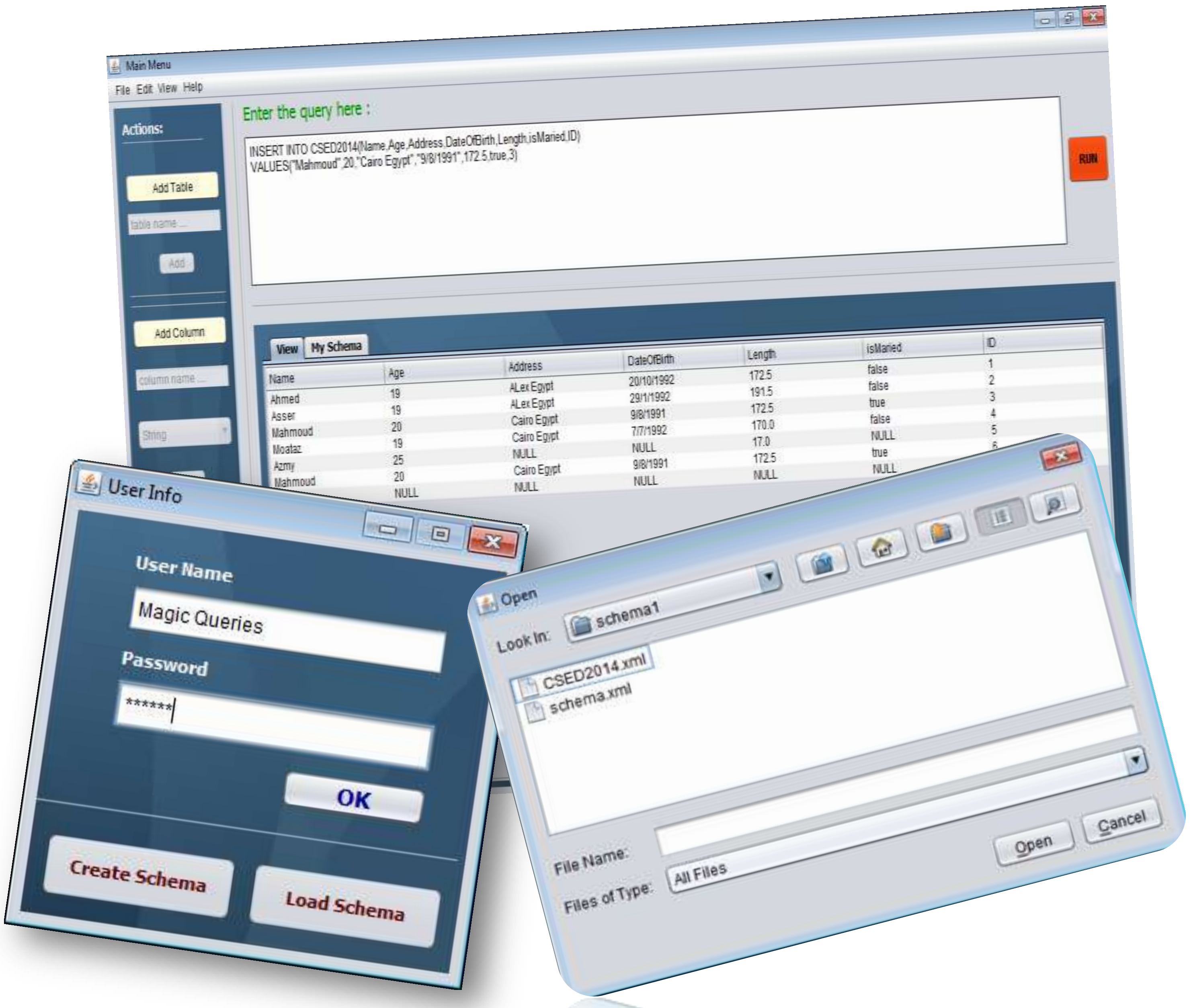
2011

MagicQueries

Developed By :

Mahmoud Nabil El-magrhayby
Ahmed Hesham Fathy
Asser Mohamed Hassan
Moataz Gamal El-Shaer
Mohamed Osama Azmy
Mohamed Ahmed El-Sayed

Magic Queries



Enter the query here :

```
INSERT INTO CSED2014(Name,Age,Address,DateOfBirth,Length,isMaried,ID)
VALUES("Mahmoud",20,"Cairo Egypt","9/8/1991",172.5,true,3)
```

Name	Age	Address	DateOfBirth	Length	isMaried	ID
Ahmed	19	ALex Egypt	20/10/1992	172.5	false	1
Asser	19	ALex Egypt	29/1/1992	191.5	false	2
Mahmoud	20	Cairo Egypt	9/8/1991	172.5	true	3
Moataz	19	Cairo Egypt	7/7/1992	170.0	false	4
Azmy	25	NULL	NULL	17.0	NULL	5
Mahmoud	20	Cairo Egypt	9/8/1991	172.5	true	6
	NULL	NULL	NULL	NULL	NULL	

User Name: Magic Queries

Password: ******

Look In: schema1

CSED2014.xml

schema.xml

Files of Type: All Files

Section 1:

User Guide

In this program, user can create his own data base, add tables ,add columns ,delete tables ,insert data in tables , delete data from tables and change data in tables.

