L4
Relational Algebra

Eugene Wu

Fall 2015

Reading

Ramakrishnan

Sections 4.1 and 4.2

Helpful References

https://en.wikipedia.org/wiki/Relational_algebra

Overview

Last time, learned about

- pre-relational models
- an informal introduction to relational model
- an introduction to the SQL query language.

Learn about formal relational query languages

- Relational Algebra (algebra: perform operations)
- Relational Calculus (logic: are statements true?)

Keys to understanding SQL and query processing

Who Cares?

- Clean query semantics & rich program analysis
- Helps/enables optimization
- Opens up rich set of topics
  - Materialized views
  - Data lineage/provenance
  - Query by example
  - Distributed query execution
  - …

What's a Query Language?

Allows manipulation and retrieval of data from a database.

Traditionally: QL != programming language

- Doesn't need to be turing complete
- Not designed for computation
- Supports easy, efficient access to large databases

Recent Years

- Scaling to large datasets is a reality
- Query languages are a powerful way to
  - think about data algorithms scale
  - think about asynchronous/parallel programming

What's a Query Language?

MapReduce is **not** the answer

- MapReduce is a powerful primitive to do many kinds of parallel data processing
- BUT
  - Little control of data flow
  - Fault tolerance guarantees not always necessary
  - Simplicity leads to inefficiencies
  - Does not interface with existing analysis software
  - Industry has existing training in SQL

➡ **SQL interface for Hadoop critical for mass adoption**

Tutorial at VLDB 2015 Abadi et al.
http://www.slideshare.net/abadid/sqlonhadoop-tutorial

Formal Relational Query Languages

Formal basis for real languages e.g., SQL

Relational Algebra

Operational, used to represent execution plans

Relational Calculus

Logical, describes what data users want

(not operational, fully declarative)

Prelims

Query is a function over relation instances

$$Q(R_1,\ldots,Rn) = R_{result}$$

Schemas of input and output relations are *fixed* and well defined by the query Q.

Positional vs Named field notation

- Position easier for formal defs
  - one-indexed (not 0-indexed!!!)
- Named more readable
- Both used in SQL

Prelims

Relation (for this lecture)

- Instance is a set of tuples
- Schema defines field names and types (domains)
- Students(<u>sid</u> int, name text, major text, gpa int)

How are relations different than generic sets ($\mathbb{R}$)?

- Can assume item structure due to schema
- Some algebra operations (x) need to be modified
- Will use this later

Relational Algebra Overview

Core 5 operations

- PROJECT (π)
- SELECT (σ)
- UNION (U)
- SET DIFFERENCE (-)
- CROSSPRODUCT (x)

Additional operations

- RENAME (p)
- INTERSECT (∩)
- JOIN (⋈)
- DIVIDE (/)

Instances Used Today: Library

Students, Reservations

Use positional or named field notation

Fields in query results are inherited from input relations (unless specified)

R1

sid	rid	day
1	101	10/10
2	102	11/11

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barak	3	21
3	trump	2	88
5	rusty	3.5	21

Project

$$\pi_{<attr1,\ldots>}(A) = R_{result}$$

Pick out desired attributes (subset of columns)

Schema is subset of input schema in the projection list

- $\pi_{<a,b,c>}(A)$ has output schema (a,b,c) w/ types carried over

Project

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barak	3	21
3	trump	2	88
5	rusty	3.5	21

$$\pi_{name,age}(S2) =$$

name	age
aziz	21
barak	21
trump	88
rusty	21

Project

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barak	3	21
3	trump	2	88
5	rusty	3.5	21

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

age
21
88

Where did all the rows go?
Real systems typically don't remove duplicates. Why?

Select

$$\sigma_{<p>}(A) = R_{result}$$

Select subset of rows that satisfy condition *p*

Won't have duplicates in result. Why?

