

CS352 Lecture - Relational Calculus; QBE

Last revised December 18, 2020

Objectives:

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

Materials

1. Projectable of java and RC code examples
2. Prolog demo of a declarative program
3. QBE Demo
4. Projectable form of QBE Examples in lecture

I. **The Relational Calculus**

A. The relational calculus is a non-procedural formal query language. 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.

Example: suppose we wanted to find the positive integer square root of a number that is a perfect square - e.g the positive integer square root of 9 is 3, while there is no such for 10 (since it has no integer square root).

- a) All the programming languages you are likely to be familiar with are procedural. To accomplish this task procedurally, we would specify a process for computing it - e.g. newton's method, which we might encode this way in java:

```

int positive_integer_square_root(int Square) {
    int guess = Square / 2;
    while (guess * guess != Square)
        guess = (guess + Square / guess) / 2;
    return guess;
}

```

PROJECT

(Of course, a similar program could easily be written in python or C++ or whatever)

b) Now, consider a non_procedural way for carrying out the same task.

(1) First, we define what we mean to say something is the square root of something: A number Root is the positive integer square root of some number Square if Root is a positive integer and $Root * Root = Square$.

This might be said in Prolog (the most widely known non-procedural language) this way:

```

positive_integer_square_root(Root, Square) :-
    positive_integer(Root),
    Square is Root * Root.

```

(2)Of course, we also need to define what we mean by a positive integer. One formal mathematical definition is that 1 is a positive integer, and number that is 1 more than a positive integer is also a positive integer. In Prolog:

```

positive_integer(1).
positive_integer(X) :-
    positive_integer(Y),
    X is Y + 1.

```

(3)Believe it or not, this a actually a valid Prolog program that will find the positive integer square root of any integer having such a root.

DEMO

(Note that it will go into an infinite loop for any number not having such a root - but a simple fix can allow it to fail or to give the ceiling or the floor of the answer - but this is beyond the scope of this lecture!)

B. Just as the relational algebra serves as the mathematical foundation for the commercial query language SQL, the relational calculus has served as the mathematical foundation for several commercial query languages, including one we will look at this briefly.

C. There are two variants of the relational calculus: the tuple relational calculus and the domain relational calculus. Both use variables in formulating predicates, but they use them in different ways.

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:

$$\{ t \mid t \in \text{book} \wedge t[\text{author}] = \text{dog} \}$$

PROJECT

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

Ex: The above query:

$$\{ \langle \text{call_number}, \text{title}, \text{author} \rangle \mid \langle \text{call_number}, \text{title}, \text{author} \rangle \in \text{book} \wedge \text{author} = \text{dog} \}$$

PROJECT

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.) (In fact, the demonstration program we will use for a query language based on the relational calculus - QBE - does just that, translating a query into SQL and then executing that form.)

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

1. Relational algebra SELECTION:

σ book
author = dog

See the two examples above

2. Relational algebra PROJECTION:

Π borrower
last_name
first_name

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)

PROJECT

b) domain r.c.:

$$\{ \langle \text{last_name}, \text{first_name} \rangle \mid \exists \text{borrower_id} \\ (\langle \text{borrower_id}, \text{last_name}, \text{first_name} \rangle \in \text{borrower}) \}$$

PROJECT

3. Relational algebra NATURAL JOIN: checked_out \bowtie borrower

a) i. tuple r.c.:

$$\{ t \mid \exists u, v (u \in \text{checked_out} \wedge v \in \text{borrower} \wedge \\ u[\text{borrower_id}] = v[\text{borrower_id}] \wedge \\ t[\text{borrower_id}] = u[\text{borrower_id}] \wedge \\ t[\text{call_number}] = u[\text{call_number}] \wedge \\ t[\text{date_due}] = u[\text{date_due}] \wedge \\ t[\text{last_name}] = v[\text{last_name}] \wedge \\ t[\text{first_name}] = v[\text{first_name}]) \}$$

where t is on the new scheme

$$(\text{borrower_id}, \text{call_number}, \text{date_due}, \text{last_name}, \text{first_name})$$

b) domain r.c.:

$$\{ \langle \text{borrower_id}, \text{call_number}, \text{date_due}, \text{last_name}, \text{first_name} \rangle \mid \\ \langle \text{borrower_id}, \text{call_number}, \text{date_due} \rangle \in \text{checked_out} \wedge \\ \langle \text{borrower_id}, \text{last_name}, \text{first_name} \rangle \in \text{borrower} \}$$

