Principles of Database Systems CSE 544

Lecture #5

Views, Relational Query Languages

Announcements

- Homework 1 due next Monday
- Next reading assignment due next Wednesday
- Lecture on Thursday, May 2nd:
  - Moved to 9am-10:30am, CSE 403

Applications of Views

What applications does the paper describe?

Applications of Views

What applications does the paper describe?

- Query optimization
  - E.g. Indexes
- Physical and logical data independence
  - E.g. de-normalization, data partitioning
- Semantic caching
- Data integration

Denormalization

- Scenario: we have a relational schema that is in BCNF (recall: this means only the key implies any other attribute(s))

Purchase(<u>pid</u>, customer, product, store)
Product(<u>pname</u>, price)

- But we often need to join these two relations, so we compute their join

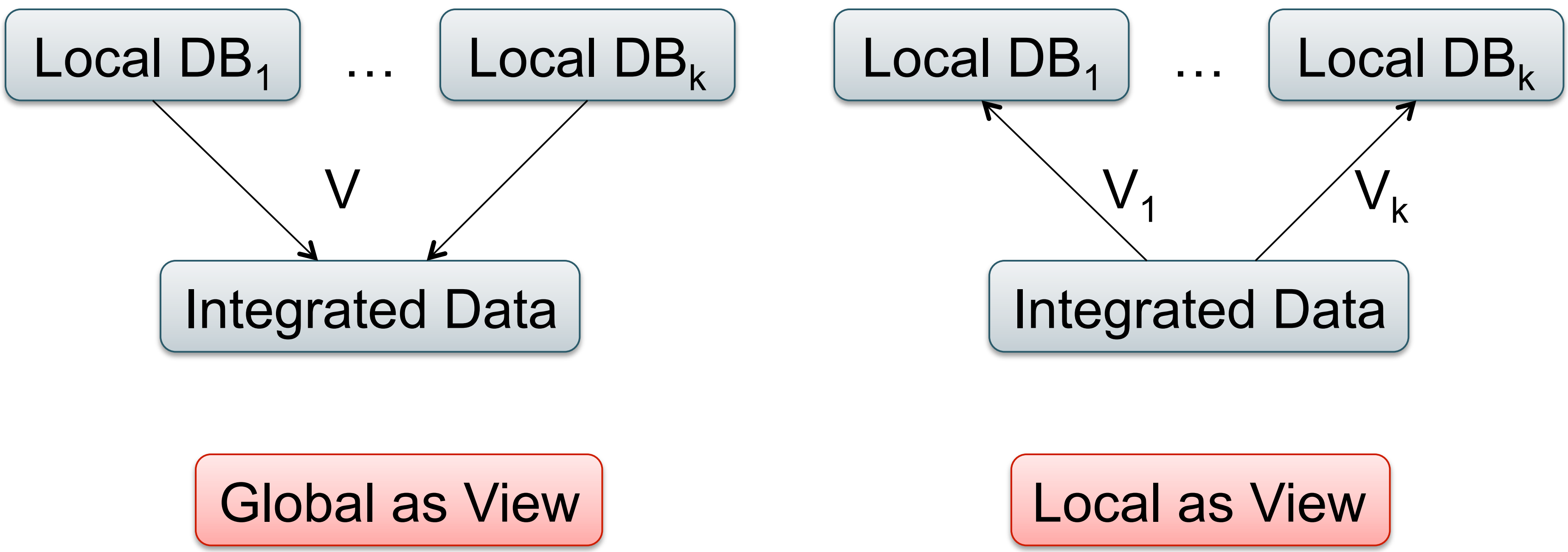
Denormalization

```
CREATE Table CustomerPurchase  AS
      SELECT  x.pid, x.customer, x.store, y.pname, y.price
      FROM    Purchase x, Product y
      WHERE   x.product = y.pname
```

- This table is not in BCNF (why not?)
- But that's OK, the application still sees the original two relations. How?

Purchase(pid, customer, product, store) – a view…
Product(pname, price) – a view…

Data Integration Terminology



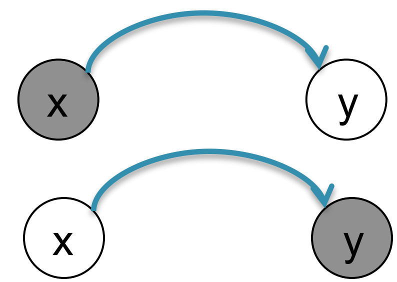
Which one needs query expansion,
which one needs query answering using views ?

