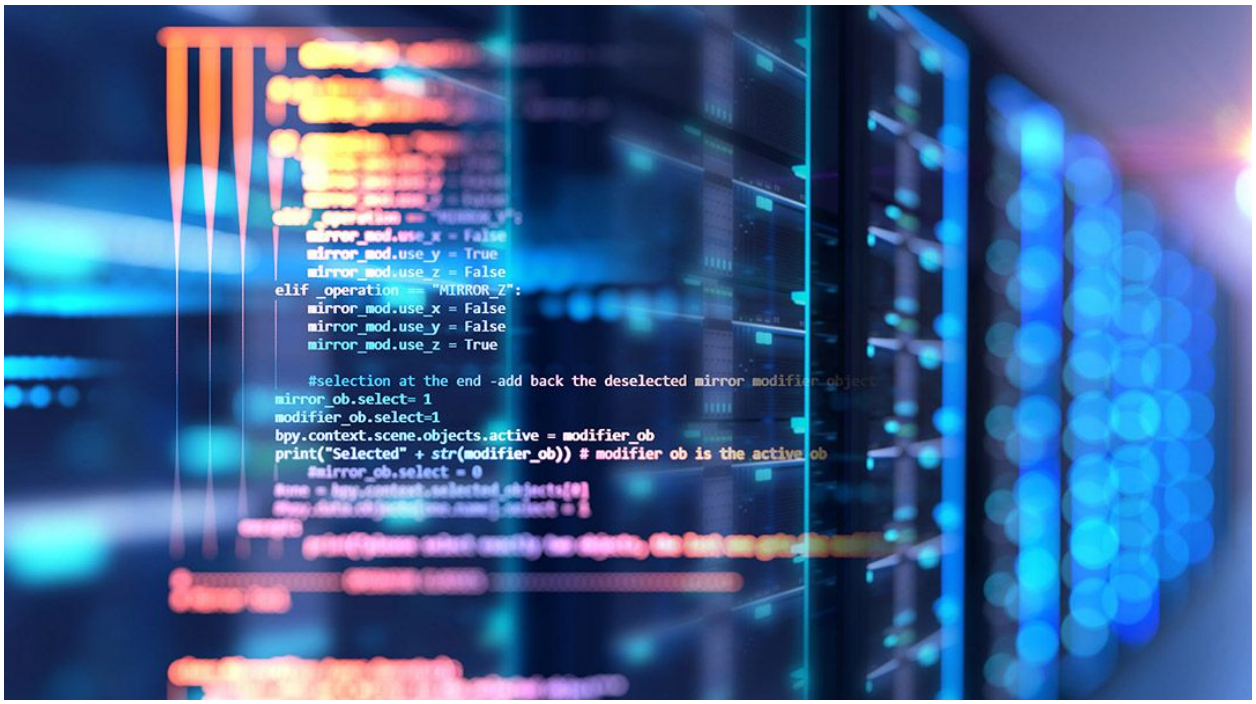


SIMPLE DBMS

29.11.2019

Ahmed Ali , Ahmed Ayman , Ahmed Kamal , Mahmoud Kamal *
Faculty Of Engineering, Alexandria University.

* names are ordered in a descendent order.

Overview

A database management system (DBMS) is system software for creating and managing databases. A DBMS makes it possible for end users to create, read, update and delete data in a database. The DBMS essentially serves as an interface between the database and end users or application programs, ensuring that data is consistently organized and remains easily accessible.

Goals

1. Get familiar with the usage of storing data using a programming language like sql but in some lower level which simulates SQL_LITE.
2. Get friendly with creating tables to store your important data on a machine using only some " Enlish " like code.
3. This program is intended for users of different degrees of knowledge and experience with manipulated data.

USER MANUAL

This program is a sql database allowing next features

1- Create DataBase

By using query

create database " database name "

then the database will stored in file DataBasesDirectory

If the database is already exists then the program will use it otherwise it will create it

Or by using

create database " database name " drop if not exist

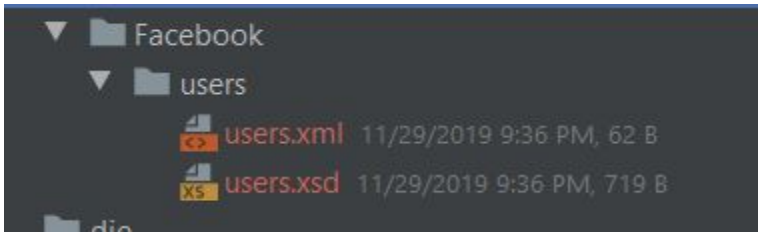
If the database is already exists then the program will delete it , then create it , otherwise it will create it .

