Introduction to Database Systems
CSE 414

Lecture 8: Relational Algebra
Announcements

• HW3 is out – due Friday
  – git pull upstream master
  – Make sure you have email from Microsoft Azure and log in

• Web quiz 2 due tonight
Relational Algebra
Relational Algebra

• Set-at-a-time algebra, which manipulates relations
• In SQL we say *what* we want
• In RA we can express *how* to get it
• Every DBMS implementation converts a SQL query to RA in order to execute it
• An RA expression is called a *query plan*
Why study another relational query language?

- RA is how SQL is implemented in DBMS
  - We will see more of this in a few weeks

- RA opens up opportunities for query optimization
Basics

• Relations and attributes
• Functions that are applied to relations
  – Return relations
    \[ R2 = \sigma (R1) \]
  – Can be composed together
    \[ R3 = \pi (\sigma (R1)) \]
  – Often displayed using a tree rather than linearly
  – Use Greek symbols: \( \sigma \), \( \pi \), \( \delta \), etc
Sets v.s. Bags

- Sets: \{a,b,c\}, \{a,d,e,f\}, \{\}\ldots
- Bags: \{a, a, b, c\}, \{b, b, b, b, b\}, \ldots

Relational Algebra has two flavors:
- Set semantics = standard Relational Algebra
- Bag semantics = extended Relational Algebra

DB systems implement bag semantics (Why?)
Relational Algebra Operators

- Union \( \cup \), intersection \( \cap \), difference -
- Selection \( \sigma \)
- Projection \( \pi \)
- Cartesian product \( \times \), join \( \Join \)
- (Rename \( \rho \))
- Duplicate elimination \( \delta \)
- Grouping and aggregation \( \gamma \)
- Sorting \( \tau \)

All operators take in 1 or more relations as inputs and return another relation
Union and Difference

\[ R1 \cup R2 \]
\[ R1 - R2 \]

Only make sense if R1, R2 have the same schema

What do they mean over bags?
What about Intersection?

- Derived operator using minus

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

- Derived using join

\[
R_1 \cap R_2 = R_1 \bowtie R_2
\]
Selection

- Returns all tuples which satisfy a condition

\[ \sigma_c(R) \]

- Examples
  - \( \sigma_{\text{Salary} > 40000} \) (Employee)
  - \( \sigma_{\text{name} = \text{“Smith”}} \) (Employee)

- The condition \( c \) can be =, <, <=, >, >=, <> combined with AND, OR, NOT
### Employee

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>50000</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{Salary} > 40000} (\text{Employee}) \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>50000</td>
</tr>
</tbody>
</table>
Projection

- Eliminates columns

\[ \pi_{A_1, \ldots, A_n}(R) \]

- Example: project social-security number and names:
  \[ \pi_{\text{SSN}, \text{Name}}(\text{Employee}) \rightarrow \text{Answer(\text{SSN, Name})} \]

Different semantics over sets or bags! Why?
\[ \pi_{Name,\text{Salary}} (\text{Employee}) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>John</td>
<td>60000</td>
</tr>
<tr>
<td>John</td>
<td>20000</td>
</tr>
</tbody>
</table>

Bag semantics

Set semantics

Which is more efficient?
Composing RA Operators

<table>
<thead>
<tr>
<th>no</th>
<th>name</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p1</td>
<td>98125</td>
<td>flu</td>
</tr>
<tr>
<td>2</td>
<td>p2</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>3</td>
<td>p3</td>
<td>98120</td>
<td>lung</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{disease} = \text{'heart'}}(\text{Patient}) \]

<table>
<thead>
<tr>
<th>no</th>
<th>name</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>p2</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>

\[ \Pi_{\text{zip}, \text{disease}}(\sigma_{\text{disease} = \text{'heart'}}(\text{Patient})) \]

<table>
<thead>
<tr>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>98125</td>
<td>flu</td>
</tr>
<tr>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>98120</td>
<td>lung</td>
</tr>
<tr>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>
Cartesian Product

• Each tuple in R1 with each tuple in R2

\[ R1 \times R2 \]

• Rare in practice; mainly used to express joins
## Cross-Product Example

### Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>

### Dependent

<table>
<thead>
<tr>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

### Employee X Dependent

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>999999999</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>
Renaming