[Duschka&Genesereth'97]

Query Rewriting Using Views

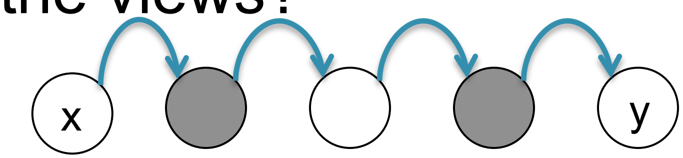
Suppose you only have these two views:

```
v1(x,y) :- black(x), edge(x,y)
v2(x,y) :- edge(x,y), black(y)
```

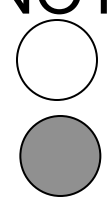


Can you rewrite this query in terms of the views?

```
q(x,y) :- edge(x,z1), black(z1),
          edge(z1,z2),edge(z2,z3)
          black(z3), edge(z3,y)
```



NOTE:



means "any color"

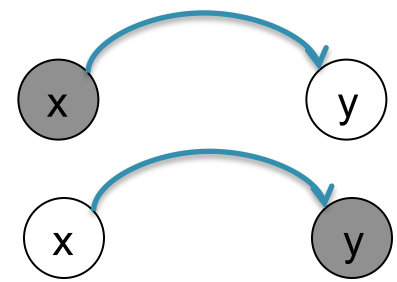
means "black"

[Duschka&Genesereth'97]

Query Rewriting Using Views

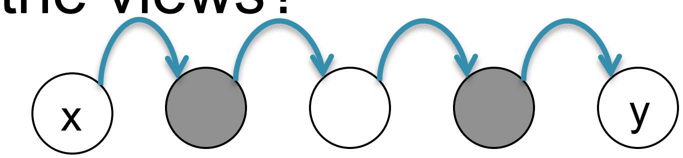
Suppose you only have these two views:

v1(x,y) :- black(x), edge(x,y)
v2(x,y) :- edge(x,y), black(y)



Can you rewrite this query in terms of the views?

q(x,y) :- edge(x,z1), black(z1),
edge(z1,z2),edge(z2,z3)
black(z3), edge(z3,y)



Answer:

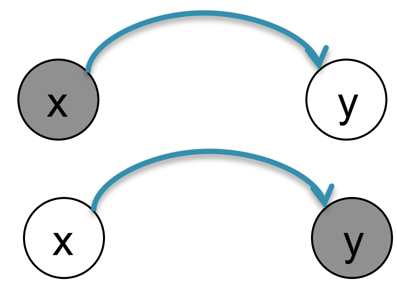
q(x,y) :- v2(x,z1),v1(z1,z2),v2(z2,z3),v1(z3,y)

[Duschka&Genesereth'97]

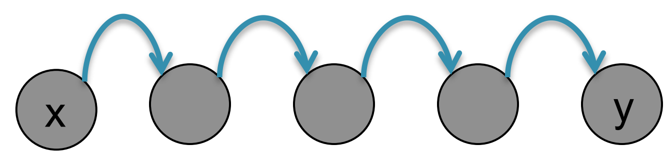
Query Rewriting Using Views

Suppose you only have these two views:

v1(x,y) :- black(x), edge(x,y)
v2(x,y) :- edge(x,y), black(y)



What about this query?

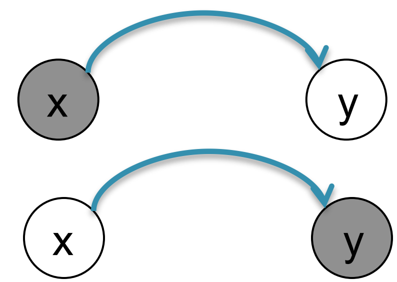


[Duschka&Genesereth'97]

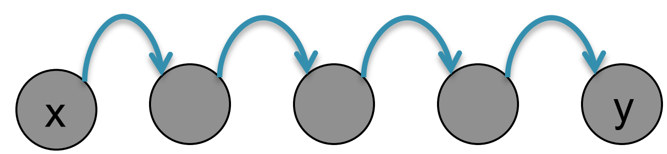
Query Rewriting Using Views

Suppose you only have these two views:

v1(x,y) :- black(x), edge(x,y)
v2(x,y) :- edge(x,y), black(y)



What about this query?



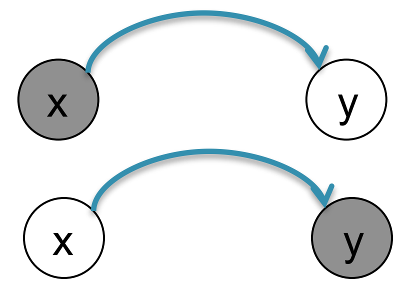
q(x,y) :- black(x),edge(x,z1), black(z1),
edge(z1,z2),black(z2),edge(z2,z3)
black(z3), edge(z3,y),black(y)

[Duschka&Genesereth'97]

Query Rewriting Using Views

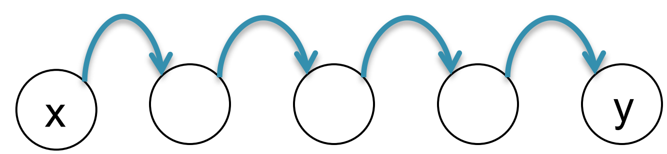
Suppose you only have these two views:

```
v1(x,y) :- black(x), edge(x,y)
v2(x,y) :- edge(x,y), black(y)
```



