# CSE344: Lecture 10
# Relational Query Languages

## Relational Algebra, Datalog, Relational Calculus

# Announcements

- Makeup lecture:
  - Tue, Jan 31st 2012, 3:30-4:20, MOR 220
  - Optional: we will practice datalog, RA, RC
- Homework 3 due Wednesday
- Midterm next Monday

- For today's lecture: read accompanying paper (see the Calendar page)

# Relational Query Languages

1. Relational Algebra

2. Recursion-free datalog with negation

3. Relational Calculus

# 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 the database engine to execute queries

# 1. Relational Algebra

The Basic Five operators:

- Union: $\cup$
- Difference: -
- Selection: $\sigma$
- Projection: $\Pi$
- Join: $\bowtie$

Renaming: $\rho$ (for named perspective)

# 1. Relational Algebra (Details)

- Selection: returns tuples that satisfy condition
  - Named perspective: $\sigma_{\text{year = '1910'}}$(Movie)
  - Unnamed perspective: $\sigma_{\text{3 = '1910'}}$ (Movie)

- Projection: returns only some attributes
  - Named perspective: $\Pi_{\text{fname,lname}}$(Actor)
  - Unnamed perspective: $\Pi_{2,3}$(Actor)

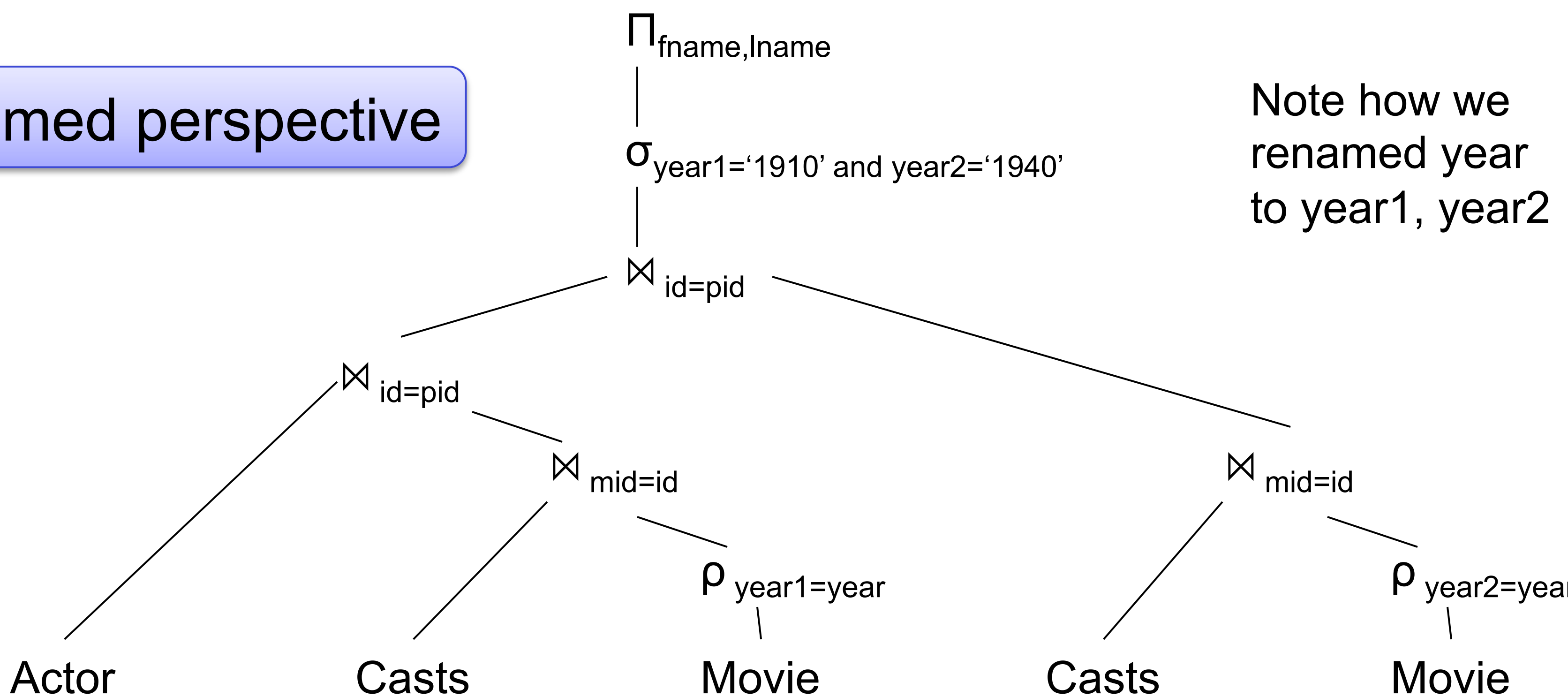
- Join: joins two tables on a condition
  - Named perspective: Casts $\bowtie_{\text{mid=id}}$ Movie
  - Unnamed perspectivie: Casts $\bowtie_{2=1}$ Movie

