CSCI 403: Databases
11 - The Relational Algebra

Relational Model Redux

We return to the theory of relational databases as developed by E.F.Codd in his 1970 paper. Recall that a relation is defined as a set of tuples, where a tuple is a set of values associated with named attributes. The relational algebra considers the useful operations that can be applied to relations and the resulting algebra of relations. These operations all have realizations in the SQL language (which came about substantially later than the algebra), although as we've already seen, SQL does not adhere tightly to the relational model (e.g., multiset relations).

There is also a relational calculus, which we will not cover in this course, but which is discussed in your textbook. The relational calculus forms part of the foundation of the SQL language.

Unary Operations

The relational algebra has two unary operations, projection and selection, which give rise to the SELECT and WHERE clauses (respectively) as well as a renaming operation which gives rise to the renaming and aliasing facilities in SQL (i.e., the AS keyword). There is notation associated with each of these operations, which lets us compactly write expressions in the algebra.

Selection

The Select operation chooses a subset of tuples from a relation according to a condition on the attributes of the tuples (think WHERE clause). In effect, the Select operation partitions the tuples into two sets according to whether they satisfy the condition, discarding the set of tuples which do not satisfy the condition.

Letting R represent some relation, the Select operation is notated as

$$\sigma_{condition}(R).$$

R can be a variable, as in

$$R = mines\_courses,$$

or it can be the bare name of a relation, or it can be an expression in the algebra which results in a relation.

For example,

$$\sigma_{course\_id='CSCI403'}(mines_courses)$$

applies the selection operation to the mines_courses relation to obtain only those tuples where the attribute course_id is equal to the constant 'CSCI403'. Note that the result here is another relation, that is, a set of tuples which retain the attribute names from the original relation.

As with the WHERE clause of a SQL query, we can make compound conditions using the usual AND, OR, and NOT operations. A common extension is to also allow type-appropriate functions on attributes as well, such as a substring operation:

$$\sigma_{substring(course\_id\ from\ 1\ for\ 4)>=300}(mines\_courses)$$

selects tuples from mines_courses for courses at the 300 level or higher.

The selection operation is unary (applies to a single relation) and the condition applies to single tuples; no aggregate functions here!

Properties of the Select operator

- The degree of the relation (# of attributes) resulting is the same as the degree of the of the original relation.

- The number of tuples in the resulting relation is always less than or equal to the number in the original. The fraction of tuples selected is referred to as the selectivity of the selection condition.

- Selection is commutative:
$$\sigma_{cond1}(\sigma_{cond2}(R)) = \sigma_{cond2}(\sigma_{cond1}(R))$$

- A sequence of Select operations can always be converted into a single Select operation with the conjunction (AND) of all the Select conditions:
$$\sigma_{cond1}(\sigma_{cond2}(R)) = \sigma_{cond1ANDcond2}(R).$$

Projection

Whereas the Select operation partitions relations horizontally (by rows), the Project operation partitions relations vertically (by attributes). The Project operation is notated as

$$\pi_{attr1,attr2,...}(R),$$

where attr1, attr2, ... specifies the attributes of R to be retained in the resulting relation. Note that we can repeat attributes, so the resulting relation can have higher degree than the original.

For example:

$$\pi_{instructor,course\_id}(mines\_courses)$$

would result in a relation with only unique tuples of instructor and course_id from the mines_courses relation.

Note that the Project operator functions much like the SELECT part of a SQL query (while the Select operator functions like the WHERE clause).

Properties of the Project operator

- The number of tuples in the resulting relation is always ¡= the number of tuples in the original. It can be less than due to the elimination of duplicates (if the selected attributes do not form a superkey of the relation, for instance).

- The projection operation is not commutative, rather
$$\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>},$$
assuming list1 only contains attributes also found in list2 (otherwise the expression is malformed).

Renaming

Formally, the renaming operator lets us rename relations, attributes, or both as a unary operation - similar to using the AS keyword in SQL. The general form is

$$\rho_{S(B1,B2,...)}(R)$$

where S is the new name of the relation, and B1, B2, ... are the new names of R's attributes. S or the attribute list are optional. If the attribute list is included, it must match in degree the number of attributes in R, and the attributes will be renamed in the same order as the usual ordering of the attributes.