Result schema same as input

Select

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

$$\sigma_{age<30}\ (S1) =$$

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21

$$\pi_{name}(\sigma_{age<30}\ (S1)) =$$

name
eugene
barak

Commutatively

$$A + B = B + A$$

$$A * B = B * A$$

$$A + (B * C) = (B * C) + A$$

$$A + (B + C) = (A + B) + C$$

$$A + (B * C) = (A + B) * C$$

Commutatively

$$A + B = B + A$$

$$A * B = B * A$$

$$A + (B * C) = (B * C) + A$$

$$A + (B + C) = (A + B) + C$$

$$\cancel{A + (B * C) = (A + B) * C}$$

Commutatively

$\pi_{age}(\sigma_{age<30}(S1))$

$\sigma_{age<30}$ (

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

)

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21

Commutatively

$$\pi_{age}(\sigma_{age<30}(S1))$$

$\sigma_{age<30}$

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

π_{age}

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21

=

age
20
21

Commutatively

$$\sigma_{age<30}(\pi_{age}(S1))$$

π_{age} (

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

)

age
20
21
88

Commutatively

$\sigma_{age<30}(\pi_{age}(S1))$

π_{age} (

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

)

$\sigma_{age<30}$ (

age
20
21
88

) =

age
20
21

Commutatively

Does Project and Select commute?

$$\pi_{age}(\sigma_{age<30}\ (S1)) = \sigma_{age<30}(\pi_{age}(S1))$$

What about

$$\pi_{name}(\sigma_{age<30}\ (S1))?$$

Commutatively

Does Project and Select commute?

$$\pi_{age}(\sigma_{age<30}(S1)) = \sigma_{age<30}(\pi_{age}(S1))$$

What about

$$\pi_{name}(\sigma_{age<30}(S1)) \mathrel{!=} \sigma_{age<30}(\pi_{name}(S1))$$

Commutatively

Does Project and Select commute?

$$\pi_{age}(\sigma_{age<30}(S1)) = \sigma_{age<30}(\pi_{age}(S1))$$

What about

$$\pi_{name}(\sigma_{age<30}(S1)) \mathrel{!=} \sigma_{age<30}(\pi_{name,age}(S1))$$

Commutatively

Does Project and Select commute?

$$\pi_{age}(\sigma_{age<30}(S1)) = \sigma_{age<30}(\pi_{age}(S1))$$

What about

$$\pi_{name}(\sigma_{age<30}(S1)) = \pi_{name}(\sigma_{age<30}(\pi_{name,age}(S1)))$$

OK!

Union, Set-Difference

$$A \text{ op } B = R_{result}$$

A, B must be *union-compatible*

- Same number of fields
- Field i in each schema have same type

Result Schema borrowed from first arg (A)

- A(big int, poppa int) U B(thug int, life int) = ?

Union, Set-Difference

$$A \text{ op } B = R_{result}$$

A, B must be *union-compatible*

- Same number of fields
- Field i in each schema have same type

Result Schema borrowed from first arg (A)

- A(big int, poppa int) U B(thug int, life int) = R_{result}(big int, poppa int)

Union, Intersect, Set-Difference

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barak	3	21
3	trump	2	88
5	rusty	3.5	21

S1US2 =

sid	name	gpa	age
1	eugene	4	20
4	aziz	3.2	21
5	rusty	3.5	21
3	trump	2	88
2	barak	3	21

Union, Intersect, Set-Difference

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barak	3	21
3	trump	2	88
5	rusty	3.5	21

S1 - S2 =

sid	name	gpa	age
1	eugene	4	20

Note on Set Difference & Performance

Notice that most operators are monotonic

- increasing size of inputs → outputs grow
- if $A \supseteq B \rightarrow Q(A, T) \supseteq Q(B, T)$
- can compute *incrementally*

Set Difference is *not monotonic*

- if $A \supseteq B \rightarrow T - A \subseteq T - B$
- e.g., $5 > 1 \rightarrow 9 - 5 \subseteq 9 - 1$

Set difference is *blocking*:

- For T – S, must wait for all S tuples before any results

Cross-Product

$$A(a_1,\ldots,a_n) \times B(a_{n+1},\ldots,a_m) = R_{result}(a_1,\ldots,a_m)$$

Each row of A paired with each row of B

- Result schema concats A and B's fields, inherit if possible
- Conflict: students and reservations have *sid* field

Different than mathematical "X" by flattening results:

- math $A \times B = \{ (a, b) \mid a \in A \wedge b \in B \}$
- e.g., $\{1, 2\} \times \{3, 4\} = \{ (1, 3), (1, 4), (2, 3), (2, 4) \}$
- what is $\{1, 2\} \times \{3, 4\} \times \{5, 6\}$?

