CSE 344

APRIL 6TH – RELATIONAL ALGEBRA

ASSORTED MINUTIAE

- **HW2 out (Due Wednesday)**
  - git pull upstream master
- **OQ1 due tonight**
- **OQ2/3 Out**
  - Both due next Friday
- **Azure accounts will be created over the weekend**
  - Needed for HW3

RELATIONAL ALGEBRA

- **Remember from last week**
  - SQL queries are combinations of functions on tables
  - Each one receives tables as input and has a table as an output

RELATIONAL ALGEBRA

Set-at-a-time algebra, which manipulates relations

In SQL we say *what* we want

In RA we can express *how* to get it

Every DBMS implementations converts a SQL query to RA in order to execute it

An RA expression is called a *query plan*

BASICS

- Relations and attributes
- Functions that are applied to relations
  - Return relations
  - Can be composed together
  - Often displayed using a tree rather than linearly
  - Use Greek symbols: σ, π, δ, etc

SETS V.S. BAGS

Sets: {a,b,c}, {a,d,e,f}, { }, . . .

Bags: {a, a, b, c}, {b, b, b, b, b}, . . .

Relational Algebra has two flavors:

Set semantics = standard Relational Algebra

Bag semantics = extended Relational Algebra

DB systems implement bag semantics (Why?)

RELATIONAL ALGEBRA OPERATORS

Union ∪, ~~intersection ∩~~, difference -

Selection σ

Projection π

Cartesian product X, join ⋈

(Rename ρ)

Duplicate elimination δ

Grouping and aggregation γ

Sorting τ

RA

Extended RA

All operators take in 1 or more relations as inputs and return another relation

UNION AND DIFFERENCE

$$R1 \cup R2$$
$$R1 - R2$$

Only make sense if R1, R2 have the same schema

What do they mean over bags ?

WHAT ABOUT INTERSECTION ?

Derived operator using minus

$$\boxed{R1 \cap R2 = R1 - (R1 - R2)}$$

Derived using join

$$\boxed{R1 \cap R2 = R1 \bowtie R2}$$

SELECTION

Returns all tuples which satisfy a condition

$$\boxed{\sigma_c(R)}$$

Examples

- $\sigma_{\text{Salary > 40000}}$ (Employee)
- $\sigma_{\text{name = “Smith”}}$ (Employee)

The condition c can be =, <, <=, >, >=, <> combined with AND, OR, NOT

Employee

SSN	Name	Salary
1234545	John	20000
5423341	Smith	60000
4352342	Fred	50000

$\sigma_{Salary > 40000}$ (Employee)

SSN	Name	Salary
5423341	Smith	60000
4352342	Fred	50000

PROJECTION

Eliminates columns

$$\boxed{\pi_{A1,\ldots,An}(R)}$$

Example: project social-security number and names:

- $\pi_{SSN,\ Name}$ (Employee) → Answer(SSN, Name)

Different semantics over sets or bags! Why?

Employee

SSN	Name	Salary
1234545	John	20000
5423341	John	60000
4352342	John	20000

$\pi_{\text{Name,Salary}}$ (Employee)

Name	Salary
John	20000
John	60000
John	20000

Bag semantics

Name	Salary
John	20000
John	60000

Set semantics

Which is more efficient?

COMPOSING RA OPERATORS

Patient

no	name	zip	disease
1	p1	98125	flu
2	p2	98125	heart
3	p3	98120	lung
4	p4	98120	heart

