

# BIG DATA SYSTEMS - REDIS

The main purpose of this project is to automate the process of importing relational tables to Redis. The project was developed by students Ilias Katsabalos and Leon Kalderon.

## 1.Steps for making the application work

- Open the Redis Server
- Unzip the file called **RedisProject**, anywhere you want
- The file **RedisProject** contains two folders

### 1.2 The **Application** folder

This folder contains two bat files, **InsertTables** and **InsertQuery**. Click on the desired **bat** file and **insert** the path of the **text** file that you want to process.

Another workaround is to put the txt files (tables, or queries) in the two respective folders that exist in the application folder. In that case, when executing **InsertTables.bat** you should add "tables\" before the name of the txt file. If you execute **InsertQuery.bat** you should put "queries\" before the name of the txt file.

### 1.3 The **Source** folder

This folder contains all the java source code. The first project is **RelToRed** and is responsible for inserting the data given in a txt file, into Redis, in the form of a hash set. The second project is **Query** and is responsible for executing queries on the database that you have created.

## 2. File format for smooth execution

We suggest you follow the guidelines listed below for a smooth execution.

### 2.1 Table format Constraints

- first line contains only table's name
- second line contains the primary key's name - assume a single attribute
- the rest of the attributes are in a single line each
- a line containing the character ';'.
- one or more lines, representing records, delimited by the character ";
- assume all attributes are of type string
- There is no whitespace in between the data or the end of the lines

Example relation: Student (SSN, FName, LName, Address, Age)

File contents:

Student  
SSN  
FName

LName  
Address  
Age  
;  
12938;Nikos;Papadopoulos;Hydras 28, Athens;42  
18298;Maria;Nikolaou;Kifisias 33, Marousi;34  
81129;Dimitris;Panagiotou;Alamanas 44, Petralona;29

## 2.2 Query format constraints

**first line (SELECT):** a list of relation\_name.attribute\_name, delimited by the character ","

**second line (FROM):** a list of relation names, delimited by the character ","

**third line (WHERE):** a very simple condition, consisting only of AND, OR, =, <>, > and <

Example:

Student.FName, Student.LName, Grade.Mark  
Student, Grade  
Student.SSN=Grade.SSN

Detailed constraints on the 'Where clause':

- There must be NO whitespace between the two operands and the arithmetic operator  
Correct: `Student.SSN=Grade.SSN`  
Wrong: `Student.SSN = Grade.SSN`
- There must be whitespace between the logical operators AND/OR  
Correct: `Student.SSN=Grade.SSN AND Grade.Mark>"7.5"`  
Wrong: `Student.SSN=Grade.SSNANDGrade.Mark>"7.5"`
- All constants, even numeric should be closed in double quotation marks as seen here `Grade.Mark>"7.5"`, since redis uses only strings as primitive types
- According to the `compareTo(String)` java method, upper-case letters are "bigger" than lower-case letters when compared

## 3. Understanding the workflow

As listed in the section 1.3, there are two projects. Here we will describe the workflow of the tasks. For more in depth information, please visit the source code and review the comments.

### 3.1 RelToRed Project

This projects contains two classes. The first one is **FileData**, who is responsible for inserting the records in Redis in the form of a hash set. The file is opened and processed using the second class called **OpenFile**.

## 3.2 Query Project

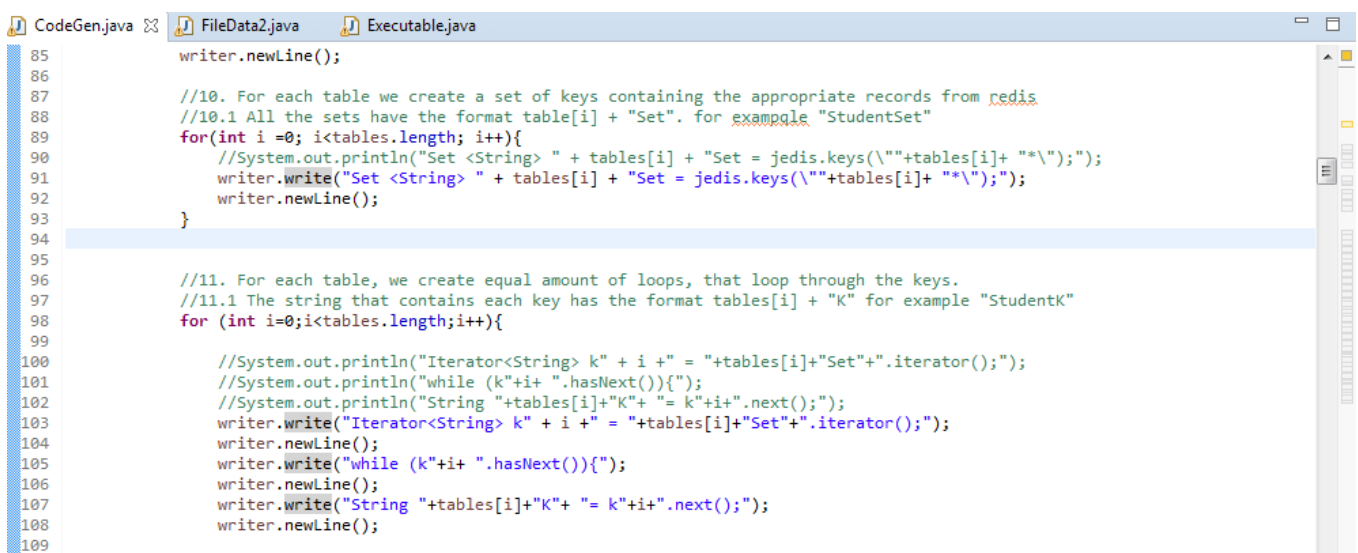
This project contains two classes. The first class is **ReadFile** and is responsible for reading the query txt file.