Can we rewrite this query?

```
q(x,y) :- edge(x,z1),edge(z1,z2),
          edge(z2,z3), edge(z3,y)
```

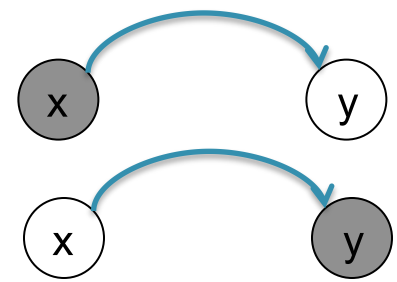


[Duschka&Genesereth'97]

Query Rewriting Using Views

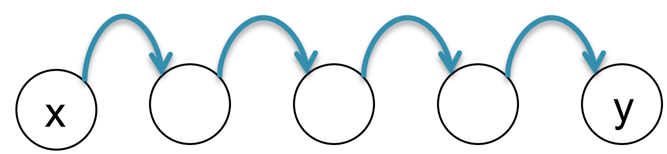
Suppose you only have these two views:

v1(x,y) :- black(x), edge(x,y)
v2(x,y) :- edge(x,y), black(y)

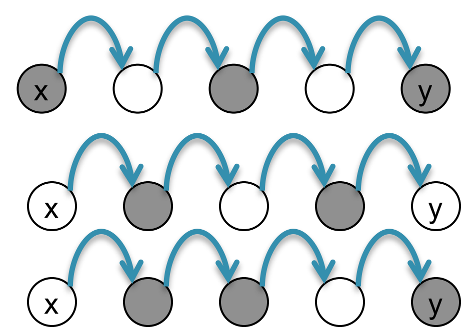


Can we rewrite this query?

q(x,y) :- edge(x,z1),edge(z1,z2),
edge(z2,z3), edge(z3,y)



No! but you can retrieve all *certain* answers:



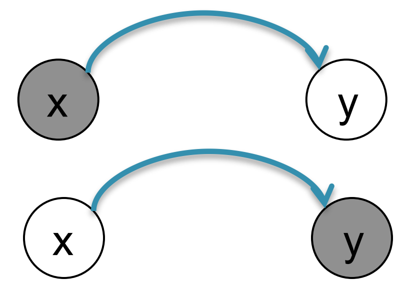
. . . . .

[Duschka&Genesereth'97]

Query Rewriting Using Views

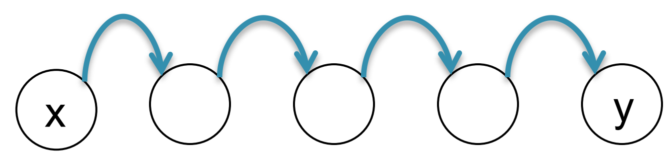
Suppose you only have these two views:

v1(x,y) :- black(x), edge(x,y)
v2(x,y) :- edge(x,y), black(y)



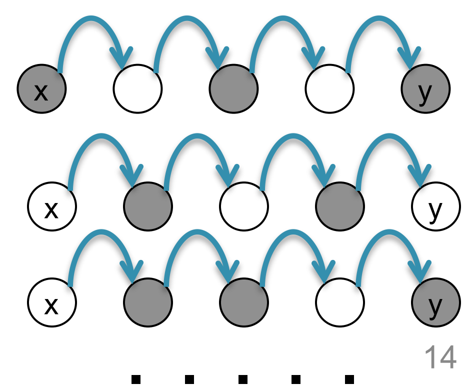
Can we rewrite this query?

q(x,y) :- edge(x,z1),edge(z1,z2),
edge(z2,z3), edge(z3,y)



Maximally contained rewriting is:

q(x,y) :- v1(x,z1),v2(z1,z2),v1(z2,z3),v2(z3,y)
q(x,y) :- v2(x,z1),v1(z1,z2),v2(z2,z3),v1(z3,y)
q(x,y) :- v2(x,z1),v1(z1,z2),v1(z2,z3),v2(z3,y)
. . . .