Cross-Product

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

R1

sid	rid	day
1	101	10/10
2	102	11/11

S1 x R1 =

(sid)	name	gpa	age	(sid)	rid	day
1	eugene	4	20	1	101	10/10
2	barak	3	21	1	101	10/10
3	trump	2	88	1	101	10/10
1	eugene	4	20	2	102	11/11
2	barak	3	21	2	102	11/11
3	trump	2	88	2	102	11/11

Rename

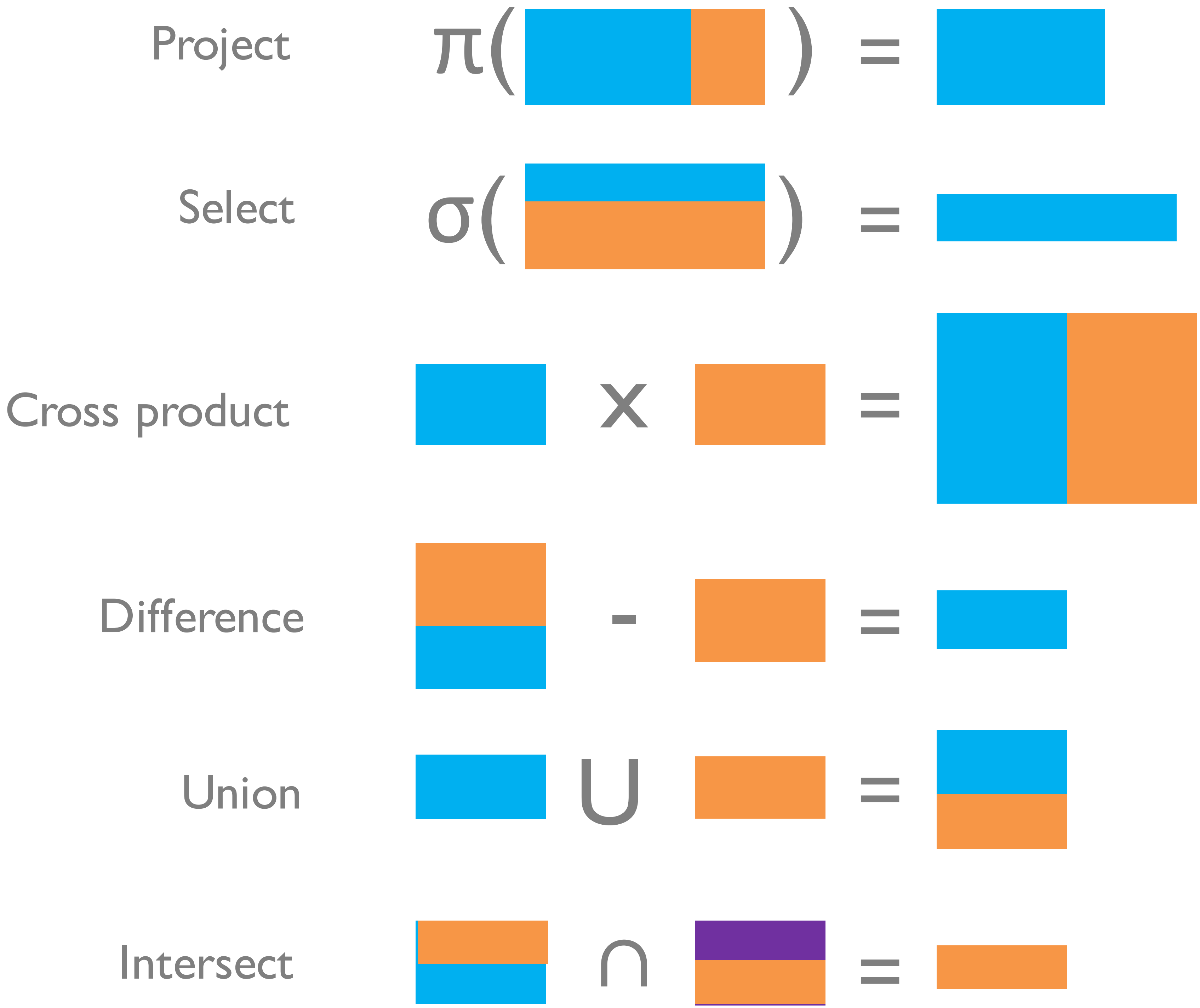
p(<new_name>(<mappings>),Q)

Explicitly defines/changes field names of schema

p(C(1 → sid1, 5 → sid2), S1 x R1)

