Databases: The Continuing Saga

When last we left databases...
- We had decided they were great things
- We knew how to conceptually model them in ER diagrams
- We knew how to logically model them in the relational model
- We could formally model queries using
- We could query them using SQL – the relational approach
- Are there any other good ways to query?

Learning Goals
- Given a set of tuples (an input relation) and rules, compute the output relation for a Datalog program.
- Write Datalog programs to query an input relation.
- Explain why we want to extend RA or SQL with recursive queries. Provide good examples of such queries.
- Explain the importance of safe queries, and what makes a Datalog query safe.

Motivation (Surprisingly difficult)

- Write an SQL query to find all of the components required for a trike

Datalog

- Our final relational query language
- Based on logic notation (Prolog)
- Can express queries that are not expressible in relational algebra or standard SQL (recursion).
- Uses sets (like RA, unlike SQL)
- No grouping and aggregation, order by.
- Cleaner \(\rightarrow\) convenient for analysis

A nice and easy example to start

Likes(dee, jan).
Likes(jan, jamie).
Likes(dee, wally).
Likes(wally, jean).
Likes(A, C) :- Likes(A, B), Likes(B, C).

Loves(dee, jamie).
Loves(dee, jean).

Based on some facts on some rules, you can deduce new facts.
Predicates and Atoms
- Relations are represented by predicates.
- Tuples are represented by atoms.
  Likes(dee, jan)

Arithmetic comparison atoms:
  \(X < 100, \ X + Y + 5 > Z/2, \ X <> 42\)

- Negated atoms:
  NOT Likes(dee, jean)

Datalog Definitions
- A Datalog rule:
  \(\text{atom} :- \text{atom}_1, \ldots, \text{atom}_n\)

Datalog program = a collection of rules

A single rule can express exactly select-from-where queries.

The Meaning of Datalog Rules
Likes(dee, jan), Likes(jan, jamie), Likes(wally, jeanie), Loves(A, C) :- Likes(A, B), Likes(B, C).

Consider every assignment from the variables in the body to the constants in the database. (same variable name means require the same value)

If each atom in the body is in the database, then the tuple for the head is in the result.

Running example
Product (pid, name, price, category, maker-cid)
Purchase (buyer-sin, seller-sin, store, pid)
Company (cid, name, stock price, country)
Person(sin, name, phone number, city)

Projection practice
Product (pid, name, price, category, maker-cid)
Purchase (buyer-sin, seller-sin, store, pid)
Company (cid, name, stock price, country)
Person(sin, name, phone number, city)

Find the countries of all the companies
Selection

Product ( pid, name, price, category, maker-cid)
Purchase (buyer-sin, seller-sin, store, pid)
Company (cid, name, stock price, country)
Person(sin, name, phone number, city)

Selection is performed by either using the same variable, a constant, or adding an arithmetic comparison:
- Find all purchases with the same buyer and seller:
  RA: \( \sigma_{\text{buyer-sin}} = \sigma_{\text{seller-sin}} \) (Purchase)
  Datalog: \( \text{Ans}(B,B,S,P) \) :- Purchase(B,B,S,P)
- Find all Canadian companies:
  RA: \( \sigma_{\text{country}} = \text{Canada} \) (Company)
  Datalog: \( \text{Ans}(C,N,S, 'Canada') \) :- Company(C,N,S, 'Canada')

Selection practice

Product ( pid, name, price, category, maker-cid)
Purchase (buyer-sin, seller-sin, store, pid)
Person(sin, name, phone number, city)

Find all products over $99.99:
RA: \( \sigma_{\text{price}>99.99} \) (Product)

Find all English companies with stock prices less than $100:
RA: \( \sigma_{\text{country} = \text{Canada}} \) (Company)
Datalog: \( \text{Ans}(C,N,S, 'Canada') \) :- Company(C,N,S, 'Canada')

Selection & Projection

Product ( pid, name, price, category, maker-cid)
Purchase (buyer-sin, seller-sin, store, pid)
Company (cid, name, stock price, country)
Person(sin, name, phone number, city)

Find the names of all products over $99.99:
RA: \( \pi_{\text{name}}(\sigma_{\text{price}>99.99}(\text{Product})) \)

Selection & Projection and Joins

Product ( pid, name, price, category, maker-cid)
Purchase (buyer-sin, seller-sin, store, pid)
Company (cid, name, stock price, country)
Person(sin, name, phone number, city)