Purchase(buyer, seller, product, store)
Person(pname, city)

Query Rewriting Using Views

Have this materialized view:

```
CREATE VIEW SeattleView AS
       SELECT  y.buyer, y.seller, y.product, y.store
       FROM    Person x, Purchase y
       WHERE   x.city = 'Seattle'
          AND  x.pname = y.buyer
```

Goal: rewrite this query in terms of the view

```
SELECT  y.buyer, y.seller
FROM    Person x, Purchase y
WHERE   x.city = 'Seattle'
   AND  x.pname = y.buyer
   AND  y.product='gizmo'
```

Purchase(buyer, seller, product, store)
Person(pname, city)

Query Rewriting Using Views

```
SELECT  buyer, seller
FROM    SeattleView
WHERE   product= ‘gizmo’
```

```
SELECT  y.buyer, y.seller
FROM    Person x, Purchase y
WHERE   x.city = ‘Seattle’
   AND  x.pname = y.buyer
   AND  y.product=‘gizmo’
```

Purchase(buyer, seller, product, store)
Person(pname, city)

Query Rewriting Using Views

```
CREATE VIEW DifferentView AS
        SELECT  y.buyer, y.seller, y.product, y.store
        FROM    Person x, Purchase y, Product z
        WHERE   x.city = 'Seattle'   AND
                x.pname = y.buyer AND
                y.product = z.pname AND
                z.price < 100
```

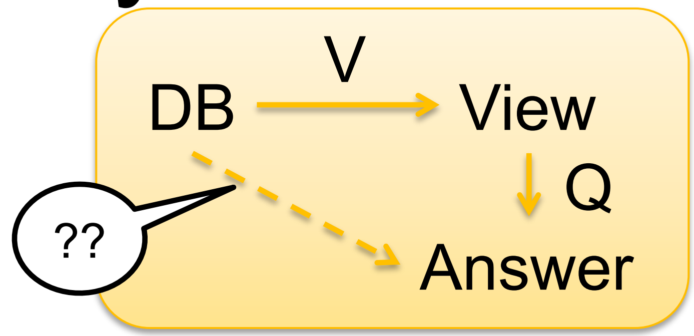
```
SELECT  y.buyer, y.seller
FROM    Person x, Purchase y
WHERE   x.city = 'Seattle'   AND
        x.pname = y.buyer AND
        y.product='gizmo'
```

"Maximally contained rewriting"

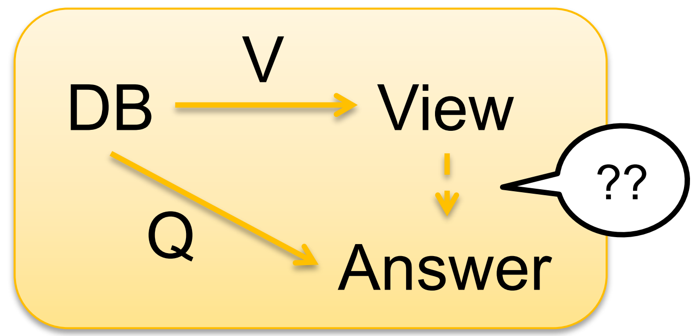
```
SELECT  buyer, seller
FROM    DifferentView
WHERE   product= 'gizmo'
```

Summary

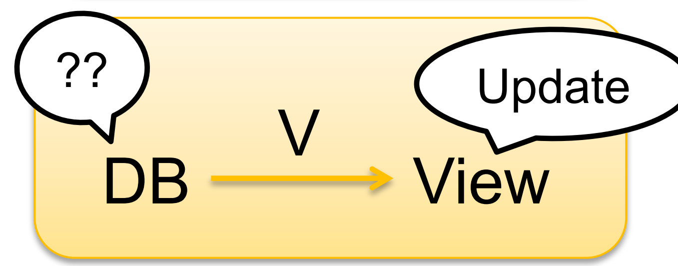
- View inlining, or query modification



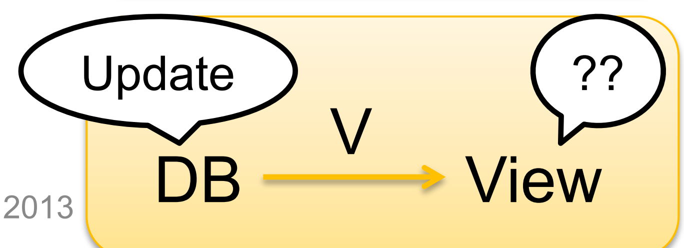
- Query answering/rewriting using views



- Updating views



- Incremental view update



Relational Query Languages

1. Relational Algebra

2. Recursion-free datalog with negation
   - This is the core of SQL, cleaned up

3. Relational Calculus

These three formalisms express the same class of queries

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

Running Example

Find all actors who acted both in 1910 and in 1940:

```
Q: SELECT DISTINCT a.fname, a.lname
   FROM   Actor a, Casts c1, Movie m1, Casts c2, Movie m2
   WHERE  a.id = c1.pid   AND c1.mid = m1.id
      AND a.id = c2.pid   AND c2.mid = m2.id
      AND m1.year = 1910 AND m2.year = 1940;
```

Two Perspectives

- Named Perspective:
  Actor(id, fname, lname)
  Casts(pid,mid)
  Movie(id,name,year)
- Unnamed Perspective:
  Actor = arity 3
  Casts = arity 2
  Movie = arity 3

1. Relational Algebra

Used internally by RDBMs to execute queries

The Basic Five operators:

- Union: $\cup$
- Difference: -
- Selection: σ
- Projection: Π
- Join: $\bowtie$

Renaming: ρ (for named perspective)

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

1. Relational Algebra (Details)

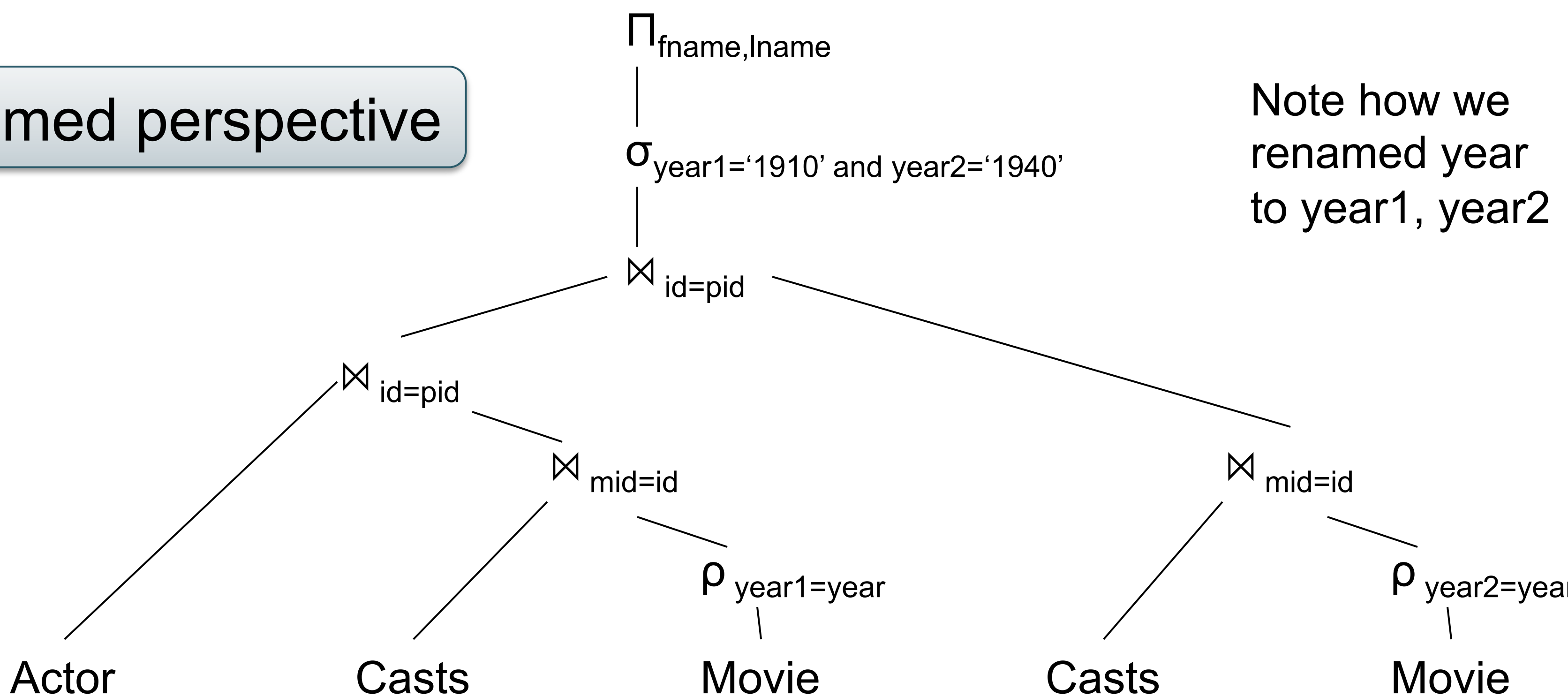
- Selection: returns tuples that satisfy condition
  - Named perspective: $\sigma_{year = \text{'1910'}}(\text{Movie})$
  - Unnamed perspective: $\sigma_{3 = \text{'1910'}}(\text{Movie})$
- Projection: returns only some attributes
  - Named perspective: $\Pi_{fname,lname}(\text{Actor})$
  - Unnamed perspective: $\Pi_{2,3}(\text{Actor})$
- Join: joins two tables on a condition
  - Named perspective: $\text{Casts} \bowtie_{mid=id} \text{Movie}$
  - Unnamed perspective: $\text{Casts} \bowtie_{2=1} \text{Movie}$

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

1. Relational Algebra

```
Q: SELECT DISTINCT a.fname, a.lname
   FROM   Actor a, Casts c1, Movie m1, Casts c2, Movie m2
   WHERE  a.id = c1.pid       AND c1.mid = m1.id
      AND a.id = c2.pid       AND c2.mid = m2.id
      AND m1.year = 1910   AND m2.year = 1940;
```

