Relational Algebra

Eugene Wu

Supplemental Materials

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

Relational algebra is the basis for the most popular **query language** in the universe*

*known universe

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 (very) large databases

Recent years

- Scaling to large datasets is a reality
- Powerful way to think about…
  - data algorithms that scale
  - asynchronous/parallel programming

Formal Relational Query Languages

Relational Algebra

Operational, used to represent execution plans

$\pi_{name}(\sigma_{age<30}(Sailors))$ sailor names younger than 30

Relational Calculus

Logical, describes what data users want (declarative)

`{ s: name | s ∈ Sailors ^ s.age < 30 }`

`(this is shorthand)`

Journey of a Query

SQL — SELECT … FROM …

Relational Algebra — $\pi_b(P \bowtie Q \bowtie ...)$

Query Rewriting — $\pi_b(P \bowtie S) \bowtie Q...)$

Query execution

https://cseweb.ucsd.edu/classes/fa19/cse132A-a/slides/relational-calculus.pdf

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 is more readable
- Both used in SQL

Prelims

Relation (for this lecture)

- Instance: **set** of tuples (important!)
- Schema: list of 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 ($\cup$)
- SET DIFFERENCE (-)
- CROSSPRODUCT (x)

Additional operations

- RENAME (ρ)
- INTERSECT ($\cap$)
- JOIN ($\bowtie$)
- 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	barb	3	21
3	tanya	2	88

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barb	3	21
3	tanya	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	barb	3	21
3	tanya	2	88
5	rusty	3.5	21

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

name	age
aziz	21
barb	21
tanya	88
rusty	21

Project

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barb	3	21
3	tanya	2	88
5	rusty	3.5	21

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

name	name	age
aziz	aziz	21
barb	barb	21
tanya	tanya	88
rusty	rusty	21

Project

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barb	3	21
3	tanya	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 by default. Why?

Select

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

Select subset of rows that satisfy condition *p*

p: Boolean expr over constants and attributes in A

Won't have duplicates in result. Why?

Result schema same as input

Select

S1

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	2	88

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

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

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

name
eugene
barb

Commutative Operations

$$A + B = B + A$$

$$A * B = B * A$$

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

Associative Operations

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

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

Commutative Operations

$$A + B = B + A$$

$$A * B = B * A$$

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

Associative Operations

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

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

Commutatively

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

$$\underbrace{\sigma_{age<30}\left(\begin{array}{|l|l|l|l|} \hline \text{sid} & \text{name} & \text{gpa} & \text{age} \\ \hline 1 & \text{eugene} & 4 & 20 \\ \hline 2 & \text{barb} & 3 & 21 \\ \hline 3 & \text{tanya} & 2 & 88 \\ \hline \end{array} \right)}$$

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

Commutatively

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

$$\underbrace{\sigma_{age<30}\left(\begin{array}{|l|l|l|l|}\hline sid & name & gpa & age \\ \hline 1 & eugene & 4 & 20 \\ \hline 2 & barb & 3 & 21 \\ \hline 3 & tanya & 2 & 88 \\ \hline\end{array}\right)}$$

$$\pi_{age}\left(\begin{array}{|l|l|l|l|}\hline sid & name & gpa & age \\ \hline 1 & eugene & 4 & 20 \\ \hline 2 & barb & 3 & 21 \\ \hline\end{array}\right) = \begin{array}{|l|}\hline age \\ \hline 20 \\ \hline 21 \\ \hline\end{array}$$

Commutatively

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

π_{age} (

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	2	88

)

age
20
21
88

Commutatively

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

π_{age} (

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	2	88

)

$\sigma_{age<30}$ (

age
20
21
88

) =

age
20
21

Commutatively

Does Project and Select **always** 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

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

Commutatively

Does Project and Select commute?

