Chapter 8
Relational Database Languages: Relational Calculus

Overview

- the relational calculus is a specialization of first-order logic, tailored to relational databases.
- straightforward: the only structuring means of relational databases are relations – each relation can be seen as an interpretation of a predicate.
- there exists a **declarative** semantics.

Relational Calculus vs FOL

- FOL allows for reasoning, based on a *model theory*,
- the relational calculus does not require model theory,
- it is only concerned with *validity* of a formula in a given, fixed model (the database state).

8.1 Bridge Section: Motivation and Preparation for the "Deductive Databases" Lecture

- The lecture "Database Theory" or "Deductive Databases" (MSc or advanced BSc) builds upon the "Introduction to Databases" lecture and requires knowledge about First-Order Logic (e.g., courses "Formal Systems" or "Artificial Intelligence")
- for a diagram with the database concepts, notions and buzzwords related to the DBIS lectures, see `https://www.dbis.informatik.uni-goettingen.de/Teaching/dbnotions.pdf`
- This section summarizes that knowledge and motivates the main idea of the lecture.
- a database can be seen as a purely relational FOL structure
  - predicate symbols of different arities,
  - only 0-ary functions = constants
    - in relational DB: these are the literals (numbers, strings, dates ...)
    - in object-relational DB: also object identifiers
    - in RDF: also URIs, which basically serve as object identifiers

Computer Science: Theoretical and Practical Aspects – illustrated with Deductive Databases

CS in Practice: | Theoretical CS, maths, etc.:

Databases

Software Engineering, Programming languages, ...

Formal Methods: formalization of commonsense reasoning

- modeling,
- symbolic reasoning,
- deduction/calculi,
- parsing, grammars, semantics, ...

metalevel: properties of the FMs

application-specific formal methods:

- relational algebra → SQL
- normalization theory → SQL
- transaction theory

- polynomial
- equalities, complexity
- $\rightarrow$ optimization

SQL

phil. & math. logic in general → first-order logic

- first-order logic → relational calculus
- undecidable

- relational calculus → SQL
- polynomial

- Datalog: queries+rules deductive DB*
- polynomial,
- up to exponential

* I did not place deductive DB to completely practical CS

Relational Algebra, First-Order Logic, Relational Calculus, Datalog

Relational Model
named tables
n named columns $(n \geq 1)$

First-Order Logic
n-ary predicates and functions
$p(...)/n, f(...)/n$ $(n \geq 0)$
$\mathcal{S} \models \psi$ — $\mathcal{S}$ satisfies ψ
$\varphi \models \psi$ — φ entails ψ

Relational Algebra
n-tuples with named columns
$\pi, \sigma, \rho, \bowtie, \setminus, \cup, \cap$

$\equiv$

(slotted) Relational Calculus
slotted n-records $(n \geq 1)$
constants (data values)
dot-syntax (e.g. country.name)
$\wedge, \vee, \neg, \exists, \forall$ over tuples
$\mathcal{S} \models \psi$

$\equiv$

(positional) Relational Calculus
positional n-tuples/ n-ary predicates $(n \geq 1)$
constants (data values)
$\wedge, \vee, \neg, \exists, \forall$ over values
$\mathcal{S} \models \psi$, $(\mathcal{S}, \beta) \models \psi$

First-Order Logic → (positional) Relational Calculus: $\subseteq$

(positional) Relational Calculus → (slotted) Relational Calculus: $\subsetneq$

Relational Algebra → SQL: $\pi, \sigma, \rho, \bowtie, \setminus, \cup, \cap$

(slotted) Relational Calculus → SQL: tuple vars

(positional) Relational Calculus → SQL: $\exists, \neg\exists$

(positional) Relational Calculus → Datalog: $\wedge, \rightarrow, \exists, \neg\exists$, free vars

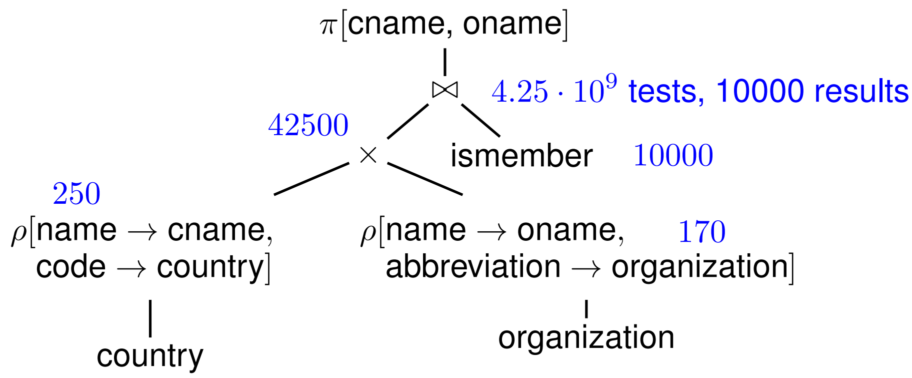
SQL
tables with named columns
$\pi, \sigma, \rho, \bowtie, \setminus, \cup, \cap,$ $\exists$

Datalog
positional n-ary predicates $(n \geq 1)$
Herbrand-style
queries $\quad ? - body$
logical rules $\quad head \leftarrow body$
allows recursion/transitive closure

Declarative Querying & Algebraic Semantics

Query: all pairs *country* (name) and *organization* (name) such that the country is a member of the organization.