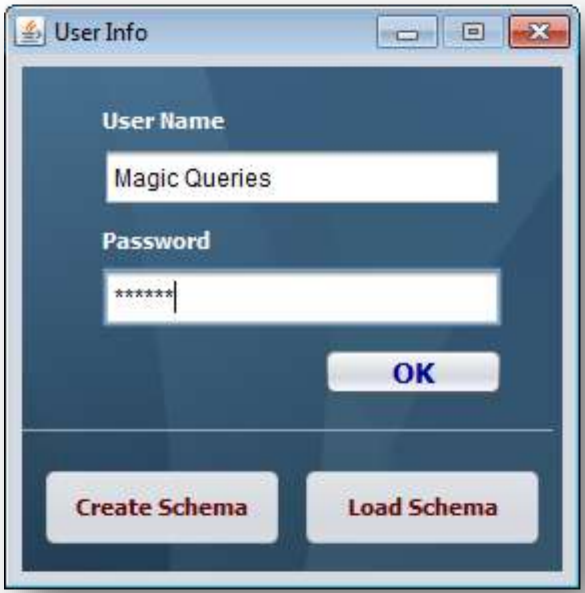
The user can see every change in his data base in a table, the table will be updated after any action the user enters.

The user should enter the valid user name and password to access Magic Queries, then the user can create new database or load previously created database.

User Info:

Username: Magic Queries

Password : MAMMAM



Magic Queries

******

The user can enter any query that applies an action on the data base

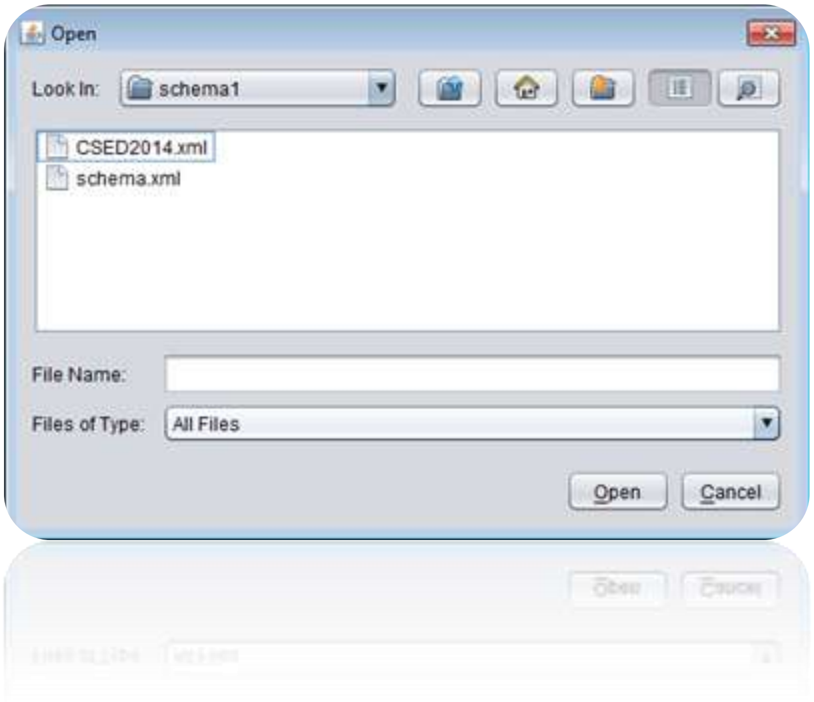
Queries:

1. Insert :

EX:

INSERT INTO Table_Name
(column_Name,…..)
VALUES (“STRING_VALUE”,………….)

This query inserts the data that the user add into the specified table.



2. Delete:

   EX:

   DELETE Table_Name

   WHERE Column_NameOperator Value

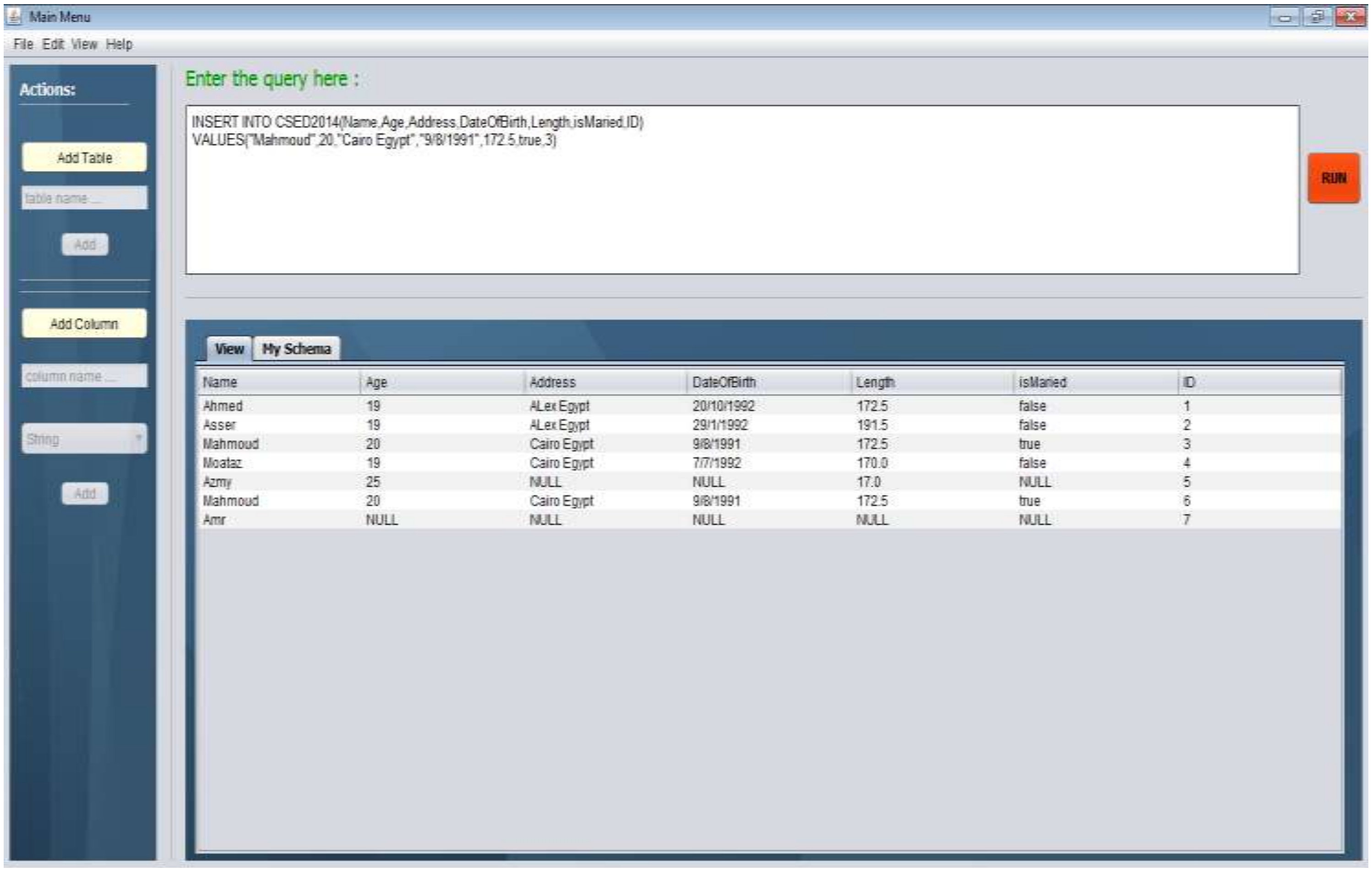
   - Delete all:
     Delete all data from the table.
   - Delete :
     Delete data from the specified cell.

