A Survey of Deductive Databases

Raghu Ramakrishnan and Jeffrey D. Ullman

CS 848, Fall 2016
University of Waterloo
Presented by: Siddhartha Sahu
Overview

- Relational Databases
- Deductive Databases
- Datalog
- Example Queries
- Query Execution
- Conclusion and Discussion
Relational Databases
Relational Databases

Predominant model for data storage and processing
Relational Databases

Predominant model for data storage and processing

Declarative language: focus on what rather than how
Relational Databases
Relational Databases

```
INSERT INTO edges (...)
```

<table>
<thead>
<tr>
<th>id_from</th>
<th>id_to</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>d</td>
<td>c</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
</tr>
</tbody>
</table>
Q: List vertices that vertex ‘b’ have an outgoing edge to.
Q: List vertices that vertex ‘b’ have an outgoing edge to.

A: SELECT id_to from edges WHERE id_from = ‘b’
Q: List all vertex pairs \((x,y)\), such that \(y\) is reachable from \(x\).

A: ?
Deductive Databases
Deductive Databases

Support a superset of relational algebra.
- Supports all queries from relational algebra.
- Supports recursions.
Deductive Databases

Support a superset of relational algebra.
● Supports all queries from relational algebra.
● Supports recursions.

Datalog: subset of Prolog, a logic programming language
● Database centric requirements
● Emphasis on completeness and termination
● Queries on data stored on secondary storage
Deductive Databases

Support a superset of relational algebra.
- Supports all queries from relational algebra.
- Supports recursions.

Datalog: subset of Prolog, a logic programming language
- Database centric requirements
- Emphasis on completeness and termination
- Queries on data stored on secondary storage

A database of facts.
A set of rules for deriving new facts from existing facts.
Datalog: Terminology
Datalog: Terminology

edge(a,b).
Datalog: Terminology

Facts

edge(a,b).

Rules

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).
Datalog: Terminology

**Facts**

- edge(a,b).

**Rules**

- connected(X,Y) :- edge(X,Y).
- connected(X,Y) :- edge(X,Z), connected(Z,Y).

**Implication/Clause:** $A_0 : A_1, A_2, \ldots, A_k$ where $A_0$ is true if $A_1$ and $A_2 \ldots$ and $A_k$ are true.

- $k = 0$: fact; $k > 0$: rule
Datalog: Terminology

Datalog

(edge(a,b).

Facts

constant symbol

connected(X,Y) :- edge(X,Y).

Rules

logical variable

connected(X,Y) :- edge(X,Z), connected(Z,Y).

Implication/Clause: $A_0 : A_1, A_2, ..., A_k$ where $A_0$ is true if $A_1$ and $A_2$ ... and $A_k$ are true.

$k = 0$: fact; $k > 0$: rule
Datalog: Terminology

**EDB**
- **Facts**
  - `edge(a,b).`

**IDB**
- **Rules**
  - `connected(X,Y) :- edge(X,Y).`
  - `connected(X,Y) :- edge(X,Z), connected(Z,Y).`

**Implication/Clause**: $A_0 :- A_1, A_2, ..., A_k$ where $A_0$ is **true** if $A_1$ and $A_2 ...$ and $A_k$ are **true**.

- **$k = 0$**: fact; **$k > 0$**: rule
Datalog: Examples

<table>
<thead>
<tr>
<th>users</th>
<th>accounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>uid</td>
</tr>
<tr>
<td>name</td>
<td>account_type</td>
</tr>
<tr>
<td>age</td>
<td>amount</td>
</tr>
</tbody>
</table>
### Datalog: Examples

<table>
<thead>
<tr>
<th>users</th>
<th>accounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>uid</td>
</tr>
<tr>
<td>name</td>
<td>account_type</td>
</tr>
<tr>
<td>age</td>
<td>amount</td>
</tr>
</tbody>
</table>

users(42, ‘Jane Doe’, 26).
accounts(42, ‘savings’, 5692.23)
Datalog: Examples

Selection

Q: List all users with age > 23.

users(42, 'Jane Doe', 26).
accounts(42, 'savings', 5692.23)
Datalog: Examples

Selection

Q: List all users with age > 23.

