Database Management Systems
CSEP 544

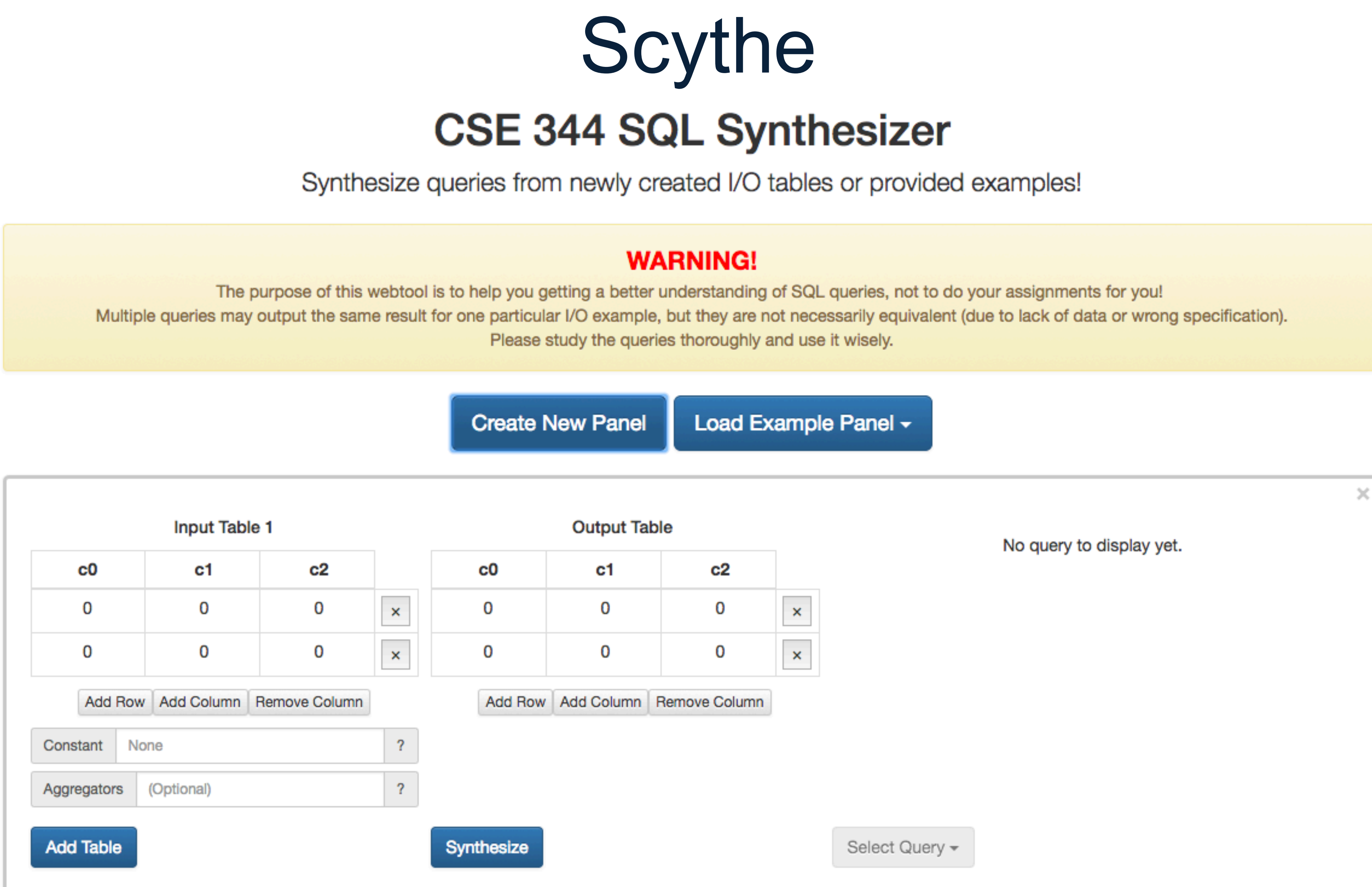
Lecture 3: SQL
Relational Algebra, and Datalog

Announcements

- HW2 due tonight (11:59pm)
- PA3 & HW3 released

HW3

- We will be using SQL Server in the cloud (Azure)
  - Same dataset
  - More complex queries ☺
- Logistics
  - You will receive an email from invites@microsoft.com to join the “Default Directory organization” --- accept it!
  - You are allocated $100 to use for this quarter
  - We will use Azure for two HW assignments
  - Use SQL Server Management Studio to access the DB
    - Installed on all CSE lab machines and VDI machines


Scythe
CSE 344 SQL Synthesizer
Synthesize queries from newly created I/O tables or provided examples!
WARNING!
The purpose of this webtool is to help you getting a better understanding of SQL queries, not to do your assignments for you!
Multiple queries may output the same result for one particular I/O example, but they are not necessarily equivalent (due to lack of data or wrong specification).
Please study the queries thoroughly and use it wisely.
Create New Panel
Load Example Panel
Input Table 1
c0
c1
c2
0
0
0
0
0
0
Add Row
Add Column
Remove Column
Constant
None
?
Aggregators
(Optional)
?
Add Table
Output Table
c0
c1
c2
0
0
0
0
0
0
Add Row
Add Column
Remove Column
Synthesize
No query to display yet.
Select Query

Plan for Today

- Wrap up SQL

- Study two other languages for the relational data model
  - Relational algebra
  - Datalog

Reading Assignment 2

- Normal form
- Compositionality of relations and operators

foo(...).bar(...)

t = foo(..)