C =

sid1	name	gpa	age	sid2	rid	day
1	eugene	4	20	1	101	10/10
2	barak	3	21	1	101	10/10
3	trump	2	88	1	101	10/10
1	eugene	4	20	2	102	11/11
2	barak	3	21	2	102	11/11
3	trump	2	88	2	102	11/11



Project
π( ) =
Select
σ( ) =
Cross product
x =
Difference
- =
Union
∪ =
Intersect
∩ =

Compound/Convenience Operators

INTERSECT ($\cap$)

JOIN ($\bowtie$)

DIVIDE (/)

Intersect

$$A \cap B = R_{result}$$

A, B must be *union-compatible*

Intersect

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barak	3	21
3	trump	2	88
5	rusty	3.5	21

$S1 \cap S2 =$

sid	name	gpa	age
2	barak	3	21
3	trump	2	88

Intersect

$$A \cap B = R_{result}$$

A, B must be *union-compatible*

Can we express using core operators?

$$A \cap B = ?$$

Intersect

$$A \cap B = R_{result}$$

A, B must be *union-compatible*

Can we express using core operators?

$A \cap B = A - ?$ (think venn diagram)

Intersect

$$A \cap B = R_{result}$$

A, B must be *union-compatible*

Can we express using core operators?

$$A \cap B = A - (A - B)$$

theta (θ) Join

$$A \bowtie_c B = \sigma_c(A \times B)$$

Most general form

Result schema same as cross product

Often *far* more efficient to compute than cross product

Commutative

$$(A \bowtie_c B) \bowtie_c C = A \bowtie_c (B \bowtie_c C)$$

theta (θ) Join

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

R1

sid	rid	day
1	101	10/10
2	102	11/11

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

(sid)	name	gpa	age	(sid)	rid	day
1	eugene	4	20	1	101	10/10
1	eugene	4	20	2	102	11/11
2	barak	3	21	2	102	11/11

Equi-Join

$$A \bowtie_{attr} B = A \bowtie_{A.attr\,=\,B.attr} B$$

Special case where the condition is attribute equality

Result schema only keeps one copy of equality fields

Natural Join ($A \bowtie B$):

Equijoin on *all* shared fields

Equi-Join

S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	2	88

R1

sid	rid	day
1	101	10/10
2	102	11/11

$S1 \bowtie_{sid} R1 =$

sid	name	gpa	age	rid	day
1	eugene	4	20	101	10/10
2	barak	3	21	102	11/11

Division

Let us have relations A(x, y), B(y)

$$A/B = \{ <x> \mid \exists <x,y> \in A \ \forall_A <y> \in B\}$$

Good to ponder, not supported in most systems (why?)

Find all students that have reserved all books

A/B = all x (students) s.t. for every y (reservation), $<x,y> \in A$

Generalization

y can be a list of fields in B

x U y is fields in A

Examples

A

sid	rid
1	1
1	2
1	3
1	4
2	1
2	2
3	2
4	2
4	4

R1

rid
2

R2

rid
2
4

R3

rid
1
2
4

sid
1
2
3
4

A/R1

sid
1
4

A/R2

sid
1

A/R3

Is A/B a Fundamental Operation?

No. Shorthand like Joins

joins so common, it's natively supported

Hint: Find all *x*s not 'disqualified' by some *y* in *B*.

x value is *disqualified* if by attaching *y* value from *B*, we obtain an *x*,*y* tuple that is not in *A*.

A

sid	rid
1	1
1	2
1	3
1	4
2	1
2	2
3	2
4	2
4	4

B

rid
2
4

Disqualified =

A/B =

A

sid	rid
1	1
1	2
1	3
1	4
2	1
2	2
3	2
4	2
4	4

B

rid
2
4

$\pi_x(A) \times B$

sid	rid
1	2
1	4
2	2
2	4
3	2
3	4
4	2
4	4

Disqualified = $(\pi_x(A) \times B)$

A/B =

A

sid	rid
1	1
1	2
1	3
1	4
2	1
2	2
3	2
4	2
4	4

B

rid
2
4

$\pi_x(A) \times B$

sid	rid
1	2
1	4
2	2
2	4
3	2
3	4
4	2
4	4

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

sid	rid
2	4
3	4

Disqualified = $((\pi_x(A) \times B) - A)$

A/B =

A

sid	rid
1	1
1	2
1	3
1	4
2	1
2	2
3	2
4	2
4	4

B

rid
2
4

$\pi_x(A) \times B$

sid	rid
1	2
1	4
2	2
2	4
3	2
3	4
4	2
4	4

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

sid	rid
2	4
3	4

sid
1
4

A/B

$$\text{Disqualified} = \pi_x((\pi_x(A) \times B) - A)$$
$$A/B = \pi_x(A) - \text{Disqualified}$$

