Advanced Data Modeling

Steffen Staab
Contact:
Prof. Dr. Steffen Staab: staab@uni-koblenz.de
Office hours: Wed, 10.45 Uhr - 11.45 Uhr
Scheduling: via email to hissnauer@uni-koblenz.de

Teaching assistant: Renata Dividino

https://west.uni-koblenz.de/teaching/ss13/advanced-data-modeling
For whom?

Advanced Data Modeling

Addressees:
Master Inf/CV/Web Science
Diplom Hauptstudium Inf/CV

Lecture INSS02 is Part of the „Schwerpunkt“ Data & Knowledge Engineering in the Master’s Programme of Computer Science

Bachelor students may elect advanced courses such as Advanced Data Modeling as „Wahl-/Wahlpflicht“
Examination

- Tutorials:
  - for improving understanding and getting hands on experience
  - Hand in exercises
    - 75% fulfillment: 1 grade level better (e.g. 1.7 -> 1.3)
    - 90% fulfillment: 2 grade levels better (e.g. 2.0 -> 1.3)

- Course is 2 SWS + 2 SWS
  - Compressed to the first 8 weeks of the summer term, i.e.
    - 3 SWS + 3 SWS, only during 8 weeks

- Probably oral, maybe written:
  - Around end of June.
  - Your own responsibility to schedule an appointment via Mrs Hissnauer
  - No later first examinations in Advanced Data Modeling
  - no admission criteria, but do not expect to pass if you did not do the assignments
Assumption

- Did you participate in Introduction to Databases?
- Did you participate in the Logics course?
Structure of the lecture

- Logics for Data Engineering
- Relational data model;
- Deductive data model;
- Recursive definitions and their semantics
Why Logics for Data Engineering?

- Many current applications:
  - Hidden part of advanced databases like Oracle, DB2,…
  - Policy languages / Access control
  - Ontologies & Semantic Web
  - Software engineering: OCL

- Why?
  - Generalization of other data models
    - Relational
    - Semi-structured (RDF, XML, OEM,…)

- Well-understood theory
  - Yet still evolving….!
Deductive Databases

- evolved during the 1980s, based on the ideas developed in relational databases and logic programming.

- developed with the aim of increasing the expressive power of relational query languages, and in particular in connection with the inability of the latter to express recursive queries.
Query languages

- navigational (early DBMS);
- declarative (relational DBMS).
Why logics?

- Logic tried to solve problems similar to those arising in foundations of databases:
  - how to formalize the application world (language);
  - How to express its properties (semantics, model theory);
  - How to reason about these properties (proof theory).
Why logics?

- Logic can handle in a **uniform framework**
- recursive definitions;
- integrity constraint;
- deduction, induction and abduction;
- Models for complex values . . .
Informal overview of deductive databases

- Extensionally defined relations.
- Intensionally defined relations.
- Integrity constraints.
- Recursion.
- Complex values.
- Negation
Extensionally defined relations

- **Extensional** definition: by explicit enumeration of all tuples in the relation.

- ("Maier", "Mozartstrasse", 678);
- ...
- ("Schmidt", "Raiffeisenstrasse", 857);
- ...

...
Extensionally defined relation

- In deductive databases we use the language of first-order logic and represent this relation by a set of **facts**:
  
  - `entry("Maier", "Mozartstrasse", 678);`
  - `...`
  - `entry("Schmidt", "Raiffeisenstrasse", 857);`
  - `...`
Extensional database

The extensional database defines relations by sets of facts, for example

hasHighestDegree("Maier", BSc);
hasHighestDegree("Schmidt", MSc);
...
higherDegree(MSc, BSc);
...

Analogue of tables in relational databases.
Suppose we want to define a relation `personWithHigherDegree` among persons:

Person $A$ has higher degree than person $B$ if the highest degree of $A$ is higher than the highest degree of $B$. 
Intensionally defined relations. Rules

Extensional definition

personWithHigherDegree("Schmidt","Maier").
personWithHigherDegree("Maier","Kunz").
...

is dangerous

(too large, may become inconsistent after updates).
For each pair of people $A$, $B$, $A$ has higher degree than $B$ if the highest degree of $A$ is $DA$ and the highest degree of $B$ is $DB$ and $DA$ is a higher degree than $DB$. 
personWithHigherDegree(A, B) :-
  hasHighestDegree(A, DA),
  hasHighestDegree(B, DB),
  higherDegree(DA, DB).

% head of the clause

% body

% of the

% clause
SELECT
    D1.person, D2.person
FROM
    hasHighestDegree D1,
    hasHighestDegree D2,
    higherDegree
WHERE
    D1.degree = higherDegree.higher AND
    D2.degree = higherDegree.lower
The relation `personWithHigherDegree` holds between objects $A$, $B$ if
the relation `hasHighestDegree` holds between objects $A$, $DA$
and
the relation `asHighestDegree` holds between objects $B$, $DB$
and
the relation `higherDegree` holds between objects $RA, RB$. 
Variables

For all objects A, B, DA, DB
the relation personWithHigherDegree holds between objects A, B
if
the relation hasHighestDegree holds between objects A, DA
and
the relation asHighestDegree holds between objects B, DB
and
the relation higherDegree holds between objects RA, RB.
Constants

subordinate(O, president) :- officer(O).

Here O is a variable, while president is a constant.

How to say this syntactically?

Different conventions:

- Possibility 1: All variables are explicitly quantified
- Possibility 2: Variables are implicitly quantified
  (universally or existentially – needs to be agreed by convention)

Sets of variables and constants are defined as such
**Disjunction**

How to express *every human is either a woman or a man*?

```prolog
human(A) :- man(A).

human(A) :- woman(A).
```
How to express that every doctor has the same qualification as Doctor No, with the exception of Doctor No himself?

\[
\text{sameAs}(A,A) :- \text{Object}(A).
\]

\[
\text{sameQualification}(A,B) :- \begin{align*}
\text{hasHighestDegree}(A, D), \\
\text{hasHighestDegree}(B, D), \\
\text{notSameAs}(A,B).
\end{align*}
\]

\[
\text{hasHighestDegree}(\text{DrNo}, \text{PhD}).
\]
Negation

Use negation:

\[
\text{sameQualification}(A,B) :- \\
\quad \text{hasHighestDegree}(A, D), \\
\quad \text{hasHighestDegree}(B, D), \\
\quad \textbf{not} \; \text{SameAs}(A,B).
\]

Negation is handled using the \textit{closed world assumption}.
Goals

:- likes(X, Y), not likes(Y, X).

:- sameQualification(drNo, Y).
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).
Combinations 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/
% 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).
A higher-leagued club is a club who is in a higher league

```prolog
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),
  player(Lower, LowerClub),
  higherRanknedClub(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 \text{ league}(x, y)).

% only premier league player may foul more than one player
\forall p, c, z1, z2
(\text{player}(p,c) \land \text{foul}(p, z1) \land \text{foul}(p, z2) \\
\rightarrow \\
\quad z1 = z2 \lor \text{league}(c, \text{premier})).