3. Update:

   EX:
   UPDATE Table_Name
   SET Column_Name = Value , Column_Name = Value ,…………
   WHERE Column_Name Operator Value

   Change the values in the specified cells.

4. Select:



Enter the query here :

```
INSERT INTO CSED2014(Name,Age,Address,DateOfBirth,Length,isMaried,ID)
VALUES("Mahmoud",20,"Cairo Egypt","9/8/1991",172.5,true,3)
```

Name	Age	Address	DateOfBirth	Length	isMaried	ID
Ahmed	19	ALex Egypt	20/10/1992	172.5	false	1
Asser	19	ALex Egypt	29/1/1992	191.5	false	2
Mahmoud	20	Cairo Egypt	9/8/1991	172.5	true	3
Moataz	19	Cairo Egypt	7/7/1992	170.0	false	4
Azmy	25	NULL	NULL	17.0	NULL	5
Mahmoud	20	Cairo Egypt	9/8/1991	172.5	true	6
Amr	NULL	NULL	NULL	NULL	NULL	7

EX:

Select *|(Column_Name,………..)
FROM Table_Name
WHERE Column_Name Operator Value

- Select all:
  Returns all data in the table.
- Select:
  Returns the specified cells from the table.

"All the queries are Line separated."

UML Diagram

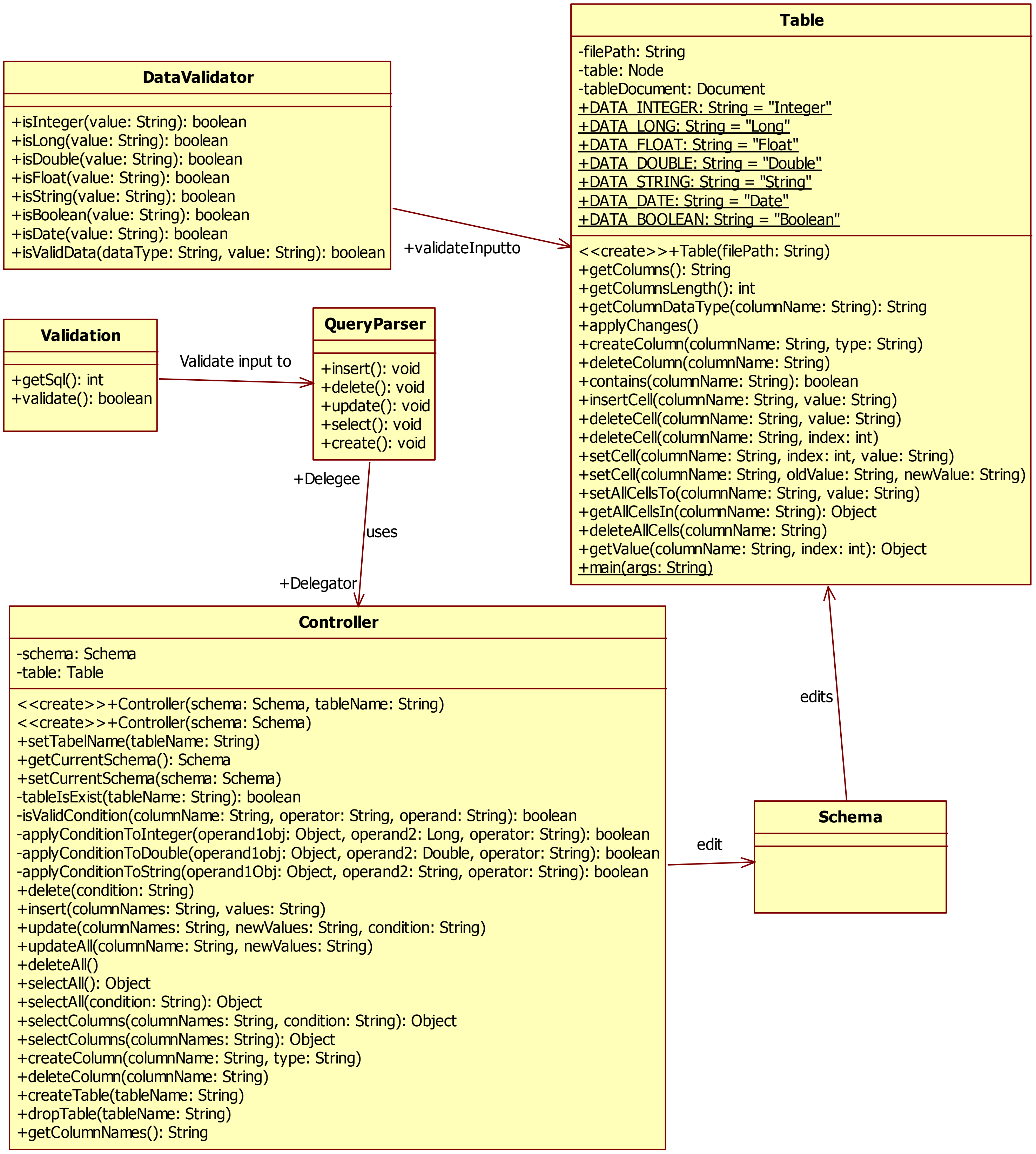


Table
-filePath: String
-table: Node
-tableDocument: Document
+DATA_INTEGER: String = "Integer"
+DATA_LONG: String = "Long"
+DATA_FLOAT: String = "Float"
+DATA_DOUBLE: String = "Double"
+DATA_STRING: String = "String"
+DATA_DATE: String = "Date"
+DATA_BOOLEAN: String = "Boolean"
<<create>>+Table(filePath: String)
+getColumns(): String
+getColumnsLength(): int
+getColumnDataType(columnName: String): String
+applyChanges()
+createColumn(columnName: String, type: String)
+deleteColumn(columnName: String)
+contains(columnName: String): boolean
+insertCell(columnName: String, value: String)
+deleteCell(columnName: String, value: String)
+deleteCell(columnName: String, index: int)
+setCell(columnName: String, index: int, value: String)
+setCell(columnName: String, oldValue: String, newValue: String)
+setAllCellsTo(columnName: String, value: String)
+getAllCellsIn(columnName: String): Object
+deleteAllCells(columnName: String)
+getValue(columnName: String, index: int): Object
+main(args: String)
DataValidator
+isInteger(value: String): boolean
+isLong(value: String): boolean
+isDouble(value: String): boolean
+isFloat(value: String): boolean
+isString(value: String): boolean
+isBoolean(value: String): boolean
+isDate(value: String): boolean
+isValidData(dataType: String, value: String): boolean
+validateInputto
Validation
+getSql(): int
+validate(): boolean
Validate input to
QueryParser
+insert(): void
+delete(): void
+update(): void
+select(): void
+create(): void
+Delegee
uses
+Delegator
Controller
