Relational Algebra Implementation in C++

Ishank Arora
Roll No: 14074009
Third Year CSE IDD

1 Introduction

Relational algebra is a procedural query language, which takes instances of relations as input and in turn yields instances of relations as output, using operators to perform queries. An operator can be either unary or binary. It has the following six basic operators:

- Select
- Project
- Rename
- Union
- Set Difference
- Cartesian Product

The other operations can be derived using these six operators.

2 Implementation in C++

The following files make up the project:

- rel_algebra.cpp: Contains the code implementing all the functions.
- queries.txt: Contains some sample queries demonstrating the relational algebra operations which have been implemented.
- relations: The folder contains four relations - students, courses, enrollment and houses, on which the sample queries have been implemented.

Error checking has been implemented to ensure that syntactically wrong queries entered by the user are responded with appropriate error message. Please note that either C++11 or C++14 is required to compile the CPP file, as it involves procedures which were not implemented in older versions of the language.

Usage: `g++ -std=c++14 rel_algebra.cpp`

The following operators of Relational Algebra have been implemented in C++.

- Select Operation: Select records from the table based on one or multiple predicates combined using AND(^) and OR(|) logical operators. An atomic predicate can be of the form 'column_name *operator* constant' or 'column_name_1 *operator* column_name_2' where *operator* is one of $\{>, <, \geq, \leq, =, \neq\}$.

  Usage: `S[predicate1 ^ (predicate2 | predicate3)] (relation_name)`

- Project Operation: Project columns from the given table.

  Usage: `P[column_list] (relation_name)`

- Rename Operation: Renames the relation as well as the attributes of the relation. The new list of column names is optional.

  Usage: `R[new_relation_name(new_column_names)] (relation_name)`

- Union Operation: Takes the union of the records of two union-compatible relations. The corresponding column names of the tables are assumed to be the same for union-compatibility.

  Usage: `U[relation_1_name] (relation_2_name)`

- Set Difference: Returns the records of first relation which are not present in the second relation, given that the relations are union-compatible.

  Usage: `D[relation_1_name] (relation_2_name)`

- Cartesian Product: Takes the cartesian product of the two relations.

  Usage: `C[relation_1_name] (relation_2_name)`

- Intersection Operation: Takes the intersection of the records of two union-compatible relations.

  Usage: `I[relation_1_name] (relation_2_name)`

- Join Operation: Takes the join of two relations, keeping only those records which have the same values for all the common attributes among the relation.

  Usage: `J[relation_1_name] (relation_2_name)`

- Division Operation: Returns projection of attributes present in the first relation but not the second, such that only those records which occur for all distinct values of attributes in the second relation are present.

  Usage: `V[relation_1_name] (relation_2_name)`

- Aggregrate Operations: The following aggregrate functions have been implemented:

  - Select Maximum Operation: Selects the maximum value for a given attribute from the relation.
    Usage: `X[column_name] (relation_name)`
  - Select Minimum Operation: Selects the minimum value for a given attribute from the relation.
    Usage: `N[column_name] (relation_name)`
  - Select Sum Operation: Selects the sum of values for a given numerical attribute from the relation.
    Usage: `T[column_name] (relation_name)`
  - Select Average Operation: Selects the average of values for a given numerical attribute from the relation.
    Usage: `A[column_name] (relation_name)`
  - Select Count Operation: Selects the count of values for a given attribute from the relation.
    Usage: `O[column_name] (relation_name)`