Names of students that reserved book 2

$\pi_{name}(\sigma_{rid=2}\ (R1) \bowtie S1)$

Equivalent Queries

$p(Tmp1,\ \sigma_{rid=2}\ (R1))$
$p(Tmp2,\ Tmp1 \bowtie S1)$
$\pi_{name}(Tmp2)$

$\pi_{name}(\sigma_{rid=2}(R1 \bowtie S1))$

Names of students that reserved db books

Book(bid, type)   Reserve(sid, bid)   Student(sid)

Need to join with Instructors to access role

$\pi_{name}(\sigma_{type='db'}(\text{Book}) \bowtie \text{Reserve} \bowtie \text{Student})$

More efficient query

$\pi_{name}(\pi_{sid}((\pi_{cid}\sigma_{type='db'}(\text{Book})) \bowtie \text{Reserve}) \bowtie \text{Student})$

Query optimizer can find the more efficient query!

Students that reserved DB or HCI book

1. Find all DB or HCI books
2. Find students that reserved one of those books

$$p(tmp, (\sigma_{type='DB' \vee type='HCI'} (Book))$$

$$\pi_{name}(tmp \bowtie Reserve \bowtie Student)$$

Alternatives

define tmp using UNION (how?)

what if we replaced v with ^ in the query?

Students that reserved a DB and HCI book

Does previous approach work?

$$p(tmp, (\sigma_{type='DB' \wedge type='HCI'} (Book))$$

$$\pi_{name}(tmp \bowtie Reserve \bowtie Student)$$

NO

Students that reserved a DB and HCI book

Does previous approach work?

1. Find students that reserved DB books
2. Find students that reversed HCI books
3. Intersection

$$p(tmpDB, \pi_{sid}(\sigma_{type='DB'} \text{ Book}) \bowtie \text{Reserve})$$
$$p(tmpHCI, \pi_{sid}(\sigma_{type='HCI'} \text{ Book}) \bowtie \text{Reserve})$$
$$\pi_{name}((tmpDB \cap tmpHCI) \bowtie \text{Student})$$

Is the intersection always allowed?

What if it projected book name?

Students that reserved all books

Use division

Be careful with schemas of inputs to / !

$$p(tmp, (\pi_{sid,bid} \text{ Reserves}) \;/\; (\pi_{bid} \text{Books}))$$

$$\pi_{name}(tmp \bowtie \text{Student})$$

What if want students that reserved all horror books?

$$p(tmp, (\pi_{sid,bid} \text{ Reserves}) \;/\; (\pi_{bid}(\sigma_{type='horror'} \text{ Book})))$$

Let's step back

Relational algebra is expressiveness benchmark

- A language equal in expressiveness as relational algebra is relationally complete

But has limitations

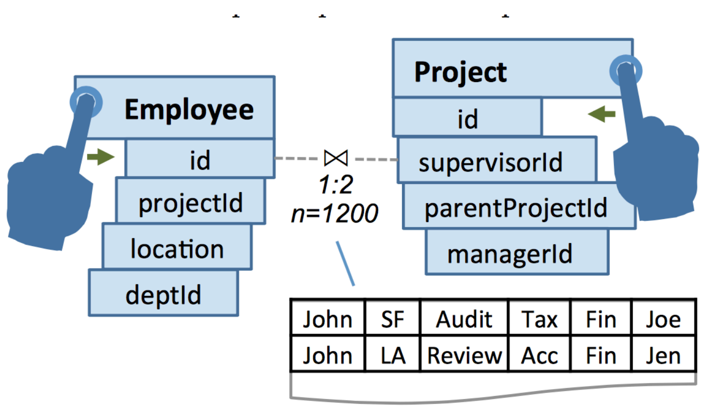
- nulls
- aggregation
- recursion
- duplicates

What can we do with RA?

Query by example

Here's my data and examples of the result, *generate the query for me*

Novel relationally complete interfaces



GestureDB. Nandi et al.

Summary

Relational Algebra (RA) operators

Operators are closed (inputs & outputs are relations)

Multiple Relational Algebra queries can have same semantics (always return same results)

- Forms basis for optimizations

Next Time

Relational Calculus