CS352 Lecture - Relational Calculus; QBE

Last revised 9/26/08

Objectives:

1. To briefly introduce the tuple and domain relational calculi
2. To briefly introduce QBE.

Materials

1. Jason Rozen QBE demo Senior Project

I. The Relational Calculus

A. The relational calculus is an alternative to the relational algebra as a basis for query languages. It is derived from predicate calculus.

1. A predicate is an assertion that we require to be true. When we formulate a query in the relational calculus, we specify a predicate that the object(s) we are looking for must satisfy.

2. Unlike relational algebra - which is procedural - relational calculus is non-procedural - i.e. we specify what requirements the result must satisfy, not how to compute it.

B. Just as the relational algebra serves as the mathematical foundation for the commercial query language SQL, the relational calculus serves as the mathematical foundation for various commercial visual query languages.

C. There are two variants of the relational calculus: the tuple relational calculus and the domain relational calculus.

1. In the tuple relational calculus, variables represent tuples, and predicates are formulated in terms of attributes of a tuple variable.

   Ex: Find book tuples for which author = dog:

   { t | t ε book ^ t[author] = dog }

2. In the domain relational calculus, variables represent individual attributes, and complete tuples are represented as lists of attributes.

   Ex: The above query:

   { <call_number, title, author> |
   <call_number, title, author> ε book ^ author = dog }

3. In either case, we represent a query by a predicate which we want the result to satisfy.

D. It is important to see that there are definite analogues between operations in relational algebra and predicates in relational calculus.

1. In fact, it can be shown that the systems are of equivalent power, in the sense that any query that can be formulated in the relational algebra (without the extensions we have discussed) can be formulated in either of the relational calculi, and any safe formula (we define this later) in either of the relational calculi is equivalent to some query in the relational algebra.

2. Thus, it is possible for one system to support query languages of both types by internally translating from a query of one type into the language used internally (most often a variant of relational algebra, since this is procedural in nature.)

E. We will now consider examples of relational calculus equivalents to each basic relational algebra operation.

1. Relational algebra SELECTION:

$\sigma_{\text{author = dog}}(\text{book})$

See the two examples above

2. Relational algebra PROJECTION:

$\pi_{\text{last\_name, first\_name}}(\text{borrower})$

a) tuple r.c.:

$\{ t \mid \exists s (s \in \text{borrower} \wedge t[\text{last\_name}]=s[\text{last\_name}] \wedge t[\text{first\_name}]=s[\text{first\_name}]) \}$

where t is on the new scheme (last_name, first_name)

b) domain r.c.:

$\{ \langle \text{last\_name, first\_name} \rangle \mid \exists \text{ borrower\_id } (\langle \text{borrower\_id, last\_name, first\_name} \rangle \in \text{borrower}) \}$

3. Relational algebra NATURAL JOIN: checked_out |X| borrower

a) i. tuple r.c.:

{ t | ∃ u, v (u ε checked_out ^ v ε borrower ^
u[borrower_id] = v[borrower_id] ^
t[borrower_id] = u[borrower_id] ^
t[call_number] = u[call_number] ^
t[date_due] = u[date_due] ^
t[last_name] = v[last_name] ^
t[first_name] = v[first_name]) }

where t is on the new scheme

(borrower_id, call_number, date_due, last_name, first_name)

b) domain r.c.:

{ <borrower_id, call_number, date_due, last_name, first_name> |
<borrower_id, call_number, date_due> ε checked_out ^
<borrower_id, last_name, first_name> ε borrower }

4. Relational algebra UNION: Suppose we have two tables that have the same scheme - say student_borrower and fac_staff_borrower - and want a table including all persons in either group:

a) tuple r.c.:

{ t | (t ε student_borrower) v (t ε fac_staff_borrower) }

b) domain r.c.:

{ <borrower_id, last_name, first_name> |
(<borrower_id, last_name, first_name> ε student_borrower) v
(<borrower_id, last_name, first_name> ε fac_staff_borrower)}

5. Relational algebra DIFFERENCE: Suppose, instead, we have a general borrower table - which contains information on all borrowers - plus a student_borrower table, which contains only student borrowers, and we want to list borrowers who are not students. In relational algebra, this would be borrower - student_borrower

a) tuple r.c.:

{ t | (t ε borrower) ^ ¬ (t ε student_borrower) }

b) domain r.c.:

{ <borrower_id, last_name, first_name> |
(<borrower_id, last_name, first_name> ε borrower) ^
¬ (<borrower_id, last_name, first_name> ε student_borrower)}

6. A more complete example:

Find the last name of all borrowers who have an overdue book

π σ checked_out |X| borrower
last_name date_due <
-- whatever today is --

Ask class to do in each relational calculus:

tuple:

{ t | ∃ u,v
(u ε checked_out ^ v ε borrower ^
t[last_name] = v[last_name] ^
u[borrower_id] = v[borrower_id] ^
u[date_due] < "September 18, 2002") }

domain:

{ <last_name> | ∃ borrower_id, call_number, date_due, first_name
(<borrower_id, call_number, date_due> ε checked_out ^
<borrower_id, last_name, first_name> ε borrower ^
date_due < "September 18, 2002") }

F. One important property of relational calculus formulas is SAFETY. A relational calculus formula is said to be safe iff it does not require us to inspect infinitely many objects.