$\pi_{zip,disease}(\text{Patient})$

zip	disease
98125	flu
98125	heart
98120	lung
98120	heart

$\sigma_{disease=\text{'heart'}}(\text{Patient})$

no	name	zip	disease
2	p2	98125	heart
4	p4	98120	heart

$\pi_{zip,disease}(\sigma_{disease=\text{'heart'}}(\text{Patient}))$

zip	disease
98125	heart
98120	heart

CARTESIAN PRODUCT

Each tuple in R1 with each tuple in R2

$$\boxed{R1 \times R2}$$

Rare in practice; mainly used to express joins

CROSS-PRODUCT EXAMPLE

Employee

Name	SSN
John	999999999
Tony	777777777

Dependent

EmpSSN	DepName
999999999	Emily
777777777	Joe

Employee X Dependent

Name	SSN	EmpSSN	DepName
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

NATURAL JOIN

$$\boxed{R1 \bowtie R2}$$

Meaning: $R1 \bowtie R2 = \Pi_A(\sigma_\theta (R1 \times R2))$

Where:

- Selection σ_θ checks equality of all common attributes (i.e., attributes with same names)
- Projection Π_A eliminates duplicate common attributes

NATURAL JOIN EXAMPLE

R

A	B
X	Y
X	Z
Y	Z
Z	V

S

B	C
Z	U
V	W
Z	V

R $\bowtie$ S =
$\Pi_{ABC}(\sigma_{R.B=S.B}(R \times S))$

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

NATURAL JOIN EXAMPLE 2

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
Alice	54	98125
Bob	20	98120

$P \bowtie V$

age	zip	disease	name
54	98125	heart	Alice
20	98120	flu	Bob

AnonPatient (age, zip, disease)
Voters (name, age, zip)

THETA JOIN

A join that involves a predicate

$$\boxed{R1 \bowtie_{\theta} R2 \;=\; \sigma_{\theta}\,(R1 \times R2)}$$

Here θ can be any condition

No projection in this case!

For our voters/patients example:

$$P \bowtie_{P.zip = V.zip \text{ and } P.age >= V.age - 1 \text{ and } P.age <= V.age + 1} V$$

EQUIJOIN

A theta join where θ is an equality predicate

$$\boxed{R1 \bowtie_{\theta} R2 \quad = \sigma_{\theta} (R1 \times R2)}$$

By far the most used variant of join in practice

What is the relationship with natural join?

EQUIJOIN EXAMPLE

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie_{P.age=V.age} V$

P.age	P.zip	P.disease	V.name	V.age	V.zip
54	98125	heart	p1	54	98125
20	98120	flu	p2	20	98120

JOIN SUMMARY

Theta-join: $R \bowtie_\theta S = \sigma_\theta (R \times S)$

- Join of R and S with a join condition θ
- Cross-product followed by selection θ
- No projection

Equijoin: $R \bowtie_\theta S = \sigma_\theta (R \times S)$

- Join condition θ consists only of equalities
- No projection

Natural join: $R \bowtie S = \pi_A (\sigma_\theta (R \times S))$

- Equality on **all** fields with same name in R and in S
- Projection π_A drops all redundant attributes

SO WHICH JOIN IS IT ?

When we write $R \bowtie S$ we usually mean an equijoin, but we often omit the equality predicate when it is clear from the context

MORE JOINS

Outer join

- Include tuples with no matches in the output
- Use NULL values for missing attributes
- Does not eliminate duplicate columns

Variants

- Left outer join
- Right outer join
- Full outer join

OUTER JOIN EXAMPLE

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu
33	98120	lung

AnnonJob J

job	age	zip
lawyer	54	98125
cashier	20	98120

P ⟕ J

P.age	P.zip	P.disease	J.job	J.age	J.zip
54	98125	heart	lawyer	54	98125
20	98120	flu	cashier	20	98120
33	98120	lung	null	null	null

