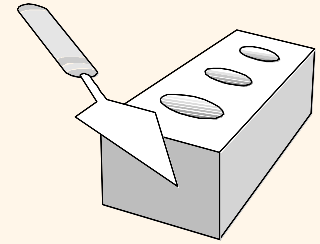
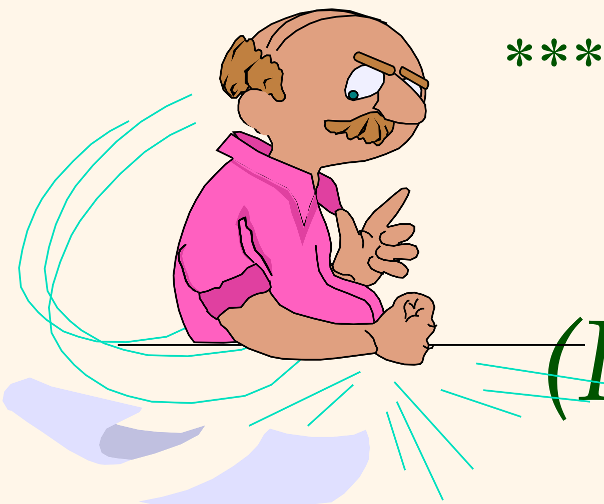


Introduction to Data Management

*** The “Flipped” Edition ***



Lecture #9
(Relational Languages I)

Instructor: Mike Carey
mjcarey@ics.uci.edu

Today's Notices

- Keep one eye on the wiki page...
  - http://www.ics.uci.edu/~cs122a/
- ... and the other eye on Piazza Q&A!
  - piazza.com/uci/fall2021/cs122aeecs116
- HW #2 should be done!
  - Due "today" at 6PM to be on time
  - Then comes the 24-hour grace period (-10%)
- Today's lecture plan:
  - Relational languages – ***the next frontier***…
  - ***Note*:** Today's material won't be on Midterm 1! (☺)

Quick Time Check...

Topic Coverage and Exam Schedule

Syllabus

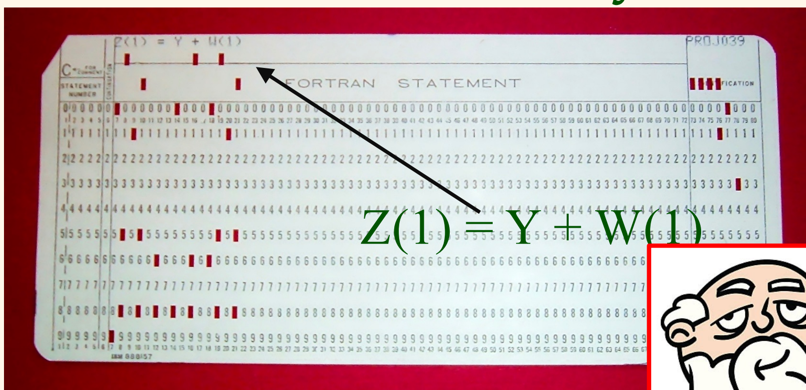
Topic	Reading (Required!)
Databases and DB Systems	Ch. 1
Entity-Relationship (E-R) Data Model	Ch. 6.1-6.5, 6.8-6.9
Relational Data Model	Ch. 2.1-2.4, 3.1-3.2
E-R to Relational Translation	Ch. 6.6-6.7
Relational Design Theory	Ch. 7.1-7.4.2
Midterm Exam 1	***Fri, Oct 22*** (during lecture time)
Relational 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

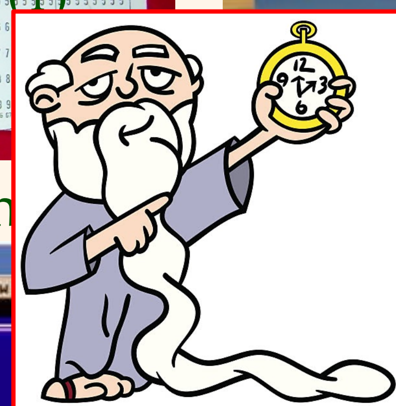
First: A Word on Learning…

- ”When ****I**** was your age…” (☺)

