Final Project

I have designed a simple query/declarative database language that is not focused on relational databases but rather a generic way of storing and retrieving any type of data. The language does not support variables but rather just displays a message such as "X data inserted into key Y on database Z", "Retrieved: X from database Y", and so on. The syntax and interpreter are written in Java.

Download source code: https://github.com/ArtyVazquez/QueryDBLangDesign

Syntax:

This language has the following query operations:

QUERY :=

CREATE<DatabaseIdent> |

INSERT <DatabaseIdent><KeyIdent><Data> |

RETRIEVE<DatabaseIdent><KeyIdent> |

UPDATE<DatabaseIdent><KeyIdent><Data>

REMOVE<DatabaseIdent><KeyIdent> |

COMBINE<DatabaseIdent><KeyIdent><KeyIdent><KeyIdent> |

DELETE<DatabaseIdent>

DatabaseIdent and **KeyIdent** are types which identify either the name of the database or the key of the data stored.

Data is a type which is allowed to be stored in the database. Data types are IntVal, StrVal, DoubleVal, and ArrayVal which takes in as many data types values.

Syntax code:

```java
package com.company;

public abstract class Query {

    public static class CREATE extends Query {
        public final Ident DBname;

        CREATE(Ident DBname) {
            this.DBname = DBname;
        }
    }

    public static class INSERT extends Query {
        public final Ident DBname;
        public final Ident key;
        public final Data data;

        public INSERT(Ident DBname, Ident key, Data data) {
            this.DBname = DBname;
            this.key = key;
            this.data = data;
        }
    }

    public static class RETRIEVE extends Query {
        public final Ident DBname;
        public final Ident key;

        public RETRIEVE(Ident DBname, Ident key) {
            this.DBname = DBname;
            this.key = key;
        }
    }

    public static class UPDATE extends Query {
        public final Ident DBname;
        public final Ident key;
        public final Data updatedData;

        public UPDATE(Ident DBname, Ident key, Data updatedData) {
            this.DBname = DBname;
            this.key = key;
            this.updatedData = updatedData;
        }
    }
```

```
46
47      public static class REMOVE extends Query {
48          public final Ident DBname;
49          public final Ident key;
50
51          public REMOVE(Ident DBname, Ident key) {
52              this.DBname = DBname;
53              this.key = key;
54          }
55      }
56
57      public static class DELETE extends Query {
58          public final Ident DBname;
59
60          public DELETE(Ident DBname) {
61              this.DBname = DBname;
62          }
63      }
64
65      public static class COMBINE extends Query {
66          public final Ident DBname;
67          public final Ident newKey;
68          public final Ident key1;
69          public final Ident key2;
70
71          public COMBINE(Ident DBname, Ident newKey, Ident key1, Ident key2) {
72              this.DBname = DBname;
73              this.newKey = newKey;
74              this.key1 = key1;
75              this.key2 = key2;
76          }
77      }
78  }
```

Source code:
https://github.com/ArtyVazquez/QueryDBLangDesign/blob/master/src/com/company/Query.java

Data type code:

```java
package com.company;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.concurrent.atomic.AtomicReference;

public abstract class Data {
    // Types of data that can be stores in the database

    public static class IntVal extends Data {
        public final int val;

        public IntVal(int val) {
            this.val = val;
        }

        @Override
        public String toString() {
            return "IntVal{" + val + '}';
        }
    }

    public static class StrVal extends Data {
        public final String val;

        public StrVal(String val) {
            this.val = val;
        }

        @Override
        public String toString() {
            return "StrVal{"  + val + '}';
        }
    }


    public static class DoubleVal extends Data {
        public final Double val;

        public DoubleVal(Double val) {
            this.val = val;
        }

        @Override
        public String toString() {
            return "DoubleVal{" + val + '}';
        }
    }
```

```java

    public static class ArrayVal extends Data {
        public final ArrayList<Data> val;

        public ArrayVal(Data... val) {
            this.val = new ArrayList<>(Arrays.asList(val));
        }


        @Override
        public String toString() {
            AtomicReference<String> str = new AtomicReference<>("ArrayVal{");
            val.forEach(i-> toStringArrHelper(i, str));
            return str + "}";
        }

    }

    // Case that an ArrayVal has many ArrayVal
    protected void toStringArrHelper(Data val, AtomicReference<String> str) {
        switch (val.getClass().getSimpleName()) {
            case "FloatVal":
            case "StrVal":
            case "IntVal": {
                str.set(str + val.toString());
                return;
            }
            case "ArrayVal": {
                ArrayVal a = (ArrayVal) val;
                a.val.forEach(i->toStringArrHelper(i, str));
            }
        }
    }
}
```

