Courses B0B36DBS, A4B33DS, A7B36DBS: Database Systems

Lecture 02:

Relational Model

Martin Svoboda
28. 2. 2017

Faculty of Electrical Engineering, Czech Technical University in Prague
Lecture Outline

• Logical database models
  ▪ Basic overview

• Model-Driven Development

• Relational model
  ▪ Description and features
  ▪ Transformation of ER / UML conceptual schemas
Logical Database Models
Layers of Database Modeling

- **Conceptual layer**
  - Models a part of the structured real world relevant for applications built on top of our database

- **Logical layer**
  - Specifies how conceptual components (i.e. entity types, relationship types, and their characteristics) are represented in logical data structures that are interpretable by machines

- **Physical layer**
  - Specifies how logical database structures are implemented in a specific technical environment
Logical Layer

• What are these logical structures?
  ▪ Formally...
    – Sets, relations, functions, graphs, trees, ...
      • I.e. traditional and well-defined mathematical structures
  ▪ Or in a more friendly way...
    – Tables, rows, columns, ...
    – Objects, pointers, ...
    – Collections, ...
    – ...

Logical Models

• Models based on **tables**
  - Structure
    - **Rows** for entities
    - **Columns** for attributes
  - Operations
    - Selection, projection, join, ...
  - Examples
    - **Relational model**
    - ... and various derived **table models** introduced by:
      • **SQL** (as it is standardized)
      • and particular implementations like Oracle, MySQL, ...
Logical Models

• Models based on objects
  ▪ Structure
    – Objects with attributes
    – Pointers between objects
  ▪ Motivation
    – Object-oriented programming (OOP)
    – Encapsulation, inheritance, ...
  ▪ Operations
    – Navigation
Logical Models

• Models based on **graphs**
  - **Structure**
    - Vertices, edges, attributes
  - **Operations**
    - Traversals, pattern matching, graph algorithms
  - **Examples**
    - Network model (one of the very first database models)
    - **Resource Description Framework** (RDF)
    - **Neo4j**, InfiniteGraph, OrientDB, FlockDB, ...
Logical Models

• Models based on **trees**
  - **Structure**
    - **Vertices** with attributes
    - **Edges** between vertices
  - **Motivation**
    - Hierarchies, categorization, semi-structured data
  - **Examples**
    - Hierarchical model (one of the very first database models)
    - **XML** documents
    - **JSON** documents
Overview of Logical Models

• There are plenty of (different / similar) models
  – The previous overview was intended just as an insight into some of the basic ideas and models

  ▪ Hierarchical, network, relational, object, object-relational, XML, key-value, document-oriented, graph, ...

• Why so many of them?
  ▪ Different models are suitable in different situations
  ▪ Not everything is (yet) standardized, proprietary approaches or extensions often exist
Logical Modeling

• Step 1: Selection of the right logical model

Conceptual schema

• Note that...
  ▪ Relational model is not always the best solution
Logical Modeling

• Step 1: Selection of the right logical model
  ▪ According to...
    – Data characteristics
      • True nature of real-world entities and their relationships
    – Query possibilities
      • Available access patterns, expressive power, ...
    – Intended usage
      • Storage (JSON data in document-oriented databases, ...)
      • Exchange (XML documents sent by Web Service, ...)
      • Publication (RDF triples forming the Web of Data, ...)
      • ...
    – Identified requirements
Logical Modeling

- **Step 2: Creation of a logical schema**
Logical Modeling

• Step 2: **Creation of a logical schema**
  
  ▪ **Goal**
    - Transformation of a conceptual schema to a logical one
  
  ▪ Real-world applications often need **multiple schemas**
    - Focus on different parts of the real world
    - Serve different components of the system
    - Even expressed in different logical models
  
  ▪ **Challenge:** can this be achieved automatically?
    - Or at least semi-automatically?
    - Answer: **Model-Driven Development**
Model-Driven Development (MDD)
Model-Driven Development

• MDD
  ▪ Software development approach
    – Executable schemas instead of executable code
      • I.e. schemas that can be automatically (or at least semi-automatically) converted to executable code
    – Unfortunately, just in theory... recent ideas, not yet fully applicable in practice today (lack of suitable tools)

