**CS542 Project Three**

*Xiaoyan Sun & Shi Wang*

***1. Design and Assumptions***

In this project, we simulated a query execuation engine using the iteratior framework and observer pattern. The query executed are expressed as the following in SQL:

SELECT City.Name

From City, County

Where City.CountryCode=Country.Code

AND City.Poplulation > Country.Population

To execute this query, three operators are required, which are join, selection and projection. Here are some of the key points and assumptions:

* Relations are read in from .db files, and stored in memory as arraylist of tuples
* Tuples are stored in String[]
* The join function will produce, in an iterating manner, joined tuples of the two relations City and Country on their common attribute countrycode/code. For each call of the GetNext() in join operation, a new tuple of joined attributes is returned.
* The selection operation will select tuples that meet the requirement that city population is more than 40% of country population.
* Obeserver pattern is used between operator join and select. The select operator is notified when a new tuple is outputted by the join operator.
* Operations are executed in sequential order and the output is to print on screen, so no writing back to disk is necessary.

***2. Algorithm***

*2.1 the relation class*

the relation class contains three methods: Open(), GetNext() and Close()

Open(): read in data from .db file and store them in the arraylist of tuples

GetNext(): return the first tuple in the arraylist, remove the previous processed tuple from the arraylist.

*GetNext():*

|  |  |
| --- | --- |
| *public tuple GetNext(){* |  |
|  | *//Get the next tuple in the relation* |
|  | *if(!tuples.isEmpty()){* |
|  | *tuple t= tuples.get(0);* |
|  | *tuples.remove(0);* |
|  | *return t;* |
|  | *}* |
|  | *else{* |
|  | *return null;* |
|  | *}* |
|  | *}* |

*2.4 Join*

* We join two relations in the new tuples for the selection method.
* We use iterator methods for tuple-based nested-loop join of two relations. We repeatedly read two relations city and country.

*the join algorithm:*

*Open(){*

*R.Open();*

*S.Open();*

*s=S.GetNext();*

*}*

*GetNext(){*

*REPEAT{*

*r=R.GetNext();*

*IF(r=NotFound){*

*R.Close();*

*s=S.GetNext();*

*IF(s=NotFound){ Return NotFound;}*

*R.Open();*

*r.GetNext();*

*}*

*}UNTIL(r and s join);*

*RETURN the join tuple of r and s;*

*}*

*Close(){*

*R.Close();*

*S.Close();*

*}*

*2.5 Select & Project*

* We select the new tuples after join to find the required tuples.
* Then we project the required attribute.

*abstract of code:*

*//do the selection with an observer*

public void update(Observable o, Object arg)

t = (tuple) arg;

//Choose the required tuples by the condition

if(Integer.parseInt(t.getOthers()[4]) > Integer.parseInt(t.getOthers()[10])\*0.4)

***3. test and result***

*3.1 Use the class to find all cities whose population is more than 40% of the population of their entire country.*

*(1)method and process*

* We join the city and country tables with the same country code into new tuples.
* We select the tuples where the city’s population is more than 40% of the country’s population.
* We project the attribute city name.

*(2)result*

* We get 18 cities whose population is more than 40% of the country’s population. The order of the outputted tuples is according to relation city as expected.



*(3)correctness*

* The number of the city’s population is more than 40% of the country’s population.

