CSE 544
Relational Calculus

Lecture #2
January 11th, 2011
Announcements

• HW1 is posted

• First paper review due next week

• Friday: project groups are due
Announcements

• Guest lecture on Tuesday, 1/18, by Bill Howe
  SQLShare: Smart Services for Ad Hoc Databases

• Need to makeup **one** lecture this/next week. When ?
  – Friday 1/14, 12:30-2pm ?
  – Wednesday, 1/19, morning (9…12) ?
  – Friday, 1/21, late morning (11…1pm) ?
Today’s Agenda

Relational calculus:
• Domain Relational Calculus
• Non-recursive datalog (a reasonable abstraction of SQL)
• Relational algebra

They are equivalent and why we care
Domain Relational Calculus

Given:

- A vocabulary: $R_1, \ldots, R_k$
- An arity, $ar(R_i)$, for each $i=1,\ldots,k$
- An infinite supply of variables $x_1, x_2, x_3,$ ...
- Constants: $c_1, c_2, c_3,$ ...
Domain Relational Calculus

Terms \( t \) and Formulas \( \varphi \) are:

\[

t ::= x \mid a \quad /* \text{variable or constant} */ \\
\varphi ::= R(t_1, \ldots, t_k) \mid t_i = t_j \\
\quad \mid \varphi \land \varphi' \mid \varphi \lor \varphi' \mid \neg \varphi' \\
\quad \mid \forall x.\varphi \mid \exists x.\varphi
\]

A query \( Q \) is any formula \( \varphi \), usually written as:

\[
Q = \{x \mid \varphi\}
\]

where \( x = (x_1,x_2,\ldots) \) are the free variables in \( \varphi \).
Example: Querying a Graph

R encodes a graph

What do these queries return?

\[ \{ x \mid \exists y. R(x, y) \} \]

\[ \{ x \mid \exists y. \exists z. \exists u. (R(x, y) \land R(y, z) \land R(z, u)) \} \]

\[ \{ (x, y) \mid \forall z. (R(x, z) \implies R(y, z)) \} \]
Example: Querying a Graph

R encodes a graph

What do these queries return?

\{x \mid \exists y. R(x,y)\}

Nodes that have at least one child: \{1,2,3\}

\{x \mid \exists y. \exists z. \exists u. (R(x,y) \land R(y,z) \land R(z,u))\}

Nodes that have a great-grand-child: \{1,2\}

\{(x,y) \mid \forall z. (R(x,z) \rightarrow R(y,z))\}

Every child of x is a child of y:
\{(1,1),(2,2),(3,1),(3,3),(4,1),(4,2),(4,3)\}

R encodes a graph

\[
R = \{ x \mid \exists y. R(x,y) \}
\]

Nodes that have at least one child: \{1,2,3\}

\[
\{ x \mid \exists y. \exists z. \exists u. (R(x,y) \land R(y,z) \land R(z,u)) \}
\]

Nodes that have a great-grand-child: \{1,2\}

\[
\{(x,y) \mid \forall z. (R(x,z) \rightarrow R(y,z)) \}
\]

Every child of x is a child of y:
\{(1,1),(2,2),(3,1),(3,3),(4,1),(4,2),(4,3)\}
Semantics of a Formula

• A database instance (or a model) is
  \[ D = (D, R_1^D, \ldots, R_k^D) \]
  – \( D \) = a set, called domain, or universe
  – \( R_i^D \subseteq D \times D \times \ldots \times D \), \( (\text{ar}(R_i) \text{ times}) \)
    \[ i = 1, \ldots, k \]

• A valuation is a function \( s : \{x_1, x_2, \ldots\} \rightarrow D \)

• The semantics of a formula \( \varphi \) says when
  \( D \vDash \varphi[s] \) holds

The domain \( D \) is not part of the database, yet we need it to define the semantics. Where?
Semantics of \( D \models \varphi[s] \)

- \( D \models (R(t_1, ..., t_n))[s] \)
  If \((s(t_1), ..., s(t_n)) \in R^D\)

- \( D \models (t = t')[s] \)
  If \(s(t) = s(t')\)

