An Algebraic Query Language

August 8, 2022

Contents

1 Introduction

We can think of Relational Algebra as a means to construct new relations from given relations. That is, we use this algebra for the purpose of querying and modifying data. In the future, we will connect this idea to the notion of a View, and how to instantiate / save the results of a relational algebra expression to a relation.

The operations of our relational algebra fall into the following classes. We call such operations of any complexity, queries.

2 Operations

1. Set Operations
    - **Prerequisite:** The two sets must have schemas with identical sets of attributes, and an identical domain. Additionally, the columns must be sorted in the same order.
    - **Union($\cup$):** The set of elements that are in both sets. Do note that an element appears only once in the union. Similar to mathematics, there are no repetitions in sets.
    - **Intersection($\cap$):** The set of elements that are shared between both relations.
    - **Difference($-$):** The set of elements that are in the LHO, but not the RHO.
2. Selection
    - $\sigma_C(R)$ produces a new relation with a subset of R's tuples. The tuples in the resulting relation are those that satisfy condition C, involving the attributes of R.
    - Do note that the schema for the relation produced by the selection operator is the same for R.
3. Projection
    - $\pi_{A_1,A_2,...,A_n}(R)$ produces a relation from R, that only has some of its original columns, denoted by the list provided as $A_1, A_2, ..., A_n$
4. Cartesian Product
    - The relation produced by $R \times S$ produces the set, which returns the pair of all possible tuples between R and S.

- The relation schema is the union of the schemas R, S. Do note that if R and S share attributes with the same name, you must perform renaming to prevent issues.
- **Exercise:** You have been given sample code on GitHub. Run it on PostgreSQL, and see if your intuition follows from the SQL representation of the relational algebra!

5. Joins

- We use the concept of joins to pair tuples of two relations, only on values that match in some way. There is a natural definition, and a theta definition, which allows the programmer to specify a condition on which to join tuples.
- **Natural Joins**

  a) $R \bowtie S$ produces a relation, which effectively pairs tuples from R and S that agree in the intersection of attributes.

  b) We call such successful pairings, joined tuples. A tuple that fails to pair is a dangling tuple.
- **Theta-Joins**

  a) A theta-join, $R \bowtie_C S$ allows the programmer to pair tuples, using a specified condition, C.

  b) The resulting relational schema is the union of schemas R and S, with dot notation, if it is required as such.
- Note: The easiest way to think about theta-joins, is to break it down into its constituent sub-operations, underneath the hood of it all. That is, $R \bowtie_C S$:

  1)$X = R \times S$

  2)$Y = \sigma_C (R)$

6. **Naming and Renaming**

- We shall use $\rho_{s(A_1,A_2,...,A_n)} (R)$ to rename a relation R to S. Additionally, we can rename its attributes by specifying its new said names in the list, $A_1, A_2, ..., A_n$
- If we do not wish to rename any attributes, omit the list, following S.

3 Additional Exercises

1. Are there operational equivalences in our defined relational algebra, on sets? If so, provide some examples.

2. Are there operations in relational algebra, which cannot be defined by using other operations? If so, which operations are independent from each other?

3. For each operation, provide the PostgreSQL implementation (command).

4. Play around with the Library Database that has been provided. Write some queries, and get some hands on experience with relational algebra and its equivalent PostgreSQL implementation!