Joins are performed by using the same variable in different relations:
- Find store names where Fred bought something:
  RA: \( \pi_{\text{store}}(\sigma_{\text{name}} = \text{Fred}(\text{Person})) \) \( \bowtie \) \( \sigma_{\text{buyer-sin}} = \text{seller-sin} \) (Purchase)
  Datalog: \( \text{S}(N) :- \text{Person}(S, 'Fred',T,C), \text{Purchase}(S,L,N,P) \)

Anonymous Variables

Product ( pid, name, price, category, maker-cid)
Purchase (buyer-sin, seller-sin, store, pid)
Company (cid, name, stock price, country)
Person(sin, name, phone number, city)

Find names of people who bought from "Gizmo Store"
- E.g.: \( \text{Ans4}(N) \) :- Person(S, N, \_ , \_ , \_ ), Purchase (S, \_ ,"Gizmo Store", \_)
  Each \_ means a fresh, new variable
  Very useful: makes Datalog even easier to read

Exercise part 1

Product ( pid, name, price, category, maker-cid)
Purchase (buyer-sin, seller-sin, store, pid)
Company (cid, name, stock price, country)
Person(sin, name, phone number, city)

Ex #1: Find SINs of people who bought products in the "computers" category.

Ex #2: Find the sin of people who bought Canadian products
Exercise part 2

Product (pid, name, price, category, maker-cid)
Purchase (buyer-sin, seller-sin, store, pid)
Company (cid, name, stock price, country)
Person(sin, name, phone number, city)

Ex #3: Find names of people who bought Canadian products that cost under 50

Multiple Datalog Rules

Product (pid, name, price, category, maker-cid)
Purchase (buyer-sin, seller-sin, store, pid)
Company (cid, name, stock price, country)
Person(sin, name, phone number, city)

Find names of people that are either buyers or sellers:

A(N) :- Person(S,N,A,B), Purchase(S,C,D,E)
A(N) :- Person(S,N,A,B), Purchase(C,S,D,E)

Multiple rules correspond to union

Exercise part 3

Product (pid, name, price, category, maker-cid)
Purchase (buyer-sin, seller-sin, store, pid)
Company (cid, name, stock price, country)
Person(sin, name, phone number, city)

Ex #4: Find sins of people who bought stuff from a person named Joe or bought products from a company whose stock prices is more than $50.

find all of the components required for trike

Components (Part, Subpart) :- Assembly (Part, Subpart, Qty)

Components (Part, Subpart) :- Assembly (Part, Part2, Qty), Components(Part2, Subpart)

Component(trike, wheel)
Component(trike, frame)
Component(trike, spoke)
Component(trike, tire)
Component(trike, seat)
Component(trike, pedal)
Component(trike, rim)
Component(trike, tube)

When should we stop??

The Fixpoint Operator

A fixpoint of a function f is a value v

f(v) = v

Example

do double : doubles every element of a list
   - Double(1,2,5) = (2,4,10)
   - Double*(1,2,5) = (1,2,4,5,10)
   - Double*(even integers) = (even integers) (fixpoint)

Fixpoint over set of tuples

Components (Part, Subpart) :- Assembly (Part, Subpart, Qty)

Components (Part, Subpart) :- Assembly (Part, Part2, Qty), Components(Part2, Subpart)

Component(trike, wheel)
Component(trike, frame)
Component(trike, spoke)
Component(trike, tire)
Component(trike, seat)
Component(trike, pedal)
Component(trike, rim)
Component(trike, tube)

f(v) = v

v is a Fixpoint for f

I cannot use any of my rules to deduce any new facts
Rule Safety

- Every variable in the head of a rule must also appear in the body.
  - PriceYarts(Part, Price) :- Assembly(Part, Subpart, Qty), Qty > 2.
  - Can generate infinite new facts

- Every variable must appear in a relation
  - Ans(Id) :- Product(Id, Name, Price, Category, Cid), Id < Stock_price

What is the value of stock_price?

Rule Safety (with negation)

- Every variable that appears anywhere in the query must appear also in a relational, non-negated atom in the query.
  - Ans(Id) :- Product(Id, Name, Price, Category, Cid), NOT Company(Cid, Cname, Stock, Country)
  - Cname, Stock, Country are unsafe

Exercise part 4

- Product(pid, name, price, category, maker-cid)
- Purchase(buyer-sin, seller-sin, store, pid)
- Company(cid, name, stock price, country)
- Person(sin, name, phone number, city)
- Ex #5: Find the sins of people who are not named ‘Joe’

More on rule safety

- Is trike considered a big part or a small part?

