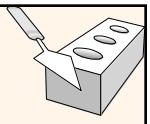
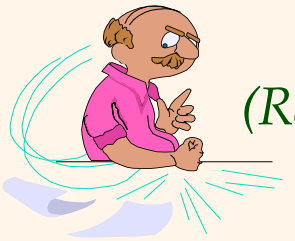
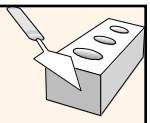


Introduction to Data Management



Lecture #13
(Relational Languages III)

Instructor: Mike Carey
mjcarey@ics.uci.edu



Announcements

- Midterm #1 is now behind you!
  - We're done with logical database design
  - We'll get the exams graded as quickly as we can
- We are now in relational query-land
  - Last lecture we finished the relational algebra
  - We then started the tuple relational calculus
- HW#4 is now available (due next ***Monday***)
  - Focus is on relational algebra queries *(RelaX)*
  - Today we'll finish the calculus and start SQL!

Announcements

Topic Coverage and Exam Schedule

Syllabus

Topic	Reading
Databases and DB Systems	Ch. 1
Entity-Relationship (E-R) Data Model	Ch. 2.1-2.5, 2.8
Relational Data Model	Ch. 3.1-3.2
E-R to Relational Translation	3.5
Relational Design Theory	Ch. 19.1-19.6, 20.8
Midterm Exam 1	*Mon, Apr 29* (in class)
Relational Algebra	Ch. 4.1-4.2
Relational Calculus	Ch. 4.3-4.4
SQL Basics (SPJ and Nested Queries)	Ch. 3.4, 5.1-5.3
SQL Analytics (Aggregation, Nulls, and Outer Joins)	Ch. 5.4-5.6
Advanced SQL Goodies (Constraints, Triggers, Views, and Security)	Ch. 3.3, 3.6, 5.7-5.9, 21.1-21.3, 21.7
Midterm Exam 2	*Wed, May 22* (in class)
Tree-Based Indexing	Ch. 9.1, 8.1-8.3, 10.1-10.2
Hash-Based Indexing	Ch. 10.3-10.8, 11.1
Physical DB Design	Ch. 8.5, 20.1-20.7
Semistructured Data Management (*a.k.a.* NoSQL)	AsterixDB SQL++ Primer, Couchbase SQL++ Book
Basics of Transactions	Ch. 16 and Lecture Notes
Endterm Exam	*Fri, Jun 7* (in class)

You are here…

Ex: TRC Query Semantics

t(a1, a2, …)

Sailors

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	4	25.5
95	Bob	3	63.5

sid	bid	date
22	101	10/10/98

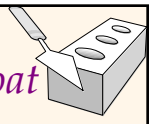
bid	bname	color
101	Interlake	blue

sid	sname	rating	age
22	Dustin	7	45.0

nid	nname	nvalue
1	Pi	3.14159...

sid	sname	rating	age
66	Donald	0	70.0

Find names of sailors who've reserved a red boat



Sailors(sid, sname, rating, age) Reserves(sid, bid, day)
Boats(bid, bname, color)

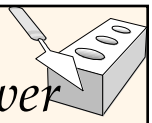
{ t(sname) | ∃s ∈ Sailors (t.sname = s.sname ∧
∃r ∈ Reserves (r.sid = s.sid ∧
∃b ∈ Boats (b.bid = r.bid ∧ b.color = 'red'))) }

table与table之间链接

- Things to notice:
  - Again, how result schema and values are specified
  - How joins appear here as value-matching predicates
  - *Highly declarative nature of this form of query language!*

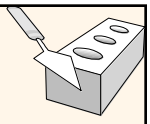
∃∄∀∈∉ ¬ ∧∨⇒=≠<>≤≥

Unsafe Queries and Expressive Power



- It is possible to write syntactically correct calculus queries that have an *infinite* number of answers! Such queries are called *unsafe*.
  - *E.g.,* s | ¬ (s ∈ Sailors)
- It is known that every query that can be expressed in relational algebra can be expressed as a safe query in DRC / TRC; the converse is also true.
- *Relational Completeness*: Query language (e.g., SQL) can express every query that is expressible in relational algebra/calculus.

