**CS 542 – DATABASE MANAGEMENT SYSTEMS**

**INTERNALS PROGRAMMING OPTION**

**Project 3 – A query execution engine**

By:

Neha Satish Mahajan

Arun Vadivel

**CONTENTS**

Contents

[**INTRODUCTION:** 3](#_Toc431425085)

[**PURPOSE OF THE PROJECT:** 3](#_Toc431425086)

[**TECHNOLOGIES INVOLVED:** 3](#_Toc431425087)

[**DATA STORAGE STRUCTURE DESIGN:** 4](#_Toc431425088)

[**QUERY EXECUTION PLAN:** 5](#_Toc431425089)

[**LIST OF CLASSES AND METHODS:** 6](#_Toc431425090)

[**INTERACTIVE GUI TO COMMUINCATE WITH THE DATABASE SYSTEM:** 8](#_Toc431425091)

[**SAMPLE OUTPUT:** 9](#_Toc431425092)

[**TEST CASE:** 9](#_Toc431425093)

[**ASSUMPTIONS:** 12](#_Toc431425094)

# **INTRODUCTION:**

A database management system (DBMS) is a collection of programs that enables storage, modification, and information extraction from a database. In the first project we simply implemented key-value store functionality. In this project we focus on queries and how the queries are executed to fetch required data from the relation tables.

# **PURPOSE OF THE PROJECT:**

The project is focused on understanding the working of a query. How the queries are parsed into query execution plan and how data flows between each operator involved in the query. In this project we have a simple query which access two relations, city and country. The query focuses on finding the names of the cities whose population is 40% more than their country population.

# **TECHNOLOGIES INVOLVED:**

Language Implemented: Java (JDK 8)

GUI toolkit: Java Swing

Platform used: Eclipse

Files:

* 1. Relation.java
  2. Constants.java
  3. DemoApplication.java
  4. QueryExecution.java
  5. QueryExecution2.java
  6. Join.java
  7. Join2.java
  8. Selection.java
  9. Projection.java
  10. TestCountryData.java

# **DATA STORAGE STRUCTURE DESIGN:**

As the query accesses two different relation we have two files city.db and country.db each representing respective city and country table.

Each file has a maximum storage capacity of 5 MB. 1 MB of the total storage is allocated to the Metadata, so the file can store only 4 MB of actual data.

The 1 MB of metadata contains the auto generated key, location and length of the data that is stored in the data section. The metadata also contains the list of attributes associated with the table. The lists of attributes are stored as comma separated values in .db files.

Also the metadata section contains pointers which points to the free space in the metadata table and free space in the data section. There is another pointer to hold the value of the free space available in the data section.

The data for city and country table is obtained from the MySQL [World Database](http://dev.mysql.com/doc/world-setup/en/index.html) sample.

# **QUERY EXECUTION PLAN:**

The project implements two types of Query Execution Plan. Let us review them both in details.

1. QUERY EXECUTION PLAN 1:

The plan is executed by the QueryExecution.java class. The steps involved are as follows:

1. The city table has CountryCode as the Foreign key for the Code attribute of the country table.
2. So we first join both the tables such that city.CountryCode = country.Code. This operation is handled by the Join.java class. The input to the Join operator is Relation city and country. The join operator scans sequentially for both the tables and returns an intermediate result set with tuples from both the tables joined. Each tuple in the intermediate table has all the attributes from the city and the country table.
3. The result set of the Join operator is then send to the Selection operator where the tuples from the intermediate result set are filtered with the condition such that city.population > 40% of country population.
4. The filtered tuples from the Selection operator are fed into Projection operator which now prints only the city.name.

In this plan the number of tuples returned by the Join operator is 4079 since each city.CountryCode matches with only one country.Code.

The numbers of tuples returned by the Selection operator are 18, as only few satisfy the selection criteria.

And finally the Projection operator lists down the names of these 18 cities.

1. QUERY EXECUTION PLAN 2:

The plan is executed by the QueryExecution2.java class. The steps involved are as follows:

1. The city table has CountryCode as the Foreign key for the Code attribute of the country table.
2. The Join2.java class joins the two tables using city.CountryCode = country.Code and also with the selection criteria city.population > 40% of country population.
3. The result set of the Join operator is then send to the Projection operator which now prints only the city.name.