- \( D \models (\varphi \land \varphi')[s] \)
  If \(D \models \varphi[s]\) and \(D \models (\varphi')[s]\)

- \( D \models (\varphi \lor \varphi')[s] \)
  If \(D \models \varphi[s]\) or \(D \models \varphi'[s]\)

- \( D \models (\neg \varphi')[s] \)
  If it is not the case \(D \models \varphi[s]\)
Semantics of $D \models \varphi[s]$

Notation: $s_{a/x} = $ the valuation $s$ modified to map $x$ to $a$

If for every constant $a$ in $D$, $D \models \varphi[s_{a/x}]$

If for some constant $a$ in $D$, $D \models \varphi[s_{a/x}]$
Semantics of $D \models \varphi[s]$

Notation: $s_{a/x} = \text{the valuation } s \text{ modified to map } x \text{ to } a$

- $D \models (\forall x. \varphi)[s]$  
  If for every constant $a$ in $D$,  
  $D \models \varphi[s_{a/x}]$

- $D \models (\exists x. \varphi)[s]$  
  If for some constant $a$ in $D$,  
  $D \models \varphi[s_{a/x}]$

Note: here we need the domain $D$
Semantics of a Query

The semantics of a query $Q = \{ \mathbf{x} \mid \varphi \}$ is

$$Q(D) = \{ s(\mathbf{x}) \mid \text{for all } s \text{ such that } D \models \varphi[s] \}$$
The drinkers-bars-beers example

What does this query compute?

\[
x: \; \exists y. \; \exists z. \; \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z)
\]
The drinkers-bars-beers example

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

What does this query compute?

\[ x: \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

Find drinkers that frequent some bar that serves some beer they like.
The drinkers-bars-beers example

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

What does this query compute?

\[ x: \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y, z) \land \text{Likes}(x, z)) \]
The drinkers-bars-beers example

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

What does this query compute?

Find drinkers that frequent only bars that serves some beer they like.
The drinkers-bars-beers example

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

What does this query compute?

\[ x: \exists y. \text{Frequents}(x, y) \land \forall z. (\text{Serves}(y, z) \implies \text{Likes}(x, z)) \]
The drinkers-bars-beers example

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

What does this query compute?

\[ x : \exists y. \text{Frequents}(x, y) \land \forall z. (\text{Serves}(y, z) \Rightarrow \text{Likes}(x, z)) \]

Find drinkers that frequent some bar that serves only beers they like.
The drinkers-bars-beers example

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

What does this query compute?

x: ∀y.Frequents(x, y) ⇒ ∀z.(Serves(y,z) ⇒ Likes(x,z))
The drinkers-bars-beers example

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

\[ \forall y. \text{Frequents}(x, y) \Rightarrow \forall z. (\text{Serves}(y, z) \Rightarrow \text{Likes}(x, z)) \]

What does this query compute?
Summary of Relational Calculus

• Same as First Order Logic
• See book for the two variants:
  – Domain relational calculus (what we discussed)
  – Tuple relational calculus
• This is a powerful, concise language
Datalog Rule

\[ Q(x) \quad :- \quad L_1, \quad L_2, \quad \ldots, \quad L_k \]

\( L_i \) = a *literal*, or *atom*, or *subgoal*; can be one of:

- \( R(x_1, x_2, \ldots) = \) relational atom
- \( u = v, \) or \( u \neq v, \) where \( u, v = \) variables or constants
- \( S = \) a new symbol
- \( x = (x_1, x_2, \ldots) \) are *head variables*

Example:  
\[ Q(x) \quad :- \quad \text{Likes}(x,y),\text{Likes}(x,z), y \neq z \]
Semantics of Datalog Rule

\[ Q(x) : - L_1, L_2, \ldots, L_k \]

\[ Q = \{ x | \exists y_1.\exists y_2 \ldots L_1 \wedge L_2 \wedge \ldots \wedge L_k \} \]

\[ y_1, y_2, \ldots \text{ are all non-head variables} \]

The semantics is:

\[ Q(D) : - \{ s(x) | s = \text{valuation s.t. } D \models L_1[s], \ldots, D \models L_k[s] \} \]

Note: a datalog rule is also called a \textit{conjunctive query}
Non-Recursive Datalog

Each symbol $S_k$ may appear in body$_{k+1}$, body$_{k+2}$, ..., but may not appear in body$_1$, ..., body$_k$.