SOME EXAMPLES

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)
```

Name of supplier of parts with size greater than 10

$\pi_{sname}(Supplier \bowtie Supply \bowtie (\sigma_{psize>10} (Part))$

Name of supplier of red parts or parts with size greater than 10

$\pi_{sname}(Supplier \bowtie Supply \bowtie (\sigma_{psize>10} (Part) \cup \sigma_{pcolor='red'} (Part)))$

$\pi_{sname}(Supplier \bowtie Supply \bowtie (\sigma_{psize>10 \vee pcolor='red'} (Part)))$

Can be represented as trees as well

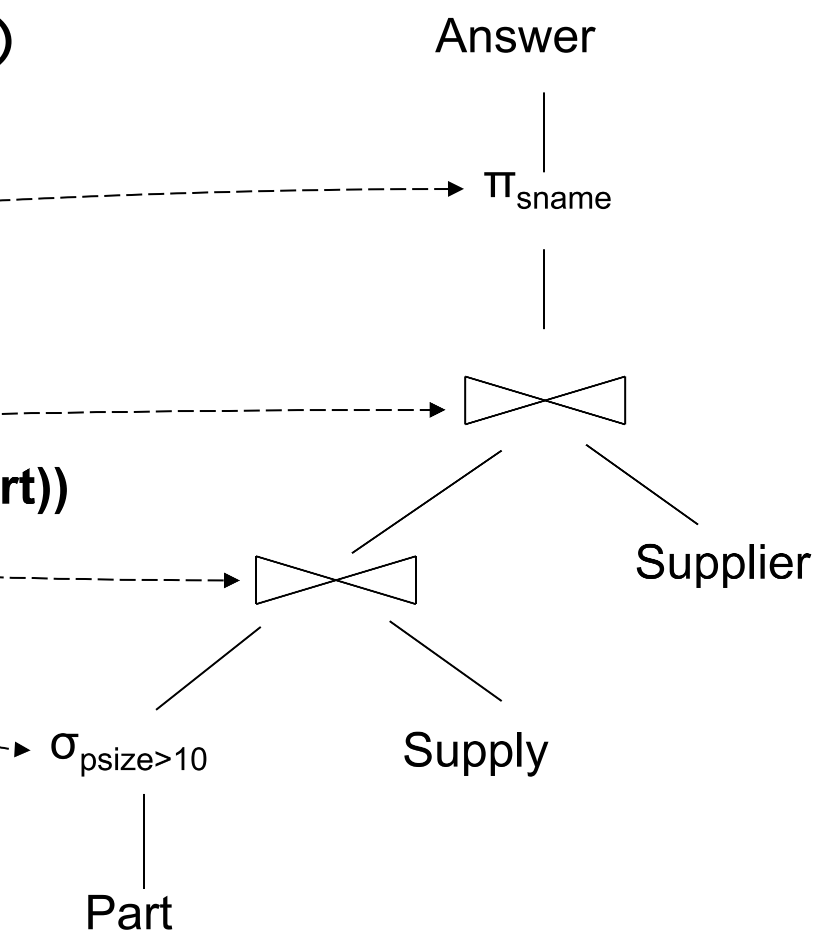
REPRESENTING RA QUERIES AS TREES

Supplier(<u>sno</u>,sname,scity,sstate)

Part(<u>pno</u>,pname,psize,pcolor)

Supply(<u>sno,pno</u>,qty,price)

$$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10}\ (\text{Part})))$$



RELATIONAL ALGEBRA OPERATORS

Union $\cup$, ~~intersection $\cap$,~~ difference -

Selection σ

Projection π

Cartesian product X, join $\bowtie$

(Rename ρ)

Duplicate elimination δ

Grouping and aggregation γ

Sorting τ

RA

Extended RA

All operators take in 1 or more relations as inputs and return another relation

EXTENDED RA: OPERATORS ON BAGS

Duplicate elimination δ

- Turns bags into sets (no other arguments)

Grouping γ

- Takes in relation and a list of grouping operations (e.g., aggregates). Returns a new relation.
- Can also perform renames at the same time

Sorting τ

- Takes in a relation, a list of attributes to sort on, and an order. Returns a new relation.

USING EXTENDED RA OPERATORS

