CompSci 516
Data Intensive Computing Systems

Lecture 4
Relational Algebra
and
Relational Calculus

Instructor: Sudeepa Roy

Announcements

- HW1 reminder:
  - Due on 09/21 (Thurs), 11:55 pm, no late days
- Project proposal deadline extended:
  - Preliminary idea and team members due by 09/18 (Mon) by email to the instructor
  - Proposal due on sakai by 09/25 (Mon), 11:55 pm
  - One report per group
  - Form your group soon
  - Look at recent research papers/demonstrations in top db conferences for ideas (see the project idea file on sakai)
- Your piazza, sakai, gradiance accounts should be active
  - Occasional Pop up quizzes will start from next week
  - Bring a laptop in class

Today's topics

- Finish NULLs and Views in SQL from Lecture 3
- Relational Algebra (RA) and Relational Calculus (RC)
- 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.

Nulls and Views in SQL

Null Values

- Field values in a tuple are sometimes
  - unknown, e.g., a rating has not been assigned, or
  - inapplicable, e.g., no spouse's name
  - SQL provides a special value null for such situations.

Standard Boolean 2-valued logic

- True = 1, False = 0
- Suppose X = 5
  - (X < 100) AND (X >= 1) is T $\wedge$ T = T
  - (X > 100) OR (X >= 1) is F $\vee$ T = T
  - (X > 100) AND (X >= 1) is F $\wedge$ T = F
  - NOT(X = 5) is ¬T = F

- Intuitively,
  - T = 1, F = 0
  - For V1, V2 $\in$ {1, 0}
  - V1 $\wedge$ V2 = MIN (V1, V2)
  - V1 $\vee$ V2 = MAX(V1, V2)
  - ¬(V1) = 1 – V1

2-valued logic does not work for nulls

- Suppose rating = null, X = 5
- Is rating>8 true or false?
- What about AND, OR and NOT connectives?
  - (rating > 8) AND (X = 5)?
- What if we have such a condition in the WHERE clause?

3-Valued Logic For Null

- TRUE (= 1), FALSE (= 0), UNKNOWN (= 0.5)
  - unknown is treated as 0.5

- Now you can apply rules from 2-valued logic!
  - For V1, V2 $\in$ {1, 0, 0.5}
  - V1 $\wedge$ V2 = MIN (V1, V2)
  - V1 $\vee$ V2 = MAX(V1, V2)
  - $\neg$(V1) = 1 – V1

- Therefore,
  - NOT UNKNOWN = UNKNOWN
  - UNKNOWN OR TRUE = TRUE
  - UNKNOWN AND TRUE = UNKNOWN
  - UNKNOWN AND FALSE = FALSE
  - UNKNOWN OR FALSE = UNKNOWN

New issues for Null

- The presence of null complicates many issues. E.g.:
  - Special operators needed to check if value IS/IS NOT NULL
  - Be careful!
  - "WHERE X = NULL" does not work!
  - Need to write "WHERE X IS NULL"
- Meaning of constructs must be defined carefully
  - e.g., WHERE clause eliminates rows that don't evaluate to true
  - So not only FALSE, but UNKNOWNs are eliminated too
  - very important to remember!
- But NULL allows new operators (e.g. outer joins)
- Arithmetic with NULL
  - all of +, -, *, / return null if any argument is null
- Can force "no nulls" while creating a table
  - sname char(20) NOT NULL
  - primary key is always not null

Aggregates with NULL

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?

Aggregates with NULL

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?
- Ans: 3 for both

Aggregates with NULL

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?
- Ans: 3 for both

sid	sname	rating	age
22	dustin	7	45
31	lubber	null	55
58	rusty	10	35

R2

- What do you get for
- SELECT count(*) from R2?
- SELECT count(rating) from R2?

Aggregates with NULL

sid	sname	rating	age
22	dustin	7	45
31	lubber	8	55
58	rusty	10	35

R1

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?
- Ans: 3 for both

sid	sname	rating	age
22	dustin	7	45
31	lubber	null	55
58	rusty	10	35

R2

- What do you get for
- SELECT count(*) from R2?
- SELECT count(rating) from R2?
- Ans: First 3, then 2

Aggregates with NULL

- COUNT, SUM, AVG, MIN, MAX (with or without DISTINCT)
  - Discards null values first
  - Then applies the aggregate
  - Except count(*)
- If only applied to null values, the result is null

sid	sname	rating	age
22	dustin	7	45
31	lubber	null	55
58	rusty	10	35

R2

- SELECT sum(rating) from R2?
- Ans: 17

sid	sname	rating	age
22	dustin	null	45
31	lubber	null	55
58	rusty	null	35

R3

- SELECT sum(rating) from R3?
- Ans: null

Overview: General Constraints

- Useful when more general ICs than keys are involved
- There are also ASSERTIONS to specify constraints that span across multiple tables
- There are TRIGGERS too : procedure that starts automatically if specified changes occur to the DBMS
  - see additional slides at the end

```
CREATE TABLE Sailors
    ( sid INTEGER,
    sname CHAR(10),
    rating INTEGER,
    age REAL,
    PRIMARY KEY (sid),
    CHECK ( rating >= 1
        AND rating <= 10 )
```

```
CREATE TABLE Reserves
    ( sname CHAR(10),
    bid INTEGER,
    day DATE,
    PRIMARY KEY (bid,day),
    CONSTRAINT noInterlakeRes
    CHECK (`Interlake' <>
        ( SELECT B.bname
        FROM Boats B
        WHERE B.bid=bid)))
```

Views

- A view is just a relation, but we store a definition, rather than a set of tuples

```
CREATE VIEW YoungActiveStudents (name, grade)
    AS SELECT S.name, E.grade
    FROM Students S, Enrolled E
    WHERE S.sid = E.sid and S.age<21
```

- Views can be dropped using the DROP VIEW command

- Views and Security: Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s)
  - the above view hides courses "cid" from E

Can create a new table from a query on other tables too

SELECT… INTO…. FROM…. WHERE

```
SELECT S.name, E.grade
INTO YoungActiveStudents
 FROM  Students S, Enrolled E
WHERE  S.sid = E.sid and S.age<21
```

"WITH" clause – very useful!

