Retracting Assumptions

- Data are always explicitly given

  → deductive data-bases

- Data are always well-structured

  → semi-structured or unstructured data

- Data have to be administered centrally

  → distributed systems, semantic web

- Every data base has to be in normal form

  → data warehousing

- The user knows exactly which information he is in need of

  → data mining

- Data can be indexed along a single dimension

  → Index Structures for Similarity Queries

Database and Information Systems

Part II

Deductive Databases

Readings:

- Ceri, Stefano; Gottlob, Georg; Tanca, Leticia: Logic Programming and Databases, Springer-Verlag, Berlin 1990.
- Kemper, Alfons; Eickler Andre: Datenbanksysteme: Eine Einführung, Oldenbourg, München 2006, Kapitel 15.
- Elmasri, Ramez; Navathe, Shamkant B.: Fundamentals of Database Systems, Addison-Wesley, Boston etc. 2011, Chapter 26.
- De Moor, Oege; Gottlob, Georg; Furche, Tim; Sellers, Andrew (eds.): Datalog Reloaded. Proc. 1st Int. Workshop Datalog 2010. Springer-Verlag, Heidelberg u.a., 2011.

Deductive Databases

- Deduction
- Deductive Databases
- Derivation in Deductive Databases
- Extensions to Pure Datalog
- Comparison with Prolog
- Integrity Constraints
- Recursion in SQL
- Applications of Datalog

Deduction

- requirements for relational databases:
  - data independence $\rightarrow$ declarative specification of data
  - avoidance of redundancy $\rightarrow$ normalization
- many information tasks require the derivation of data from other data
  - e.g. data transformations: date of birth $\rightarrow$ age
  - e.g. data combination: high income $\wedge$ low age $\rightarrow$ interesting customer
  - e.g. transitive closure: time table enquiries

Deduction

- transformations, data combination $\rightarrow$ complex queries, views
- views of views?
- recursive views?
- usually: computation in separate application procedures
- drawbacks
  - application specific solutions
  - danger of inefficient solutions
  - separate administration of data and programs
  - impedance mismatch: declarative vs. imperative specifications

Deductive Databases

- deductive databases
  - extensional database, facts
  - intensional database, rules
  - consistency constraints

Deductive Databases

- extensional database

```
parent_of(mary,ellen).
parent_of(ellen,john).
parent_of(mary,dan).
parent_of(ellen,ann).
male(john).
male(dan).
female(mary).
female(ellen).
female(ann).
```

- corresponds to a relational database

Deductive Databases

- intensional database

```
mother_of(X,Y) :- parent_of(X,Y),female(X).
father_of(X,Y) :- parent_of(X,Y),male(X).
grandmother(X,Y) :- mother_of(X,Z),parent_of(Z,Y).
grandfather(X,Y) :- father_of(X,Z),parent_of(Z,Y).
ancestor_of(X,Y) :- parent_of(X,Y).
ancestor_of(X,Y) :- ancestor_of(X,Z),
                    ancestor_of(Z,Y).
```

- rules allow the system to derive facts from other facts (deduction)

Deductive Databases

- original facts

```
parent_of(mary,ellen).
parent_of(ellen,john).
parent_of(mary,dan).
parent_of(ellen,ann).
female(mary).
female(ellen).
```

- rule

```
mother_of(X,Y) :- parent_of(X,Y),female(X).
```

- derived facts

```
mother_of(mary,ellen).
mother_of(ellen,john).
mother_of(mary,dan).
mother_of(ellen,ann).
```

Deductive Databases

- pure datalog
- horn clauses: simplest form of 1st order predicate logic formulae
  ⟨clause⟩ := ⟨fact⟩ | ⟨rule⟩ | ⟨goal⟩
  ⟨rule⟩ := ⟨head⟩ {':-' ⟨body⟩ } '.'
  ⟨head⟩ := ⟨literal⟩
  ⟨body⟩ := ⟨literal⟩ {',' ⟨literal⟩ }
  ⟨literal⟩ := ⟨functor⟩ '(' ⟨argument⟩ {',' ⟨argument⟩} ')'
  ⟨functor⟩ := ⟨atom⟩
  ⟨argument⟩ := ⟨variable⟩ | ⟨atom⟩ | ⟨number⟩ | ⟨string⟩
  ⟨variable⟩ := ⟨upper case character⟩ {character}
  ⟨atom⟩ := ⟨lower case character⟩ {character}
  ⟨fact⟩ := ⟨head⟩ '.'
  ⟨goal⟩ := ⟨body⟩ '.'
- facts are rules with an empty body (unconditionally valid assertions)

Deductive Databases

- a predicate definition corresponds to a view in a relational db
- Datalog *programs* can be translated into relational algebra

```
mother_of(X,Y) :- parent_of(X,Y),female(X).
```

$$\Longrightarrow \quad \pi_{1,2}(PARENT \underset{1=1}{\bowtie} FEMALE)$$

$\pi_{1,2}($

PARENT	
1	2
mary	ellen
ellen	john
...	...

$\underset{1=1}{\bowtie}$

FEMALE
1
mary
ellen
...

