Alexandria University
Faculty of Engineering
Computer and Systems Engineering Dept.

CS221: Object Oriented Programming

SIMPLE DBMS

DOCTOR

Dr. Khaled Nagy

TAS

Eng. Shehab K. El-semmany

Eng. Pa*ula B. Bassily*

STUDENTS

Abanoub Ashraf Ezzat Zaki (01)
Arsany Atef (08)
Kirellos Malak Habib (33)
Michael Said Beshara (36)

Problem Statement:

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. Extensible Markup Language (XML) (encoding: ISO-8859-1) is a set of rules for encoding documents in machine readable form.

Design Patterns used:

- ✓ **Singleton pattern:**
  - **OurSql (Concrete class implementing database interface).**
  - **GeneralParser.**
  - **JavaScriptEngine.**
  - **FileManager.**
  - **ConditionParser.**
  - **RelationalSolver.**
  - **ConditionalSolver.**
  - **dataChecker.**
- ✓ **Object Pool pattern:**
  - **Table .**
- ✓ **InterFace Pattern:**
  - **IQuery.**
  - **IQueryFactory.**
  - **DataBase.**
- ✓ **Abstract Class Pattern:**
  - **OurQuery.**
  - **OurQueryFactory.**
- ✓ **Delegation Pattern:**
  - **RelationalCondition**
  - **LogicalCondition.**
- ✓ **Filter Pattern:**
  - **GeneralParser.**
- ✓ **Abstract Factory:**
  - **OurQueryFactory.**
  - **SelectQueryFactory.**

- **StructureQueryFactory.**
- **UpdateQueryFactory.**

✓ **Products AbstractFactory Pattern:**

- **CreateDatabase.**
- **CreateTable.**
- **DropTable.**
- **DropDatabase.**
- **DeleteFrom.**
- **Update.**
- **SelectFrom.**
- **InsertInto.**

✓ **Thread Pool Pattern:**

- **In handling buffering data on same table and saving it as cache data to write it finally when the database or the current table changes or the program running exits.**

✓ **Command Pattern:**

- **In Execution of all the queries.**

✓ **ChainOfResponsibility Pattern:**

- **In handling errors and Exceptions.**

Design Decision:

- ✓ There is a GeneralParser that filters the string queries calling appropriate methods from OurSql class which calls QueryFactory class to create the appropriate query object.
- ✓ After that using the command pattern call on execute method in each query object to execute its unique functionality.
- ✓ Regex is used in parsing string queries and for parsing complex logical and relational conditions.
- ✓ The parser should validate the SQL statements and reject bad ones.
- ✓ There is a FileManager class that handles reading & writing data files.
- ✓ There is a dataChecker class that handles putting data of the same type of the column (Unless the column type is Varchar) .
- ✓ Java Script Engine is used in handling conditions.
- ✓ There are two types of conditions , relational & logical conditions.
- ✓ Every logical condition contains a relational one at least.
- ✓ In case of insertion of a non complete row the rest of the non inserted values are saved as null objects.
- ✓ If a null object is found in a relational condition it will always give true.
- ✓ The User can enter logical conditions as many as he can.
- ✓ We provide selecting more than one column in the table applying conditions in the selection.

- ✓ Caching is used to collect data of the same table and write it finally when the database or the current table changes or the program running exits using a thread that detects the time when the program running ends.
- ✓ There are two types of termination of program running , The first type is when the user terminates the program intentionally, The second is when any problem occurs and consequently the program terminates (eg. When the battery fails).

User Manual:

- This is a console based application since its available only for developers helping them to manage their data bases.
- The user only has to enter his valid queries consequently and then he sees the results in the console.
- We only support :
  - ✓ Creation of databases and tables.
  - ✓ Dropping of databases and tables.
  - ✓ Selection of multiple columns or the whole table applying complex logical condition on the selection.
  - ✓ Insertion of data into tables.
  - ✓ Updating multiple columns or the whole table applying complex logical condition on updating.
  - ✓ Deleting data from tables applying complex logical condition on deleting.
- To exit the program you just need to enter the word "exit" so the program terminates and the data is updated then.
- It's preferable to put the string values between (' ' / " ").
- If an Exception occurs the error is printed in the console with red color. (eg. `There is no database selected`).

