CompSci 516
Data Intensive Computing Systems

Lecture 21
Datalog

Instructor: Sudeepa Roy
Announcement

• HW3 due next Wednesday: 11/16
Today

• Datalog
  – for recursion in database queries

• A quick look at Incremental View Maintenance (IVM)
Reading Material: Datalog

Optional:
1. The datalog chapters in the “Alice Book”
   Foundations of Databases
   Abiteboul-Hull-Vianu
   Available online: http://webdam.inria.fr/Alice/

2. Datalog tutorial
   SIGMOD 2011
   “Datalog and Emerging Applications: An Interactive Tutorial”
Brief History of Datalog

• Motivated by Prolog – started back in 1970-80’s – then quiet for a long time

• A long argument in the Database community whether recursion should be supported in query languages
  – “No practical applications of recursive query theory ... have been found to date” — Michael Stonebraker, 1998
    Readings in Database Systems, 3rd Edition Stonebraker and Hellerstein, eds.
  – Recent work by Hellerstein et al. on Datalog-extensions to build networking protocols and distributed systems. [Link]
Datalog is resurging!

• Number of papers and tutorials in DB conferences

• Applications in
  – data integration, declarative networking, program analysis, information extraction, network monitoring, security, and cloud computing

• Systems supporting datalog in both academia and industry:
  – Lixto (information extraction)
  – LogicBlox (enterprise decision automation)
  – Semmle (program analysis)
  – BOOM/Dedalus (Berlekey)
  – Coral
  – LDL++
Recall our drinker example in RC (Lecture 4)

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

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

Drinker example is from slides by Profs. Balazinska and Suciu and the [GUW] book
Write it as a Datalog Rule

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

RC:
Q(x) = ∃y. ∃z. Frequents(x, y) ∧ Serves(y, z) ∧ Likes(x, z)

Datalog:
Q(x) :- Frequents(x, y), Serves(y, z), Likes(x, z)
Write it as a Datalog Rule

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

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

**Datalog:**
\[ Q(x) \leftarrow \text{Frequents}(x, y), \text{Serves}(y, z), \text{Likes}(x, z) \]

- **Quick differences:**
  - Uses “:-” not =
  - no need for \( \exists \) (assumed by default)
  - Use “,” on the right hand side (RHS)
  - Anything on RHS the of :- is assumed to be combined with \( \land \) by default
  - \( \forall, \Rightarrow \), not allowed – they need to use negation \( \neg \)
  - Standard “Datalog” does not allow negation
  - Negation allowed in datalog with negation

- **How to specify disjunction (OR / \( \lor \))?**
Example: OR in Datalog

Find drinkers that (a) either frequent some bar that serves some beer they like, (b) or like beer “BestBeer”

RC:
Q(x) = [∃y. ∃z. Frequents(x, y) ∧ Serves(y, z) ∧ Likes(x, z)] ∨ [Likes(x, “BestBeer”)]

Datalog:
Q(x) :- Frequents(x, y), Serves(y, z), Likes(x, z)
Q(x) :- Likes(x, “BestBeer”)
Example: OR in Datalog

Find drinkers that (a) either frequent some bar that serves some beer they like, (b) or like beer “BestBeer”, (c) or, frequent bars that “Joe” frequents.

RC:
Q(x) = [∃y. ∃z. Frequents(x, y) ∧ Serves(y,z) ∧ Likes(x,z)] ∨ [Likes(x, “BestBeer”)]
   ∨ [∃w Frequents(x, w) ∧ Frequents(“Joe”, w)]

Datalog:
JoeFrequents(w) :- Frequents(“Joe”, w)
Q(x) :- Frequents(x, y), Serves(y,z), Likes(x,z)
Q(x) :- Likes(x, “BestBeer”)
Q(x) :- Frequents(x, w), JoeFrequents(w)

• To specify “OR”, write multiple rules with the same “Head”
• Next: terminology for Datalog
Each rule is of the form \texttt{Head : - Body}

Each variable in the head of each rule must appear in the body of the rule.

