Today's Lecture

- Useful queries that RA/RC/basic SQL cannot express?
- Datalog: a logical query language
- Recursion in Datalog
- Semantics and computation

Some queries cannot be written in basic SQL

- Consider the following relation schema
  Assembly(Part, Subpart, qty)
- Query: Find all the subparts of part “Wheel”

<table>
<thead>
<tr>
<th>Part</th>
<th>Subpart</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trike</td>
<td>Wheel</td>
<td>3</td>
</tr>
<tr>
<td>Trike</td>
<td>Frame</td>
<td>1</td>
</tr>
<tr>
<td>Frame</td>
<td>Seat</td>
<td>1</td>
</tr>
<tr>
<td>Frame</td>
<td>Pedal</td>
<td>1</td>
</tr>
<tr>
<td>Wheel</td>
<td>Spoke</td>
<td>2</td>
</tr>
<tr>
<td>Wheel</td>
<td>Tire</td>
<td>1</td>
</tr>
<tr>
<td>Tire</td>
<td>rim</td>
<td>1</td>
</tr>
<tr>
<td>Tire</td>
<td>Tube</td>
<td>1</td>
</tr>
</tbody>
</table>
Tree view of Assembly relation

- First, find all subparts of “wheel”
  - a select, followed by project
  - Answer: spoke, tire
- Then, find all subparts of the results above
  - join result above with Assembly relation
  - Answer: rim, tube
- Take the union of answers above
  - spoke, tire, rim, tube
- But what if there are subparts beneath rim or tube?

• The above query works only for this given instance of relation
• More precisely, it works if wheel has at most subparts of depth 2
• However, we cannot run the same query in another instance of the relation where wheel has subparts of depth 3 or 4 ...

It is impossible to write a query, (whether in RA, RC, or basic SQL), that will find all subparts of the part wheel for every possible instance of the Assembly relation.
The problem

- Finding all subparts of the part wheel requires us to go down the tree as far as possible
- To translate this into a query, it will require as many joins as needed
- The number of joins cannot be fixed in advance. It really depends on the depth of the subtree rooted at “wheels”. Therefore, there is no way to write a query that will work for all possible instances of Assembly
- Therefore, this query cannot be expressed in basic SQL, RA, or RC; they do not have constructs for specifying “as many joins as needed”
- But clearly, this type of recursive queries is often asked

As a consequence ...

- Since recursive queries are frequently asked, SQL is extended with constructs for recursive queries
- Note that it is possible to write recursive programs through the use of Embedded SQL (without resorting to SQL’s recursive constructs) but this lacks logical independence
- We next learn a logical query language: Datalog. Datalog is essentially prolog with no functions allowed
- In contrast with SQL, writing recursive queries in Datalog is simple
Datalog: Using logic as a query language

- The general idea behind Datalog is to use Horn-clauses -- “if-then” rules -- as a query language for relational databases.
- Predicates correspond to relations
  - The predicate Assembly(X, Y, Z) is true iff (X, Y, Z) is a tuple in the Assembly relation
- Note the positional interpretation to arguments
  - X, Y, Z represents values in the 1st, 2nd, and 3rd columns of the Assembly relation respectively
- Ground predicate: all arguments are constants
  - Assembly(wheel, spoke, 2)
- Convention: variables as represented as uppercase letters, constants are represented in lowercase

Datalog rules

- A rule has the form p :- q
  - “:-” to be read as “if”
  - This means if q is true then p is true
  - Subparts(X, Y) :- Assembly(X, Y, Z), Assembly(X,U,V)

- Body consists of a logical AND or zero or more subgoals
- A datalog program is a collection of rules
**Extensional vs. Intensional Predicates**

- EDB – extensional database consists of predicates which are defined in relations
  
  \[
  \text{Assembly(trike, wheel, 3)}
  \]
  \[
  \text{Assembly(trike, frame, 1)}
  \]
  \[
  \text{Assembly(frame, wheel, 1)}
  \]
  
  …

- IDB – intensional database consists of predicates that are defined through some datalog rules.

