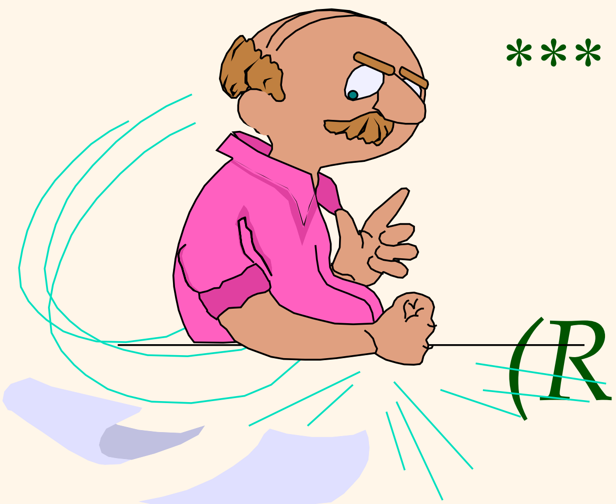
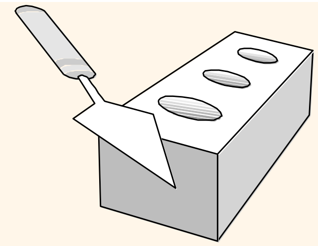
Introduction to Data Management

**** The "Flipped" Edition ****

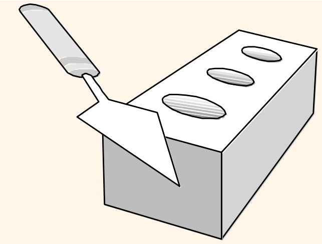


Lecture #11
(Relational Languages III)

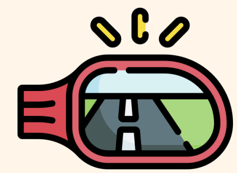
Instructor: Mike Carey
mjcarey@ics.uci.edu



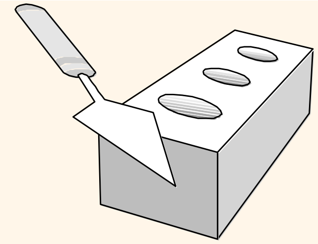
Today's Notices



- *SWOOSH* HW series status
  - HW1 is graded, HW2 grading is underway
  - HW3 is open for business! (Due Wed at 6 PM)



- Midterm 1 info:
  - See Piazza for the full set of rules (!)
    - In-person exam (using Gradescope)
    - Laptops needed (open ***only*** to Gradescope)
    - No cell phones or other devices permitted
    - No textbook access allowed
    - 2-sided hardcopy cheat sheet highly recommended



Pre-Midterm Time Check!

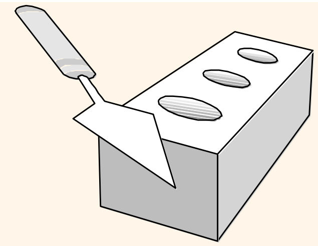
Topic Coverage and Exam Schedule

Syllabus

Topic	Reading (Required!)
Databases and DB Systems	Ch. 1
Entity-Relationship (E-R) Data Model	Ch. 6.1-6.5, 6.8-6.9
Relational Data Model	Ch. 2.1-2.4, 3.1-3.2
E-R to Relational Translation	Ch. 6.6-6.7
Relational Design Theory	Ch. 7.1-7.4.2
Midterm Exam 1	***Fri, Oct 22*** (during lecture time)
Relational Algebra	Ch. 2.5-2.7
Relational Calculus	Wikipedia: Tuple relational calculus
SQL Basics (SPJ and Nested Queries)	Ch. 3.3-3.5
SQL Analytics: Aggregation, Nulls, and Outer Joins	Ch. 3.6-3.9, 4.1
Advanced SQL: Constraints, Triggers, Views, and Security	Ch. 4.2, 4.4-4.5, 4.7
Midterm Exam 2	***Mon, Nov 15*** (during lecture time)
Storage	Ch. 12.1-12.4, 12.6-12.7
Indexing	Ch. 14.1-14.4, 14.5
Physical DB Design	Ch. 14.6-14.7, 15.1-15.3, 15.5.3
Semistructured Data Management (*a.k.a.* NoSQL)	Ch. 8.1, AsterixDB SQL++ Primer, Couchbase SQL++ Book
Data Science 1: Advanced SQL Analytics	Ch. 5.5, 11.3
Data Science 2: Notebooks, Dataframes, and Python/Pandas	Lecture notes and Jupyter notebook
Basics of Transactions	Ch. 4.3, Ch. 17
Endterm Exam	***Fri, Dec 3*** (during lecture time)

