Introduction to Database Systems
CSE 414

Lecture 7: SQL Wrap-up and Relational Algebra

Announcements

- Additional Office Hours and room changes
  – Website calendar is up-to-date

- Check email for Microsoft Azure invite
  "Action required: Accept your Azure lab assignment"

FWGHOS

Subqueries

- A subquery is a SQL query nested inside a larger query
- Such inner-outer queries are called nested queries
- A subquery may occur in:
  - A `SELECT` clause
    - Must return single value
  - A `FROM` clause
    - Can return multi-valued relation
  - A `WHERE` clause
- Rule of thumb: avoid nested queries when possible
  - But sometimes it's impossible, as we will see

Subqueries in FROM

Sometimes we need to compute an intermediate table only to use it later in a SELECT-FROM-WHERE

- Option 1: use a subquery in the FROM clause
- Option 2: use the WITH clause
  - See textbook for details

```
Product (pname,  price, cid)
Company (cid, cname, city)
```

2. Subqueries in FROM

```
SELECT X.pname
FROM (SELECT *
      FROM Product AS Y
      WHERE price > 20) as X
WHERE X.price < 500
```

=

A subquery whose result we called myTable

```
WITH myTable AS (SELECT * FROM Product AS Y WHERE price > 20)
  SELECT X.pname
  FROM myTable as X
  WHERE X.price < 500
```

Subqueries in WHERE

- SELECT ……….. WHERE EXISTS (sub);
- SELECT ……….. WHERE NOT EXISTS (sub);
- SELECT ……….. WHERE attribute IN (sub);
- SELECT ……….. WHERE attribute NOT IN (sub);
- SELECT ……….. WHERE attribute > ANY (sub);
- SELECT ……….. WHERE attribute > ALL (sub);

```
Product (pname,  price, cid)
Company (cid, cname, city)
```

Monotone Queries

- Definition A query Q is monotone if:
  - Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples

Monotone Queries

- Theorem: If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.

Monotone Queries

- Theorem: If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.

- Proof. We use the nested loop semantics: if we insert a tuple in a relation R_i, this will not remove any tuples from the answer

```
SELECT a₁, a₂, …, aₖ
FROM   R₁ AS x₁, R₂ AS x₂, …, Rₙ AS xₙ
WHERE  Conditions
```

```
for x₁ in R₁ do
  for x₂ in R₂ do
    …
      for xₙ in Rₙ do
        if Conditions
          output (a₁,…,aₖ)
```

```
Product (pname,  price, cid)
Company (cid, cname, city)
```

Monotone Queries

- The query:

Find all companies s.t. all their products have price < 200
is not monotone

```
Product (pname,  price, cid)
Company (cid, cname, city)
```

Monotone Queries

- The query:

Find all companies s.t. <u>all</u> their products have price < 200
is not monotone

pname	price	cid
Gizmo	19.99	c001

cid	cname	city
c001	Sunworks	Bonn

⇒

cname
Sunworks

```
Product (pname,  price, cid)
Company (cid, cname, city)
```

Monotone Queries

- The query:

Find all companies s.t. <u>all</u> their products have price < 200
is not monotone

pname	price	cid
Gizmo	19.99	c001

cid	cname	city
c001	Sunworks	Bonn

cname
Sunworks

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c001

cid	cname	city
c001	Sunworks	Bonn

cname

- <u>Consequence</u>: If a query is not monotonic, then we cannot write it as a SELECT-FROM-WHERE query without nested subqueries

Queries that must be nested

- Queries with universal quantifiers or with negation

Queries that must be nested

- Queries with universal quantifiers or with negation

- Queries that use aggregates in certain ways
  - `sum(..)` and `count(*)` are NOT monotone, because they do not satisfy set containment
  - `select count(*) from R` is not monotone!

SQL Idioms

```
Product (pname,  price, cid)
Company (cid, cname, city)
```

Finding Witnesses

For each city, find the most expensive product made in that city

```
Product (pname, price, cid)
Company (cid, cname, city)
```

Finding Witnesses

For each city, find the most expensive product made in that city

Finding the maximum price is easy…

```
SELECT x.city, max(y.price)
FROM   Company x, Product y
WHERE  x.cid = y.cid
GROUP BY x.city;
```

But we need the *witnesses*, i.e., the products with max price

Product (<u>pname</u>, price, cid)
Company (<u>cid</u>, cname, city)

Finding Witnesses

To find the witnesses, compute the maximum price in a subquery (in FROM or in WITH)

```sql
WITH CityMax AS
  (SELECT x.city, max(y.price) as maxprice
   FROM Company x, Product y
   WHERE x.cid = y.cid
   GROUP BY x.city)
SELECT DISTINCT u.city, v.pname, v.price
FROM Company u, Product v, CityMax w
WHERE u.cid = v.cid
      and u.city = w.city
      and v.price = w.maxprice;
```

```
Product (pname,  price, cid)
Company (cid, cname, city)
```