Find ids of sailors who've reserved a red boat and a green boat

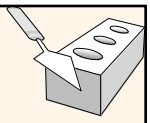


Sailors(sid, sname, rating, age) Reserves(sid, bid, day)
Boats(bid, bname, color)

$$\{ t(sid) \mid \exists s \in Sailors\ (t.sid = s.sid \wedge \exists r1 \in Reserves\ (r1.sid = s.sid \wedge \exists b1 \in Boats\ (b1.bid = r1.bid \wedge b1.color = 'red')) \wedge \exists r2 \in Reserves\ (r2.sid = s.sid \wedge \exists b2 \in Boats\ (b2.bid = r2.bid \wedge b2.color = 'green')))\}$$

- Things to notice:
  - This required ***several*** more variables! (**Q:** Why?)
  - *Q:* Could we have done this with just s, **r**, b1, and b2? (And why?)
  - Think of tuple variables as fingers pointing at the tables' rows

Example: Tuple Variable Bindings



Sailors

sid	sname	rating	age	
22	Dustin	7	45.0	s
29	Brutus	1	33.0	
31	Lubber	8	55.5	
32	Andy	8	25.5	
58	Rusty	10	35.0	
64	Horatio	7	35.0	
71	Zorba	10	16.0	
74	Horatio	9	35.0	
85	Art	4	25.5	
95	Bob	3	63.5	

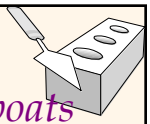
(Bindings at one point in time… ☺)

Reserves

sid	bid	date	
22	101	10/10/98	
22	102	10/10/98	r1
22	103	10/8/98	r2
22	104	10/7/98	
31	102	11/10/98	
31	103	11/6/98	
31	104	11/12/98	
64	101	9/5/98	
64	102	9/8/98	
74	103	9/8/93	

Boats

bid	bname	color	
101	Interlake	blue	
102	Interlake	red	b1
103	Clipper	green	b2
104	Marine	red	

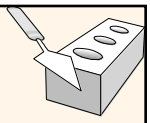


Find the names of sailors who've reserved all boats

Sailors(sid, sname, rating, age) Reserves(sid, bid, day)
Boats(bid, bname, color)

$$\{ t(\text{sname}) \mid \exists s \in \text{Sailors} (t.\text{sname} = s.\text{sname} \wedge \forall b \in \text{Boats} (\exists r \in \text{Reserves} (r.\text{sid} = s.\text{sid} \wedge b.\text{bid} = r.\text{bid}))) \}$$

- Things to notice:
  - Universal quantification addresses the "all" query use case
  - *Highly declarative nature of this form of query language!*



Find the names of sailors who've reserved all Interlake boats

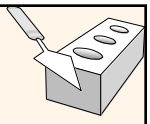
Sailors(sid, sname, rating, age) Reserves(sid, bid, day)
Boats(bid, bname, color)

$$\{ t(\text{sname}) \mid \exists s \in \text{Sailors} \; (t.\text{sname} = s.\text{sname} \wedge \forall b \in \text{Boats} (b.\text{bname} = \text{'Interlake'} \Rightarrow (\exists r \in \text{Reserves} \; (r.\text{sid} = s.\text{sid} \wedge b.\text{bid} = r.\text{bid})))) \}$$

- Or, if you prefer:

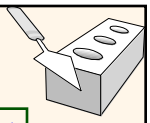
$$\{ t(\text{sname}) \mid \exists s \in \text{Sailors} \; (t.\text{sname} = s.\text{sname} \wedge \forall b \in \text{Boats} (b.\text{bname} \neq \text{'Interlake'} \vee (\exists r \in \text{Reserves} \; (r.\text{sid} = s.\text{sid} \wedge b.\text{bid} = r.\text{bid})))) \}$$

Relational Calculus Summary



- Relational calculus is non-operational, so users define queries in terms of ***what*** they want and not in terms of ***how*** to compute it. (*Declarativeness:* "What, not how!")
- Algebra and safe calculus subset have the same expressive power, leading to the concept of relational completeness for query languages.
- Two calculus variants: TRC (tuple relational calculus, which we've just studied) and DRC (domain relational calculus).

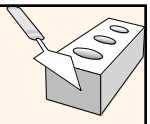
On to SQL...!