Named perspective



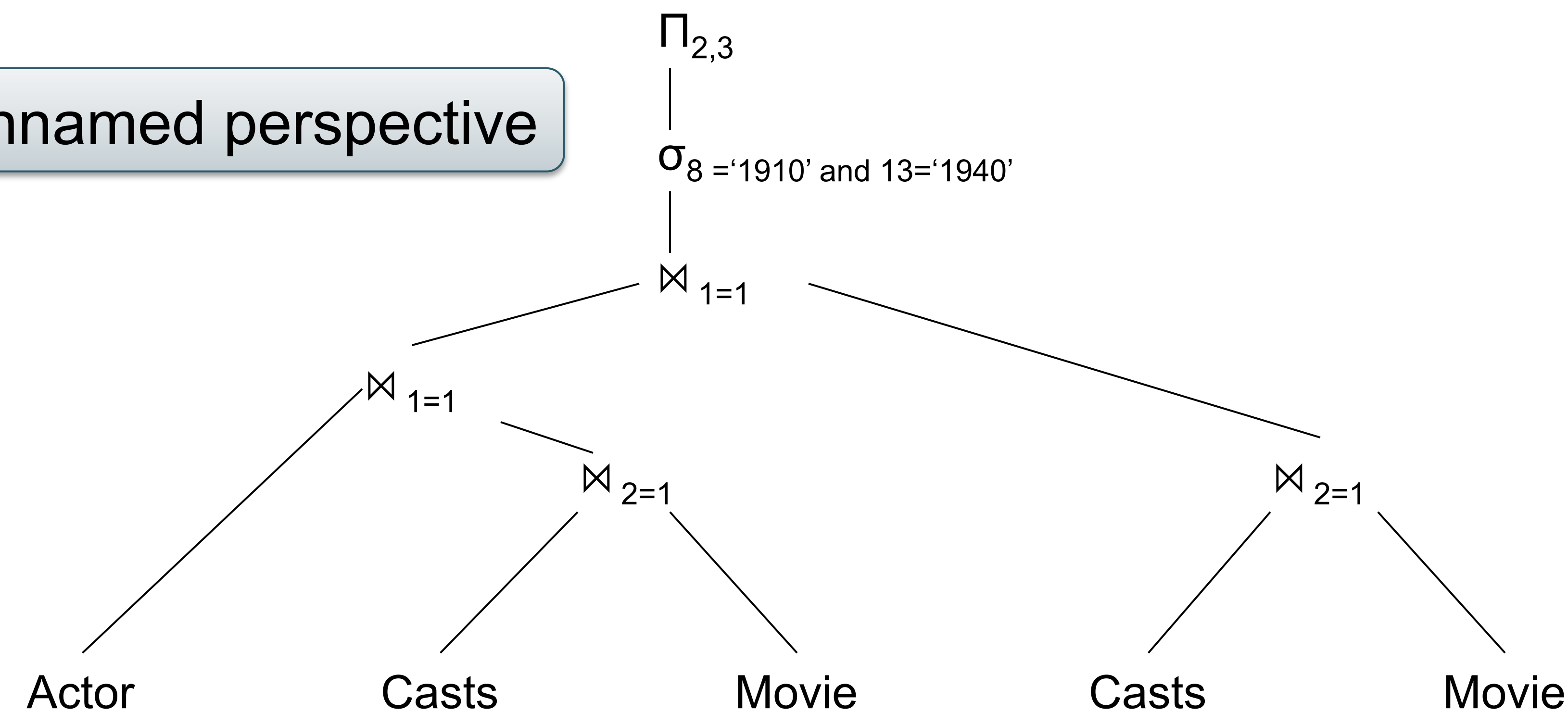
Note how we renamed year to year1, year2

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

1. Relational Algebra

```
Q: SELECT DISTINCT a.fname, a.lname
   FROM   Actor a, Casts c1, Movie m1, Casts c2, Movie m2
   WHERE  a.id = c1.pid       AND c1.mid = m1.id
     AND  a.id = c2.pid       AND c2.mid = m2.id
     AND  m1.year = 1910   AND m2.year = 1940;
```

Unnamed perspective



2. Datalog

- Very friendly notation for queries
- Designed for *recursive* queries in the 80s
- Today it's used everywhere: commercial implementations (LogicBlox), networking (Overlog), programming languages, …

- In class
  – *recursion-free* datalog with negation (next)
  – *recursive datalog*, (in the "Theory" part)

2. Datalog

How to try out datalog quickly:

- Download DLV from http://www.dbai.tuwien.ac.at/proj/dlv/
- Run DLV on this file:

```
parent(william, john).
parent(john, james).
parent(james, bill).
parent(sue, bill).
parent(james, carol).
parent(sue, carol).

male(john).
male(james).
female(sue).
male(bill).
female(carol).

grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
father(X, Y) :- parent(X, Y), male(X).
mother(X, Y) :- parent(X, Y), female(X).
brother(X, Y) :- parent(P, X), parent(P, Y), male(X), X != Y.
sister(X, Y)  :- parent(P, X), parent(P, Y), female(X), X != Y.
```