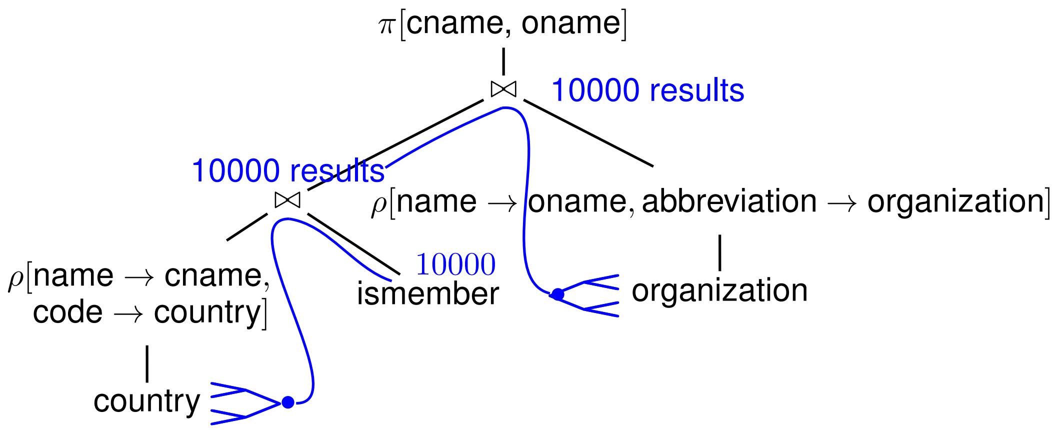
```
SELECT c.name, o.name
FROM country c, organization o
WHERE (c.code, o.abbreviation) IN (SELECT country, organization
                                   FROM ismember)
```



- declarative query in SQL and as algebra tree (bottom-up inductive semantics)
- actual naive evaluation would be inefficient.

Declarative Querying & Algebraic Semantics

- algebraically equivalent rewriting of the tree,
- efficient evaluation using internal algorithms (more efficient, but correct wrt. the set-oriented algebraic semantics of the operators) and indexes (physical layer):
- start with ismember, search ismember.country→country.code primary key index, then join results.organization→organization.abbreviation primary key index



Relational Calculus: Logic-Based Declarative Querying

- positional matching of predicate patterns:

$$q(pop) \equiv \quad \exists\, cc, cap, capprov, area, \text{country}(\text{“Germany”}, cc, cap, capprov, pop, area).$$

$$q(cn, on) \equiv \quad \exists\, cc, cap, capprov, cpop, ca, abbrev, hq, hqc, hqprov, est, type :$$
$$\text{country}(cn, cc, cap, capprov, cpop, ca) \wedge$$
$$\text{organization}(abbrev, on, hq, hqc, hqprov, est) \wedge$$
$$\text{ismember}(cc, abbrev, type)$$

- purely declarative
- “conjunctive query”, translatable to relational algebra SPJR-query (selection-projection-renaming-join)
- free variables (here, cn, on) create the result tuples,

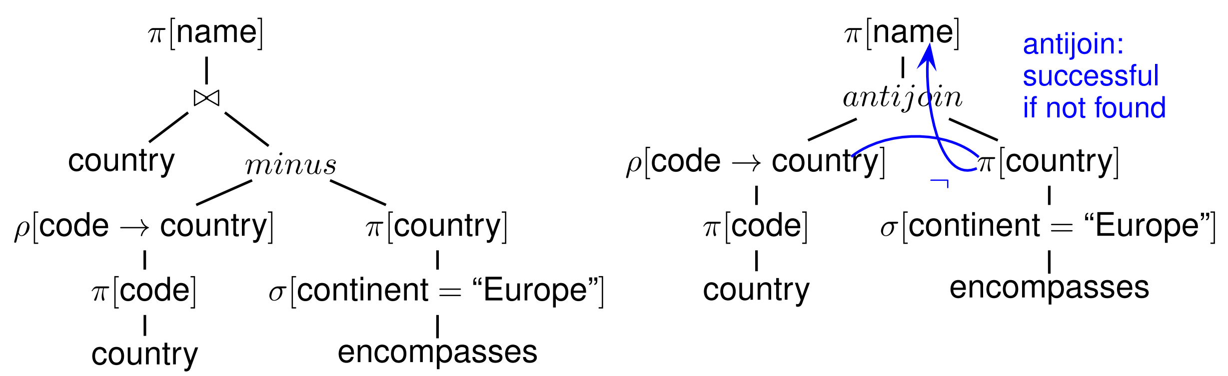
$$answer = \{\ \{cn/\text{“Germany”}, on/\text{“Europ.Union”}\}, \{cn/\text{“Germany”}, on/\text{“North.Atl.Tr.Org”}\}, \ldots,$$
$$\{cn/\text{“France”}, on/\text{“Europ.Union”}\}, \{cn/\text{“France”}, on/\text{“North.Atl.Tr.Org”}\}, \ldots,$$
$$\vdots$$
$$\}$$

