Advanced Data Modeling

Steffen Staab
with
Simon Schenk
Recursion

connected(StartPoint, EndPoint) :-
    arc(StartPoint, EndPoint).

connected(StartPoint, EndPoint) :-
    arc(StartPoint, NextPoint),
    connected(NextPoint, EndPoint).
Recursion

StartPoint and EndPoint are connected

if

StartPoint and EndPoint are connected by an arc

or

there exists an intermediate point NextPoint such that

StartPoint and NextPoint are connected by an arc and

NextPoint and EndPoint are connected.
Integrity constraints

Each class has at least one lecturer:

\[ \forall x \ (\text{class}(x) \rightarrow \exists y \ \text{lecturer}(y, x)) \].

Each class has at most one lecturer:

\[ \forall x (\text{class}(x) \rightarrow \forall y \forall z (\text{lecturer}(y, x) \land \text{lecturer}(z, x) \rightarrow y=z)). \]
Complex values

route(StartPoint, EndPoint, [StartPoint, EndPoint]) :-
    arc(StartPoint, EndPoint).

route(StartPoint, EndPoint, [StartPoint | Route]) :-
    arc(StartPoint, NextPoint),
    route(NextPoint, EndPoint, Route).
Combination of following three features create problems with defining semantics of deductive databases and designing query answering algorithms for them:

- Negation;
- Recursion;
- Complex values.

Restrictions may be required on the use of (combinations of) these features.
SWI Prolog:

http://www.swi-prolog.org/
Soccer database

% EXTENSIONAL DATABASE
% Relation nextLeague describes the hierarchy of leagues in % UK

nextLeague(league2, league1).
nextLeague(league1, championship).
nextLeague(championship, premier).
% the list of clubs
club(arsenal).
club(watford).
club(leedsU).
club(miltonKeynesDons).

% the list of leagues of clubs
league(arsenal, premier).
league(watford, championship).
league(leedsU, league1).
league(miltonKeynesDons, league2).
% the list of players and where they are playing
player(andy, arsenal).
player(wim, watford).
player(liam, leedsU).
player(mike, miltonKeynesDons).
% some players foul other players
foul(andy, wim).
foul(andy, bert).
foul(andy, chris).
foul(andy, dan).
foul(wim, andy).
foul(wim, dan).
% INTENSIONAL DATABASE
% Relation nextLeagues describes the order on leagues
% It is defined as the transitive closure of nextLeague

higherLeague(LowerLeague, HigherLeague) :-
    nextLeague(LowerLeague, HigherLeague).

higherLeague(LowerLeague, HigherLeague) :-
    nextLeague(LowerLeague, MiddleLeague),
    higherLeague(MiddleLeague, HigherLeague),
    higherLeague(MiddleLeague, HigherLeague).

% A higher-leagued club is a club who is in a higher league

higherLeaguedClub(Higher, Lower) :-
  league(Higher, HigherLeague),
  league(Lower, LowerLeague),
  higherLeague(LowerLeague, HigherLeague).
% likes is a relation among players.

% (i) every player likes himself
like(Player, Player) :-
    player(Player).

% (ii) every player likes all players in higher-ranked clubs
like(Lower, Higher) :-
    player(Lower, LowerClub),
    player(Higher, HigherClub),
    higherRankedClub(HigherClub, LowerClub).
% (iii) a player likes a lower-ranked player when
% players of the lower-ranked club
% do not foul other players of his club

likes(Higher, Lower) :-
  player(Higher, HigherClub),
  lower(LowerClub),
  higherRankedClub(HigherClub, LowerClub),
  not hasPlayerWhoFoulsSomePlayerFrom(LowerClub, HigherClub).
% an auxiliary relation: hasPlayerWhoFoulsSomePlayerFrom

hasPlayerWhoFoulsSomePlayerFrom(Club1, Club2) :-
    player(Player1, Club1),
    player(Player2, Club2),
    foul(Player1, Player2).
% INTEGRITY CONSTRAINTS

% every club has a league
\( \forall x (\text{club}(x) \rightarrow \exists y \ league(x, y)) \).

% only premier league player may foul more than one player
\( \forall p, c, z_1, z_2 \)
\[
\text{(player}(p,c) \land \text{foul}(p, z_1) \land \text{foul}(p, z_2) \\
\rightarrow \\
\text{z}_1 = z_2 \lor \ text{league}(c, premier)) .
\]
Relational Data Model
Overview

- Relational data model;
- Tuples and relations;
- Schemas and instances;
- Named vs. unnamed perspective;
- Relational algebra;
<table>
<thead>
<tr>
<th>Player</th>
<th>Birth Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>1980</td>
</tr>
<tr>
<td>Wim</td>
<td>1975</td>
</tr>
<tr>
<td>Liam</td>
<td>1985</td>
</tr>
<tr>
<td>Mike</td>
<td>1988</td>
</tr>
<tr>
<td>Bert</td>
<td>1971</td>
</tr>
</tbody>
</table>
Observations

- The **rows** of the table contain pairs occurring in the relation **player**.
- There are two **columns**, labeled respectively by “name” and “birth year”.
- The **values** in each column belong to different **domains** of possible values.
How to specify a relation

1. specifying the names of the columns (also called fields or attributes);

2. specifying a domain of possible values for each column;

3. enumerate all tuples in the relation.