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

2. 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)

2. 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)

2. 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)

2. 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

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

2. Datalog: Terminology

$$\overbrace{\text{Q2(f, l)}}^{\text{head}}\ \text{:-}\ \overbrace{\overbrace{\text{Actor(z,f,l)}}^{\text{atom}},\ \overbrace{\text{Casts(z,x)}}^{\text{atom}},\ \overbrace{\text{Movie(x,y,'1940')}}^{\text{atom}}}^{\text{body}}.$$

f, l = head variables
x,y,z = existential variables

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

2. Datalog program

Find all actors with Bacon number ≤ 2

```
B0(x) :- Actor(x,'Kevin', 'Bacon')
B1(x) :- Actor(x,f,l), Casts(x,z), Casts(y,z), B0(y)
B2(x) :- Actor(x,f,l), Casts(x,z), Casts(y,z), B1(y)
Q4(x) :- B0(x)
Q4(x) :- B1(x)
Q4(x) :- B2(x)
```

Note: Q4 is the *union* of B1 and B2

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

2. Datalog with negation

Find all actors with Bacon number ≥ 2

```
B0(x) :- Actor(x,'Kevin', 'Bacon')
B1(x) :- Actor(x,f,l), Casts(x,z), Casts(y,z), B0(y)
Q6(x) :- Actor(x,f,l), not B1(x), not B0(x)
```

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

2. Safe Datalog Rules

Here are *unsafe* datalog rules. What's "unsafe" about them ?

U1(x,y) :- Movie(x,z,1994), y>1910

U2(x) :- Movie(x,z,1994), not Casts(u,x)

A datalog rule is *safe* if every variable appears in some positive relational atom

2. Datalog v.s. SQL

- Non-recursive datalog with negation is a cleaned-up, core of SQL
- You should be able to translate easily between non-recursive datalog with negation and SQL

Relational Calculus

- Aka *predicate calculus* or *first order logic*

- TRC = Tuple RC
  - See book
- DRC = Domain RC = unnamed perspective
  - We study only this one

Relational Calculus

Relational predicate P is a formula given by this grammar:

$$P ::= \text{atom} \mid P \wedge P \mid P \vee P \mid P \Rightarrow P \mid \text{not}(P) \mid \forall x.P \mid \exists x.P$$

Query Q:

$$Q(x1, \ldots, xk) = P$$

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

Relational Calculus

Relational predicate P is a formula given by this grammar:

$$P ::= \text{atom} \mid P \wedge P \mid P \vee P \mid P \Rightarrow P \mid \text{not}(P) \mid \forall x.P \mid \exists x.P$$

Query Q:

$$Q(x1, \ldots, xk) = P$$

Example: find the first/last names of actors who acted in 1940

$$Q(f,l) = \exists x.\ \exists y.\ \exists z.\ (\text{Actor}(z,f,l) \wedge \text{Casts}(z,x) \wedge \text{Movie}(x,y,1940))$$

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

Relational Calculus

Relational predicate P is a formula given by this grammar:

$$P ::= \text{atom} \mid P \land P \mid P \lor P \mid P \Rightarrow P \mid \text{not}(P) \mid \forall x.P \mid \exists x.P$$

Query Q:

$$Q(x1, \ldots, xk) = P$$

Example: find the first/last names of actors who acted in 1940

$$Q(f,l) = \exists x.\ \exists y.\ \exists z.\ (\text{Actor}(z,f,l) \land \text{Casts}(z,x) \land \text{Movie}(x,y,1940))$$

What does this query return ?

$$Q(f,l) = \exists z.\ (\text{Actor}(z,f,l) \land \forall x.(\text{Casts}(z,x) \Rightarrow \exists y.\text{Movie}(x,y,1940)))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Important Observation

Find all bars that serve all beers that Fred likes

$$A(x) = \forall y.\ \text{Likes}(\text{"Fred"}, y) \Rightarrow \text{Serves}(x,y)$$

- Note: P ==> Q (read P implies Q) is the same as (not P) OR Q
  In this query: If Fred likes a beer the bar must serve it (P ==> Q)
  In other words: Either Fred does not like the beer (not P) OR the bar serves that beer (Q).

$$A(x) = \forall y.\ \text{not}(\text{Likes}(\text{"Fred"}, y))\ \text{OR}\ \text{Serves}(x,y)$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent some bar that serves some beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent only bars that serves some beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \land \text{Likes}(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent some bar that serves some beer they like.

$$Q(x) = \exists y.\ \exists z.\ \mathrm{Frequents}(x, y) \wedge \mathrm{Serves}(y,z) \wedge \mathrm{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent only bars that serves some beer they like.