-schema: Schema
-table: Table
<<create>>+Controller(schema: Schema, tableName: String)
<<create>>+Controller(schema: Schema)
+setTabelName(tableName: String)
+getCurrentSchema(): Schema
+setCurrentSchema(schema: Schema)
-tableIsExist(tableName: String): boolean
-isValidCondition(columnName: String, operator: String, operand: String): boolean
-applyConditionToInteger(operand1obj: Object, operand2: Long, operator: String): boolean
-applyConditionToDouble(operand1obj: Object, operand2: Double, operator: String): boolean
-applyConditionToString(operand1Obj: Object, operand2: String, operator: String): boolean
+delete(condition: String)
+insert(columnNames: String, values: String)
+update(columnNames: String, newValues: String, condition: String)
+updateAll(columnName: String, newValues: String)
+deleteAll()
+selectAll(): Object
+selectAll(condition: String): Object
+selectColumns(columnNames: String, condition: String): Object
+selectColumns(columnNames: String): Object
+createColumn(columnName: String, type: String)
+deleteColumn(columnName: String)
+createTable(tableName: String)
+dropTable(tableName: String)
+getColumnNames(): String
edits
edit
Schema

Section 2 :

Main Classes and Algorithms

Schema Class:

Attributes:

1- name: String.
2- tableNames: ArrayList <String>.
3- table: Node.
4- tableDocument : Document .

methods :

1-

- signature : `public Schema(String url, String schemaName, boolean isExist);`
- @param :
  - url : the directory of the schema file which we want to load.
  - schemaName : the schema Name.
  - isExist: a flag to know whether this schema is will be loaded or Not.
- Description: this constructor is invoked when we want to load an existing schema.

2-

- signature : `public Schema(String url, String schemaName) ;`
- @param :
  - url : the directory of the schema file which we want to load.
  - schemaName : the schema Name.
- Description: the default constructor of fo creating a new schema.

3-

- signature : `public void createTable(String tableName);`
- @param :
  - tableName: the name of the new table.
- Description: this method is to create a new table in the current schema.

4-

- signature : `public void dropTable(String tableName);`
- @param :
  - tableName: the name the table which we want to delete it.
- Description: this method is to delete the table.

5-

- signature : `public void applyChanges();`
- Description: this method is to apply all the changes and write all the updates in schema file in its URI.