Source code : https://github.com/ArtyVazquez/QueryDBLangDesign/blob/master/src/com/company/Data.java

Typecheker

Constructors for the syntax only allow to build type correct programs thus the typerchecker was discarded.

Interpreter

The way that I implemented the interpreter is by allowing multiple queries to be interpreted. I am using a **Map<String, Map<String, Data>>** as a model for the database. **CREATE** is only a valid query when there is not a DatabaseIdent named with the name specified. This query will create a Database with the specified DatabaseIdent. **INSERT** is only valid when the DatabaseIdent specified is in the Map storing all the databases and the KeyIdent specified does not belong to DatabaseIdent. This query will insert into a database with some given DatabaseIdent some Data which is mapped to some KeyIdent. **RETRIEVE** and **UPDATE** are only valid when the DatabaseIdent specified and the KeyIdent specified both exist in the database model and **UPDATE** should also have data of type Data to be valid. **RETRIEVE** will get the Data mapped to some KeyIdent stored in the specified DatabaseIdent and print it on the console. **UPDATE** will update Data mapped to some KeyIdent in the specified DatabaseIdent. **REMOVE** is only valid when the DatabaseIdent specified is in the Map storing all the databases and the KeyIdent specified belongs to DatabaseIdent. This query will remove the KeyIdent and its mapping to some Data. **COMBINE** is only valid when the DatabaseIdent specified is in the Map storing all the databases and the first KeyIdent "newKey" does not belong to the DatabaseIdent specified but the other two KeyIdent belong to the DatabaseIdent specified. This query will take the data from two KeyIdent which belong to the same DatabaseIdent and store the data from the two KeyIdent to some new KeyIdent . **DELETE** is only a valid query when there is a DatabaseIdent named with the name specified. This query will delete a DatabaseIdent from the database.

Interpreter code

```java
package com.company;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

public class Interpreter {
    private static final Map<String, Map<String, Data>> databases = new HashMap<>();

    public void query(Query... queries) {
        Arrays.stream(queries).forEach(this::query);
    }

    public void query(Query q) {
        switch (q.getClass().getSimpleName()) {
            case "CREATE": {
                Query.CREATE c = (Query.CREATE) q;

                if (checkDBExists(c.DBname.name) != null)
                    throw new Error(c.DBname.name + " already exists");

                databases.put(c.DBname.name, new HashMap<>());
                System.out.println(c.DBname.name + " created.");
                return;
            }
            case "INSERT": {
                Query.INSERT i = (Query.INSERT) q;

                if (checkDBAndKeyExists(i.DBname.name, i.key.name) != null)
                    throw new Error("Invalid DB or Key does not exists");

                databases.get(i.DBname.name).put( i.key.name, i.data);
                System.out.println(i.data + " inserted into key {" + i.key.name + "} on database {"
                        + i.DBname.name + "}");
                return;
            }
            case "RETRIEVE": {
                Query.RETRIEVE g = (Query.RETRIEVE) q;

                if (checkDBAndKeyExists(g.DBname.name, g.key.name) == null)
                    throw new Error("Invalid DB or Key does not exists");

                System.out.println("Retrieved: " + databases.get(g.DBname.name).get(g.key.name) +
                        " from database " + g.DBname.name);
                return;
            }
```

```
            case "UPDATE": {
                Query.UPDATE u = (Query.UPDATE) q;

                if (checkDBAndKeyExists(u.DBname.name, u.key.name) == null)
                    throw new Error("Invalid DB or Key does not exists");

                databases.get(u.DBname.name).put(u.key.name, u.updatedData);
                System.out.println("Key " + u.key.name + " updated data to " + u.updatedData);
                return;
            }
            case "REMOVE": {
                Query.REMOVE r = (Query.REMOVE) q;

                if (checkDBAndKeyExists(r.DBname.name, r.key.name) == null)
                    throw new Error("Invalid DB or Key does not exists");

                databases.get(r.DBname.name).remove(r.key.name);
                System.out.println("Removed " + r.key.name + " from database " + r.DBname.name);
                return;
            }
            case "DELETE": {
                Query.DELETE d = (Query.DELETE) q;

                if (checkDBExists(d.DBname.name) == null)
                    throw new Error("DB does not exist");

                databases.remove(d.DBname.name);

                System.out.println("Deleted database " + d.DBname.name);

                return;
            }
            case "COMBINE": {
                Query.COMBINE c = (Query.COMBINE) q;

                if (checkDBAndKeyExists(c.DBname.name, c.key1.name) == null ||
                        checkDBAndKeyExists(c.DBname.name, c.key2.name) == null)
                    throw new Error("Invalid DB or Key does not exists");

                Data d1 = databases.get(c.DBname.name).get(c.key1.name);
                Data d2 = databases.get(c.DBname.name).get(c.key2.name);
                databases.get(c.DBname.name).put(c.newKey.name, new Data.ArrayVal(d1, d2));

                System.out.println("Combined data with key " + c.key1.name + " and data with key " + c.key2.name +
                        "  into a new key " + c.newKey.name);
                return;
            }
            default: throw new Error("I don't know the query: " + q.getClass().getSimpleName());
        }
    }
```