Example:

What does Q compute?

A(x,y) :- R(x,z),R(z,y)
B(x,y) :- A(x,z),A(z,y)
C(x,y) :- B(x,z),B(z,y)
Q(x,y) :- C(x,z),C(z,y)
Non-Recursive Datalog

\[ S_1(x_1) :- \text{body}_1 \]
\[ S_2(x_2) :- \text{body}_2 \]
\[ S_3(x_3) :- \text{body}_3 \]
\[ \ldots \]

Each symbol \( S_k \) may appear in \( \text{body}_{k+1}, \text{body}_{k+2}, \ldots \), but may not appear in \( \text{body}_1, \ldots, \text{body}_k \).

Example:

\[ A(x,y) :- R(x,z), R(z,y) \]
\[ B(x,y) :- A(x,z), A(z,y) \]
\[ C(x,y) :- B(x,z), B(z,y) \]
\[ Q(x,y) :- C(x,z), C(z,y) \]

What does \( Q \) compute?

A: all pairs \((x,y)\) connected by a path of length 16.
Comparison

Is non-recursive datalog equivalent to the Relational Calculus?
Comparison

Is non-recursive datalog equivalent to the Relational Calculus?

What about writing this query in non-recursive datalog?

\[
Q = \{ x \mid \exists y. \text{Frequents}(x,y) \land \text{Serves}(y,\text{’Bud’}) \lor \\
\exists z. \text{Frequents}(x,z) \land \text{Serves}(z,\text{’Miller’}) \}
\]
Comparison

Is non-recursive datalog equivalent to the Relational Calculus?

What about writing this query in non-recursive datalog?

\[ Q = \{ x \mid \exists y. \text{Frequents}(x,y) \land \text{Serves}(y,'Bud') \lor \exists z. \text{Frequents}(x,z) \land \text{Serves}(z,'Miller') \} \]

\[
Q(x) :- \text{Frequents}(x,y), \text{Serves}(y,'Bud')
\]

\[
Q(x) :- \text{Frequents}(x,z), \text{Serves}(z,'Miller')
\]
Comparison

Is non-recursive datalog equivalent to the Relational Calculus?

Now what about writing this query in non-recursive datalog?

\[ Q = \{ x \mid \exists y. \text{Frequents}(x,y) \land \neg \exists z. \text{Likes}(x,z) \land \text{Serves}(y,z) \} \]
Comparison

Is non-recursive datalog equivalent to the Relational Calculus?

Now what about writing this query in non-recursive datalog?

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

Impossible! Proof on next slides.
Monotone Queries

• Given two database instances $D = (D, R_1, \ldots, R_k)$ and $D' = (D', R'_1, \ldots, R'_k)$ we write $D \preceq D'$ if $D \subseteq D'$, $R_1 \subseteq R'_1, \ldots, R_k \subseteq R'_k$.

• In other words, $D'$ is obtained by adding tuples to $D$.

• We say that $Q$ is monotone if whenever $D \subseteq D'$, $Q(D) \subseteq Q(D')$. 
Monotone Queries

**Fact.** Every datalog rule is monotone. Why?

**Fact.** Every datalog program is monotone. Why?

**Fact.** This query is not monotone:
\[ Q = \{ x \mid \exists y. \text{Likes}(x, y) \land \neg \exists z. \text{Frequents}(x, z) \land \text{Serves}(z, y) \} \] Why?
Monotone Queries

Fact. Every datalog rule is monotone.

Recall the semantics: \( Q(D) = \{ s(x) \mid D \models L_1[s], \ldots, D \models L_k[s] \} \)

If \( D \models L_i[s] \) then \( D' \models L_i[s] \) hence \( Q(D) \subseteq Q(D') \).

Fact. Every datalog program is monotone.

