Introduction to Data Management
CSE 344

Lecture 14: Datalog (cont.)
(Ch 5.3–5.4)
Announcements

- HW3 was due yesterday.

- HW4 will be up today. Due next Tuesday.
  - No coding, Datalog and RC on paper.

- WQ4 is due next Monday
  - it will be useful review for the midterm
  - finish it early if you have time

- Midterm on Friday, July 21h, in class…
  - All the web quizzes are open if that helps you study
Announcements

• Change in Syllabus

<table>
<thead>
<tr>
<th></th>
<th>Old</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Web Quiz</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Participation</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Midterm</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>Final</td>
<td>30</td>
<td>25</td>
</tr>
</tbody>
</table>

Final will be 1h and similar format to midterm. Focus on second half of class.
Introducing Cosette

Today we are thrilled to announce our official 1.0 release of Cosette, a SQL solver for automatically checking semantic equivalences of SQL queries. With Cosette, one can easily verify the correctness of SQL rewrite rules, find errors in buggy SQL rewrites, building auto-graders for SQL assignments, developing SQL optimizers, busting “fake SQLs,” etc.

https://medium.com/@uwdb/introducing-cosette-527898504bd6
Midterm

• Content
  – Lectures 1 through 13 (Monday)
  – HW 1–3, WQ 1–4

• Closed book. No computers, phones, watches, etc.!

• Can bring one letter-sized piece of paper with notes, but…
  – test will not be about memorization
  – formulas provided for join algorithms & selectivity
  – can ask me during test about anything you could look up

• Similar in format & content to CSE 344 17wi midterm
  – Previous midterms on course webpage
Midterm Concept Review I

• relational data model
  – set semantics vs bag semantics
  – primary & secondary keys
  – foreign keys
  – schemas

• SQL
  – CREATE TABLE
  – SELECT-FROM-WHERE (SFW)
  – joins: inner vs outer, natural
  – group by & aggregation
  – ordering
  – CREATE INDEX
Midterm Concept Review II

• relational queries
  – languages for writing them:
    • standard relational algebra
    • datalog (even without recursion)
    • SQL (even without grouping / aggregation)
  – monotone queries are a proper subset
  – SFW queries (i.e., w/out subqueries) are monotone

• Given an English problem statement you should be able to write a query in:
  – Relational Algebra, Relational Calculus
  – Datalog, SQL
Midterm Concept Review III

- types of indexes
  - B+ tree vs hash
    - hash indexes use at most 2 disk accesses
    - B+ tree can be used for < predicates
    - B+ tree index on (X,Y) also allows searching for X=a matches
  - clustered vs non-clustered
    - selectivity above 1-2% => not helped by non-clustered indexes

- cost-based query optimization
  - consider choices over logical and physical query plans
    - most important choice in latter is choice of join algorithm
    - those include nested loop, sorted merge, hash, and indexed joins
  - primary goal of the optimizer is to avoid really bad plans
Today

• More Datalog
• Midterm Review
What is Datalog?

- Another query language for relational model
  - Simple and elegant
  - Initially designed for recursive queries
  - Some companies use datalog for data analytics
    - e.g. LogicBlox
  - Increased interest due to recursive analytics

- We discuss only recursion-free or non-recursion-free datalog and add negation
Why Do We Learn Datalog?

• Datalog can be translated to SQL
  – Helps to express complex queries

• Increase in datalog interest due to recursive analytics

• A query language that is closest to mathematical logic
  – Good language to reason about query properties
  – Can show that:
    1. Non-recursive datalog & RA have equivalent power
    2. Recursive datalog is strictly more powerful than RA
    3. Extended RA & SQL92 is strictly more powerful than datalog

CSE 344 - Summer 2017
Datalog vs Relational Algebra

• Every expression in standard relational algebra can be expressed as a Datalog query

• But operations in the extended relational algebra (grouping, aggregation, and sorting) have no corresponding features in the version of datalog that we discussed today

• Similarly, datalog can express recursion, which relational algebra cannot
Datalog vs Relational Algebra

- Standard RA
- Datalog + neg
- Datalog + neg + recursion

Extended RA

Grouping & aggregation
Datalog with negation

Find all actors who only acted in 1994.

\[ A(x) :\neg \text{Actor}(x,_,_) \text{ Cast}(x,m) \text{ Movie}(m,_,1994) \]

\[ \text{Not} \ 1994(x) :\neg A(x,_,_ \text{ Cast}(x,m) \text{ Movie}(m,_,y) \ y \neq 1994 \]

\[ \text{nonAns}(x) :\neg \text{Actor}(x,_,_) \text{ Cast}(x,m) \text{ Movie}(m,_,y),y!\neq1994 \]

\[ A(x) :\neg \text{Actor}(x,_,_), \text{ not nonAns}(x) \]
Find all of Joe's friends who do not have any friends except for Joe:

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

• facts (extensional relations - EDBs) and rules (intensional relations - IDBs)
  – rules can use relations, arithmetic, union, intersect, …

• As with SQL, existential quantifiers are easier
  – use negation to handle universal
Using what we have learned

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
From RC to Datalog$^-$ to SQL

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 Datalog to SQL

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

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

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)
From RC to Datalog\(^-\) to SQL

**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

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

**Step 2:** Make sure the query is domain independent

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

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

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

H(x, y)

H(x, y) :- Likes(x, y), Serves(z, y), not Frequents(x, z)
Q(x) :- Likes(x, y), not H(x, y)
From RC to Datalog to SQL

Step 4: Write it in SQL

\[
\text{H}(x,y) \leftarrow \text{Likes}(x,y), \text{Serves}(z,y), \text{not Frequents}(x,z) \\
\text{Q}(x) \leftarrow \text{Likes}(x,y), \text{not H}(x,y)
\]

\[
\text{SELECT DISTINCT } L.\text{drinker} \text{ FROM Likes L} \\
\text{WHERE not exists } \\
(\text{SELECT * FROM Likes L2, Serves S} \\
\text{WHERE … …})
\]
From RC to Datalog™ to SQL

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))
```
From RC to Datalog to SQL

Unsafe rule

\[
H(x,y) \leftarrow \text{Likes}(x,y), \text{Serves}(z,y), \text{not Frequents}(x,z)
\]

\[
Q(x) \leftarrow \text{Likes}(x,y), \text{not } H(x,y)
\]

Improve the SQL query by using an unsafe datalog rule

\[
\text{SELECT DISTINCT } \text{L.drinker FROM Likes L}
\]
\[
\text{WHERE not exists (SELECT * FROM Serves S}
\]
\[
\text{WHERE L.beer=S.beer and not exists (SELECT * FROM Frequents F}
\]
\[
\text{WHERE F.drinker=L.drinker and F.bar=S.bar))}
\]
Summary: all these formalisms are equivalent!

• We have seen these translations:
  – RA $\rightarrow$ datalog$^-$
  – RC $\rightarrow$ datalog$^-$

• Practice at home, and read Query Language Primer:
  – Nonrecursive datalog$^-$ $\rightarrow$ RA
  – RA $\rightarrow$ RC

• Summary:
  – RA, RC, and non-recursive datalog$^-$ can express the same class of queries, called Relational Queries
Query Optimizer Summary

• **Input**: A logical query plan

• **Output**: A good physical query plan

• **Basic query optimization algorithm**
  – Enumerate alternative plans (logical and physical)
  – Compute estimated cost of each plan
    • Compute number of I/Os
    • Optionally take into account other resources
  – Choose plan with lowest cost
  – This is called cost-based optimization
Cost of Join Algorithms

• Nested Loop
  – $B(R) + B(R)B(S)$

• Nested Loop (with index)
  – If index on S is clustered: $B(R) + T(R)B(S)/V(S,A)$
  – If index on S is unclustered: $B(R) + T(R)T(S)/V(S,A)$

• Hash Join
  – $B(R) + B(S)$
  – uses more disk space when $B(R) > M$
Review: Physical Query
We only care about Disk I/O operations

Supplier \((\text{sid, sname, scity, sstate})\)

Supply \((\text{pno, sid, quanty})\)

\[
\begin{align*}
\text{T(Supplier)} &= 1000 \\
\text{T(Supply)} &= 10,000 \\
\text{B(Supplier)} &= 100 \\
\text{B(Supply)} &= 100 \\
\text{V(Supplier, scity)} &= 20 \\
\text{V(Supplier, state)} &= 10 \\
\text{V(Supply, pno)} &= 2,500 \\
\text{M} &= 11
\end{align*}
\]

\[
\text{SELECT sname} \\
\text{FROM Supplier x, Supply y} \\