= t.bar(..)

Review

- SQL
  - Selection
  - Projection
  - Join
  - Ordering
  - Grouping
  - Aggregates
  - Subqueries
- Query Evaluation

FWGHOS

`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

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c004
Camera	149.99	c003

Company

cid	cname	city
c002	Sunworks	Bonn
c001	DB Inc.	Lyon
c003	Builder	Lodtz

Q

pname	city
Gizmo	Lyon
Camera	Lodtz

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c004
Camera	149.99	c003
iPad	499.99	c001

Company

cid	cname	city
c002	Sunworks	Bonn
c001	DB Inc.	Lyon
c003	Builder	Lodtz

Q

pname	city
Gizmo	Lyon
Camera	Lodtz
iPad	Lyon

SQL Idioms

FWGHOS

Including Empty Groups

- In the result of a group by query, there is one row per group in the result

Count(*) is never 0

```
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.pname = y.product
GROUP BY x.manufacturer
```

Including Empty Groups

```
SELECT x.manufacturer, count(y.pid)
FROM Product x LEFT OUTER JOIN Purchase y
ON x.pname = y.product
GROUP BY x.manufacturer
```

Count(pid) is 0 when all pid's in the group are NULL

Purchase(pid, product, quantity, price)

GROUP BY vs. Nested Queries

```
SELECT    product, Sum(quantity) AS TotalSales
FROM      Purchase
WHERE     price > 1
GROUP BY  product
```

```
SELECT DISTINCT x.product, (SELECT Sum(y.quantity)
                            FROM  Purchase y
                            WHERE x.product = y.product
                              AND y.price > 1)
                            AS TotalSales
FROM  Purchase x
WHERE x.price > 1
```

Why twice ?

```
Author(login,name)
Wrote(login,url)
```

More Unnesting

Find authors who wrote ≥ 10 documents:

```
Author(login,name)
Wrote(login,url)
```

More Unnesting

Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

This is SQL by a novice

```
SELECT DISTINCT Author.name
FROM   Author
WHERE (SELECT count(Wrote.url)
       FROM Wrote
       WHERE Author.login=Wrote.login)
      >= 10
```

```
Author(login,name)
Wrote(login,url)
```

More Unnesting

Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

Attempt 2: using GROUP BY and HAVING

```
SELECT   Author.name
FROM     Author, Wrote
WHERE    Author.login=Wrote.login
GROUP BY Author.name
HAVING   count(wrote.url) >= 10
```

This is SQL by an expert

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

Finding Witnesses

To find the witnesses, compute the maximum price in a subquery

```
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

Or we can use a subquery in where clause

