

c) <u>Table:</u>

Responsible for representing and maintaining data validating data (Columns names, data types, name, and path) of the used table, and also for passing it to the XML parser to perform the chosen action.



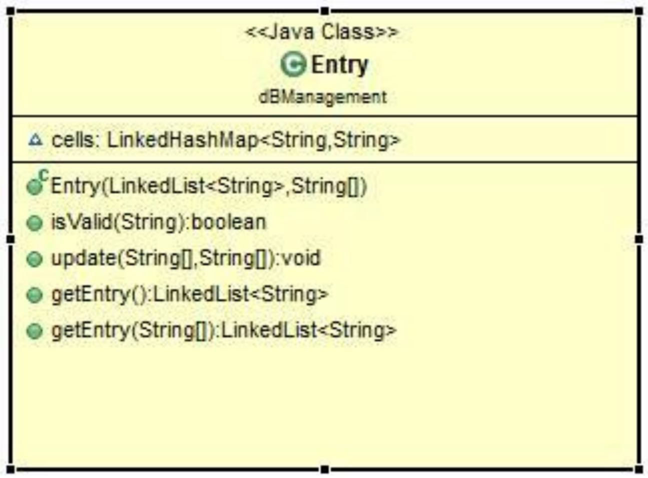
```
<<Java Class>>
Table
dBManagement

name: String
path: String
tablePath: String

Table(String,String,String[],String[])
getPath():String
Table(String,String)
Table()
selectFromTable(String[],String,String):LinkedList<LinkedList<String>>
selectAllColumns(String,String):LinkedList<LinkedList<String>>
selectedColumns(String[],String,String):LinkedList<LinkedList<String>>
checkSelectedColumns(String,String[]):boolean
fixArray(String[]):String[]
insertIntoTable(String[],String[],String):boolean
invalidString(String):boolean
checkType(String,String):boolean
deleteFromTable(String,String):void
checkType(String[],String[],String):boolean
updateTable(String[],String[],String,String):void
getName():String
```

d) Entry:

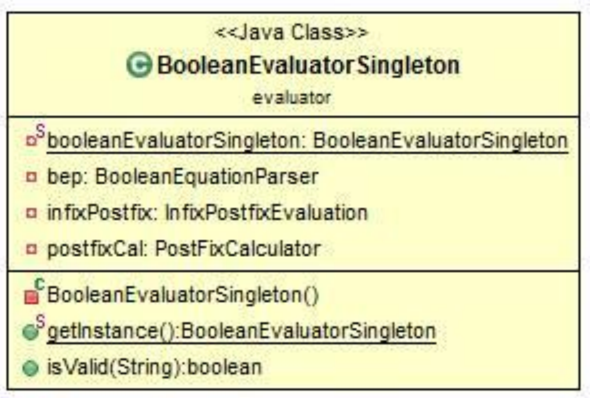
Responsible for representing and maintaining the data of the record which is being currently used to test across a given condition and for returning the accepted records.


<<Java Class>>
Entry
dBManagement
cells: LinkedHashMap<String,String>
Entry(LinkedList<String>,String[])
isValid(String):boolean
update(String[],String[]):void
getEntry():LinkedList<String>
getEntry(String[]):LinkedList<String>

II. Evaluator:

a) BooleanEvaluatorSingleton:

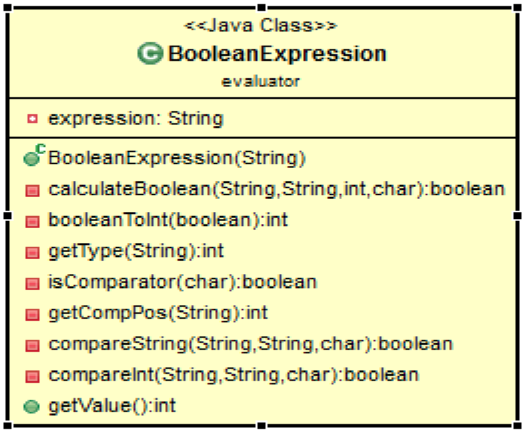
In case of evaluating condition for entries to know whether they're chosen or not, the BooleanevaluatorSingleton is used to as a façade where it connects the evaluation in entry to Parsing Boolean condition, changing the Boolean condition from infix to postfix, and get the value of the condition


<<Java Class>>
BooleanEvaluatorSingleton
evaluator
booleanEvaluatorSingleton: BooleanEvaluatorSingleton
bep: BooleanEquationParser
infixPostfix: InfixPostfixEvaluation
postfixCal: PostFixCalculator
BooleanEvaluatorSingleton()
getInstance():BooleanEvaluatorSingleton
isValid(String):boolean

b) BooleanExpression:

Used to represent a single Boolean expression which contains only two operands and an operator.

Also it’s used to calculate the value of this expression.



<<Java Class>>

BooleanExpression

evaluator

□ expression: String

● BooleanExpression(String)

■ calculateBoolean(String,String,int,char):boolean

■ booleanToInt(boolean):int

■ getType(String):int

■ isComparator(char):boolean

■ getCompPos(String):int

■ compareString(String,String,char):boolean

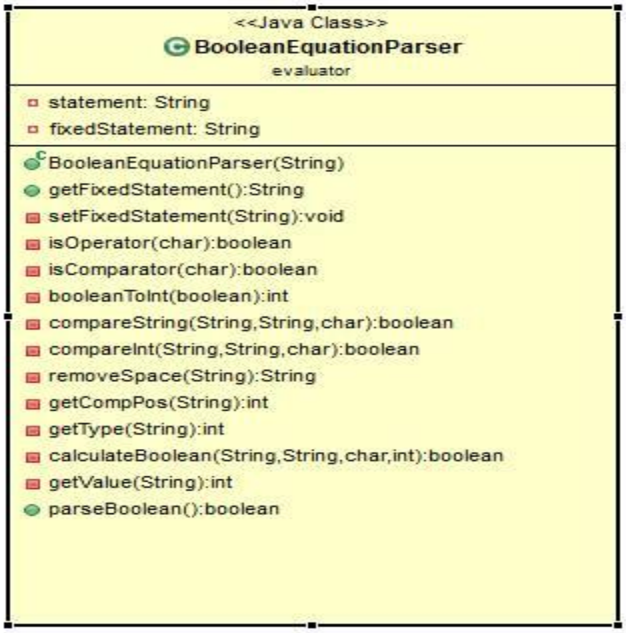
■ compareInt(String,String,char):boolean

● getValue():int

c) BooleanEquationParser:

Used to parse the Boolean condition asserting if they are well formed and satisfying the condition for a Boolean condition. It also reforms the condition to be changeable from infix to postfix using the infix to postfix converter.

