

Introduction to Database Systems

CSE 414

Lecture 8: Relational Algebra

Announcements

- HW3 is out – due Friday
 - git pull upstream master
 - Make sure you have email from Microsoft Azure and log in
- Web quiz 2 due tonight

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
$$R2 = \sigma (R1)$$
 - Can be composed together
$$R3 = \pi (\sigma (R1))$$
 - 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}, . . .

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 \cup , intersection \cap , difference $-$

Use with NOT EXISTS

- Selection $\sigma_{\text{condition}}$ removes tuples that don't meet condition WHERE or HAVING

- Projection π selects specific attributes to be passed to next relationship
SELECT

- Cartesian product \times , join



LEFT OUTER JOIN

Natural join = join on equality
amongst same named attributes
remove duplicate attribute names
Equijoin: equality condition
Theta-Join: join on condition (no projection like there is in natural)

- (Rename ρ) Aliasing

- Duplicate elimination δ DISTINCT

- Grouping and aggregation γ

attribute1, ..., sum(c) \rightarrow s, etc
even aggregates in selection occur here

- Sorting $\tau_{\text{attribute}}$ ORDER BY

RA

Extended RA

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

Union and Difference

$$\begin{array}{l} R_1 \cup R_2 \\ R_1 - R_2 \end{array}$$

Only make sense if R_1 , R_2 have the same schema

What do they mean over bags ?

What about Intersection ?

- Derived operator using minus

$$R1 \cap R2 = R1 - (R1 - R2)$$

- Derived using join

$$R1 \cap R2 = R1 \bowtie R2$$

Selection

- Returns all tuples which satisfy a condition

$$\sigma_c(R)$$

- Examples
 - $\sigma_{\text{Salary} > 40000}(\text{Employee})$
 - $\sigma_{\text{name} = \text{"Smith"}}(\text{Employee})$
- The condition c can be $=, <, \leq, >, \geq, \neq$ 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

$$\pi_{A_1, \dots, A_n}(R)$$

- Example: project social-security number and names:
 - $\pi_{\text{SSN}, \text{Name}}(\text{Employee}) \rightarrow \text{Answer}(\text{SSN}, \text{Name})$

Different semantics over sets or bags! Why?

Employee

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

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

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

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

zip	disease
98125	flu
98125	heart
98120	lung
98120	heart

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

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

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

zip	disease
98125	heart
98120	heart

Cartesian Product

- Each tuple in R1 with each tuple in R2

$$R1 \times R2$$

- 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

$$\rho_{B_1, \dots, B_n} (R)$$

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

Natural Join

$$R1 \bowtie R2$$

- 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

$\mathbf{R} \bowtie \mathbf{S} =$

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

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 $R \bowtie S$?

A B C D E

- Given $R(A, B, C)$, $S(D, E)$, what is $R \bowtie S$?

EMPTY; no common attribute names

- Given $R(A, B)$, $S(A, B)$, what is $R \bowtie S$?

A B

AnonPatient (age, zip, disease)

Voters (name, age, zip)

Theta Join

- A join that involves a predicate

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

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

$$P \bowtie P.zip = V.zip \text{ and } P.age \geq V.age - 1 \text{ and } P.age \leq V.age + 1 \quad V$$

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

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

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

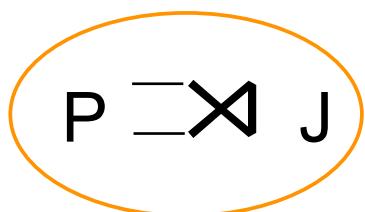
Outer Join Example

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



symbolizes a left outer
join by leaving left side
of symbol open

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

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

Using symbols:

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

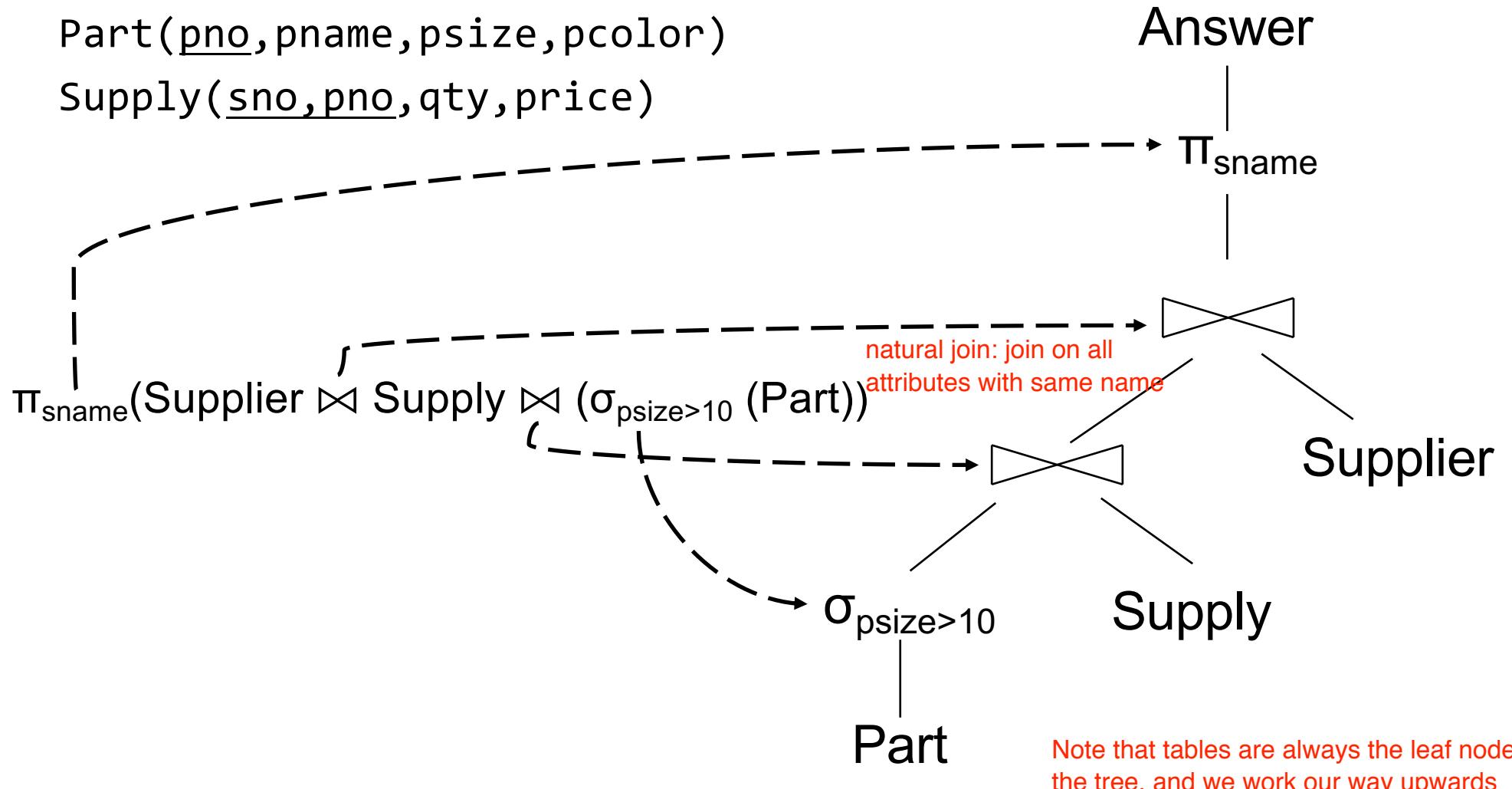
Can be represented as trees as well

Representing RA Queries as Trees

$\text{Supplier}(\underline{sno}, \text{sname}, \text{scity}, \text{sstate})$

$\text{Part}(\underline{pno}, \text{pname}, \text{psize}, \text{pcolor})$

$\text{Supply}(\underline{sno}, \underline{pno}, \text{qty}, \text{price})$



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

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

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

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

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

Relational Algebra Operators

- Union \cup , intersection \cap , difference $-$
- Selection σ
WHERE (filter out certain tuples based on some given conditions)
- Projection Π
(only select specific attributes/columns)
- Cartesian product \times , join \bowtie
- (Rename ρ) aliasing
- Duplicate elimination δ DISTINCT
- Grouping and aggregation γ
Group by and SUM/Count/max etc
- Sorting τ (ORDER BY)

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.

