Advanced Data Modeling

Steffen Staab
Organizational Issues

Wed April 22, 14-16, Lecture 1
Tue April 28, 8-10, Lecture 2
Wed April 29, 14-16, Lecture 3
Tue May 5, 8-10, Exercise 1 (Dividino)
Wed May 6, 14-16, Lecture 4
Tue May 12, 8-10, Lecture 5
Wed May 13, 14-16, Exercise 2 (Dividino)
Tue May 19, 8-10, Exercise 3 (Bercovici)
Wed May 20, 14-16, Exercise 4 (Bercovici)
Tue May 26, 8-10, Lecture 6
Wed May 27, 16-18, Lecture 7, Prof. Weikum
Tue June 2, 8-10, Exercise 5
Wed June 3, 14-16, Lecture 8
Tue June 9, 8-10, Exercise 6

Wed June 10, 14-16, Exercise 7
Tue June 16, 8-10, Lecture 9
Wed June 17, 14-16, Exercise 8
Tue June 23, 8-10, Lecture 10
Wed June 24, 14-16, Lecture 11
Tue June 30, 8-10 Exercise 9
Wed July 1, 14-16 Cancelled
Tue July 7, 8-10 Exercise 10
Wed July 8, 14-16 Lecture 12
Tue July 14, 8-10 Exercise 11
Wed July 15, 14-16 Lecture 13
Tue July 21, 8-10 Exercise 12
Wed July 22, 14-16, Questions

Tue July 28, 9-10, Exam
Organizational Issues

Web Page:
http://isweb.uni-koblenz.de/Teaching/SS09/adm09

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

Also eligible as Wahl- / Wahlpflicht in the Bachelor/Master
Examinations

Assignments and exercises
• bonus points

Exam at the end of the term
• may be written or oral
• no admission criteria, but do not expect to pass if you did not do the assignments
Assumption

- You have successfully participated in Databases 1
- 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

Management of Distributed Data / Cloud Computing
- Hadoop Processing Model
- HBase
- Distributed Joins

Possibly:
- Provenance Queries
Why Logics for Data Engineering?

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

Why?
- Generalization of other data models
  - Relational
  - Semi-structured (RDF, XML, OEM,…)
- Well-understood theory
  - Yet still evolving….!
Why Distributed Data Engineering

Massive amounts of data
- E.g. Google stores WWW on $10^5$-$10^6$ computers
- Sensors produce more data than can be stored

Data is distributed
- Organizations spread all over the world
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

- Extentionally defined relations.
- Intentionally defined relations.
- Integrity constraints.
- Recursion.
- Complex values.
Extensionally defined relations

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

(”Maier”, ”Mozartstrasse”, 678);

…

(”Schmidt”, ”Raiffeisenstrasse”, 857);

…
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);

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

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

Analogue of **tables** in relational databases.
Suppose we want to define a relation \texttt{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 $DB$. 
Clause (rule)

personWithHigherDegree(A,B) :- % head of the clause

hasHighestDegree(A,DA),
hasHighestDegree(B,DB),
higherDegree(DA,DB). % 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
Relations

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 $RA$, $RB$. 
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
How to express every human is either a woman or a man?

human(A) :-
  man(A).

human(A) :-
  woman(A).
Negation

How to express that every doctor has the same qualification as Doctor No, with the exception of Doctor No himself?

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

\[
\text{sameQualification}(A,B) :-
\quad \text{hasHighestDegree}(A, D),
\quad \text{hasHighestDegree}(B, D),
\quad \text{notSameAs}(A,B).
\]

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

Use negation:

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

Negation is handled using the closed world assumption.
Until here: lecture 1

From here: lecture 2
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).
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:

∀x (class(x) → ∃y lecturer(y, x)).

Each class has at most one lecturer:

∀x(class(x) → ∀y ∀z(lecturer(y, x) ∧ lecturer(z, x) → y=z)).
Complex values

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

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


Problems

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/
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),
    player(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 \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 
    z1 = z2 \lor \text{league}(c, \text{premier})).