CENG 242

Hw #4

Spring 2006/2007

(Due: April 29th, 2007 Sunday 23:59)

In this homework, you will write a C++ code simulating a very simple Database Management System (DBMS).

You will have a class called **Table**. A table consists of attributes (columns) and some values for these attributes (rows). An example table can be:

Name	**ID**	**Grade**
Ali	114	90
Veli	234	80
Ali	285	77
Mehmet	333	93

In this example, Name, ID and Grade are attributes and (Ali,114,90), (Veli,234,80), (Ali,285,77) and (Mehmet,333,93) are values.

In the homework, you will implement some methods of relational algebra. Names of the attributes and types of all fields of values are assumed be **string**s in this simple model. The methods you will implement are:

- Table(vector<string> attributes)

  This method will create an empty table. The names of attributes are given as a vector of string.

Name	**ID**	**Grade**

- void insertRow(vector<string> value)

  This method will insert the given row to the table. If same value exists in the table throw an **Error** typed value REPEATEDVALUE. If size of the given vector is not same as number of attributes of the table, then throw an Error typed value INVALIDSIZE. Error is an enumerated type:

enum Error {INVALIDSIZE, REPEATEDVALUE, NONEXISTENTVALUE, INCOMPARABLE};

Before Insertion

Name	**ID**	**Grade**
Ali	114	90
Veli	234	80

After Insertion of (Ali,285,77)

Name	**ID**	**Grade**
Ali	114	90
Veli	234	80
Ali	285	77

- void deleteRow(vector<string> value)

  This method will delete the given row from the table. If the value does not exist in the table throw NONEXISTENTVALUE. If size of the given vector is not same as number of attributes of the table, then throw INVALIDSIZE.

Before Deletion

Name	**ID**	**Grade**
Ali	114	90
Veli	234	80
Ali	285	77

After Deletion of (Veli,234,80)

Name	**ID**	**Grade**
Ali	114	90
Ali	285	77

- Table select(vector<string> attributes, vector<Condition> conditions)

  This method returns a table having the attributes given as first argument and values satisfying **all** the conditions given as second argument. A Condition is a structure:

```
struct Condition {
    string attributeName;
    Oprtr op;
    string someValue;
};

enum Oprtr {EQ,NE,GT,GE,LT,LE}; // ==, !=, >, >=, <, <=
```

  All these operators should work as string comparison operators.

The Table

Name	ID	Grade
Ali	114	90
Veli	234	80
Ali	285	77

Resulting Table after Select (Name,ID) where (Grade != "77", ID > "100")

Name	ID
Ali	114
Veli	234

You should eliminate the duplicate values in the resulting table.

- Table join(const Table & otherTable)

  Joins the table with the given table and returns the resulting table. The result of the join operation is the set of all combinations of tuples in both tables that are equal on their common attribute names. e.g.

Table1

Name	ID	Dept
Ali	114	CENG
Veli	234	CENG
Ali	285	EEE
Mehmet	333	IE

Table2

Dept	Chair
CENG	Atalay
EEE	Erkmen
MATH	Alpay

Table1 join Table2

Name	ID	Dept	Chair
Ali	114	CENG	Atalay
Veli	234	CENG	Atalay
Ali	285	EEE	Erkmen

In the resulting table, the order of attributes should satisfy this:

  - The first table's attributes should come first, in the same order.
  - The second table's remaining attributes will come after, in the same order.

- Table project(vector<string> attributes)

  Projects the table with the given attributes and return the resulting table. You should eliminate the duplicate values in the resulting table.

The Table

Name	ID	Grade
Ali	114	90
Veli	234	80
Ali	285	77

Resulting Table after Project (ID,Grade)

ID	Grade
114	90
234	80
285	77

- Table division(const Table & otherTable)

  This method returns the division of first table with second table. You should eliminate the duplicate values in the resulting table.

Table1

Name	Dept
Ali	CENG
Veli	CENG
Ali	EEE
Mehmet	EEE
Veli	EEE
Mehmet	IE

Table2

Dept
CENG
EEE

Table1 division Table2

Name
Ali
Veli

- Table unionT(const Table & otherTable)

  This method takes the union of two tables having the same attributes. The order of the attributes in the resulting table should be the same as the first table's. You should remove the duplicates in the resulting table. If the tables have different attributes throw "INCOMPARABLE".