Table Class:

Attributes:

1- filePath :String.
2- table: Node.
3- tableDocument.
4- DATA_INTEGER: String.
5- DATA_LONG: String.
6- DATA_FLOAT: String.
7- DATA_DOUBLE: String.
8- DATA_STRING: String.
9- DATA_DATE: String.
10- DATA_BOOLEAN: String.

methods :

1-

- Signature : **public** Table(String filePath);
- @param: String contains the path of the xml file.
- Description: The default constructor to load a table.

2-

- Signature : **public** String[] getColumns();
- Return: Array of strings contains the names of the columns.
- Description: This method is to get the names of all the columns in the table.

3-

- Signature : **public** String getColumnDataType(String columnName);
- @param: column name
- Return: The name of the data type which this column supports.
- Description: This method to get column Data Type

4-

- Signature : **public void** applyChanges();
- Description: This method is to apply chnges to the actual XML file of the table.

5-

- Signature : **public void** createColumn(String columnName, String type);
- @param: column name, the type of the entry.
- Description: method to create and append a new column to table.

6-

- Signature : **public void** deleteColumn(String columnName);
- @param: Column name.
- Description: this method delete a column from the table.

7-

- Signature : `public boolean contains(String columnName);`
- @param: column name.
- Return: true if this column exist in the table and false if not.
- Description: `Method to determine whether the table contain a certain column or not`

8-

- Signature : `public void insertCell(String columnName, String value);`
- @param: column name and the value of the cell which we want to insert.
- Description: insert a value in the inserted column.

9-

- Signature : `public void deleteCell(String columnName, String value);`
- @param: column name and value which we want to delete it.
- Description: This method delete a value from a specified column.

10-

- Signature : `public void deleteCell(String columnName, int index);`
- @param: Column name and the index of cell in this column.
- Description: this method delete a certain cell in the column which names is equal to the columnName.

11-

- Signature : `public void setCell(String columnName, int index, String value);`
- @param: Column name , index of the cell and the new value of this cell.
- Description: Method to set the value of a certain cell in the column by its index.

12-

- Signature : `public void setAllCellsTo(String columnName, String value);`
- @param: column name and the value
- Description: set all the cell by the new value.

13-

- Signature : `public Object[] getAllCellsIn(String columnName);`
- @param: column name.
- Return: object array hold all the cells in this column.
- Description: Method to get all the column in the column.

14-

- Signature : `public void deleteAllCells(String columnName);`
- @param: column name.
- Description: method to delete all the cells in this column.

15-

- Signature : `public Object getValue(String columnName, int index);`
- @param: column name and the index.
- Return: the value of this cell.

Algorithms:

1- createTable :

```
createTable(String tableName) throws Exception
{
if (not contains(tableName)) {
                    try {

          tableNames.add(tableName);
                                        Node schema <- table;
                                        Element newTable <- tableDocument.createElement("Table");
                                        newTable.appendChild(tableDocument.createTextNode(tableName));
                                        schema.appendChild(newTable);
                                        table <- tableDocument.getFirstChild();

                                        try {
                                                  DocumentBuilderFactory docFactory <- DocumentBuilderFactory
                                                                      .newInstance();
                                                  DocumentBuilder builder = docFactory.newDocumentBuilder();
                                                  Document doc <- builder.newDocument();
                                                  Element rootElement <- doc.createElement("table");
                                                  rootElement.setAttribute("Name", tableName);
                                                  doc.appendChild(rootElement);
                                                  DOMSource source <- new DOMSource(doc);
                                                  StreamResult result <- new StreamResult(new File(
                                                                      getTableURL(tableName)));
                                                  TransformerFactory transformerFactory <- TransformerFactory
                                                                      .newInstance();
                                                  Transformer transformer <- transformerFactory
                                                                      .newTransformer();
                                                  transformer.transform(source, result);
                                        } catch (Exception) {
                                                   ("error");
                                        }
                              } catch (Exception) {

                    }
                    } else {
                              throw new Exception("Table already exists"); } }
```

2- dropTable:

```
dropTable(String tableName) throws Exception {
                    if (contains(tableName)) {
                              try {
                                        Node schema <- table;
                                        NodeList list <- schema.getChildNodes();
                                        for (int i <- 0; i < list.getLength(); i++) {
                                                  Node node <- list.item(i);
                                                  if (tableName.equalsIgnoreCase(node.getTextContent())) {
```

```
                                        schema.removeChild(node);
                                    }
                                }
                                table = tableDocument.getFirstChild();

                                // delete the XML file of the deleted table.
                                File file <- new File(getTableURL(tableName));
                                file.delete();
                                tableNames.remove((String)tableName);
                        } catch (Exception e)
                        {
                                throw new Exception("Error creating the schema");
                        }
                } else
                {
                throw new Exception("Table not found");
                }
        }
```

3- createColumn :

```
createColumn(String columnName, String type) {

                // append a new node -column- to table
                if (!contains(columnName)) {
                        Element newColumn <- tableDocument.createElement("column");
                        newColumn.setAttribute("dataType", type);
                        newColumn.setAttribute("name", columnName);
                        table.appendChild(newColumn);
                } else {
                        throw new Exception("Column already exist");
                }
        }
```

4- deleteColumn :

```
deleteColumn(String columnName) {
                // TODO Auto-generated method stub

                NodeList columns <- tableDocument.getElementsByTagName("column");

                boolean isFound <- false;

                for (int i = 0; i < columns.getLength(); i++) {
                        Node node <- columns.item(i);
                        NamedNodeMap attr <- node.getAttributes();
                        Node nodeAttr <- attr.getNamedItem("name");

                        if (nodeAttr.getTextContent().equals(columnName)) {
                                table.removeChild(node);
                                isFound <- true;
                                break;
                        }
                }
                if (!isFound) {
                        throw new Exception("Column Not Found");
                }

        }
```