Sample Runs:

```
Enter Queries :
create database 555
You have made changes to the databases.
create table 5 (name varchar, id int, c varchar)
You have made changes to the databases.
insert into 5 values ('bebo', 21, maths)
You have made changes to 1 rows.
insert into 5 values ('koko', 22, maths)
You have made changes to 1 rows.
insert into 5 values ('kiro malak', 24, p)
You have made changes to 1 rows.
insert into 5 values (bebo, 23, my)
You have made changes to 1 rows.
insert into 5 values ("mickey saeed", 25, mn)
You have made changes to 1 rows.
insert into 5 values ("bebo", 21, maths)
You have made changes to 1 rows.
insert into 5 (name) values (Arsany Atef);
You have made changes to 1 rows.
```

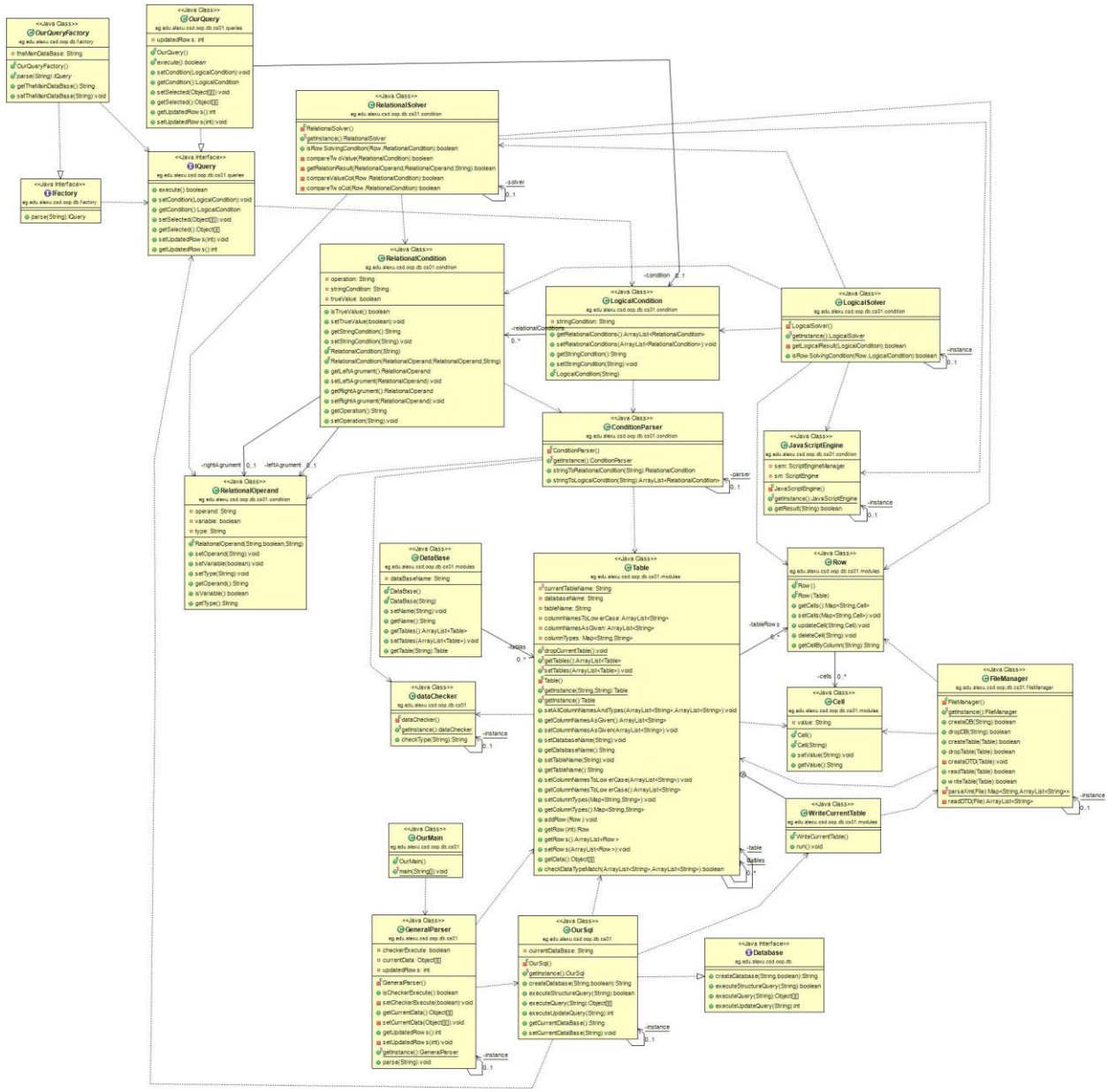
```
Enter Queries :
create database 555
You have made changes to the databases.
select * from 5
name id c
bebo 21 maths
koko 22 maths
kiro malak 24 p
bebo 23 my
mickey saeed 25 mn
bebo 21 maths
Arsany Atef null null
mico saeed null null
mico null null
A A null null
```