```
SELECT u.city, v.pname, v.price
FROM Company u, Product v
WHERE u.cid = v.cid
  and v.price >= ALL (SELECT y.price
                      FROM Company x, Product y
                      WHERE u.city=x.city
                      and x.cid=y.cid);
```

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

Class overview

- Data models
  - Relational: SQL, RA, and Datalog
  - NoSQL: SQL++
- RDMBS internals
  - Query processing and optimization
  - Physical design
- Parallel query processing
  - Spark and Hadoop
- Conceptual design
  - E/R diagrams
  - Schema normalization
- Transactions
  - Locking and schedules
  - Writing DB applications

Data models

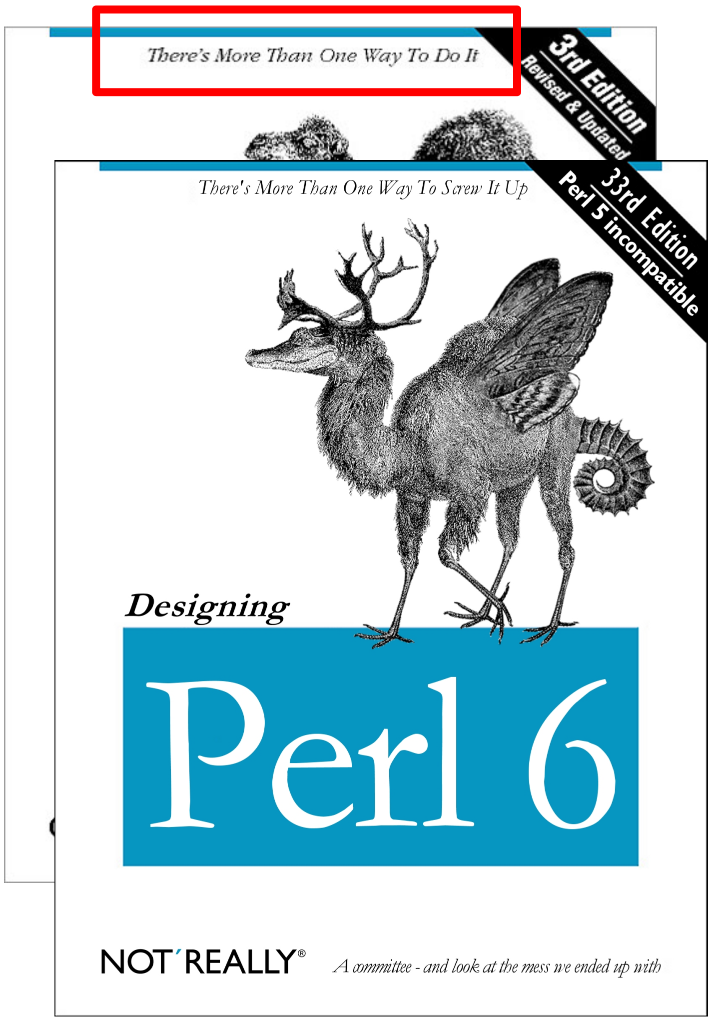
Query Processing

Using DBMS

Next: Relational Algebra

- Our second language for the relational model
  - Developed before SQL
  - Simpler syntax than SQL

Why bother with another language?



- Used extensively by DBMS implementations
  – As we will see in 2 weeks
- RA influences the design SQL

Relational Algebra

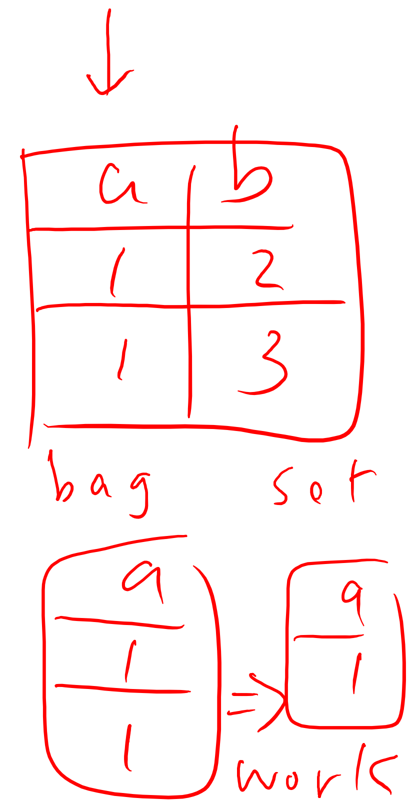
- In SQL we say *what* we want
- In RA we can express *how* to get it
- Set-at-a-time algebra, which manipulates relations
- Every RDBMS implementations converts a SQL query to RA in order to execute it

- An RA expression is also 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

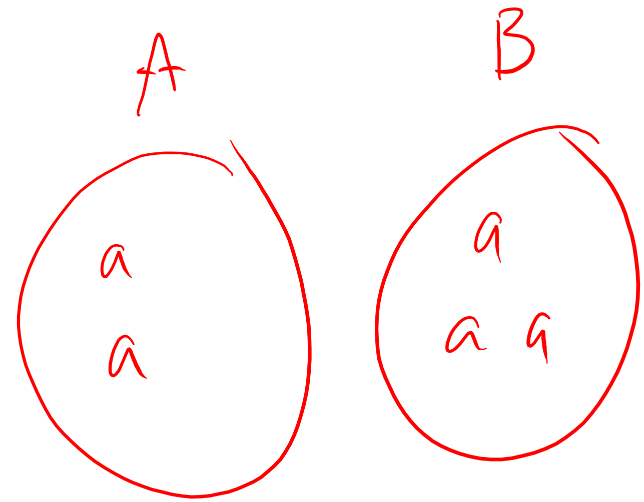
Extended RA

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

Union and Difference

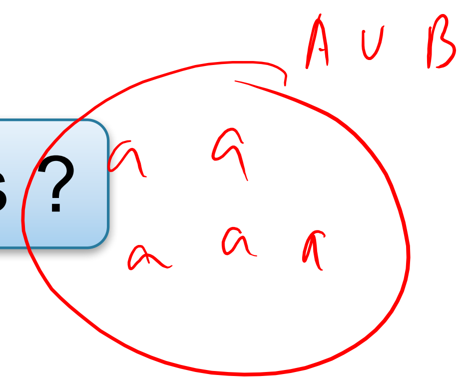
R1 ∪ R2

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

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

- Derived using join

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

Selection

- Returns all tuples which satisfy a condition

$$\sigma_c(R)$$

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

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

- 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?