Relational Algebra: $\sigma_{\text{age} > 23}(\text{users})$

SQL: SELECT * FROM users WHERE age > 23;

<table>
<thead>
<tr>
<th>users</th>
<th>accounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>uid</td>
</tr>
<tr>
<td>name</td>
<td>account_type</td>
</tr>
<tr>
<td>age</td>
<td>amount</td>
</tr>
</tbody>
</table>

users(42, 'Jane Doe', 26).
accounts(42, 'savings', 5692.23)
Datalog: Examples

Selection

Q: List all users with \( \text{age} > 23 \).

Relational Algebra: \( \sigma_{\text{age} > 23}(\text{users}) \)

SQL: SELECT * FROM users WHERE \( \text{age} > 23 \);

Datalog: \( S(\text{Uid}, \text{Name}, \text{Age}) \) :- users(\text{Uid}, \text{Name}, \text{Age}), \text{Age} > 23. \)
### Datalog: Examples

#### Projection

Q: List **name** of users with age > 23.

<table>
<thead>
<tr>
<th>users</th>
<th>accounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>uid</td>
</tr>
<tr>
<td>name</td>
<td>account_type</td>
</tr>
<tr>
<td>age</td>
<td>amount</td>
</tr>
</tbody>
</table>

users(42, ‘Jane Doe’, 26).
accounts(42, ‘savings’, 5692.23)
Datalog: Examples

Projection

Q: List name of users with age > 23.

Relational Algebra: $\pi_{\text{name}}(\sigma_{\text{age} > 23}(\text{users}))$

SQL: SELECT name FROM users WHERE age > 23;

<table>
<thead>
<tr>
<th>users</th>
<th>accounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>uid</td>
</tr>
<tr>
<td>name</td>
<td>account_type</td>
</tr>
<tr>
<td>age</td>
<td>amount</td>
</tr>
</tbody>
</table>

users(42, ‘Jane Doe’, 26).

accounts(42, ‘savings’, 5692.23)
Datalog: Examples

Projection

Q: List name of users with age > 23.

Relational Algebra:
\[ \pi_{\text{name}}(\sigma_{\text{age} > 23}(\text{users})) \]

SQL:
SELECT name FROM users WHERE age > 23;

Datalog:
\[ \text{P}(\text{Name}) :- \text{users}(\text{Uid, Name, Age}), \text{Age} > 23. \]
Datalog: Examples

Join

Q: List name, amount of users with age > 23.

users(42, 'Jane Doe', 26).
accounts(42, 'savings', 5692.23)
Datalog: Examples

Join

Q: List name, amount of users with age > 23.

Relational Algebra:
\[ \pi_{\text{name}, \text{amount}}(\sigma_{\text{age} > 23}(\text{users} \bowtie_{\text{uid}} \text{accounts})) \]

SQL:
SELECT name, amount FROM users, accounts
WHERE users.uid = accounts.uid AND age > 23;

users(42, ‘Jane Doe’, 26).
accounts(42, ‘savings’, 5692.23)
Datalog: Examples

Join

Q: List name, amount of users with age > 23.

Relational Algebra:
\[ \pi_{name, amount}(\sigma_{\text{age} > 23}(\text{users} \Join_{\text{uid}} \text{accounts})) \]

SQL:
SELECT name, amount FROM users, accounts
WHERE users.uid = accounts.uid AND age > 23;

Datalog:
\[ J(\text{Name}, \text{Amount}) :\neg \text{users}(\text{Uid}, \text{Name}, \text{Age}), \text{accounts}(\text{Uid}, \text{Account_type}, \text{Amount}), \text{Age} > 23. \]
Datalog: Examples
Datalog: Examples

Datalog

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

edge(a,b).
edge(b,d).
edge(b,e).
edge(d,c).
edge(f,e).
Datalog: Examples

edge(a,b).
edge(b,d).
edge(b,e).
edge(d,c).
edge(f,e).

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

Q: List vertices that vertex ‘b’ have an outgoing edge to.
Q: List vertices that vertex ‘b’ have an outgoing edge to.
A: query(X) :- edge(b,X).
Datalog: Examples

Q: List all vertex pairs \((x,y)\), such that \(y\) is reachable from \(x\).
Datalog: Examples

Q: List all vertex pairs \((x,y)\), such that \(y\) is reachable from \(x\).