- \texttt{JoeFrequents(w) : - Frequents("Joe", w)}
- \texttt{Q(x) : - Frequents(x, y), Serves(y,z), Likes(x,z)}
- \texttt{Q(x) : - Likes(x, "BestBeer")}
- \texttt{Q(x) : - Frequents(x, w), JoeFrequents(w)}
EDBs and IDBs

• **Extensional DataBases (EDBs)**
  - Input relation names
  - e.g. Likes, Frequents, Serves
  - can only be on the RHS of a rule

```
JoeFrequents(w) :- Frequents(“Joe”, w)
Q(x) :- Frequents(x, y), Serves(y,z), Likes(x,z)
Q(x) :- Likes(x, “BestBeer”)  
Q(x) :- Frequents(x, w), JoeFrequents(w)
```

• **Intensional DataBases (IDBs)**
  - Relations that are derived
  - Can be intermediate or final output tables
  - e.g. JoeFrequents, Q
  - Can be on the LHS or RHS (e.g. JoeFrequents)

Tuple in an EDB or an IDB: a FACT

either belongs to a given EDB relation, or is derived in an IDB relation
Graph Example

E (edge relation)

<table>
<thead>
<tr>
<th>V1</th>
<th>V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>
Example 1

Write a Datalog program to find paths of length two (output start and finish vertices)
Example 1

Write a Datalog program to find paths of length two (output start and finish vertices)

P2(x, y) :- E(x, z), E(z, y)
Example 1: Execution

Write a Datalog program to find paths of length two (output start and finish vertices)

\[ P2(x, y) : - E(x, z), E(z, y) \]

same as \( E \bowtie_{E.V2=\text{E.V1}} E \)
Write a Datalog program to find all pairs of vertices \((u, v)\) such that \(v\) is reachable from \(u\).

- Can you write a SQL/RA/RC query for reachability?
Can you write a SQL/RA/RC query for reachability?

NO - SQL/RA/RC cannot express reachability

Write a Datalog program to find all pairs of vertices (u, v) such that v is reachable from u
Write a Datalog program to find all pairs of vertices \((u, v)\) such that \(v\) is reachable from \(u\)

\[
\begin{align*}
R(x, y) &: E(x, y) \\
R(x, y) &: E(x, z), R(z, y)
\end{align*}
\]

Option 1
Example 2

Write a Datalog program to find all pairs of vertices (u, v) such that v is reachable from u

E (edge relation)

<table>
<thead>
<tr>
<th>V1</th>
<th>V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

Option 1

R(x, y) :- E(x, y)
R(x, y) :- E(x, z), R(z, y)

Option 2

R(x, y) :- E(x, y)
R(x, y) :- R(x, z), E(z, y)

Option 3

R(x, y) :- E(x, y)
R(x, y) :- R(x, z), R(z, y)

linear

non-linear
Linear Datalog

• Linear rule
  – at most one atom in the body that is recursive with the head of the rule
  – e.g. $R(x, y) :\neg E(x, z), R(z, y)$

• Linear datalog program
  – if all rules are linear
  – like linear recursion

• Top-down and bottom-up evaluation are possible
  – we will focus on bottom-up
**Example 2: Execution**

\[
R(x, y) :\text{ E}(x, y) \\
R(x, y) :\text{ E}(x, z), R(z, y)
\]

Option 1

(Vertices reachable in 1-hop by a direct edge)

### Table

<table>
<thead>
<tr>
<th>V1</th>
<th>V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

**Iteration 1**

\[ R = E \]
Example 2: Execution

R(x, y) :- E(x, y)
R(x, y) :- E(x, z), R(z, y)

Option 1

(Vertices reachable in 2-hops)
Example 2: Execution

**Option 1**

\[ R(x, y) :- E(x, y) \]
\[ R(x, y) :- E(x, z), R(z, y) \]