- You will find "WITH" clause very useful!

```
WITH Temp1 AS
     (SELECT ….. ..),
     Temp2 AS
     (SELECT ….. ..)
SELECT X, Y
FROM TEMP1, TEMP2
WHERE….
```

- Can simplify complex nested queries

Summary

- SQL has a huge number of constructs and possibilities
  - You need to learn and practice it on your own
  - Given a problem, you should be able to write a SQL query and verify whether a given one is correct
- Pay attention to NULLs
- Can limit answers using “LIMIT” or “TOP” clauses
  - e.g. to output TOP 20 results according to an aggregate
  - also can sort using ASC or DESC keywords

Relational Query Languages

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, or procedural)
- Note: Declarative (RC, SQL) vs. Operational (RA)

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

Logic Notations

- $\exists$ There exists
- $\forall$ For all
- $\wedge$ Logical AND
- $\vee$ Logical OR
- $\neg$ NOT

Relational Algebra (RA)

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

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

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

Composition of Operators

- Result relation can be the input for another relational algebra operation
  - Operator composition

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

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

- Note: no duplicate
  - "Set semantic"
  - SQL: UNION
  - SQL allows "bag semantic" as well: UNION ALL

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

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

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 ρ

$$(\rho_{sid \rightarrow sid1}\, S1) \times (\rho_{sid \rightarrow sid1}\, R1)$$

or

$$\rho(C(1 \rightarrow sid1,\ 5 \rightarrow sid2),\ S1 \times R1)$$

C is the new relation name

(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

- In general, can use ρ(<Temp>, <RA-expression>)

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

Find names of sailors who've reserved boat #103

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

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

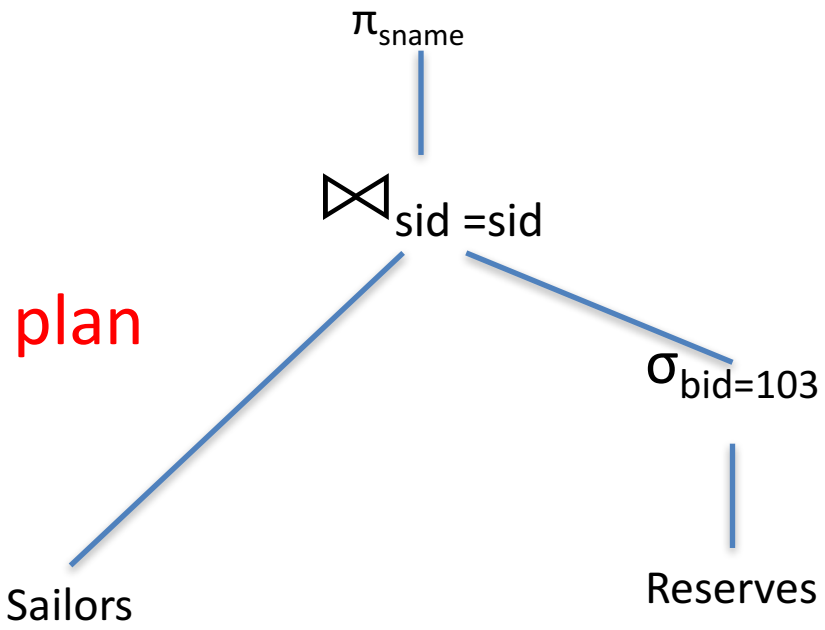
Expressing an RA expression as a Tree

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

Also called a
logical query plan



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

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:

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

$$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$$

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

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
- $\gamma_{\text{age, avg(rating)} \rightarrow \text{avgr}}$ 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 (RC)

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

$\exists$ There exists

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

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

$A \Rightarrow B$

- A "implies" B
- Equivalently, if A is true, B must be true
- Equivalently, $\neg A \vee B$, i.e.
  - either A is false (then B can be anything)
  - otherwise (i.e. A is true) B must be true

Useful Logical Equivalences

$\exists$	There exists
$\forall$	For all
$\bigwedge$	Logical AND
$\bigvee$	Logical OR
$\neg$	NOT

- $\forall x\ P(x) \quad = \quad \neg\exists x \quad [\neg P(x)]$

- $\neg(P \bigvee Q) \quad = \quad \neg P \bigwedge \neg Q$
- $\neg(P \bigwedge Q) \quad = \quad \neg P \bigvee \neg Q$

de Morgan's laws

  - Similarly, $\neg(\neg P \bigvee Q) = P \bigwedge \neg Q$ etc.

- $A \Rightarrow B \quad = \quad \neg A \bigvee B$

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

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

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

TRC: example

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

- Find the names of sailors who have reserved all boats
- Called the "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 in RA!

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

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

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

$\{P \mid \exists S \in Sailors (\forall B \in Boats (B.color = \text{'red'} \Rightarrow (\exists R \in Reserves (S.sid = R.sid \wedge R.bid = B.bid))) \wedge P.sname = S.sname)\}$

Recall that $A \Rightarrow B$ is logically equivalent to $\neg A \vee B$

so $\Rightarrow$ can be avoided, but it is cleaner and more intuitive