\text{WHERE x.sid = y.sid} \\
\text{and y.pno = 2} \\
\text{and x.scity = 'Seattle'} \\
\text{and x.sstate = 'WA'}
\]
Physical Query: Naive Plan

$\pi_{\text{sname}}(\sigma_{\text{scity}='Seattle' \text{ and sstate}='WA' \text{ and pno}=2}(\text{Supplier } \bowtie \text{ Supply}))$

(On the fly) $\pi_{\text{sname}}$

(On the fly) $\sigma_{\text{scity}='Seattle' \text{ and sstate}='WA' \text{ and pno}=2}$

(Nested loop) $\text{sid} = \text{sid}$

(File scan) Supplier

(File scan) Supply

\begin{align*}
\text{T(Supplier)} & = 1000 \\
\text{B(Supplier)} & = 100 \quad \text{V(Supplier,scity)} = 20 \\
\text{T(Supply)} & = 10,000 \quad \text{B(Supply)} = 100 \quad \text{V(Supplier,sstate)} = 10 \\
& \quad \text{V(Supply,pno)} = 2,500 \quad \text{M} = 11
\end{align*}
Physical Query: Naive Plan

\[
\begin{align*}
T(\text{Supplier}) &= 1000 \\
T(\text{Supply}) &= 10,000 \\
B(\text{Supplier}) &= 100 \\
B(\text{Supply}) &= 100 \\
V(\text{Supplier}, \text{scity}) &= 20 \\
V(\text{Supplier}, \text{sstate}) &= 10 \\
V(\text{Supply}, \text{pno}) &= 2,500 \\
M &= 11
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad \text{sname} \\
\text{FROM} & \quad \text{Supplier} \ x, \ \text{Supply} \ y \\
\text{WHERE} & \quad x.\text{sid} = y.\text{sid} \\
& \quad \text{and} \ y.\text{pno} = 2 \\
& \quad \text{and} \ x.\text{scity} = \text{‘Seattle’} \\
& \quad \text{and} \ x.\text{sstate} = \text{‘WA’}
\end{align*}
\]

(On the fly) \quad \sigma_{\text{scity}=\text{‘Seattle’} \text{ and } \text{sstate}=\text{‘WA’} \text{ and } \text{pno}=2}

\text{Selection and project on-the-fly} \\
\rightarrow \text{No additional cost.}

(On the fly) \quad \pi_{\text{sname}}

\text{Total cost of plan is thus cost of join:}
\quad = B(\text{Supplier}) + B(\text{Supplier}) \times B(\text{Supply})
\quad = 100 + 100 \times 100
\quad = 10,100 \text{ I/Os}
Physical Query: Optimized

1. \( \sigma_{\text{scity}='Seattle' \text{ and } sstate='WA'} \)
   - \( \text{Supplier} \) (File scan)
   - \( \text{Supply} \) (File scan)

2. \( \sigma_{\text{pno}=2} \)
   - \( \text{Supplier} \) (File scan)
   - \( \text{Supply} \) (File scan)

3. (Sort-merge join) \( \pi_{\text{sname}} \)
   - \( \text{sid} = \text{sid} \)
   - \( \text{Scan write to } T1 \)
   - \( \text{Scan write to } T2 \)

4. (On the fly) \( \pi_{\text{sname}} \)

**Total cost**

(step 1) \( 100 + 100 \times 1/20 \times 1/10 \approx 100 \)

(step 2) \( 100 + 100 \times 1/2500 \approx 100 \)

(step 3) 2

(step 4) 0

Total cost \( \approx 204 \text{ I/Os} \)

\[ \text{SELECT sname} \]
\[ \text{FROM Supplier x, Supply y} \]
\[ \text{WHERE x.sid = y.sid} \]
\[ \text{and y.pno = 2} \]
\[ \text{and x.scity = 'Seattle'} \]
\[ \text{and x.sstate = 'WA'} \]

T(Supplier) = 1000
T(Supply) = 10,000
B(Supplier) = 100
B(Supply) = 100
V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500
M = 11
Physical Query: Using Indexes

1. \( \sigma_{pno=2} \) (Index on pno)
   - Assume: clustered
   - Total cost \approx 5 \text{ I/Os (or 6)}

2. \( \text{sid} = \text{sid} \) (Index on sid)
   - \( T(\text{Supplier}) = 1000 \)
   - \( T(\text{Supply}) = 10,000 \)
   - \( B(\text{Supplier}) = 100 \)
   - \( B(\text{Supply}) = 100 \)
   - \( V(\text{Supplier}, \text{scity}) = 20 \)
   - \( V(\text{Supplier}, \text{state}) = 10 \)
   - \( V(\text{Supply}, \text{pno}) = 2,500 \)

3. \( \sigma_{\text{scity}='Seattle' \text{ and sstate}='WA'} \)
   - \( \pi_{\text{sname}} \)

4. \( B(R) \cup (R_1 \alpha) + T(R_2) \cup (R_1 \alpha) \) (Index nested loop)

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'

\((2 \times \frac{1}{2500} \times 10000) = 4 \text{ tuples}\)