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

What about

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

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(id int, imgid int) U B(blah int, gloop 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 taken from first relation (A)

- A(id int, imgid int) U B(blah int, gloop int) = R_{result}(id int, imgid int)

Union, Intersect, Set-Difference

S1

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	2	88

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barb	3	21
3	tanya	2	88
5	rusty	3.5	21

S1 U S2 =

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

Union, Intersect, Set-Difference

S1

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	2	88

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barb	3	21
3	tanya	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., $5 > 1 \rightarrow 9 - 5 < 9 - 1$

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
- Fields found in both A and B are undefined in result
- (some DBMSes set a default)

Not same as mathematical "X", which returns **nested** results:

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

Cross-Product

S1

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	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	barb	3	21	1	101	10/10
3	tanya	2	88	1	101	10/10
1	eugene	4	20	2	102	11/11
2	barb	3	21	2	102	11/11
3	tanya	2	88	2	102	11/11

Rename

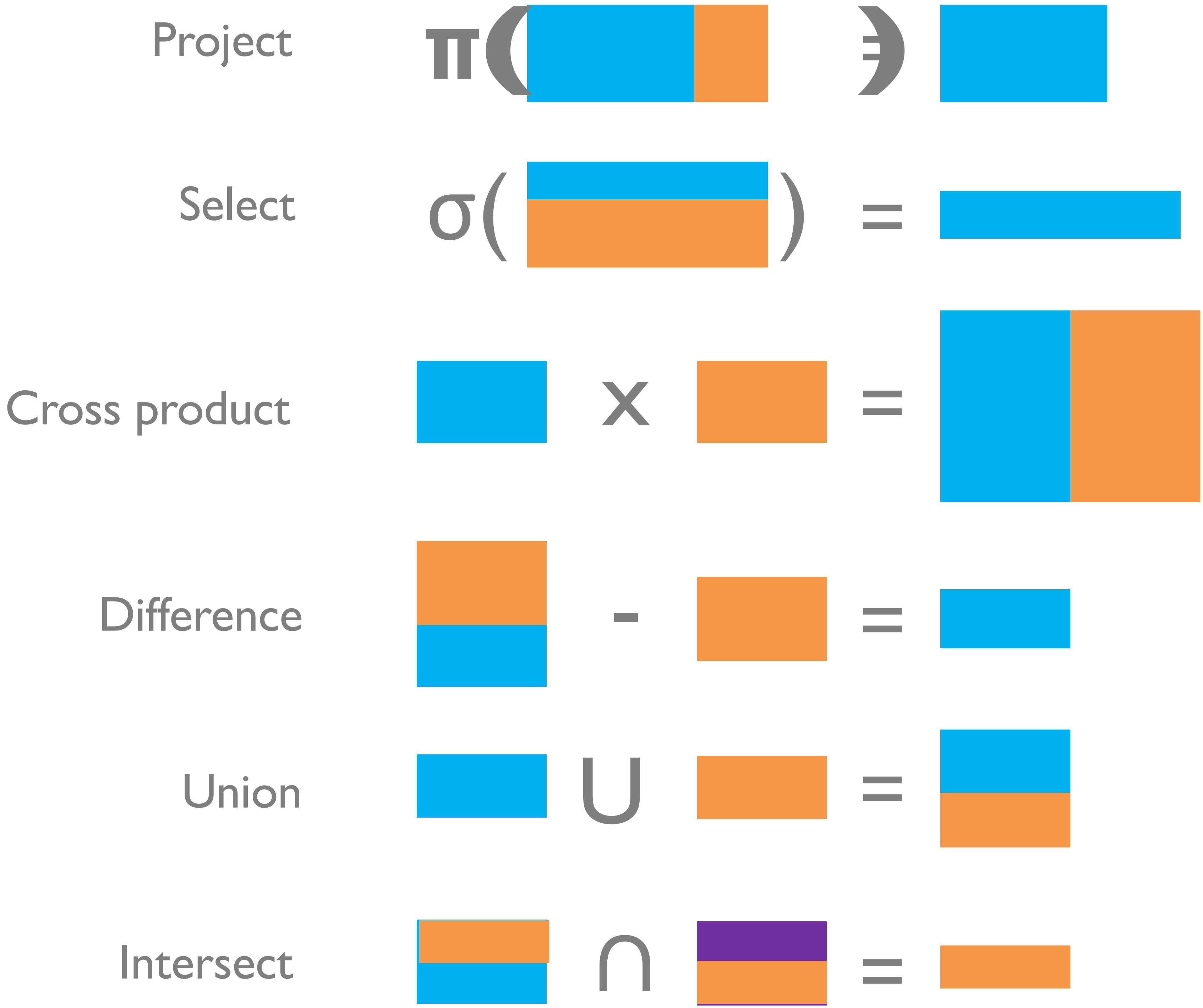
p(<newRelationName>(<mappings>), Q)

