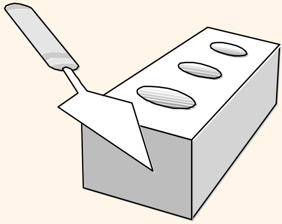
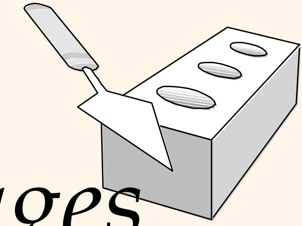
Relational Algebra & Calculus



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 implementation:
  - *Relational Algebra*: More operational (procedural), very useful for representing execution plans.
  - *Relational Calculus*: Lets users describe what they want, rather than how to compute it: Non-operational, *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.
  - The schema for the *result* of a given query is also fixed! - determined by definition of query language constructs.
- Positional vs. named-field notation:
  - Positional notation easier for formal definitions, named-field notation more readable.
  - Both used in SQL

Example Instances

R1

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

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

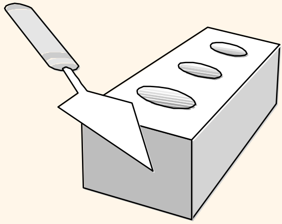
S1

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

S2

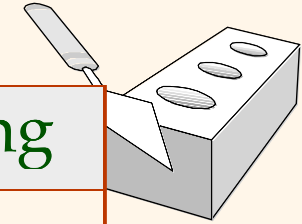
sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Relational Algebra



- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes 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.
- Since each operation returns a relation, operations can be *composed*: algebra is "closed".

Projection



- Deletes 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 input relation.
- Projection operator has to eliminate *duplicates*! Why?
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it (by DISTINCT). 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

- Selects rows that satisfy *selection condition*.
- No duplicates in result! Why?
- *Schema* of result identical to schema of input relation.
- What is Operator composition?
- Selection is distributive over binary operators
- Selection is commutative

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

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

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 have 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

$S1 \cap S2$

Cross-Product (Cartesian Product)

- Each row of S1 is paired with each row of R1.
- *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

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

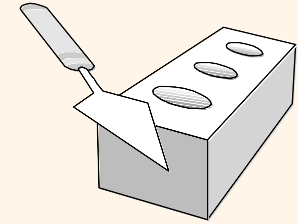
Joins: used to combine relations

- *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, might be able to compute more efficiently
- Sometimes called a *theta-join*.



Join

- *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

$$S1 \bowtie_{sid} R1$$

- *Result schema* similar to cross-product, but only one copy of fields for which equality is specified.
- *Natural Join*: Equijoin on *all* common fields.

Properties of join

- Selecting power: can join be used for selection?
- Is join commutative? $S1 \bowtie R1 \quad = \quad R1 \bowtie S1$?
- Is join associative? $S1 \bowtie (R1 \bowtie C1) = (S1 \bowtie R1) \bowtie C1$?
- Join and projection perform complementary functions
- Lossless and lossy decomposition

Division

- Not supported as a primitive operator, but useful for expressing queries like:

  Find sailors who have reserved ***<u>all</u>*** *boats.*

- Let A have 2 fields, x and y; B have only field y:
  - $A/B = \left\{ \langle x \rangle \mid \exists \langle x, y \rangle \in A \;\; \forall \langle y \rangle \in B \right\}$
  - i.e., ***A/B*** **contains all** ***x*** **tuples (sailors) such that for** ***<u>every</u>*** ***y*** **tuple (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

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

Example of Division

- Find all customers who have an account at all branches located in Chville
  - Branch (bname, assets, bcity)
  - Account (bname, acct#, cname, balance)

Example of Division

R1: Find all branches in Chville

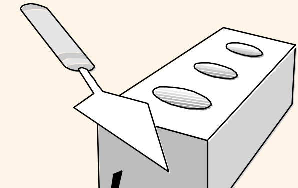
R2: Find (bname, cname) pair from Account

R3: Customers in r2 with every branch name in r1

$$r1 = \pi_{bname}(\sigma_{bcity='Chville'} Branch)$$

$$r2 = \pi_{bname,cname}(Account)$$

$$r3 = r2 \div r1$$



Expressing A/B Using Basic Operators

- Division is not essential op; just a useful shorthand.
  - Also true of joins, but joins are so common that systems implement joins specially.
- *Idea*: For *A/B*, compute all *x* values that are not \`disqualified' by some *y* value in *B*.
  - *x* value is *disqualified* if by attaching *y* value from *B*, we obtain an *xy* tuple that is not in *A*.