Also it uses the Comparator class to compare string and calculate the value of each single expression to form the full expression.



<<Java Class>>

BooleanEquationParser

evaluator

□ statement: String

□ fixedStatement: String

● BooleanEquationParser(String)

● getFixedStatement():String

■ setFixedStatement(String):void

■ isOperator(char):boolean

■ isComparator(char):boolean

■ booleanToInt(boolean):int

■ compareString(String,String,char):boolean

■ compareInt(String,String,char):boolean

■ removeSpace(String):String

■ getCompPos(String):int

■ getType(String):int

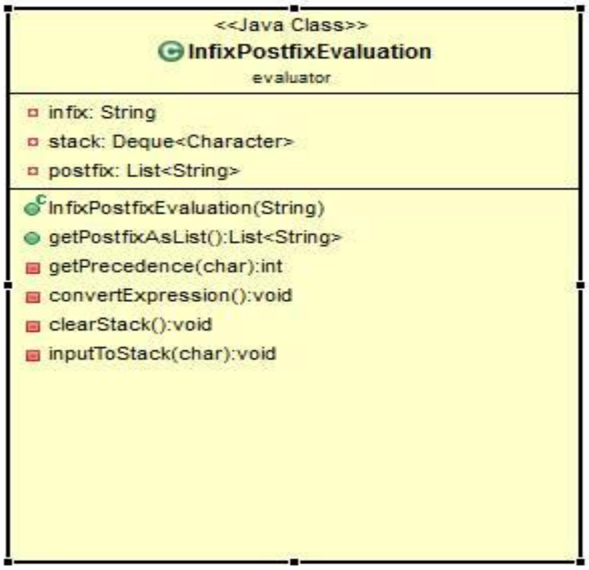
■ calculateBoolean(String,String,char,int):boolean

■ getValue(String):int

● parseBoolean():boolean

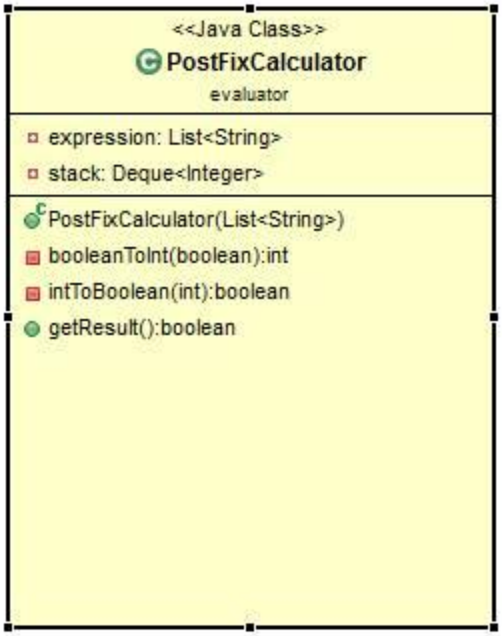
d) InfixPostfixEvaluation:

Converts Boolean statements from infix to postfix to be calculated more easily with far less complications using the PostFixCalculator.



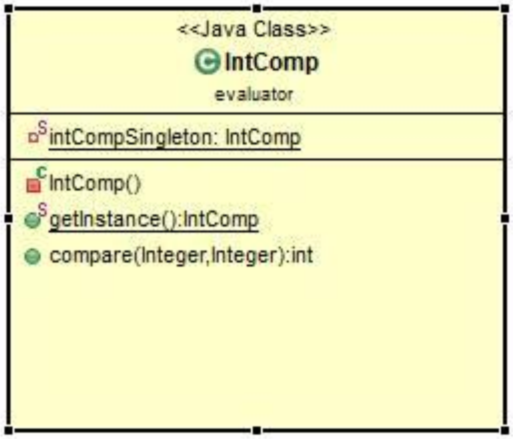
e) PostFixCalculator:

Calculates the result value of a Boolean statement represented using Postfix.



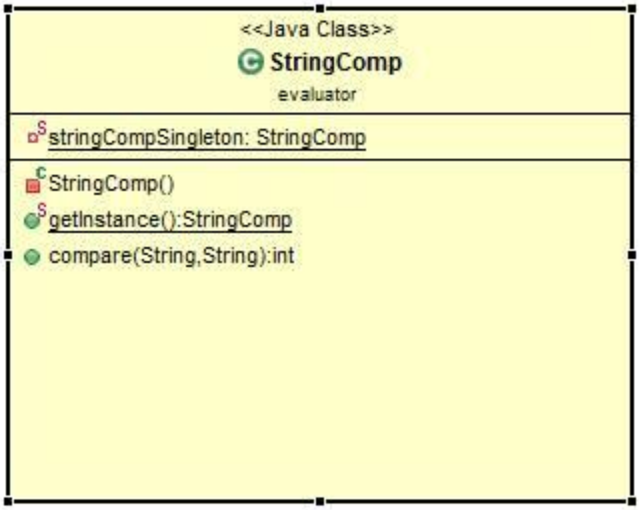
f) <u>IntComp:</u>

Implements Comparator<Integer>, and responsible for comparing Integers.



g) <u>StringComp:</u>

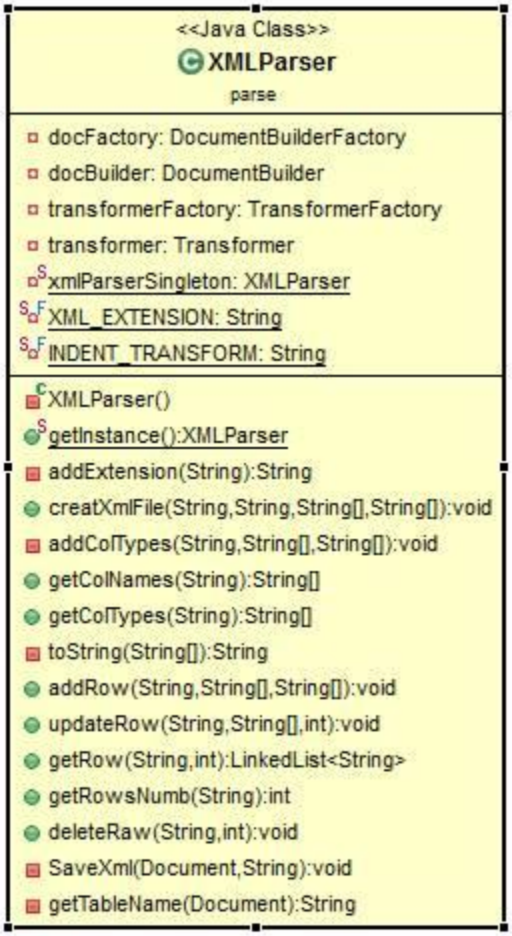
Implements Comparator<String>, and responsible for comparing Strings.