Source code: https://github.com/ArtyVazquez/QueryDBLangDesign/blob/master/src/com/company/Interpreter.java

Running Queries

In the RunQueries class there are nine sample programs that run all the queries in the language combined. Note: run a program at a time since the programs all create a database which may or may not create a database that already exists which causes there to be an error thrown "X already exists". The programs and output are:

Program 1:

```
// Program 1 CREATE
i.query(new Query.CREATE(
        new DatabaseIdent( name: "Database CS476")));
```

Output:

```
Database CS476 created.


Process finished with exit code 0
```

Program 2:

```
// Program 2 CREATE, INSERT
i.query(new Query.CREATE(
                new DatabaseIdent( name: "Database CS476")),
        new Query.INSERT(
                new DatabaseIdent( name: "Database CS476"),
                new KeyIdent( name: "Student1"),
                new Data.StrVal("Arturo")));
```

Output:

```
Database CS476 created.

StrVal{Arturo} inserted into key {Student1} on database {Database CS476}
```

Program 3:

```
// Program 3 CREATE, INSERT. RETRIEVE
i.query(new Query.CREATE(
                new DatabaseIdent( name: "Database CS476")),
        new Query.INSERT(
                new DatabaseIdent( name: "Database CS476"),
                new KeyIdent( name: "Student1"),
                new Data.StrVal("Arturo")),
        new Query.RETRIEVE(new DatabaseIdent( name: "Database CS476"),
                new KeyIdent( name: "Student1")));
```

Output:

```
Database CS476 created.

StrVal{Arturo} inserted into key {Student1} on database {Database CS476}

Retrieved: StrVal{Arturo} from database Database CS476
```

Program 4:

```
// Program 4 CREATE, INSERT, UPDATE, RETRIEVE
i.query(new Query.CREATE(
                new DatabaseIdent( name: "Database CS476")),
        new Query.INSERT(
                new DatabaseIdent( name: "Database CS476"),
                new KeyIdent( name: "Student1"),
                new Data.StrVal("Arturo")),
        new Query.UPDATE(new DatabaseIdent( name: "Database CS476"),
                new KeyIdent( name: "Student1"),
                new Data.StrVal("Arturo Vazquez")),
        new Query.RETRIEVE(new DatabaseIdent( name: "Database CS476"),
                new KeyIdent( name: "Student1")));
```

Output:

```
Database CS476 created.
StrVal{Arturo} inserted into key {Student1} on database {Database CS476}
key Student1 updated data to StrVal{Arturo Vazquez}
Retrieved: StrVal{Arturo Vazquez} from database Database CS476
```