**Example**

- Suppose the Assembly relation on the right is defined in the EDB
- Find all immediate subparts of “frame”

\[
\text{FrameSubparts}(X) \leftarrow \text{Assembly}(\text{Frame}, X, Y)
\]

**Answer:**

<table>
<thead>
<tr>
<th>Part</th>
<th>Subpart</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trike</td>
<td>Wheel</td>
<td>3</td>
</tr>
<tr>
<td>Trike</td>
<td>Frame</td>
<td>1</td>
</tr>
<tr>
<td>Frame</td>
<td>Seat</td>
<td>1</td>
</tr>
<tr>
<td>Frame</td>
<td>Pedal</td>
<td>1</td>
</tr>
<tr>
<td>Wheel</td>
<td>Spoke</td>
<td>2</td>
</tr>
<tr>
<td>Wheel</td>
<td>Tire</td>
<td>1</td>
</tr>
<tr>
<td>Tire</td>
<td>rim</td>
<td>1</td>
</tr>
<tr>
<td>Tire</td>
<td>Tube</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\text{FrameSubparts(Seat)}
\]

\[
\text{FrameSubparts(Pedal)}
\]
**Example - union in datalog**

- Find all immediate subparts of frame or tire
- Use datalog rules with the same head

Ans(X) :- Assembly(Frame, X, Y)
Ans(X) :- Assembly(Tire, X, Y)

**Example - find all subparts of wheel**

WheelSubparts(Y) :- Assembly(Wheel, Y, _)
WheelSubparts(Y) :- WheelSupbparts(X), Assembly(X,Y, _)

- The output of this datalog program is what we can deduce or infer using these rules
- Consider the first rule
  - If Assembly(wheel, spoke, 2), then WheelSubparts(spoke)
  - If Assembly(wheel, tire, 1), then WheelSubparts(tire)

<table>
<thead>
<tr>
<th>Part</th>
<th>Subpart</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trike</td>
<td>Wheel</td>
<td>3</td>
</tr>
<tr>
<td>Trike</td>
<td>Frame</td>
<td>1</td>
</tr>
<tr>
<td>Frame</td>
<td>Seat</td>
<td>1</td>
</tr>
<tr>
<td>Frame</td>
<td>Pedal</td>
<td>1</td>
</tr>
<tr>
<td>Wheel</td>
<td>Spoke</td>
<td>2</td>
</tr>
<tr>
<td>Wheel</td>
<td>Tire</td>
<td>1</td>
</tr>
<tr>
<td>Tire</td>
<td>rim</td>
<td>1</td>
</tr>
<tr>
<td>Tire</td>
<td>Tube</td>
<td>1</td>
</tr>
</tbody>
</table>
Example

- Output so far: WheelSubparts(spoke), WheelSubparts(tire)
  - Consider the second rule:
    - If WheelSubparts(tire), Assembly(tire, rim, 1), then WheelSubpart(rim)
    - If WheelSubparts(tire), Assembly(tire, tube, 1), then WheelSubpart(tube)
  - Output so far: WheelSubparts(spoke), WheelSubparts(tire), WheelSubpart(rim), WheelSubpart(tube)
  - Consider the second rule again:
    - No new predicates can be inferred!
    - Therefore the result is \{ WheelSubparts(spoke), WheelSubparts(tire), WheelSubpart(rim), WheelSubpart(tube) \}

When is a datalog program recursive?

- The datalog program, WheelSubparts, is recursive
- Construct a dependency graph from the datalog program
- Example: P :– S1, ..., Sn

\[ P \rightarrow \quad S_1 \quad \rightarrow \quad \vdots \quad \rightarrow \quad S_n \]

- The program is recursive if the graph contains a cycle
Examples

WheelSubparts(Y) :- Assembly(wheel, Y, _)
WheelSubparts(Y) :- WheelSupbparts(X), Assembly(X,Y, _)

WheelSupbparts Assembly

S(X,Y,Z) :- T(X,Y,Z)
P(X,Y) :- R(X,a,Y), S(Y,b,c)
P(X,Z) :- R(X,Z,Y)

S T
P R

S(X,Y,Z) :- T(X,Y,Z)
P(X,Y) :- R(X,a,Y), S(Y,b,c)
T(X,Y,Y) :- P(X,Y)

S T
P R

SQL with recursion

• SQL: 1999 syntax
• WITH RECURSIVE IDB(arguments) AS
  <def of IDB (a query involving IDB)>

Select * from IDB
**SQL with recursion**

- SQL:1999 syntax
- **WITH RECURSIVE** WheelSubparts(Part) AS
  - (SELECT A.subpart
    FROM Assembly A
    WHERE A.part="Wheel")
  - UNION
  - (SELECT A.subpart
    FROM Assembly A, WheelSubparts W
    WHERE W.subpart = A.part)

  SELECT * FROM WheelSubparts

**Example - Unsafe datalog**

- Age(A) :- Employee(X, Y, Z), A > Y
- Names(N) :- ~Student(N, X, Y)

- Infinite answers
  - If there exists Employee(John, 35, Boston), then there are infinitely many numbers greater than 35
  - There are infinitely many students not in the database