III. Parse:

a) XMLParser:

Responsible for saving and retrieving the tables’ data from the XML files using DOM parser.



<<Java Class>>

XMLParser

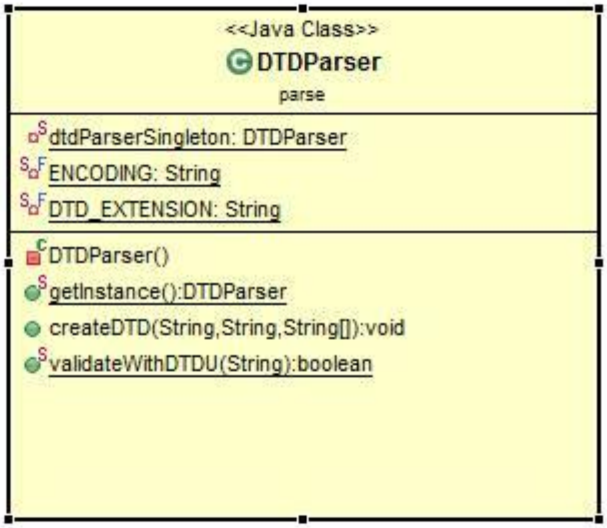
parse

docFactory: DocumentBuilderFactory
docBuilder: DocumentBuilder
transformerFactory: TransformerFactory
transformer: Transformer
xmlParserSingleton: XMLParser
XML_EXTENSION: String
INDENT_TRANSFORM: String

XMLParser()
getInstance():XMLParser
addExtension(String):String
creatXmlFile(String,String,String[],String[]):void
addColTypes(String,String[],String[]):void
getColNames(String):String[]
getColTypes(String):String[]
toString(String[]):String
addRow(String,String[],String[]):void
updateRow(String,String[],int):void
getRow(String,int):LinkedList<String>
getRowsNumb(String):int
deleteRaw(String,int):void
SaveXml(Document,String):void
getTableName(Document):String

b) DTDParser:

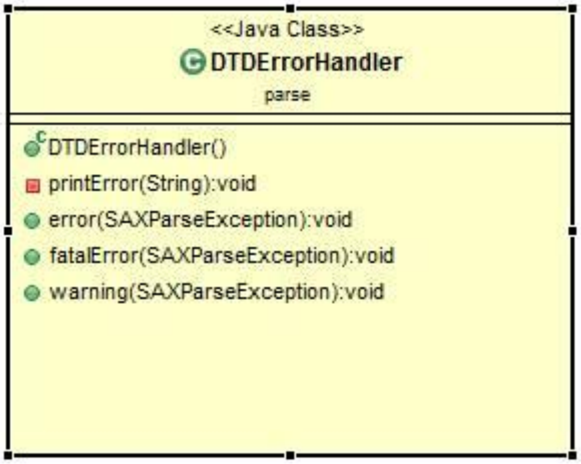
Responsible for creating and saving the DTD files for the tables which contain the structure and the legal elements and attributes of the XML document.



<<Java Class>>
DTDParser
parse
dtdParserSingleton: DTDParser
ENCODING: String
DTD_EXTENSION: String
DTDParser()
getInstance():DTDParser
createDTD(String,String,String[]):void
validateWithDTDU(String):boolean

c) DTDErrorHandler:

Handling Exceptions during the save and load of XML DTD files.

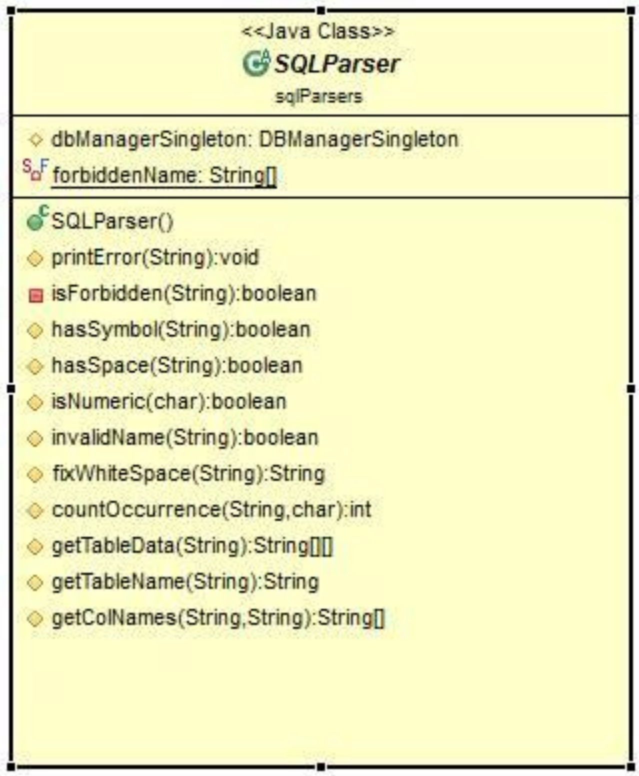


<<Java Class>>
DTDErrorHandler
parse
DTDErrorHandler()
printError(String):void
error(SAXParseException):void
fatalError(SAXParseException):void
warning(SAXParseException):void

IV. sqlParsrs:

a) SQLParser:

Abstract class where all the SQL parsers inherits from it where it contains method that are used in any SQL parser independent from its specific type of parsing.