1. Example: the query { t | ¬ (t ε R) } is not safe. For any relation R, there are infinitely many tuples that are not members of that relation!

2. Unsafe formulas are most often the result of improper use of not. In general, the set of all values NOT satisfying some predicate is infinite; so when not is used it must be coupled with an additional condition that narrows the scope of consideration - e.g. if P(x) and Q(x) are safe formulas then

¬ Q(x)

is unsafe, but

P(x) ^ ¬ Q(x)

is safe.

3. The most common way to ensure that a formula is safe is to include a formula that restricts consideration to tuples from a particular relation - i.e. a formula of the form t ε R or <a,b,c> ε R.

4. Notice that the issue of safety is unique to the relational calculus. One cannot formulate an unsafe query in the relational algebra - hence only safe relational calculus formulas have relational algebra equivalents.

II. Another Commercial Query Language: QBE

A. As we noted earlier, there have been many different commercial relational query languages. One other we will look at briefly is QBE - Query by Example.

1. Like SQL, QBE was developed by IBM. Today, it is supported for accessing databases from personal desktop assistants (for which SQL is a poor tool because of the lack of a standard keyboard).

2. Microsoft Access includes a visual query facility called QBE which is very similar to the original IBM model, though not identical.

3. The original QBE was designed for use with text-based terminals. Micro-computer QBE implementations make use of graphics and direct manipulation.

B. The basic idea in QBE is this: one formulates a query by selecting one or more tables.

1. In the original QBE, one then specified for each of the columns of a selected table whether it is to be:

   a) Constrained to have a specific value, or a value lying within a specific range.

   b) Constrained to match the value in some column of some other table involved in the query.

   c) Printed regardless of what value it contains

   d) Ignored

   Example: Suppose one used a QBE-like facility with our sample library database, and wanted to see the first names of borrowers whose last name is "Aardvark". Using the format of the original QBE display, one would need to use just the borrowers table, and would setup the grid as follows:

borrower	borrower_id	last_name	first_name
		Aardvark	P.

When the query was run, the desired names would appear in the first_name column.

Example: Suppose one wanted to see the names of borrowers together with the titles of books they have checked out.

- One needs to use information from three tables: the borrower table (for borrower names), the book table (for book titles) and the checkout table.

- Using the format of the original QBE display, one would select these tables and set up something like this:

borrower	borrower_id	last_name	first_name
	_x	P.	P.

book	call_number	title	au
	_y	P.	

checked_out	borrower_id	call_number	date_due
	_x	_y	

2. In a graphical version of QBE, one would only choose the columns one was interested in (thus borrower_id would not be selected for the first query, and author and date_due would not be selected for the second.) Moreover, the "join-condition" would be specified by drawing a line between columns, rather than by using variables.

3. Note that the sample queries are equivalent to the following domain relational calculus queries:

   { < f > | < i, "Aardvark", f > ∈ borrower }

$$\{ < l, f, t > | < i, l, f > \in \text{borrower} \wedge < c, t, a > \in \text{book} \wedge < i, c, d > \in \text{checked\_out} \}$$

4. Of course, there is also a relational algebra or SQL equivalent - e.g. for SQL:

   select first_name from borrower where last_name = "Aardvark";

   select last_name, first_name, title
   from borrower natural join book natural join checked_out

   (In fact, a QBE implementation might actually translate the query into SQL if that is the "native" query language of the DBMS)

C. Demonstration: Jason Rozen Senior Project

1. Launch (double click jar - but leave on intro screen) Connect to the cs211 LIBRARY database on moses

   URL: `jdbc:mysql://moses.cs.gordon.edu:3306/LIBRARY`
   Username/Password: bjork

2. Connect directly to the same database on moses:

   ```
   mysql -u bjork -p
   use LIBRARY; show tables;
   - select * from each
   ```

   Note the same list of tables in the dropdown; put up all three and note how column names match results on direct connect

3. Now formulate the following query in QBE: list the titles of all overdue books (assume today is February 1, 2001 in light of data in database!) (Enter < '2001-02-01' for dateDue, _c for callNo in CheckedOut and Book; check print Title)

4. Now formulate the following query in QBE: list the titles of all books that Donna Dog has out.

   a) From manual inspection of tables, who should this be?

   b) Do in QBE. (Enter Dog as lastName for borrower; _i as borrowerID in both borrower and checkedOut; _c as callNo in both checkedOut and Book; check Print for title in Book; copy generated SQL to clipboard then Submit Query)

   c) Look at generated SQL by paste to a text editor, then run on moses