CompSci 516
Data Intensive Computing Systems

Lecture 3
Relational Algebra
and
Relational Calculus

Instructor: Sudeepa Roy

Announcement

- Homework 1 – Part 1 has been posted
  - You need it for Part 2
  - Part 2 will be posted soon
  - Each homework (all parts together) is due after 14 days the last part is posted
- To review background material
  - See CompSci 316 : e.g. http://sites.duke.edu/compsci316_01_f2015/
- Send me emails for feedback or suggestions!

Today's topics

- Relational Algebra (RA) and Relational Calculus (RC)
  - Normalization (intro, in detail in the next lecture)
- Reading material
  - [RG] Chapter 4 (RA, RC)
  - [GUW] Chapters 2.4, 5.1, 5.2

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.

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 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, 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
    - query will run regardless of instance
  - 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

Example Schema and Instances

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

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

R1

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

Relational Algebra

- Takes one or more relations as input, and produces a relation as output
  - operator
  - operand
  - semantic
  - so an algebra!
- Since each operation returns a relation, operations can be *composed*
  - Algebra is “closed”

Relational Algebra

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* (x) Allows us to combine two relations.
  - *Set-difference* (-) Tuples in reln. 1, but not in reln. 2.
  - *Union* (∪) Tuples in reln. 1 or in reln. 2.
- Additional operations:
  - Intersection (∩)
  - *join* ⋈
  - division(/)
  - renaming (ρ)
  - Not essential, but (very) useful.

Projection

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

- 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 (only) 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 (performance)

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 (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation
  - (*Operator composition*)

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

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

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

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - `Corresponding' fields have the same type
  - same schema as the inputs

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

$S1 \cup S2$

Union, Intersection, Set-Difference

S1

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

S2

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

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

- 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

$$R \bowtie_c S = \sigma_c (R \times S)$$

(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
- Recall *theta-, equi-, natural-join*.

Division

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

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

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

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:

all disqualified tuples

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

Find names of sailors who've reserved boat #103

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

Find names of sailors who've reserved boat #103

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

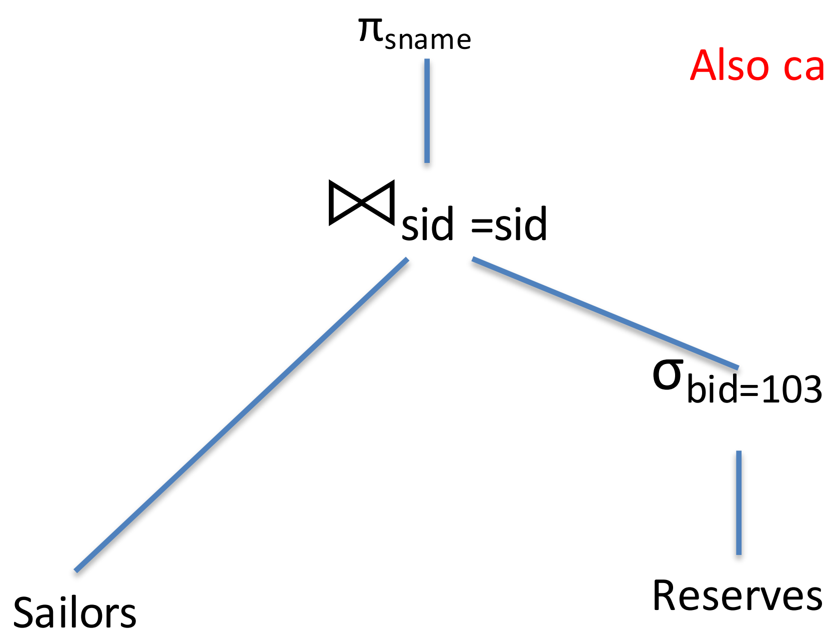
- Solution 1: $\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie Sailors)$
- Solution 2: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie Sailors))$

Expressing an RA expression as a Tree

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)



Also called a logical query plan

$$\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie Sailors)$$

Find sailors who've reserved a red or a green boat

Sailors(<u>sid</u>, sname, rating, age)
Boats(<u>bid</u>, bname, color)
Reserves(<u>sid, bid, day</u>)

Use of rename operation

- Can identify all red or green boats, then find sailors who've reserved one of these boats:

$$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'}\ Boats))$$

$$\pi_{sname}(Tempboats \bowtie \text{Re}serves \bowtie Sailors)$$

Can also define Tempboats using union
Try the "AND" version yourself

Find the names of sailors who've reserved all boats

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

- Uses division; schemas of the input relations to / must be carefully chosen:

$$\rho\ (Tempsids, (\pi_{sid,bid} \text{Re}serves) \,/\, (\pi_{bid} Boats))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

- To find sailors who've reserved all 'Interlake' boats:

$$\ldots\ldots \quad / \, \pi_{bid}(\sigma_{bname='Interlake'} Boats)$$

Try yourself

- Obtain an RA expression for each SQL query in Lecture 2
- You can discuss with other students

What about aggregates?