Midterm Exam 1

Time: Fri, Oct 22, Lecture Time
Place: SSLH 100



TRC Formulas

- *Atomic formula:*
  - $r \in R$, or $r \notin R$, or $r.a$ *op* $s.b$, or $r.a$ *op* constant
  - *op* is one of $<, >, \leq, \geq, \neq, =$
- *Formula:*
  - an atomic formula, or
  - $\neg P$, $P \wedge Q$, $P \vee Q$, where P and Q are formulas, or
  - $\exists r \in R\ (P(r))$, where variable r is *free* in $P(\ldots)$, or
  - $\forall r \in R\ (P(r))$, where variable r is *free* in $P(\ldots)$, or
  - $P \Rightarrow Q$ (pronounced “implies”, equivalent to $(\neg P) \vee Q)$)

$\boxed{\exists \nexists \forall \in \notin \neg \wedge \vee \Rightarrow = \neq < > \leq \geq}$

Free and Bound Variables

- The use of a quantifier such as $\exists t \in T$ or $\forall t \in T$ in a formula is said to *bind* t.
  - A variable that is not bound is free.
- Now let us revisit the definition of a TRC query:
  - { t(a1, a2, …) **|** P(t) }
- There is an important restriction: the variable t that appears to the left of the **|** ("such that") symbol must be the *only* free variable in the formula P(...).
- Let's look at some examples...

Find sailors with a rating above 7

Sailors(sid, sname, rating, age)   Reserves(sid, bid, date)

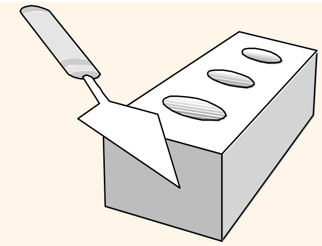
Boats(bid, bname, color)

$$\{\, s \mid s \in Sailors \wedge s.rating > 7 \,\}$$

- This is equivalent to the more general form:

$$\{\, t(id, nm, rtg, age) \mid \exists s \in Sailors\ (\, t.id = s.sid \wedge t.nm = s.sname \wedge t.rtg = s.rating \wedge t.age = s.age \wedge s.rating > 7 \,)\,\}$$

*(**Q:** See how each one specifies the answer's schema and values…?*
*Note that the second query's result schema is **different** as we've specified it.*
The Wikipedia article uses a very similar notation: {t:{id,nm,rtg,age} | … }).



Find ids of sailors who are older than 30.0 or who have a rating under 8 and are named "Horatio"

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, date)