The second class is **CodeGen** and is responsible for generating the code for the **Executable** class. In other words, because we do not have the insight on how many tables exist in our query file, we need to store this metadata in **CodeGen** class and create the respective amount of loops dynamically in a new java class called **Executable**. These loops will iterate through all of our keys. In the lowest level of loops, we check if the records match the criteria of the SQL WHERE clause using the **JedBool** method. Let us navigate through an example:

Given Query:

Student.FName ,Student.LName, Grade.Mark, Country.CountryName  
Student, Country ,Grade  
Student.SSN=Grade.SSN AND Student.ID\_Country=Country.IDC

- **ReadFile** class reads our txt file that contains the query.
- **CodeGen** saves the metadata of the query in three arrays.
  - `select[]` : contains all the attributes that need to be selected, splitted by a comma. These attributes are passed into the **SelectRes** method and are replaced with `jedis.hget(StudentK, "Fname") + "," + jedis.hget(StudentK, "Lname") + ... +` This string is printed as code in the **Executable** class.
  - `tables[]`: contains all the tables that the query uses. According the length of this array, we print an equal amount loops in the **Executable** class, that iterate through the keys. In this example, we will create three sets, one for each table and three "while" loops to iterate through them.



```
85 writer.newLine();
86
87 //10. For each table we create a set of keys containing the appropriate records from jedis
88 //10.1 All the sets have the format table[i] + "Set". for example "StudentSet"
89 for(int i =0; i<tables.length; i++){
90     //System.out.println("Set <String> " + tables[i] + "Set = jedis.keys(\""+tables[i]+ "\");");
91     writer.write("Set <String> " + tables[i] + "Set = jedis.keys(\""+tables[i]+ "\");");
92     writer.newLine();
93 }
94
95
96 //11. For each table, we create equal amount of loops, that loop through the keys.
97 //11.1 The string that contains each key has the format tables[i] + "K" for example "StudentK"
98 for (int i=0;i<tables.length;i++){
99
100     //System.out.println("Iterator<String> k" + i + " = "+tables[i]+"Set"+".iterator();");
101     //System.out.println("while (k"+i+ ".hasNext()){");
102     //System.out.println("String "+tables[i]+"K"+ " = k"+i+".next();");
103     writer.write("Iterator<String> k" + i + " = "+tables[i]+"Set"+".iterator();");
104     writer.newLine();
105     writer.write("while (k"+i+ ".hasNext()){");
106     writer.newLine();
107     writer.write("String "+tables[i]+"K"+ " = k"+i+".next();");
108     writer.newLine();
109 }
```

The code in this picture, in the **CodeGen** class produces the following code in the **Executable** class

```

6 Jedis jedis = new Jedis("localhost");
7 Set <String> StudentSet = jedis.keys("Student*");
8 Set <String> CountrySet = jedis.keys("Country*");
9 Set <String> GradeSet = jedis.keys("Grade*");
10 Iterator<String> k0 = StudentSet.iterator();
11 while (k0.hasNext()){
12     String StudentK= k0.next();
13     Iterator<String> k1 = CountrySet.iterator();
14     while (k1.hasNext()){
15         String CountryK= k1.next();
16         Iterator<String> k2 = GradeSet.iterator();
17         while (k2.hasNext()){
18             String GradeK= k2.next();

```

- condition[]: contains the where clause of the sql query, splitted according to the regex "(?=&|OR)". The whole array is passed to the **Parser** method, that returns a String. In this example it returns :

```

7 if(JedBool(jedis.hget(StudentK,"SSN"), "=", jedis.hget(GradeK,"SSN")) & JedBool(jedis.hget(StudentK,"ID_Country"), "=", jedis.hget(CountryK,"IDC"))
8 System.out.println(jedis.hget(StudentK,"FName") + "," + jedis.hget(StudentK,"LName") + "," + jedis.hget(GradeK,"Mark") + "," + jedis.hget(Cou
,

```

The **JedBool** method has as input three parameters (String val1, String oper, String val2) and returns a Boolean. This method exists in the **Executable** class. The reason we created the JedBool method was the comparison between two strings. The **compareTo** method returns an integer, and we needed to add an extra mechanism in order to transpose this integer into a Boolean value.

First we check whether the values inserted are numeric or Strings. Then we make the appropriate transformations, either for double, integer or we leave them as strings. Then according to the oper parameter, we compare the values and return the Boolean. Finally, if the statement is true, the values are printed in the console.