(Vertices reachable in 3-hops)
Example 2: Execution

R(x, y) :- E(x, y)
R(x, y) :- E(x, z), R(z, y)

Option 1

R unchanged - stop
Examples 3 and 4

Write a Datalog program to find all vertices reachable from b

R(x, y) :- E(x, y)
R(x, y) :- E(x, z), R(z, y)
QB(y) :- R(b, y)

Write a Datalog program to find all vertices u reachable from themselves R(u, u)

R(x, y) :- E(x, y)
R(x, y) :- E(x, z), R(z, y)
Q(x) :- R(x, x)
Termination of a Datalog Program

Q. A Datalog program always terminates – why?
Termination of a Datalog Program

Q. A Datalog program always terminates – why?

• Because the values of the variables are coming from the “active domain” in the input relations (EDBs)

• Active domain = (finite) values from the (possibly infinite) domain appearing in the instance of a database
  – e.g. age can be any integer (infinite), but active domain is only finitely many in R(id, name, age)

• Therefore the number of possible values in each of the IDBs is finite

• e.g. in the reachability example R(x, y), the values of x and y come from \{a, b, c, d, e\}
  – at most 5 x 5 = 25 tuples possible in the IDB R(x, y)
  – in any iteration, at least one new tuple is added in at least one IDB
  – Must stop after finite steps
  – e.g. the maximum number of iteration in the reachability example for any graph with five vertices is 25 (it was only 4 in our example)
Bottom-up Evaluation of a Datalog Program

• Naïve evaluation

• Semi-naïve evaluation
Naïve evaluation - 1

Iteration 1:
R = E = R1 (say)

In all subsequent iteration, check if any of the rules can be applied

Do union of all the rules with the same head IDB
Naïve evaluation - 2

Iteration 1:
\[ R = E = R1 \] (say)

Iteration 2:
\[ R = E \cup E \bowtie R1 \]
\[ = R2 \] (say)

\[ R1 \neq R2 \] so continue

\[ \begin{array}{c|c|c|c|c|c} 
V1 & V2 \\
\hline 
a & c \hline 
b & a \hline 
b & d \hline 
c & d \hline 
d & a \hline 
d & e 
\end{array} \]

\[ \begin{array}{c|c} 
V1 & V2 \\
\hline 
a & c \\
b & a \\
b & d \\
c & d \\
d & a \\
d & e 
\end{array} \]

\[ \begin{array}{c|c|c|c|c|c} 
V1 & V2 \\
\hline 
a & d \hline 
b & c \\
b & e \hline 
c & a \\
c & e \hline 
d & c 
\end{array} \]
Naïve evaluation - 3

Iteration 1:
R = E = R1 (say)

Iteration 2:
R = E ∪ E ⨝ R1 = R2 (say)

R1 ≠ R2 so continue

Iteration 3:
R = E ∪ E ⨝ R2 = R3 (say)

R2 ≠ R3 so continue
Naïve evaluation - 4

Iteration 1:
R = E = R1 (say)

Iteration 2:
R = E ∪ E ▷◁ R1
= R2 (say)

R1 ≠ R2 so continue

Iteration 3:
R = E ∪ E ▷◁ R2
= R3 (say)

R2 ≠ R3 so continue

Iteration 4:
R = E ∪ E ▷◁ R3
= R4 (say)

R3 = R4 so STOP
Problem with Naïve Evaluation

• The same IDB facts are discovered again and again
  – e.g. in each iteration all edges in E are included in R
  – In the 2\textsuperscript{nd}-4\textsuperscript{th} iterations, the first six tuples in R are computed repeatedly

• Solution: Semi-Naïve Evaluation

• Work only with the new tuples generated in the previous iteration
Semi-Naïve evaluation - 1

Initially:
R = Φ

Iteration 1:
R = E = R1 (say)
ΔR1 = R1
Semi-Naïve evaluation - 2

Initially: 
R = ∅

Iteration 1: 
R = E = R1 (say) 
ΔR1 = R1

Iteration 2: 
R = R1 ∪ E ⊖ ΔR1 = R2 (say)