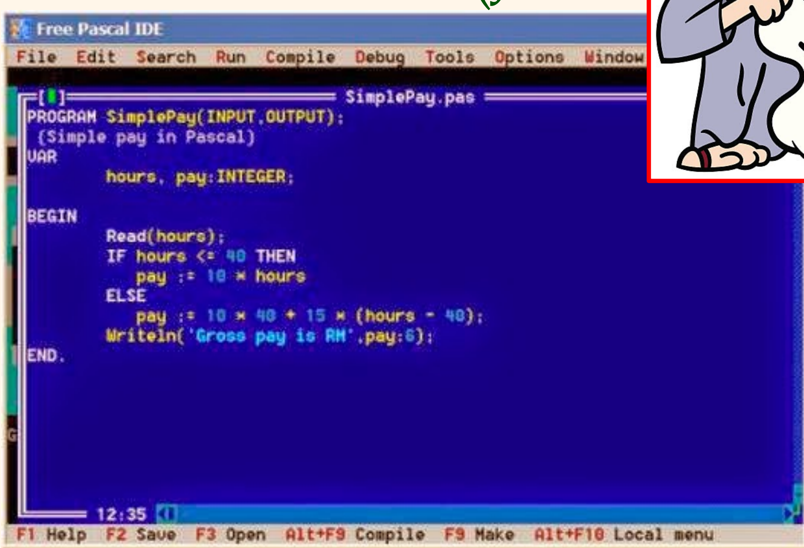


s, I

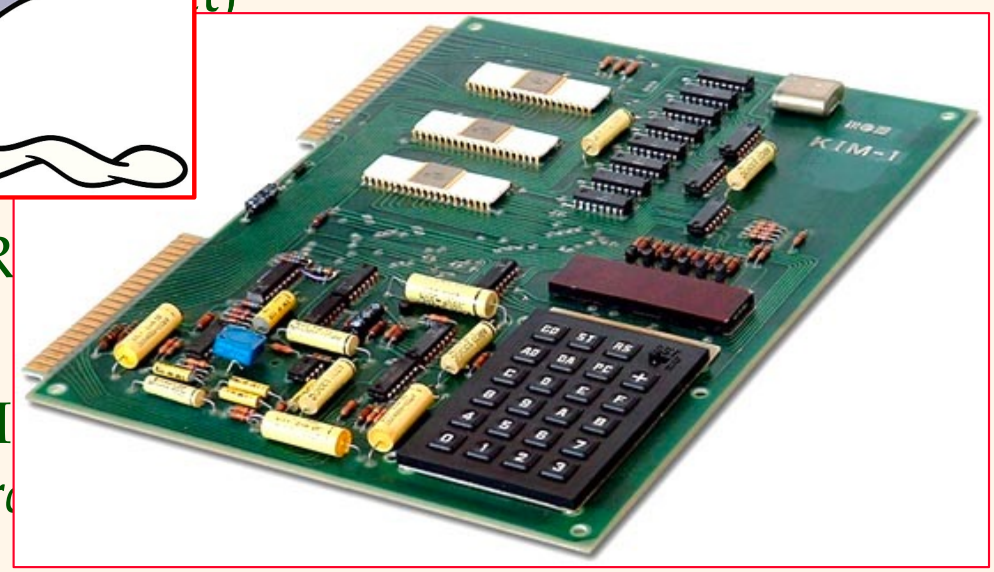
Lis



  - No web! (Just th… …et)



R



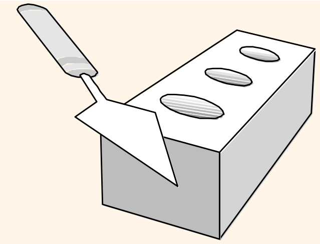
: I

ar

First: A Word on Learning...

- "When ***I*** was your age…" (☺)
  - 8-bit μprocessors, LEDs, hex keypad, 16-bit for its address space, …
  - FORTRAN, Pascal, C, Lisp, Snobol, APL, Quel …
  - Unix first appeared, Unix/INGRES DB 64K process design point, and 1MB of memory was amazing!
  - No web! (Just the early ArpaNet)
- Fast forward 35 years…
  - Your cell phones dwarf our ~1980 computing platforms
  - Python, Java, Go, C++, Ruby, SQL, SQL++, …
  - Twitter… (😐)
  - ***WHAT THIS MEANS:*** It's critical to *learn* how to read and how to *learn*, how to *search* for info/resources, etc…!

On to Relational Query Languages!



- *Query languages*: Allow manipulation and *retrieval* of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.
- Query Languages ≠ programming languages!
  - QLs not expected to be "Turing complete."
  - QLs not intended to be used for complex calculations.
  - ***QLs support easy, efficient access to large data sets.***

Formal Relational Query Languages

- Two mathematical Query Languages form the basis for "real" languages (e.g., SQL), and for their implementation:
  - *Relational Algebra*: More operational, very useful for representing execution plans.
  - *Relational Calculus*: Lets users describe what they want, rather than how to compute it. (Non-operational, or *declarative*.)

Preliminaries

- A query is applied to *relation instances*, and the result of a query is ***also*** a relation instance.
  - *Schemas* of input relations for a query are fixed (but query will run regardless of instance!)
  - The schema for the *result* of a given query is also fixed! Determined by applying the definitions of the query language's constructs.
- Positional vs. named-field notation:
  - Positional notation easier for formal definitions, but named-field notation far more readable.
  - Both used in SQL (but try to avoid positional stuff!)

