Introduction to Data Management CSE 344

Lecture 11: Relational Calculus

But First…

A few additional datalog examples

Friend(name1, name2)
Enemy(name1, name2)

Find Joe's friends, and Joe's friends of friends.

```
A(x) :- Friend('Joe', x)
A(x) :- Friend('Joe', z), Friend(z, x)
```

Datalog Example 2

Friend(name1, name2)
Enemy(name1, name2)

Find all of Joe's friends who do not have any friends except for Joe:

```
JoeFriends(x) :- Friend('Joe',x)
NonAns(x) :- Friend(x,y), y != 'Joe'
A(x) :- JoeFriends(x) NOT NonAns(x)
```

Friend(name1, name2)
Enemy(name1, name2)

Datalog Example 3

Find all x such that all their enemies' enemies are their friends

- Assume that if someone doesn't have any enemies nor friends, we also want them in the answer

```
Everyone(x) :- Friend(x,y)
Everyone(x) :- Friend(y,x)
Everyone(x) :- Enemy(x,y)
Everyone(x) :- Enemy(y,x)
NonAns(x) :- Enemy(x,y),Enemy(y,z) NOT Friend(x,z)
A(x) :- Everyone(x) NOT NonAns(x)
```

Friend(name1, name2)
Enemy(name1, name2)

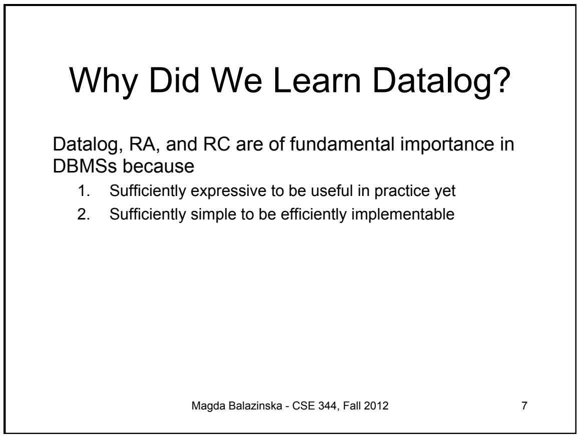
Datalog Example 4

Find all x having some friend all of whose enemies are x's enemies.

```
Everyone(x) :- Friend(x,y)
NonAns(x) :- Friend(x,y) Enemy(y,z) NOT Enemy(x,z)
A(x) :- Everyone(x) NOT NonAns(x)
```

Why Did We Learn Datalog?

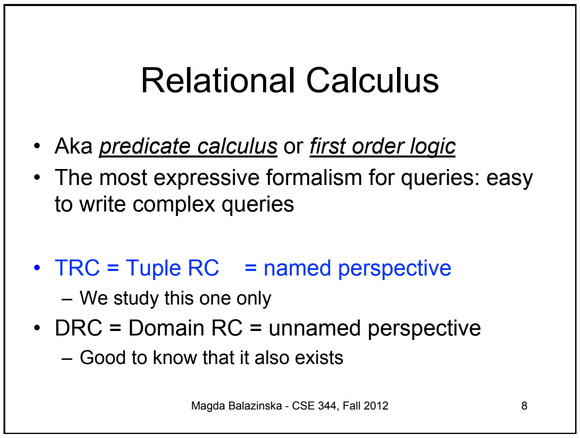
1. Simple, logic language, based on rules
2. Can be extended to recursion BUT beyond 344
3. Equivalences
    1. Datalog can be translated to SQL (practice at home !)
    2. Can also translate back and forth between datalog and relational algebra (see last lecture)
    3. Bottom line: relational algebra, non-recursive datalog with negation, and relational calculus all have the same expressive power!



Why Did We Learn Datalog?