<table>
<thead>
<tr>
<th>trike</th>
<th>wheel</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>tire</td>
<td></td>
<td></td>
</tr>
<tr>
<td>frame</td>
<td></td>
<td></td>
</tr>
<tr>
<td>frame</td>
<td></td>
<td></td>
</tr>
<tr>
<td>seat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pedal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spoke</td>
<td></td>
<td></td>
</tr>
<tr>
<td>size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rim</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tube</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- We would need to use Stratification(24.3.1) to overcome this problem (Not tested on exam)

Defining Views

- VancouverView(Buyer, Seller, Product, Store) :-
  - Person(Buyer, "Vancouver", Phone),
  - Purchase(Buyer, Seller, Product, Store)
  - not Purchase(Buyer, Seller, Product, "The Bay")

- Ans6(Buyer) :- VancouverView(Buyer, "Joe", Prod, Store)
- Ans6(Buyer) :- VancouverView(Buyer, Seller, Store, Prod),
  - Product(Prod, Price, Cat, Maker)
  - Company(Maker, Sp, Country), Sp > 50.

What is returned by Ans6?

Taking it to the next level

- Say you’re planning a beach vacation
- And you wanted to find if it’s possible to get from YVR to OGG (that’s on Maui)

Your available information:
- Flight(airline,num,origin,destination)

Now what?
A more general Example: Transitive Closure

Suppose we represent a graph with relation $Edge(X,Y)$: $Edge(a,b)$, $Edge(a,c)$, $Edge(b,d)$, $Edge(c,d)$, $Edge(d,e)$

How can I express the query:
Find all nodes reachable from $a$.

$Path(X,Y) \leftarrow Edge(X,Y)$.
$Path(X,Y) \leftarrow Path(X,Z), Path(Z,Y)$.

Evaluating Recursive Queries

$Path(X,Y) \leftarrow Edge(X,Y)$.
$Path(X,Y) \leftarrow Path(X,Z), Path(Z,Y)$.

Semantics: evaluate the rules until a fixed point:
Iteration #0: $Edge$: $\{(a,b), (a,c), (b,d), (c,d), (d,e)\}$
$Path$: $\{\}$
Iteration #1: $Path$: $\{(a,b), (a,c), (b,d), (c,d), (d,e)\}$
Iteration #2: $Path$ gets the new tuples: $\{(a,d), (b,e), (c,e)\}$
$Path$: $\{(a,b), (a,c), (b,d), (c,d), (d,e), (a,d), (b,e), (c,e)\}$
Iteration #3: $Path$ gets the new tuple: $(a,e)$
$Path$: $\{(a,b), (a,c), (b,d), (c,d), (d,e), (a,d), (b,e), (c,e), (a,e)\}$
Iteration #4: Nothing changes → Stop.

Note: # of iterations depends on the data. Cannot be anticipated by only looking at the query!

A fun Example

Kevin Bacon 6 degrees of separation 6 degrees of Kevin Bacon

More examples

- Given:
  Movie(id, title)
  Actor(id, name)
  Role(movie-id, actor-id, character)
- Find names of actors who have “Bacon numbers” (assume there’s only one “Kevin Bacon”)

Recursive SQL? Sometimes...

Given: Assembly(Part, Subpart, Quantity)
Find: all recursive components of an item
Datalog:
Comp(Part, Subpt) :- Assembly(Part, Subpt, Qty).
Comp(Part, Subpt) :- Assembly(Part, Part2, Qty), Comp(Part2, Subpt).
SQL:
WITH RECURSIVE Comp(Part, Subpt) AS
UNION
(SELECT A2.Part, C1.Subpt
FROM Assembly A2, Comp C1
WHERE A2.Subpt=C1.Part)
SELECT * FROM Comp C2

Skip the stuff on Magic Sets

- That’s Datalog
- It’s simple
- It’s based on logic
- It’s easy to see the join patterns (especially with anonymous variables)
Learning Goals Revisited

- Given a set of tuples (an input relation) and rules, compute the output relation for a Datalog program.
- Write Datalog programs to query an input relation.
- Explain why we want to extend RA or SQL with recursive queries. Provide good examples of such queries.
- Explain the importance of safe queries, and what makes a Datalog query safe.