# 1. Relational Algebra Example

```
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;
```

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

**Named perspective**



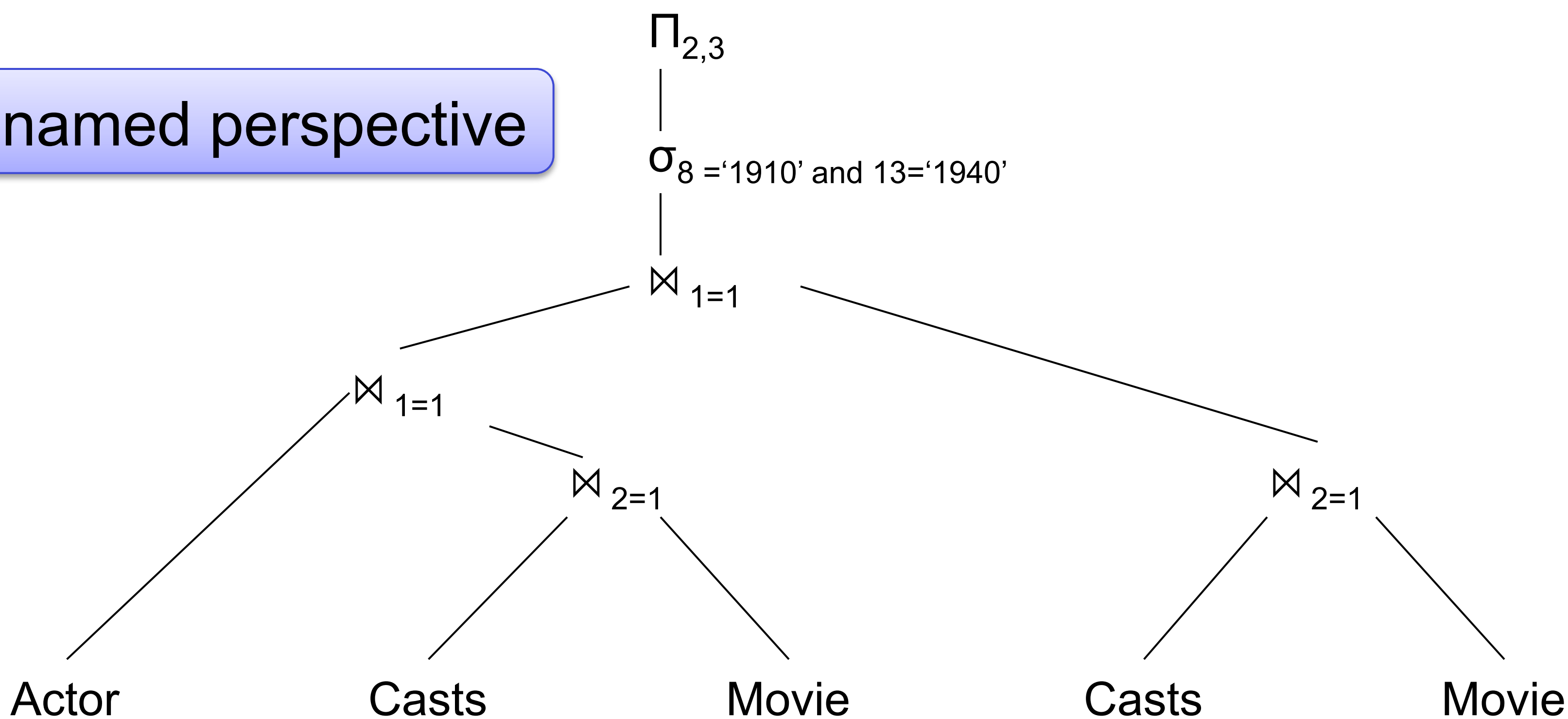

Note how we renamed year to year1, year2

# 1. Relational Algebra Example

```
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;
```

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

**Unnamed perspective**



# 2. Datalog

- Very friendly notation for queries
- Initially designed for *recursive* queries
- Some companies offer datalog implementation for data anlytics, e.g. LogicBlox
- We discuss only *recursion-free* or *non-recursive* datalog, and add negation

# 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.
```

# 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

# 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

# 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

# 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

# 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

# 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) :- B1(x)
Q4(x) :- B2(x)
```

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

# 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)
```

# 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 very close to SQL; with some practice, you should be able to translate between them back and forth without difficulty

# 3. Relational Calculus

- Also known as *predicate calculus*, or *first order logic*
- The most expressive formalism for queries: easy to write complex queries

- TRC = Tuple RC = named perspective
- DRC = Domain RC = unnamed perspective

# 3. Relational Calculus

Predicate P:

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

Query Q:

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

---

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

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

What does this query return ?

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

# 3. Relational Calculus: Example

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

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

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

# 3. Relational Calculus: Example

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

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

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

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

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

# 3. Relational Calculus: Example

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

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

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

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

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

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

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

# 3. Relational Calculus: Example

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

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

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

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

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

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

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

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

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

# 3. Domain Independent Relational Calculus

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

- Lesson: make sure your RC queries are domain independent

# 3. 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

# 3. From RC to Non-recursive Datalog w/ negation

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

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

# 3. From RC to Non-recursive Datalog w/ negation

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

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

# 3. From RC to Non-recursive Datalog w/ negation

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

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

Step 2: Make all subqueries domain independent

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

# 3. From RC to Non-recursive Datalog w/ negation

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

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

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

# 3. From RC to Non-recursive Datalog w/ negation

```
H(x,y)   :- Likes(x,y),Serves(y,z), 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)

# 3. From RC to Non-recursive Datalog w/ negation

```
H(x,y)   :- ~~Likes(x,y),~~Serves(y,z), 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))
```

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

# 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