Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur
CS69202: Software Engineering Lab, Spring 2023
Assignment – 3

Submission Date – 28th February 2023

Question (60 + 30 + 10 = 100 marks):

Write a C++ program for implementing some functionalities of a Relational database using basic operators from Relational Algebra.

A relational database is a database where all data age organised in terms of *relations or tables* of *records or tuples or rows*. Relations can be thought of as sets of records. Each record is an array of *attribute* values (also called *columns*). Finally, the database is a collection of such relations. The attributes in a relation are also described in a *schema* using attribute names, which are used as keys for various operations on the data in a database. For more details, see:
https://en.wikipedia.org/wiki/Relational_database

In your program, implement the following classes:
```
class Attr { //Base class for attributes
    // Add operators for different attribute type in derived classes
    public: virtual bool operator== (const Attr & right) = 0;
      …
      …
      …
};

class Record { //storing data for each record
    vector <Attr *> attrptr;
    //methods
      …
      …
};

class Relation { // storing a relation
    int natttr, nrecs; // number of attributes and records
    vector <string> attrnames; // schema
    vector <int> attrinds; // mapping schema to indices
    list <Record *> recs; // list of records
    // methods
      …
      …
};
```

Relational algebra (https://en.wikipedia.org/wiki/Relational_algebra) describes 5 fundamental operations, each creating a new relation object from one or two. These operations can be used to build more complex transformations on relational data.

1. Union: All records of both R1 and R2.
   `Relation * union(Relation * R1, Relation * R2)`
2. Difference: Records in R1 but not in R2
   `Relation * difference (Relation * R1, Relation * R2)`
3. Cartesian Product: All possible pairs of records from R1 and R2
   `Relation * catersianproduct(Relation * R1, Relation * R2)`

4. Projection: New relation with a subset of columns
   `Relation * **projection**(Relation * R1, list<string> projectattrs)`
5. Selection: New relation with a subset of records matching a boolean expression in the attribute values in disjunctive normal form.
   `Relation * **union** (Relation * R1, DNFformula * f)`
   where
   `struct DNFformula { list <list <tuple <string, char, Attr *> > > ops; }` is a list of list of tuples with each tuple representing a comparison. The top level list stores disjunctions of clauses, each of which represents a list of conjunctions of comparisons. For simplicity assume that there are no negations.
6. Rename: rename an attribute in schema
   `Relation * difference (Relation * R1, string s1, string s2)`

Task 1 (50 marks): Implement the above 6 functionalities assuming that the generic comparison operators (==, !=, <, <=, >, >=) have been overloaded for each Attr object. Although the prototypes given here are non-member functions, implement the functionalities and member functions, when appropriate.

Task 2 (40 marks): Implement some common Attribute types, e.g. `integerAttribute, floatAttribute, stringAttribute`, etc. and overload the operators appropriately. Also, implement the natural join operator using the above implemented primitive operations:

```
Relation * naturaljoin(Relation * R1, Relation * R2, list<string>
joinattr)
```

The list of names in `joinattr` should be attribute names in both R1 and R2, and are the join attributes. See the excerpt below for hints on how to implement (source: https://en.wikipedia.org/wiki/Relational_algebra).

The natural join can be simulated with Codd's primitives as follows. Assume that $c_1,...,c_m$ are the attribute names common to R and S, $r_1,...,r_n$ are the attribute names unique to R and $s_1,...,s_k$ are the attribute names unique to S. Furthermore, assume that the attribute names $x_1,...,x_m$ are neither in R nor in S. In a first step the common attribute names in S can be renamed:

$$T = \rho_{x_1/c_1,\ldots,x_m/c_m}(S) = \rho_{x_1/c_1}(\rho_{x_2/c_2}(\ldots\rho_{x_m/c_m}(S)\ldots)) \tag{2}$$

Then we take the Cartesian product and select the tuples that are to be joined:

$$P = \sigma_{c_1=x_1,\ldots,c_m=x_m}(R \times T) = \sigma_{c_1=x_1}(\sigma_{c_2=x_2}(\ldots\sigma_{c_m=x_m}(R \times T)\ldots)) \tag{3}$$

Finally we take a projection to get rid of the renamed attributes:

$$U = \Pi_{r_1,\ldots,r_n,c_1,\ldots,c_m,s_1,\ldots,s_k}(P) \tag{4}$$

Task 3 (10 marks): Provide a simple console based user interface which provides the following functionalities using the above developed "library":

- Create a new empty table: the user can interactively enter the schema.
- Delete an existing table with all data in it.
- Add a record to a table.
- Print a table in appropriate format.
- Create a table from existing tables using the above developed operations.

Note:
70% marks are for implementing the above functionalities correctly and efficiently. 20% marks are for adhering to the object oriented programming paradigms (e.g. type safety, encapsulation, etc.), and 10% marks are for well-structured code with good documentation.

Submission:

Divide the code into the following:

- A header file (rdb.h) which has all the Library API.
- A c++ file (rdb-basics.cpp) with implementations of all the basic operators.
- A c++ file (rdb-join.cpp) with implementation of the join operation.
- A c++ file (rdb-attr.cpp) with implementations of all supported attribute types.
- A c++ file (rdb-main.cpp) with implementation of the UI.

Submit a zip file containing all the code files for the above problem in Moodle.