• MDD principles can be used for database modeling as well
Terminology

• Levels of abstraction
  - **Platform-Independent Level**
    - Hides particular platform-specific details
  - **Platform-Specific Level**
    - Maps the conceptual schema (or its part) to a given logical model
    - Adds platform-specific details
  - **Code Level**
    - Expresses the schema in a selected machine-interpretable logical language
    - SQL, XML Schema, OWL, ...
Real-World Example

- Information System for Public Procurement
  - There are several logical models used:
    - Relational data model
      - for data storage
    - XML data model
      - for exchanging data with information systems of public authorities which issue public contracts
    - RDF data model
      - for publishing data on the Web of Linked Data in a machine-readable form (at least this is a goal...
Real-World Example

- Platform-independent schema
Real-World Example

- Platform-specific schema: relational model
Real-World Example

- Platform-specific schema: **relational model**
  - Notes to the previous UML diagram
    - It is a UML class diagram
      - But enhanced with features for modeling logical schemas in (object-)relational model
    - **Stereotypes** allow us to add **specific semantics** to basic constructs (class, attribute, association), e.g.,
      - `<<table>>` specifies that a class represents a table
      - `<<PK>>` specifies that an attribute models a primary key
      - `<<FK>>` specifies that an attribute/association models a foreign key
      - etc.
Real-World Example

• Code level: **SQL** (snippet)

```sql
CREATE TABLE Contract (  
    referenceNumber NUMBER(8) NOT NULL,  
    title VARCHAR2(50) NOT NULL,  
    description CLOB,  
    startDate DATE NOT NULL,  
    endDate DATE NOT NULL,  
    estimatedPrice NUMBER(9) NOT NULL,  
    ...  
);

ALTER TABLE Contract ADD CONSTRAINT PK_Contract  
    PRIMARY KEY (contractId);  
ALTER TABLE Contract ADD CONSTRAINT FK_Contract_Address  
    FOREIGN KEY (mainAddressId) REFERENCES Address (addressId);  
...  

CREATE TABLE Organization(...);
...  
```
Real-World Example

• Code level: **SQL** (snippet)
  - The previous code was generated **fully automatically**
    - from a platform-specific diagram
      - It has to contain all the necessary information
    - using a **CASE tool** (Computer-Aided Software Engineering)
      - Which can detect errors and
      - helps with the specification
Real-World Example

- XML
Relational Model
Relational Model

• Relational model
  ▪ Allows to store entities, relationships, and their attributes in relations
  ▪ Founded by E. F. Codd in 1970

• Informally...
  ▪ Table = collection of rows, each row represents one entity, values of attributes are stored in columns
  ▪ Tables are more intuitive, but conceal important mathematical background
Relational Model

• Definitions and terminology
  - **Schema of a relation**
    - Description of a relational structure (everything except data)
    - \( S( A_1:T_1, A_2:T_2, \ldots, A_n:T_n ) \)
      - \( S \) is a schema name
      - \( A_i \) are attribute names and \( T_i \) their types (attribute domains)
      - Specification of types is often omitted
    - Example:
      - Person(personalId, firstName, lastName)
  - **Schema of a relational database**
    - Set of relation schemas (+ integrity constraints, ...)
Relational Model

• Definitions and terminology for data
  ▪ Relation
    – Subset of the Cartesian product of attribute domains \( T_i \)
      • I.e. relation is a set
    – Items are called tuples
  ▪ Relational database
    – Set of relations
Relational Model

• Basic requirements (or consequences?)
  - **Atomicity** of attributes
    - Only simple types can be used for domains of attributes
  - **Uniqueness** of tuples
    - Relation is a set, and so *two identical tuples cannot exist*
  - **Undefined order**
    - Relation is a set, and so *tuples are not mutually ordered*
  - **Completeness** of values
    - There are no *holes* in tuples, i.e. *all values are specified*
      • However, special *NULL* values (well-known from relational databases) can be added to attribute domains
**Integrity Constraints**

- **Identification**
  - Every tuple is identified by one or more attributes
  
  - **Superkey** = set of such attributes
    - Trivial and special example: all the relation attributes
  
  - **Key** = superkey with a *minimal* number of attributes
    - I.e. no attribute can be removed so that the identification ability would still be preserved
    