SQL "SPJ" Query:

```
SELECT    [DISTINCT] target-list
FROM      relation-list
WHERE     qualification
```

- *relation-list* A list of relation names (possibly with a *range-variable* after each name).
- *target-list* A list of attributes of relations in *relation-list*
- *qualification* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of <, <=, =, >, >=, <>) combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are *not* eliminated! (Bags, not sets.)



Many SQL-Based DBMSs

- Commercial RDBMS choices include
  - DB2 (IBM)
  - Oracle
  - SQL Server (Microsoft)
  - Teradata
- Open source RDBMS options include
  - MySQL
  - PostgreSQL
- And for so-called "Big Data", we also have
  - Apache Hive (on Hadoop) + newer wannabees

Example Instances

- We'll use these instances of our usual Sailors and Reserves relations in our examples.

R1

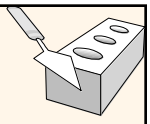
sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

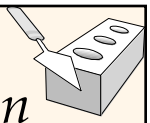
sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0



Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*. (×)
  - Discard resulting tuples if they fail *qualifications*. (σ)
  - Project out attributes that are not in *target-list*. (π)
  - If DISTINCT is specified, eliminate duplicate rows. (δ)

- This strategy is probably the **least** efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.



Example of Conceptual Evaluation

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid=103
```

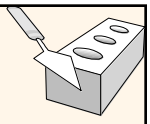
一开始就是将两个table合并了

← *using S1*

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

然后仔找合并后两边sid相同的留下来，然后是挑选

A Note on Range Variables



Sailors(sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

- Named variables "needed" when the same relation appears **twice** (or more) in the FROM clause. Previous query can also be written as:

```
SELECT  sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND bid=103
```

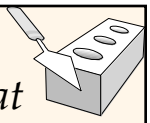
后面的叫name variable

OR

```
SELECT  sname
FROM    Sailors, Reserves
WHERE   Sailors.sid=Reserves.sid
        AND bid=103
```

It is good style, however, to use range variables ***always***!

Find sailors who've reserved at least one boat

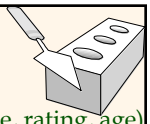


Sailors(sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

```
SELECT  S.sid
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid
```

- Would adding DISTINCT to this query make a difference? (With our example data? And what about other possible data?)
- What is the effect of replacing *S.sid* by *S.sname* in the SELECT clause? Would adding DISTINCT to *this* variant of the query make a difference?

Expressions and Strings

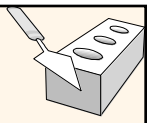


Sailors(sid, sname, rating, age)

```
SELECT  S.sname, S.age, S.age/7.0 AS dogyears
FROM  Sailors S
WHERE  S.sname LIKE 'B_%B'
```

- Illustrates use of arithmetic expressions and string pattern matching: *Find triples (names and ages of sailors plus a field defined by an expression) for sailors whose names begin and end with B and contain at least three characters.*
- AS provides a way to (re)name fields in result.
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters. (See MySQL docs for more info...)

Example Data in MySQL



Sailors

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	4	25.5
95	Bob	3	63.5
101	Joan	3	NULL
107	Johan...	NULL	35.0

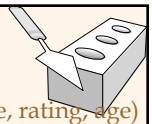
Reserves

sid	bid	date
22	101	1998-10-10
22	102	1998-10-10
22	103	1998-10-08
22	104	1998-10-07
31	102	1998-11-10
31	103	1998-11-06
31	104	1998-11-12
64	101	1998-09-05
64	102	1998-09-08
74	103	1998-09-08
NULL	103	1998-09-09
1	NULL	2001-01-11
1	NULL	2002-02-02

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Find sid's of sailors who've reserved a red or a green boat



Sailors(sid, sname, rating, age)
Reserves(sid, bid, day)
Boats(bid, bname, color)

- If we replace OR by AND in this first version, what do we get?
- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).
- Also available: EXCEPT (What would we get if we replaced UNION by EXCEPT?)

[Note: MySQL vs. RelaX – and why?]

```
SELECT DISTINCT S.sid   找不同
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
 AND (B.color='red' OR B.color='green')
```

```
(SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
      AND B.color='red')
UNION
(SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
      AND B.color='green')
```