CSE 414: Section 4
Relational Algebra
Datalog

October 18th, 2018
## 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>✖ - Cross Product</td>
<td></td>
</tr>
</tbody>
</table>

**Intersect**

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

\[ R_1 \cap R_2 = R_1 \times R_2 \]
A Few More SQL Keywords

(<sub>) INTERSECT (<sub>)

(<sub>) UNION (<sub>)

(<sub>) EXCEPT (<sub>)

3
γ Notation

Grouping and aggregation on group:

γ_{attr_1, \ldots, attr_k, count/sum/max/min(attr)} \rightarrow alias

Aggregation on the entire table:

γ_{count/sum/max/min(attr)} \rightarrow alias
Query Plans (Example SQL -> RA)

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 AS R, Table_T AS T
WHERE R.b = T.b
GROUP BY R.b, T.c
HAVING max(T.a) > 99
```
Query Plans (Example SQL -> RA)

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 AS R, Table_T AS T
WHERE R.b = T.b
GROUP BY R.b, T.c
HAVING max(T.a) > 99
```

\[
\pi_{R.b, T.c, T_{\text{max}}} (\sigma_{T_{\text{max}}>99} (\gamma_{R.b, T.c, \text{max}(T.a)} \rightarrow T_{\text{max}} (R \Join R.b=T.b T)))
\]
Datalog
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:

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

U(x) :- ParentChild("Alice",x), !ParentChild(x,y)
It is domain dependent! Unsafe!

Double negation to the rescue. Why does this work?
NonAns(x) :- ParentChild("Alice",x), ParentChild(x,y)
# All of Alice’s children with children
U(x) :- ParentChild("Alice",x), !NonAns(x)
# All of Alice’s children without children (safe!)

But we can do better...
Query Safety

But we can do better...

\[\text{hasChild}(x) \; :- \; \text{ParentChild}(x,\_)
\]

# People with children

\[\text{U}(x) \; :- \; \text{ParentChild(“Alice”,x)}, \; !\text{hasChild}(x)
\]

# 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.
Stratified Datalog

Recursion might not work well with negation

E.g.

\[ A(x) :\neg \text{Table}(x), \neg B(x) \]
\[ B(x) :\neg \text{Table}(x), \neg A(x) \]

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

```
D(x,y) <- ParentChild(x,y).
D(x,z) <- D(x,y), ParentChild(y,z).
N[x] = m  <- agg<<m = count()>> D(x,y).
Q(d) <- N["Alice"]=d.
```

```
D(x,y) <- ParentChild(x,y).
D(x,z) <- D(x,y), ParentChild(y,z).
Q(x) <- D("Alice",x), !D("Bob",x).
A() <- !B().
B() <- !A().
```

May use D in an agg because was defined in previous stratum

May use !D

Cannot use !A