Fact. This query is not monotone:
\[
Q = \{ x \mid \exists y. \text{Likes}(x, y) \land \neg \exists z. \text{Frequents}(x, z) \land \text{Serves}(z, y) \}
\]
Monotone Queries

**Fact.** Every datalog rule is monotone.

Recall the semantics: \( Q(D) = \{ s(x) \mid D \models L_1[s], \ldots, D \models L_k[s] \} \)

If \( D \models L_i[s] \) then \( D' \models L_i[s] \) hence \( Q(D) \subseteq Q(D') \).

**Fact.** Every datalog program is monotone.

By induction on the number of rules

**Fact.** This query is not monotone:
\[
Q = \{ x \mid \exists y. \text{Likes}(x,y) \land \lnot \exists z. \text{Frequents}(x,z) \land \text{Serves}(z,y) \}
\]
Monotone Queries

**Fact.** Every datalog rule is monotone.

Recall the semantics: 

\[ Q(D) = \{ s(x) \mid D \models L_1[s], \ldots, D \models L_k[s] \} \]

If \( D \models L_i[s] \) then \( D' \models L_i[s] \) hence \( Q(D) \subseteq Q(D') \).

**Fact.** Every datalog program is monotone.

By induction on the number of rules

**Fact.** This query is not monotone:

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

By adding a tuple to Frequents we may remove some answer
Monotone Queries

Recall the relational calculus:

\[
\begin{align*}
  t & ::= x \mid a \quad /* \text{variable or constant} */ \\
  \varphi & ::= R(t_1, \ldots, t_k) \mid t_i = t_j \\
           & \quad \mid \varphi \land \varphi' \mid \varphi \lor \varphi' \mid \neg \varphi' \\
           & \quad \mid \forall x.\varphi \mid \exists x.\varphi
\end{align*}
\]

What fragment of the relational calculus is monotone?
Monotone Queries

Recall the relational calculus:

\[
\begin{align*}
t & ::= x \mid a \quad /* \text{variable or constant} */ \\
\varphi & ::= R(t_1, \ldots, t_k) \mid t_i = t_j \\
& \quad \mid \varphi \land \varphi' \mid \varphi \lor \varphi' \mid \neg \varphi' \\
& \quad \mid \forall x.\varphi \mid \exists x.\varphi
\end{align*}
\]

What fragment of the relational calculus is monotone?

\[
\begin{align*}
t & ::= x \mid a \quad /* \text{variable or constant} */ \\
\varphi & ::= R(t_1, \ldots, t_k) \mid t_i = t_j \\
& \quad \mid \varphi \land \varphi' \mid \varphi \lor \varphi' \mid \exists x.\varphi
\end{align*}
\]
Non-Recursive Datalog

\[
Q(x) \; :- \; L_1, \; L_2, \; \ldots, \; L_k
\]

\(L_i = \) may also be \( \neg R(x_1, x_2, \ldots) \)

- \(R(x_1, x_2, \ldots) = \) relational atom
- \(u = v, \) or \(u \neq v,\) where \(u, v = \) variables or constants
Non-Recursive Datalog

Example:

\[ Q = \{ x \mid \exists y. \text{Likes}(x,y) \land \neg \exists z. \text{Frequents}(x,z) \land \text{Serves}(z,y) \} \]
Non-Recursive Datalog

Example:

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

FS(x,y) :- Frequents(x,z), Serves(z,y)
Q(x)   :- Likes(x,y), \neg FS(x,y)
Theorem. Every query in the Relational Calculus can be translated to non-recursive Datalog\textsuperscript{¬}
Rel. Calculus $\rightarrow$ non-rec. Datalog

Proof

First remove $\forall$ by using De Morgan: $\forall x. \varphi = \neg \exists x. \neg \varphi$

Then, inductively, for each subformula $\varphi$ create a datalog rule $F(x):$-body that computes $Q = \{ x | \varphi \}$
Rel. Calculus $\Rightarrow$ non-rec. Datalog

\[ R(t_1, \ldots, t_k) \quad \Rightarrow \quad F(x) :- R(t_1, \ldots, t_k) \]
Rel. Calculus $\rightarrow$ non-rec. Datalog

$\varphi_1 \land \varphi_2$

$F_1(x_1) \leftarrow \ldots \quad \text{/* for } \varphi_1 \text{ */}$

$F_2(x_2) \leftarrow \ldots \quad \text{/* for } \varphi_2 \text{ */}$
Rel. Calculus $\Rightarrow$ non-rec. Datalog

