Relational Algebra vs. Datalog

- **Relation Algebra**
  
  $\text{SELECT}_{A>2 \text{ OR } B<2} (R)$

- **Datalog**

  $\text{Ans}(a, b) \leftarrow \text{R}(a, b), a > 2$
  
  $\text{Ans}(a, b) \leftarrow \text{R}(a, b), b < 2$

Datlog Program and Rules

- A datlog program (query) consists of one or more rules

  1st rule
  
  $\text{Ans}(a, b) \leftarrow \text{R}(a, b), a > 2$

  $\text{Ans}(a, b) \leftarrow \text{R}(a, b), b < 2$

  2nd rule

Rules

- A rules consists of

  - Head (consequent)
  - $\leftarrow$
  - Body (antecedent)

  $\text{Ans}(a, b) \leftarrow \text{R}(a, b), a > 2$

Atoms

- A rule head consists of a single atom
- A rule body is the AND of one or more atoms
- An atom evaluates to either true of false

\[
\begin{array}{c|c}
\text{R} & \text{A} & \text{B} \\
\hline
1 & 1 \\
2 & 1 \\
3 & 2 \\
3 & 3 \\
1 & 4 \\
\end{array}
\]
More about Atoms

- An atom is also called a subgoal
- There are two types of atoms
  - Relational atoms
  - Arithmetic atoms
- And one more thing, atoms usually refer to relational atoms

```
\text{Ans}(a,b) \leftarrow \text{R}(a,b), a > 2
```

Relational atom Arithmetic atom

Predicates

- A relation atom consists of a predicate and the arguments of the predicate

```
\text{Predicate } \text{Ans} \text{ with arguments } a \text{ and } b
\text{Ans}(a,b) \leftarrow \text{R}(a,b), a > 2
```

Predicate \(R\) with arguments \(a\) and \(b\)

More about Predicates

- Two types of predicates
  - Extensional predicates (EDB) – stored relations
  - Intensional predicates (IDB) – computed relations
- No EDB in rule heads

```
\text{IDB} \quad \text{Ans}(a,b) \leftarrow \text{R}(a,b), a > 2
```

Arguments and Variables

- Arguments can be either variables or constants
- If a variable appears in the head, it’s called a distinguished variable; otherwise it’s a non-distinguished variable

```
\text{Ans}(a,b) \leftarrow \text{R}(a,b), a > 2
```

\(a,b\) are both distinguished variables

Examples of Unsafe Rules

- \(\text{S1}(x) \leftarrow \text{R}(y,z)\)
- \(\text{S2}(x) \leftarrow \text{R}(y,z), x < 10\)
- \(\text{S3}(x) \leftarrow \text{NOT } \text{R}(x,y)\)

```
<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
```

Safe Rules

- A rules is safe if
  - each distinguished variable
  - each variable in a arithmetic subgoal
  - each variable in a negated subgoal also appears in a non-negated, relational subgoal
From Relation Algebra to Datalog

- Relation algebra
  - Intersection
  - Union
  - Difference
  - Project
  - Selection
  - Product/Join

- Datalog
  - ??

Need for Recursion ...

- Need for Recursion
  - Compute a relation Paths(x, y) – (x, y) is a tuple in Paths if there exists a path from x to y
  - Self-join is not enough
  - In fact, it's cannot be done in relational algebra

Recursive Datalog Solution

- Paths(x, y) ← Edges(x, y)
- Paths(x, y) ← Paths(x, z), Edges(z, y)

Evaluation of Recursive Rules – Round 1

<table>
<thead>
<tr>
<th>Edges</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>x</td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

Paths(x, y) ← Edges(x, y)
Paths(x, y) ← Paths(x, z), Edges(z, y)

Evaluation of Recursive Rules – Round 1

<table>
<thead>
<tr>
<th>Edges</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>x</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Paths(x, y) ← Edges(x, y)
Paths(x, y) ← Paths(x, z), Edges(z, y)
### Evaluation of Recursive Rules – Round 2

<table>
<thead>
<tr>
<th>Edges</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Paths(x, y) ← Edges(x, y)
Paths(x, y) ← Paths(x, z), Edges(z, y)

### Evaluation of Recursive Rules – Round 3

<table>
<thead>
<tr>
<th>Edges</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Paths(x, y) ← Edges(x, y)
Paths(x, y) ← Paths(x, z), Edges(z, y)

### Evaluation of Recursive Rules – Done

- No more tuples can be added to Paths – we have reached a fixed-point.

### Definition of Recursion

- Form a *dependency graph* whose nodes = 1DB predicates.
- Arc X → Y if and only if there is a rule with X in the head and Y in the body.
- Cycle = recursion; no cycle = no recursion.
More Complex Recursive Examples

- *Cousins in Ullman’s notes*

Recursion + Negation = Trouble

- **Example 1**
  \[ R: \{ 1 \} \]
  
  \[ P(x) \leftarrow R(x), \text{NOT } Q(x) \]
  
  \[ Q(x) \leftarrow R(x), \text{NOT } P(x) \]

- **Example 2**
  
  \[ P(x) \leftarrow R(x), \text{NOT } P(x) \]

More Use of the Dependency Graph

- Form a dependency graph whose nodes = IDB predicates.
- Arc \( X \rightarrow Y \) if and only if there is a rule with \( X \) in the head and \( Y \) in the body.
- Cycle = recursion; no cycle = no recursion.
- Label Arc \( X \rightarrow Y \) negative with a - sign

Stratified Recursion

- If a cycle in the dependency graph has no negative arc, it’s a stratified recursion.
- We restrict ourselves to only recursions that are stratified.

Un-stratified recursion:

Exercises

- 10.1.1, 10.1.2
- 10.2.1, 10.2.2, 10.2.5
- 10.3.2