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

A/B: $\pi_X(A)$ $-$ all disqualified tuples

Exercises

Given relational schema:

Sailors (sid, sname, rating, age)

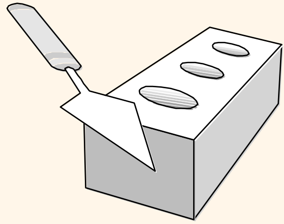
Reservation (sid, bid,_date)

Boats (bid, bname, color)

1) Find names of sailors who've reserved boat #103
2) Find names of sailors who've reserved a red boat
3) Find sailors who've reserved a red or a green boat
4) Find sailors who've reserved a red and a green boat
5) Find the names of sailors who've reserved all boats

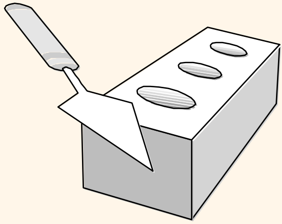
Summary of Relational Algebra

- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version.



Relational Calculus

- Comes in two flavors: *Tuple relational calculus* (TRC) and *Domain relational calculus* (DRC).
- Calculus has *variables, constants, comparison ops, logical connectives* and *quantifiers*.
  - *TRC*: Variables range over (i.e., get bound to) *tuples*.
  - *DRC*: Variables range over *domain elements* (= field values).
  - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called *formulas*. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to *true*.



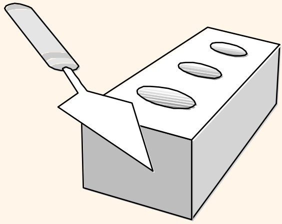
Domain Relational Calculus

- *Query* has the form:

$$\left\{\left\langle x1, x2, \ldots, xn \right\rangle \mid p\left(\left\langle x1, x2, \ldots, xn \right\rangle\right)\right\}$$

- *Answer* includes all tuples $\left\langle x1, x2, \ldots, xn \right\rangle$ that make the *formula* $p\left(\left\langle x1, x2, \ldots, xn \right\rangle\right)$ be *true*.

- *Formula* is recursively defined, starting with simple *atomic formulas* (getting tuples from relations or making comparisons of values), and building bigger and better formulas using the *logical connectives*.



DRC Formulas

- *Atomic formula:*
  - $\langle x1, x2, ..., xn \rangle \in Rname$, or X *op* Y, or X *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
  - $\exists X\,(p(X))$, where X is *a domain variable* or
  - $\forall X\,(p(X))$, where X is *a domain variable.*
- The use of quantifiers $\exists X$ and $\forall X$ is said to *bind* X.

Free and Bound Variables

- The use of quantifiers $\exists X$ and $\forall X$ in a formula is said to *bind* X.
  - A variable that is not bound is free.
- Let us revisit the definition of a query:

$$\left\{ \left\langle x1, x2, \ldots, xn \right\rangle \mid p\left(\left\langle x1, x2, \ldots, xn \right\rangle \right) \right\}$$

- There is an important restriction: the variables x1, ..., xn that appear to the left of \`|' must be the *only* free variables in the formula p(...).

Find all sailors with a rating above 7

$$\left\{ \langle I,N,T,A \rangle \mid \langle I,N,T,A \rangle \in Sailors \wedge T > 7 \right\}$$

- The condition $\langle I,N,T,A \rangle \in Sailors$ ensures that the domain variables *I, N, T* and *A* are bound to fields of the same Sailors tuple.
- The term $\langle I,N,T,A \rangle$ to the left of \`|' (which should be read as *such that*) says that every tuple $\langle I,N,T,A \rangle$ that satisfies *T*>7 is in the answer.
- Modify this query to answer:
  - Find sailors who are older than 18 or have a rating under 9, and are called 'Joe'.