Sequences of Operations

All of the above unary operations can be nested, e.g.,

$$\rho_{(name,course\_id)}(\pi_{instructor,course\_id}(\sigma_{department='EECS'}(mines\_courses))),$$

which corresponds to (is not necessarily equivalent to) the SQL query

```
SELECT instructor AS name, course_id
FROM mines_courses
WHERE department = 'EECS';
```

Alternately, we can do a sequence of operations, giving a name to each relation (think variables) as we go:

$$\begin{aligned} R1 &= \sigma_{department='EECS'}(mines\_courses) \\ R2 &= \pi_{instructor,course\_id}(R1) \\ R3 &= \rho_{(name,course\_id)}(R2) \end{aligned}$$

Set Operations

Union, Intersection, and Set Difference (or Minus) can be applied as binary operations when both operands are relations with the same number and types of attributes. These work the same as UNION, INTERSECTION, and EXCEPT in SQL.

Notation:
Union: $A \cup B$
Intersection: $A \cap B$
Difference: $A - B$

Properties of Set Operations

- Union and Intersection are both commutative:

$$\begin{aligned} A \cup B &= B \cup A \\ A \cap B &= B \cap A \end{aligned}$$

- Union and Intersection are both associative:

$$\begin{aligned} A \cup (B \cup C) &= (A \cup B) \cup C \\ A \cap (B \cap C) &= (A \cap B) \cap C \end{aligned}$$

- Set difference is **not** commutative or associative!
- Intersection can be expressed in terms of Union and Difference (and thus is not technically necessary for completeness):

$$A \cap B = ((A \cup B) - (A - B)) - B - A$$

  or even just in terms of Difference:

$$A \cap B = A - (A - B)$$

Cartesian Product and Joins

The Cartesian (cross) product:

$$A \times B$$

has the same meaning in relational algebra as a cross-product join in SQL: the resulting relation has each tuple from A paired with every tuple from B. The attributes of the newrelation are the concatenation of attributes from A and B. Thus if A has m attribute and B has n attributes, then $A \times B$ has $m + n$ attributes. The size of $A \times B$ is $|A| \times |B|$.

As in SQL, this operation has little utility on its own. Typically it is paired in relational algebra with a subsequent Select operation to eliminate irrelevant pairings of tuples. Since this is such a common pattern, another operation, Join, was created. The notation for the binary operation Join is:

$$A \bowtie_{condition} B$$

which equates to

$$\sigma_{condition}(A \times B)$$

Note: there are some variations in notation between the book and other sources I've looked at concerning joins and related operations. I will try to follow the book's notation where there are differences.

For example, we could perform the join

$$mines\_courses \bowtie_{instructor=name} mines\_eecs\_faculty.$$

When we have $A \bowtie_{cond} B$ and the join condition is of the general form
cond1 AND cond2 AND ...,
and cond$n$ is of the form $A_i \theta B_j$ where $A_i \in A$ and $B_j \in B$ and θ is one of the usual comparison operators (equals, less than, etc.), then the join is called a "theta-join".

When θ is the equality operation, the join is called an "equijoin". The nature of this join is such that the resulting relation will agree completely on every value between the pairs of columns participating in the join expression. Therefore, it is usually desirable to project away one of the duplicates.

In the frequent case in which the paired attributes have the same name in both relations, we can apply yet another special join: the "natural join". In the book, the notation for the natural join is

$$A * B$$

but other sources use $\bowtie$ with no conditions to mean a natural join.

E.g.,

$$mines\_courses \ * \ mines\_courses\_meetings$$

would join on the equality of the CRN attribute.

If the paired attributes do not have the same name, then a renaming must be done before applying the natural join, e.g.:

$$mines\_courses * \rho_{name=instructor}(mines\_eecs\_faculty).$$

Note that the natural join will equate all attributes of the same name in the two relations.
Terminology: the join selectivity of a condition is the expected size of the join divided by the max size of the cross product.

Completeness

It can be demonstrated that the set $\{\sigma, \pi, \cup, \rho, -, \times\}$ forms a complete set of operators for the relational algebra, at least as originally formulated. That is, all other relational algebra operations can be expressed as some combination of operations using the above set.

Other Operations and Extensions

Division ($\div$) - this is a truly odd operator, which is difficult to explain the utility of, and which has no equivalent in SQL. Please read the book for more info. (Division is equivalent to a sequence of basic relational algebra operations.)

Aggregates/grouping: this is not expressible using the base relational algebra, but obviously has importance for SQL databases. The notation for doing aggregates and grouping is

$$< grouping\ attributes > \Im_{<function\ list>}(R)$$

Generalized projection: projection is generally extended to provide for functions on attributes, e.g.

$$\pi_{F_1(A_1),F_2(A_2),\ldots}(R)$$

Recursive or transitive closure: no real notation. This is some hardcore relational juju that actually does have a SQL equivalent, and may be useful for certain kinds of self-referential querying (e.g., getting a hierarchical company view from supervisor-employee relationships).

Outer joins: no LATEXsymbols found for these! Look in book - they look like the join symbol with extensions.