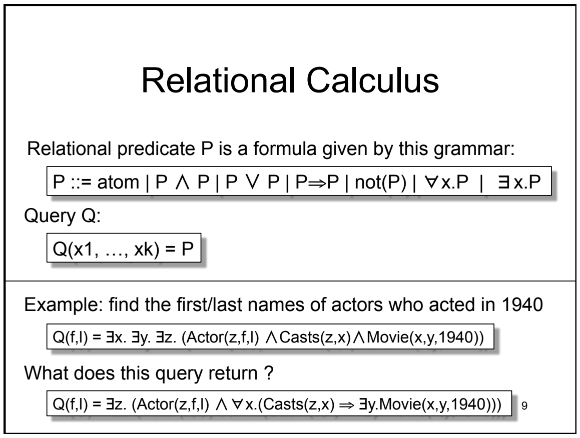
Datalog, RA, and RC are of fundamental importance in DBMSs because

1. Sufficiently expressive to be useful in practice yet
2. Sufficiently simple to be efficiently implementable



Relational Calculus

- Aka *predicate calculus* or *first order logic*
- The most expressive formalism for queries: easy to write complex queries
- TRC = Tuple RC = named perspective
  - We study this one only
- DRC = Domain RC = unnamed perspective
  - Good to know that it also exists



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:

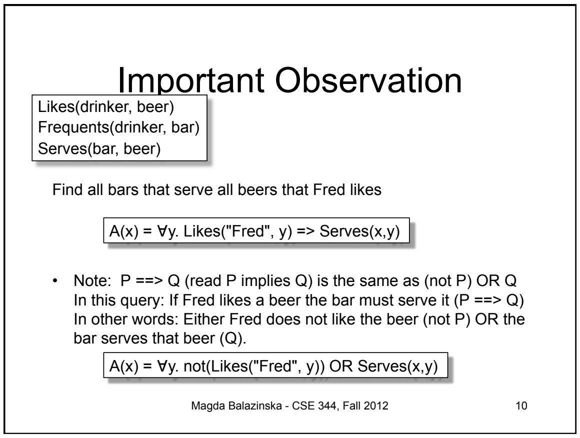
$Q(x1, \ldots, xk) = P$

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

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

What does this query return ?

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



Important Observation

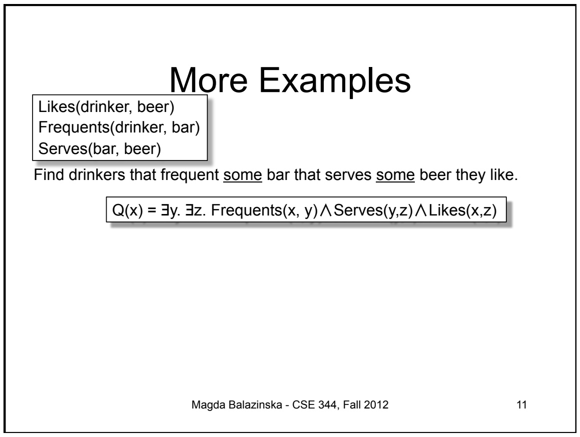
Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

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).

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

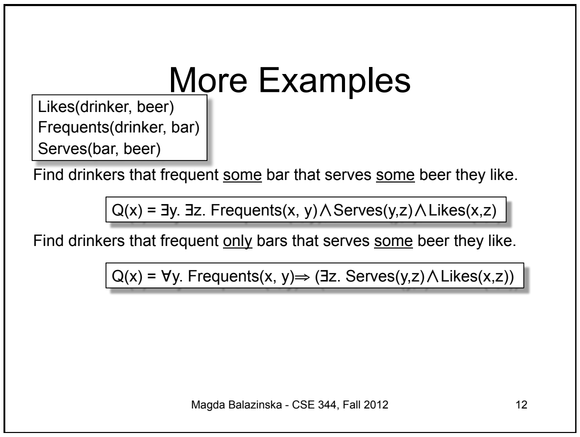


More Examples

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

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

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



More Examples

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

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

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

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

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

More Examples

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.

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

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

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

More Examples

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.

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

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

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

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

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

Domain Independent Relational Calculus

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

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

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

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

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

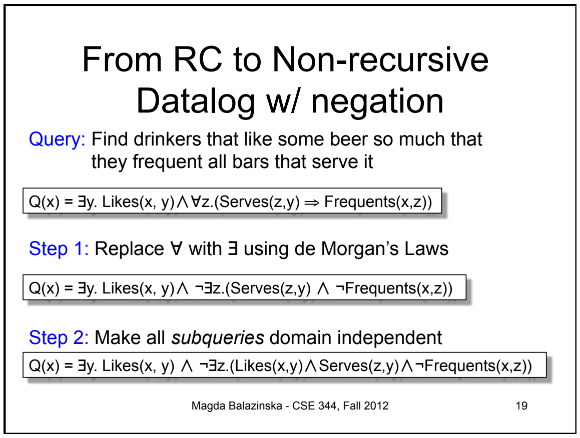
$Q(x) = \exists y.\ \text{Likes}(x, y) \land \forall z.(\text{Serves}(z,y) \Rightarrow \text{Frequents}(x,z))$