PROJECT

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.:

$$\{ t \mid (t \in \text{student_borrower}) \vee (t \in \text{fac_staff_borrower}) \}$$

PROJECT

b) domain r.c.:

$$\{ \langle \text{borrower_id}, \text{last_name}, \text{first_name} \rangle \mid \\ (\langle \text{borrower_id}, \text{last_name}, \text{first_name} \rangle \in \text{student_borrower}) \vee \\ (\langle \text{borrower_id}, \text{last_name}, \text{first_name} \rangle \in \text{fac_staff_borrower}) \}$$

PROJECT

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 $\text{borrower} - \text{student_borrower}$

a) tuple r.c.:

$$\{ t \mid (t \in \text{borrower}) \wedge \neg (t \in \text{student_borrower}) \}$$

b) domain r.c.:

$$\{ \langle \text{borrower_id}, \text{last_name}, \text{first_name} \rangle \mid \\ (\langle \text{borrower_id}, \text{last_name}, \text{first_name} \rangle \in \text{borrower}) \wedge \\ \neg (\langle \text{borrower_id}, \text{last_name}, \text{first_name} \rangle \in \text{student_borrower}) \}$$

PROJECT

6. A more complete example:

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

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

Ask class to do in each relational calculus:

a) tuple:

$$\{ t \mid \exists u, v \\ (u \in \text{checked_out} \wedge v \in \text{borrower} \wedge \\ t[\text{last_name}] = v[\text{last_name}] \wedge \\ u[\text{borrower_id}] = v[\text{borrower_id}] \wedge \\ u[\text{date_due}] < \text{"February 13, 2017"}) \}$$

PROJECT

b) domain:

$$\{ \langle \text{last_name} \rangle \mid \exists \text{borrower_id, call_number, date_due, first_name} \\ (\langle \text{borrower_id, call_number, date_due} \rangle \in \text{checked_out} \wedge \\ \langle \text{borrower_id, last_name, first_name} \rangle \in \text{borrower} \wedge \\ \text{date_due} < \text{"February 13, 2017"}) \}$$

PROJECT

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 $\{ t \mid \neg (t \in R) \}$ 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

$$\neg Q(x)$$

is unsafe, but

$$P(x) \wedge \neg Q(x)$$

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 $t \in R$ or $\langle a,b,c \rangle \in R$.
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. Query Languages based on the Relational Calculus

- A. The tuple relational calculus was the basis of the query language QUEL, which was developed in conjunction with the INGRES DBMS research project. Ingres later morphed into a product called postgres, but along the way also switched over to using SQL,
- B. The book discusses a database query language called Datalog - a subset of Prolog which is based the domain relational calculus and is still used in a variety of specialized areas including some places in machine learning.
- C. A graphical query language called QBE - Query by Example is also based on the domain relational calculus.
 1. Like SQL, QBE was developed by IBM.
 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.
- D. 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
PROJECT		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	author
	_y	P.	
checked_out	borrower_id	call_number	date_due
	_x	_y	

PROJECT

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

$$\{ \langle f \rangle \mid \langle i, \text{"Aardvark"}, f \rangle \in \text{borrower} \}$$

$$\{ \langle l, f, t \rangle \mid \langle i, l, f \rangle \in \text{borrower} \wedge \langle c, t, a \rangle \in \text{book} \wedge \langle i, c, d \rangle \in \text{checked_out} \}$$