    - Multiple keys may exist in one relation
      - They even do not need to have the same number of attributes
  
  - Notation: keys are underlined
    - Relation(\text{Key}, \text{CompositeKeyPart1}, \text{CompositeKeyPart2}, \ldots)
    - Note the difference between simple and composite keys
Integrity Constraints

- Referential integrity
  - **Foreign key** = set of attributes of the referencing relation which corresponds to a (super)key of the referenced relation
    - It is usually not a (super)key in the referencing relation
    - Notation
      - \( \text{ReferencingTable.foreignKey} \subseteq \text{ReferencedTable.Key} \)
      - \( \text{foreignKey} \subseteq \text{ReferencedTable.Key} \)
Sample Relational Database

- **Schema**
  
  - **Course**(Code, Name, ...)
  
  - **Schedule**(Id, Event, Day, Time, ...), Event ⊆ Course.Code

- **Data**

<table>
<thead>
<tr>
<th>Id</th>
<th>Event</th>
<th>Day</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A7B36DBS</td>
<td>THU</td>
<td>11:00</td>
</tr>
<tr>
<td>2</td>
<td>A7B36DBS</td>
<td>THU</td>
<td>12:45</td>
</tr>
<tr>
<td>3</td>
<td>A7B36DBS</td>
<td>THU</td>
<td>14:30</td>
</tr>
<tr>
<td>4</td>
<td>A7B36XML</td>
<td>FRI</td>
<td>09:15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A7B36DBS</td>
<td>Database systems</td>
</tr>
<tr>
<td>A7B36XML</td>
<td>XML technologies</td>
</tr>
<tr>
<td>A7B36PSI</td>
<td>Computer networks</td>
</tr>
</tbody>
</table>
Relations vs. Tables

• Tables
  ▪ Table header ~ relation schema
  ▪ Row ~ tuple
  ▪ Column ~ attribute

• However...
  ▪ Tables are not sets, and so...
    – there can be duplicate rows in tables
    – rows in tables can be ordered
  ▪ I.e. SQL and existing RDBMS do not (always) follow the formal relational model strictly
Object vs. (Object-)Relational Model

- **Relational model**
  - Data stored in flat tables
  - Suitable for data-intensive batch operations

- **Object model**
  - Data stored as graphs of objects
  - Suitable for individual navigational access to entities

- **Object-Relational model**
  - Relational model enriched by object elements
    - Attributes may be of complex data types
    - Methods can be defined on data types as well
Transformation of UML / ER to RM
Conceptual Schema Transformation

- **Basic idea**
  - What we have
    - ER: entity types, attributes, identifiers, relationship types, ISA hierarchies
    - UML: classes, attributes, associations
  - What we need
    - Schemas of relations with attributes, keys, and foreign keys
  - How to do it
    - Classes with attributes $\rightarrow$ relation schemas
    - Associations $\rightarrow$ separate relation schemas or together with classes (depending on cardinalities...)
Classes

• **Class** →
  - **Separate table**
    - `Person(personalNumber, address, age)`
  - **Artificial keys**
    - Artificially added *integer identifiers*
      - with no correspondence in the real world
      - but with several efficiency and also design advantages
      - usually automatically generated and assigned
    - `Person(person1d, personNumber, address, age)`
Attributes

- Multivalued attribute →

  - Separate table
    - Person(personalNumber)
    - Phone(personalNumber, phone)
    - Phone.personalNumber ⊆ Person.personalNumber

<table>
<thead>
<tr>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>- personalNumber</td>
</tr>
<tr>
<td>- phone: String [1..*]</td>
</tr>
</tbody>
</table>
Attributes

- **Composite attribute** →

  - **Separate table**
    - `Person(personalNumber)`
    - `Address(personalNumber, street, city, country)`
    - `Address.personalNumber ⊆ Person.personalNumber`

  - **Sub-attributes can also be inlined**
    - But only in case of (1,1) cardinality
    - `Person(personNumber, street, city, country)`
Binary Associations

- Multiplicity (1,1):(1,1) →

- Single table
  - Person(personalNumber, address, age, serialNumber, color)
Binary Associations