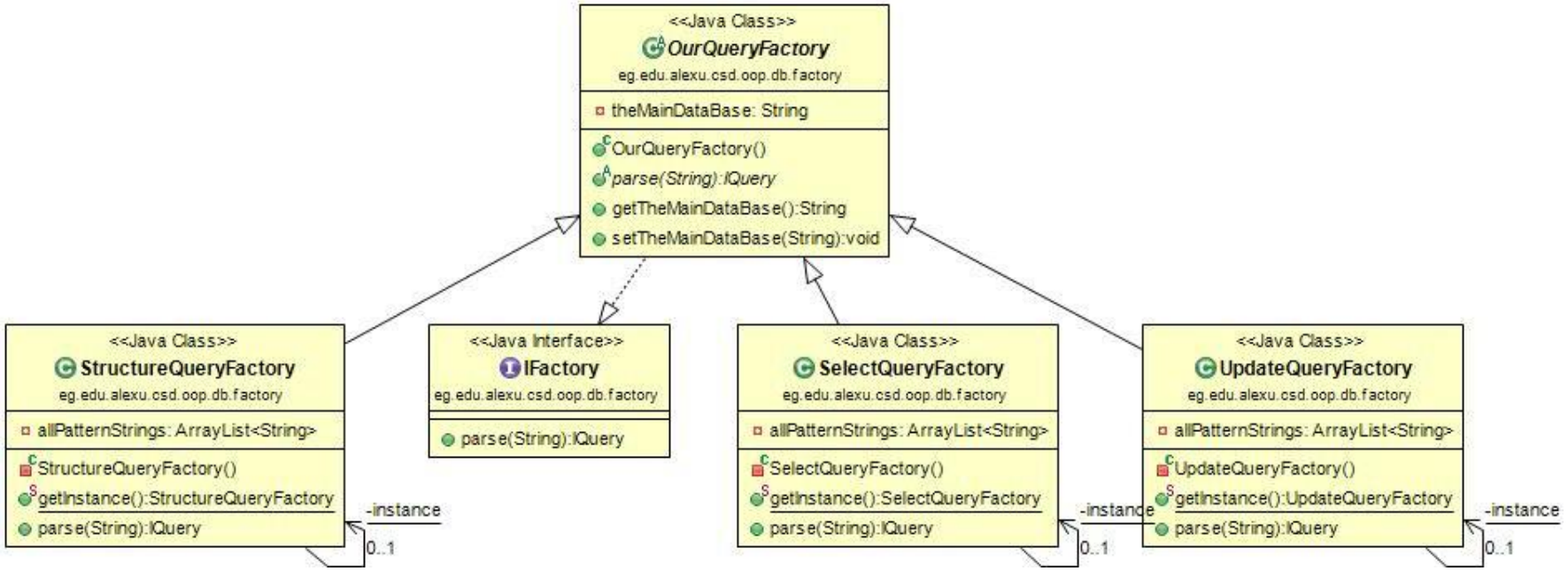
```
select name, id from 5
name id c
bebo 21
koko 22
kiro malak 24
bebo 23
mickey saeed 25
bebo 21
Arsany Atef null
mico saeed null
mico null
A A null
```

