CS2 Current Technologies note 7

Relational Algebra.

Peter Buneman

7.1 Relational Algebra

The reason for having relational databases and SQL is because there is a very simple *algebra* of database operations. SQL is, roughly speaking, turned into a relational algebra expression, this expression is rewritten into a more efficient form and then evaluated using a bunch of well-developed algorithms.

It's important to remember that relational algebra is just that – a set of operations that act on tables to produce new tables. Just as we operate on numbers with arithmetic, we operate on tables with relational algebra.

- Selection selects rows from a table. The symbol for selection is σ_P where P is a predicate on the attributes. E.g.

  $$\sigma_{\texttt{room}=52}(\text{courses})$$

  Conditions are made up of comparisons between attributes or between attributes and constants. These can be combined with conjunction, disjunction and negation.

  As will be seen shortly, selection is equivalent to `SELECT * FROM ... WHERE` in SQL.

- Equally simple is projection. which corresponds to a simple `SELECT ... FROM ...` (no `WHERE`)

  $$\pi_{\texttt{cname},\texttt{teacher}}(\text{courses})$$

  We shall see that projection is realized in SQL simply as `SELECT ... FROM ...` (no `WHERE`).

- Union ($\cup$) and difference ($\setminus$) work as before, but now we shall assume that the we can only take the union of two tables if they have the same schema. Note that we haven't included intersection in this catalogue because it can be expressed in terms of difference:

  $$R \cap S = R \setminus (R \setminus S)$$

- Join. This comes in a variety of flavors, but we shall use just one, $\bowtie$, called *natural join*. Here is how $R \bowtie S$ works. Suppose the set of attributes of R is A_R and that of S is A_S. The set of attributes of $R \bowtie S$ is $A_R \cup A_S$. A tuple in R and a tuple in S *match* if they agree on their common attribute values (the attributes in $A_R \cap A_S$). Two such matching tuples can be combined into a tuple on $A_R \cup A_S$.

  Example. Consider

  students:

id	name	email
S0123	White	sw@dot.com
S0456	Prince	prince@dd.edu
...	...	...

  takes:

id	cname	marks
S0123	CompSci3	75
S0456	Hist4	62
S0123	French3	70
S0456	CompSci3	60
...	...	...

  The natural join of these tables is:

id	name	email	cname	marks
S0123	White	sw@dot.com	CompSci3	75
S0456	Prince	prince@dd.edu	Hist4	62
S0123	White	sw@dot.com	French3	70
S0456	Prince	prince@dd.edu	CompSci3	60
...	...	...	...	...

- Relabeling. Since our operations are "label sensitive", we need a method for relabeling attributes (for example, we might want to join the student and teacher table to see if there are any students with the same names as teachers. This is an operation that we tend to forget about when thinking about optimisation or expressive power, but it's important if we are to complete the relational algebra. The relabeling operation is ρ and it is subscripted by a set of $\mathit{old\text{-}name} \to \mathit{new\text{-}name}$ pairs. E.g.

$$\rho_{\mathtt{id}\to\mathtt{sid},\mathtt{name}\to\mathtt{sname}}(\mathtt{students})$$

  Note that if we relabel a field involved in the join, it changes what the join does.

The relational algebra is an *idealisation* of what happens in practice. SQL does not use natural join, but it does optimise joins which work on equality of attributes, and we can use ideas from the relational algebra, though their expression as code in the SQL optimiser is a bit more messy.

For example, consider $R \bowtie S \bowtie T$ and suppose that R contains 10^4 tuples, S contains 10^5 tuples and T contains 10 tuples. Which of the following expressions is likely to be a good evaluation strategy: $(R \bowtie S) \bowtie T$, $R \bowtie (S \bowtie T)$, $(R \bowtie T) \bowtie S$? Note that $\bowtie$ is an associative, commutative idempotent operation.

7.2 Relational Algebra exercises

1. Do the following identities hold[1]?

   - $R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$,
   - $R \bowtie (S \setminus T) = (R \bowtie S) \setminus (R \bowtie T)$,
   - $\pi_F(R \cup T) = \pi_F(R) \cup \pi_F(T)$, and
   - $\pi_F(R \setminus T) = \pi_F(R) \setminus \pi_F(T)$.