Functional Composition of 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

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

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

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

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

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

Natural Join

$$R1 \bowtie R2$$

project, select

- 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 $\bowtie$ S ?

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

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

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

Theta Join

- A join that involves a predicate

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

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

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

Equijoin

- A theta join where θ is an equality predicate

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

- 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

J ⟖ P

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

LOJ ⟕
ROJ ⟖
FOJ ⟗

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

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

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

or ∧ and

Can be represented as trees as well

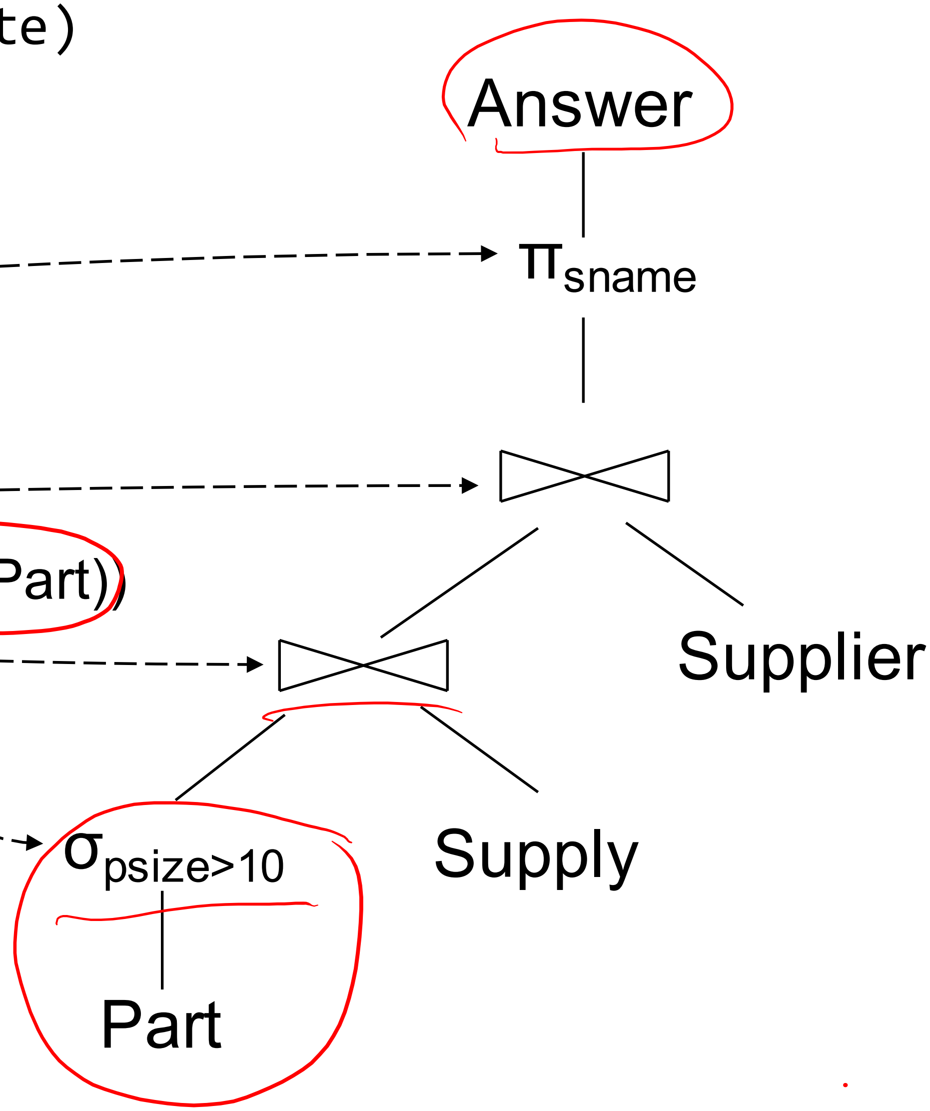
Representing RA Queries as Trees

Supplier(sno,sname,scity,sstate)

Part(pno,pname,psize,pcolor)

Supply(sno,pno,qty,price)

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



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

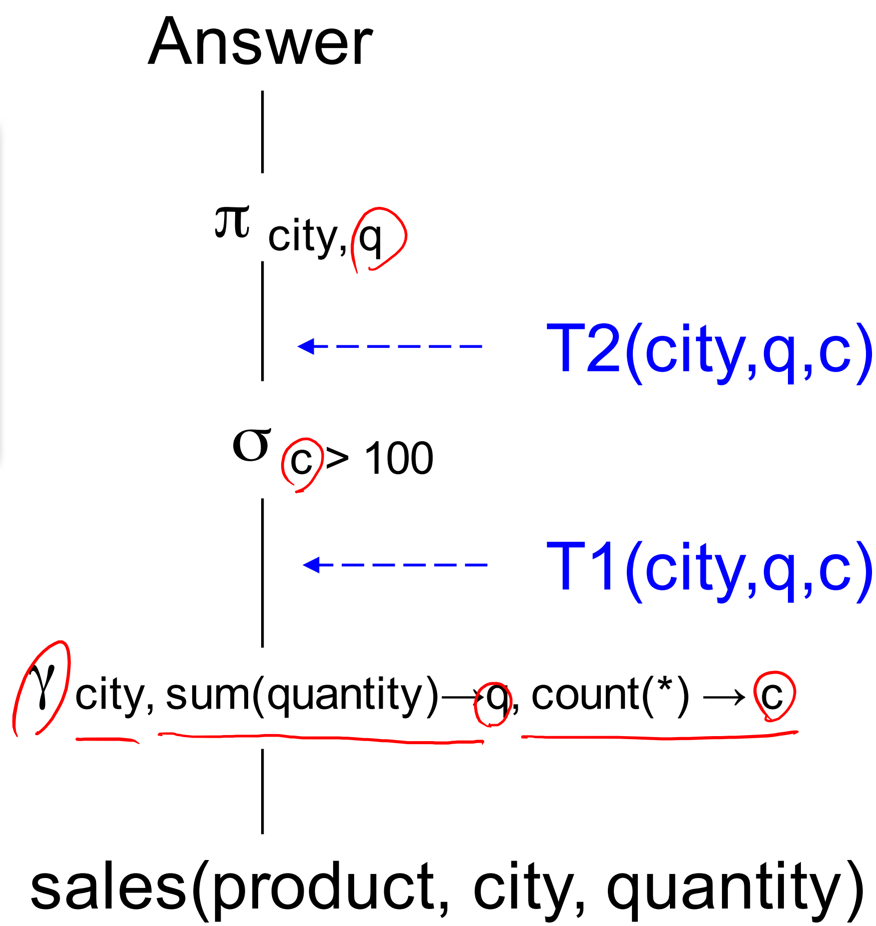
All operators take in 1 or more relations as inputs and return another relation

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.

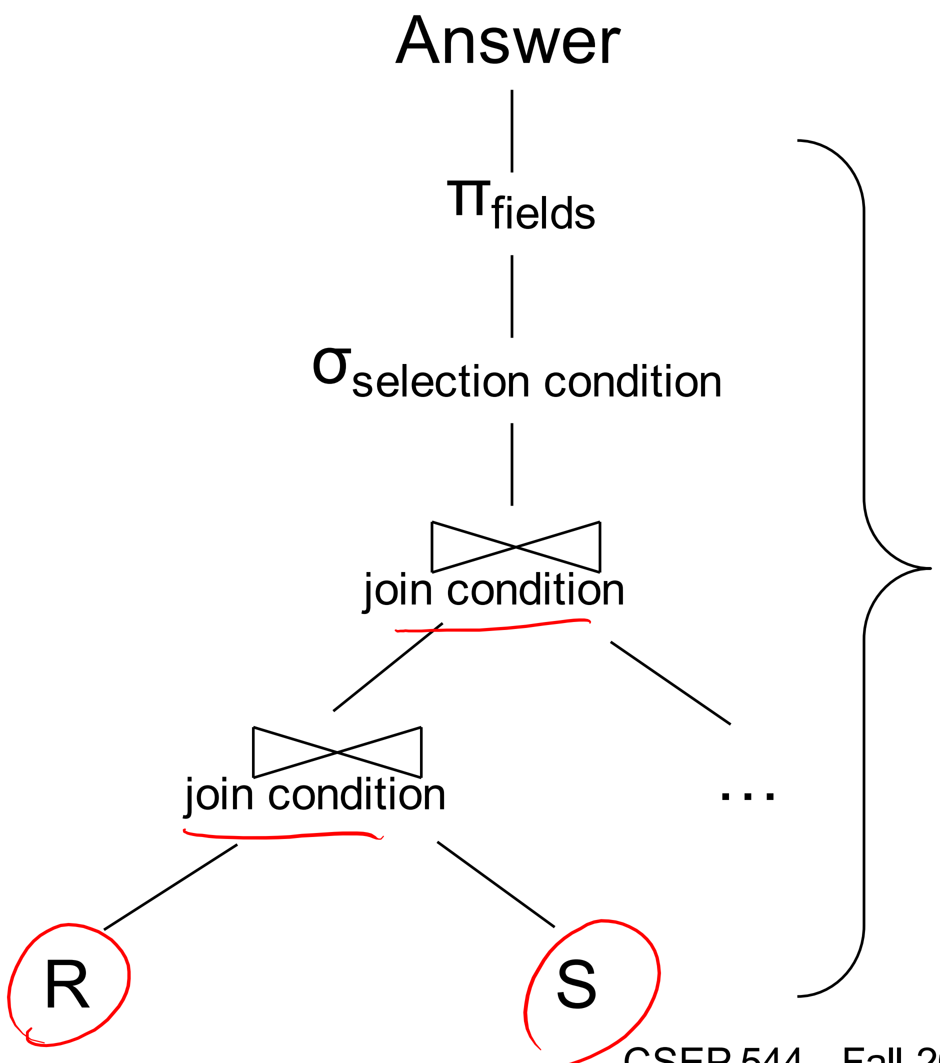
Using Extended RA Operators

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



T1, T2 = temporary tables

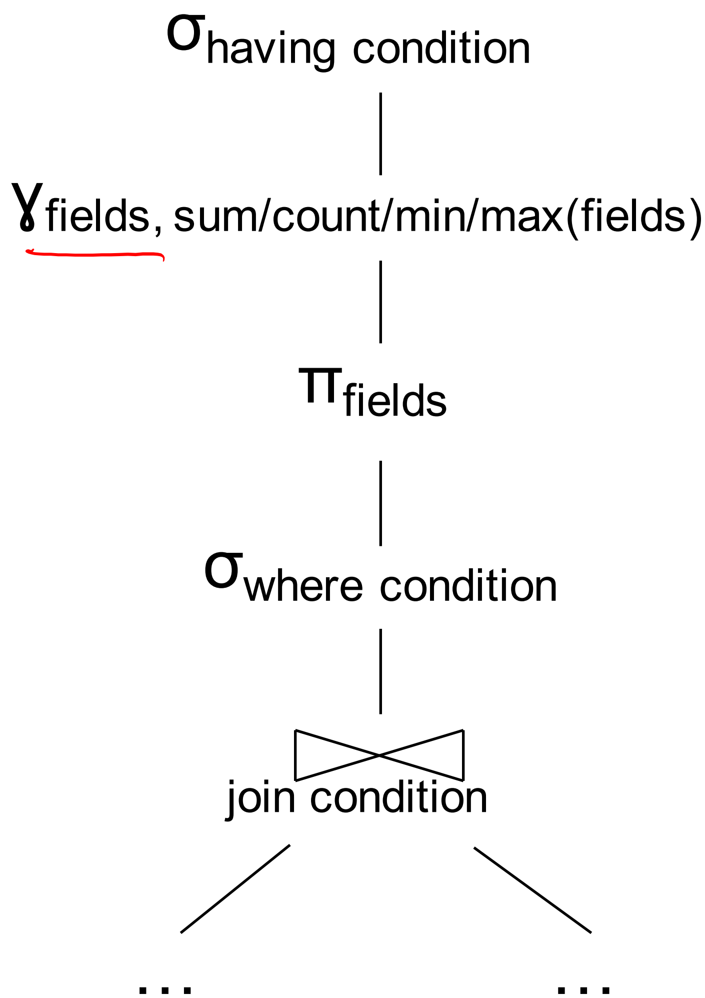
Typical Plan for a Query (1/2)