• Changes the schema, not the instance

\[ \rho_{B_1, \ldots, B_n}(R) \]

• Example:
  – Given Employee(Name, SSN)
  – \( \rho_{N, S}(\text{Employee}) \) \( \rightarrow \) Answer(N, S)
Natural Join

\[ R_1 \bowtie R_2 \]

• Meaning: \[ R_1 \bowtie R_2 = \Pi_A(\sigma_\theta (R_1 \times R_2)) \]

• Where:
  – Selection \( \sigma_\theta \) checks equality of all common attributes (i.e., attributes with same names)
  – Projection \( \Pi_A \) eliminates duplicate common attributes
**Natural Join Example**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
</tr>
</tbody>
</table>

\[ \mathbf{R} \bowtie \mathbf{S} = \Pi_{ABC}(\sigma_{R.B=S.B}(\mathbf{R} \times \mathbf{S})) \]
Natural Join Example 2

AnonPatient P

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
</tbody>
</table>

Voters V

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>Bob</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

P $\bowtie$ V

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>Alice</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>Bob</td>
</tr>
</tbody>
</table>
Natural Join

• Given schemas $R(A, B, C, D)$, $S(A, C, E)$, what is the schema of $R \bowtie S$?

• Given $R(A, B, C)$, $S(D, E)$, what is $R \bowtie S$?

• Given $R(A, B)$, $S(A, B)$, what is $R \bowtie S$?
Theta Join

- A join that involves a predicate

\[ R1 \bowtie_{\theta} R2 = \sigma_{\theta} (R1 \times R2) \]

- Here \( \theta \) can be any condition
- No projection in this case!
- For our voters/patients example:

\[ P \bowtie P.zip = V.zip \text{ and } P.age \geq V.age - 1 \text{ and } P.age \leq V.age + 1 \]

AnonPatient (age, zip, disease)
Voters (name, age, zip)
Equijoin

• A theta join where \( \theta \) is an equality predicate

\[
R1 \bowtie_\theta R2 = \sigma_\theta (R1 \times R2)
\]

• By far the most used variant of join in practice
• What is the relationship with natural join?
### Equijoin Example

**AnonPatient P**

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
</tbody>
</table>

**Voters V**

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

$$P \bowtie_{P\text{.age}=V\text{.age}} V$$

<table>
<thead>
<tr>
<th>P.age</th>
<th>P.zip</th>
<th>P.disease</th>
<th>V.name</th>
<th>V.age</th>
<th>V.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>
Join Summary

- **Theta-join**: $R \Join_\theta S = \sigma_\theta (R \times S)$
  - Join of $R$ and $S$ with a join condition $\theta$
  - Cross-product followed by selection $\theta$
  - No projection

- **Equijoin**: $R \Join_\theta S = \sigma_\theta (R \times S)$
  - Join condition $\theta$ consists only of equalities
  - No projection

- **Natural join**: $R \Join S = \pi_A (\sigma_\theta (R \times S))$
  - Equality on all fields with same name in $R$ and in $S$
  - Projection $\pi_A$ drops all redundant attributes
So Which Join Is It?

When we write $R \bowtie S$ we usually mean an equijoin, but we often omit the equality predicate when it is clear from the context.
More Joins

• **Outer join**
  – Include tuples with no matches in the output
  – Use NULL values for missing attributes
  – Does not eliminate duplicate columns

• **Variants**
  – Left outer join
  – Right outer join
  – Full outer join
## Outer Join Example

**AnonPatient P**

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
</tr>
</tbody>
</table>

**AnnonJob J**

<table>
<thead>
<tr>
<th>job</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>lawyer</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>cashier</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P.age</th>
<th>P.zip</th>
<th>P.disease</th>
<th>J.job</th>
<th>J.age</th>
<th>J.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>lawyer</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
<td>20</td>
<td>98120</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Some Examples

Supplier(sno, sname, scity, sstate)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)

Name of supplier of parts with size greater than 10
Project[sname](Supplier Join[sno=sno]
   (Supply Join[pno=pno] (Select[psize>10](Part))))

