TECHNISCHE UNIVERSITÄT DRESDEN

DATABASE THEORY

Lecture 1: Introduction / Relational data model

David Carral

Knowledge-Based Systems

TU Dresden, 2nd Apr 2019

Course information before COVID-19

- Lectures: Tuesday, DS2 and DS 3 (9:20–12:40)
- Exercise classes: to be decided (communicated in next week's lectures)
  $\rightsquigarrow$ taught by Maximilian Marx
- Oral examination (details based on applicable examination regulations)
- Course homepage (dates, slides, excercise sheets):

  `https://iccl.inf.tu-dresden.de/web/Database_Theory_(SS2020)/en`

Online lecture

- At the beginning of each week, we will publish two videos with the weekly lectures and one video with the weekly exercises.
- Every week, we will host a "live session" where you can ask us questions.
- We have set up an online forum for discussions.
- No exercise video or "live session" in the first week of the semester.
- For more information about all of the above see the course homepage:

  https://iccl.inf.tu-dresden.de/web/Database_Theory_(SS2020)/en

Aims of the course

Obtain an understanding of key topics in database theory with a special focus on query formalisms:

- Relational data model
- Basic and advanced query languages
- Expressive power of query languages
- Complexity of query answering + some algorithmic approaches
- Modelling with constraints

Connect databases with other advanced topics in logic/KR/formal methods

Literature, prerequisites, related courses

- Serge Abiteboul, Richard Hull, Victor Vianu:
  Foundations of Databases. Addison-Wesley. 1994.
  - Available at `http://webdam.inria.fr/Alice/`
  - Slight deviations in the lecture
  - Further literature will be given for advanced topics
- Prerequisites: basics of first-order logic, Turing machines, worst-case complexity
- Related courses at TUD:
  - Advanced Logic
  - Foundations of Semantic Web Technologies
  - Introduction to Logic Programming
  - Introduction to Constraint Programming
  - Datenbanken (Grundlagen)
  - Theoretische Informatik & Logik

What is a database?

A Database Management System (DBMS) is a software to manage collections of data.

$\leadsto$ highly important class of software systems
$\leadsto$ major role in industry and in research
$\leadsto$ extremely wide variety of concepts and implementations

General three-level architecture of DBMS:

- External Level: Application-specific user views
- Logical Level: Abstract data model, independent of implementation, conceptual view
- Physical Level: Data structures and algorithms, platform-specific

In this lecture: focus on logical view for relational data model

What is a database? (2)

Basic functionality of DBMS:

- Schema definition: specify how data should be logically organised
- Update: insert/delete/update stored data
- Query: retrieve stored data or information derived from it
- Administration: user rights management, configuration, recovery, data export, etc.

Many related concerns:

- Persistence: data retained when DBMS is shut down
- Optimisation: ensure maximal efficiency
- Scalability: cope with increasing loads by adding resources
- Concurrency: support many update and query operations in parallel
- Distribution: combine data from several locations
- Interfaces: APIs, query languages, update languages, etc.
- ...

In this lecture: schema, query languages, some optimisation

Overview: Topics covered (excerpt)

1. Introduction | Relational data model
2. First-order queries
3. Complexity of query answering
4. Complexity of FO query answering
5. Conjunctive queries
6. Tree-like conjunctive queries
7. Query optimisation
8. Conjunctive Query Optimisation / First-Order Expressiveness
9. First-Order Expressiveness / Introduction to Datalog
10. Expressive Power and Complexity of Datalog
11. Optimisation and Evaluation of Datalog
12. Database dependencies
13. Query answering under constraints

The Relational Data Model

Database = collection of tables

Lines:

Line	Type
85	bus
3	tram
F1	ferry
...	...

Stops:

SID	Stop	Accessible
17	Hauptbahnhof	true
42	Helmholtzstr.	true
57	Stadtgutstr.	true
123	Gustav-Freytag-Str.	false
...	...	...

Connect:

From	To	Line
57	42	85
17	789	3
...	...	...

Every table has a schema:

- Lines[Line:string, Type:string]
- Stops[SID:int, Stop:string, Accessible:bool]
- Connect[From:int, To:int, Line:string]

Towards a formal definition of “table”

A table row has one value for each column
$\leadsto$ row = function from the attributes of the table schema to specific values

Example: The row

SID	**Stop**	**Accessible**
...	...	...
42	Helmholtzstr.	true
...	...	...

can be represented by the function:

$$f : \{\text{SID} \mapsto 42, \text{Stop} \mapsto \texttt{"Helmholtzstr."}, \text{Accessible} \mapsto \text{true}\}$$

Database = set of tables

Let **dom** ("domain") be the (infinite) set of conceivable values in tables.

For simplicity, we drop the datatypes of database columns and assume that each column uses the same datatype that supports all values in **dom**.

Definition 1.1:

- A relation schema $R[U]$ consists of a relation name R and a finite set U of attributes ($|U|$ is the arity of $R[U]$)
- A table for $R[U]$ is a finite set of functions from U to **dom**
- A database instance $\mathcal{I}$ is a finite set of tables

Note: we disregard the order and multiplicity of rows.