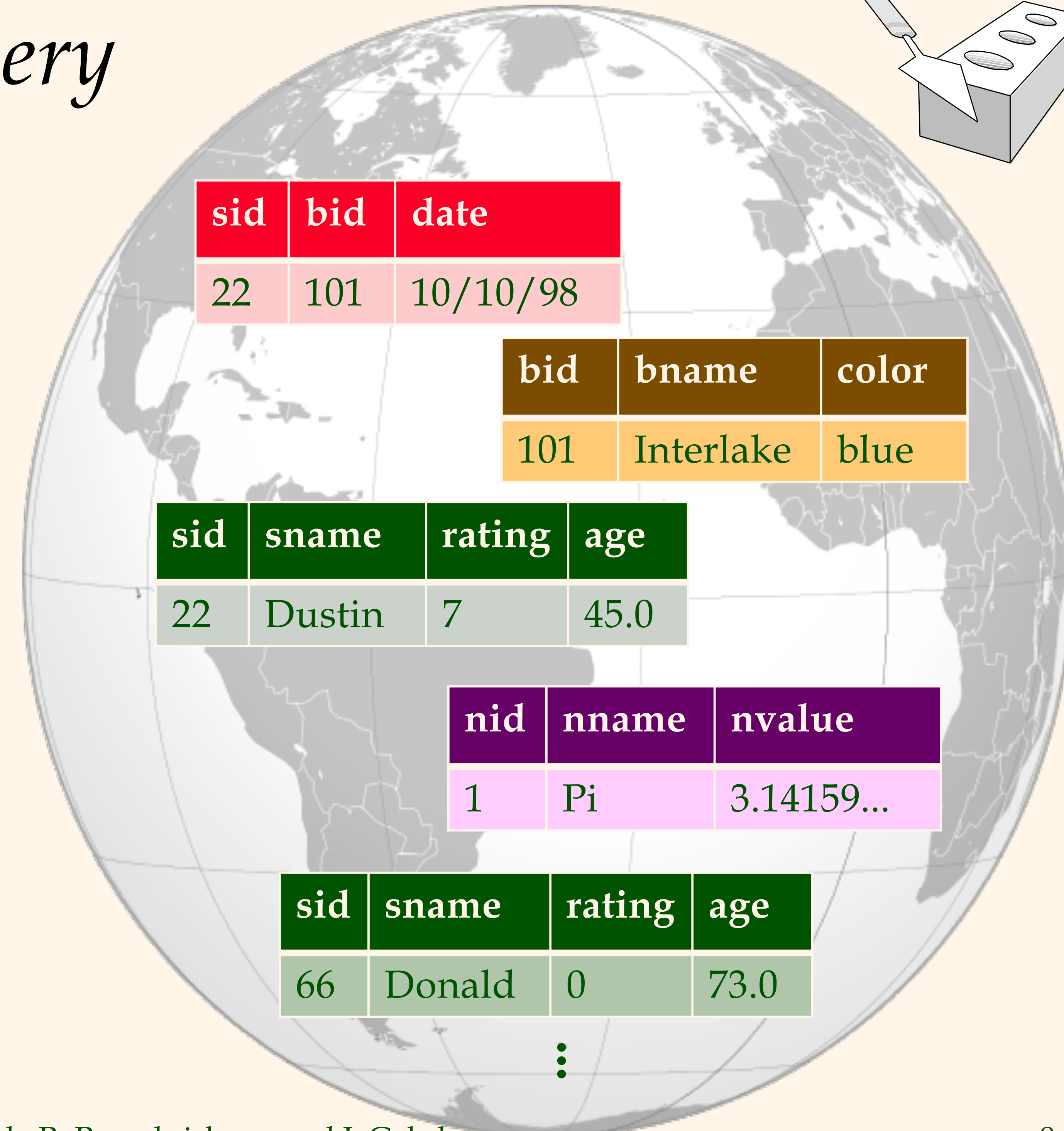
$$\{\ t(sid)\ \mid\ \exists s \in Sailors\ (\ (s.age > 30.0 \\ \lor (s.rating < 8 \land s.sname = \text{"Horatio"})) \\ \land\ t.sid = s.sid\ )\ \}$$

- Things to notice:
  - Again, how result schema and values are specified
  - Use of Boolean formula to specify the query constraints
  - *Highly **declarative** nature of this form of query language!*

<u>Ex:</u> TRC Query Semantics

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	73.0

⋮

<u>Ex:</u> 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	73.0

⋮

Find names of sailors who've reserved a red boat

Sailors(sid, sname, rating, age)

Reserves(sid, bid, date)

Boats(bid, bname, color)

$$
\begin{aligned}
\{\, t(sname) \mid\ & \exists s \in Sailors\ (t.sname = s.sname \land \\
& \quad \exists r \in Reserves\ (r.sid = s.sid \land \\
& \quad\quad \exists b \in Boats\ (b.bid = r.bid \land b.color = \text{'red'})))\, \}
\end{aligned}
$$

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

$\exists \nexists \forall \in \notin\ \neg\ \land \lor \Rightarrow = \neq < > \leq \geq$

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.,* $s \mid \neg (s \in Sailors)$
- 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) that can express every query that is expressible in the relational algebra/(safe) calculus.