Grouping

- Specify groups and aggregates

$$\gamma_{A1, \dots, An, \text{sum/max}(B1) \dots} (R)$$

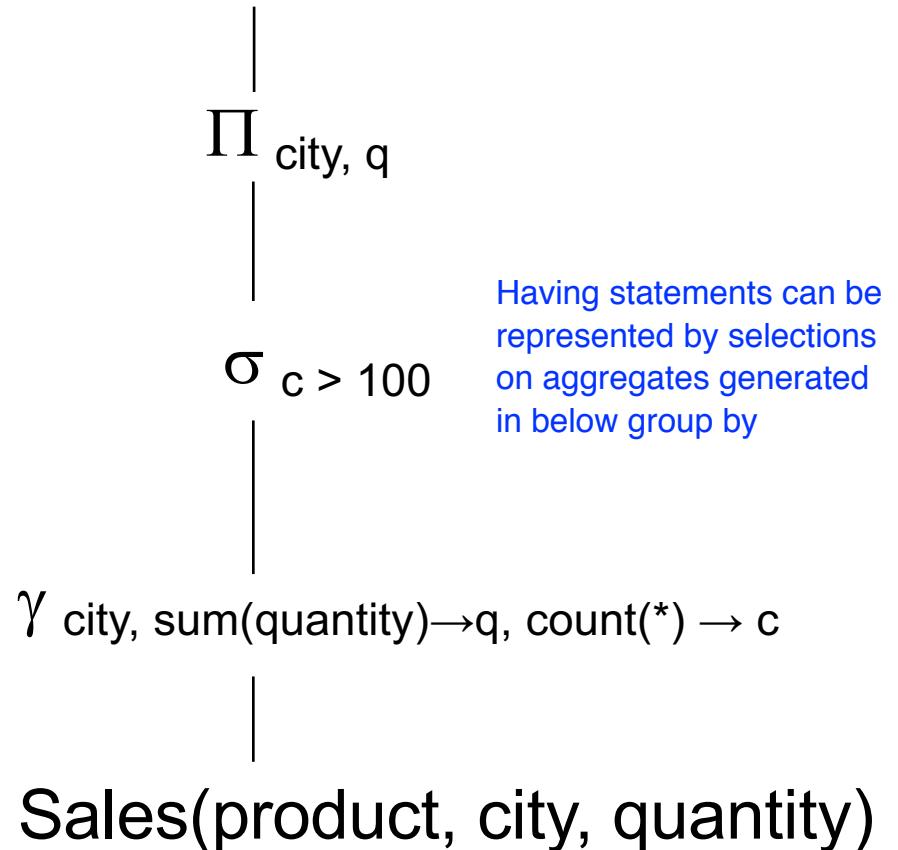
- Example: project social-security number and names:
- Output is like project: only output is attributes in the subscript
- Can also rename: $\gamma_{A, \text{count}(B)} \rightarrow \text{count} (R)$