Finding Witnesses

To find the witnesses, compute the maximum price in a subquery (in FROM or in WITH)

```
SELECT DISTINCT u.city, v.pname, v.price
FROM Company u, Product v,
     (SELECT x.city, max(y.price) as maxprice
      FROM Company x, Product y
      WHERE x.cid = y.cid
      GROUP BY x.city) w
WHERE u.cid = v.cid
      and u.city = w.city
      and v.price = w.maxprice;
```

```
Product (pname,  price, cid)
Company (cid, cname, city)
```

Finding Witnesses

There is a more concise solution here:

```
SELECT u.city, v.pname, v.price
FROM Company u, Product v, Company x, Product y
WHERE u.cid = v.cid and u.city = x.city
and x.cid = y.cid
GROUP BY u.city, v.pname, v.price
HAVING v.price = max(y.price)
```

SQL: Our first language for the relational model

- Projections
- Selections
- Joins (inner and outer)
- Inserts, updates, and deletes
- Aggregates
- Grouping
- Ordering
- Nested queries

Relational Algebra

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 implementation converts a SQL query to RA in order to execute it
- An RA expression is called a *query plan*

Why study another relational query language?

- RA is how SQL is implemented in DBMS
  - We will see more of this in a few weeks

- RA opens up opportunities for *query optimization*

Basics

- Relations and attributes
- Functions that are applied to relations
  – Return relations

    R2 = σ (R1)
  – Can be composed together

    R3 = π (σ (R1))
  – 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

$\sigma_{\text{Salary} > 40000}$ (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

$\pi_{Name,Salary}$ (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

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

zip	disease
98125	flu
98125	heart
98120	lung
98120	heart

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

Renaming

- Changes the schema, not the instance

$$\boxed{\rho_{B1,\ldots,Bn}\ (R)}$$

- Example:
  - Given Employee(Name, SSN)
  - $\rho_{N,\ S}$(Employee) $\rightarrow$ Answer(N, S)

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 ⋈ 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

Natural Join

- Given schemas R(A, B, C, D), S(A, C, E), what is the schema of R ⋈ S ?

- Given R(A, B, C), S(D, E), what is R ⋈ S?

- Given R(A, B), S(A, B), what is R ⋈ S?

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

Theta Join

- A join that involves a predicate

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

- Here θ can be any condition
- No projection in this case!
- For our voters/patients example:

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

Equijoin

- A theta join where θ is an equality predicate

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

- 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 ⋈ 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.diseas e	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

```
Project[sname](Supplier Join[sno=sno]
                (Supply Join[pno=pno] (Select[psize>10](Part))))
```

Using symbols:

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

Can be represented as trees as well

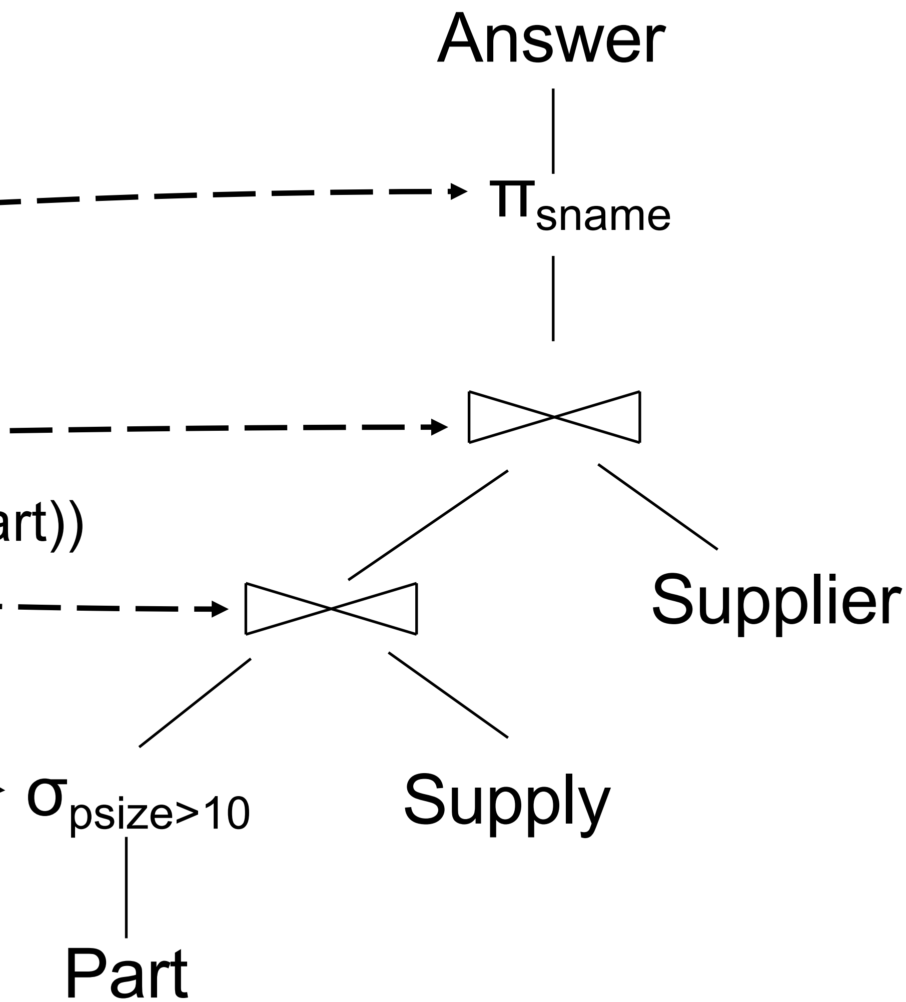
Representing RA Queries as Trees

`Supplier(sno,sname,scity,sstate)`

`Part(pno,pname,psize,pcolor)`

`Supply(sno,pno,qty,price)`

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



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

```
Project[sname](Supplier Join[sno=sno]
              (Supply Join[pno=pno] (Select[psize>10](Part))))
```

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

```
Project[sname](Supplier Join[sno=sno]
              (Supply Join[pno=pno]
              ((Select[psize>10](Part)) Union
                                       (Select[pcolor='red'](Part)))
```

```
Project[sname](Supplier Join[sno=sno] (Supply Join[pno=pno]
                               (Select[psize>10 OR pcolor='red'](Part))))
```

Can be represented as trees as well

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

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

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

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

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

Can be represented as trees as well

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

Extended RA: Operators on Bags

- Duplicate elimination δ
- Grouping γ
  - Takes in relation and a list of grouping operations (e.g., aggregates). Returns a new relation.
- 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

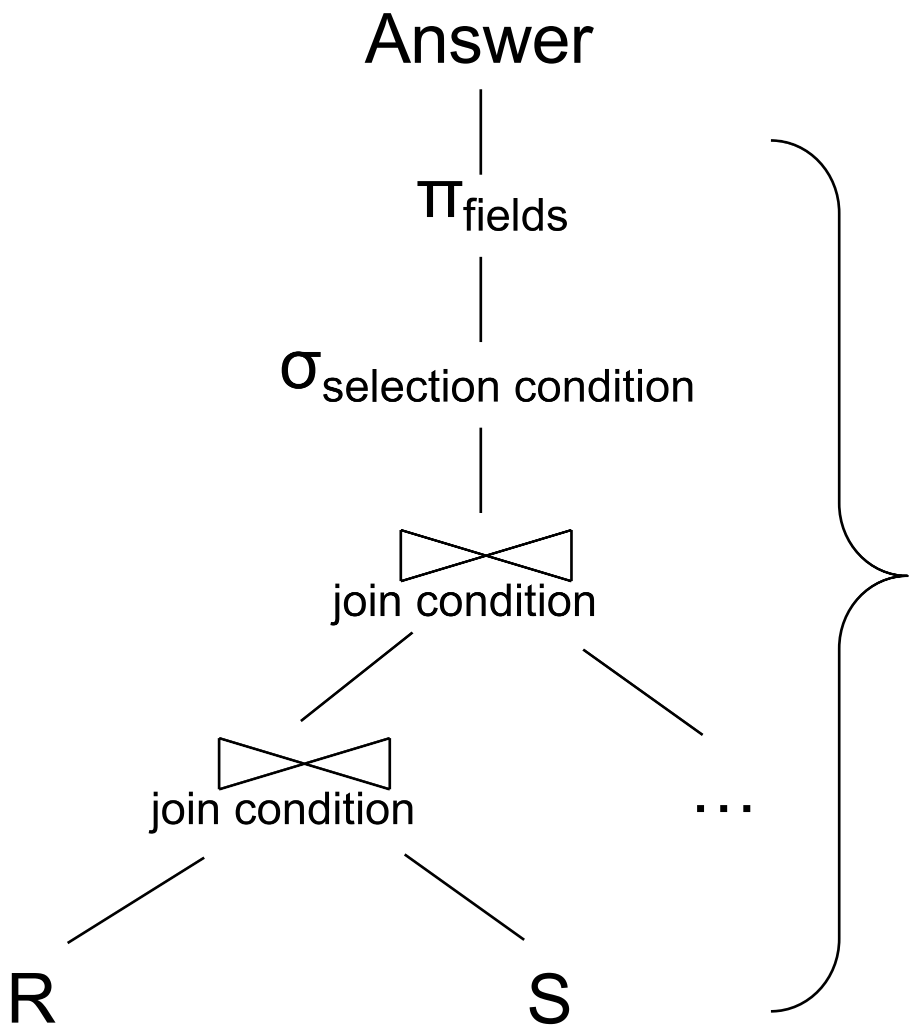
$\Pi_{\text{city, q}}$

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

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

Sales(product, city, quantity)