Ex: *TRC Query Safety*

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

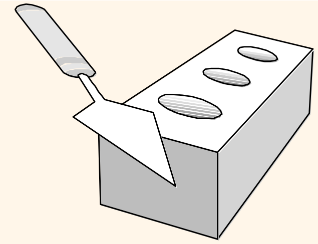
sid	sname	rating	age
22	Dustin	7	45.0

nid	nname	nvalue
1	Pi	3.14159...

sid	sname	rating	age
66	Donald	0	73.0

⋮

Find ids of sailors who've reserved a red boat <u>and</u> a green boat



Sailors(sid, sname, rating, age) Reserves(sid, bid, date)

Boats(bid, bname, color)

$$
\begin{aligned}
\{\, t(sid) \mid\ & \exists s \in Sailors\ (t.sid = s.sid \wedge \\
& \quad \exists r1 \in Reserves\ (r1.sid = s.sid \wedge \\
& \quad\quad \exists b1 \in Boats\ (b1.bid = r1.bid \wedge b1.color = \text{'red'})) \wedge \\
& \quad \exists r2 \in Reserves\ (r2.sid = s.sid \wedge \\
& \quad\quad \exists b2 \in Boats\ (b2.bid = r2.bid \wedge b2.color = \text{'green'})))\,\}
\end{aligned}
$$

- 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	

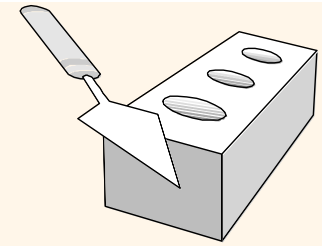
Find the names of sailors who've reserved all boats

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

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

(For all boats b, sailor s has a reservation r for it)

- 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, date)

Boats(bid, bname, color)

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

(For all boats b, if b is an 'Interlake' then sailor s has a reservation r for it)

- Or, if you prefer:

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

(For all boats b, either b is not an 'Interlake' or sailor s has a reservation r for it)

Find the names of sailors who've reserved all <u>Interlake</u> boats (Gradescope-friendly version)

Sailors(sid, sname, rating, age) Reserves(sid, bid, date)

Boats(bid, bname, color)

{ t(sname) | ***some*** s ***in*** Sailors
  (t.sname = s.sname ***and***
    all b ***in*** Boats (b.bname = 'Interlake' ***implies*** (***some*** r ***in*** Reserves
      (r.sid = s.sid ***and*** b.bid = r.bid)))) }

(For all boats b, <u>if</u> b is an 'Interlake' <u>then</u> sailor s has a reservation r for it)

- Or, if you prefer:

{ t(sname) | ***some*** s ***in*** Sailors
  (t.sname = s.sname ***and***
    all b ***in*** Boats (b.bname != 'Interlake' ***or*** (***some*** r ***in*** Reserves
      (r.sid = s.sid ***and*** b.bid = r.bid)))) }

(For all boats b, either b is <u>not</u> an 'Interlake' <u>or</u> sailor s has a reservation r for it)

Find the names of sailors who've reserved all <u>Interlake</u> boats (📱 –friendly version ☺)

Sailors(sid, sname, rating, age) Reserves(sid, bid, date)

Boats(bid, bname, color)

{ t(sname) | 🙂 Sailors s

(t.sname = s.sname 🐜

😁 Boats b (b.bname = 'Interlake' 👉 (🙂 Reserves r

(r.sid = s.sid 🐜 b.bid = r.bid)))) }

(For all boats b, <u>if</u> b is an 'Interlake' <u>then</u> sailor s has a reservation r for it)

- Or, if you prefer:

{ t(sname) | 🙂 Sailors s

(t.sname = s.sname 🐜

😁 Boats b (b.bname 👎= 'Interlake' 🚣 (🙂 Reserves r

(r.sid = s.sid 🐜 b.bid = r.bid)))) }

(For all boats b, either b is <u>not</u> an 'Interlake' <u>or</u> sailor s has a reservation r for it)

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, not covered here).