$\varphi_1 \land \varphi_2$

$F_1(x_1) : - \ldots \quad */ \text{for } \varphi_1 */$

$F_2(x_2) : - \ldots \quad */ \text{for } \varphi_2 */$

$F(x) : - F_1(x_1), F_2(x_2)$
Rel. Calculus \rightarrow \text{non-rec. Datalog}

\begin{align*}
\varphi_1 \lor \varphi_2
\end{align*}

\begin{align*}
F_1(x_1) &:- \ldots /* \text{for } \varphi_1 */ \\
F_2(x_2) &:- \ldots /* \text{for } \varphi_2 */
\end{align*}
Rel. Calculus $\Rightarrow$ non-rec. Datalog

$\varphi_1 \lor \varphi_2$

\[ F_1(x_1) : \cdots \quad /* \text{for } \varphi_1 */ \]
\[ F_2(x_2) : \cdots \quad /* \text{for } \varphi_2 */ \]

\[ F(x) : \text{F}_1(x_1) \]
\[ F(x) : \text{F}_2(x_2) \]
Rel. Calculus $\rightarrow$ non-rec. Datalog

$\exists x. \varphi_1 \rightarrow \ldots$ /* for $\varphi_1$ */
Rel. Calculus $\Rightarrow$ non-rec. Datalog

$\exists x. \varphi_1$  $\Rightarrow$  $F(y) :- F_1(x)$

where $y = x - \{x\}$
Rel. Calculus $\rightarrow$ non-rec. Datalog

$\neg \varphi_1$ $\rightarrow$ $F(x) \ :- \ \neg F_1(x)$

F_1(x) \ :- \ … /* for \varphi_1 */

End of Proof
Unsafe Queries

Q(x) :- x=y

What’s wrong?

Q(x) :- ¬R(x,y)
Definition. A datalog rule $S : - L_1, \ldots, L_k$ is called **safe**, if every variable $x$ appears in at least one positive relational atom.

Examples

\[ Q(x) : - T(x), S(y), x = y \]

\[ Q(x) : - T(x), S(y), \neg R(x, y) \]

We require that all rules of a non-recursive datalog program to be safe.
Unsafe Queries

But what about Relational Calculus? What does it mean for a query $Q$ to be safe?

Are these queries safe?

$Q = \{ x \mid \forall y. \text{Frequents}(x,y) \}$

$Q = \{ (x,y) \mid \text{Faculty}(x) \lor \text{Student}(y) \}$
Unsafe Queries

**Definition.** A query \( Q \) is **domain independent**, if for any two database instances \( D, D' \), such that they have the same relations \( R_1, ..., R_k \) but possibly different domains \( D, D' \), \( Q(D) = Q(D') \).

A safe datalog rule is domain independent (obvious). Does the converse hold?

**Note:** domain independent queries are also called **safe** queries, somewhat confusingly.
Unsafe Queries

Explain why these queries are not domain independent:

\[ Q = \{ x \mid \forall y. \text{Frequents}(x, y) \} \]

\[ Q = \{ (x, y) \mid \text{Faculty}(x) \lor \text{Student}(y) \} \]
Unsafe Queries

**Theorem.** The problem: “given a relational query Q decide whether Q is safe” is undecidable.

Bummer !...

Why doesn’t the following work?
Translate Q to a datalog program, then check that each rule is safe.
Is this a decision procedure for domain independence?
Active Domain Semantics

**Definition.** The *active domain* $\text{ADom}$ of a database instance $D = (D, R_1^D, \ldots, R_k^D)$, is the set of all constants in $R_1^D, \ldots, R_k^D$.