Example Instances

- "Sailors" and "Reserves" relations for our examples.
- We'll use positional or named field notation and assume that names of fields in query results are "inherited" from names of fields in query input relations (when possible).

R1

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

S1

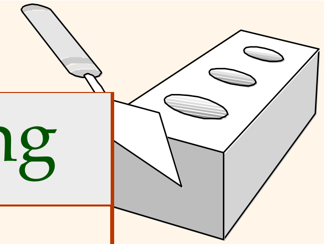
sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

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

Relational Algebra

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Omits unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and in reln. 2.
- Additional operations:
  - Intersection, *join*, division, renaming: Not essential, but (very!) useful. (I.e., don't add expressive power, but…)
- Since each operation returns a relation, operations can be ***composed!*** (Algebra is "closed".)



Projection

- Removes attributes that are not in *projection list*.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- *Relational* projection operator has to eliminate *duplicates*! (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (**Q:** Why not?)

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$$\pi_{sname,rating}(S2)$$

age
35.0
55.5

$$\pi_{age}(S2)$$

Selection

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$$\sigma_{rating>8}(S2)$$

- Selects rows that satisfy a *selection condition*.
- No duplicates in result! (*Q*: Why not?)
- *Schema* of result identical to schema of its (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (This is *operator composition*.)

sname	rating
yuppy	9
rusty	10

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - "Corresponding" fields are of the same type.
- What is the *schema* of result?

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

Q: Any issues w/duplicates?

Cross-Product

- S1 **x** R1: Each S1 row is paired with each R1 row.
- *Result schema* has one field per field of S1 and R1, with field names "inherited" if possible.
  - *Conflict*: Both S1 and R1 have a field called *sid*.

(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

Result relation name

Attribute renaming list

Source expression E (anything!)

- <u>*Renaming operator*</u>: $\rho\,(C(1 \rightarrow sid1, 5 \rightarrow sid2),\ S1 \times R1)$

Renaming (Messy but Needed)

- *Conflict*: S1 and R1 both had *sid* fields, giving:

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

- Several renaming notations available:

$$\rho\,(S1R1(1 \rightarrow sid1),\ S1 \times R1)$$

Positional renaming

$$\rho\,(TempS1(sid \rightarrow sid1),\ S1)$$

$$TempS1 \times R1$$

Name-based renaming

$$(\pi_{sid \rightarrow sid1, sname, rating, age}(S1)) \times R1$$

Generalized projection

(I like this one best! ☺)

$$\rho_{TempS1(sid1,\ sname,\ rating,\ age,\ sid,\ bid,\ day)}\,(S1 \times R1)$$

SKS book's renaming

Joins

- *Condition Join*: $R \bowtie_c S = \sigma_c (R \times S)$

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

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, so might be able to compute it more efficiently
- Sometimes (often!) called a *theta-join*.

More Joins

- *Equi-Join*: A special case of condition join where the condition *c* contains only ***equalities***.

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$$S1 \bowtie_{S1.sid = R1.sid} R1$$

- *Result schema* is similar to cross-product, but only one copy of fields for which equality is specified.
- *Natural Join*: An equijoin on ***all*** commonly named fields (denoted simply by $\bowtie$).

Division

- Like join, division is not a primitive operator, but is extremely useful for expressing queries, such as:

  *Find sailors who have reserved **all** boats.*

- Let A have 2 fields, x and y, while B has one field y, so we have relations A(x,y) and B(y):
  - ***A/B* contains the *x* tuples (e.g., sailors) such that for *every* *y* tuple (e.g., boat) in *B*, there is an *xy* tuple in *A*.**
  - *Or*: If the set of y values (boats) associated with an x value (sailor) in A contains all y values in B, the x value is in A/B.
- In general, x and y can be any lists of fields; y is the list of fields in B, and $x \cup y$ is the list of fields of A.

Examples of Division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

sno
s1
s2
s3
s4

A/B1

pno
p2
p4

B2

sno
s1
s4

A/B2

pno
p1
p2
p4

B3

sno
s1

A/B3

Expressing A/B Using Basic Operators *(Advanced Topic – Just FYI ☺)*

- Division not an essential op; just a useful shorthand.

  (Also true of joins, but joins are so common and important that relational database systems implement joins specially.)

- *Idea*: For *A(x,y)/B(y)*, compute all *x* values that are not "disqualified" by some *y* value in *B*.
  - *x* value is *disqualified* if by attaching a y value from B, we obtain an xy tuple that *does not appear* in *A*.

Disqualified *x* values (*D*): $\pi_x((\pi_x(A) \times B) - A)$

A/B: $\pi_x(A) - D$

All x's in A

All x's in A combined with all y's in B

Tuples in the cross product but not in A

To Be Continued….

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

Reserves

sid	bid	date
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
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
103	Clipper	green
104	Marine	red