Datalog
Datalog

• A nonprocedural language based on Prolog
  – Describe what instead of how: specifying the information desired without giving a specific procedure of obtaining that information
  – Resemble the syntax of Prolog

• A purely declarative manner
  – Simplify writing simple queries
  – Make query optimization easier
Basic Example

• Define a view relation $v1$ containing account numbers and balances for accounts at the Perryridge branch with a balance of over $700$
  – $v1(A, B) :- account(A, “Perryridge”, B), B > 700$
  – for all $A, B$
    if $(A, “Perryridge”, B) \in account$ and $B > 700$
    then $(A, B) \in v1$

• A Datalog program consists of a set of rules
Evaluation of a Datalog Program

- $v_1(A, B) \ :- \ account(A, \ "Perryridge", \ B), \ B > 700$

<table>
<thead>
<tr>
<th>account-number</th>
<th>branch-name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-101</td>
<td>Downtown</td>
<td>500</td>
</tr>
<tr>
<td>A-215</td>
<td>Mianus</td>
<td>700</td>
</tr>
<tr>
<td>A-102</td>
<td>Perryridge</td>
<td>400</td>
</tr>
<tr>
<td>A-305</td>
<td>Round Hill</td>
<td>350</td>
</tr>
<tr>
<td>A-201</td>
<td>Perryridge</td>
<td>900</td>
</tr>
<tr>
<td>A-222</td>
<td>Redwood</td>
<td>700</td>
</tr>
<tr>
<td>A-217</td>
<td>Perryridge</td>
<td>750</td>
</tr>
</tbody>
</table>

- $A-201 \quad 900$
- $A-217 \quad 750$
Retrieving Tuples

• Retrieve the balance of account number “A-217” in the view relation v1
  \[ ? \, v1(\text{“A-217”}, B) \]
  – Answer: (A-217, 750)

• Find account number and balance of all accounts in v1 that have a balance greater than 800
  \[ ? \, v1(A,B), B > 800 \]
  – Answer: (A-201, 900)
A Program of Multiple Rules

• The interest rates for accounts
  \[ \text{interest-rate}(A, 5) :\neg \text{account}(A, N, B), B < 10000 \]
  \[ \text{interest-rate}(A, 6) :\neg \text{account}(A, N, B), B \geq 10000 \]

• The set of tuples in a view relation is defined as the union of all the sets of tuples defined by the rules for the view relation
Negation

• Define a view relation $c$ that contains the names of all customers who have a deposit but no loan at the bank

\[ c(N) :– \text{depositor}(N, A), \textbf{not} \ \text{is-borrower}(N). \]
\[ \text{is-borrower}(N) :– \text{borrower} (N,L) \]

• Using \textbf{not} \ borrower \ $(N, L)$ in the first rule results in a different meaning, namely there is some loan L for which N is not a borrower
  – To prevent such confusion, we require all variables in negated “predicate” to also be present in non-negated predicates
Syntax of Datalog Rules

• Positive literal: $p(t_1, t_2 \ldots, t_n)$
  – $p$ is the name of a relation with $n$ attributes
  – Each $t_i$ is either a constant or variable
  – Example: account(A, “Perryridge”, B)

• Negative literal: \textbf{not} $p(t_1, t_2 \ldots, t_n)$

• Comparison and arithmetic are treated as positive predicates
  – $X > Y$ is treated as a predicate $>(X,Y)$
  – $A = B + C$ is treated as $+(B, C, A)$
Fact and Rules

- Fact $p(v_1, v_2, ..., v_n)$
  - Tuple $(v_1, v_2, ..., v_n)$ is in relation $p$

- Rules: $p(t_1, t_2, ..., t_n) :– L_1, L_2, ..., L_m$
  - Each of the $L_i$’s is a literal
  - Head – the literal $p(t_1, t_2, ..., t_n)$
  - Body – the rest of the literals