```
SELECT city, sum(quantity)
FROM sales
GROUP BY city
HAVING count(*) > 100
```

Answer

$\Pi_{\text{city, q}}$

T2(city,q,c)

$\sigma_{\text{c > 100}}$

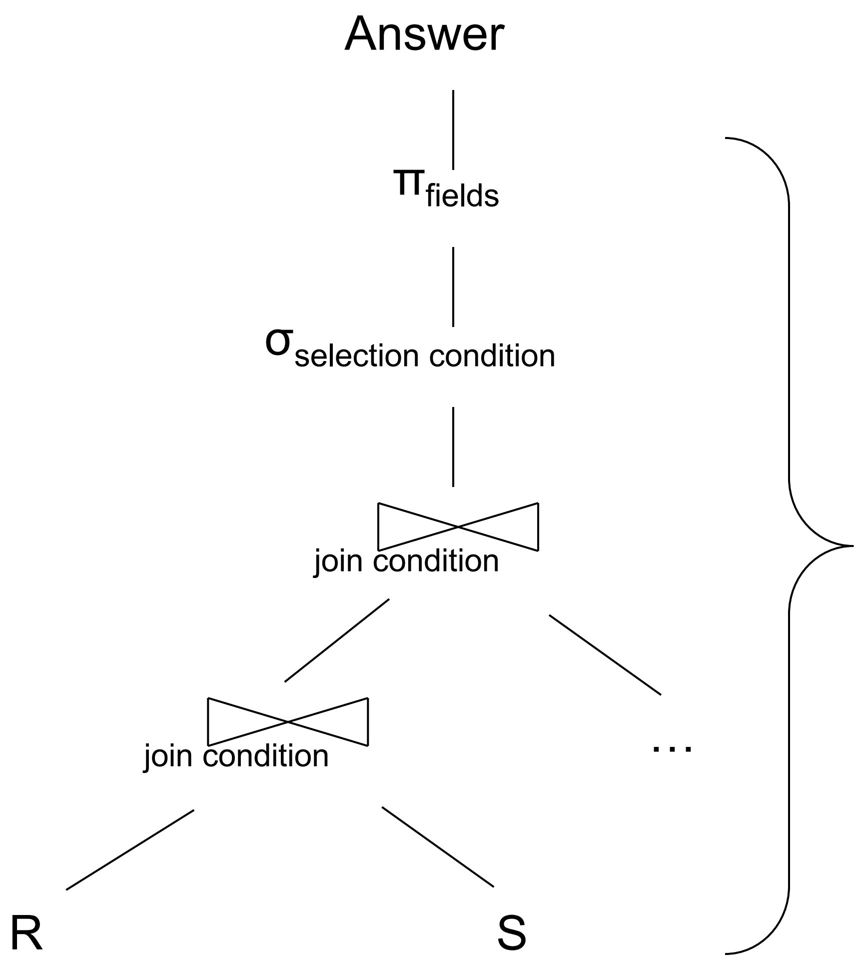
T1(city,q,c)

$\gamma_{\text{city, sum(quantity)}\rightarrow\text{q, count(*)}\rightarrow\text{c}}$

sales(product, city, quantity)

T1, T2 = temporary tables

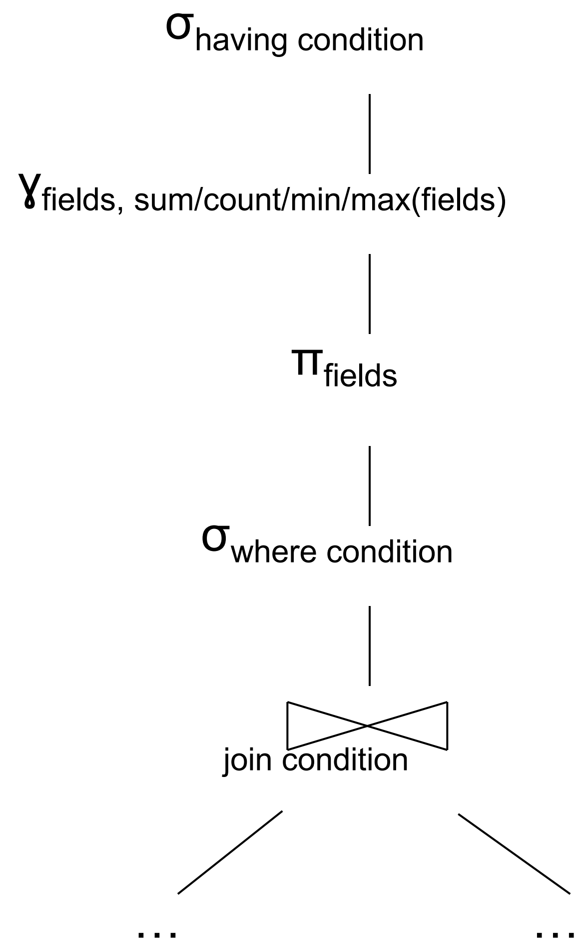
TYPICAL PLAN FOR A QUERY (1/2)