<<Java Class>>

SQLParser

sqlParsers

◇ dbManagerSingleton: DBManagerSingleton

forbiddenName: String[]

SQLParser()

◇ printError(String):void

isForbidden(String):boolean

◇ hasSymbol(String):boolean

◇ hasSpace(String):boolean

◇ isNumeric(char):boolean

◇ invalidName(String):boolean

◇ fixWhiteSpace(String):String

◇ countOccurrence(String,char):int

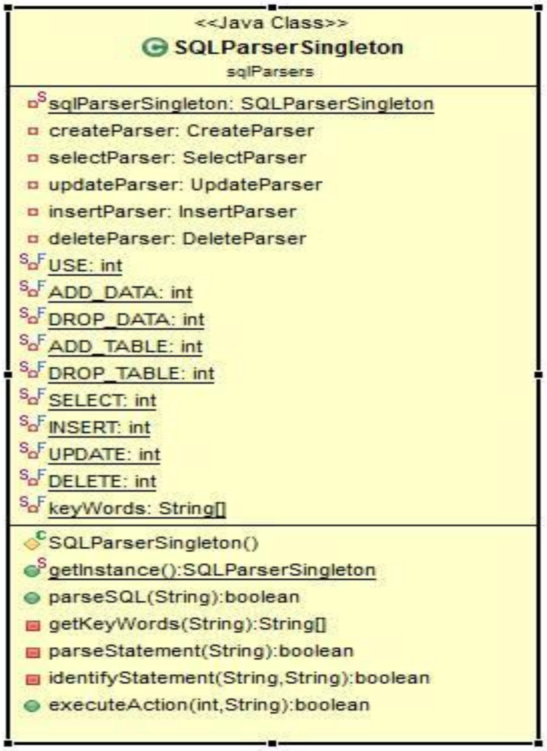
◇ getTableData(String):String[][]

◇ getTableName(String):String

◇ getColNames(String,String):String[]

b) SQLParserSingleton:

Singleton for the parser where it takes the commands parsing it and acting as a fascad deciding which parser to use where there's a creational parser, select parser, update parser, insert parser and delete parser.



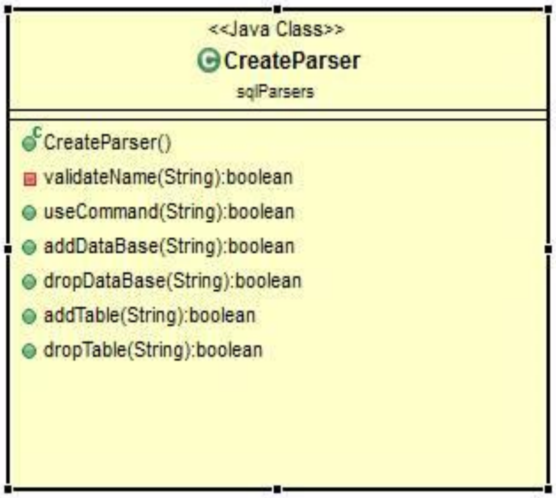
```
<<Java Class>>
SQLParserSingleton
sqlParsers

sqlParserSingleton: SQLParserSingleton
createParser: CreateParser
selectParser: SelectParser
updateParser: UpdateParser
insertParser: InsertParser
deleteParser: DeleteParser
USE: int
ADD_DATA: int
DROP_DATA: int
ADD_TABLE: int
DROP_TABLE: int
SELECT: int
INSERT: int
UPDATE: int
DELETE: int
keyWords: String[]

SQLParserSingleton()
getInstance():SQLParserSingleton
parseSQL(String):boolean
getKeyWords(String):String[]
parseStatement(String):boolean
identifyStatement(String,String):boolean
executeAction(int,String):boolean
```

c) CreateParser:

Parser used to the creation or dropping of new databases or tables or specifying the currently in use database.

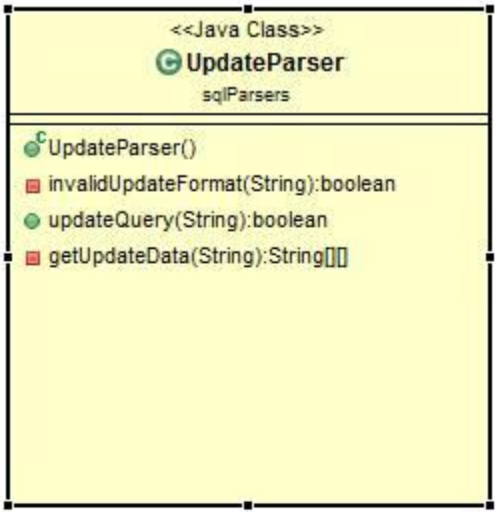


```
<<Java Class>>
CreateParser
sqlParsers

CreateParser()
validateName(String):boolean
useCommand(String):boolean
addDataBase(String):boolean
dropDataBase(String):boolean
addTable(String):boolean
dropTable(String):boolean
```

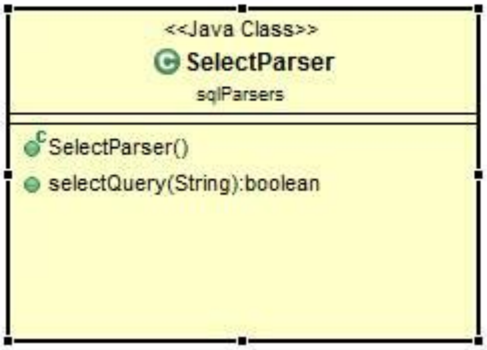
d) UpdateParser:

Parser used to parse the update SQL commands.


<<Java Class>>
UpdateParser
sqlParsers
UpdateParser()
invalidUpdateFormat(String):boolean
updateQuery(String):boolean
getUpdateData(String):String[][]

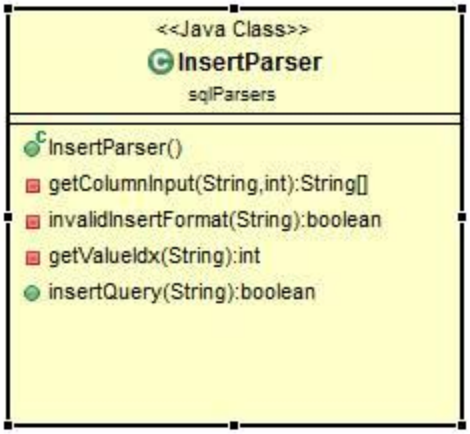
e) SelectParser:

Parser used to parse the select SQL commands.



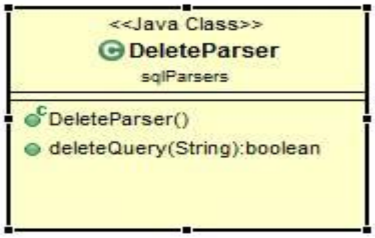
f) InsertParser:

Parser used to parse the insert SQL commands.


<<Java Class>>
InsertParser
sqlParsers
InsertParser()
getColumnInput(String,int):String[]
invalidInsertFormat(String):boolean
getValueIdx(String):int
insertQuery(String):boolean

g) DeleteParser:

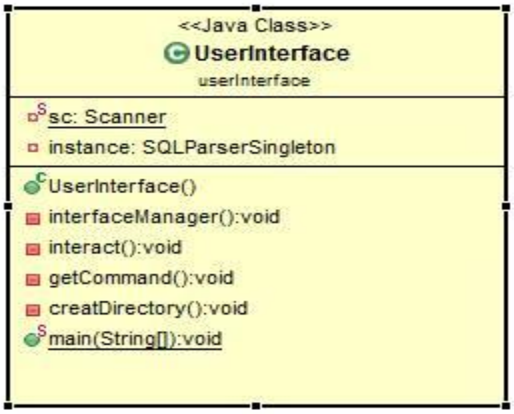
Parser used to parse the SQL delete commands.