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	barb	3	21	1	101	10/10
3	tanya	2	88	1	101	10/10
1	eugene	4	20	2	102	11/11
2	barb	3	21	2	102	11/11
3	tanya	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	barb	3	21
3	tanya	2	88

S2

sid	name	gpa	age
4	aziz	3.2	21
2	barb	3	21
3	tanya	2	88
5	rusty	3.5	21

$S1 \cap S2 =$

sid	name	gpa	age
2	barb	3	21
3	tanya	2	88

Intersect

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

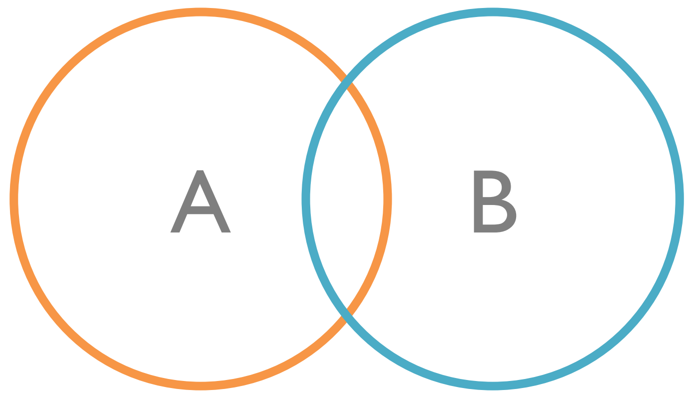
A, B must be *union-compatible*

Can we express using core operators?

$$A \cap B \;=\; ?$$

Intersect

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

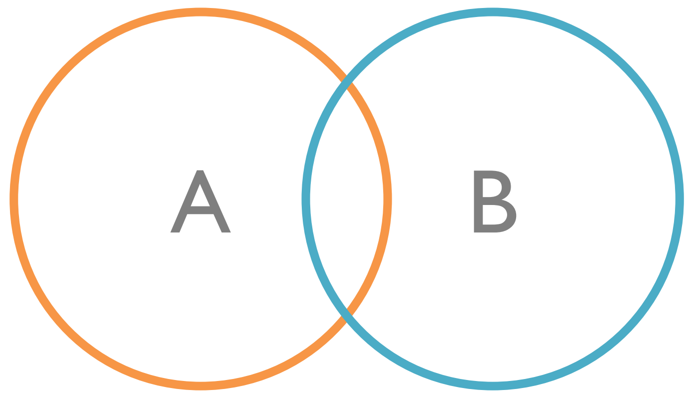


Can we express using core operators?