Program 5:

```
// Program 5 CREATE, INSERT, RETRIEVE
i.query(new Query.CREATE(
                new DatabaseIdent( name: "Plants")),
        new Query.INSERT(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Fruits"),
                new Data.ArrayVal(
                        new Data.StrVal("Apple"),
                        new Data.StrVal("Pear"),
                        new Data.StrVal("Grape"),
                        new Data.StrVal("Mango"),
                        new Data.StrVal("Cherry"))),
        new Query.INSERT(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Vegetables"),
                new Data.ArrayVal(
                        new Data.StrVal("Tomato"),
                        new Data.StrVal("Cauliflower"),
                        new Data.StrVal("Broccoli"),
                        new Data.StrVal("Zucchini"))),
        new Query.RETRIEVE(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Vegetables")));
```

Output:

```
Plants created.
ArrayVal{StrVal{Apple}StrVal{Pear}StrVal{Grape}StrVal{Mango}StrVal{Cherry}} inserted into key {Fruits} on database {Plants}
ArrayVal{StrVal{Tomato}StrVal{Cauliflower}StrVal{Broccoli}StrVal{Zucchini}} inserted into key {Vegetables} on database {Plants}
Retrieved: ArrayVal{StrVal{Tomato}StrVal{Cauliflower}StrVal{Broccoli}StrVal{Zucchini}} from database Plants
```

Program 6:

```
// Program 6 CREATE, INSERT, RETRIEVE
i.query(new Query.CREATE(
                new DatabaseIdent( name: "Plants")),
        new Query.INSERT(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Fruits"),
                new Data.ArrayVal(
                        new Data.StrVal("Apple"),
                        new Data.StrVal("Pear"),
                        new Data.StrVal("Grape"),
                        new Data.StrVal("Mango"),
                        new Data.StrVal("Cherry"))),
        new Query.INSERT(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Vegetables"),
                new Data.ArrayVal(
                        new Data.StrVal("Tomato"),
                        new Data.StrVal("Cauliflower"),
                        new Data.StrVal("Broccoli"),
                        new Data.StrVal("Zucchini"))),
        new Query.RETRIEVE(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Fruits")));
```

Output:

```
Plants created.
ArrayVal{StrVal{Apple}StrVal{Pear}StrVal{Grape}StrVal{Mango}StrVal{Cherry}} inserted into key {Fruits} on database {Plants}
ArrayVal{StrVal{Tomato}StrVal{Cauliflower}StrVal{Broccoli}StrVal{Zucchini}} inserted into key {Vegetables} on database {Plants}
Retrieved: ArrayVal{StrVal{Apple}StrVal{Pear}StrVal{Grape}StrVal{Mango}StrVal{Cherry}} from database Plants
```

Program 7:

```
// Program 7 CREATE INSERT REMOVE RETRIEVE
// ---> error thrown : Invalid DB or Key does not exist. Since Fruits was removed then we tried to retrieve
i.query(new Query.CREATE(
                new DatabaseIdent( name: "Plants")),
        new Query.INSERT(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Fruits"),
                new Data.ArrayVal(
                        new Data.StrVal("Apple"),
                        new Data.StrVal("Pear"),
                        new Data.StrVal("Grape"),
                        new Data.StrVal("Mango"),
                        new Data.StrVal("Cherry"))),
        new Query.INSERT(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Vegetables"),
                new Data.ArrayVal(
                        new Data.StrVal("Tomato"),
                        new Data.StrVal("Cauliflower"),
                        new Data.StrVal("Broccoli"),
                        new Data.StrVal("Zucchini"))),
        new Query.REMOVE(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Fruits")),
        new Query.RETRIEVE(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Fruits")
        ));
```

Output:

```
Plants created.
ArrayVal{StrVal{Apple}StrVal{Pear}StrVal{Grape}StrVal{Mango}StrVal{Cherry}} inserted into key {Fruits} on database {Plants}
ArrayVal{StrVal{Tomato}StrVal{Cauliflower}StrVal{Broccoli}StrVal{Zucchini}} inserted into key {Vegetables} on database {Plants}
Removed Fruits from database Plants
Exception in thread "main" java.lang.Error Create breakpoint : Invalid DB or Key does not exists
    at com.company.Interpreter.query(Interpreter.java:41) <2 internal lines>
    at com.company.Interpreter.query(Interpreter.java:11)
    at com.company.RunQueries.main(RunQueries.java:95)
```