Using symbols:
π_sname(Supplier ⨝ (Supply ⨝ (σ_{psize>10}(Part))))

Can be represented as trees as well
Representing RA Queries as Trees

$\pi_{\text{sname}}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{\text{psize}>10} (\text{Part})))$

Answer

$\pi_{\text{sname}}$

CSE 414 - Autumn 2018
Some Examples

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)

Name of supplier of parts with size greater than 10
π_{sname}(Supplier ⊙ (Supply ⊙ (σ_{psize>10} (Part))))

Name of supplier of red parts or parts with size greater than 10
π_{sname}(Supplier ⊙ (Supply ⊙ (σ_{psize>10} ∨ pcolor='red' (Part))))
π_{sname}(Supplier ⊙ (Supply ⊙ (σ_{psize>10} (Part) ∪ σ_{pcolor='red'} (Part))))
Relational Algebra Operators

- Union $\cup$, intersection $\cap$, difference $-$
- Selection $\sigma$
- Projection $\pi$
- Cartesian product $\times$, join $\Join$
- (Rename $\rho$)
- Duplicate elimination $\delta$
- Grouping and aggregation $\gamma$
- Sorting $\tau$

All operators take in 1 or more relations as inputs and return another relation
Extended RA: Operators on Bags

• Duplicate elimination $\delta$
• Grouping $\gamma$
  – Takes in relation and a list of grouping operations (e.g., aggregates). Returns a new relation.
• Sorting $\tau$
  – Takes in a relation, a list of attributes to sort on, and an order. Returns a new relation.
Grouping

• Specify groups and aggregates

\[ \gamma_{A_1, \ldots, A_n, \text{sum/max}(B_1), \ldots} (R) \]

• Example: project social-security number and names:

• Output is like project: only output is attributes in the subscript

• Can also rename: \[ \gamma_{A, \text{count}(B) \rightarrow \text{count}} (R) \]
Using Extended RA Operators

\[
\begin{align*}
\text{SELECT } & \text{city, sum(quantity)} \\
\text{FROM } & \text{Sales} \\
\text{GROUP BY } & \text{city} \\
\text{HAVING } & \text{count(*) > 100}
\end{align*}
\]

Answer

\[
\begin{align*}
\Pi_{\text{city, q}} & \quad \sigma_{c > 100} \\
\gamma_{\text{city, sum(quantity)→q, count(*)→c}} & \quad \text{Sales(product, city, quantity)}
\end{align*}
\]
Typical Plan for a Query (1/2)

Answer

\[ \sigma \text{selection condition} \]

\[ \pi_{\text{fields}} \]

SELECT-PROJECT-JOIN Query

\{ SELECT fields \\
FROM R, S, ... \\
WHERE condition \}

\[ \text{JOIN conditions} \]

R

S

CSE 414 - Autumn 2018
Typical Plan for a Query (1/2)

\[ \sigma_{\text{having condition}} \]
\[ \gamma_{\text{fields, sum/count/min/max(fields)}} \]
\[ \Pi_{\text{fields}} \]
\[ \sigma_{\text{where condition}} \]
join condition

SELECT fields
FROM R, S, ...
WHERE condition
GROUP BY fields
HAVING condition

CSE 414 - Autumn 2018
How about Subqueries?

Return all suppliers in WA that sell no products greater than $100
How about Subqueries?

```sql
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
    and not exists
    (SELECT *
     FROM Supply P
     WHERE P.sno = Q.sno
         and P.price > 100)

Return all suppliers in WA that sell no products greater than $100
```
How about Subqueries?

Option 1: create nested plans

$$\text{SELECT } Q.\text{sno}$$
$$\text{FROM Supplier } Q$$
$$\text{WHERE } Q.\text{sstate} = \text{‘WA’ and not exists}$$
$$(\text{SELECT } *$$
$$\text{FROM Supply } P$$
$$\text{WHERE } P.\text{sno} = Q.\text{sno and } P.\text{price} > 100)$$
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
and not exists
(SELECT *
    FROM Supply P
    WHERE P.sno = Q.sno
    and P.price > 100)

How about Subqueries?
How about Subqueries?

SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
  and not exists
    (SELECT *
      FROM Supply P
      WHERE P.sno = Q.sno
        and P.price > 100)

De-Correlation

SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
  and Q.sno not in
    (SELECT P.sno
     FROM Supply P
     WHERE P.price > 100)

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
(SELECT Q.sno
  FROM Supplier Q
  WHERE Q.sstate = 'WA')
EXCEPT
(SELECT P.sno
  FROM Supply P
  WHERE P.price > 100)
EXCEPT = set difference
How about Subqueries?

Finally...
Summary of RA and SQL

- SQL = a declarative language where we say *what* data we want to retrieve
- RA = an algebra where we say *how* we want to retrieve the data
- **Theorem**: SQL and RA can express exactly the same class of queries

RDBMS translate SQL $\rightarrow$ RA, then optimize RA
Summary of RA and SQL

- SQL (and RA) cannot express ALL queries that we could write in, say, Java

- Example:
  - Parent(p, c): find all descendants of ‘Alice’
  - No RA query can compute this!
  - This is called a recursive query

- Next lecture: Datalog is an extension that can compute recursive queries
Class Overview

• Unit 1: Intro
• Unit 2: Relational Data Models and Query Languages
  – Data models, SQL, Relational Algebra, Datalog
• Unit 3: Non-relational data
• Unit 4: RDMBS internals and query optimization
• Unit 5: Parallel query processing
• Unit 6: DBMS usability, conceptual design
• Unit 7: Transactions
What is Datalog?

• Another query language for relational model
  – Designed in the 80’s
  – Simple, concise, elegant
  – Extends relational queries with recursion

• Today is a hot topic:
  – Souffle (we will use in HW4)
  – Eve http://witheve.com/
  – Differential datalog
    https://github.com/frankmcsherry/differential-dataflow
  – Beyond databases in many research projects: network protocols, static program analysis
• Open-source implementation of Datalog DBMS
• Under active development
• Commercial implementations are available
  – More difficult to set up and use
• “sqlite” of Datalog
  – Set-based rather than bag-based

• Install in your VM
  – Run `sudo yum install souffle` in terminal
  – More details in upcoming HW4
Why bother with *yet* another relational query language?
Example: storing FB friends

As a graph

Peter
  Mary
  John
  Phil

Or

<table>
<thead>
<tr>
<th>Person1</th>
<th>Person2</th>
<th>is_friend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>John</td>
<td>1</td>
</tr>
<tr>
<td>John</td>
<td>Mary</td>
<td>0</td>
</tr>
<tr>
<td>Mary</td>
<td>Phil</td>
<td>1</td>
</tr>
<tr>
<td>Phil</td>
<td>Peter</td>
<td>1</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

We will learn the tradeoffs of different data models later this quarter
Compute your friends graph

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>isFriend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>John</td>
<td>1</td>
</tr>
<tr>
<td>John</td>
<td>Mary</td>
<td>0</td>
</tr>
<tr>
<td>Mary</td>
<td>Phil</td>
<td>1</td>
</tr>
<tr>
<td>Phil</td>
<td>Peter</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Friends(p1, p2, isFriend)

My own friends

```
SELECT f1.p2
FROM Friends as f1,
(SELECT f.p2
FROM Friends as f
WHERE f.p1 = 'me' AND f.isFriend = 1) as f2
WHERE f1.p1 = f2.p2 AND f1.isFriend = 1
```

My FoF

```
SELECT f.p2
FROM Friends as f
WHERE f.p1 = 'me' AND f.isFriend = 1
```

My FoFoF

```
(SELECT f.p2
FROM Friends as f
WHERE f.p1 = 'me' AND f.isFriend = 1) as f2
WHERE f1.p1 = f2.p2 AND f1.isFriend = 1
```

My FoFoFoF

Datalog allows us to write recursive queries easily

When does it end???
Datalog: Facts and Rules

**Facts** = tuples in the database  
**Rules** = queries

Schema

Actor(id, fname, lname)  
Casts(pid, mid)  
Movie(id, name, year)
Datalog: Facts and Rules

**Facts** = tuples in the database

**Rules** = queries

```
.decl Actor(id: number, fname: symbol, lname: symbol)
.decl Casts(pid: number, mid: number)
.decl Movie(id: number, name: symbol, year: number)

Actor(344759, 'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).
```

Table declaration

Types in Souffle:
- number
- symbol (aka varchar)

Insert data
Datalog: Facts and Rules

Facts = tuples in the database

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Rules = queries

Actor(344759, ‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).