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

Intersect

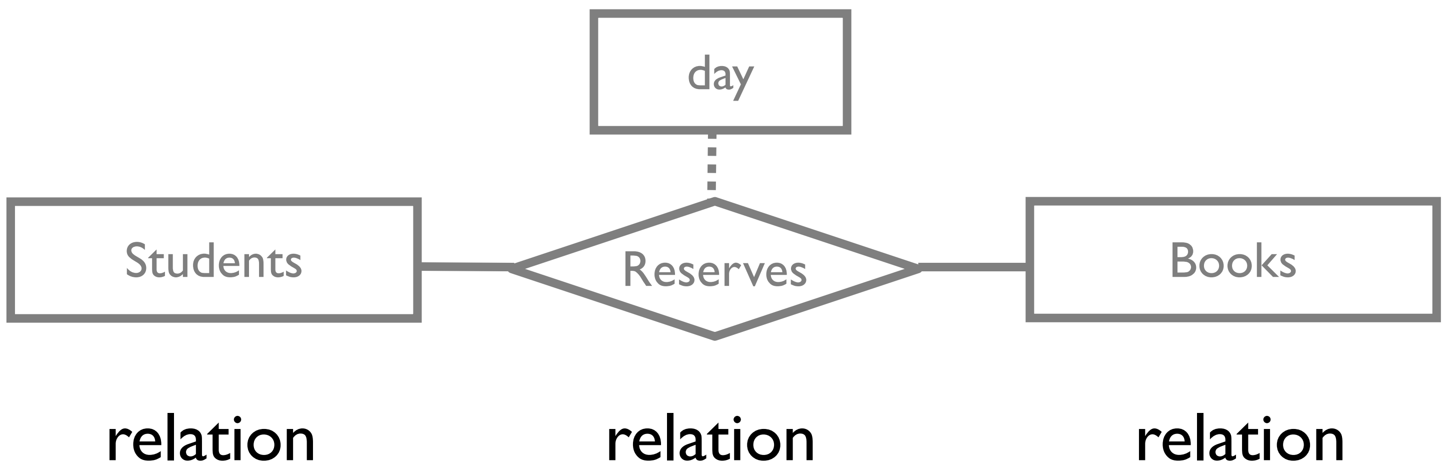
$$A \cap B = R_{result}$$



Can we express using core operators?

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

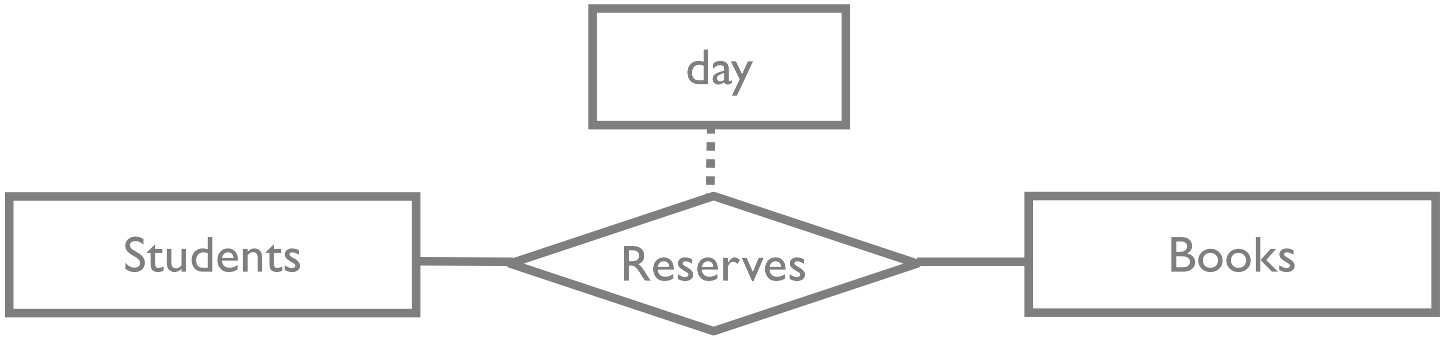
Joins (high level)



What if you want to query across all three tables?
e.g., all names of students that reserved "The Purple Crayon"

Need to combine these tables
Cross product? But that ignores foreign key references