(1)–(2) refer to the schema of this relation, while (3) to an instance.
Domains and attributes

- A set of **domains** $\mathcal{D}_1, \mathcal{D}_2$ (sets of values);

- A set of corresponding **domain names** $d_1, d_2, \ldots$

- A set of **attributes** $a_1, a_2, \ldots$
Tuple: any finite sequence \((v_1, \ldots, v_n)\).

- \(n\) is the arity of this tuple.

Relation schema:

- \(r(a_1:d_1, \ldots, a_n:d_n)\)

- where \(n \geq 0\), \(r\) is a relation name, \(a_1, \ldots, a_n\) are distinct attributes, \(d_1, \ldots, d_n\) are domain names.

Relation instance:

finite set of tuples \((v_1, \ldots, v_n)\) of arity \(n\) such that \(v_i \in D_i\) for all \(i\).
**Observation**

1. The attributes in each column must be unique.

2. A relation is a **set**. Therefore, when we represent a relation by a table, the order of rows in the table does not matter.

Let us add to this:

3. The order of attributes does not matter.
A tuple is a set of pairs \( \{(a_1, v_1), \ldots, (a_n, v_n)\} \) denoted by \( \{a_1=v_1, \ldots, a_n=v_n\} \).

Let \( d_1, \ldots, d_n \) be domain names and \( D_1, \ldots, D_n \) be the corresponding domains.

The tuple **conforms** to a relation schema \( r(a_1:d_1, \ldots, a_n:d_n) \) if \( v_i \in D_i \) for all \( i \).
Relational data are structured

Note that in the relational data model tuples stored in a table are structured:

- all tuples conform to the same relation schema;
- the values in the same column belong to the same domain.

Untyped perspective: there is a single domain, so the second condition can be dropped.
Typed or untyped?

Consider the relation admire:

<table>
<thead>
<tr>
<th>admirer</th>
<th>admired</th>
</tr>
</thead>
<tbody>
<tr>
<td>wim</td>
<td>andy</td>
</tr>
<tr>
<td>mike</td>
<td>wim</td>
</tr>
<tr>
<td>liam</td>
<td>andy</td>
</tr>
<tr>
<td>liam</td>
<td>arsenal</td>
</tr>
</tbody>
</table>
Database schema and instance


- Relational database instance conforming to a relational database schema $S$:
  - a mapping $I$ from the relation names of $S$ to relation instances such that
    - for every relation schema $r(a_1:d_1, \ldots, a_n:d_n)$ in $S$
      the relation instance $I(r)$ conforms to this relation schema.
No attributes

- a tuple is simply a sequence \((v_1, \ldots, v_n)\) of values.

- The components of tuples can therefore be identified by their position in the tuple.
From Unnamed to Named Perspective

- Introduce a collection of attributes \#1, \#2, ..., 

- Identify tuple \((v_1, \ldots, v_n)\) with the tuple 
  \[ \{ \#1 = v_1, \ldots, \#n = v_n \} \].

- Likewise, identify relation schema \(r(d_1, \ldots, d_n)\) with 
  \(r(\#1:d_1, \ldots, \#n:d_n)\).
Relational Algebra and SQL

1. Can **define** new relations from existing ones;
2. Uses a collection of **operations** on relations to do so.
Constant

\{ (v_{11}, \ldots, v_{1n}), \\
\ldots \\
(v_{k1}, \ldots, v_{kn}) \}
$R_1 \cup R_2 = \{(c_1, \ldots, c_k) \mid (c_1, \ldots, c_k) \in R_1 \text{ or } (c_1, \ldots, c_k) \in R_2\}$
Set difference

\[ R_1 - R_2 = \{ (c_1, \ldots, c_k) \mid (c_1, \ldots, c_k) \in R_1 \text{ and } (c_1, \ldots, c_k) \not\in R_2 \} \]
Cartesian product

\[ R_1 \times R_2 = \{(c_1, \ldots, c_k, d_1, \ldots, d_m) \mid (c_1, \ldots, c_k) \in R_1 \text{ and } (d_1, \ldots, d_m) \in R_2 \}. \]
Let now $R$ be a relation of arity $k$ and $i_1, \ldots, i_m$ be numbers in \{1, \ldots, k\}.

$$\pi_{i_1, \ldots, i_m}(R) = \{(c_{i_1}, \ldots, c_{i_m}) \mid (c_1, \ldots, c_k) \in R\}.$$ 

We say that $\pi_{i_1, \ldots, i_m}(R)$ is obtained from $R$ by \textbf{projection} (on arguments $i_1, \ldots, i_m$).
Assume **formulas on domains** with “variables” #1, #2, ....

For example, #1 = #2.

\[ \sigma_F(R) = \{(c_1, \ldots, c_k) \mid (c_1, \ldots, c_k) \in R \text{ and } F \text{ holds on } (c_1, \ldots, c_k)\} . \]