Introduction to Data Management CSE 344

Lecture 12: Relational Calculus

Announcements

- WQ4 due tomorrow night (Tuesday)
- Homework 3 due on Thursday night

- For the material in the last three lectures: read *Query Language Primer*, posted on the website

Datalog Summary

- EDB (base relations) and IDB (derived relations)
- Datalog program = set of rules
- Datalog is recursive
- Pure datalog does not have negation;
  if we want negation we say “datalog+negation”
- Multiple atoms in a rule mean join (or intersection)
- Multiple rules with same head mean union
- All variables in the body are existentially quantified
- If we need universal quantifiers, we use DeMorgan’s laws and negation

Friend(name1, name2)

Enemy(name1, name2)

Review: Datalog

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

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

Person(name)
Friend(name1, name2)
Enemy(name1, name2)

Review: Datalog+negation

Find all people 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
- (All people) – (people X having some enemy's enemy who is not X's friend)

```
NonAns(x) :- ….
A(x) :- Person(x), NOT NonAns(x)
```

Person(name)
Friend(name1, name2)
Enemy(name1, name2)

Review: Datalog+negation

Find all people 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

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

Person(name)
Friend(name1, name2)
Enemy(name1, name2)

Review: Datalog+negation

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

- (All people x) [(y is a friend of x) – (y has some enemy z who is NOT x's enemy)]

```
NonAnsFriend(x, y) :- ….
A(x) :- Person(x), Friend(x, y), NOT NonAnsFriend(x, y)
```

Person(name)
Friend(name1, name2)
Enemy(name1, name2)

Review: Datalog+negation

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

```
NonAnsFriend(x,y) :- Friend(x, y), Enemy(y, z), NOT Enemy(x, z)
A(x) :- Person(x), Friend(x, y), NOT NonAnsFriend(x, y)
```

See the conversion of
Relational Calculus to datalog + negation
at the end of these lecture slides
to understand the above rules

Next, Relational Calculus

•We will discus briefly RC in class and mostly cover universal and existential quantiers so that you can master their usage.

• If you are not sure how to write a complex SQL query, you can do RC => Datalog + negation => SQL

• You already know RC from Discrete Mathematics, under the name Predicate Logic, please review it from your favorite textbook in Discrete Mathematics!

Relational Calculus

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

- TRC = Tuple RC
  - See book, e.g. {t.id | t.lname = 'Bacon'}

- DRC = Domain RC = unnamed perspective
  - E.g. Q(x) = ∃ y Actor(x, y, 'Bacon')
  - We study only this one
  - Also see: *Query Language Primer*

Relational Calculus

Relational predicate P is a formula given by this grammar:

P ::= atom | P ∧ P | P ∨ P | P⇒P | not(P) | ∀x.P | ∃x.P

Query Q:

Q(x1, …, xk) = P

Predicate evaluates to true/false on a given database

Relational Calculus

Relational predicate P is a formula given by this grammar:

P ::= atom | P ∧ P | P ∨ P | P⇒P | not(P) | ∀x.P | ∃x.P

Query Q:

Q(x1, …, xk) = P

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

Q(f,l) = ∃x. ∃y. ∃z. (Actor(z,f,l) ∧Casts(z,x)∧Movie(x,y,1940))

Relational Calculus

Relational predicate P is a formula given by this grammar:

P ::= atom | P ∧ P | P ∨ P | P⇒P | not(P) | ∀x.P | ∃x.P

Query Q:

Q(x1, …, xk) = P

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

Q(f,l) = ∃x. ∃y. ∃z. (Actor(z,f,l) ∧Casts(z,x)∧Movie(x,y,1940))

What does this query return ?

Q(f,l) = ∃z. (Actor(z,f,l) ∧∀x.(Casts(z,x) ⇒ ∃y.Movie(x,y,1940)))

Relational Calculus

Relational predicate P is a formula given by this grammar:

P ::= atom | P ∧ P | P ∨ P | P⇒P | not(P) | ∀x.P | ∃x.P

Query Q:

Q(x1, …, xk) = P

What does this query return ?

Q(f,l) = ∃z. (Actor(z,f,l) ∧∀x.(Casts(z,x) ⇒ ∃y.Movie(x,y,1940)))

Ans: the first/last name of all actors who acted only in 1940

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

Important Observation

Find all bars that serve all beers that Fred likes

A(x) = …..

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 some bar that serves some beer they like.

Q(x) = …..

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.

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

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.

$$Q(x) = \ldots$$

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.

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

Prudent Peter

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

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

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.

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

Prudent Peter

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

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

Cautious Carl

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

$$Q(x) = \ldots$$

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.

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

Prudent Peter

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

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

Cautious Carl

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

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

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.