Joins (high level)



relation relation relation

S1

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	2	88

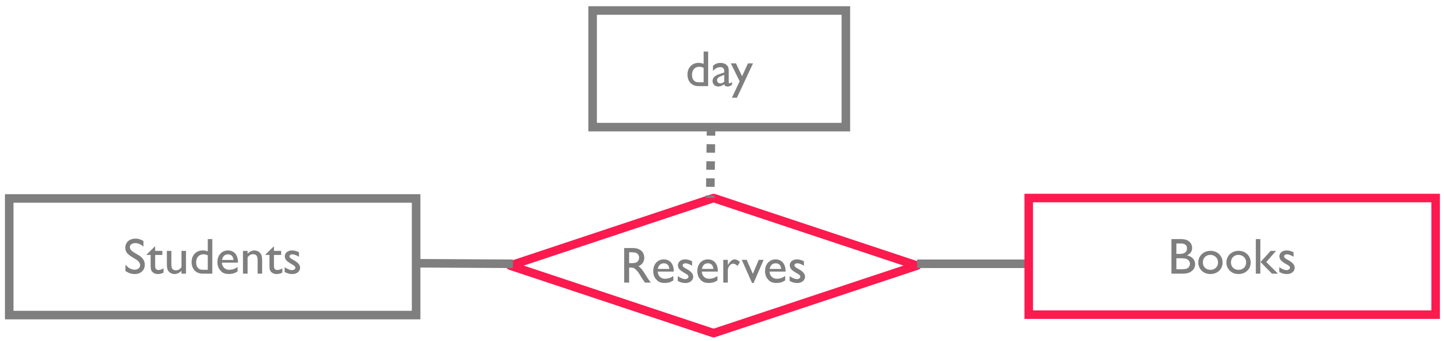
R1

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

B1

rid	name
101	The Purple Crayon
102	1984

Joins (high level)



relation relation relation

S1

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	2	88

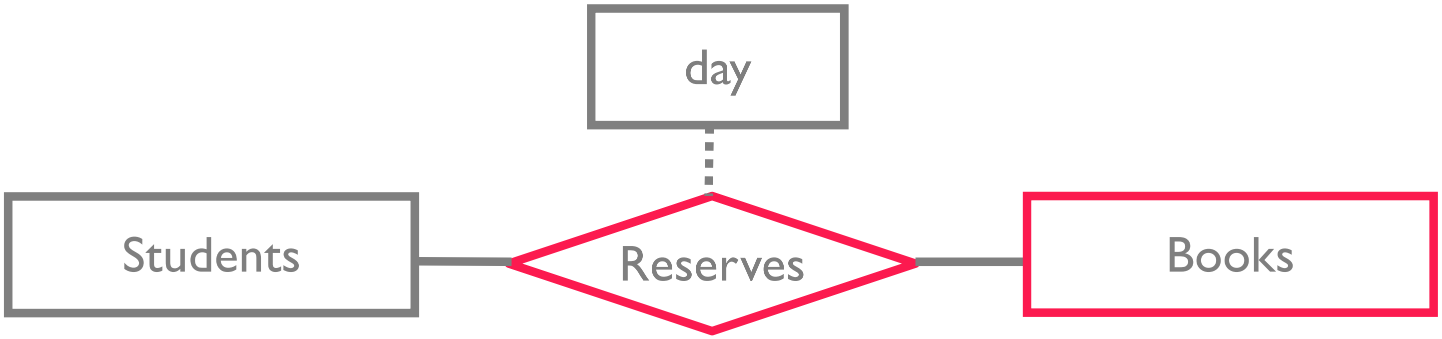
R1

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

B1

rid	name
101	The Purple Crayon
102	1984

Joins (high level)