The active domain can be computed by a boring relational query:

- $\text{ADom}(x) :- \text{Frequents}(x,y)$
- $\text{ADom}(y) :- \text{Frequents}(x,y)$
- $\text{ADom}(x) :- \text{Likes}(x,y)$
- $\text{ADom}(y) :- \text{Likes}(x,y)$
- $\text{ADom}(x) :- \text{Serves}(x,y)$
- $\text{ADom}(y) :- \text{Serves}(x,y)$
Active Domain Semantics

Make the following two changes to standard semantics:

\[ D \models (\forall x. \varphi) [s] \]

If for every constant \( a \) in \( \text{ADom} \),
\[ D \models \varphi[s_{a/x}] \]

\[ D \models (\exists x. \varphi) [s] \]

If for some constant \( a \) in \( \text{ADom} \),
\[ D \models \varphi[s_{a/x}] \]
Active Domain Semantics

**Theorem.** If $Q$ is domain-independent, then its standard semantics coincides with the active domain semantics

Why?
Theorem. If Q is domain-independent, then its standard semantics coincides with the active domain semantics.

Why?

Let $D = (D, R_1^D, \ldots, R_k^D)$, be a database instance. Denote $D' = (ADom, R_1^D, \ldots, R_k^D)$.

Since Q is d.i. we have $Q(D) = Q(D')$.

$Q(D')$ is the same as the active-domain semantics on $D$. 
Rel. Calculus $\rightarrow$ non-rec. Datalog$^\neg$

If $Q$ is *domain independent*,
then we want to translate it into
a *safe* non-rec. datalog$^\neg$ program

$$\neg \varphi_1 \rightarrow F(x) : \neg F_1(x)$$

This rule is unsafe!
How do we make it safe?
Rel. Calculus $\Rightarrow$ non-rec. Datalog$^-$

If Q is *domain independent*, then we want to translate it into a *safe* non-rec. datalog$^-$ program

$$\neg \varphi_1$$

$$F(\mathbf{x}) :\neg \text{Adom}(x_1), \text{Adom}(x_2), \ldots, \text{Adom}(x_k), \neg F_1(\mathbf{x})$$

Where $\mathbf{x} = (x_1, x_2, \ldots, x_k)$
Discussion

• Non-recursive datalog is a simple, friendly, very popular notation for queries
• Naturally compositional: sequence of rules
• Exposes a hierarchy of queries
  – Single rule datalog = conjunctive queries
  – Non-rec. datalog program = monotone-RC
  – Non-rec. datalog program = RC
• No quantifiers! Because all are $\exists$
Why We Care

SQL is non-recursive datalog - !

Q(x) :- R(..), S(..), T(..)

Q1(x) :- ...
Q2(y) :- ...
Q(z) :- Q1(x), Q2(y)

SELECT x FROM R, S, T WHERE ...

Subqueries in the FROM clause

Subquery in the WHERE clause
Why We Care

• When we use a database system we need to write some complex queries
  – *The Masochists*: Find drinkers that frequent only bars that server no beer that they like
• Write it first in Relational Calculus
  – Easy exercise
• Translate it to Non-recursive datalog \( ^\neg \)
  – This is subtle, but we have seen it in detail
• Translate Non-recursive datalog \( ^\neg \) to SQL
  – Easy exercise
Relational Algebra

The Basic Five operators:

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

Plus: cartesian product $\times$, renaming $\rho$

We will study RA in detail when we discuss Query Execution
Complex RA Expressions

$\Pi_{\text{u.name}}$

$x.ssnn=\text{y.buyer-ssn}$

$\Pi_{\text{ssn}}$

$\sigma_{\text{name}=\text{fred}}$

$\Pi_{\text{pid}}$

$\sigma_{\text{name}=\text{gizmo}}$

Person x

Purchase y

Person z

Product u
Equivalence Theorem

**Theorem.** Every domain independent query in RC can be translated to safe non-recursive Datalog

Done that.

**Theorem.** Every safe non-recursive Datalog can be translated to Relational Algebra.

Easy exercise – to do at home

**Theorem.** Relational Algebra expressions can be translated to a domain independent RC query

Easy exercise – to do at home
Discussion

• What is the monotone fragment of RA?
  –

• What are the safe queries in RA?
  –

• Where do we use RA (applications)?
Discussion

• What is the monotone fragment of RA?
  – No difference: ∪, σ, Π, ⋈

• What are the safe queries in RA?
  – All RA queries are safe

• Where do we use RA (applications)?
  – Translating SQL (WHAT → HOW)
  – Some new query languages (Pig-Latin)