V. userInterface:

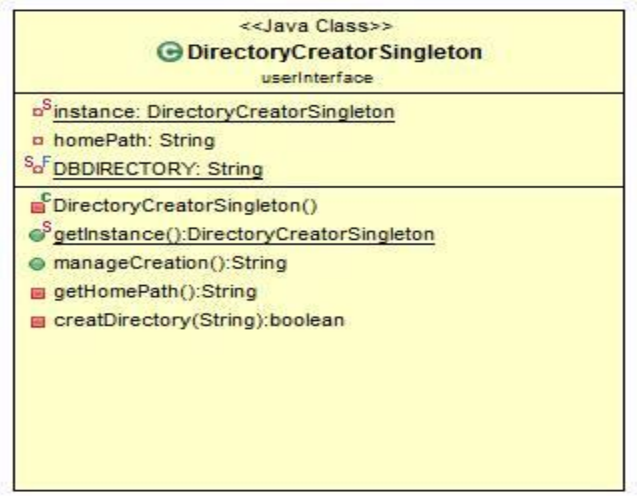
a) UserInterface:

Interface class which acts as a connection between the user and the DBMS where it takes the command sending it to the parser fascad.

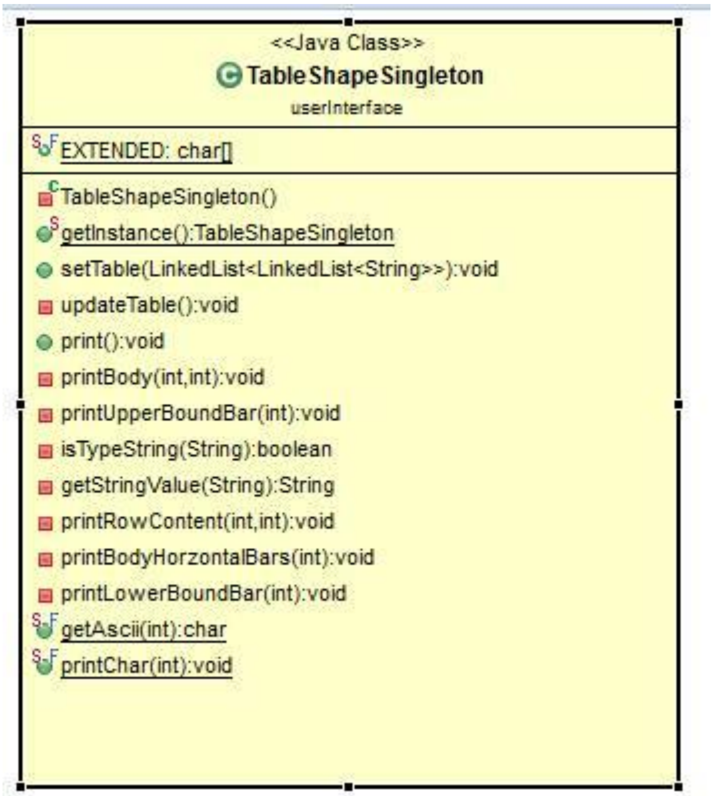


b) DirectoryCreatorSingleton:

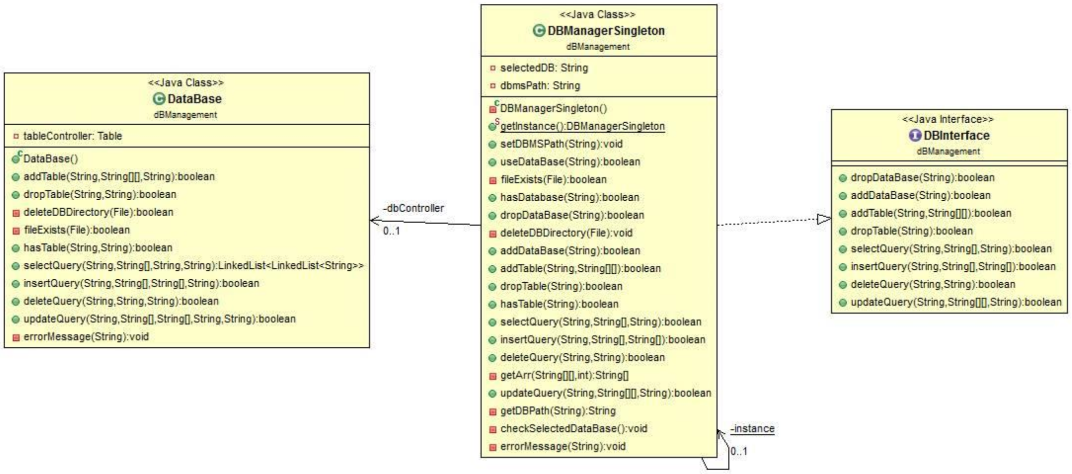
Singleton responsible for creation of the DB directory where the databases data are saved to incase it doesn't exist initially.