Using Extended RA Operators

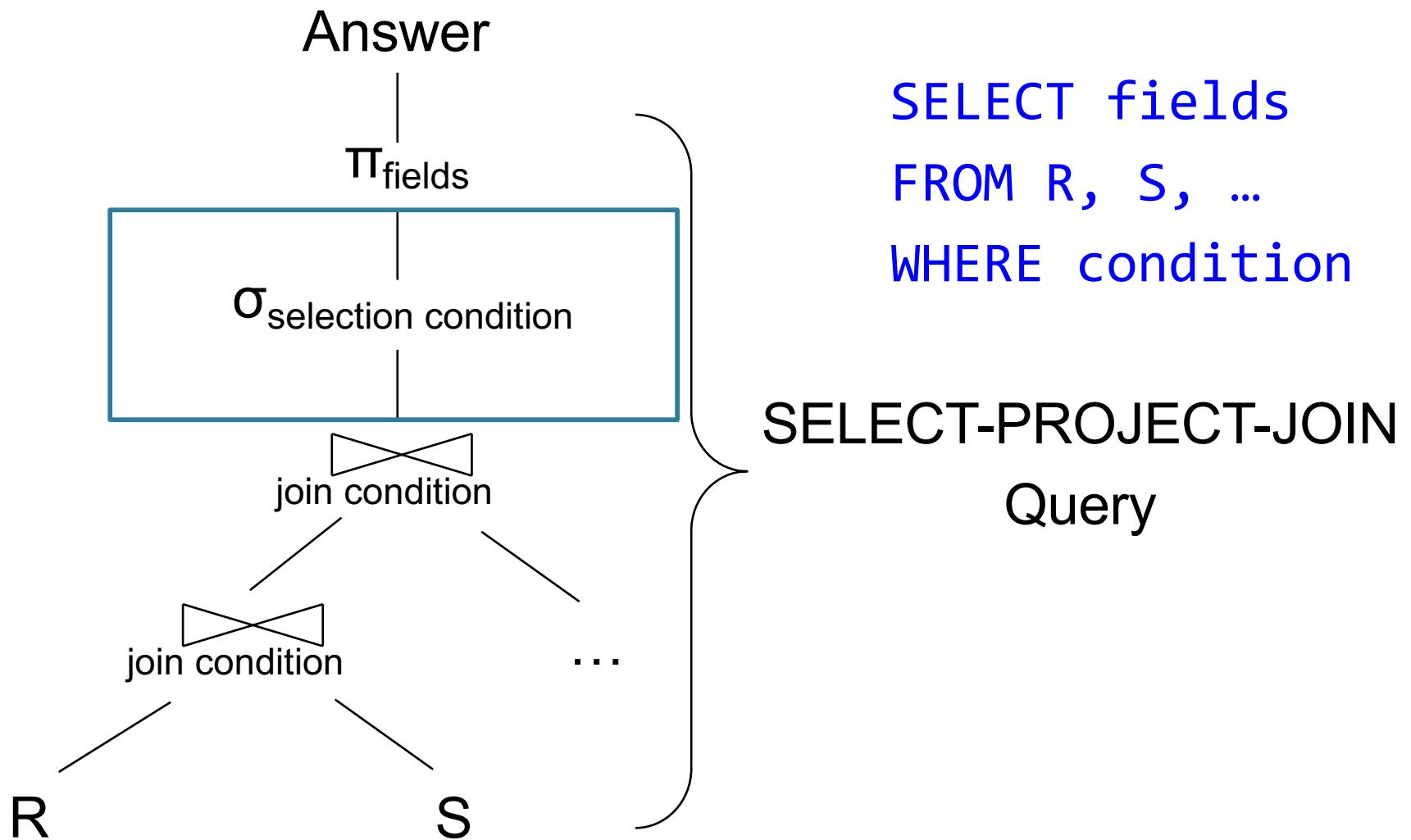
```
SELECT city, sum(quantity)
FROM Sales
GROUP BY city
HAVING count(*) > 100
```

grouping and aggregation
MUST happen together.