```
SELECT fields
FROM R, S, …
WHERE condition
```

SELECT-PROJECT-JOIN Query

TYPICAL PLAN FOR A QUERY (1/2)




```
SELECT fields
FROM R, S, …
WHERE condition
GROUP BY fields
HAVING condition
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

HOW ABOUT SUBQUERIES?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
  and not exists
  (SELECT *
   FROM Supply P
   WHERE P.sno = Q.sno
         and P.price > 100)
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

HOW ABOUT SUBQUERIES?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and not exists
  (SELECT *
   FROM Supply P
   WHERE P.sno = Q.sno
         and P.price > 100)
```

Correlation !

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

HOW ABOUT SUBQUERIES?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and not exists
  (SELECT *
   FROM Supply P
   WHERE P.sno = Q.sno
         and P.price > 100)
```

De-Correlation

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and Q.sno not in
  (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

HOW ABOUT SUBQUERIES?

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

Un-nesting

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = 'WA')
    EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and Q.sno not in
  (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

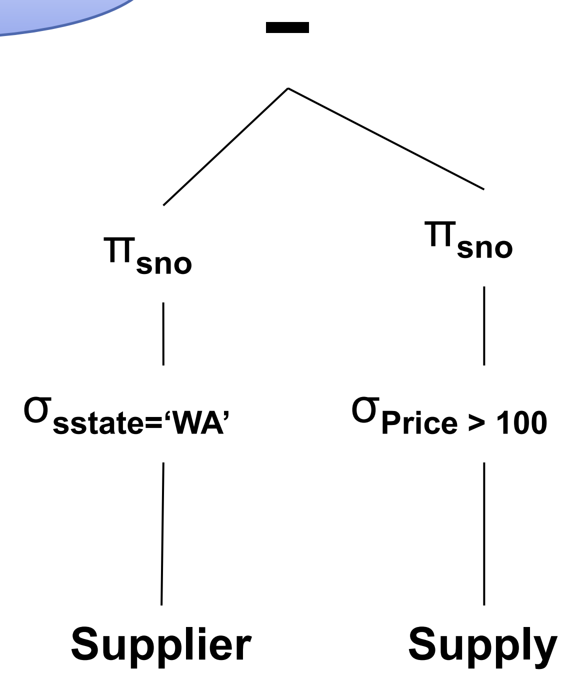
EXCEPT = set difference

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

HOW ABOUT SUBQUERIES?

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = 'WA')
    EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

Finally…



SUMMARY OF RA AND SQL

SQL = a declarative language where we say *what* data we want to retrieve

RA = an algebra where we say *how* we want to retrieve the data

Theorem: SQL and RA can express exactly the same class of queries

RDBMS translate SQL → RA, then optimize RA

SUMMARY OF RA AND SQL

SQL (and RA) cannot express ALL queries that we could write in, say, Java

Example:

- Parent(p,c): find all descendants of ‘Alice’
- No RA query can compute this!
- This is called a *recursive query*

Next lecture: Datalog is an extension that can compute recursive queries