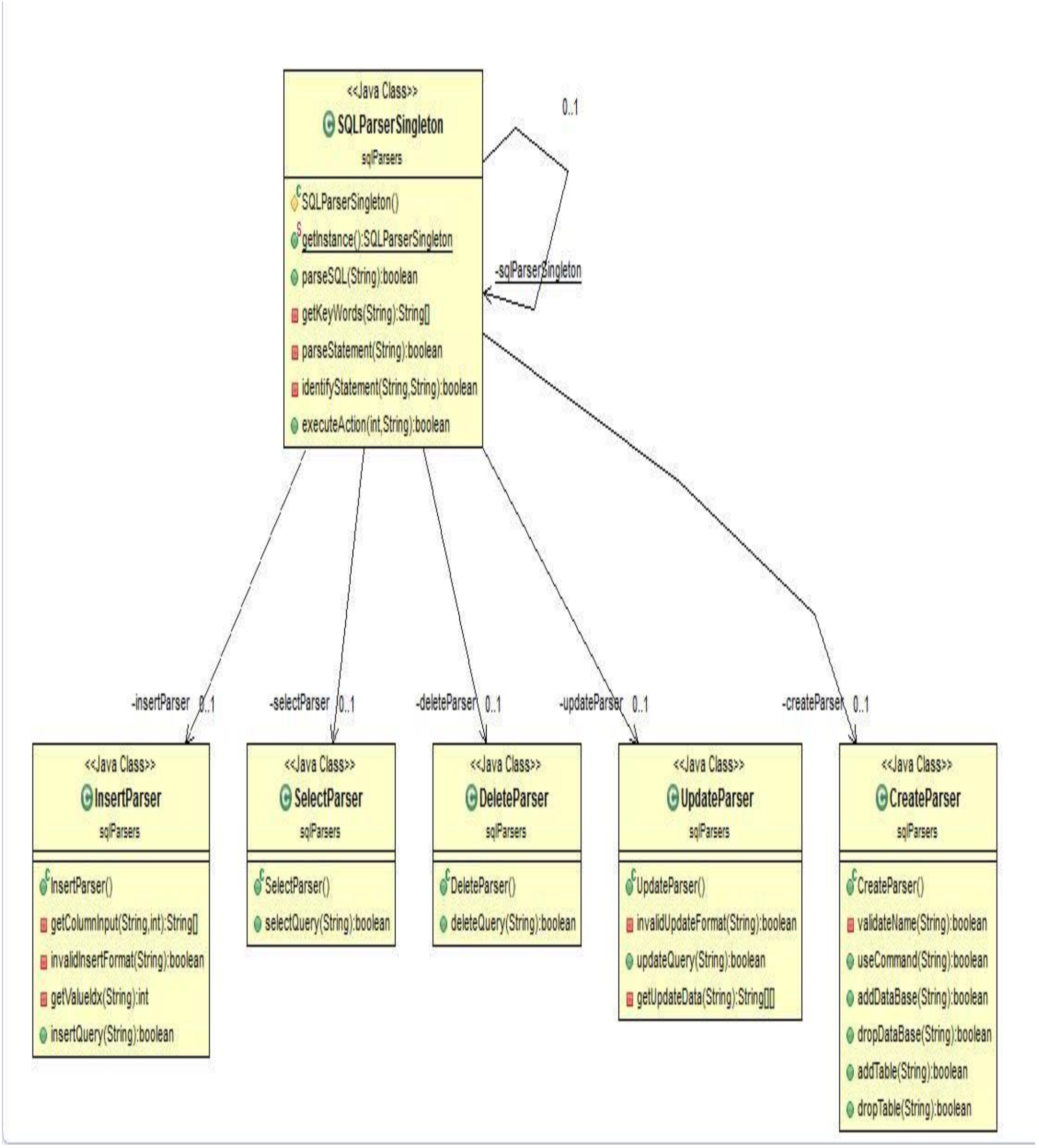

<<Java Class>>
DirectoryCreatorSingleton
userInterface
instance: DirectoryCreatorSingleton
homePath: String
DBDIRECTORY: String
DirectoryCreatorSingleton()
getInstance():DirectoryCreatorSingleton
manageCreation():String
getHomePath():String
creatDirectory(String):boolean

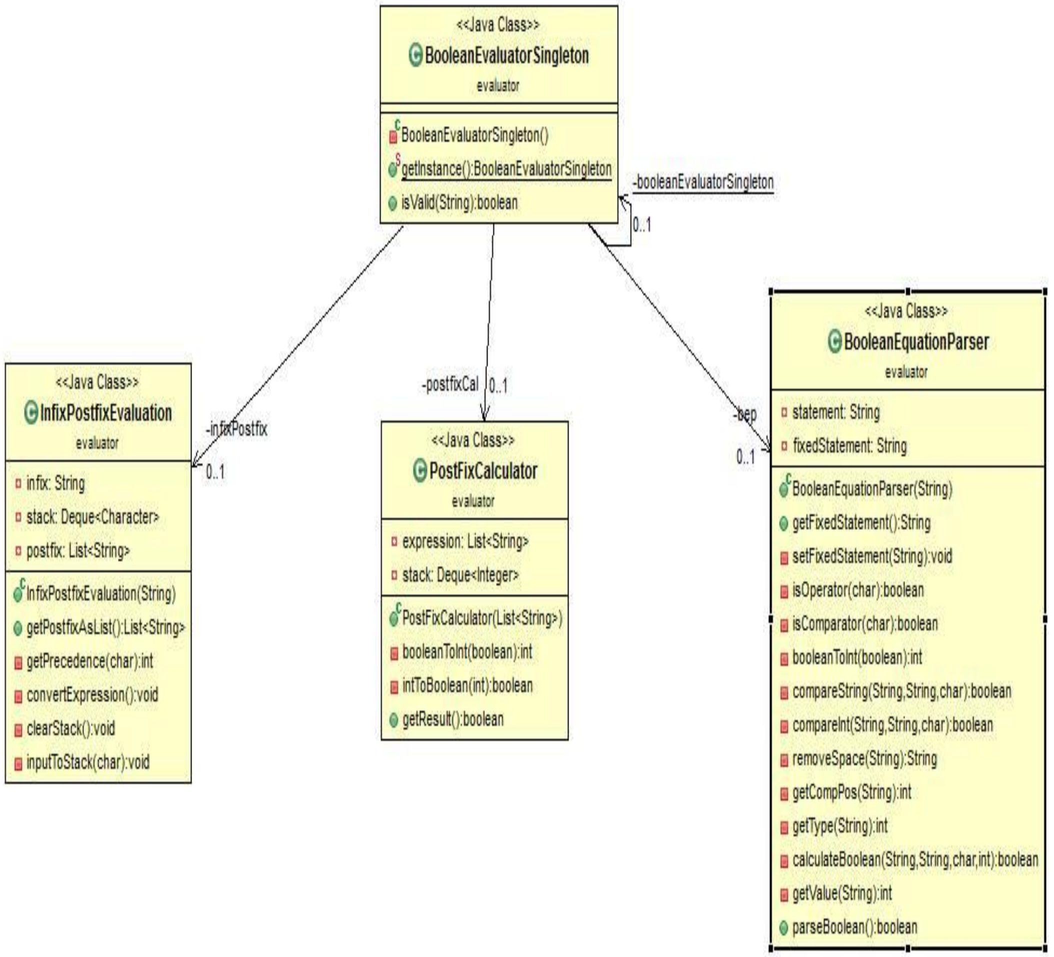
c) <u>TableShapeSingleton:</u>
Singleton responsible for the printing of data selected by the user in the form of tables using extended ASCII not Unicode to be independent from the platform.