$) = \pi_{1,2}$

1	2	3
mary	ellen	mary
ellen	john	ellen
...	...	...

$=$

MOTHER	
1	2
mary	ellen
ellen	john
...	...

```
grandmother_of(X,Y) :- mother_of(X,Z),
                       parent_of(Z,Y).
```

$$\Longrightarrow \quad \pi_{1,4}(MOTHER \underset{2=1}{\bowtie} PARENT)$$

$$\Longrightarrow \quad \pi_{1,4}((\pi_{1,2}(PARENT \underset{1=1}{\bowtie} FEMALE)) \underset{2=1}{\bowtie} PARENT)$$

Deductive Databases

- goals are queries to

$$\text{extensional database} \cup \text{intensional database}$$

- Datalog *goals* can be translated into relational algebra

```
?- mother_of(X,Y).
```

$$\Longrightarrow \quad \mathit{MOTHER}$$

$$\Longrightarrow \quad \pi_{1,2}(\mathit{PARENT} \underset{1=1}{\bowtie} \mathit{FEMALE})$$

- special case: constant selection

```
?- mother_of(mary,X).
```

$$\Longrightarrow \quad \sigma_{1=\text{mary}} \mathit{MOTHER}$$

$$\Longrightarrow \quad \sigma_{1=\text{mary}}(\pi_{1,2}(\mathit{PARENT} \underset{1=1}{\bowtie} \mathit{FEMALE}))$$

Deductive Databases

- rules can be recursive

```
ancestor_of(X,Y) :- parent_of(X,Y).
ancestor_of(X,Y) :- ancestor_of(X,Z),
                    ancestor_of(Z,Y).
```

- recursive rules can cause termination problems
  → additional safety conditions needed
  - facts must not contain any variables
  - each variable which occurs in the head of a rule must also occur in the body of the same rule

Derivation in Deductive Databases

- Datalog comes with a fully declarative semantics
  $\rightarrow$ results are insensitive to the derivation strategy
- top-down derivation (a la Prolog):
  - problem generators
  - goals are seen as problems to be solved
  - a rule generates simpler problems by decomposing more complex ones
  - problem: single solutions instead of answer sets (impedance mismatch)

Derivation in Deductive Databases

- alternative: bottom-up derivation:
  - productions
  - generating all the consequences of a rule until no more facts can be derived (fixpoint)

- Algorithm:
  - F set of initial facts, R set of rules, $F' = \emptyset$
  - repeat until $F = F'$
    - $F' \leftarrow F$
    - $\forall$ rules $r \in R$: $F \leftarrow F \cup cons(r, F)$

- intermediate results can be stored in the extensional database $\rightarrow$ materialization

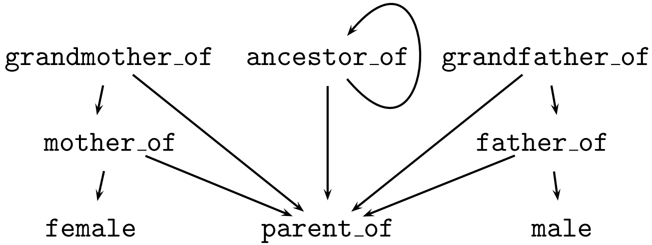
Derivation in Deductive Databases

- naive bottom-up: apply the rules to original and derived facts
- semi-naive bottom-up: consider derivations only if newly derived facts are involved

- serious efficiency problems
  - generating the same derivations over and over again
    $\rightarrow$ dependency analysis
  - considering completely irrelevant derivations
    $\rightarrow$ magic sets

Derivation in Deductive Databases

- bottom up derivation: which clauses to consider
  $\rightarrow$ dependency graph
- A literal X depends on a literal Y if Y occurs as a subgoal of a clause with literal X as head.
- dependencies can be represented as edges in a directed graph



- if the dependency graph contains cycles $\rightarrow$ recursive program

Derivation in Deductive Databases

- transformational approaches: magic sets
- problem with bottom up derivation:
  - generates the whole relation
  - ignoring constraining information possibly provided with the goal (e.g. constant selection)
  - idea: adding additional constraints to the original program to force it to consider the variable bindings imposed by the goal

Derivation in Deductive Databases

- transformation of the program

```
grandmother_of(X,Y) :- mother_of(X,Z),
                       parent_of(Z,Y).
```

- for the variable bindings of the goal

```
?- grandmother_of(X,dan).
```

- into a derived program

```
grandmother_of(X,Y) :- magic(Z),
                       mother_of(X,Z),
                       parent_of(Z,Y).


magic(dan).
magic(X) :- magic(Y), parent_of(X,Y).
```

- bottom-up derivation of a transformed program may produce more results than specified by the goal
  → final filtering may be necessary

Extending Pure Datalog

- built-in comparison predicates
- negation
- complex objects

Comparison Predicates

- not critical: $=$
- critical: $\neq, <, >, \leq, \geq$
  - correspond to infinite relations
  - can compromise the safety of a Datalog program

```
sister_of(X,Y) :- parent_of(Z,X),
                  parent_of(Z,Y),
                  female(X),
                  X \= Y.

adult(X) :- person(X),
            age(X,Y),
            Y>17.
```

Comparison Predicates

- extended safety conditions:
  every variable in the head of a clause ...
  - ... has to also occur in a non-built-in predicate in the body of the clause or ...
  - ... is unified with a constant or a variable for which safety has been shown already
- evaluation of the predicate needs to be deferred until all its arguments are bound.