In this plan the number of tuples returned by the Join operator is 18. And the Projection operator lists down the names of these 18 cities.

As you can see both the Query Execution Plan returns the same number of tuples.

# **LIST OF CLASSES AND METHODS:**

1. **DemoApplication.java**

**public class DemoApplication**

1. public static void main(String[] args) - Launches the application
2. private void initialize() - Initializes the contents of the frame. Displays the GUI to interact to choose the action to be performed.
3. private byte[] readFile(String filename) – Append the filename with the correct address and read the contents of the file.
4. **Relation.java**

**public class Relation**

1. public void createDB(String filepath, String sAttributes) – Creates city.db and country.db file with the attributes.
2. public Relation(String sTableName, String sAttributes) – Creates new table with the specified table name and attributes. This method internally calls createDBto create .db files for the table name.
3. public void open() – Initializes the counter to start fetching the data from the first record in the table.
4. public String GetNext() - Returns a single tuple from the table. If end of table is reached returns NOT FOUND.
5. public void close().
6. public void insert(String tuple) – Inserts single tuple into .db file.

1. **Constants.java**

**public class Constants**

1. public static final int METADATA\_CURRENT\_POINTER = 0;

Points to the first byte address where the current metadata pointer starts in the file. Size is of 4 bytes. The 4 bytes contains the address to the location where next free space is available in the metadata table.

1. public static final int DATA\_CURRENT\_POINTER = 4;

Points to the first byte address where the current data pointer starts in the file.Size is of 4 bytes. The 4 bytes contains the address to the location where next free space is available in the data section.

1. public static final int FREESPACE\_POINTER = 8;

Points to the first byte address where the free space pointer starts in the file.Size is of 4 bytes. The 4 bytes contains the value of total free space available in the data section.

1. public static final int SIZE\_OF\_POINTER = 4;

The number of bytes allocated for each of the pointers.

1. public static final int METADATA\_TABLE\_POINTER = 1024;

Points to the address where the metadata table starts in the file.

1. public static final int METADATA\_TABLE\_END = 1048575;

Points to the address where the metadata table ends in the file.

1. public static final int DATA\_TABLE\_POINTER = 1048576;

Points to the address where the data section starts in a file.

1. public static final int SIZE\_OF\_KEY\_FIELD = 4;

Byte length of key column of the metadata table. Stores the key value.

1. public static final int SIZE\_OF\_POSITION\_FIELD = 4;

Byte length of position column of the metadata table. Stores the location of the data in the data section corresponding to the key.

1. public static final int SIZE\_OF\_LENGTH\_FIELD = 4;

Byte length of length column of the metadata table. Stores the total length of data in the data section corresponding to the key.

1. public static final int FILE\_LENGTH = 5242880;

Total number of bytes allocated to the entire file (pointers + metadata + data)

1. public static final int DELETED\_KEY\_VALUE = 0;

Default value for key when it is deleted

1. **QueryExecution.java**

**public class QueryExecution**

1. public QueryExecution () – Creates the query execution plan 1. Initializes the Join, Selection and Projection operators. Prints the result of the query.
2. **Join.java**

**public class Join**

1. public Join (Relation city, Relation country)
2. public void open() – Calls the city.open() method.
3. public ArrayList<String> getNext() – Sequentially scans through the city and country table and returns the joined tuple.
4. public void close()
5. public Boolean checkCondition(String cityTuple, String countryTuple) – Checks the condition city.CountryCode = country.Code.
6. **Selection.java**

**public class Selection**

1. public Selection (ArrayList<String> JoinResult)
2. public void open() – Initializes the JoinResult table iterator.
3. public ArrayList<String> getNext() – Sequentially scans through the JoinResult, returns the tuple which satisfy the condition city.population > 40% of country.population. If end of table reached returns NOT FOUND.
4. public void close()
5. **Projection.java**

**public class Projection**

1. public Projection (ArrayList<String> SelectionResult)
2. public void open() – Initializes the SelectionResult table iterator.
3. public ArrayList<String> getNext() – Sequentially scans through the SelectionResult, returns city.name. If end of table reached returns NOT FOUND.
4. public void close()
5. **QueryExecution2.java**