ΔR2 = R2 – R1

ΔR2 ≠ ∅ so continue
Semi-Naïve evaluation - 3

Initially:
\[ R = \emptyset \]

\[ V1 \quad V2 \]
\[ a \quad c \]
\[ b \quad a \]
\[ b \quad d \]
\[ c \quad d \]
\[ d \quad a \]
\[ d \quad e \]

Iteration 1:
\[ R = E = R1 \text{ (say)} \]
\[ \Delta R1 = R1 \]
\[ V1 \quad V2 \]
\[ a \quad c \]
\[ b \quad a \]
\[ b \quad d \]
\[ c \quad d \]
\[ d \quad a \]
\[ d \quad e \]

\[ V1 \quad V2 \]
\[ a \quad d \]
\[ b \quad c \]
\[ b \quad e \]
\[ c \quad a \]
\[ c \quad e \]
\[ d \quad c \]

Iteration 2:
\[ R = R1 \cup E \bowtie \Delta R1 = R2 \text{ (say)} \]
\[ \Delta R2 = R2 - R1 \]
\[ \Delta R2 \neq \emptyset \text{ so continue} \]
\[ V1 \quad V2 \]
\[ a \quad c \]
\[ b \quad a \]
\[ b \quad d \]
\[ c \quad d \]
\[ d \quad a \]
\[ d \quad e \]

\[ V1 \quad V2 \]
\[ a \quad d \]
\[ b \quad c \]
\[ b \quad e \]
\[ c \quad a \]
\[ c \quad e \]
\[ d \quad c \]

Iteration 3:
\[ R = R2 \cup E \bowtie \Delta R2 = R3 \text{ (say)} \]
\[ \Delta R3 = R3 - R2 \]
\[ \Delta R3 \neq \emptyset \text{ so continue} \]
\[ V1 \quad V2 \]
\[ a \quad d \]
\[ b \quad c \]
\[ b \quad e \]
\[ c \quad a \]
\[ c \quad e \]
\[ d \quad c \]

\[ V1 \quad V2 \]
\[ a \quad e \]
\[ a \quad a \]
\[ c \quad c \]
\[ d \quad d \]
Semi-Naïve evaluation - 4

Initially:
R = \emptyset

Iteration 1:
R = E = R1 (say)
ΔR1 = R1

Iteration 2:
R = R1 ∪ E △ R1
ΔR2 = R2 − R1
ΔR2 ≠ \emptyset
so continue

Iteration 3:
R = R2 ∪ E △ ΔR2
ΔR3 = R3 − R2
ΔR3 ≠ \emptyset
so continue

Iteration 4:
R = R3 ∪ E △ ΔR3
ΔR4 = R4 − R3
ΔR = \emptyset
(CHECK 😊)
so STOP
Incremental View Maintenance (IVM)

• Why did the semi-naïve algorithm work?
• Because of the generic technique of Incremental View Maintenance (IVM)

• Suppose you have
  – a database D = (R1, R2, R3)
  – a query Q that gives answer Q(D)
  – D = (R1, R2, R3) gets updated to D’ = (R1’, R2’, R3’)
  – e.g. R1’ = R1 \cup ΔR1 (insertion), R2’ = R2 - ΔR1 (deletion) etc.
Incremental View Maintenance (IVM)

• Why did the semi-naïve algorithm work?
• Because of the generic technique of Incremental View Maintenance (IVM)

• Suppose you have
  – a database $D = (R_1, R_2, R_3)$
  – a query $Q$ that gives answer $Q(D)$
  – $D = (R_1, R_2, R_3)$ gets updated to $D' = (R_1', R_2', R_3')$
  – e.g. $R_1' = R_1 \cup \Delta R_1$ (insertion), $R_2' = R_2 - \Delta R_1$ (deletion) etc.