- A Datalog program is a set of rules
An Example Datalog Program

• Define interest on Perryridge accounts

\[
\text{interest}(A, I) :- \text{account}(A, \text{“Perryridge”}, B), \\
\quad \text{interest-rate}(A, R), I = B \times R / 100. \\
\text{interest-rate}(A, 5) :- \text{account}(A, N, B), B < 10000. \\
\text{interest-rate}(A, 6) :- \text{account}(A, N, B), B \geq 10000. 
\]
Dependency of View Relations

• View relation $v_1$ depends directly on $v_2$ if $v_2$ is used in the expression defining $v_1$
  – Relation interest depends directly on relations interest-rate and account

• View relation $v_1$ depends indirectly on $v_2$ if there is a sequence of intermediate relations $v_1=i_1, \ldots, i_n=v_2$ such that $v_j$ depends directly on $v_{j+1}$ for $1 \leq j < n$
  – Relation interest depends indirectly on relation account

• View relation $v_1$ depends on $v_2$ if $v_1$ depends directly or indirectly on $v_2$
Recursive Relation

• A view relation $v$ is recursive if it depends on itself, otherwise, it is nonrecursive

• An example – defining the relation employment

  $\text{empl}(X, Y) :- \text{manager}(X, Y)$.
  $\text{empl}(X, Y) :- \text{manager}(X, Z), \text{empl}(Z, Y)$
Semantics of Nonrecursive Datalog

• A ground instantiation of a rule (or simply instantiation) is the result of replacing each variable in the rule by some constant
  – Rule: \( v_1(A, B) \) :– \( \text{account}(A, \text{"Perryridge"}, B), B > 700. \)
  – An instantiation:
    \( v_1(\text{"A-217"}, 750) \) :– \( \text{account}(\text{"A-217"}, \text{"Perryridge"}, 750), 750 > 700. \)

• The body of rule instantiation \( R' \) is satisfied in a set of facts (database instance) \( I \) if
  – For each positive literal \( q_i(v_{i,1}, ..., v_{i,n_i}) \) in the body of \( R' \), \( I \) contains the fact \( q_i(v_{i,1}, ..., v_{i,n_i}) \); and
  – For each negative literal \( \text{not } q_j(v_{j,1}, ..., v_{j,n_j}) \) in the body of \( R' \), \( I \) does not contain the fact \( q_j(v_{j,1}, ..., v_{j,n_j}) \)
Inferring Facts

- The set of facts that can be inferred from a given set of facts $l$ using rule $R$ as: $\text{infer}(R, l) = \{ p(t_1, \ldots, t_n) \mid \text{there is a ground instantiation } R' \text{ of } R \text{ where } p(t_1, \ldots, t_n) \text{ is the head of } R', \text{ and the body of } R' \text{ is satisfied in } l \}$

- Given a set of rules $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$, define
  \[
  \text{infer}(\mathcal{R}, l) = \text{infer}(R_1, l) \cup \text{infer}(R_2, l) \cup \ldots \cup \text{infer}(R_n, l)
  \]
Example

• Rule: \( v1(A,B) :- \) account \((A, \text{“Perryridge”}, B), B > 700 \)

<table>
<thead>
<tr>
<th>account-number</th>
<th>branch-name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-101</td>
<td>Downtown</td>
<td>500</td>
</tr>
<tr>
<td>A-215</td>
<td>Mianus</td>
<td>700</td>
</tr>
<tr>
<td>A-102</td>
<td>Perryridge</td>
<td>400</td>
</tr>
<tr>
<td>A-305</td>
<td>Round Hill</td>
<td>350</td>
</tr>
<tr>
<td>A-201</td>
<td>Perryridge</td>
<td>900</td>
</tr>
<tr>
<td>A-222</td>
<td>Redwood</td>
<td>700</td>
</tr>
<tr>
<td>A-217</td>
<td>Perryridge</td>
<td>750</td>
</tr>
</tbody>
</table>

A set of facts I

\[ \text{infer(R, I)} \]