<<Java Class>>
TableShapeSingleton
userInterface
EXTENDED: char[]
TableShapeSingleton()
getInstance():TableShapeSingleton
setTable(LinkedList<LinkedList<String>>):void
updateTable():void
print():void
printBody(int,int):void
printUpperBoundBar(int):void
isTypeString(String):boolean
getStringValue(String):String
printRowContent(int,int):void
printBodyHorzontalBars(int):void
printLowerBoundBar(int):void
getAscii(int):char
printChar(int):void

Other important UML diagrams:


<<Java Class>>
DataBase
dBManagement
tableController: Table
DataBase()
addTable(String,String[][],String):boolean
dropTable(String,String):boolean
deleteDBDirectory(File):boolean
fileExists(File):boolean
hasTable(String,String):boolean
selectQuery(String,String[],String,String):LinkedList<LinkedList<String>>
insertQuery(String,String[],String[],String):boolean
deleteQuery(String,String,String):boolean
updateQuery(String,String[],String[],String,String):boolean
errorMessage(String):void
-dbController
0..1
<<Java Class>>
DBManagerSingleton
dBManagement
selectedDB: String
dbmsPath: String
DBManagerSingleton()
getInstance():DBManagerSingleton
setDBMSPath(String):void
useDataBase(String):boolean
fileExists(File):boolean
hasDatabase(String):boolean
dropDataBase(String):boolean
deleteDBDirectory(File):void
addDataBase(String):boolean
addTable(String,String[][]):boolean
dropTable(String):boolean
hasTable(String):boolean
selectQuery(String,String[],String):boolean
insertQuery(String,String[],String[]):boolean
deleteQuery(String,String):boolean
getArr(String[][],int):String[]
updateQuery(String,String[][],String):boolean
getDBPath(String):String
checkSelectedDataBase():void
errorMessage(String):void
-instance
0..1
<<Java Interface>>
DBInterface
dBManagement
dropDataBase(String):boolean
addDataBase(String):boolean
addTable(String,String[][]):boolean
dropTable(String):boolean
selectQuery(String,String[],String):boolean
insertQuery(String,String[],String[]):boolean
deleteQuery(String,String):boolean
updateQuery(String,String[][],String):boolean


<<Java Class>>
SQLParserSingleton
sqlParsers
SQLParserSingleton()
getInstance():SQLParserSingleton
parseSQL(String):boolean
getKeyWords(String):String[]
parseStatement(String):boolean
identifyStatement(String,String):boolean
executeAction(int,String):boolean
0..1
-sqlParserSingleton
-insertParser 0..1
-selectParser 0..1
-deleteParser 0..1
-updateParser 0..1
-createParser 0..1
<<Java Class>>
InsertParser
sqlParsers
InsertParser()
getColumnInput(String,int):String[]
invalidInsertFormat(String):boolean
getValueIdx(String):int
insertQuery(String):boolean
<<Java Class>>
SelectParser
sqlParsers
SelectParser()
selectQuery(String):boolean
<<Java Class>>
DeleteParser
sqlParsers
DeleteParser()
deleteQuery(String):boolean
<<Java Class>>
UpdateParser
sqlParsers
UpdateParser()
invalidUpdateFormat(String):boolean
updateQuery(String):boolean
getUpdateData(String):String[][]
<<Java Class>>
CreateParser
sqlParsers
CreateParser()
validateName(String):boolean
useCommand(String):boolean
addDataBase(String):boolean
dropDataBase(String):boolean
addTable(String):boolean
dropTable(String):boolean



<<Java Class>>
BooleanEvaluatorSingleton
evaluator
BooleanEvaluatorSingleton()
getInstance():BooleanEvaluatorSingleton
isValid(String):boolean
-booleanEvaluatorSingleton
0..1
-infixPostfix
0..1
-postfixCal
0..1
-bep
0..1
<<Java Class>>
InfixPostfixEvaluation
evaluator
infix: String
stack: Deque<Character>
postfix: List<String>
InfixPostfixEvaluation(String)
getPostfixAsList():List<String>
getPrecedence(char):int
convertExpression():void
clearStack():void
inputToStack(char):void
<<Java Class>>
PostFixCalculator
evaluator
expression: List<String>
stack: Deque<Integer>
PostFixCalculator(List<String>)
booleanToInt(boolean):int
intToBoolean(int):boolean
getResult():boolean
<<Java Class>>
BooleanEquationParser
evaluator
statement: String
fixedStatement: String
BooleanEquationParser(String)
getFixedStatement():String
setFixedStatement(String):void
isOperator(char):boolean
isComparator(char):boolean
booleanToInt(boolean):int
compareString(String,String,char):boolean
compareInt(String,String,char):boolean
removeSpace(String):String
getCompPos(String):int
getType(String):int
calculateBoolean(String,String,char,int):boolean
getValue(String):int
parseBoolean():boolean

3) Design patterns:

Design Patterns provide some easy to use OOP solutions to common problems. Some design patterns were really helpful implementing this project.

- Fascad pattern was used due to the increase in the complexity of the DBMS system and large number of interdependent class in the parsers where there is the SQLParserSingleton which acts like fascad where it acts like a single wrapper for all the other parsers such as createParser, deleteParser, insertParser, and updateParser.

  The user would send a command to the parser and the parser would then decide which specific parser to use after doing basic parsing to the command.

  It was an important decision to use fascad here which facilitates the connection between classes and provided a simple interface for the parser singleton.

- Singleton pattern: a creational pattern implemented by creating a class with a method that creates a new instance of the class if one does not exist. If an instance already exists, it simply returns a reference to that object. To make sure that the object cannot be instantiated any other way, the constructor is made private.

  We used the singleton to specifically provide a global access to the database manger where the system can contain only one data base management system and to ensure the there’s only one instance of the class to avoid interference between multiple management objects.

Classes using Singleton pattern:

- DBManagerSingleton.
- BooleanEvaluatorSingleton.
- IntComp.
- StringComp.
- DirectoryCreatorSingleton.
- TableShapeSingleton.
- SQLParserSingleton.
- XMLParser.
- DTDParser.

- Delegation pattern: a fundamental pattern allows an object to delegate one or more tasks to a helper object. Two classes are used to achieve this.

  Classes using Delegation pattern:

  - Table.
  - SQLParserSingleton.
  - DBManagerSingleton
  - DataBase.

- Interface pattern: Interface contains method signatures that are guaranteed to be implemented by the implementing class.

  Interfaces Used:

  - DBInterface.

