HW1 out
Proj 1 Part 1 due next Monday
Proj 1 Part 2 out next Monday!

L7
Relational Algebra

Eugene Wu
Fall 2016

Supplemental Materials

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
- …

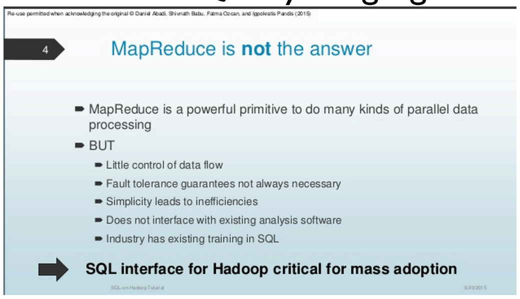
What's a Query Language?

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?



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)

Relational Algebra

1+2
2+1
0+3
…

{ (a,b) | a, b are integers and a+b=3 }

Prelims

Query is a function over relation instances

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

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 (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	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

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

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
~~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

S1 U S2 =

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 $A \supseteq B$ → $Q(A,T) \supseteq Q(B,T)$

can compute *incrementally*

Set Difference is *not monotonic*

if $A \supseteq B$ → $T - A \subseteq T - B$

e.g., 5 > 1 → 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

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} X {3, 4} = { (1, 3), (1, 4), (2, 3), (2, 4) }

what is {1, 2} x {3, 4} x {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

Administrivia

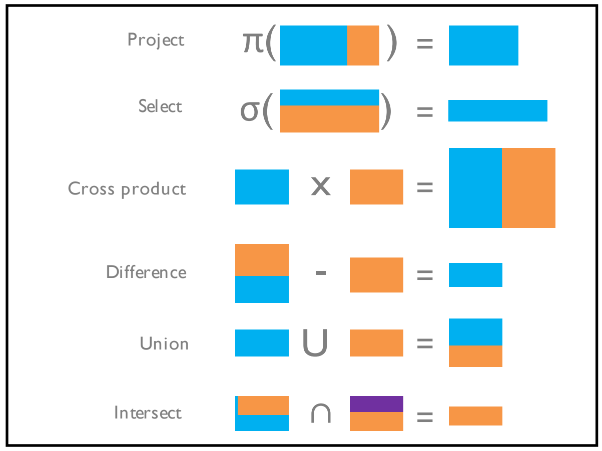
Azure codes – should have gotten google invite
- What kind of transactional guarantees do you think google spreadsheets provides?

Redeem: https://www.microsoftazurepass.com/
- (on course website)

You are not required to understand lec3.md
- They were my lecture notes

Proj1 Part1 returned today
- PostgreSQL passwords written on them

HW2 out on Friday



Compound/Convenience Operators

INTERSECT (∩)

JOIN (⋈)

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

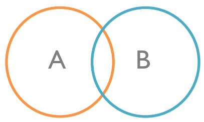
$A \cap B = R_{result}$

A, B must be *union-compatible*

Can we express using core operators?
- $A \cap B$ = ?

Intersect

$A \cap B = R_{result}$

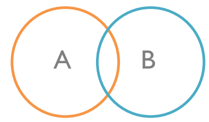


Can we express using core operators?

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

Intersect

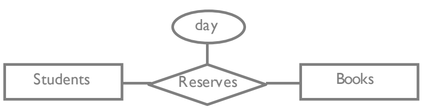
$A \cap B = R_{result}$



Can we express using core operators?

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

Joins (high level)



relation relation relation

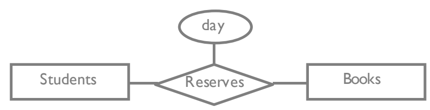
What if you want to query across all three tables?

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	barak	3	21
3	trump	2	88

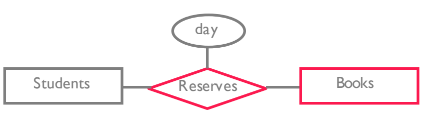
R1

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	barak	3	21
3	trump	2	88

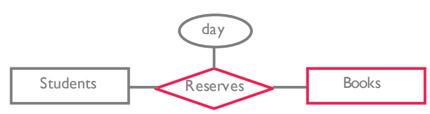
R1

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

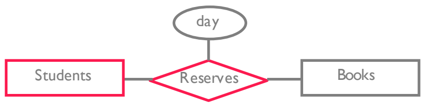
S1

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

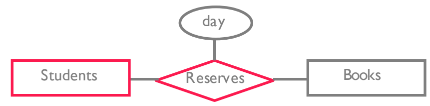
S1

sid	name	gpa	age
1	eugene	4	20
2	barak	3	21
3	trump	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	barak	3	21	2	102	11/11	102	1984

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

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

(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

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

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 (fields w/ same name)