relation relation relation

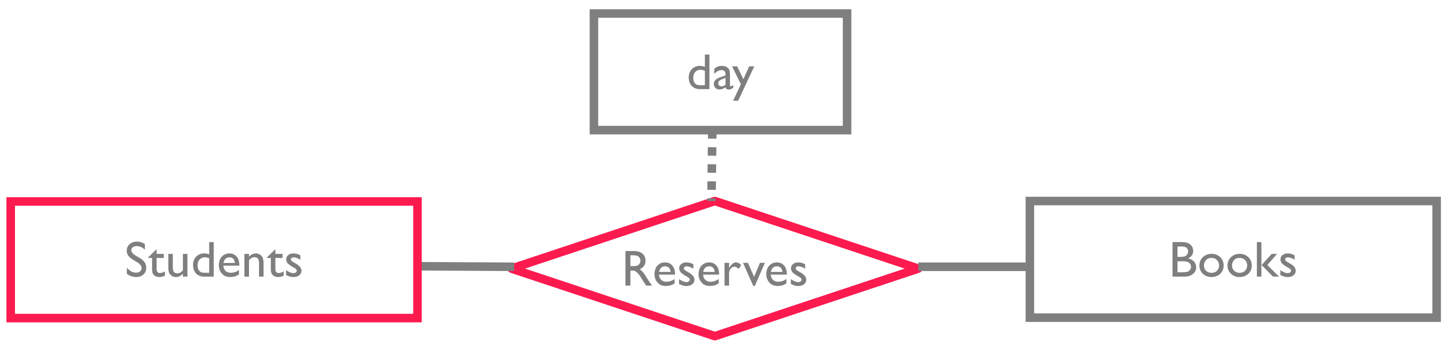
S1

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	2	88

RB1

sid	(rid)	day	(rid)	name
1	101	10/10	101	The Purple Crayon
2	102	11/11	102	1984

Joins (high level)



relation relation relation

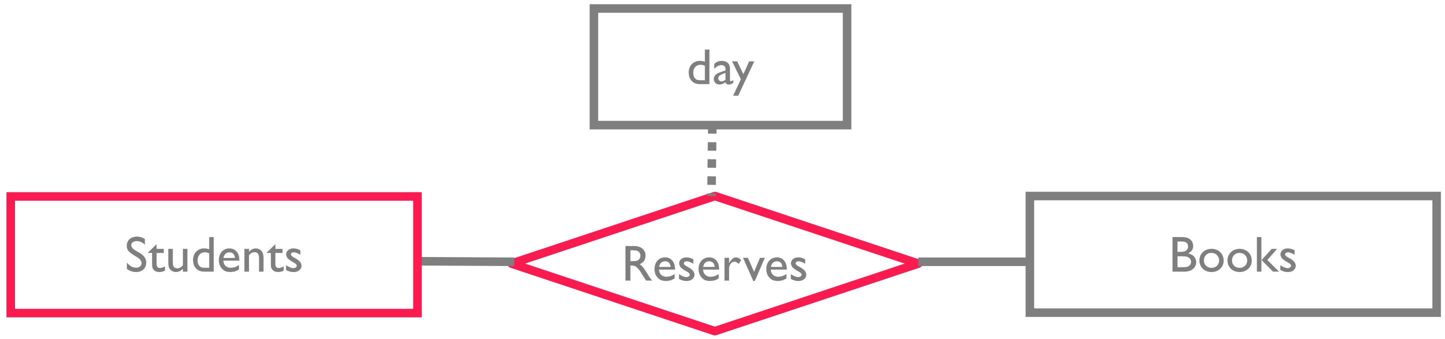
S1

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	2	88

RB1

sid	(rid)	day	(rid)	name
1	101	10/10	101	The Purple Crayon
2	102	11/11	102	1984

Joins (high level)



relation relation relation

SRB1

(sid)	(name)	gpa	age	(sid)	(rid)	day	(rid)	(name)
1	eugene	4	20	1	101	10/10	101	The Purple Crayon
2	barb	3	21	2	102	11/11	102	1984

Joins

Theta (θ) Join

Equi-join

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	barb	3	21
3	tanya	2	88

R1

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

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

$\sigma_{S1.sid \leq R1.sid}(S1 \times R1) =$

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

theta (θ) Join

S1

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	2	88

R1

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

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

$\sigma_{S1.sid \leq R1.sid}(S1 \times R1) =$

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

theta (θ) Join

S1

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	2	88

R1

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

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

$\sigma_{S1.sid \leq R1.sid}(S1 \times R1) =$

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

Equi-Join

$$A \bowtie_{attr} B = \boldsymbol{\pi}_{\text{all attrs } \textit{except} \text{ B.attr}}(A \bowtie_{A.attr = B.attr} B)$$

List the attributes that the two relations will be joined on explicitly.