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

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

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

$\neg(\neg P \lor Q)$ same as $P \land \neg Q$



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

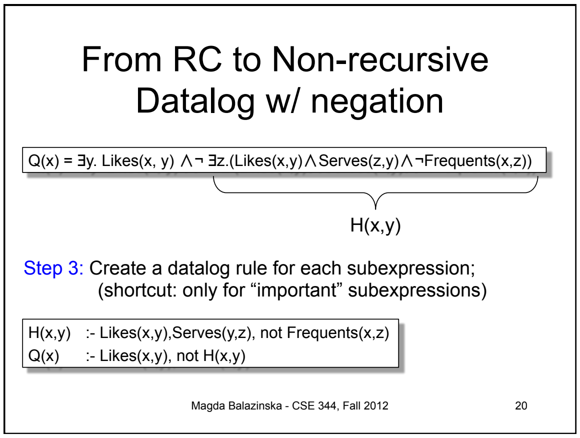
$Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$

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

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

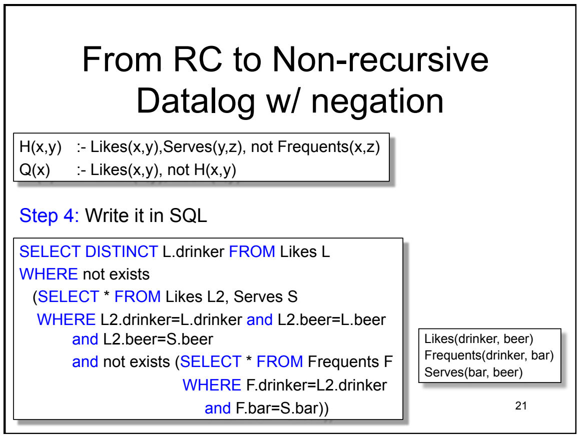


From RC to Non-recursive Datalog w/ negation

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

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

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



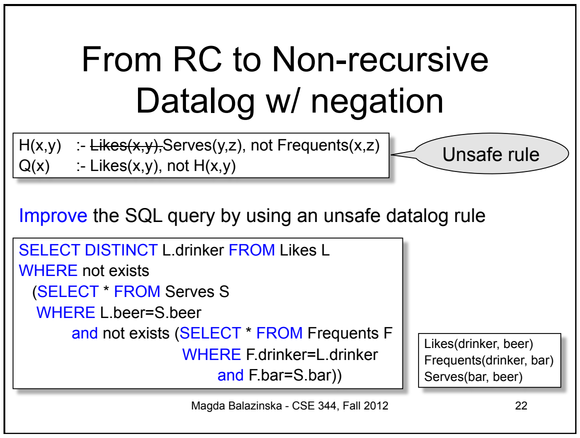
From RC to Non-recursive Datalog w/ negation

```
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

```
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 Non-recursive Datalog w/ negation

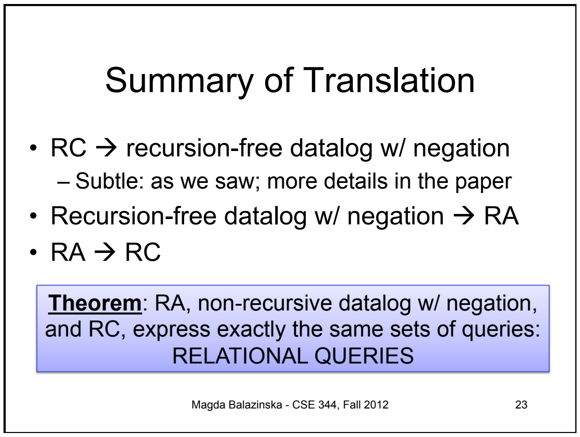
H(x,y) :- ~~Likes(x,y),~~Serves(y,z), not Frequents(x,z)
Q(x) :- Likes(x,y), not H(x,y)

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