$$Q(x) = \forall y.\ \mathrm{Frequents}(x, y) \Rightarrow (\exists z.\ \mathrm{Serves}(y,z) \wedge \mathrm{Likes}(x,z))$$

Cautious Carl

Find drinkers that frequent some bar that serves only beers they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$$

Cautious Carl

Find drinkers that frequent <u>some</u> bar that serves <u>only</u> beers they like.

$$Q(x) = \exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent some bar that serves some beer they like.

$$Q(x) = \exists y.\ \exists z.\ \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z)$$

Prudent Peter

Find drinkers that frequent only bars that serves some beer they like.

$$Q(x) = \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \land \text{Likes}(x,z))$$

Cautious Carl

Find drinkers that frequent some bar that serves only beers they like.

$$Q(x) = \exists y.\ \text{Frequents}(x, y) \land \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Paranoid Paul

Find drinkers that frequent only bars that serves only beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

More Examples

Average Joe

Find drinkers that frequent some bar that serves some beer they like.

$$Q(x) = \exists y.\ \exists z.\ Frequents(x, y) \land Serves(y,z) \land Likes(x,z)$$

Prudent Peter

Find drinkers that frequent only bars that serves some beer they like.

$$Q(x) = \forall y.\ Frequents(x, y) \Rightarrow (\exists z.\ Serves(y,z) \land Likes(x,z))$$

Cautious Carl

Find drinkers that frequent some bar that serves only beers they like.

$$Q(x) = \exists y.\ Frequents(x, y) \land \forall z.(Serves(y,z) \Rightarrow Likes(x,z))$$

Paranoid Paul

Find drinkers that frequent only bars that serves only beer they like.

$$Q(x) = \forall y.\ Frequents(x, y) \Rightarrow \forall z.(Serves(y,z) \Rightarrow Likes(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Domain Independent RC

- As in datalog, one can write "unsafe" RC queries; they are also called *domain dependent*

A(x) = not Likes("Fred", x)
A(x,y) = Likes("Fred", x) OR Serves("Bar", y)
A(x) = ∀y. Serves(x,y)

- Lesson: make sure your RC queries are domain independent

Relational Calculus

How to write a complex SQL query:

- Write it in RC
- Translate RC to datalog (see next)
- Translate datalog to SQL

Take shortcuts when you know what you're doing

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$$Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$$Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$$

$\forall x\ P(x)$ same as $\neg\exists x\ \neg P(x)$

Step 1: Replace $\forall$ with $\exists$ using de Morgan's Laws

$\neg(\neg P \vee Q)$ same as $P \wedge \neg Q$

$$Q(x) = \exists y.\ Likes(x, y) \wedge \neg\exists z.(Serves(z,y) \wedge \neg Frequents(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^\neg$ to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$$Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$$

Step 1: Replace ∀ with ∃ using de Morgan's Laws

$\forall x\ P(x)$ same as $\neg\exists x\ \neg P(x)$

$$Q(x) = \exists y.\ Likes(x, y) \wedge \neg\exists z.(Serves(z,y) \wedge \neg Frequents(x,z))$$

$\neg(\neg P \vee Q)$ same as $P \wedge \neg Q$

Step 2: Make all *subqueries* domain independent

$$Q(x) = \exists y.\ Likes(x, y) \wedge \neg\exists z.(Likes(x,y) \wedge Serves(z,y) \wedge \neg Frequents(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

$$Q(x) = \exists y.\ Likes(x, y) \wedge \neg \underbrace{\exists z.(Likes(x,y) \wedge Serves(z,y) \wedge \neg Frequents(x,z))}_{H(x,y)}$$

Step 3: Create a datalog rule for each subexpression;
(shortcut: only for "important" subexpressions)

```
H(x,y)   :- Likes(x,y),Serves(z,y), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

```
H(x,y)   :- Likes(x,y),Serves(z,y), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

Step 4: Write it in SQL

```sql
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Likes L2, Serves S
   WHERE L2.drinker=L.drinker and L2.beer=L.beer
        and L2.beer=S.beer
        and not exists (SELECT * FROM Frequents F
                        WHERE F.drinker=L2.drinker
                          and F.bar=S.bar))
```

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

H(x,y) :- ~~Likes(x,y),~~Serves(z,y), not Frequents(x,z)
Q(x) :- Likes(x,y), not H(x,y)

Unsafe rule

Improve the SQL query by using an unsafe datalog rule

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Serves S
   WHERE L.beer=S.beer
         and not exists (SELECT * FROM Frequents F
                         WHERE F.drinker=L.drinker
                           and F.bar=S.bar))
```

Summary of Translation

- RC → recursion-free datalog w/ negation
  - Subtle: as we saw; more details in the paper
- Recursion-free datalog w/ negation → RA
- RA → RC

Theorem: RA, non-recursive datalog w/ negation, and RC, express exactly the same sets of queries: RELATIONAL QUERIES