$A \bowtie_{x,y} B$ is an equijoin on attributes x and y

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 (fields w/ same name)

Equi-Join

S1

sid	name	gpa	age
1	eugene	4	20
2	barb	3	21
3	tanya	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	barb	3	21	102	11/11

Equi-Join

S1

sid	name	day	gpa	age
1	eugene	10/10	4	20
2	barb	12/12	3	21
3	tanya	3/3	2	88

R1

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

$S1 \bowtie_{sid,name} R1$ = INVALID! name not in R1

$$S1 \bowtie_{sid,day} R1 = S1 \bowtie_{s1.sid=R1.sid \wedge s1.day = R1.day} R1$$

sid	name	day	gpa	age	rid
1	eugene	10/10	4	20	101

$$S1 \bowtie R1 = S1 \bowtie_{sid,day} R1$$

Different Plans, Same Results

Semantic equivalence:

results are *always* the same

Note that it is independent

of the database instance!

Names of students that reserved book 2

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

Equivalent Queries

$\rho(tmp1, \sigma_{rid=2}(R1))$
$\rho(tmp2, tmp1 \bowtie S1)$
$\pi_{name}(tmp2)$

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

Names of students that reserved db books

Book(rid, type) Reserve(sid, rid) Student(sid, name)

Need to join DB books with reserve and students

$\sigma_{type='db'}$ (Book)

Names of students that reserved db books

Book(rid, type)   Reserve(sid, rid)   Student(sid, name)

Need to join DB books with reserve and students

$\sigma_{type='db'}$ (Book) $\bowtie$ Reserve

Names of students that reserved db books

Book(rid, type) Reserve(sid, rid) Student(sid, name)

Need to join DB books with reserve and students

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

Names of students that reserved db books

Book(rid, type)   Reserve(sid, rid)   Student(sid, name)

Need to join DB books with reserve and students

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

Names of students that reserved db books

Book(rid, type)   Reserve(sid, rid)   Student(sid, name)

Need to join DB books with reserve and students

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

More efficient query

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

Query optimizer can find the more efficient query!

Names of students that reserved db books

Book(rid, type)   Reserve(sid, rid)   Student(sid, name)

Need to join DB books with reserve and students

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

More efficient query

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

Query optimizer can find the more efficient query!

Names of students that reserved db books

Book(rid, type)   Reserve(sid, rid)   Student(sid, name)

Need to join DB books with reserve and students

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

More efficient query

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

Query optimizer can find the more efficient query!

Names of students that reserved db books

Book(rid, type) Reserve(sid, rid) Student(sid, name)

Need to join DB books with reserve and students

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

More efficient query

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

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

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

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

"v" means logical OR

Alternatives

define tmp using UNION (how?)

Using UNION

$p(tmp1, (\sigma_{type='DB'}\ (Book))$

$p(dbnames, \pi_{name}(tmp1 \bowtie Reserve \bowtie Student))$

$p(tmp2, (\sigma_{type='HCI'}\ (Book))$

$p(hcinames, \pi_{name}(tmp2 \bowtie Reserve \bowtie Student))$

dbnames UNION hcinames

Students that reserved a DB and HCI book

Can we change v into ^ (AND)?

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

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

NO

Why?

