Administrivia

- HW3 due **TOMORROW**, Oct. 20th @ 11:00pm
- WQ4 due **Tuesday**, Oct. 24th @ 11:59pm
RA Operators

<table>
<thead>
<tr>
<th>Standard</th>
<th>Joins</th>
<th>Extended</th>
</tr>
</thead>
<tbody>
<tr>
<td>⋃ - Union</td>
<td>⨝ - Nat. Join</td>
<td>δ - Duplicate Elim.</td>
</tr>
<tr>
<td>□ - Diff.</td>
<td>□ - L.O. Join</td>
<td>γ - Group/Agg.</td>
</tr>
<tr>
<td>σ - Select</td>
<td>□ - R.O. Join</td>
<td>τ - Sorting</td>
</tr>
<tr>
<td>π - Project</td>
<td>□ - F.O. Join</td>
<td></td>
</tr>
<tr>
<td>ρ - Rename</td>
<td>ℷ - Cross Product</td>
<td></td>
</tr>
</tbody>
</table>

\[ R_1 \cap R_2 = R_1 - (R_1 - R_2) \]
\[ R_1 \cap R_2 = R_1 \times R_2 \]


\[ \text{\textbullet{\textdegree}} \text{ Notation} \]

Grouping and aggregation on group:

\[ \text{\textbullet{\textdegree}} \text{ attr}_1, \ldots, \text{attr}_k, \text{count/sum/max/min}(\text{attr}) \rightarrow \text{alias} \]

Aggregation on the entire table:

\[ \text{\textbullet{\textdegree}} \text{count/sum/max/min}(\text{attr}) \rightarrow \text{alias} \]
Query Plans

Select-Join-Project structure

Make this SQL query into RA (remember FWGHOS):

```sql
SELECT R.b, T.c, max(T.a) AS T_max
  FROM Table_R R, Table_T T
  WHERE R.b = T.b
  GROUP BY R.b, T.c
  HAVING max(T.a) > 99
```
Query Plans

Select-Join-Project structure

Make this SQL query into RA (remember FWGHOS):

SELECT R.b, T.c, max(T.a) AS T_max
FROM Table_R R, Table_T T
WHERE R.b = T.b
GROUP BY R.b, T.c
HAVING max(T.a) > 99

\[ \pi_{R.b, T.c, T_{max}}(\sigma_{T_{max}>99}(\gamma_{R.b, T.c, max(T.a)}->T_{max})(R \bowtie_{R.b=T.b} T))) \]
Datalog Terminology

Head - Body - Atom/Subgoal/Relational predicate

Base Relations (EDB) vs Derived Relations (IDB)
- Negation + Aggregate

Wildcard

`Helper(a,b):-Base1(a,b,_)`
`NonAns(j):-Base2(j,k),!Base3(k)`
`Ans(x):-Helper(x,y),!NonAns(y)`
Query Safety

Need a positive relational atom of every variable

What’s wrong with this query?

Find all of Alice’s children without children:

\[ U(x) :\neg \text{ParentChild(“Alice”,}x) , \neg \text{ParentChild}(x,y) \]
Query Safety

\[ U(x) \rightleftharpoons \text{ParentChild}(“Alice”,x), \! \text{ParentChild}(x,y) \]

It is domain dependent! Unsafe!

Double negation to the rescue. Why does this work?

\[ \text{NonAns}(x) \rightleftharpoons \text{ParentChild}(“Alice”,x), \text{ParentChild}(x,y) \]

# All of Alice’s children with children

\[ U(x) \rightleftharpoons \text{ParentChild}(“Alice”,x), \! \text{NonAns}(x) \]

# All of Alice’s children without children (safe!)

But we can do better...
Query Safety

But we can do better...

\texttt{hasChild}(x) :- \texttt{ParentChild}(x,\_)
# People with children
\texttt{U}(x) :- \texttt{ParentChild}(“Alice”,x), \texttt{!hasChild}(x)
# All of Alice’s children without children (safe!)
Datalog with Recursion

Able to write complicated queries in a few lines

Graph analysis

Done with query once output does not change.

VERY similar idea to context-free grammars (CSE 311)
Stratified Datalog

Recursion might not work well with negation

E.g.
\[ A(x) :\leftarrow \text{Table}(x), \neg B(x) \]
\[ B(x) :\leftarrow \text{Table}(x), \neg A(x) \]

Solution: Don’t negate or aggregate on an IDB predicate until it is defined

Stratified Datalog Query
Stratified Datalog

Only IDB predicates defined in strata 1, 2, ..., n may appear under ! or agg in stratum n+1
Expressive Capability

Nothing can do everything.

Forms of RA and Datalog can express things the other cannot.

<table>
<thead>
<tr>
<th></th>
<th>Positive Relations</th>
<th>Negation</th>
<th>Aggregates</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recursive</strong></td>
<td>Pure Datalog</td>
<td>Stratified Datalog</td>
<td>Stratified Datalog + agg.</td>
</tr>
<tr>
<td><strong>Non-recursive</strong></td>
<td>Non-recursive Datalog</td>
<td>RA</td>
<td>Extended RA</td>
</tr>
<tr>
<td></td>
<td>Positive RA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>