2 - create table

By using query

Create table table_name ("name1" type1, "name2" type2, …. " name-n type-n "

```
create table users (  name varchar , age int , email varchar , friends int , marry bool )
```



Then file xsd will be created with names and types of column

```
<?xml version = "1.0"?>
   <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
   <xs:element name = 'users'>
         <xs:complexType>
              <xs:sequence>
                  <xs:element name = 'record' type = 'record'   minOccurs='0' maxOccurs = 'unbounded' />
              </xs:sequence>
         </xs:complexType>
   </xs:element>

<xs:complexType name = 'record'>
      <xs:sequence>
        <xs:element name ="name" type ="xs:string"/>
        <xs:element name ="age" type ="xs:int"/>
        <xs:element name ="email" type ="xs:string"/>
        <xs:element name ="friends" type ="xs:int"/>
        <xs:element name ="marry" type ="xs:boolean"/>
      </xs:sequence>
 </xs:complexType>
</xs:schema>
```

There are 3 allowed types :

1 - varchar (string)

2- int (Integer “Numbers”)

3 - bool (Boolean)

3 - insert into table

By using query

Insert into “ table name “ (c1 , c2 ,…. , cn) Values (V1, V2, …. Vn)

```
insert into users ( name , age , email , friends , marry ) values ( "ahmed" , 20 , "ahmedalzayady295@gmail.com" , 453, false )
insert into users ( name , age , email , friends , marry ) values ( "kamal" , 22 , "mahmoudkamal@gmail.com" , 600, true )
insert into users ( name , age , email , friends , marry ) values ( "zaharan" , 22 , "zahran@gmail.com" , 650, true )
insert into users ( name , age , email , friends , marry ) values ( "ahmedkamal" , 26 , "ahmedkamal@gmail.com" , 655, true )
```

Then xml file be updated :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><users>
    <record><name>"ahmed"</name><age>20</age><email>"ahmedalzayady295@gmail.com"</email><friends>453</friends><marry>fa
    <record><name>"kamal"</name><age>22</age><email>"mahmoudkamal@gmail.com"</email><friends>600</friends><marry>true</
    <record><name>"zaharan"</name><age>22</age><email>"zahran@gmail.com"</email><friends>650</friends><marry>true</marr
    <record><name>"ahmedkamal"</name><age>26</age><email>"ahmedkamal@gmail.com"</email><friends>655</friends><marry>tru
</users>
```

* you can insert your data in a changeable order but by the types and cols

4 - drop table

By using query

Drop table “ table-name “

- Then the table will be deleted

```
create table f (a int )
drop table f
```

- If table doesn’t exist an erroe message will appear

```
table name not exist!
```

5 - Delete from table

By using query

delete from table_name where condition

* this query will delete some specific rows in the chosen table whose data agreed with the condition. If you want to update the hole table just don't write " where + condition " like :

delete from table_name

```
create database facebook
select * from users
"ahmed"|20|"ahmedalzayady295@gmail.com"|453|false
"kamal"|22|"mahmoudkamal@gmail.com"|600|true
"zaharan"|22|"zahran@gmail.com"|650|true
"ahmedkamal"|26|"ahmedkamal@gmail.com"|655|true
delete from users where not age <25 and name="ahmedkamal"
select * from users
"ahmed"|20|"ahmedalzayady295@gmail.com"|453|false
"kamal"|22|"mahmoudkamal@gmail.com"|600|true
"zaharan"|22|"zahran@gmail.com"|650|true
delete from users where not marry
select * from users
"kamal"|22|"mahmoudkamal@gmail.com"|600|true
"zaharan"|22|"zahran@gmail.com"|650|true
```

6 - Drop DataBase

By using query

drop database " database name "

* this query will drop your delete if exists and prints an error message O.W