<table>
<thead>
<tr>
<th>account-number</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>A-217</td>
<td>750</td>
</tr>
</tbody>
</table>
Layer the View Relations

- Program
  
  \[
  \text{interest}(A, I) \leftarrow \text{perryridge-account}(A, B), \text{interest-rate}(A, R), I = B \times R/100. \\
  \text{perryridge-account}(A, B) \leftarrow \text{account}(A, \text{“Perryridge”}, B). \\
  \text{interest-rate}(A, 5) \leftarrow \text{account}(N, A, B), B < 10000. \\
  \text{interest-rate}(A, 6) \leftarrow \text{account}(N, A, B), B \geq 10000. \\
  \]

```
Layers

• A relation is in layer 1 if all relations used in the bodies of rules defining it are stored in the database.

• A relation is in layer 2 if all relations used in the bodies of rules defining it are either stored in the database, or are in layer 1.

• A relation $p$ is in layer $i + 1$ if
  – It is not in layers 1, 2, ..., $i$.
  – All relations used in the bodies of rules defining a $p$ are either stored in the database, or are in layers 1, 2, ..., $i$. 
Semantics of a Program

- Let the layers in a given program be 1, 2, ..., n. Let $\mathcal{R}_i$ denote the set of all rules defining view relations in layer i.
- Define $I_0 =$ the set of facts stored in the database.
- Recursively define $l_{i+1} = l_i \cup \text{infer}(\mathcal{R}_{i+1}, l_i)$.
- The set of facts in the view relations defined by the program (also called the semantics of the program) is given by the set of facts $l_n$ corresponding to the highest layer n.
Example

• Program

\[
\text{interest}(A, I) :\text{--} \ perryridge\text{-account}(A,B), \\
\hspace{1cm} \text{interest-rate}(A,R), I = B \times R/100.
\]

\[
perryridge\text{-account}(A,B) :\text{--} \account(A, \text{``Perryridge''}, B).
\]

\[
\text{interest-rate}(A,5) :\text{--} \account(N, A, B), B < 10000.
\]

\[
\text{interest-rate}(A,6) :\text{--} \account(N, A, B), B \geq 10000.
\]

• \(I_0\): account
• \(I_1\): account, insterst-rate
• \(I_2\): account, interst-rate, interest
Safety

• Unsafe rules – lead to infinite answers
  – $gt(X, Y) :- X > Y$
  – $not-in-loan(B, L) :- not \ loan(B, L)$
  – $P(A) :- q(B)$

• Safety conditions
  – Every variable that appears in the head of the rule also appears in a non-arithmetic positive literal in the body of the rule
  – Every variable appearing in a negative literal in the body of the rule also appears in some positive literal in the body of the rule

• If a nonrecursive Datalog program satisfies the safety conditions, then all the view relations defined in the program are finite
Relational Operations

- Project out attribute *account-name* from account.
  
  \[ \text{query}(A) :- \text{account}(A, N, B). \]

- Cartesian product of relations \( r_1 \) and \( r_2 \).
  
  \[ \text{query}(X_1, X_2, ..., X_n, Y_1, Y_1, Y_2, ..., Y_m) :- \\
  r_1(X_1, X_2, ..., X_n), r_2(Y_1, Y_2, ..., Y_m). \]

- Union of relations \( r_1 \) and \( r_2 \).
  
  \[ \text{query}(X_1, X_2, ..., X_n) :- r_1(X_1, X_2, ..., X_n), \]
  \[ \text{query}(X_1, X_2, ..., X_n) :- r_2(X_1, X_2, ..., X_n), \]
  
  \[ \text{query}(X_1, X_2, ..., X_n) :- r_1(X_1, X_2, ..., X_n), \text{not} r_2(X_1, X_2, ..., X_n) \]
Recursion

Relation schema manager(employee, manager)

\[
\text{empl-jones}(X) \ :- \ \text{manager}(X, \text{Jones}).
\]

\[