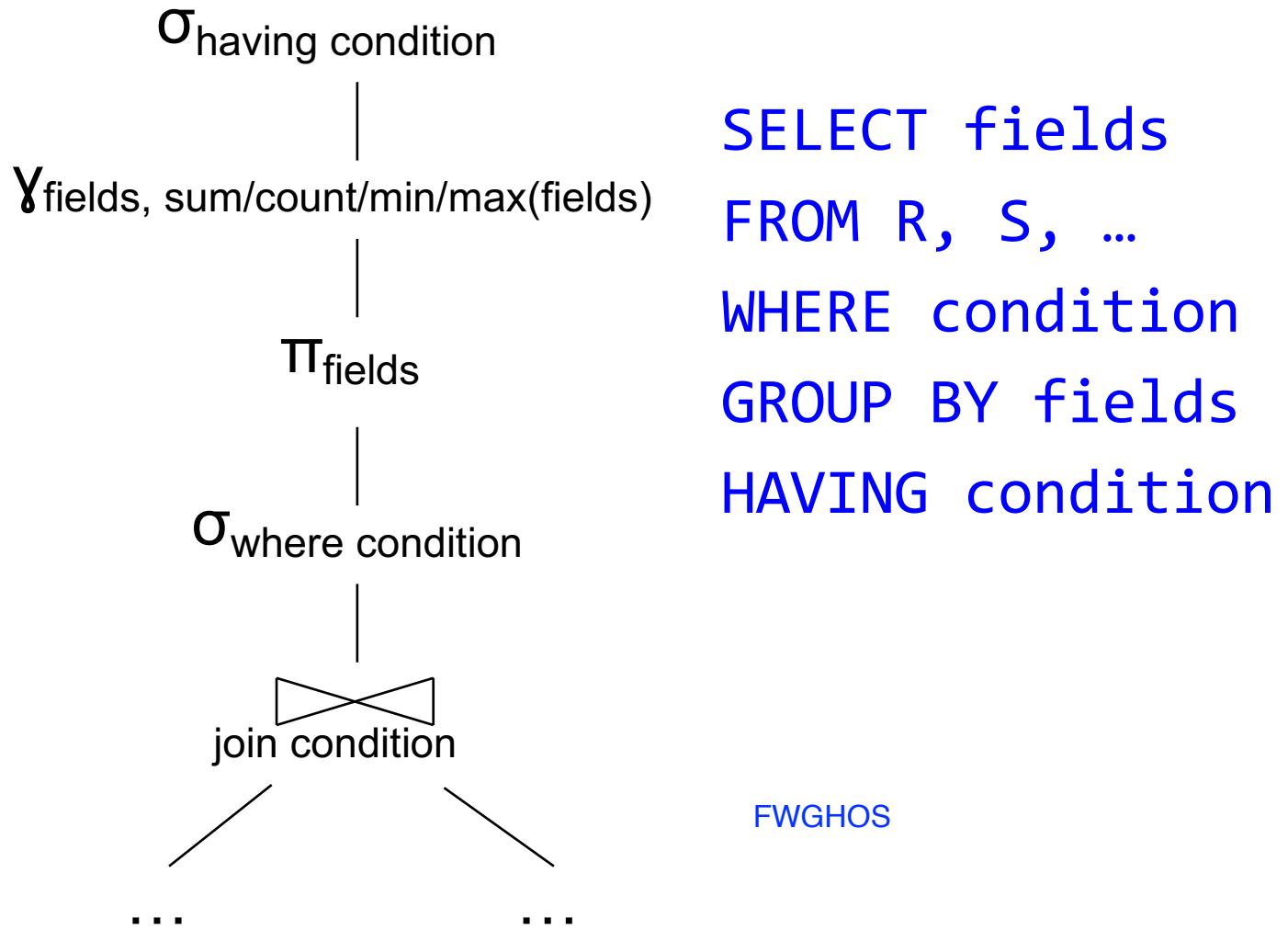
Answer



Typical Plan for a Query (1/2)



Typical Plan for a Query (1/2)



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

How about Subqueries?