```
create database kamal
create table temp ( name varchar )
drop database kamal
drop database kamal
Data base doesn't exist
```

7 - Update data into table

By using query

update table_name set " cols_name = value " where condition

* this query will update some specific rows in the chosen table whose data agreed with the condition. If you want to update the hole table just don't write " where + condition " like :

update table_name set " cols_name = value "

```
update users set marry = false where marry
select * from users
"kamal"|22|"mahmoudkamal@gmail.com"|600|false
"zaharan"|22|"zahran@gmail.com"|650|false
```

8- select from table

By using query

select c1, c2 ..... cn from table_name where condition

* this query will select some specific rows in the chosen table whose data agreed with the condition. If you want to select the hole table just don't write " where + condition " like :

select * from table_name

* If you want to select all rows of of specific columns of the just write "cols names" and don't write " where + condition " like :

select c1, c2 ..... cn from table_name

```
select name , email from users
"kamal"|"mahmoudkamal@gmail.com"
"zaharan"|"zahran@gmail.com"
select name , email from users where name="kamal" and age = 22 and marry
"kamal"|"mahmoudkamal@gmail.com"
```

9 -schema

By using query

.schema

- To get name of column and types of them for current database

```
.schema
CREATE TABLE users VALUES(name varchar ,age int ,email varchar ,friends int ,marry );
create table tenp ( v1 varchar ,  v2 int )
.schema
CREATE TABLE users VALUES(name varchar ,age int ,email varchar ,friends int ,marry );
CREATE TABLE tenp VALUES(v1 varchar ,v2 int );
```

10 - Exit command

By using query

.quit