5- contains :

```
contains(String columnName) {
                NodeList columns <- tableDocument.getElementsByTagName("column");

                boolean isFound <- false;

                for (int i <- 0; i < columns.getLength(); i++) {
                        Node node <- columns.item(i);
                        NamedNodeMap attr <- node.getAttributes();
                        Node nodeAttr <- attr.getNamedItem("name");

                        if (nodeAttr.getTextContent().equals(columnName)) {
                                isFound <- true;
                                break;
                        }
                }
                return isFound;
        }
```

6 –insertCell :

```
insertCell(String columnName, String value) throws Exception {

                DataValidator validator <- new DataValidator();
                String dataType <- getColumnDataType(columnName);
                boolean isValid <- validator.isValidData(dataType, value);

                if (isValid) {
                        boolean isFound <- false;

                        NodeList columns <- tableDocument.getElementsByTagName("column");

                        for (int i <- 0; i < columns.getLength(); i++) {
                                Node node <- columns.item(i);
                                NamedNodeMap attr <- node.getAttributes();
                                Node nodeAttr <- attr.getNamedItem("name");

                                if (nodeAttr.getTextContent().equals(columnName)) {
                                        Element newElement <-tableDocument.createElement(columnName
                                                        + "Element");
                                        newElement.setTextContent(value);
                                        node.appendChild(newElement);
                                        isFound = true;
                                        break;
                                }
                        }
                if (!isFound) {
                    throw new Exception("Column not Found");
                }
            } else {
                throw new Exception("Wrong Data Type");
            }
        }
```

7 – deleteCell :

```
deleteCell(String columnName, String value) throws Exception {

            boolean isFound <- false;

            NodeList columns <- tableDocument.getElementsByTagName("column");

            for (int i <- 0; i < columns.getLength(); i++) {
                Node node <- columns.item(i);
                NamedNodeMap attr <- node.getAttributes();
                Node nodeAttr <- attr.getNamedItem("name");

                if (nodeAttr.getTextContent().equals(columnName)) {
                    NodeList values <- tableDocument.getElementsByTagName(columnName
                                + "Element");

                    boolean isValueFound <- false;

                    for (int j <- 0; j < values.getLength(); j++) {
                        Node valueNode <- values.item(j);

                        if (valueNode.getTextContent().equals(value)) {
                            node.removeChild(valueNode);
                            isValueFound = true;
                            break;
                        }
                    }

                    if (!isValueFound) {
                        throw new Exception("Value Not Found");
                    }

                    isFound <- true;
                    break;
                }
            }

            if (!isFound) {
                throw new Exception("Column not Found");
            }
        }
```

8 – deleteCell (2)

```
deleteCell(String columnName, int index) {
        boolean isFound <- false;

        NodeList columns <- tableDocument.getElementsByTagName("column");

        for (int i <- 0; i < columns.getLength(); i++) {
                Node node <- columns.item(i);
                NamedNodeMap attr <- node.getAttributes();
                Node nodeAttr <- attr.getNamedItem("name");

                if (nodeAttr.getTextContent().equals(columnName)) {
                        NodeList values = tableDocument.getElementsByTagName(columnName
                                        + "Element");

                        try {
                                Node valueNode <- values.item(index);
                                node.removeChild(valueNode);
                        } catch (Exception e) {
                                throw new Exception("Invalid index");
                        }
                        isFound <- true;
                        break;
                }
        }

        if (!isFound) {
                throw new Exception("Column not Found");
        } }
```

9- setCell

```
setCell(String columnName, int index, String value)
                {

        DataValidator validator <- new DataValidator();
        String dataType <- getColumnDataType(columnName);
        boolean isValid <- validator.isValidData(dataType, value);

        if (isValid) {
                boolean isFound <- false;

                NodeList columns <- tableDocument.getElementsByTagName("column");

                for (int i <- 0; i < columns.getLength(); i++) {
                        Node node <- columns.item(i);
                        NamedNodeMap attr <- node.getAttributes();
                        Node nodeAttr <- attr.getNamedItem("name");

                        if (nodeAttr.getTextContent().equals(columnName)) {
                                NodeList values <- tableDocument
                                                .getElementsByTagName(columnName + "Element");
                                try {
                                        Node valueNode <- values.item(index);
                                        valueNode.setTextContent(value);
                                } catch (Exception e) {
                                        throw new Exception("Invalid index");
                                }
                                isFound <- true;
                                break;
                        }
                }

                if (!isFound) {
                        throw new Exception("Column not Found");
                }
        } else {
                throw new Exception("Invalid Data Type");
        }
}
```

10 – setCell :

```
setCell(String columnName, String oldValue, String newValue)
                {

        DataValidator validator <- new DataValidator();
        String dataType <- getColumnDataType(columnName);
        boolean isValid <- validator.isValidData(dataType, oldValue);

        if (isValid) {

                boolean isFound <- false;

                NodeList columns <- tableDocument.getElementsByTagName("column");

                for (int i <- 0; i < columns.getLength(); i++) {
                        Node node <- columns.item(i);
                        NamedNodeMap attr <- node.getAttributes();
                        Node nodeAttr < attr.getNamedItem("name");

                        if (nodeAttr.getTextContent().equals(columnName)) {
                                NodeList values <- tableDocument
                                                .getElementsByTagName(columnName + "Element");

                                boolean isValueFound <- false;

                                for (int j <- 0; j < values.getLength(); j++) {
                                        Node valueNode <- values.item(j);

                                        if (valueNode.getTextContent().equals(oldValue)) {
                                                valueNode.setTextContent(newValue);
                                                isValueFound <- true;
                                                break;
                                        }
                        }

                        if (!isValueFound) {
                                throw new Exception("Value Not Found");
                        }

                        isFound <- true;
                        break;
                    }
                }

                if (!isFound) {
                    throw new Exception("Column not Found");
                }
            } else {
                throw new Exception("Invalid Data Type");
            }
    }
```