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

Prudent Peter

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

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

Cautious Carl

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

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

Paranoid Paul

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

$$Q(x) = \ldots$$

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.

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

Prudent Peter

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

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

Cautious Carl

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

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

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 Relational Calculus

- As in datalog, one can write “unsafe” RC queries; they are also called *domain dependent* (answer depends on whether entire domain or active domain from the relations is considered)

```
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 (only depends on database)

Relational Calculus

How to write a complex SQL query:

- Write it in RC
- Translate RC to datalog
- 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¬ to SQL - 1

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

Q(x) = ….

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

From RC to Datalog¬ to SQL - 2

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¬ to SQL – 3

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

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

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

Step 1: Replace $\forall$ with $\exists$ 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 $P \wedge \neg\ Q$

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

From RC to Datalog¬ to SQL - 4

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 $\forall$ with $\exists$ 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¬ to SQL - 5

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

H(x,y)

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

```
H(x,y)   :- Likes(x,y),Serves(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¬ to SQL - 6

```
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

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE ……
```

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

From RC to Datalog¬ to SQL - 7

```
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

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Likes L2, Serves S
   WHERE … …)
```

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

From RC to Datalog¬ to SQL - 8

```
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

```
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¬ to SQL - 9

```
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))
```

Another (old) example: RC to Datalog¬ -1

Person(name)
Friend(name1, name2)
Enemy(name1, name2)

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

$$A(x) = Person(x) \wedge \exists y\ Friend(x, y) \wedge \forall z\ (Enemy(y, z) \Rightarrow Enemy(x, z))$$

Another (old) example: RC to Datalog¬ - 2

Person(name)
Friend(name1, name2)
Enemy(name1, name2)

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

$$A(x) = Person(x) \wedge \exists y\ Friend(x, y) \wedge \forall z\ (Enemy(y, z) \Rightarrow Enemy(x, z))$$

Note: there are two interpretations (of this type of questions)

(1) the friend y has to have at least one enemy, then [∃w Enemy(y, w)] will be added. Just append Enemy (y, w) to the final datalog rule

(2) if the friend y does not have any enemy then that friend y satisfies this condition, which is captured by the expression A(x) above.

- We are going to assume the second interpretation, for exam/homeworks, either is fine.

  Please mention whatever you are assuming.

Another (old) example: RC to Datalog¬ - 3

Person(name)
Friend(name1, name2)
Enemy(name1, name2)

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

$$A(x) = Person(x) \wedge \exists y\ Friend(x, y) \wedge \forall z\ (Enemy(y, z) \Rightarrow Enemy(x, z))$$

$$\equiv \quad Person(x) \wedge \exists y\ Friend(x, y) \wedge \forall z\ (\neg Enemy(y, z) \vee Enemy(x, z))$$

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

$$A(x) = Person(x) \wedge \exists y\ Friend(x, y) \wedge \neg \exists z\ (Enemy(y, z) \wedge \neg Enemy(x, z))$$

Another (old) example: RC to Datalog¬

Person(name)
Friend(name1, name2)
Enemy(name1, name2)

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

$$A(x) = Person(x) \wedge \exists y\ Friend(x, y) \wedge \forall z\ (Enemy(y, z) \Rightarrow Enemy(x, z))$$

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

$$A(x) = Person(x) \wedge \exists y\ Friend(x, y) \wedge \neg \exists z\ (Enemy(y, z) \wedge \neg Enemy(x, z))$$

Step 2: Make all *subqueries* domain independent

$$A(x) = Person(x) \wedge \exists y\ Friend(x, y) \wedge \neg \exists z\ (Friend(x, y) \wedge Enemy(y, z) \wedge \neg Enemy(x, z))$$

Another (old) example: RC to Datalog¬ -4

Person(name)
Friend(name1, name2)
Enemy(name1, name2)

$$A(x) = Person(x) \wedge \exists y\ Friend(x, y) \wedge \neg \exists z\ (Friend(x, y) \wedge Enemy(y, z) \wedge \neg Enemy(x, z))$$

NonAnsFriend(x,y)

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

```
NonAnsFriend(x,y) :- Friend(x, y) , Enemy (y, z), NOT  Enemy (x, z)
A(x)      :-  Person(x), Friend(x,y),  NOT NonAnsFriend(x,y)
```

Summary: all these formalisms are equivalent!

- We have seen these translations:
  - RA → datalog¬
  - RC → datalog¬
- Practice at home, or read *Query Language Primer:*
  - Nonrecursive datalog¬ → RA
  - RA → RC
- Summary:
  - RA, RC, and non-recursive datalog¬ can express the same class of queries, called Relational Queries