Logic-Based Declarative Querying: Negation – not exists

Query: all (names of) countries that are not located in Europe:

```
SELECT c.name
FROM country c
WHERE NOT EXISTS (SELECT *
                  FROM encompasses
                  WHERE e.continent='Europe'
                    AND e.country = c.code)
```

$$q(cn) \equiv \quad \exists\, cc, cap, capprov, cpop, ca$$
$$\text{country}(cn, cc, cap, capprov, cpop, ca) \wedge \neg\exists perc : \text{encompasses}(cc, \text{“Europe”}, p)$$



Closed-World-Assumption: Negation – not exists

- In databases, all tuples that are not there are implicit *negative knowledge*
- query from previous slide:
  "all countries *such that there is no tuple in the the database* that states that the country would be located in Europe"

⇒ "Negation by default"

⇒ consistent with the assumption that the database contains complete knowledge.

- as a first-order/predicate logic interpretation, for all answer bindings β (that bind the variable cn),

$$(\mathcal{S}, \beta) \models \exists\, cc, cap, capprov, cpop, ca : \text{country}(cn, cc, cap, capprov, cpop, ca) \wedge \neg\exists perc : \text{encompasses}(cc, \text{“Europe”}, p)$$

- let φ the conjunction of all facts (=atoms) that are true in the database,

$$\varphi \not\models \exists\, cc, cap, capprov, cpop, ca : \text{country}(cn, cc, cap, capprov, cpop, ca) \wedge \neg\exists perc : \text{encompasses}(cc, \text{“Europe”}, p)$$

  since $\neg\exists perc : \text{encompasses}(cc, \text{“Europe”}, p)$ *cannot logically be concluded* ("Open World")

Negation: Safety of Variables

Consider just a binary *isMember* relationship for mondial without the membership type:

$$q(c) \equiv \neg\text{ismember}(c, \text{“EU”})$$

- what are the answers?
- "USA", "AUS", ..., but also "Moscow", "Berlin", 356000, 3.1415 etc., infinitely many, for which the tuple is not true.

⇒ depends on the considered *domain*.

⇒ every query must be *safe*, i.e., the variables must have a positive occurrence that restricts the possible values:

$$q'(c) \equiv \exists\, name, c, cap, capprov, cpop, ca : \text{country}(name, c, cap, capprov, cpop, ca) \wedge \neg\text{ismember}(c, \text{“EU”})$$

Rule-Based Languages

$$head \leftarrow body$$

- SQL: body = FROM ... WHERE ...,
  head = SELECT ..., DELETE
  similar: MODIFY <relname> WHERE ..., INSERT INTO ... (SFW ...)
- SQL views: derive new tuple(s) when body is satisfied
- An SQL view must not be recursive (i.e., contain itself in the "body" part)

Datalog: Queries and Logical Rules

```
?- country(N, _C, _Cap, _CapProv, _Pop, _Area), not isMember(_C, 'EU', _).
```

Two rules that together compute for each river, to which sea its water finally flows:

```
:- include(mondial).
tc(N,S) :- river(N,R,L,S,_,_,_,_,_,_,_,_,_), not (S = null).
tc(N,S) :- river(N,R,L,S2,_,_,_,_,_,_,_,_,_), not (R = null), tc(R,S).
```

[Filename: Datalog/tcRivers.P]

- Declarative "fixpoint" semantics: apply rules bottom-up as long as possible.

The Universal Quantifier in Query Languages

- SQL: `EXISTS/NOT EXISTS` has been integrated into the SQL syntax (implemented via Join, Minus, Anti-Join)
- The universal quantifier must be rewritten as `NOT EXISTS ... WHERE NOT EXISTS ...`
- the relational calculus obviously allows it:

$$q(cn) \equiv \exists\, cc, cap, capprov, pop, area : (\text{country}(cn, cc, cap, capprov, pop, area) \wedge \forall n, prov, cpoplat, long, el : (\text{city}(n, cc, prov, cpop, lat, long, el) \rightarrow cpop > 1000000))$$

- Datalog: universal quantifier must be encoded into rules
- XQuery (query language for XML data) has it:

```
//country[.//city/population
          and
          (every $cp in .//city/population satisfies $cp > 1000000)]/name
```

- note: null values and missing values (in XML) have been ignored here.