   Answer:

   - $R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$ – yes.
   - $R \bowtie (S \setminus T) = (R \bowtie S) \setminus (R \bowtie T)$ – yes.
   - $\pi_F(R \cup T) = \pi_F(R) \cup \pi_F(T)$ – yes.
   - $\pi_F(R \setminus T) = \pi_F(R) \setminus \pi_F(T)$ – no. E.g., $F = \{B\}$, $R = \begin{array}{|c|c|}\hline A & B \\ \hline 1 & 2 \\ 1 & 3 \\ \hline \end{array}$, $S = \begin{array}{|c|c|}\hline A & B \\ \hline 2 & 2 \\ \hline \end{array}$.

2. Under what conditions is $\sigma_C(\pi_F)(R) = \pi_F(\sigma_C(R))$? C is a condition and F is a subset of the attributes of R.

   Answer: Suppose the condition C mentions the attributes F' of R. If $F' \not\subseteq F$ then $\pi_F(\sigma_C(R))$ is not properly defined. In all other cases, both sides are well-defined and equal.

3. Describe $R \bowtie S$ when (a) R and S have exactly the same attributes and (b) when they have no attributes in common.

   Answer:

   (a) Intersection.

   (b) Something that looks like a Cartesian product. Every tuple of R combined with every tuple of S.

4. Consider $\sigma_C(R \bowtie S)$ (this is a very common form of query.) How would one go about rewriting this as a more efficient query.

   Answer: As a general rule, joins are expensive and selections are cheap, so one does selections first. Let F_C be the attributes mentioned in the condition C and let F_R and F_S be the attributes of R and S respectively.

   Suppose $F_C \subseteq F_R$, then we can rewrite the query as $\sigma_C(R) \bowtie S$. Similarly if $F_C \subseteq F_S$.

   Suppose $C = C' \wedge C''$ with $F_{C'} \subseteq F_R$ and $F_{C''} \subseteq F_S$. We can rewrite the query as $\sigma_{C'}(R) \bowtie \sigma_{C''}(S)$.

   If $C = C' \vee C''$ with $F_{C'} \subseteq F_R$ and $F_{C''} \subseteq F_S$, the query can still be rewritten as $(\sigma_{C'}(R) \bowtie S) \cup (R \bowtie \sigma_{C''}(S))$, but whether this is more efficient than the original query now depends on other questions such as how "selective" the two selections $\sigma_{C'}, \sigma_{C''}$ are.

[1] The last two of these were not on the original handout.

5. Suppose that the table R has a numeric attribute A and that numeric comparisons are allowed in selections. How would you find the rows in R with the fifth largest value for A? Do *not* try to write down the whole query, but explain in detail how to construct the query.

   Answer: The general strategy is to find the rows in R with the largest value for A, subtract those rows, and repeat the process three more times to get a table from which we take the rows with the largest value for A.

   The hard part is finding the rows with the largest value for A. This is a form of universal quantification; as we shall see, SQL will handle this in more or less the same way with a long query.

   First we construct a table of all A values and all possible pairs of A-values:

   $$R_A = \pi_A(R) \qquad P = \rho_{A\to A_1}(R_A) \bowtie \rho_{A\to A_2}(R_A)$$

   Here, $\rho_{\ldots}$ is the relabelling operation. It is needed because $\bowtie$ would otherwise produce the identity!

   Next, from P, we can find the A-values that are less than some other A-value:

   $$H' = \rho_{A_1\to A}(\sigma_{A_1<A_2}(P))$$

   Subtract this from H to get the single row table containing the maximum A-value. Join this with the original table R to get the rows with the maximum A-value:

   $$R_{\mathsf{max}} = R \bowtie (H \setminus H')$$

   As we shall see, in SQL one has aggregate functions so this can be shortened to something like

   ```
   SELECT *
   FROM   R
   WHERE A = SELECT MAX(A) FROM R
   ```

   However, even in SQL we have to go through the iteration to get to the fifth largest value. This shows one way in which the relational algebra fails to be turing-complete. There is no term in the relational algebra that will find the tuples with the nth largest A-values.