```
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)
```

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

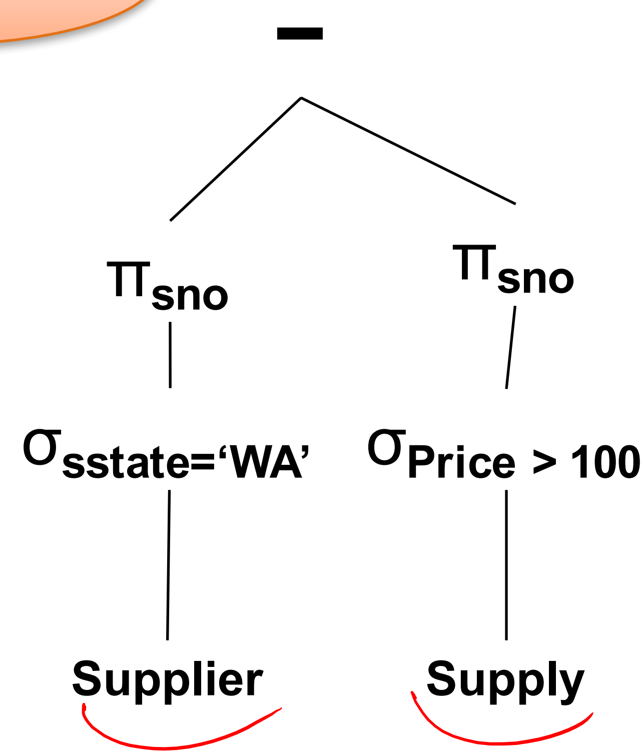
```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
   and Q.sno not in
   (SELECT P.sno
    FROM Supply P
    WHERE P.price > 100)
```