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

- Extended relational algebra
- $\gamma_{\text{age, avg(rating) -> avgr}}$ Sailors
- Also extended to "bag semantic": allow duplicates
  - Take into account cardinality
  - R and S have tuple t resp. m and n times
  - R $\cup$ S has t m+n times
  - R $\cap$ S has t min(m, n) times
  - R – S has t max(0, m-n) times
  - sorting(τ), duplicate removal (δ) operators

Relational Calculus

- RA is procedural
  - $\pi_A(\sigma_{A=a} R)$ and $\sigma_{A=a} (\pi_A R)$ are equivalent but different expressions
- RC
  - non-procedural and declarative
  - describes a set of answers without being explicit about how they should be computed

- TRC (tuple relational calculus)
  - variables take tuples as values
  - we will primarily do TRC
- DRC (domain relational calculus)
  - variables range over field values

TRC: example

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

- Find the name and age of all sailors with a rating above 7

$\{P \mid \exists\ S \in Sailors\ (S.rating > 7 \wedge P.name = S.name \wedge P.age = S.age)\}$

- P is a tuple variable
  - with exactly two fields name and age (schema of the output relation)
  - P.name = S.name $\wedge$ P.age = S.age gives values to the fields of an answer tuple
- Use parentheses, $\forall$ $\exists$ $\vee$ $\wedge$ > < = ≠ etc as necessary
- $\Rightarrow$ is very useful too

TRC: example

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

- Find the names of sailors who have reserved at least two boats

TRC: example

Sailors(<u>sid</u>, sname, rating, age)

Boats(<u>bid</u>, bname, color)

Reserves(<u>sid, bid, day</u>)

- Find the names of sailors who have reserved at least two boats

$\{P \mid \exists\ S \in Sailors\ (\exists\ R1 \in Reserves\ \exists\ R2 \in Reserves \wedge S.sid = R1.sid \wedge S.sid = R2.sid \wedge R1.bid \neq R2.bid \wedge P.name = S.name)\}$

TRC: example

Sailors(<u>sid</u>, sname, rating, age)
Boats(<u>bid</u>, bname, color)
Reserves(<u>sid, bid, day</u>)

- Find the names of sailors who have reserved all boats
- Division operation

TRC: example

Sailors(<u>sid</u>, sname, rating, age)
Boats(<u>bid</u>, bname, color)
Reserves(<u>sid, bid, day</u>)

- Find the names of sailors who have reserved all boats
- Division operation

$\{P \mid \exists S \in Sailors\ \forall B \in Boats\ (\exists R \in Reserves\ (S.sid = R.sid \wedge R.bid = B.bid \wedge P.name = S.name))\}$

TRC: example

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

- Find the names of sailors who have reserved all red boats

How will you change the previous TRC expression?

TRC: example

Sailors(<u>sid</u>, sname, rating, age)
Boats(<u>bid</u>, bname, color)
Reserves(<u>sid, bid, day</u>)

- Find the names of sailors who have reserved all <u>red</u> boats

{P | $\exists$ S ϵ Sailors $\forall$ B ϵ Boats (B.color = 'red' $\Rightarrow$ ($\exists$ R ϵ Reserves (S.sid = R.sid $\wedge$ R.bid = B.bid $\wedge$ P.name = S.name)))}

Recall that A $\Rightarrow$ B is logically equivalent to $\neg$ A $\vee$ B
so $\Rightarrow$ can be avoided, but it is cleaner

DRC: example

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

- Find the name and age of all sailors with a rating above 7

TRC:

$\{P \mid \exists\ S \in Sailors\ (S.rating > 7 \land P.name = S.name \land P.age = S.age)\}$

DRC:

$\{\langle N, A\rangle \mid \exists\ \langle I, N, T, A\rangle \in Sailors \land T > 7\}$

- Variables are now domain variables
- We will use mainly use TRC

Summary

- Three languages for relational db model
  - SQL
  - RA
  - RC
- All have their own purposes
- You should be able to write a query in all three languages and convert from one to another
- However, you have to be careful, not all "valid" expressions in one may be expressed in another
  - $\{S \mid \neg (S \in Sailors)\}$ – infinitely many tuples – an unsafe query
  - More when we do "Datalog", also see Ch. 4.4 in [RG]

Database Normalization

- Only an intro, in detail in the next lecture

What will we learn?

- What goes wrong if we have redundant info in a database?
- Why and how should you refine a schema?
- Functional Dependencies
- Normal Forms

Example

The list of hourly employees in an organization

<u>ssn (S)</u>	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

- key = SSN

Example

The list of hourly employees in an organization

<u>ssn (S)</u>	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

- key = SSN
- Suppose for a given rating, there is only one hourly_wage value
- Functional dependency

  R → W
- Redundancy in the table

Why is redundancy bad?

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

Why is redundancy bad?

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

1. Redundant storage:
   - Some information is stored repeatedly
   - The rating value 8 corresponds to hourly_wage 10, which is stored three times

Why is redundancy bad?

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10 -> 9	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

2. Update anomalies
   - If one copy of data is updated, an inconsistency is created unless all copies are similarly updated
   - Suppose you update the hourly_wage value in the first tuple using UPDATE statement in SQL -- inconsistency

Why is redundancy bad?

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

3. Insertion anomalies:
   - It may not be possible to store certain information unless some other, unrelated info is stored as well
   - We cannot insert a tuple for an employee unless we know the hourly wage for the employee's rating value

Why is redundancy bad?

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

4. Deletion anomalies:
   - It may not be possible delete certain information without losing some other information as well
   - If we delete all tuples with a given rating value (Attishoo, Smiley, Madayan), we lose the association between that rating value and its hourly_wage value

Why is redundancy bad?

Therefore,

- Redundancy arises when the schema forces an association between attributes that is “not natural”
- We want schemas that do not permit redundancy
  - at least identify schemas that allow redundancy to make an informed decision (e.g. for performance reasons)
- Null value may or may not help
  - does not help redundant storage or update anomalies
  - can insert a tuple with null value in the hourly_wage field
  - but cannot record hourly_wage for a rating unless there is such an employee (SSN cannot be null)