Return all suppliers in WA that sell no products greater than \$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)
```

Return all suppliers in WA that sell no products greater than \$100

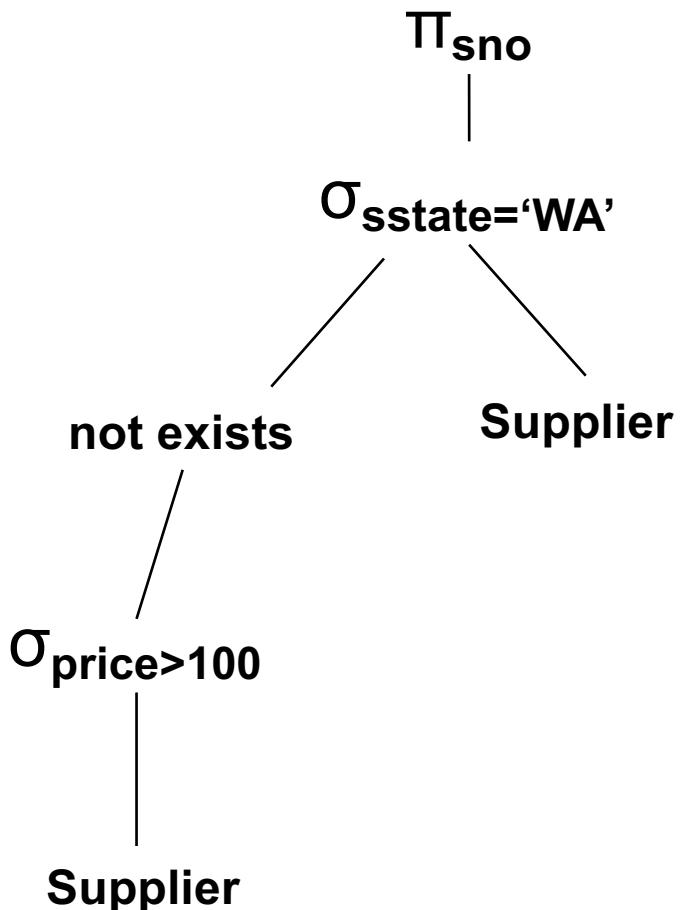
$\text{Supplier}(\underline{sno}, \underline{sname}, \underline{scity}, \underline{sstate})$
 $\text{Part}(\underline{pno}, \underline{pname}, \underline{psize}, \underline{pcolor})$
 $\text{Supply}(\underline{sno}, \underline{pno}, \underline{\text{price}})$

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)
```

```
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

Two relations that return same type
of information (sno) and does set difference

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

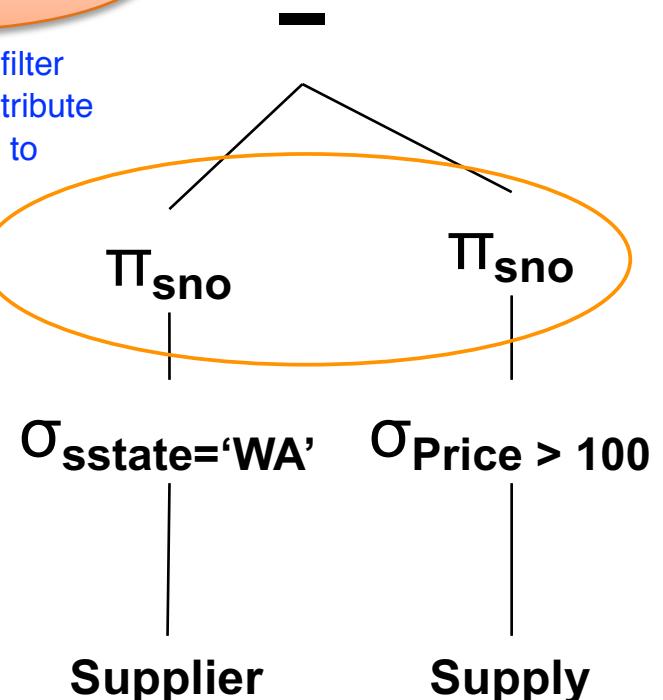
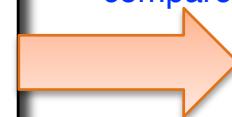
How about Subqueries?

```

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

Finally...

note that we filter
first by the attribute
we are going to
compare by



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

Class Overview

- Unit 1: Intro
- Unit 2: Relational Data Models and Query Languages
 - Data models, SQL, Relational Algebra, Datalog
- Unit 3: Non-relational data
- Unit 4: RDMBS internals and query optimization
- Unit 5: Parallel query processing
- Unit 6: DBMS usability, conceptual design
- Unit 7: Transactions

What is Datalog?

- Another query language for relational model
 - Designed in the 80's
 - Simple, concise, elegant
 - Extends relational queries with *recursion*
- Today is a hot topic:
 - Souffle (we will use in HW4)
 - Eve <http://witheve.com/>
 - Differential datalog
<https://github.com/frankmcsherry/differential-dataflow>
 - Beyond databases in many research projects:
network protocols, static program analysis

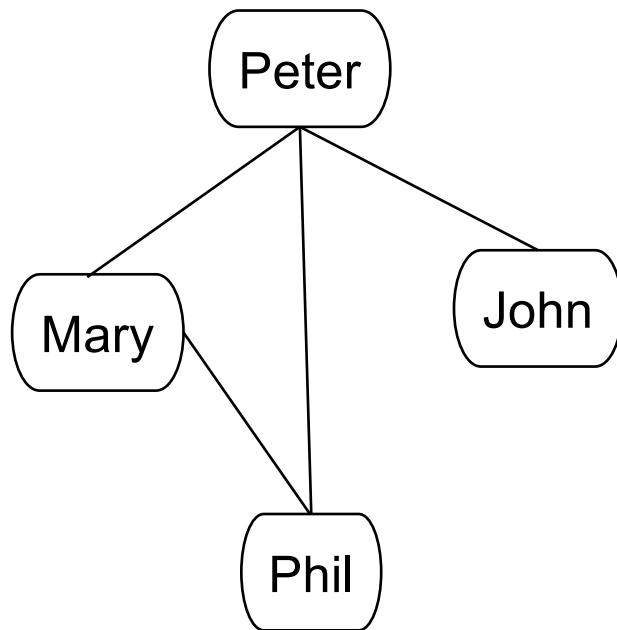


Soufflé

- Open-source implementation of Datalog DBMS
- Under active development
- Commercial implementations are available
 - More difficult to set up and use
- “sqlite” of Datalog
 - Set-based rather than bag-based
- Install in your VM
 - Run `sudo yum install souffle` in terminal
 - More details in upcoming HW4

Why bother with *yet* another relational query language?