Find sailors rated > 7 who have reserved boat #103

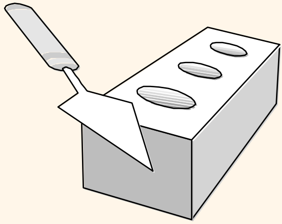
$$\left\{\left\langle I,N,T,A\right\rangle \mid \left\langle I,N,T,A\right\rangle \in Sailors \wedge T>7 \wedge \right.$$

$$\left. \exists\; Ir,Br,D \left(\left\langle Ir,Br,D\right\rangle \in \mathrm{Re}serves \wedge Ir=I \wedge Br=103\right)\right\}$$

- We have used $\exists\; Ir, Br, D\ (\ldots)$ as a shorthand for $\exists\; Ir\left(\exists\; Br\left(\exists\; D(\ldots)\right)\right)$

- Note the use of $\exists$ to find a tuple in Reserves that \`joins with' the Sailors tuple under consideration.

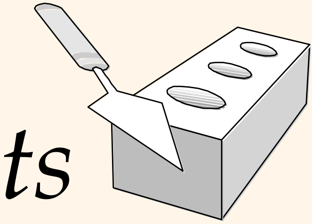
Find sailors rated > 7 who've reserved a red boat



$$\left\{\left\langle I,N,T,A\right\rangle \mid \left\langle I,N,T,A\right\rangle \in Sailors \wedge T>7 \wedge \right.$$
$$\exists\ Ir,Br,D\left(\left\langle Ir,Br,D\right\rangle \in \mathrm{Re}serves \wedge Ir=I \wedge\right.$$
$$\left.\left.\left.\exists\ B,BN,C\left(\left\langle B,BN,C\right\rangle \in Boats \wedge B=Br \wedge C='red'\right)\right)\right\}$$

- Observe how the parentheses control the scope of each quantifier's binding.
- This may look cumbersome, but with a good user interface, it could be intuitive. (MS Access, QBE)

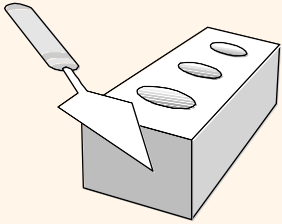
Find sailors who've reserved all boats



$$\Big\{ \langle I,N,T,A \rangle \mid \langle I,N,T,A \rangle \in Sailors \wedge$$
$$\forall\ B,BN,C \Big(\neg \big(\langle B,BN,C \rangle \in Boats \big) \vee$$
$$\Big(\exists\ Ir,Br,D \ \big(\langle Ir,Br,D \rangle \in \mathrm{Re}serves \wedge I = Ir \wedge Br = B \big) \Big) \Big) \Big\}$$

- Find all sailors *I* such that for each 3-tuple $\langle B,BN,C \rangle$ either it is not a tuple in Boats or there is a tuple in Reserves showing that sailor *I* has reserved it.

Find sailors who've reserved all boats (again!)



$$\left\{ \langle I,N,T,A \rangle \mid \langle I,N,T,A \rangle \in Sailors \wedge \right.$$
$$\forall \langle B,BN,C \rangle \in Boats$$
$$\left. \left(\exists \langle Ir,Br,D \rangle \in \mathrm{Re}serves \left(I=Ir \wedge Br=B \right) \right) \right\}$$

- Simpler notation, same query. (Much clearer!)
- To find sailors who've reserved all red boats:

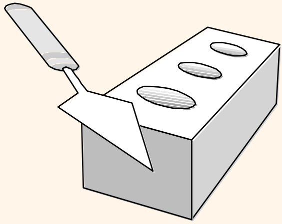
$$\ldots\ldots \left(C \neq 'red' \vee \exists \langle Ir,Br,D \rangle \in \mathrm{Re}serves \left(I=Ir \wedge Br=B \right) \right) \Big\}$$

Any other way to specify it? Equivalence in logic

Unsafe Queries, 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., $\left\{ S \mid \neg\left(S \in Sailors \right) \right\}$
- 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.

Exercise of tuple calculus



Given relational schema:

Sailors (sid, sname, rating, age)

Reservation (sid, bid, date)

Boats (bid, bname, color)

1) Find all sialors with a rating above 7.
2) Find the names and ages of sailors with a rating above 7
3) Find the sailor name, boal id, and reservation date for each reservation
4) Find the names of the sailors who reserved all boats.

Summary of Relational Calculus

- Relational calculus is non-operational, and users define queries in terms of what they want, not in terms of how to compute it. (Declarativeness.)
- Algebra and safe calculus have same expressive power, leading to the notion of relational completeness.