Negation

- negation by means of a

closed world assumption (CWA)

If a fact does not logically follow from a set of clauses then we can conclude that the negation of the fact is true

- pure Datalog + CWA allows to deduce negative facts
- but deduced negative facts can not be used to derive further facts

Negation

- extension with negated literals in the body of a clause necessary

```
% marriage(Man,Woman,Date). divorce(Man,Woman,Date).
marriage(john,eve,'1965.03.12').
marriage(paul,jane,'1989.11.04').
divorce(paul,jane,'1990.02.17').

unmarried(X) :- person(X), not(marriage(X,_,_)),
                not(marriage(_,X,_)).

married(X,Y) :- person(X), marriage(X,Y,D1),
                not(and(divorce(X,Y,D2),D1<D2)).
married(X,Y) :- person(X), marriage(Y,X,D1),
                not(and(divorce(Y,X,D2),D1<D2)).
```

Negation

- examples cont.

```
divorced(X) :- person(X), divorce(X,Y,D1),
               not(married(X,_)).
divorced(X) :- person(X), divorce(Y,X,D1),
               not(married(X,_)).

widowed(X) :- person(X), married(X,Y), dead(Y).
widowed(X) :- person(X), married(Y,X), dead(Y).
```

Negation

- safety constraint: every variable which occurs in a negated literal must also occur in a non-negated one
- a negated subgoal must not depend on the head of the clause
  $\rightarrow$ stratified Datalog, stratified programs:
  - evaluate the predicate under the negation symbol
  - if not true assume the negation to be true
- a stratified evaluation order can be generated by means of an extended dependency tree

Complex Objects

- representation as function symbols of a 1st order logic and sets

```
person(name(ken,smith),
       birthdate(1976,may,22),
       children(\{ann,dan,susan\}))
```

- complex objects may compromise the safety
  - undecidable whether a program has finitely or infinitely many results
  - finiteness of sets is undecidable
- self-referential set definitions (sets which include themselves) have no well-defined semantics

Comparison with Prolog

- syntactically Datalog is a subset of Prolog
- every Datalog clause is a valid Prolog clause
- differences in the semantics

Datalog	Prolog
fully declarative semantics	procedural elements
many equivalent derivation strategies	fixed derivation strategy
termination guaranteed	termination depends on the order of clauses and subgoals
safety constraints	full Horn logic
set oriented derivation	fact oriented derivation

Integrity Constraints

- integrity constraints have the general form

```
false :- not(condition).
```

- cannot be used to derive new facts
- have to be fulfilled after every update (static integrity constraints)

```
false :- marriage(X,_,_), not(male(X)).
false :- marriage(_,X,_), not(female(X)).
false :- age(X,Y), X>150.
false :- marriage(X,Y,_),
         first_grade_relatives(X,Y).
```

- integrity constraints can be inconsistent
  → no valid database content is possible
  → satisfiability checks required

Recursion in SQL

- restricted form of recursion is part of SQL-99
- `with`-clause defines a table to be used in another query
- `with recursive` makes recursive self-reference possible

```
with recursive ancestor as
  (select * from parent
    union
   select parent.parent, ancestor.successor
     from parent, ancestor
     where parent.child = ancestor.ancestor)
```

- no semantic means to ensure termination

Recursion in SQL

- safety has to be achieved by the programmer by controlling
  - processing order

    ```
    search depth/breadth first ... set
    ```

  - maximum recursive depth
  - cycle markup

    ```
    cycle Attribute set Cycle_Mark_Attribute
      to Marke using Path_Attribute
    ```

    - no cycle detection
    - has to be programmed individually based on the markup provided by the system
- SQL allows unrestricted negation, scalar functions and aggregation and is therefore inherently unsafe!
- individual cycle monitoring is highly error-prone

Applications of Datalog

- increasing relevance for the design of (domain-specific) declarative languages (Datalog 2.0, Datalog relaunched, ...)

- e.g. BOOM (U Berkeley):

  Building order-of-magnitudes larger systems ...

  ... with order-of-magnitudes less code

- reduction of coding effort in massively parallel environments
  - exploiting data parallelism
  - facilitated by the absence of side effects
  - applications in distributed systems and cloud computing

- highly scalable, efficient implementations on modern (parallel) computer architectures (GPU, FPGA, cloud computing, ...)

Applications of Datalog

- rapid development of integrated enterprise applications
  - decision automation
  - analytics
  - planning
- vision: the enterprise version of spreadsheets

	book keeping	decision making
desktop	Access	Excel
enterprise	Oracle, DB2, etc.	?

adapted from Aref (2011), LogicBlox

Applications of Datalog

- networking and distributed systems
  - reasoning over web data, context-aware quering
  - decentralized social networking
  - information extraction (query and restructuring)
- computer games and visualizations
- machine learning and robotics
- compilers
- security protocols