A: query\((X,Y)\) :- connected\((X,Y)\).
Query Evaluation: Naïve algorithm
Query Evaluation: Naïve algorithm

\[
P_0 = \text{InitialValue} \\
\text{Repeat} \\
P_k = f(P_{k-1}) \\
\text{Until } \text{no-more-change}
\]
Query Evaluation: Naïve algorithm

1. Begin by assuming all IDB relations are empty.

\[
P_0 = \text{Initial Value} \\
\text{Repeat} \\
P_k = f(P_{k-1}) \\
\text{Until no-more-change}
\]
Query Evaluation: Naïve algorithm

1. Begin by assuming all IDB relations are empty.

2. Repeatedly evaluate the rules using the EDB and the previous IDB to get a new IDB.

\[
P_0 = \text{Initial Value} \\
\text{Repeat} \\
P_k = f(P_{k-1}) \\
\text{Until no-more-change}
\]
Query Evaluation: Naïve algorithm

1. Begin by assuming all IDB relations are empty.

2. Repeatedly evaluate the rules using the EDB and the previous IDB to get a new IDB.

3. End when there is no change to the IDB.
Query Evaluation: Naïve algorithm

\[
\text{connected}(X,Y) :- \text{edge}(X,Y). \\
\text{connected}(X,Y) :- \text{edge}(X,Z), \text{connected}(Z,Y).
\]

connected(X,Y).
connected(X,Y).

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

edges

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
</tr>
</tbody>
</table>
connected(X,Y).

∅

l = 0

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

<table>
<thead>
<tr>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>d</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>f</td>
</tr>
</tbody>
</table>
Query Evaluation: Naïve algorithm

connected(X,Y).

edges
a b
d c
e f

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

∅

I = 0

I = 1

Query Evaluation: Naïve algorithm

connected(X,Y).
connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

I = 0
I = 1
∅

edges
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
</tr>
</tbody>
</table>

Query Evaluation: Naïve algorithm

connected(X,Y).

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

<table>
<thead>
<tr>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>d</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>f</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>l = 0</td>
</tr>
<tr>
<td>l = 1</td>
</tr>
<tr>
<td>l = 2</td>
</tr>
<tr>
<td>I = 0</td>
</tr>
<tr>
<td>I = 1</td>
</tr>
<tr>
<td>I = 2</td>
</tr>
</tbody>
</table>

query Evaluation: Naïve algorithm

connected(X,Y).

\[
\begin{array}{ll}
\text{edges} & a\ b \\
       & b\ d \\
       & d\ c \\
       & b\ e \\
       & f\ e \\
\end{array}
\]

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

\[
\begin{array}{ll}
\text{I = 0} & a\ b \\
       & b\ d \\
       & d\ c \\
       & b\ e \\
       & f\ e \\
\end{array}
\]

\[
\begin{array}{ll}
\text{I = 1} & a\ b \\
       & b\ d \\
       & d\ c \\
       & b\ e \\
       & f\ e \\
\end{array}
\]

\[
\begin{array}{ll}
\text{I = 2} & a\ b \\
       & b\ d \\
       & d\ c \\
       & b\ e \\
       & f\ e \\
\end{array}
\]
Query Evaluation: Naïve algorithm

\[
\text{connected}(X, Y) :\text{-} \text{edge}(X, Y).
\]

\[
\text{connected}(X, Y) :\text{-} \text{edge}(X, Z), \text{connected}(Z, Y).
\]
Query Evaluation: Naïve algorithm

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

I = 0
∅

I = 1

I = 2

Query Evaluation: Naïve algorithm

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

Connected(X,Y).

edges

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

A graph with nodes a, b, c, d, e, and f, with edges between a-b, a-d, b-d, b-e, and f-e.

<table>
<thead>
<tr>
<th>I = 0</th>
<th>I = 1</th>
<th>I = 2</th>
<th>I = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>a b</td>
<td>b d</td>
<td>b d</td>
</tr>
<tr>
<td></td>
<td>b d</td>
<td>d c</td>
<td>d c</td>
</tr>
<tr>
<td></td>
<td>b e</td>
<td>b e</td>
<td>f e</td>
</tr>
</tbody>
</table>
Query Evaluation: Naïve algorithm

connected(X,Y).

equations

<table>
<thead>
<tr>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>d</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>f</td>
</tr>
</tbody>
</table>

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

I = 0
∅
I = 1
a b
b d
d c
b e
f e
I = 2
b c
a d
a e
b e
f e
I = 3
b c
a d
a e
b e
f e

Query Evaluation: Naïve algorithm

connected(X, Y).