• IVM: Can you compute $Q(D')$ using $Q(D)$ and $\Delta R_1, \Delta R_2, \Delta R_3$ without computing it from scratch (i.e. do not rerun the query $Q$)?
**IVM Example: Selection**

- \( \sigma_{V1=b} (R \cup \Delta R) = \sigma_{V1=b} R \cup \sigma_{V1=b} \Delta R \)
- It suffices to apply the selection condition only on \( \Delta R \) – and include with the original solution

\[
\begin{array}{cc}
V1 & V2 \\
n & b & a \\
d & a \\
c & d \\
\end{array}
\]

\[
\begin{array}{cc}
V1 & V2 \\
\sigma_{V1=b} R \\
\end{array}
\]

\[
\begin{array}{cc}
V1 & V2 \\
\Delta R \\
\end{array}
\]

\[
\begin{array}{cc}
V1 & V2 \\
R' \Rightarrow R \cup \Delta R \\
\end{array}
\]

\[
\begin{array}{cc}
V1 & V2 \\
\sigma_{V1=b} \Delta R \\
\end{array}
\]

\[
\begin{array}{cc}
V1 & V2 \\
R \cup \Delta R \\
\end{array}
\]

\[
\begin{array}{cc}
V1 & V2 \\
\sigma_{V1=b} R \\
\end{array}
\]

\[
\begin{array}{cc}
V1 & V2 \\
\sigma_{V1=b} \Delta R \\
\end{array}
\]
**IVM Example: Projection**

\[
\pi_{V1}(R \cup \Delta R) = \pi_{V1} R \cup \pi_{V1} \Delta R
\]

- It suffices to apply the projection condition **only** on \(\Delta R\)
  - and include with the original solution
IVM Example: Join

\[ \begin{array}{cc}
A & B \\
a1 & b1 \\
a2 & b2 \\
a3 & b1 \\
\end{array} \bowtie \begin{array}{cc}
B & C \\
b1 & c1 \\
b2 & c2 \\
\end{array} = \begin{array}{ccc}
A & B & C \\
a1 & b1 & c1 \\
\end{array} \]

\[ \begin{array}{cc}
A & B \\
a1 & b1 \\
a2 & b2 \\
a3 & b1 \\
\end{array} \bowtie \begin{array}{cc}
B & C \\
b1 & c1 \\
b2 & c2 \\
\end{array} = \begin{array}{ccc}
A & B & C \\
a1 & b1 & c1 \\
\end{array} \]

R' = R ∪ ΔR

S' = S ∪ ΔS

\[ (R \cup ΔR) \bowtie (S \cup ΔS) = (R \bowtie S) \cup (R \bowtie ΔS) \cup (ΔR \bowtie S) \cup (ΔR \bowtie ΔS) \]
**IVM for Linear Datalog Rule**

\[
\begin{align*}
A & \quad B \\
a1 & \quad b1 \\
a2 & \quad b2 \\
a3 & \quad b1
\end{align*}
\]

\[\otimes\]

\[
\begin{align*}
B & \quad C \\
b1 & \quad c1
\end{align*}
\]

\[=\]

\[
\begin{align*}
A & \quad B & \quad C \\
a1 & \quad b1 & \quad c1
\end{align*}
\]

- \[R(x, y) :- E(x, z), R(z, y)\]
  - i.e. \[R_{\text{new}} = E \otimes R\]

- But \(E\) is EDB
  - \[\Delta E = \emptyset\]

- Therefore,
  \[E \otimes (R \cup \Delta R) = (E \otimes R) \cup (E \otimes \Delta R)\]

- It suffices to join with the difference \(\Delta R\) and include in the result in the previous round \(E \otimes R\)

- Advantage of having “linear rule”

\[
\begin{align*}
A & \quad B & \quad C \\
a1 & \quad b1 & \quad c1 \\
a3 & \quad b1 & \quad c1
\end{align*}
\]

\[(R \cup \Delta R) \otimes (S \cup \Delta S) = (R \otimes S) \cup (R \otimes \Delta S) \cup (\Delta R \otimes S) \cup (\Delta R \otimes \Delta S)\]
(Non-recursive) Datalog with Negation