```
.quit
thanks for enjoying our DBMS
```

11 - Used OOP Design patterns :

1 - facade pattern

2 - singleton pattern

3- factory pattern

Allowed operations:

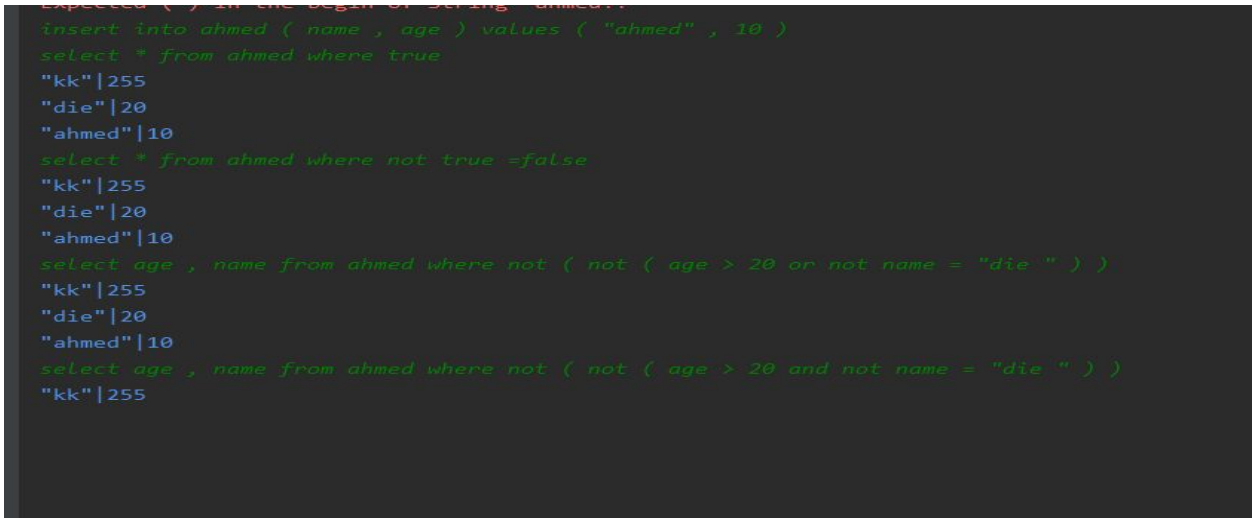
1 - > < =

2- * + - \

3 - and not or

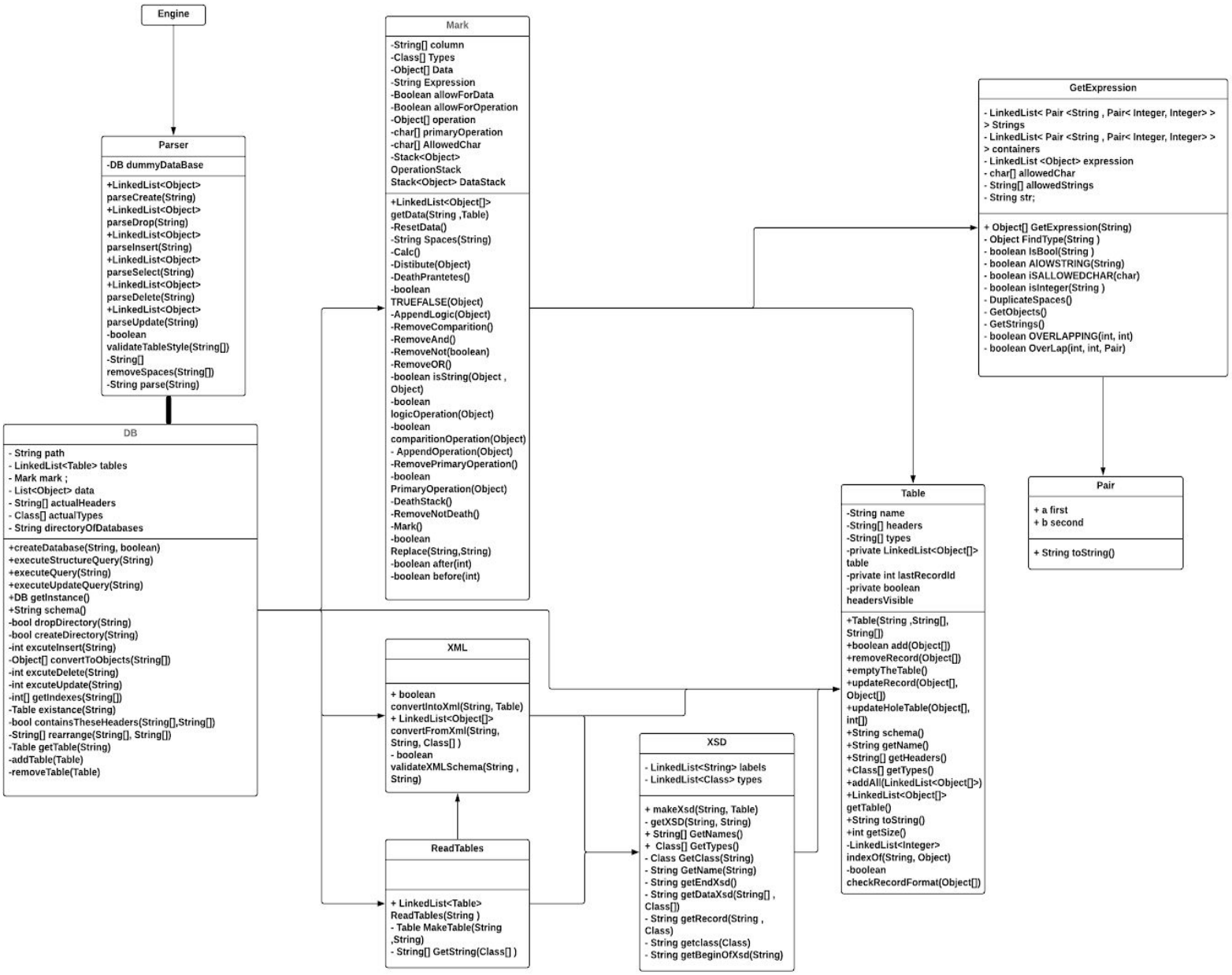
4- ( )

Examples :