11 – setAllCellsTo

```
setAllCellsTo(String columnName, String value) {

        DataValidator validator <- new DataValidator();
        String dataType <- getColumnDataType(columnName);
        boolean isValid <- validator.isValidData(dataType, value);

        if (isValid) {

            boolean isFound <- false;

            NodeList columns <- tableDocument.getElementsByTagName("column");

            for (int i <- 0; i < columns.getLength(); i++) {
                Node node <- columns.item(i);
                NamedNodeMap attr <- node.getAttributes();
                Node nodeAttr <- attr.getNamedItem("name");

                if (nodeAttr.getTextContent().equals(columnName)) {
                    NodeList values <- tableDocument
                                .getElementsByTagName(columnName + "Element");

                    for (int j <- 0; j < values.getLength(); j++) {
                        Node valueNode = values.item(j);
                        valueNode.setTextContent(value);
                    }

                    isFound <- true;
                    break;
                }
            }

            if (!isFound) {
                throw new Exception("Column not Found");
```

```
                    }
            } else {
                    throw new Exception("Invalid Data Type");
            }
    }
```

12 – getAllCellsIn

```
Object[] getAllCellsIn(String columnName) throws Exception {

        boolean isFound <- false;

        Object[] valuesObjects <- null;

        NodeList columns = tableDocument.getElementsByTagName("column");

        for (int i <- 0; i < columns.getLength(); i++) {
                Node node <- columns.item(i);
                NamedNodeMap attr <- node.getAttributes();
                Node nodeAttr <- attr.getNamedItem("name");

                if (nodeAttr.getTextContent().equals(columnName)) {
                        NodeList values <- tableDocument.getElementsByTagName(columnName
                                        + "Element");

                        valuesObjects <- new Object[values.getLength()];

                        for (int j = 0; j < values.getLength(); j++) {
                                Node valueNode <- values.item(j);
                                Object currentObject<-null;
                                String dataType <- getColumnDataType(columnName);
                                String value = valueNode.getTextContent();

                                if(!value.equals("null"))
                                {
                                if (dataType.equals("Integer")) {
                                        currentObject <- Integer.parseInt(value);
                                }else if(dataType.equals("Long")) {
                                        currentObject <- Long.parseLong(value);
                                }
                                else if (dataType.equals("Float")) {
                                        currentObject <- Float.parseFloat(value);
                                } else if (dataType.equals("Double")) {
                                        currentObject <- Double.parseDouble(value);
                                } else if (dataType.equals("Boolean")) {
                                        currentObject <- Boolean.parseBoolean(value);
                                } else if (dataType.equals("String")) {
                                        currentObject <- value;
                                } else if (dataType.equals("Date")) {
                                        currentObject <- value;
                                } else {
                                        currentObject <- null;
                                }
                                }
                        valuesObjects[j] <- currentObject;
                    }

                    isFound <- true;
                    break;
                }
            }

            if (!isFound) {
                throw new Exception("Column not Found");
            }

            return valuesObjects;
        }
```

13 – deleteAllCells:

```
deleteAllCells(String columnName) throws Exception {

            boolean isFound <- false;

            NodeList columns <- tableDocument.getElementsByTagName("column");

            for (int i <- 0; i < columns.getLength(); i++) {
                Node node <- columns.item(i);
                NamedNodeMap attr <- node.getAttributes();
                Node nodeAttr <- attr.getNamedItem("name");

                if (nodeAttr.getTextContent().equals(columnName)) {
                    NodeList values <- tableDocument.getElementsByTagName(columnName
                                    + "Element");

                    int length <- values.getLength();
                    for (int j <- 0; j < length; j++) {
                        Node valueNode <- values.item(0);
                        node.removeChild(valueNode);
                    }

                    isFound <- true;
                    break;
                }
            }

            if (!isFound) {
                throw new Exception("Column not Found");
            }

        }
```

14 – getValue:

```
getValue(String columnName, int index) throws Exception {
        boolean isFound <- false;

        NodeList columns <- tableDocument.getElementsByTagName("column");
        String value <- "";
        for (int i <- 0; i < columns.getLength(); i++) {
                Node node <- columns.item(i);
                NamedNodeMap attr <- node.getAttributes();
                Node nodeAttr <- attr.getNamedItem("name");

                if (nodeAttr.getTextContent().equals(columnName)) {
                        NodeList values <- tableDocument.getElementsByTagName(columnName
                                        + "Element");

                        try {
                                Node valueNode <- values.item(index);
                                value <- valueNode.getTextContent();
                        } catch (Exception e) {
                                throw new Exception("Invalid index");
                        }
                        isFound <- true;
                        break;
                }
        }

        if (!isFound) {
                throw new Exception("Column not Found");
        }

        String dataType <- getColumnDataType(columnName);

        if(!value.equals("null"))
        {
        if (dataType.equals(DATA_INTEGER)) {
                return (Object) Integer.parseInt(value);
        } else if (dataType.equals(DATA_FLOAT)) {
                return (Object) Float.parseFloat(value);
        } else if (dataType.equals(DATA_DOUBLE)) {
                return (Object) Double.parseDouble(value);
        } else if (dataType.equals(DATA_BOOLEAN)) {
                return (Object) Boolean.parseBoolean(value);
        } else if (dataType.equals(DATA_STRING)) {
                return (Object) value;
        } else if (dataType.equals(DATA_DATE)) {
                return (Object) value;
        }else if (dataType.equals(DATA_LONG))
                return (Object) value ;
        else {
                return null;
        }
        }else
                return null ;
    }
```