Tables are also called relation instances. The table with relation schema $R[U]$ in the database instance $\mathcal{I}$ is written $R^{\mathcal{I}}$.

Database = set of relations

Observation: Attribute names don't matter. Instead of the function

$$\{\text{SID} \mapsto 42, \text{Stop} \mapsto \texttt{"Helmholtzstr."}, \text{Accessible} \mapsto \text{true}\}$$

we could also use a tuple:

$$\langle 42, \texttt{"Helmholtzstr."}, \text{true}\rangle$$

Necessary assumption: Attributes have a fixed order.

Definition 1.2:

- A relation schema $R[U]$ is defined as before
- A table for $R[U]$ is a finite subset of $\mathbf{dom}^{|U|}$
- A database instance $\mathcal{I}$ is a finite set of tables

Recall that a subset of $\mathbf{dom}^{|U|}$ is just a $|U|$-ary relation. Sets of relations are also called relational structures.

Database = interpretation of first-order logic

Recall:

- First-order logic is based on predicate symbols with a fixed arity (we won't need function symbols here)
- An interpretation $\mathcal{I}$ of first-order logic is a pair $\langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$:
  - $\Delta^{\mathcal{I}}$ is a set (the domain of interpretation)
  - $\cdot^{\mathcal{I}}$ maps n-ary predicates p to n-ary relations $p^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^n$

This is (almost) a database instance!

Definition 1.3:

- domain of interpretation $\Delta^{\mathcal{I}}$ = database domain **dom**
- predicate symbol = relation name
- interpretation of predicate symbol (if finite!) = table
- finite first-order logic interpretation = database instance

Database = set of facts

Another convenient way to write databases:

Lines(85, `"bus"`)
Lines(F1, `"ferry"`)
Stops(42, `"Helmholtzstr."`, true)
...

Definition 1.4: A fact is an expression $p(t_1, \ldots, t_n)$ where

- p is an n-ary predicate symbol
- $t_1, \ldots t_n$ are constant symbols

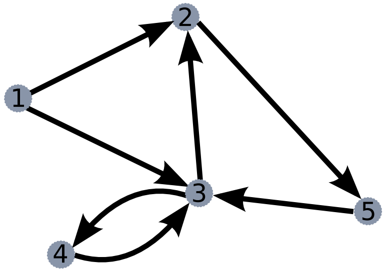
A database instance is a finite set of facts.

When interpreting these facts logically, their least model is again the database instance (viewed as a first-order logic interpretation).

Visualising relations

Binary relations (sets of pairs) can be viewed as directed graphs.
Example:

Source	Target
1	2
1	3
2	5
3	2
3	4
4	3
5	3



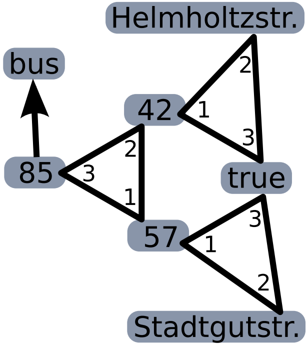
Many binary tables in one graph? Use table name to label edges!

Database = hypergraph

What to do with tables of arity $\neq 2$?
$\rightsquigarrow$ generalise graphs to hypergraphs

Definition 1.5: A hypergraph is a triple $\langle V, E, \rho \rangle$, where

- V is a set of vertices
- E is a set of edge names
- ρ maps each edge name $e \in E$ to an n-ary relation $\rho(e) \subseteq V^n$



In other words: finite hypergraphs are databases.

Summary: the relational model

Relational databases are everywhere:

- sets of tables with named attributes (“named perspective”)
- sets of relations (“unnamed perspective”)
- first-order logic interpretations
- sets of logical facts (ground atoms)
- hypergraphs (and graphs as a special case)

…all restricted to finite sets

Important elements of the theory of relational databases are very widely applicable, also to many datamodels that are not the classical relational one (e.g., graph databases, RDF databases, XML databases).

The Relational Algebra

Relational Algebra Queries

Query language based on a set of operations on databases.

Each operation refers to one or more tables and produces another table
(we often simplify notation and write a table name rather than a table instance)

Main operations of the named perspective:

- Selection σ
- Projection π
- Join $\bowtie$
- Renaming δ
- Difference $-$
- Union $\cup$
- Intersection $\cap$

Selection

"Find all bus lines"

$$\sigma_{\text{Type}=\texttt{"bus"}}\text{Lines}$$

"Find all connections that begin and end in the same stop"

$$\sigma_{\text{From}=\text{To}}\text{Connect}$$

Definition 1.6: The selection operator has the form $\sigma_{n=m}$

- n is an attribute name
- m is an attribute name or a constant value

Consider a table $R^{\mathcal{I}}$ for $R[U]$.

- For m constant value: $\sigma_{n=m}(R^{\mathcal{I}}) = \{f \in R^{\mathcal{I}} \mid f(n) = m\}$
- For m attribute name: $\sigma_{n=m}(R^{\mathcal{I}}) = \{f \in R^{\mathcal{I}} \mid f(n) = f(m)\}$

This is only defined if U contains the required attribute names.