∅

I = 0

I = 1

I = 2

I = 3

connected(X, Y) :- edge(X, Y).
connected(X, Y) :- edge(X, Z), connected(Z, Y).

Query Evaluation: Naïve algorithm

connected(X,Y).

\[
\begin{array}{c c}
\text{edges} \\
a & b \\
b & d \\
d & c \\
b & e \\
f & e \\
\end{array}
\]

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

I = 0
∅
I = 1
\[
\begin{array}{c c}
a & b \\
b & d \\
d & c \\
b & e \\
f & e \\
\end{array}
\]
I = 2
\[
\begin{array}{c c}
a & b \\
b & d \\
d & c \\
b & e \\
f & e \\
\end{array}
\]
I = 3
\[
\begin{array}{c c}
a & c \\
a & d \\
a & e \\
\end{array}
\]

Query Evaluation: Naïve algorithm

connected(X,Y).

<table>
<thead>
<tr>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>d</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>f</td>
</tr>
</tbody>
</table>

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

I = 0
∅

I = 1

I = 2

I = 3

I = 4
Query Evaluation: Naïve algorithm

\[ \text{connected}(X,Y). \]

\[
\begin{array}{|c|c|}
\hline
\text{edges} & \text{connected}(X,Y) :\text{edge}(X,Y). \\
\hline
\text{a} & \text{b} \\
\text{b} & \text{d} \\
\text{d} & \text{c} \\
\text{b} & \text{e} \\
\text{f} & \text{e} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{I} & 0 \\
\hline
\text{∅} & \text{I} = 0 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{I} & 1 \\
\hline
\text{a} & \text{c} \\
\text{b} & \text{c} \\
\text{a} & \text{d} \\
\text{a} & \text{e} \\
\text{a} & \text{b} \\
\text{b} & \text{d} \\
\text{d} & \text{c} \\
\text{b} & \text{e} \\
\text{f} & \text{e} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{I} & 2 \\
\hline
\text{a} & \text{c} \\
\text{b} & \text{c} \\
\text{a} & \text{d} \\
\text{a} & \text{e} \\
\text{a} & \text{b} \\
\text{b} & \text{d} \\
\text{d} & \text{c} \\
\text{b} & \text{e} \\
\text{f} & \text{e} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{I} & 3 \\
\hline
\text{a} & \text{c} \\
\text{b} & \text{c} \\
\text{a} & \text{d} \\
\text{a} & \text{e} \\
\text{a} & \text{b} \\
\text{b} & \text{d} \\
\text{d} & \text{c} \\
\text{b} & \text{e} \\
\text{f} & \text{e} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{I} & 4 \\
\hline
\text{a} & \text{c} \\
\text{b} & \text{c} \\
\text{a} & \text{d} \\
\text{a} & \text{e} \\
\text{a} & \text{b} \\
\text{b} & \text{d} \\
\text{d} & \text{c} \\
\text{b} & \text{e} \\
\text{f} & \text{e} \\
\hline
\end{array}
\]
Query Evaluation: Naïve algorithm

connected(X, Y) :- edge(X, Y).
connected(X, Y) :- edge(X, Z), connected(Z, Y).

Query Evaluation: Naïve algorithm

connected(X, Y) :- edge(X, Y).
connected(X, Y) :- edge(X, Z), connected(Z, Y).

edges
a b
d c
b e
f e

connected(X, Y).

∅
l = 0

l = 1

l = 2

l = 3

l = 4

Query Evaluation: Semi-Naïve algorithm

* Avoid repeating computations already done in previous iterations.

* Focus on only the newly derived tuples (deltas) from previous iterations.
Query Evaluation: Semi-Naïve algorithm

* Avoid repeating computations already done in previous iterations.

* Focus on only the newly derived tuples (deltas) from previous iterations.

for each IDB predicate $p$
  
  do  
  \begin{align*}
  p^{[0]} & := \emptyset \\
  \delta(p)^{[0]} & := \text{tuples produced by rules using only EDB's}
  \end{align*}

  \begin{array}{c}
  i := 1 \\
  \text{repeat}
  \begin{array}{c}
  p^{[i]} := p^{[i-1]} \cup \delta(p)^{[i-1]} \\
  \text{evaluate } \Delta(p)^{[i]} \\
  \delta(p)^{[i]} := \Delta(p)^{[i]} - p^{[i]} \\
  i := i + 1
  \end{array}
  \end{array}

  \text{until } \delta(p)^{[i]} = \emptyset \text{ for each IDB predicate } p
Query Evaluation: Semi-Naïve algorithm

* Avoid repeating computations already done in previous iterations.