\text{empl-jones}(X) \ :- \ \text{manager}(X, Y), \ \text{empl-jones}(Y).
\]

<table>
<thead>
<tr>
<th>employee-name</th>
<th>manager-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alon</td>
<td>Barinsky</td>
</tr>
<tr>
<td>Barinsky</td>
<td>Estovar</td>
</tr>
<tr>
<td>Corbin</td>
<td>Duarte</td>
</tr>
<tr>
<td>Duarte</td>
<td>Jones</td>
</tr>
<tr>
<td>Estovar</td>
<td>Jones</td>
</tr>
<tr>
<td>Jones</td>
<td>Klinger</td>
</tr>
<tr>
<td>Rensal</td>
<td>Klinger</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration number</th>
<th>Tuples in <strong>empl-jones</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(Duarte), (Estovar)</td>
</tr>
<tr>
<td>2</td>
<td>(Duarte), (Estovar), (Barinsky), (Corbin)</td>
</tr>
<tr>
<td>3</td>
<td>(Duarte), (Estovar), (Barinsky), (Corbin), (Alon)</td>
</tr>
<tr>
<td>4</td>
<td>(Duarte), (Estovar), (Barinsky), (Corbin), (Alon)</td>
</tr>
</tbody>
</table>
Datalog Fixpoint

- The view relations of a recursive program containing a set of rules $\mathcal{R}$ are defined to contain exactly the set of facts $l$ computed by the iterative procedure $Datalog-Fixpoint$

  **procedure** Datalog-Fixpoint
  
  $l = $ set of facts in the database
  
  repeat
  
  $Old_l = l$
  
  $l = l \cup \text{infer}(\mathcal{R}, l)$
  
  until $l = Old_l$

- At the end of the procedure, $\text{infer}(\mathcal{R}, l) \subseteq l$
  
  - $\text{infer}(\mathcal{R}, l) = l$ if we consider the database to be a set of facts that are part of the program

- $l$ is called a fixed point of the program
Semantics of Recursion

• Fixpoint
  – Fixpoint is unique

• Transitive closure of a relation
  – \textit{empl}(X, Y) :\neg \textit{manager}(X, Y).
    \textit{empl}(X, Y) :\neg \textit{manager}(X, Z), \textit{empl}(Z, Y)

• Another way
  – \textit{empl}(X, Y) :\neg \textit{manager}(X, Y).
    \textit{empl}(X, Y) :\neg \textit{empl}(X, Z), \textit{manager}(Z, Y).

• Cannot use negation
The Power of Recursion

- Recursive views make it possible to write queries, such as transitive closure queries, that cannot be written without recursion or iteration.
- Without recursion, a non-recursive non-iterative program can perform only a fixed number of joins.
- Programs satisfy the safety condition will terminate.
  - number(0). number(A) :- number(B), A=B+1.
  - Some programs not satisfying the safety condition do not terminate.
Monotonicity

- A view $V$ is said to be monotonic if given any two sets of facts $I_1$ and $I_2$ such that $I_1 \subseteq I_2$, then $E_V(I_1) \subseteq E_V(I_2)$, where $E_V$ is the expression used to define $V$
- A set of rules $R$ is said to be monotonic if $I_1 \subseteq I_2$ implies $\text{infer}(R, I_1) \subseteq \text{infer}(R, I_2)$,
- Relational algebra views defined using only the operations: $\Pi$, $\sigma$, $\times$, $\cup$, $\cap$, and $\rho$ are monotonic
  - Relational algebra views defined using “$-$” may not be monotonic.
- Datalog programs without negation are monotonic, but Datalog programs with negation may not be monotonic
- Monotonic expressions can use the fixpoint technique
Summary

- Datalog: a prolog-like query language
- Using Datalog to write queries
- Semantics of Datalog programs
To-Do-List

• Examine the example queries in the relational algebra section, which ones can be rewritten in Datalog?