TYPES OF KNOWLEDGE

- (positive) atomic facts:
  - DB: tuples in an n-column table of the database
  - FOL: $\mathcal{S} = (I, \mathcal{D})$: for an n-ary predicate, $I(p) \subseteq \mathcal{D}^n$
  - atoms in a formula

  $\Rightarrow$ conjunctions/sets of atomic facts
- negative atomic facts/knowledge:
  - rather “implicit”: the n-tuples “not there” in a DB or not in $I(p)$.

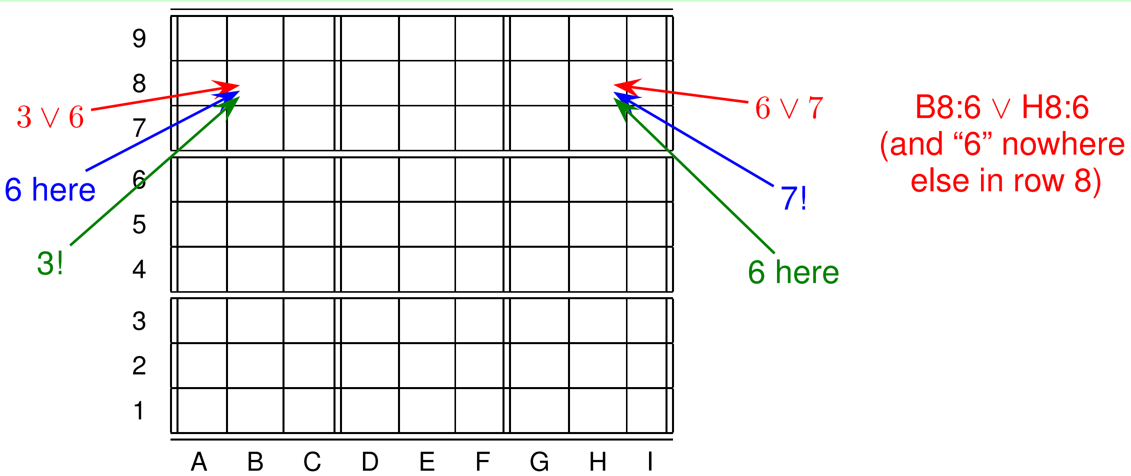
  $\Rightarrow$ queries under CWA and $\mathcal{S} \models \varphi$.
- atomic positive conclusions: INSERT into DB, Views
- atomic negative conclusions: DELETE, or inconsistencies

Disjunctive Knowledge

- “$p(x)$ or $q(y)$ does hold”
- cannot be represented by a database or a single FOL interpretation, only by formulas

$\Rightarrow$ conclusions in “knowledge base”

Disjunctive Knowledge in Human Reasoning: Sudoku



$\Rightarrow$ B8:3 ∨ H8:7

Existential Knowledge

- "every country has some city that is its capital (and which is located in this country)"
  $\forall x$:*country*(x)$\rightarrow$ $\exists y$: (*city*(y) $\wedge$ *hasCapital*(x, y) $\wedge$ *located_in*(y, x))
  - SQL: *country.capital* not null and a foreign-key-to-primary-key reference:
    country.(code, capital, capprov) references city.(country, name, province)
    only as a passive constraint, cannot conclude and insert the city (name is not known)
  - ER-Diagram: minCardinality for *capital*, but not that *isCapital* $\subseteq$ *locatedIn*
  - OWL/Description Logic: *Country* $\sqsubseteq$ $\exists$ *hasCapital.City* and *isCapitalOf* $\sqsubseteq$ *locatedIn*
- "everything which is a parent has *some* child (which is a person)"
  ER Diagram: *Parent* is a subclass of *Person*, minCardinality of *hasChild* is 1
  OWL/Description Logic: *Parent* $\equiv$ $\exists$*hasChild.Person*
  $\Leftarrow$: SQL: view, FOL: conclude an atom
  $\Rightarrow$: SQL: not possible
  FOL, e.g. tableau calculus use a skolem function and derive
  hasChild(alice, f_{child}*(alice))* and *Person(*f_{child}*(alice))*
- "every person has two parents which are persons"
  - would create/insert infinitely many new objects $\rightarrow$ needs a blocking strategy
  - in general, created objects may be equal or not (tableau calculus: $\rightarrow$ branching)