More Relational Algebra and the Relational Calculus

CS157A

Chris Pollett

Oct. 5, 2005.

Outline

- Complete sets of operators
- Division
- Aggregation and Grouping
- Outer Joins and Unions
- Tuple Relational Calculus
- Domain Relational Calculus

Complete sets of operators

- One can show that $\{\sigma, \pi, \cup, -, \times\}$ is a complete set. That is, all the other relational algebra operations we have defined can be defined using just these operations.
- These operations will turn out to be what you can do in the relational calculus.
- Already, we have seen join can be expressed using selection and product.
- Set difference can be expressed as:

$$R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$$

Division

- The division operator is useful for expressing the following kind of query: Retrieve the names of employee who work on all projects that John Smith works on:

$$SMITH \leftarrow \sigma_{FNAME='John' AND LNAME='Smith'}(EMPLOYEE)$$
$$SMITH\_PNOS \leftarrow \pi_{PNO}(WORKS\_ON \bowtie_{ESSN=SSN} SMITH)$$
$$SSN\_PNOS \leftarrow \pi_{ESSN,PNO}(WORKS\_ON)$$
$$SSNS(SSN) \leftarrow SSN\_PNOS \div SMITH\_PNOS$$
$$RESULT \leftarrow \pi_{FNAME,LNAME}(SSNS * EMPLOYEE)$$

- With a little work one can show division can be expressed in term of projection, cartesian product, and difference.

Aggregation and Grouping

- There are some useful operations which cannot be defined in terms of the basic operations of the relation algebra. For instance: counts, averages. There are other which are awkward to express like minimums, and maximums.
- These are collectively called **aggregate functions**.
- Another useful kind of operation is to be able to group counts, averages, etc by some attribute. For instance, average salaries in each department.
- The general notation for both these kinds of ops is:

$$< \text{grouping attribute} > \mathfrak{F}_{<\text{function list}>}(R)$$
$$DNO\mathfrak{F}_{COUNT\ SSN}(EMPLOYEE)$$

Recursive Closure Operations

- The relational algebra does not support transitive closures of relations.
- So for instance, given a relation Parent(x,y), it is possible to define a query that might return the grandparents of Bob. Or even great-grandparents of Bob.
- But there is no single query that will return all ancestors of Bob.
- SQL3 proposes a syntax for such transitive closures in SQL.

Outer Joins

- In R*S, only tuples in R which have matching tuples in S are kept.
- One could imagine wanting to keep all tuples in R in the output. If a tuple doesn't match anything in S, just have the values for the columns of S be null.
- This is called a **left outer join,** denoted $=\bowtie$.
- One can also have **right outer joins** and full **outer joins**. $\bowtie=$, $=\bowtie=$
- The kinds of joins we had before are sometimes called **inner joins**.

Outer Unions

- We might also want to do unions of sets which are not union compatible.
- For instance, suppose we had R(X,Y) and S(X,Z).
- We say r from R and s from S match if r(X)=s(X). In which case, in an outer union we output one tuple where we use the values for Y and Z in the remaining slots.
- If a tuple doesn't match any tuple from the other relation we include it in the output but we pad it with nulls.

Tuple Relational Calculus

- This is another query language for the relational model, this time based on logic.
- It is more declarative in the sense that we declare what data we want, rather than say how to get it.
- It turns out the Relational Calculus is of equivalent expressive power as the relational algebra.
- A typical query in the tuple relational calculus looks like: { t | COND(t)}.
- For instance,

  {t | EMPLOYEE(t) AND t.SALARY > 5000}

  {t.FNAME, t.LNAME | EMPLOYEE(t) AND t.SALARY > 5000}
- EMPLOYEE(t) is called the **range relation** of tuple t.
- Informally, a tuple calculus expression gives, (1) a range relation R for each tuple t, (2) a condition to select particular combinations of tuples, and (3) a set of selected attributes.
- We are allowed to combine atomic conditions using AND, OR, NOT.

Existential and Universal Quantifiers

- In addition to the above ways to create relational calculus expression we can also create conditions using existential and universal quantifiers.
- For instance, the query: Retrieve the name and address of all employees who work for the research department, might be expressed as:

  {t.FNAME, t.LNAME, t.ADDRESS | EMPLOYEE(t) AND (∃d)(DEPARTMENT(d) AND d.NAME = 'Research' AND d.DNUMBER=t.DNO)}

  To express: find the names of employee who work on all projects controlled by department 5.

  {e.FNAME, e.LNAME | EMPLOYEE(e) AND [(∀x)(NOT(PROJECT(x)) OR NOT (x.DNUM=5) OR ((∃w)(WORKS_ON(w) AND w.ESSN =e.SSN AND x.PNUMBER =w.PNO))]}

  Note: in the above it might be hard to figure out all the things which are not projects. So might replace the universal with:

  NOT (∃ x)(PROJECT(x) AND (x.DNUM=5)…

Safe versus Unsafe Expressions

- An expression which is guaranteed to only return a finite number of outputs is called a **safe** expression.
- Otherwise, an expression is called **unsafe**.
- For example,

  {t | NOT (EMPLOYEE(t))} is unsafe.

Domain Relational Calculus

- This is similar to the tuple relational calculus except now we are work with quantifiers over attributes.
- For example, to retrieve the birthday and address of the employee named John B Smith we might use the query:
  - {uv | (∃q) (∃r) (∃s) (∃t) (∃w) (∃x) (∃y) (∃z)(EMPLOYEE(qrstuvwxyz) AND q='John' AND r='B' AND s='SMITH')}