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)

$\sigma_{\text{having condition}}$

$\gamma_{\text{fields, sum/count/min/max(fields)}}$

π_{fields}

$\sigma_{\text{where condition}}$

$\bowtie_{\text{join condition}}$

… …

```
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?

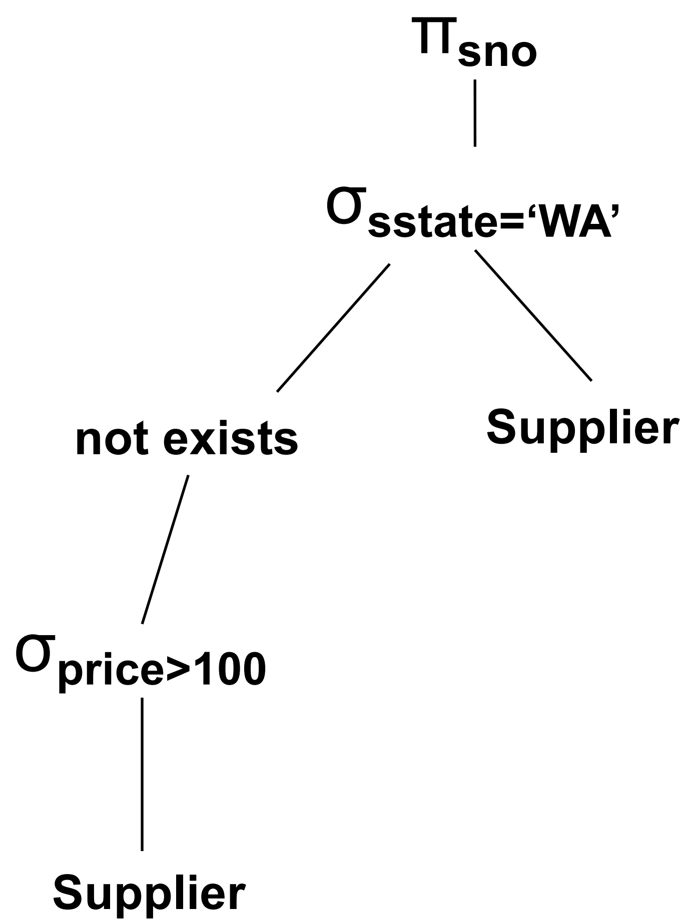
```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and not exists
  (SELECT *
   FROM Supply P
   WHERE P.sno = Q.sno
         and P.price > 100)
```

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

How about Subqueries?

Option 1: create nested plans

```
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)
```

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

How about Subqueries?

Un-nesting

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

EXCEPT = set difference

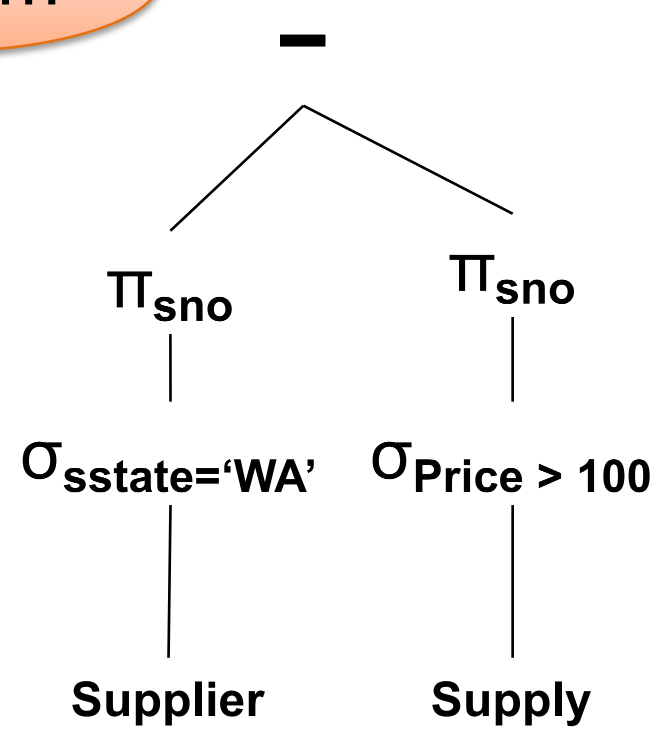
```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and Q.sno not in
  (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

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