Table1

Name	ID	Dept
Ali	114	CENG
Veli	234	CENG
Ali	285	EEE
Mehmet	333	IE

Table2

Name	Dept	ID
Hasan	CENG	242
Masan	CENG	777
Ali	CENG	114

Table1 union Table2

Name	ID	Dept
Ali	114	CENG
Veli	234	CENG
Ali	285	EEE
Mehmet	333	IE
Hasan	242	CENG
Masan	777	CENG

- Table intersection(const Table & otherTable)

  This method takes the intersection of two tables having the same attributes. The order of the attributes in the resulting table should be the same as the first table's. You should remove the duplicates in the resulting table. If the tables have different attributes throw "INCOMPARABLE".

- Table difference(const Table & otherTable)

  This method takes the difference of two tables having the same attributes. The order of the attributes in the resulting table should be the same as the first table's. You should remove the duplicates in the resulting table. If the tables have different attributes throw "INCOMPARABLE".

- bool operator==(const Table & otherTable)

  This method determines whether two tables having the same attributes have same values (all the rows). If the tables have different attributes throw "INCOMPARABLE".

- bool operator!=(const Table & otherTable)

  This method determines whether two tables having the same attributes do not have same values (all the rows). If the tables have different attributes throw "INCOMPARABLE".

- bool operator<=(const Table & otherTable)

  This method determines whether the first table is a subset of second table having the same attributes (all the rows of first table is also a row of second table). If the tables have different attributes throw "INCOMPARABLE".

- bool operator<(const Table & otherTable)

  This method determines whether the first table is a proper subset of second table having the same attributes (all the rows of first table is also a row of second table, second table has extra rows). If the tables have different attributes throw "INCOMPARABLE".

- bool operator>=(const Table & otherTable)

  This method determines whether the first table is a superset of second table having the same attributes (all the rows of second table is also a row of first table). If the tables have different attributes throw "INCOMPARABLE".

- bool operator>(const Table & otherTable)

  This method determines whether the first table is a proper superset of second table having the same attributes (all the rows of second table is also a row of first table, first table has extra rows). If the tables have different attributes throw "INCOMPARABLE".

- friend ostream & operator<<(ostream & ost, const Table & theTable)

  This method prints the Table to output stream. The table should be printed in the form:

  $Attr_1$|$Attr_2$|…|$Attr_n$|
  $Value_{11}$|$Value_{12}$|….|$Value_{1n}$|
  $Value_{21}$|$Value_{22}$|….|$Value_{2n}$|
  …
  $Value_{m1}$|$Value_{m2}$|….|$Value_{mn}$|

  e.g.

  ```
  Name|ID|Grade|
  Ali|114|90|
  Veli|234|80|
  Ali|285|77|
  Mehmet|333|93|
  ```

  As you can see, there should not be any white spaces between strings (except newlines at the end of each line).

Specifications:

- For detailed information, you can take a look at *relational algebra*.
- You can define more private methods for *Table* class, but you should define the above methods as public. The names and types are strict for these methods.
- In error conditions mentioned above, only throw the exception. Do not write any code to catch it in your submission.
- You will have two files. One is **hw4.h** for your class declarations, structures, enumerated types (Error, Condition, Oprtr, Table etc.); the other is **hw4.cpp** for your definitions. You should **not** write **main** function in these files. I will write a *main* function in **hw4main.cpp** to test your codes. If you want to test your codes before submitting, write a *main* function in *hw4main.cpp*, include *hw4.h* from this file, download the **Makefile** from website, put them in the same folder and compile your codes with the command **make** in that folder in **inek** machines. You will submit a single tar file **hw4.tar** including *hw4.h* and *hw4.cpp*. You can tar your files with the command "*tar cvf hw4.tar hw4.cpp hw4.h*". Do not send me in other formats like ".tar.gz", ".rar", ".zip" etc.
- You will submit your codes through cow system. Specifications (file name, method names, class name, types etc.) are strict. Breaking any of them will cost you getting a 0 since black box method is used.