Program 8:

```
// Program 8 CREATE INSERT COMBINE RETRIEVE
i.query(new Query.CREATE(
                new DatabaseIdent( name: "Plants")),
        new Query.INSERT(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Fruits"),
                new Data.ArrayVal(
                        new Data.StrVal("Apple"),
                        new Data.StrVal("Pear"),
                        new Data.StrVal("Grape"),
                        new Data.StrVal("Mango"),
                        new Data.StrVal("Cherry"))),
        new Query.INSERT(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Vegetables"),
                new Data.ArrayVal(
                        new Data.StrVal("Tomato"),
                        new Data.StrVal("Cauliflower"),
                        new Data.StrVal("Broccoli"),
                        new Data.StrVal("Zucchini"))),
        new Query.COMBINE(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Fruits & Veg"),
                new KeyIdent( name: "Vegetables"),
                new KeyIdent( name: "Fruits")),
        new Query.RETRIEVE(
                new DatabaseIdent( name: "Plants"),
                new KeyIdent( name: "Fruits & Veg")
        ));
```

Output:

```
Plants created.
ArrayVal{StrVal{Apple}StrVal{Pear}StrVal{Grape}StrVal{Mango}StrVal{Cherry}} inserted into key {Fruits} on database {Plants}
ArrayVal{StrVal{Tomato}StrVal{Cauliflower}StrVal{Broccoli}StrVal{Zucchini}} inserted into key {Vegetables} on database {Plants}
Combined data with key Vegetables and data with key Fruits  into a new key Fruits & Veg
Retrieved: ArrayVal{StrVal{Tomato}StrVal{Cauliflower}StrVal{Broccoli}StrVal{Zucchini}StrVal{Apple}StrVal{Pear}StrVal{Grape}StrVal{Mango}StrVal{Cherry}} from database Plants
```

Program 9:

```
// Program 9 CREATE INSERT DELETE RETRIEVE
//  ---> error thrown : Invalid DB or Key does not exist. Since database was deleted
i.query(new Query.CREATE(
                new DatabaseIdent( name: "Computer Science Department")),
        new Query.INSERT(
                new DatabaseIdent( name: "Computer Science Department"),
                new KeyIdent( name: "Courses"),
                new Data.ArrayVal(new Data.StrVal("CS476"),
                                  new Data.StrVal("CS474"),
                                  new Data.StrVal("CS401"))),
        new Query.DELETE(new DatabaseIdent( name: "Computer Science Department")),
        new Query.RETRIEVE(new DatabaseIdent( name: "Computer Science Department"),
                           new KeyIdent( name: "Courses")));
```

Output:

```
Computer Science Department created.
ArrayVal{StrVal{CS476}StrVal{CS474}StrVal{CS401}} inserted into key {Courses} on database {Computer Science Department}
Deleted database Computer Science Department
Exception in thread "main" java.lang.Error Create breakpoint : Invalid DB or Key does not exists
    at com.company.Interpreter.query(Interpreter.java:41) <2 internal lines>
    at com.company.Interpreter.query(Interpreter.java:11)
    at com.company.RunQueries.main(RunQueries.java:155)
```

Source code:
https://github.com/ArtyVazquez/QueryDBLangDesign/blob/master/src/com/company/RunQueries.java

Things learned

Some interesting things that I learned is that the things learned in the course can be applied to any language such as Java. The reason why we used a functional language in the course was to keep the code concise and because functional languages allow us to translate rules much faster and easier because of pattern matching and variant types. Using other programming languages that are not functional such as Java achieve the same goals but with much more boilerplate code. I also found it interesting that type checking might not make sense depending on what language and how the language is being designed. For my language, type checking really didn't make much sense. The constructors for

the syntax only allowed for type correct programs thus there was no reason to implement a type checker.