* Focus on only the newly derived tuples (deltas) from previous iterations.

```
for each IDB predicate p
    do
        \[ p^[0] := \emptyset \]
        \[ \delta(p)^[0] := \text{tuples produced by rules using only EDB's} \]
        \[ i := 1 \]
    repeat
        \[ p^[i] := p^[i-1] \cup \delta(p)^[i-1] \]
        evaluate \( \Delta(p)^[i] \)
        \[ \delta(p)^[i] := \Delta(p)^[i] - p^[i] \]
        \[ i := i + 1 \]
    until \( \delta(p)^[i] = \emptyset \) for each IDB predicate p
```

\[
\Delta(p)^[i] := \delta(p_1)^[i-1], p_2^[i-1], \ldots, p_n^[i-1], q_1, \ldots, q_m.
\]
\[
\Delta(p)^[i] := p_1^[i], \delta(p_2)^[i-1], p_3^[i-1], \ldots, p_n^[i-1], q_1, \ldots, q_m.
\]
\[
\ldots
\]
\[
\Delta(p)^[i] := p_1^[i], \ldots, p_{n-1}^[i], \delta(p_n)^[i-1], q_1, \ldots, q_m.
\]
Query Evaluation: Semi-Naïve algorithm

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

<table>
<thead>
<tr>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>d</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>f</td>
</tr>
</tbody>
</table>

∅

\( P_1 = 0 \)
Query Evaluation: Semi-Naïve algorithm

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

<table>
<thead>
<tr>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>d</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>f</td>
</tr>
</tbody>
</table>

\[ P_{1=0} \]
\[ \emptyset \]

\[ \delta_{1=0} \]
Query Evaluation: Semi-Naïve algorithm

edges

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

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

∅

$P_{l=0} \quad \delta_{l=0} \quad P_{l=1}$
Query Evaluation: Semi-Naïve algorithm

\[
\text{connected}(X,Y) :\text{ edge}(X,Y). \\
\text{connected}(X,Y) :\text{ edge}(X,Z), \text{connected}(Z,Y).
\]

\[
\begin{array}{c|c}
\text{edges} & \text{status} \\
\hline
a & b \\
b & d \\
d & c \\
b & e \\
f & e \\
\end{array}
\]

\[
\begin{array}{c|c|c}
P_{I=0} & P_{I=1} \\
\hline
\emptyset & a b \\
b d & b d \\
d c & d c \\
b e & b e \\
f e & f e \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\delta_{I=0} & \Delta_{I=1} \\
\hline
a d & a d \\
a e & a e \\
b c & b c \\
\end{array}
\]
Query Evaluation: Semi-Naïve algorithm

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

edges
a   b
d   c
b   e
f   e

P_{l=0}  δ_{l=0}  P_{l=1}  Δ_{l=1}
∅       a   b  a   d
        b   d  a   e
        d   c  b   c
        f   e  b   c

δ_{l=1}
Query Evaluation: Semi-Naïve algorithm

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

<table>
<thead>
<tr>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b</td>
</tr>
<tr>
<td>b d</td>
</tr>
<tr>
<td>d c</td>
</tr>
<tr>
<td>b e</td>
</tr>
<tr>
<td>f e</td>
</tr>
</tbody>
</table>

$P_{l=0}$  
$\emptyset$  
$\delta_{l=0}$  

$P_{l=1}$  
$a b$  
$b d$  
$d c$  
$b e$  
$f e$  

$\Delta_{l=1}$  
$a d$  
$a e$  
$b c$  

$P_{l=2}$  
$a d$  
$a e$  
$b c$  
$b d$  
$d c$  
$b e$  
$f e$  

$P_{l=1}$  
$a b$  
$b d$  
$d c$  
$b e$  
$f e$  

$\delta_{l=1}$  
$a d$  
$a e$  
$b c$  

$P_{l=2}$  
$a d$  
$a e$  
$b c$  
$\emptyset$  

connected(X,Y) :- edge(X,Y). connected(X,Y) :- edge(X,Z), connected(Z,Y).
Query Evaluation: Semi-Naïve algorithm