Section 3 :

JDBC Driver

Implementing JDBC API

We implemented the java.sql interfaces to use the JDBC API to access our DBMS ,

Classes used :

- **NewSqlDriver implements java.sql.Driver**

Methods :

Algorithm acceptURL (String url)
{
Checks if url starts with (jdbc : MagicQueries) and also ends with (.xml) and return true in this condition , else it returns false because the url is not accepted
}

Algorithm connect (String url , Proprties info)
{
This method makes a connection object if the url is accepted , and properties are all right
It throws an sql exception if connection can't be done.
}

- **NewSqlConnection implements java.sql.Connection**

Methods :

Algorithm NewSqlConnection(String url) throws SQLException

- Loads the schema from the given url
- If the schema is invalid , throw new SQLException

Algorithm Statement createStatement(String url)

- Creates new NewSqlStatement giving it this connection as a parameter
- Then return it

Algorithm void close( )

- Makes all object's fields null

- ## NewSqlStatement implements java.sql.Statement

Methods :

Algorithm ResultSet executeQuery (String sql) throws SQLException

- It throws an exception if the statement doesn't return a ResultSet or there is a data base access error
- Returns a ResultSet object from executing the query

Algorithm int executeUpdate (String sql) throws SQLException

- Tries to execute the given query , throws SQLException if the query return a ResultSet or there is a data base access error
- Returns the row count or constant (SUCCESS_NOINFO)

Algorithm int[] executeBatch (String sql) throws SQLException

- Loops on each element of the batch array (sqls) and invokes **executeUpdate()** on it
- Returns array of row counts corresponding to the order of statements in the batch array , or (SUCCESS_NOINFO) , or (EXCUTE_FAILED)

Algorithm boolean execute (String sql) throws SQLException

- This method checks if the sql begins with "SELECT" , calls **executeQuery(sql)** and returns true
- Else (i.e if the sql starts with any other word) it calls **executeUpdate (sql)** and returns false in this case

Algorithm void initializeTheSqlParseArray()

- This method sets the main sequals with its corresponding words
- (Ex) "SELECT" works with "FROM" and "WHERE"

Algorithm String [] parseSql (String sql) throws SQLException

- This method takes the full sql and makes it an array of strings like if the user write the sql in many lines
- If the sql has the command "create" it returns an array of two strings ; the command "create table-name" in the first string and the other string contains columns names with their data types
- The mechanism of this method that it loops at the string and cuts it to many strings

Algorithm void clearBatch ()

- This method checks if the data structure used to add batches , and if It wasn't empty , it makes it empty

Algorithm void close ()

- It makes all objects fields refer to null to close the statement and connection

- ## NewSqlResultset implements ResultSet:

Methods :

Algorithm: int getColumnIndex(String columnLabel):

-this method return the index of the passed column Label.

Algorithm: Boolean absolute(int row):

-this method update the pointer which point to the index which equal the row.

-this method return true when we succeed in updating the row and returns False when it can't update the pointer.

Algorithm: void afterLast():

-this method update the pointer to the index which is after the last index.

Algorithm: Boolean getBoolean(int columnIndex):

-This method get the Boolean value from the column which his index equal the columnIndex and at the row which the pointer point to it.

-returns the Boolean value of this cell.

Algorithm: Boolean getBoolean(int columnLabel):

-firstly this method get the column index of the columnLabel then it invoke the (Boolean getBoolean(int columnIndex)) and return the value which invoked method returns.

-returns the Boolean value of this cell.

Algorithm: statement getStatement():

-this method returns the statement which produce this resultSet.

- ## NewSqlResultSetMetaDate implements ResultSetMetaData:

Methods :

Algorithm: int getColumnCount():

-get the number of columns in the result set.

Algorithm: String getColumnName(int column):

-return the name of the current column index.

Algorithm: int getColumnType(int column):

-return the integer number which refers to the data type of the column which index is equal to column.

Algorithm: String getColumnType(int column):

-return the name of the data type of the column which index is equal to column.

Algorithm: int isNullable(int column):

-return the integer number refers to whether this column accept the null value in this column or not.

Logging

Logging is used for keeping some information about the process of the application , we used Log4j package which is provided by apache , there is a file with extension (. properties) to hold appenders and layouts of the logger object . In this program log are made in file using file appender and in a simple format using pattern layout . the log file should contains information about execution and errors which may appear . It also contains information about opening connection and closing it later . All these events are written their specific date.

Data Structures Used:

- **Arrays**
- **Array Lists**
- **Trees (XML Dom Parser)**

Section 4:

Team Members Contributions

	XML DBMS	Query Processor		GUI	Driver	Logger	Report	Testing
		Validation	Parsing					
Asser Mohamed Hassan			√		√	√	√	
Mahmoud Nabil El-Maghraby	√			√		√	√	
Ahmed Hesham Fathy		√		√			√	√
Mohamed Osama Azmy					√		√	√
Moataz Gamal El-Shaer	√				√		√	
Mohamed Ahmed Elsayed					√	√	√	