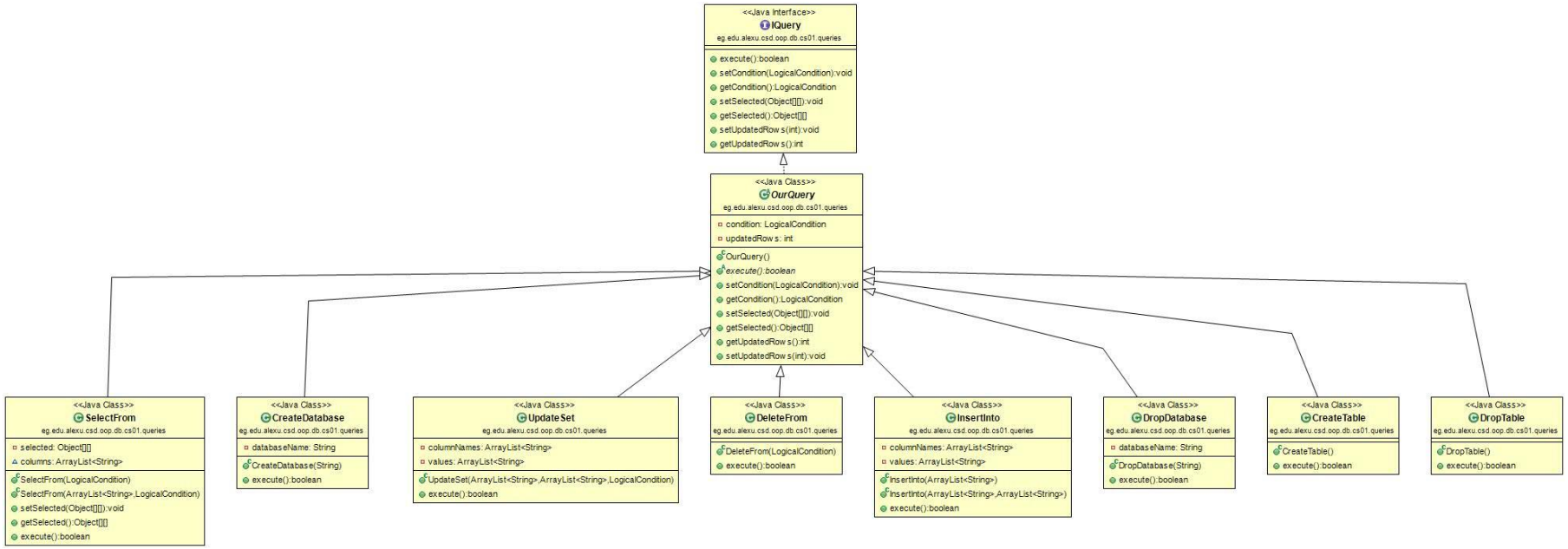
```
select name, id from 5 where ((name = "bebo" and not c = mn) or (id >= 25));
name id c
bebo 21
bebo 23
mickey saeed 25
bebo 21
select * from 5
name id c
bebo 21 maths
koko 22 maths
kiro malak 24 p
bebo 23 my
mickey saeed 25 mn
bebo 21 maths
delete from 5 where ((name = "bebo" and not c = mn) or (id >= 25));
You have made changes to 4 rows.
select * from 5
name id c
koko 22 maths
kiro malak 24 p
```

```
Enter Queries :
create database 555
You have made changes to the databases.
select * from 5
name id c
koko 22 maths
kiro malak 24 p
bebo 5 a
bony 5 b
sonson 9 j
update 5 set c = maths where not ((name != "bebo" and not c = mn) or (id >= 25))
You have made changes to 1 rows.
select * from 5
name id c
koko 22 maths
kiro malak 24 p
bebo 5 maths
bony 5 b
sonson 9 j
drop table 5;
You have made changes to the databases.
select * from 5
Table not found.
drop database 555
You have made changes to the databases.
```