- **Multiplicity (1,1):(0,1)** →

  - **Two tables**
    - `Person(personalNumber, address, age, serialNumber)`
    - `Mobile(serialNumber, color)`

  - Why not just 1 table?
    - Because a mobile phone can exist independently of a person
Binary Associations

- **Multiplicity (0,1):(0,1)** →

  ![Diagram]

  - Three tables
    - Person(personalNumber, address, age)
    - Mobile(serialNumber, color)
    - Ownership(personalNumber, serialNumber)

  Ownership.personalNumber ⊆ Person.personalNumber
  Ownership.serialNumber ⊆ Mobile.serialNumber

  - Note that a personal number and serial number are both independent keys in the Ownership table
Binary Associations

- Multiplicity (1,n)/(0,n):(1,1) →

  - Two tables
    - Person(personalNumber, address, age)
    - Mobile(serialNumber, color, personalNumber)
      Mobile.personalNumber ⊆ Person.personalNumber
    - Why a personal number is not a key in the Mobile table?
      • Because a person can own more mobile phones
Binary Associations

- Multiplicity (1,n)/(0,n):(0,1) →

  Three tables
  - Person(personalNumber, address, age)
  - Mobile(serialNumber, color)
  - Ownership(personalNumber, serialNumber)

  Why a personal number is not a key in the Ownership table?
  - Because a person can own more mobile phones
Binary Associations

- Multiplicity (1,n)/(0,n):(1,n)/(0,n) →

  - Three tables
    - Person(personalNumber, address, age)
    - Mobile(serialNumber, color)
    - Ownership(personalNumber, serialNumber)
    - Ownership.personalNumber ⊆ Person.personalNumber
    - Ownership.serialNumber ⊆ Mobile.serialNumber

- Note that there is a composite key in the Ownership table
Attributes of Associations

• Attribute of an association →

  ▪ Stored together with a given association table
    – Person(personNumber, name)
    – Team(name, url)
    – Member(personNumber, name, from, to)
    Member.personNumber ⊆ Person.personNumber
    Member.name ⊆ Team.name

  ▪ Multivalued and composite attributes are transformed analogously to attributes of ordinary classes
General Associations

- **N-ary association →**
  - Universal solution: 
    
    N tables for classes + 1 association table

    - `Person(personNumber)`
    - `Project(projectNumber)`
    - `Team(name)`
    - `Worker(personNumber, projectNumber, name)`

    Worker.personNumber ⊆ Person.personNumber
    Worker.projectNumber ⊆ Project.projectNumber
    Worker.name ⊆ Team.name

- **Less tables?** Yes, in case of nice (1,1) cardinalities...
Hierarchies

- ISA hierarchy →
  - Universal solution: separate table for each type with specific attributes only
    - Person(personalNumber, name)
    - Professor(personalNumber, phone)
    - Student(personalNumber, studiesFrom)
    - Professor.personalNumber ⊆ Person.personalNumber
    - Student.personalNumber ⊆ Person.personalNumber
  - Applicable in any case (w.r.t. covering / overlap constraints)
  - Pros: flexibility (when attributes are altered)
  - Cons: joins (when full data is reconstructed)
Hierarchies

• ISA hierarchy →
  ▪ Only one table for a hierarchy source
    – Person(personalNumber, name, phone, studiesFrom, type)
    – Universal once again, but not always suitable
      • Types of instances are distinguished by an artificial attribute
        » Enumeration or event a set depending on the overlap constraint
    – Pros: no joins
    – Cons: NULL values required (and so it is not a nice solution)
Hierarchies

- **ISA hierarchy** →
  - Separate table for each leaf type
    - Professor(`personalNumber, name, phone`)
    - Student(`personalNumber, name, studiesFrom`)
  - This solution is **not always applicable**
    - In particular when the covering constraint is false
  - Pros: no joins
  - Cons:
    - Redundancies (when the overlap constraint is false)
    - Integrity considerations (uniqueness of a personal number)
Weak Entity Types

- **Weak entity type** →
  - Separate table
    - `Institution(name)`
    - `Team(code, name)`
    - `Team.name ⊆ Institution.name`
    - Recall that the cardinality must always be (1,1)
    - Key of the weak entity type involves also a key (any when more available) from the entity type it depends on