```
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
- Both implements the relational data model
- **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: Datalog is an extension that can compute recursive queries

Summary of RA and SQL

- Translating from SQL to RA gives us a way to *evaluate* the input query
- Transforming one RA plan to another forms the basis of *query optimization*
- Will see more in 2 weeks

Datalog

What is Datalog?

- Another *declarative* query language for relational model
  - Designed in the 80's
  - Minimal syntax
  - Simple, concise, elegant
  - Extends relational queries with *recursion*

- Today:
  - Adopted by some companies for data analytics, e.g., LogicBlox (HW4)
  - Usage beyond databases: e.g., network protocols, static program analysis

```sql
USE AdventureWorks2008R2;
GO
WITH DirectReports (ManagerID, EmployeeID, Title, DeptID, Level)
AS
(
-- Anchor member definition
    SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
        0 AS Level
    FROM dbo.MyEmployees AS e
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
        ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
    WHERE ManagerID IS NULL
    UNION ALL
-- Recursive member definition
    SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID,
        Level + 1
    FROM dbo.MyEmployees AS e
    INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
        ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
    INNER JOIN DirectReports AS d
        ON e.ManagerID = d.EmployeeID
)
-- Statement that executes the CTE
SELECT ManagerID, EmployeeID, Title, DeptID, Level
FROM DirectReports
INNER JOIN HumanResources.Department AS dp
    ON DirectReports.DeptID = dp.DepartmentID
WHERE dp.GroupName = N'Sales and Marketing' OR Level = 0;
GO
```

```
Manager(eid) :- Manages(_, eid)

DirectReports(eid, 0) :-
        Employee(eid),
        not Manager(eid)

DirectReports(eid, level+1) :-
        DirectReports(mid, level),
        Manages(mid, eid)
```

SQL Query vs Datalog

(which would you rather write?)

(any Java fans out there?)

HW4: Preview

```
Welcome to the LogicBlox playground!

addblock 'r(x,y) -> int(x), int(y).'

Successfully added block 'block_1Z331BSE'
exec '+r(1,2). +r(2,1). +r(2,3). +r(1,4). +r(3,4). +r(4,5).'
print r
```

1	2
1	4
2	1
2	3
3	4
4	5

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

Rules = queries

```
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).
```

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'.
```

```
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

Datalog: Terminology

head | body

atom | atom | atom (aka subgoal)

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

f, l = head variables

x,y,z = existential variables

In this class we discuss datalog evaluated under **set semantics**

More Datalog Terminology

```
Q(args) :- R1(args), R2(args), ....
```

Your book uses:
Q(args) :- R1(args) AND R2(args) AND ....