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

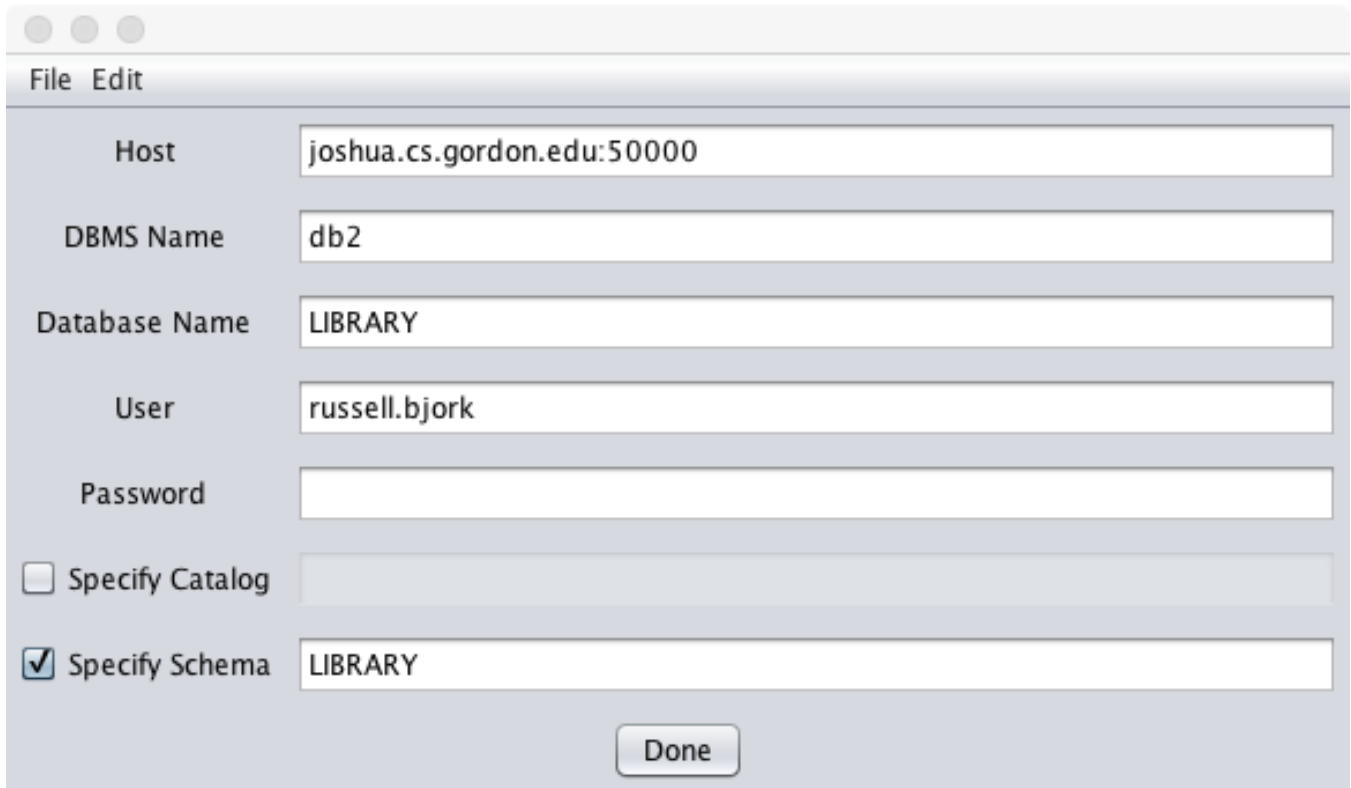
```
select first_name from borrower where last_name = "Aardvark";
```

```
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)

E. Demonstration: QBE Demo program

1. Launch QBE.jar. Connect as shown below



The screenshot shows a window titled "QBE Demo" with a menu bar containing "File" and "Edit". The window contains several input fields and checkboxes for database connection settings:

- Host:** joshua.cs.gordon.edu:50000
- DBMS Name:** db2
- Database Name:** LIBRARY
- User:** russell.bjork
- Password:** (empty field)
- Specify Catalog (empty field)
- Specify Schema LIBRARY

A "Done" button is located at the bottom center of the dialog.

2. Connect directly to the same database **using db2 -t**, then select * from each table

3.

4. In QBE, chose Query. Note the same list of tables in the dropdown; put up Borrower and note how column names match results on direct connect

a. Now formulate a query to list the full information of all borrowers.

BORROWER	BORROWERID	LASTNAME	FIRSTNAME
<input checked="" type="checkbox"/> Print All	PRINT	PRINT	PRINT
Remove			

b. Show SQL

c. Perform - use Next to step through

5. Now remove table and instead put up the book and checked_out tables.

a. Formulate the following query in QBE: list the titles of all overdue books.

BOOK	ACCESSIONNO	TITLE	AUTHOR	CALLNO
<input type="checkbox"/> Print All	OUTPUT	OUTPUT	OUTPUT	OUTPUT
				_C
Remove		Fire-hydrants I have known 21 ways to cook a cat Karate		

CHECKEDOUT	CALLNO	BORROWERID	DATEDUE
<input type="checkbox"/> Print All	OUTPUT	OUTPUT	OUTPUT
	_C		< 1/29/2019
Remove			

- b. Show SQL
 - c. Perform
6. Now put borrower table up along with other two
- a. Consider the following query in QBE: list the titles of all books that Donna Dog has out.
 - b. From manual inspection of tables, who should this be?
 - c. Do in QBE

BOOK	ACCESSIONNO	TITLE	AUTHOR	CALLNO
<input type="checkbox"/> Print All	<input type="text" value="OUTPUT"/>	<input type="text" value="OUTPUT"/>	<input type="text" value="OUTPUT"/>	<input type="text" value="OUTPUT"/>
<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value="_C"/>
<input type="text" value=""/>	<input type="text" value="Fire-hydrants I have known
21 ways to cook a cat"/>		<input type="text" value=""/>	<input type="text" value=""/>