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

<table>
<thead>
<tr>
<th>edges</th>
<th>a</th>
<th>b</th>
<th>d</th>
<th>c</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>d</td>
<td>c</td>
<td>e</td>
</tr>
<tr>
<td>δ₁ = 0</td>
<td>Δ₁ = 1</td>
<td>δ₂ = 1</td>
<td>P₁ = 2</td>
<td>δ₃ = 1</td>
<td>δ₄ = 1</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>d</td>
<td>c</td>
<td>e</td>
</tr>
<tr>
<td>P₀ = 0</td>
<td>a</td>
<td>b</td>
<td>d</td>
<td>c</td>
<td>e</td>
</tr>
</tbody>
</table>

∅
Query Evaluation: Semi-Naïve algorithm

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

```
<table>
<thead>
<tr>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b</td>
</tr>
<tr>
<td>b d</td>
</tr>
<tr>
<td>d c</td>
</tr>
<tr>
<td>b e</td>
</tr>
<tr>
<td>f e</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>P_{l=0}</th>
<th>δ_{l=0}</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>a b</td>
<td></td>
</tr>
<tr>
<td>b d</td>
<td></td>
</tr>
<tr>
<td>d c</td>
<td></td>
</tr>
<tr>
<td>b e</td>
<td></td>
</tr>
<tr>
<td>f e</td>
<td></td>
</tr>
</tbody>
</table>

P_{l=1} | δ_{l=1}
--------|--------
a b     | a d    
b d     | a e    
d c     | b c    

P_{l=2} | δ_{l=2}
--------|--------
a b     | a c    
b d     |       
d c     |       
```
Query Evaluation: Semi-Naïve algorithm

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

<table>
<thead>
<tr>
<th>edges</th>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b</td>
<td>a c</td>
</tr>
<tr>
<td>b d</td>
<td>a d</td>
</tr>
<tr>
<td>d c</td>
<td>a e</td>
</tr>
<tr>
<td>b e</td>
<td>b c</td>
</tr>
<tr>
<td>f e</td>
<td>f e</td>
</tr>
</tbody>
</table>

\[ \Delta_{I=0} \]
\[ \Delta_{I=1} \]
\[ \Delta_{I=2} \]
Query Evaluation: Semi-Naïve algorithm

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(X,Z), connected(Z,Y).

edges

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|c|c|c|c}
& \text{P}_1 = 0 & \text{δ}_1 = 0 & \text{δ}_1 = 1 & \text{P}_1 = 1 & \text{δ}_1 = 2 & \text{δ}_1 = 3 \\
\hline
\text{∅} & \text{∅} & \text{∅} & \text{∅} & \text{∅} & \text{∅} & \text{∅} \\
\end{array}
\]
Deductive Databases: Additional concepts
Deductive Databases: Additional concepts

Negation predicates
Deductive Databases: Additional concepts

Negation predicates

Safe rules
Deductive Databases: Additional concepts

Negation predicates

Safe rules

Query optimization
  Magic Sets
  Rule-Rewriting Techniques
  Iterative Fixpoint Evaluation
Deductive Databases: Additional concepts

Negation predicates

Safe rules

Query optimization
  Magic Sets
  Rule-Rewriting Techniques
  Iterative Fixpoint Evaluation

Aggregations
Conclusion
Conclusion

Deductive databases are more expressive than relational databases.

Support for recursive queries
Conclusion

Deductive databases are more expressive than relational databases.
Support for recursive queries

Datalog: query language adapted from Prolog
Conclusion

Deductive databases are more expressive than relational databases.
- Support for recursive queries

**Datalog**: query language adapted from Prolog

Focus on query optimization
Conclusion

Deductive databases are more expressive than relational databases.
  Support for recursive queries

**Datalog**: query language adapted from Prolog

Focus on query optimization

Naive vs Semi Naive query execution algorithms
  Avoid repeated computations
Discussion

SQL has recursion techniques like CTE
How does that compare to Datalog in terms of expressiveness?

Application domains best suited for Datalog?
Program analysis (recursion)
Declarative networking (NDlog)
Security (SeNDlog)

Applicability to general processing frameworks?
Hive
Spark SQL