Projection

"Find all possible types of lines"

$$\pi_{\text{Type}}\text{Lines}$$

"Find all pairs of adjacent stops on line 85"

$$\pi_{\text{From,To}}(\sigma_{\text{Line="85"}}\text{Connect})$$

Definition 1.7: The projection operator has the form $\pi_{a_1,\ldots,a_n}$ where each a_i is an attribute name.

Consider a table $R^{\mathcal{I}}$ for $R[U]$.

$$\pi_{a_1,\ldots,a_n}(R^{\mathcal{I}}) = \{f_{\{a_1,\ldots,a_n\}} \mid f \in R^{\mathcal{I}}\}$$

where $f_{\{a_1,\ldots,a_n\}}$ is the restriction of f to the domain $\{a_1,\ldots,a_n\}$, i.e., the function $\{a_1 \mapsto f(a_1),\ldots,a_n \mapsto f(a_n)\}$.
Of course this projection is only defined if $a_i \in U$ for each a_i.

Natural join

"Find all connections and their type of line"

Connect:

From	To	Line
57	42	85
17	789	3
…	…	…

Lines:

Line	Type
85	bus
3	tram
F1	ferry
…	…

Connect $\bowtie$ Lines:

From	To	Line	Type
57	42	85	bus
17	789	3	tram
…	…	…	…

Definition 1.8: The natural join operator has the form $\bowtie$.

Consider tables $R^{\mathcal{I}}$ for $R[U]$ and $S^{\mathcal{I}}$ for $S[V]$.

$$R^{\mathcal{I}} \bowtie S^{\mathcal{I}} = \{f : U \cup V \to \mathbf{dom} \mid f_U \in R^{\mathcal{I}} \text{ and } f_V \in S^{\mathcal{I}}\}$$

where f_U (f_V) is the restriction of f to elements in U (V) as before

Renaming

"Find all lines that depart from an accessible stop"

Stops:

SID	Stop	Accessible
57	Stadtgutstr.	true
123	Gustav-Freytag-Str.	false
...	...	...

Connect:

From	To	Line
57	42	85
17	789	3
...	...	...

We need to join Stops.SID with Connect.From $\rightsquigarrow$ use renaming

$$\pi_{Line}(\sigma_{\text{Accessible}=\texttt{"true"}}(\text{Stops} \bowtie \delta_{\text{From,To,Line}\to\text{SID,To,Line}}(\text{Connect})))$$

Definition 1.9: The renaming operator has the form $\delta_{a_1,\ldots,a_n\to b_1,\ldots,b_n}$ with all a_i mutually distinct attribute names, and likewise for all b_i.
Consider a table $R^{\mathcal{I}}$ for $R[\{a_1, \ldots, a_n\}]$.

$$\delta_{a_1,\ldots,a_n\to b_1,\ldots,b_n}(R^{\mathcal{I}}) = \{f \circ g \mid f \in R^{\mathcal{I}} \text{ and } g : \{b_i \mapsto a_i\}_{1\leq i\leq n}\}$$

where $f \circ g$ is function composition: $(f \circ g)(x) = f(g(x))$

Difference, Union, Intersection

Binary operators on tables of the same relational schema, defined like the usual set operations.

"Find all stops where line 3 departs, but line 8 does not depart."

"Find all stops where either line 3 or line 8 departs."

"Find all stops where both line 3 and line 8 depart."

Table constants in queries

It is sometimes convenient to define constant tables in queries.

"Find all stops near Helmholtzstr. (SID 42), including Helmholtzstr."

$$\delta_{\mathsf{To}\to\mathsf{StopId}}(\pi_{\mathsf{To}}(\sigma_{\mathsf{From}=\texttt{"42"}}\mathsf{Connect})) \cup \{\{\mathsf{StopId} \mapsto 42\}\}$$

One can generalise this to constant tables with more than one column or more than one table (no additional expressive power, see exercise).

Reachability

Generalising the previous example:

"Stops that are Helmholtzstr."

$$R_0 = \{\{\text{From} \mapsto 42\}\}$$

"Stops that are next to Helmholtzstr."

$$R_1 = \delta_{\text{To}\to\text{From}}(\pi_{\text{To}}(\text{Connect} \bowtie R_0))$$

"Stops at distance 2 from Helmholtzstr."

$$R_2 = \delta_{\text{To}\to\text{From}}(\pi_{\text{To}}(\text{Connect} \bowtie R_1))$$

Stops reachable from Helmholtzstr. with a short-distance ticket:

$$R_0 \cup R_1 \cup R_2 \cup R_3 \cup R_4$$

What about all stops reachable from Helmholtzstr.?
$\leadsto$ see upcoming lectures ...

Summary and Outlook

The relational model is very versatile

Relational algebra allows us to define queries with operators

Many operators exist, not all are really needed (see exercise)

Open questions:

- What does this have to do with logic? (next lecture)
- How hard is it to actually answer such queries? (complexity)
- How can we study the expressiveness of query languages?