4) User Guide

The supported commands – case insensitive:

- **Use database**: Selects the database on which the action is to be performed.
- **Create database**: Creates a new database.
- **Drop database**: Deletes the database from the system.
- **Create table**: Creates a new table in the selected database.
- **Drop table:** Deletes the table from the database.
- **Insert into table:** adds a new record of data to the table.
- **Select from table:** Prints the selected columns and records to the user.
- **Delete from table:** Deletes records from the table.
- **Update table:** Updated existing records in the table.

Perform an action on records that meets a certain conditions:

After the command type “Where” followed by the condition.

Select to print the whole table:

```
Select * from table.
```

Tables / databases names are case sensitive.

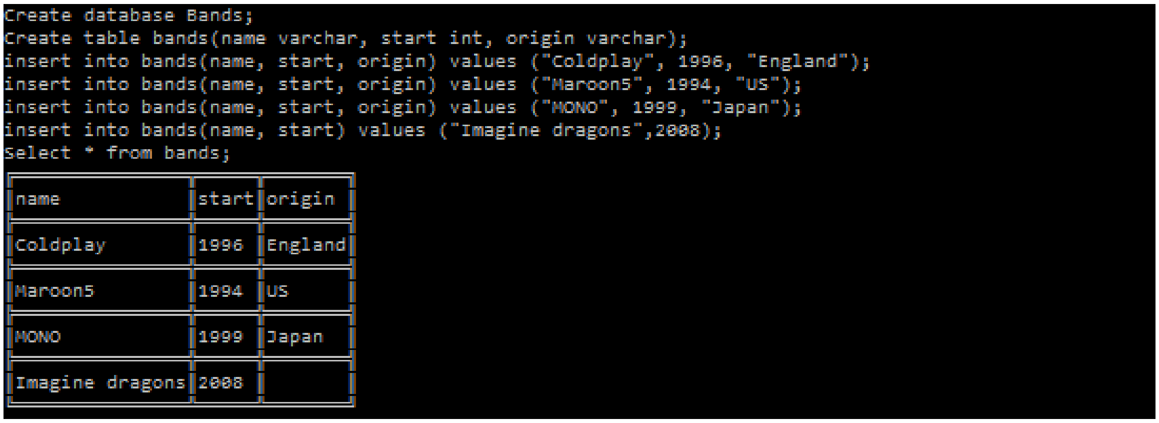
Illustrative example:

1. Create a new table “bands” that has 3 columns [name, start, origin] in the selected database.
   ```
   >> CREATE TABLE bands (name varchar, start int, origin varchar);
   ```

2. Insert records to the table:
   ```
   >> INSERT INTO bands (name, start, origin)VALUES(“Coldplay”, 1996, “England”);
   >> INSERT INTO bands (name, start, origin)VALUES(“Maroon5”, 1994, “US”);
   >> INSERT INTO bands (name, start, origin) VALUES(“MONO”, 1999, “Japan”);
   >> INSERT INTO bands (name, start)
   ```

VALUES (“Imagine Dragons”, 2008);

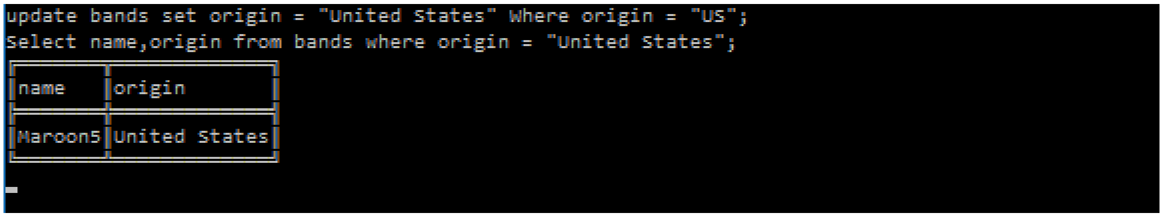
3. Print the “bands” table:
   >> Select * from bands;



```
Create database Bands;
Create table bands(name varchar, start int, origin varchar);
insert into bands(name, start, origin) values ("Coldplay", 1996, "England");
insert into bands(name, start, origin) values ("Maroon5", 1994, "US");
insert into bands(name, start, origin) values ("MONO", 1999, "Japan");
insert into bands(name, start) values ("Imagine dragons",2008);
Select * from bands;
╔════════════════╦═════╦═══════╗
║name            ║start║origin ║
╠════════════════╬═════╬═══════╣
║Coldplay        ║1996 ║England║
╠════════════════╬═════╬═══════╣
║Maroon5         ║1994 ║US     ║
╠════════════════╬═════╬═══════╣
║MONO            ║1999 ║Japan  ║
╠════════════════╬═════╬═══════╣
║Imagine dragons ║2008 ║       ║
╚════════════════╩═════╩═══════╝
```

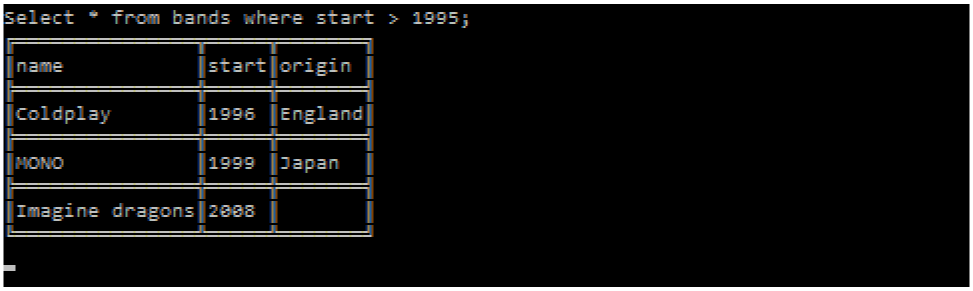
4. Update selected records:
   >> UPDATE bands SET origin = “United States”
   WHERE origin = “US”;

5. Print selected columns and records from “bands” table:
   >> SELECT name, origin FROM bands;



```
update bands set origin = "United States" where origin = "US";
Select name,origin from bands where origin = "United States";
╔═══════╦═════════════╗
║name   ║origin       ║
╠═══════╬═════════════╣
║Maroon5║United States║
╚═══════╩═════════════╝
```

6. Print the records that meets a certain condition.
   >> select * from bands where start > 1995;



```
Select * from bands where start > 1995;
╔════════════════╦═════╦═══════╗
║name            ║start║origin ║
╠════════════════╬═════╬═══════╣
║Coldplay        ║1996 ║England║
╠════════════════╬═════╬═══════╣
║MONO            ║1999 ║Japan  ║
╠════════════════╬═════╬═══════╣
║Imagine dragons ║2008 ║       ║
╚════════════════╩═════╩═══════╝
```