- $R_i(args_i)$ is called an atom, or a relational predicate
- $R_i(args_i)$ evaluates to true when relation R_i contains the tuple described by $args_i$.
  - Example: Actor(344759, 'Douglas', 'Fowley') is true
- In addition to relational predicates, we can also have arithmetic predicates
  - Example: z > '1940'.
- Note: Logicblox uses <- instead of :-

```
Q(args) <-  R1(args), R2(args), ....
```

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

Semantics of a Single Rule

- Meaning of a datalog rule = a logical statement !

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

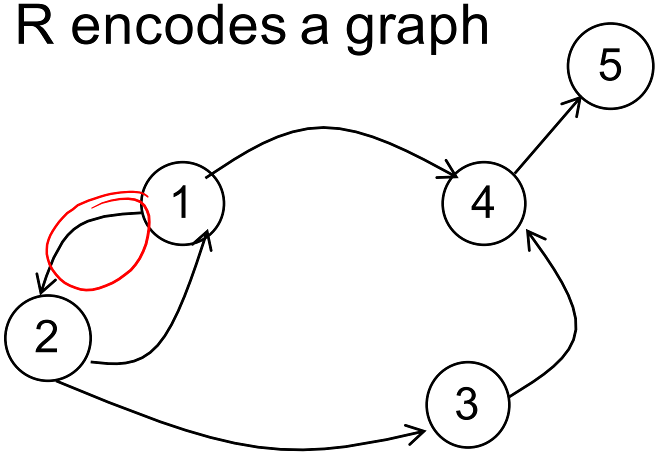
- For all values of x, y, z:
  if (x,y,z) is in the Movies relation, and that z = '1940' then y is in Q1 (i.e., it is part of the answer)
- Logically equivalent:
  $\forall$ y. [($\exists$ x. $\exists$ z. Movie(x,y,z) and z='1940') $\Rightarrow$ Q1(y)]
- That's why ~~head~~ variables are called "existential variables" non-
- We want the *smallest* set Q1 with this property (why?)

Datalog program

- A datalog program consists of several rules
- Importantly, rules may be recursive!
- Usually there is one distinguished predicate that's the output
- We will show an example first, then give the general semantics.

Example

R encodes a graph

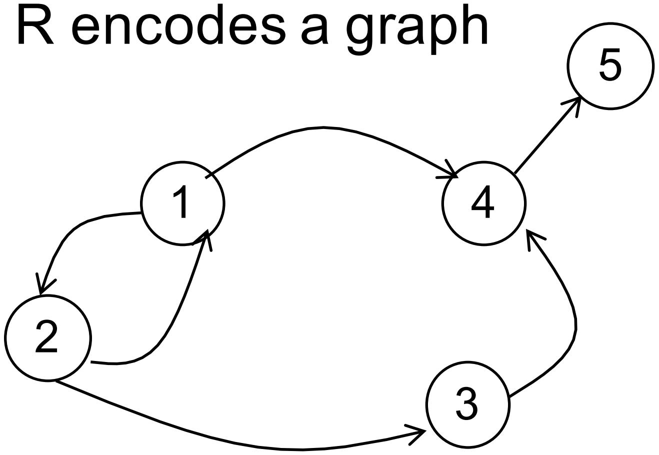


R=

1	2
2	1
2	3
1	4
3	4
4	5

Example

R encodes a graph



T(x,y) :- R(x,y)

T(x,y) :- R(x,z), T(z,y)

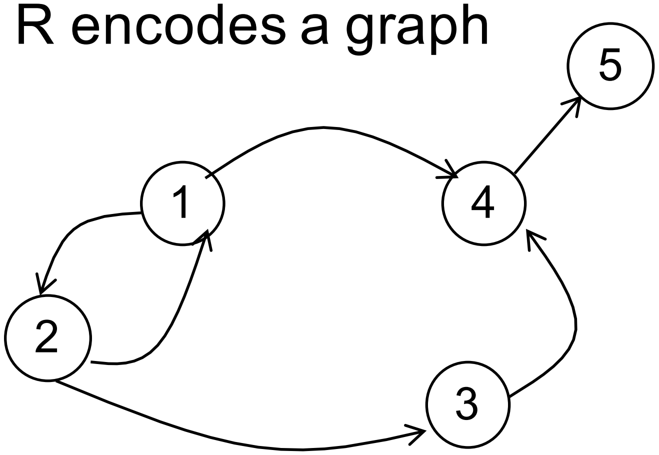
What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Example

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

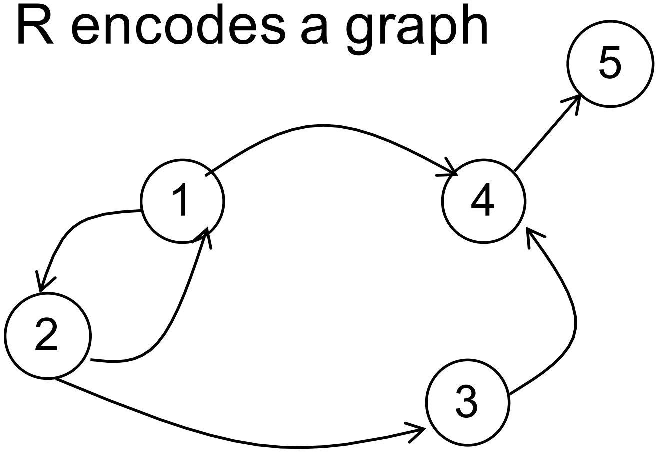
Initially:
T is empty.