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

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

```
for b in Book:
    if b.type = 'DB' and b.type = 'HCI':        // resolves to FALSE
        for r in Reserve:
            for s in Student:
                if r.sid = s.sid and r.bid = b.bid:
                    yield b.name
```

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

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

Students that reserved all books

Students where, **for all books**, the student reserved the book

no concept of "for all" in relational algebra…

Students – Students that didn't reserve all books

Students that reserved all books

Students where, **for all books**, the student reserved the book

no concept of "for all" in relational algebra…

Students – Students where there is a book that they did not reserve

Students that reserved all books

Students where, **for all books**, the student reserved the book

no concept of "for all" in relational algebra…

Students – (Students s where (Books – Books s reserved))

$$p(s\_reserved,\ \pi_{bid}\ \sigma_{sid=bob}(Reserve))$$

Students – (Students s where (Books – Books s reserved))

$$p(s\_not\_reserved,\ \pi_{bid}(Books) - s\_reserved)$$

Students – (Students s where (Books – Books s reserved))

$$\pi_{sid,bid}(Students \times Books)$$

Students that reserved all books

Students where, **for all books**, the student reserved the book

no concept of "for all" in relational algebra…

Students – (Students s where (Books – Books s reserved))

`p(s_reserved,` $\pi_{bid}\ \sigma_{sid=bob}$`(Reserve))`

Students – (Students s where (Books – Books s reserved))

`p(s_not_reserved,` π_{bid}`(Books) - s_reserved)`

Students – (Students s where (Books – Books s reserved))

$\pi_{sid,bid}$`(Students x Books) -` $\pi_{sid,bid}$`(Reserve)`

Students that reserved all books

Students where, **for all books**, the student reserved the book

no concept of "for all" in relational algebra…

Students – (Students s where (Books – Books s reserved))

```
p(s_reserved, πbid σsid=bob(Reserve))
```

$$p(s\_reserved,\ \pi_{bid}\ \sigma_{sid=bob}(Reserve))$$

Students – (Students s where (Books – Books s reserved))

$$p(s\_not\_reserved,\ \pi_{bid}(Books)\ -\ s\_reserved)$$

Students – (Students s where (Books – Books s reserved))

$$p(del\_sids,\ \pi_{sid}(\pi_{sid,bid}(Students\ x\ Books)\ -\ \pi_{sid,bid}(Reserve)))$$

Students that reserved all books

Students where, **for all books**, the student reserved the book

no concept of "for all" in relational algebra…

Students – (Students s where (Books – Books s reserved))

`p(s_reserved, `$\pi_{bid}\ \sigma_{sid=bob}$`(Reserve))`

Students – (Students s where (Books – Books s reserved))

`p(s_not_reserved, `π_{bid}`(Books) - s_reserved)`

Students – (Students s where (Books – Books s reserved))

`p(del_sids, `$\pi_{sid}(\pi_{sid,bid}$`(Students x Books) - `$\pi_{sid,bid}$`(Reserve)))`

π_{sid} `(Students) - del_sids`

Let's step back

Relational algebra is expressiveness benchmark

- A language that can express relational algebra is "relationally complete"

Limitations

- nulls
- aggregation
- recursion
- duplicates
- can't really type on keyboard…

Equi-Joins are everywhere

Matching of two sets based on shared attributes

Yelp:	Join between your location and restaurants
Market:	Join between consumers and suppliers
High five:	Join between two hands on time and space
Comm.:	Join between minds on ideas/concepts

Mat ute

Who Cares about Relational Alg?

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
- …

You see its fingerprints EVERYWHERE!

What can we do with RA?

Query(DB instance) → Relation instance

What can we do with RA?

Query(DB instance) = Relation instance

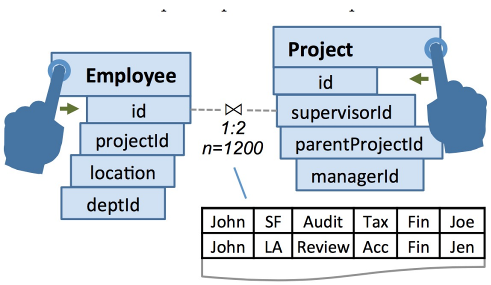
Query by example

Here's DB instance and result, generate the query

Data Generation:

Here's query and result, generate a DB instance

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 be equivalent

- It is operational
- Same semantics but different performance
- Forms basis for optimizations