- Fact: Impossible to give a necessary and sufficient condition to characterize when a datalog program is safe
- Solution: For practical purposes, sufficient conditions are given which may be more restrictive than needed
Safe Datalog Programs

• A rule is safe if
  – every variable X in the rule occurs at least once in a positive goal predicate P in the body and P is safe (note that it does not count if a variable occurs in some arithmetic expression)
  – a predicate P is safe if P is a relational predicate or every rule defining P is safe

• Except for the datalog programs in the previous slide, all datalog programs you have seen so far are safe

Unsafe
Single(X) :- ~Married(Y,Z)
Spouse(X,Y) :- Married(X,Z)
Divorced(X) :- ~Married(X,Z), Z > 30
Age(Z) :- Employee(X,Y,A), Z > A

Safe
Spouse(X,Y) :- Married(X,Y)
Persons(X) :- Married(X,Y)

Semantics of a Datalog program

• To understand the meaning of a datalog program (with no negation for now), we need to know
  – The relationship between rules and logical sentences
  – What is a model?
  – What is the intended model?
The relationship between rules and logical sentences

- $R_1(u_1) :- R_2(u_2), ..., R_n(u_n)$ where $u_i$ are sequences of variables
- $\forall X_1, ..., X_m (R_2(u_2), ..., R_n(u_n) \rightarrow R_1(u_1))$ where $X_i$ are variables in the datalog rule
- Informally, the model of a datalog program $P$ is an instance over the edb and idb of $P$ that satisfies the logical sentences of $P$

The relationship between rules and logical sentences

- Obviously, there can be many models of a datalog program
  - $S(a) :-$  
  - $R(x) :- S(x)$
- $R(a), S(a)$ is a model. $R(a), R(b), S(a)$ is also a model.
- There are infinitely many models for this program. Which is the right one?
- The least model semantics: The semantics of the datalog program is the least model; the least model of a set of sentences is the smallest possible model
Why the minimal model?

- A fact is true if it can be found in all possible models of the datalog program containing the input relations.
- Let D be the input relations.

- Fact: The minimal model of a datalog program containing D = the intersection of all possible models of that datalog program containing D.

- Therefore, the minimal model contains all true facts.
- How do we compute the minimal model?

Computing the minimal model - Naïve Evaluation

- Consider a datalog program (with no negation).
- For each IDB $P_i$
  
  $P_i = \{\}$

- Repeat until no change in IDBs {
  
  apply rules in the datalog program on edb and idbs
  
  For each $i$, if new facts $F_i$ are inferred
  
  $P_i = P_i \cup F_i$

  }Consider every possible assignment of values to variables. For every such assignment which makes all the subgoals true, the tuple corresponding to the head is true and is the current fact inferred.
Example

WheelSubparts(Y) :- Assembly(wheel, Y, _)
WheelSubparts(Y) :- WheelSubparts(X), Assembly(X,Y, _)

• WS$^0$ = {}
• Round 1:
  – New facts inferred = WS(spoke), WS(tire)
  – WS$^1$ = { WS(spoke), WS(tire) }

<table>
<thead>
<tr>
<th>Part</th>
<th>Subpart</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trike</td>
<td>Wheel</td>
<td>3</td>
</tr>
<tr>
<td>Trike</td>
<td>Frame</td>
<td>1</td>
</tr>
<tr>
<td>Frame</td>
<td>Seat</td>
<td>1</td>
</tr>
<tr>
<td>Frame</td>
<td>Pedal</td>
<td>1</td>
</tr>
<tr>
<td>Wheel</td>
<td>Spoke</td>
<td>2</td>
</tr>
<tr>
<td>Wheel</td>
<td>Tire</td>
<td>1</td>
</tr>
<tr>
<td>Tire</td>
<td>rim</td>
<td>1</td>
</tr>
<tr>
<td>Tire</td>
<td>Tube</td>
<td>1</td>
</tr>
</tbody>
</table>

• Round 2:
  – New facts inferred – WS(rim), WS(tube)
    • If WheelSubparts(tire), Assembly(tire, rim, 1), then WheelSubpart(rim)
    • If WheelSubparts(tire), Assembly(tire, tube, 1), then WheelSubparts(tube)
  – WS$^2$ = { WS(spoke), WS(tire), WS(rim), WS(tube) }
• Round 3:
  – No new facts can be inferred!
  – STOP