```
insert into ahmed ( name , age ) values ( "ahmed" , 10 )
select * from ahmed where true
"kk"|255
"die"|20
"ahmed"|10
select * from ahmed where not true =false
"kk"|255
"die"|20
"ahmed"|10
select age , name from ahmed where not ( not ( age > 20 or not name = "die " ) )
"kk"|255
"die"|20
"ahmed"|10
select age , name from ahmed where not ( not ( age > 20 and not name = "die " ) )
"kk"|255
```

The UML Diagram :


Engine
Parser
-DB dummyDataBase
+LinkedList<Object> parseCreate(String)
+LinkedList<Object> parseDrop(String)
+LinkedList<Object> parseInsert(String)
+LinkedList<Object> parseSelect(String)
+LinkedList<Object> parseDelete(String)
+LinkedList<Object> parseUpdate(String)
-boolean validateTableStyle(String[])
-String[] removeSpaces(String[])
-String parse(String)
DB
- String path
- LinkedList<Table> tables
- Mark mark ;
- List<Object> data
- String[] actualHeaders
- Class[] actualTypes
- String directoryOfDatabases
+createDatabase(String, boolean)
+executeStructureQuery(String)
+executeQuery(String)
+executeUpdateQuery(String)
+DB getInstance()
+String schema()
-bool dropDirectory(String)
-bool createDirectory(String)
-int excuteInsert(String)
-Object[] convertToObjects(String[])
-int excuteDelete(String)
-int excuteUpdate(String)
-int[] getIndexes(String[])
-Table existance(String)
-bool containsTheseHeaders(String[],String[])
-String[] rearrange(String[], String[])
-Table getTable(String)
-addTable(Table)
-removeTable(Table)
Mark
-String[] column
-Class[] Types
-Object[] Data
-String Expression
-Boolean allowForData
-Boolean allowForOperation
-Object[] operation
-char[] primaryOperation
-char[] AllowedChar
-Stack<Object> OperationStack
Stack<Object> DataStack
+LinkedList<Object[]> getData(String ,Table)
-ResetData()
-String Spaces(String)
-Calc()
-Distibute(Object)
-DeathPrantetes()
-boolean TRUEFALSE(Object)
-AppendLogic(Object)
-RemoveComparition()
-RemoveAnd()
-RemoveNot(boolean)
-RemoveOR()
-boolean isString(Object , Object)
-boolean logicOperation(Object)
-boolean comparitionOperation(Object)
- AppendOperation(Object)
-RemovePrimaryOperation()
-boolean PrimaryOperation(Object)
-DeathStack()
-RemoveNotDeath()
-Mark()
-boolean Replace(String,String)
-boolean after(int)
-boolean before(int)
XML
+ boolean convertIntoXml(String, Table)
+ LinkedList<Object[]> convertFromXml(String, String, Class[])
- boolean validateXMLSchema(String , String)
ReadTables
+ LinkedList<Table> ReadTables(String)
- Table MakeTable(String ,String)
- String[] GetString(Class[])
XSD
- LinkedList<String> labels
- LinkedList<Class> types
+ makeXsd(String, Table)
- getXSD(String, String)
+ String[] GetNames()
+ Class[] GetTypes()
- Class GetClass(String)
- String GetName(String)
- String getEndXsd()
- String getDataXsd(String[] , Class[])
- String getRecord(String , Class)
- String getclass(Class)
- String getBeginOfXsd(String)
GetExpression
- LinkedList< Pair <String , Pair< Integer, Integer> > > Strings
- LinkedList< Pair <String , Pair< Integer, Integer> > > containers
- LinkedList <Object> expression
- char[] allowedChar
- String[] allowedStrings
- String str;
+ Object[] GetExpression(String)
- Object FindType(String)
- boolean IsBool(String)
- boolean AlOWSTRING(String)
- boolean iSALLOWEDCHAR(char)
- boolean isInteger(String)
- DuplicateSpaces()
- GetObjects()
- GetStrings()
- boolean OVERLAPPING(int, int)
- boolean OverLap(int, int, Pair)
Table
-String name
-String[] headers
-String[] types
-private LinkedList<Object[]> table
-private int lastRecordId
-private boolean headersVisible
+Table(String ,String[], String[])
+boolean add(Object[])
+removeRecord(Object[])
+emptyTheTable()
+updateRecord(Object[], Object[])
+updateHoleTable(Object[], int[])
+String schema()
+String getName()
+String[] getHeaders()
+Class[] getTypes()
+addAll(LinkedList<Object[]>)
+LinkedList<Object[]> getTable()
+String toString()
+int getSize()
-LinkedList<Integer> indexOf(String, Object)
-boolean checkRecordFormat(Object[])
Pair
+ a first
+ b second
+ String toString()

Run :