UML Diagram:


OurQueryFactory
OurQuery
IQuery
Factory
RelationalSolver
RelationalCondition
LogicalCondition
LogicalSolver
ConditionParser
JavaScriptEngine
RelationalOperand
DataBase
Table
Row
Cell
FileManager
dataChecker
WriteCurrentTable
OurMain
GeneralParser
OurSql
Database


<<Java Class>>
OurQueryFactory
eg.edu.alexu.csd.oop.db.factory
theMainDataBase: String
OurQueryFactory()
parse(String):IQuery
getTheMainDataBase():String
setTheMainDataBase(String):void
<<Java Class>>
StructureQueryFactory
eg.edu.alexu.csd.oop.db.factory
allPatternStrings: ArrayList<String>
StructureQueryFactory()
getInstance():StructureQueryFactory
parse(String):IQuery
-instance
0..1
<<Java Interface>>
IFactory
eg.edu.alexu.csd.oop.db.factory
parse(String):IQuery
<<Java Class>>
SelectQueryFactory
eg.edu.alexu.csd.oop.db.factory
allPatternStrings: ArrayList<String>
SelectQueryFactory()
getInstance():SelectQueryFactory
parse(String):IQuery
-instance
0..1
<<Java Class>>
UpdateQueryFactory
eg.edu.alexu.csd.oop.db.factory
allPatternStrings: ArrayList<String>
UpdateQueryFactory()
getInstance():UpdateQueryFactory
parse(String):IQuery
-instance
0..1


<<Java Interface>>
IQuery
eg.edu.alexu.csd.oop.db.cs01.queries
execute():boolean
setCondition(LogicalCondition):void
getCondition():LogicalCondition
setSelected(Object[][]):void
getSelected():Object[][]
setUpdatedRows(int):void
getUpdatedRows():int
<<Java Class>>
OurQuery
eg.edu.alexu.csd.oop.db.cs01.queries
condition: LogicalCondition
updatedRows: int
OurQuery()
execute():boolean
setCondition(LogicalCondition):void
getCondition():LogicalCondition
setSelected(Object[][]):void
getSelected():Object[][]
getUpdatedRows():int
setUpdatedRows(int):void
<<Java Class>>
SelectFrom
eg.edu.alexu.csd.oop.db.cs01.queries
selected: Object[][]
columns: ArrayList<String>
SelectFrom(LogicalCondition)
SelectFrom(ArrayList<String>,LogicalCondition)
setSelected(Object[][]):void
getSelected():Object[][]
execute():boolean
<<Java Class>>
CreateDatabase
eg.edu.alexu.csd.oop.db.cs01.queries
databaseName: String
CreateDatabase(String)
execute():boolean
<<Java Class>>
UpdateSet
eg.edu.alexu.csd.oop.db.cs01.queries
columnNames: ArrayList<String>
values: ArrayList<String>
UpdateSet(ArrayList<String>,ArrayList<String>,LogicalCondition)
execute():boolean
<<Java Class>>
DeleteFrom
eg.edu.alexu.csd.oop.db.cs01.queries
DeleteFrom(LogicalCondition)
execute():boolean
<<Java Class>>
InsertInto
eg.edu.alexu.csd.oop.db.cs01.queries
columnNames: ArrayList<String>
values: ArrayList<String>
InsertInto(ArrayList<String>)
InsertInto(ArrayList<String>,ArrayList<String>)
execute():boolean
<<Java Class>>
DropDatabase
eg.edu.alexu.csd.oop.db.cs01.queries
databaseName: String
DropDatabase(String)
execute():boolean
<<Java Class>>
CreateTable
eg.edu.alexu.csd.oop.db.cs01.queries
CreateTable()
execute():boolean
<<Java Class>>
DropTable
eg.edu.alexu.csd.oop.db.cs01.queries
DropTable()
execute():boolean