• Recursion and negation together make Datalog execution complicated
  – there is a notion called “stratified semantic” for this purpose – compute IDB relations in strata/layers before taking a negation – not covered in this class

• We will only do negation for non-recursive Datalog
Datalog Rules

Unsafe/Safe Datalog Rules

Find drinkers who like beer “BestBeer”

\[ Q(x) : - \text{Likes}(x, "BestBeer") \]

Find drinkers who **DO NOT** like beer “BestBeer”

\[ Q(x) : - \neg \text{Likes}(x, "BestBeer") \]

- **What is the problem with this rule?**
- **What should this rule return?**
  - names of all drinkers in the world?
  - names of all drinkers in the USA?
  - names of all drinkers in Durham?
Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Find drinkers who like beer “BestBeer”

Find drinkers who DO NOT like beer “BestBeer”

Q(x) :- Likes(x, “BestBeer")
Q(x) :- ¬Likes(x, “BestBeer")

• What is the problem with this rule?
• Dependent on “domain” of drinkers
  – domain-dependent
  – infinite answers possible too..
    • keep generating “names”
Problem with Negation in Datalog Rules

Find drinkers who like beer “BestBeer”

Find drinkers who DO NOT like beer “BestBeer”

- Solution:
- Restrict to “active domain” of drinkers from the input *Likes* (or *Frequents*) relation
  – “domain-independence” – same finite answer always

- Becomes a “safe rule”
Query: Find drinkers that like some beer (so much) that they frequent all bars that serve it

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\text{Serves}(z, y) \Rightarrow \text{Frequents}(x, z)) \]
Query: Find drinkers that like some beer so much that they frequent all bars that serve it

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

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

Step 1: Replace \( \forall \) with \( \exists \) using de Morgan’s Laws

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

Ack: slides by Profs. Balazinska and Suciu
Query: Find drinkers that like some beer so much that they frequent all bars that serve it

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

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

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

(new) Step 2: Make all subqueries domain independent

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

Ack: slides by Profs. Balazinska and Suciu
Step 3:
Create a datalog rule for some subexpressions of the form

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

\[ H(x, y) \]

(new) Step 3: Create a datalog rule for some subexpressions of the form
\[ \exists x \, \exists y \ldots \, R(\ldots) \land S(\ldots) \land T(\ldots) \land \ldots \]

\[ H(x, y) : \text{Likes}(x, y), \text{Serves}(z, y), \neg \text{Frequents}(x, z) \]
\[ Q(x) : \text{Likes}(x, y), \neg H(x, y) \]

Ack: slides by Profs. Balazinska and Suciu
Step 4: Write it in SQL

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

RC → Datalog with negation → SQL (5/8)

Ack: slides by Profs. Balazinska andSuciu
Revisit example from Lecture 4

**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 ... ...)
```
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))
```

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

Possible Queries

RC → Datalog with negation → SQL (7/8)

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

Revisit example from Lecture 4

Ack: slides by Profs. Balazinska and Suciu
Sometimes can simplify the SQL query by using an unsafe datalog rule
Correctness ensured by safe outermost rule

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

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

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))

Ack: slides by Profs. Balazinska and Suciu
Optional/additional slides

An Overview of
Data Provenance
with Annotations

Selected/adapted slides from the keynote by
Prof. Val Tannen, EDBT 2010
(optional material: full slide deck is available on Val’s webpage)
Lineage

• [Cui-Widom-Wiener’00]

• Lineage:
  – Given a data item in the view
  – Determine the source data that produced it
  – The process by which it was produced
Applications

• **OLAP/OLAM (mining)**
  – origin of anomalous data to verify reliability

• **Scientific Databases**
  – how answer was produced from raw data

• **Online Network monitoring and Diagnosis system**
  – identify faulty sensor from network monitors
Applications

• Cleansed data feedback
  – Clean raw data and send report to sources

• Materialized view schema evolution
  – if view schema is changed (new column added), recomputation may not be necessary

• View update
  – translate view updates to base data updates
Data Provenance

**provenance, n.**

*The fact of coming from some particular source or quarter; origin, derivation* [Oxford English Dictionary]

- **Data provenance** [BunemanKhannaTan 01]: aims to explain how a particular result (in an experiment, simulation, query, workflow, etc.) was derived.