T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)

What does it compute?

Example

R encodes a graph



$$T(x,y) \text{ :- } R(x,y)$$
$$T(x,y) \text{ :- } R(x,z),\ T(z,y)$$

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:
T is empty.

First iteration:
T =

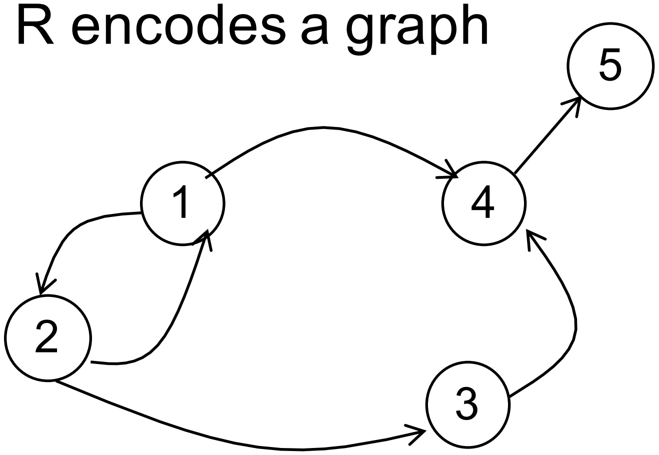
1	2
2	1
2	3
1	4
3	4
4	5

First rule generates this

Second rule generates nothing (because T is empty)

Example

R encodes a graph



T(x,y) :- R(x,y)

T(x,y) :- R(x,z), T(z,y)

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:

T is empty.

First iteration:

T =

1	2
2	1
2	3
1	4
3	4
4	5

Second iteration:

T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

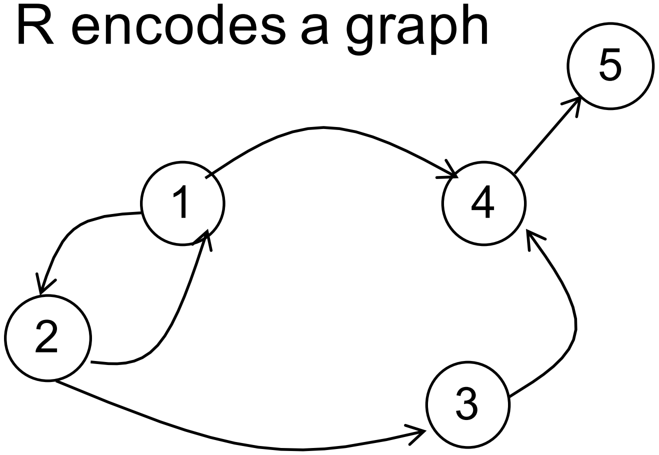
First rule generates this

Second rule generates this

New facts

Example

R encodes a graph



$$T(x,y) \text{ :- } R(x,y)$$
$$T(x,y) \text{ :- } R(x,z), T(z,y)$$

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:
T is empty.

First iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5

Second iteration:
T =

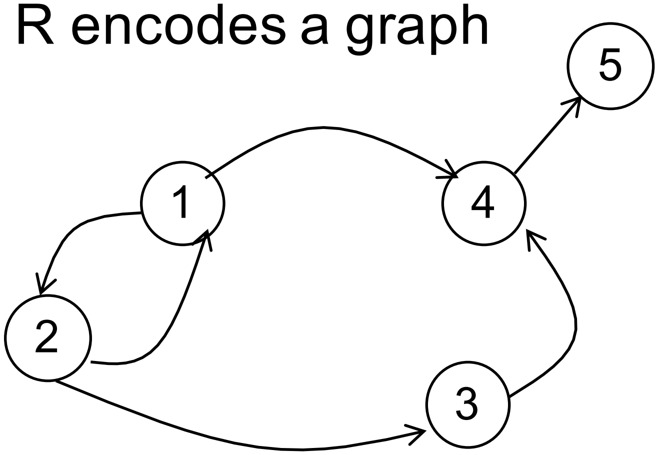
1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

Third iteration:
T =

1	2	Both rules
2	1	
2	3	First rule
1	4	
3	4	
4	5	
1	1	Second rule
2	2	
1	3	
2	4	
1	5	
3	5	
2	5	New fact

Example

R encodes a graph




```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:

T is empty.

First iteration:

T =

1	2
2	1
2	3
1	4
3	4
4	5

Second iteration:

T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

Third iteration:

T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5
2	5

Fourth iteration

T =

(same)

No new facts. DONE

This is called the **fixpoint semantics** of a datalog program