Q1(y) :- Movie(x,y,z), z=1940.
Datalog: Facts and Rules

Facts = tuples in the database

- Actor(344759, 'Douglas', 'Fowley').
- Casts(344759, 29851).
- Casts(355713, 29000).
- Movie(7909, 'A Night in Armour', 1910).
- Movie(29000, 'Arizona', 1940).
- Movie(29445, 'Ave Maria', 1940).

Rules = queries

Q1(y) :- Movie(x,y,z), z=1940.

Find Movies made in 1940
Datalog: Facts and Rules

Facts = tuples in the database
Rules = queries

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Find Movies made in 1940

Actor(344759, ‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).

Q1(y) :- Movie(x,y,z), z=1940.

SELECT name
FROM Movie
WHERE year = 1940
Datalog: Facts and Rules

Facts = tuples in the database

Actor(344759, ‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).

Rules = queries

Q1(y) :- Movie(x, y, z), z = 1940.

Order of variable matters!

Find Movies made in 1940
Datalog: Facts and Rules

**Facts** = tuples in the database

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Actor(344759, 'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).

**Rules** = queries

Q1(y) :- Movie(iDontCare, y, z), z=1940.

Find Movies made in 1940
Datalog: Facts and Rules

**Facts** = tuples in the database

- Actor(344759, ‘Douglas’, ‘Fowley’).
- Casts(344759, 29851).
- Casts(355713, 29000).

**Rules** = queries

Q1(y) :- Movie(_,y,z), z=1940.

_ = “don’t care” variables

Find Movies made in 1940
Facts = tuples in the database

Actor(344759, 'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).

Rules = queries

Q1(y) :- Movie(x, y, z), z = 1940.

Q2(f, l) :- Actor(z, f, l), Casts(z, x), Movie(x, y, 1940).
Datalog: Facts and Rules

Facts = tuples in the database

- Actor(344759, 'Douglas', 'Fowley').
- Casts(344759, 29851).
- Casts(355713, 29000).
- Movie(7909, 'A Night in Armour', 1910).
- Movie(29000, 'Arizona', 1940).
- Movie(29445, 'Ave Maria', 1940).

Rules = queries

- Q1(y) :- Movie(x, y, z), z=1940.
- Q2(f, l) :- Actor(z, f, l), Casts(z, x), Movie(x, y, 1940).

Find Actors who acted in Movies made in 1940
Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

**Facts** = tuples in the database

- Actor(344759, 'Douglas', 'Fowley').
- Casts(344759, 29851).
- Casts(355713, 29000).
- Movie(7909, 'A Night in Armour', 1910).
- Movie(29000, 'Arizona', 1940).
- Movie(29445, 'Ave Maria', 1940).

**Rules** = queries

- Q1(y) :- Movie(x,y,z), z=1940.
- Q2(f,l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,1940).
- Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910), Casts(z,x2), Movie(x2,y2,1940).
Datalog: Facts and Rules

**Facts** = tuples in the database

- Actor(344759, 'Douglas', 'Fowley').
- Casts(344759, 29851).
- Casts(355713, 29000).
- Movie(7909, 'A Night in Armour', 1910).
- Movie(29000, 'Arizona', 1940).
- Movie(29445, 'Ave Maria', 1940).

**Rules** = queries

- Q1(y) :- Movie(x, y, z), z=1940.
- Q2(f,l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,1940).
- Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910), Casts(z,x2), Movie(x2,y2,1940).

Find Actors who acted in a Movie in 1940 and in one in 1910
Datalog: Facts and Rules

Facts = tuples in the database

Actor(344759, ‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).

Rules = queries

Q1(y) :- Movie(x,y,z), z=1940.
Q2(f,l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,1940).
Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),
            Casts(z,x2), Movie(x2,y2,1940).

Extensional Database Predicates = EDB = Actor, Casts, Movie
Intensional Database Predicates = IDB = Q1, Q2, Q3