- Most science today is **data-intensive**. Scientists, eg., biologists, astronomers, worry about data provenance all the time.
Propagating annotations through database operations

The annotation $p \cdot r$ means joint use of data annotated by $p$ and data annotated by $r$.

Slide by Val Tannen, EDBT 2010
Another way to propagate annotations

The annotation $p + r$ means alternative use of data
Another use of +

\[ R \]

\[
\begin{array}{ccc}
A & B & C \\
\vdots & \vdots & \vdots \\
a & b & c_1 \\
\vdots & \vdots & \vdots \\
a & b & c_2 \\
\vdots & \vdots & \vdots \\
a & b & c_3 \\
\vdots & \vdots & \vdots \\
\end{array}
\]

\[ \pi_{AB} R \]

\[
\begin{array}{cc}
A & B \\
\vdots & \vdots \\
a & b \\
\vdots & \vdots \\
\end{array}
\]

\[ p + r + s \]

\[ + \text{ means alternative use of data} \]

Slide by Val Tannen, EDBT 2010
An example in positive relational algebra (SPJU)

\[ Q = \sigma_{C=e} \pi_{AC}(\pi_{AC}R \bowtie \pi_{BC}R \cup \pi_{AB}R \bowtie \pi_{BC}R) \]

For selection we multiply with two special annotations, 0 and 1.

Slide by Val Tannen, EDBT 2010
Provenance Example

Boolean query $Q() :\text{- HasAsthma}(x), \text{Friend}(x, y), \text{Smoker}(y)$

<table>
<thead>
<tr>
<th>HasAsthma</th>
<th>Friend</th>
<th>Smoker</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$y_1$</td>
<td>$z_1$</td>
</tr>
<tr>
<td>Ann</td>
<td>Ann</td>
<td>Joe</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$y_2$</td>
<td>$z_2$</td>
</tr>
<tr>
<td>Bob</td>
<td>Ann</td>
<td>Tom</td>
</tr>
<tr>
<td></td>
<td>$y_3$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bob</td>
<td>Tom</td>
</tr>
</tbody>
</table>

Provenance $F_{Q,D} = x_1y_1z_1 + x_1y_2z_2 + x_2y_3z_2$

- $x, y, z \in \{0, 1\}$
- $1 = \text{present}$
- $0 = \text{absent}$
- There are three alternative ways to derive the “true” answer
Application to “Deletion Propagation”

Boolean query $Q$: $\exists x \exists y \text{HasAsthma}(x) \land \text{Friend}(x, y) \land \text{Smoker}(y)$

Provenance $F_{Q,D} = x_1 y_1 z_1 + x_1 y_2 z_2 + x_2 y_3 z_2$

- $x, y, z \in \{0, 1\}$

- What happens if Ann is deleted?
  - Does the answer change to false from true?
Application to “Deletion Propagation”

Boolean query \( Q: \exists x \exists y \text{HasAsthma}(x) \land \text{Friend}(x, y) \land \text{Smoker}(y) \)

\[
\text{Provenance } F_{Q,D} = x_1 y_1 z_1 + x_1 y_2 z_2 + x_2 y_3 z_2
\]

- \( x, y, z \in \{0, 1\} \)
- What happens if Ann is deleted?
  - Does the answer change to false from true?
- No need to re-evaluate the query
  - just plug in \( x_1 = 0 \) and evaluate