<input type="button" value="Remove"/>	<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>

BORROWER	BORROWERID	LASTNAME	FIRSTNAME
<input type="checkbox"/> Print All	<input type="text" value="OUTPUT"/>	<input type="text" value="OUTPUT"/>	<input type="text" value="OUTPUT"/>
<input type="text" value=""/>	<input type="text" value="_I"/>	<input type="text" value="Dog"/>	<input type="text" value=""/>
<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>
<input type="button" value="Remove"/>	<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>

CHECKEDOUT	CALLNO	BORROWERID	DATEDUE
<input type="checkbox"/> Print All	<input type="text" value="OUTPUT"/>	<input type="text" value="OUTPUT"/>	<input type="text" value="OUTPUT"/>
<input type="text" value=""/>	<input type="text" value="_C"/>	<input type="text" value="_I"/>	<input type="text" value=""/>
<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>
<input type="button" value="Remove"/>	<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>

- d. Show SQL - copy to clipboard
 - e. Perform
 - f. Paste SQL into terminal - add terminating ;
7. Replace tables by employee table - twice
- a. Consider the following query: list names of employees who earn more than their supervisors
 - b. From manual inspection of the tables, who should this be?
 - c. Do in QBE - note use of condition box

EMPLOYEE	SSN	LAST_NAME	FIRST_NAME	SALARY	SUPERVISOR_SSN
<input type="checkbox"/> Print All	OUTPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT
				..S1	..S
Remove		elephant fox	emily frederick		

EMPLOYEE	SSN	LAST_NAME	FIRST_NAME	SALARY	SUPERVISOR_SSN
<input type="checkbox"/> Print All	OUTPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT
	..S			..S2	
Remove					

Conditions

..S1 > ..S2

d. Perform

8. Choose Insert. Add a row to borrower, then do select on terminal

BORROWER	BORROWERID	LASTNAME	FIRSTNAME
<input type="button" value="Remove"/>	<input type="checkbox"/> null	<input type="checkbox"/> null	<input type="checkbox"/> null
	<input type="text" value="99999"/>	<input type="text" value="Aardvark"/>	<input type="text" value="Aaron"/>

9. Delete this borrower, then do select on terminal

BORROWER	BORROWERID	LASTNAME	FIRSTNAME
<input type="button" value="Remove"/>	<input type="text" value="99999"/>	<input type="text"/>	<input type="text"/>