Note that WS$^0$ ⊆ WS$^1$ ⊆ WS$^2$ = WS$^3$

Winter 2003
The fixpoint operator

• WS³ is a fixpoint for the WheelSubparts program
• A fixpoint v of a function f is such that f(v) = v
• Example:
  – Double = doubles every element in an input set
  – Double({1,2,3}) = {2,4,6}
  – Define Double'(L) = L ∪ Double(L)
  – The set of all integers is a fixpoint for Double’
  – The set of all even integers is another fixpoint for Double’
• The minimal model of a datalog program is the smallest fixpoint

RA to Datalog

• Selection σ_C(R):
  – Result(X) :- R(X), C
• Projection π_{32}(R):
  – Result(Y) :- R(X,Y)
• Cross-Product R x S:
  – Result(X,Y) :- R(X), S(Y)
• Set-Difference R – S:
  – Result(X) :- R(X), ¬S(X)
• Union R ∪ S:
  – Result(X) :- R(X)
  – Result(X) :- S(X)
Selection

• AND condition is straightforward

\[ \sigma_{\text{part}="\text{wheel}" \text{ AND} \text{ qty}>1} \text{ (Assembly)} \]

• Result(P,S,Q) :- Assembly(P,S,Q), P="wheel", Q>1

• OR condition can be simulated using union

\[ \sigma_{\text{part}="\text{wheel}" \text{ OR} \text{ qty}>1} \text{ (Assembly)} \]

• Result(P,S,Q) :- Assembly(P,S,Q), P="wheel"

• Result(P,S,Q) :- Assembly(P,S,Q), Q>1

Projection

• Project is straightforward

• The variables corresponding to the projected attributes are placed in the head

\[ \pi_{\text{part, subpart}} \text{ (Assembly)} \]

• Result(P,S) :- Assembly(P,S,Q)
Product and Join

- The product of two relations $R \times S$ is expressed by a single rule with both relations as subgoals using distinct variables.
- The head contains all the variables that occur in the subgoals.
  - $\text{Result}(X,Y,Z,W) \leftarrow \text{Students}(X,Y), \text{Enrolled}(Z,W)$

- Recall that given $R(A,B)$ and $S(B,C)$, $R \bowtie S = \sigma_{R.B=S.B} R \times S$.
- So we can apply the rules we just learnt.
  - $\text{Result}(X,Y,W) \leftarrow R(X,Y), S(Z,W), Y=Z$, OR
  - $\text{Result}(X,Z,W) \leftarrow R(X,Y), S(Z,W), Y=Z$
- Or, we could substitute out the equality above to “reuse” variables.
  - $\text{Result}(X,Y,Z) \leftarrow R(X,Y), S(Y,Z)$

Union and Difference

- In class
Converting a complex RA expression into Datalog

- How can we convert the following relational algebra expression into datalog?
- $\pi_{name}(\sigma_{\text{grade}<3.0}(\text{Students} \bowtie \text{Enrolled}))$

- For each internal node in the query tree, create the corresponding rule
- The IDB corresponding to root is the result.

- $\text{Tem1}(S,N,Z,G) :- \text{Student}(S,N), \text{Enrolled}(S,Z,G)$
- $\text{Tem2}(S,N,Z,G) :- \text{Tem1}(S,N,Z,G), G<3.0$
- $\text{Ans}(N) :- \text{Tem2}(S,N,Z,G)$

Datalog to RA?

- We have seen that RA can be translated into Datalog
- Obviously, not every datalog program can be translated into relational algebra
  - A recursive datalog program can be translated into RA if a fixpoint operator is added to RA
- In general, only safe non-recursive datalog programs can be translated into relational algebra
Non-recursive Datalog

- Non-recursive datalog = datalog without recursion
- Assume also no negation
- Then the datalog program can be represented as a acyclic graph and evaluated in a bottom-up manner

\[
\begin{align*}
S(X,Y,Z) & : \neg T(X,Y,Z) \\
P(X,Y) & : R(X, a, Y), S(Y, b, c) \\
P(X, Z) & : R(X, Z, Y)
\end{align*}
\]

A useful thing to know

- Open-world assumption:
  - The database contains true facts but there are possibly facts that are true and not recorded in the database
  - If a fact is not in the database, should we consider it as true or false?
- Closed-world assumption:
  - The database completely contains all true facts
  - Therefore, if a fact is not in the database, it is considered false
Summary

- RA/basic SQL/RC cannot compute all our desired queries; in particular, recursive queries cannot be expressed in RA/basic SQL/RC unless a fix point operator is added
- Datalog is a logical query language where recursive queries can be naturally expressed
- Semantics of datalog programs (no negation) is the minimal model semantics
- Naïve evaluation computes the minimal model