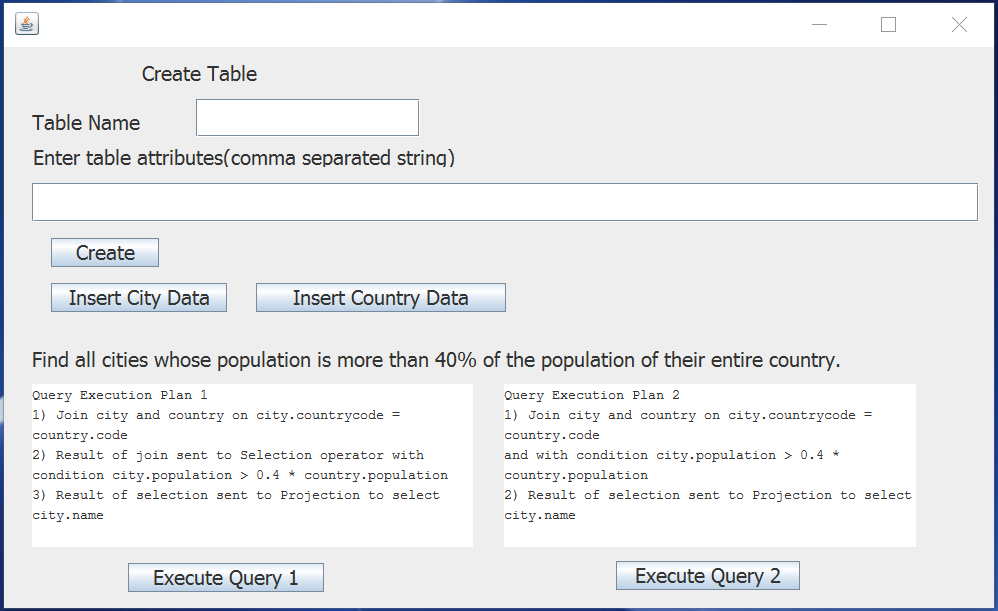
**public class QueryExecution**

1. public QueryExecution2 () – Creates the query execution plan 2. Initializes the Join2, and Projection operators. Prints the result of the query.
2. **Join2.java**

**public class Join2**

1. public Join2 (Relation city, Relation country)
2. public void open() – Calls the city.open() method.
3. public ArrayList<String> getNext() – Sequentially scans through the city and country table and returns the joined tuple.
4. public void close()
5. public Boolean checkCondition(String cityTuple, String countryTuple) – Checks the condition city.CountryCode = country.Code and city.population > 40% of country.population.

# **INTERACTIVE GUI TO COMMUINCATE WITH THE DATABASE SYSTEM:**



# **SAMPLE OUTPUT:**

1. The output of the Query Execution Plan 1 is:

Result Set length of Join operator 4079

Result Set length of Selection operator 18

Name of cities having population greater than 40 percent of the country population are:

Nassau

George Town

Avarua

Djibouti

Stanley

Gibraltar

Longyearbyen

Bantam

El-AaiÃºn

Macao

Dalap-Uliga-Darrit

Koror

Adamstown

Doha

Saint-Pierre

Victoria

Singapore

CittÃ  del Vaticano

1. The output of the Query Execution Plan 2 is:

Result Set length of Join operator18

Name of cities having population greater than 40 percent of the country population are:

Nassau

George Town

Avarua

Djibouti

Stanley

Gibraltar

Longyearbyen

Bantam

El-AaiÃºn

Macao

Dalap-Uliga-Darrit

Koror

Adamstown

Doha

Saint-Pierre

Victoria

Singapore

CittÃ  del Vaticano

# **TEST CASE:**

To test that the output of the query is correct or not, change the population of the city(i.e make its population less than 40 %) from the result of the query and check again if it appears again in the output or not. If it does not appear then the output is correct.

# **ASSUMPTIONS:**

1. Tuple values are entered from the city.txt and country.txt and they are comma separated.
2. Both city.db and country.db files are created properly.
3. The relation schema is same as given in the MySQL World Database example.
4. The Join, Selection, Projection operator classes are manipulated to work only for the given query. It is not implemented for generic queries.