Example: storing FB friends



Or

Person1	Person2	is_friend
Peter	John	1
John	Mary	0
Mary	Phil	1
Phil	Peter	1
...

As a graph

As a relation

We will learn the tradeoffs of different data models later this quarter

Compute your friends graph

p1	p2	isFriend
Peter	John	1
John	Mary	0
Mary	Phil	1
Phil	Peter	1
...

Friends(p1, p2, isFriend)

```
SELECT f.p2
FROM Friends as f
WHERE f.p1 = 'me' AND f.isFriend = 1
```

My own friends

```
SELECT f1.p2
FROM Friends as f1,
(SELECT f.p2
FROM Friends as f
WHERE f.p1 = 'me' AND
f.isFriend = 1) as f2
WHERE f1.p1 = f2.p2 AND
f1.isFriend = 1
```

My FoF

My FoFoF... My FoFoFoF...

Datalog allows us to write
recursive queries easily

CSE 414 - Autumn 2018

When does it end???



Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

← Schema

Datalog: Facts and Rules

Facts = tuples in the database

Rules = queries

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
.decl Actor(id:number, fname:symbol, lname:symbol)
.decl Casts(id:number, mid:number)
.decl Movie(id:number, name:symbol, year:number)

Actor(344759, 'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

Table declaration

Types in Souffle:
number
symbol (aka varchar)

Insert data

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :- Movie(x,y,z), z=1940.
```

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :- Movie(x,y,z), z=1940.
```

Find Movies made in 1940

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :- Movie(x,y,z), z=1940.
```

SQL

```
SELECT name  
FROM Movie  
WHERE year = 1940
```

Find Movies made in 1940

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :- Movie(x,y,z), z=1940.
```

id *name*
 ↑ ↓
 x y

Order of variable matters!

Find Movies made in 1940

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :- Movie(iDontCare,y,z),  
z=1940.
```

Find Movies made in 1940

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :- Movie(_,y,z), z=1940.
```

_ = “don’t care” variables

Find Movies made in 1940

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :- Movie(x,y,z), z=1940.
```

```
Q2(f,l) :- Actor(z,f,l), Casts(z,x),  
          Movie(x,y,1940).
```

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :- Movie(x,y,z), z=1940.
```

```
Q2(f,l) :- Actor(z,f,l), Casts(z,x),  
          Movie(x,y,1940).
```

Find Actors who acted in Movies made in 1940

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :- Movie(x,y,z), z=1940.
```

```
Q2(f,l) :- Actor(z,f,l), Casts(z,x),  
          Movie(x,y,1940).
```

```
Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),  
          Casts(z,x2), Movie(x2,y2,1940).
```

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :- Movie(x,y,z), z=1940.
```

```
Q2(f,l) :- Actor(z,f,l), Casts(z,x),  
          Movie(x,y,1940).
```

```
Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),  
          Casts(z,x2), Movie(x2,y2,1940).
```

Find Actors who acted in a Movie in 1940 and in one in 1910

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :- Movie(x,y,z), z=1940.
```

```
Q2(f,l) :- Actor(z,f,l), Casts(z,x),  
          Movie(x,y,1940).
```

```
Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),  
          Casts(z,x2), Movie(x2,y2,1940).
```

Extensional Database Predicates = EDB = Actor, Casts, Movie

Intensional Database Predicates = IDB = Q1, Q2, Q3

gradiance quiz 4

3)

$1 = \{3,6,3\}$

$2 = \{5,6,3\}$

$3 = \{1,5\}$

4)

projected order swaps C and B

5)

Answer is the list of all x,y from R that cannot be created by joining two tuples from R sharing a “middle”

i.e. joining (1,3) and (3,5) creates (1,5).

(1,4) and (5,6) cannot be joined in this way.

9)

Answer is the list of all joined tuples of R, joining by the middle element.