Course A7B36DBS: **Database Systems**

Lecture 02:

Relational Model

Martin Svoboda

Irena Holubová

Tomáš Skopal

Faculty of Electrical Engineering, Czech Technical University in Prague

Outline

- **Logical database models**
  - Short introduction
  - Model-Driven Development
- **Relational model**
  - Description and features
  - **Transformation of ER / UML conceptual schemas**

Logical Database Models

Layers of Database Modeling

Abstraction

Implementation

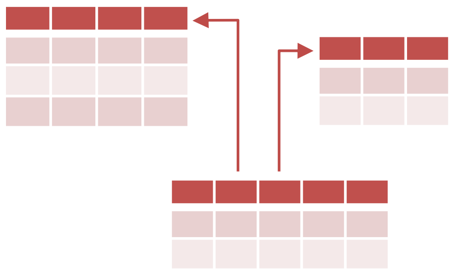
- **Conceptual layer**
  - Models a part of the structured real world (**entities, their characteristics and relationships between them**) relevant for applications built on top of our database
- **Logical layer**
  - Specifies how conceptual components are represented in **logical data structures** interpretable by machines
- **Physical layer**
  - Specifies how logical database structures are implemented in a specific technical environment

Logical Data Structures

- **What are actually these structures?**
  - Formally…
    - **Sets, relations, functions, graphs, trees, …**
      - I.e. traditional and well-defined mathematical structures
  - Or in a more friendly way…
    - **Tables, objects, pointers, collections, …**

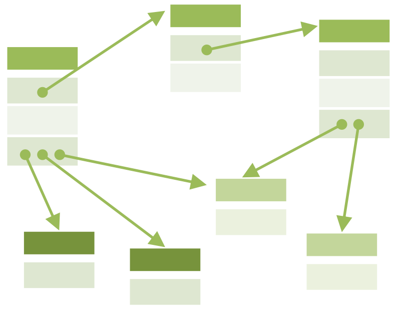
Overview of Logical Models

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



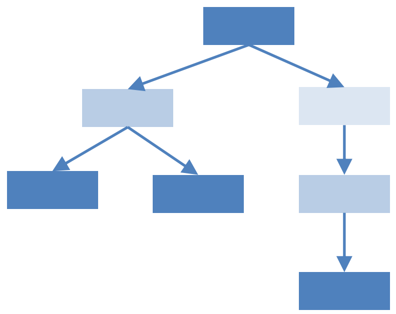
Overview of Logical Models

- Models based on **objects**
  - Structure
    - **Objects** with **attributes**
    - **Pointers** between objects
  - Motivation
    - Object-oriented programming (OOP)
    - Encapsulation, inheritance, …
  - Operations
    - **Navigation**



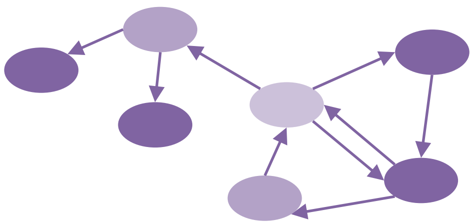
Overview of Logical Models

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



Overview of Logical Models

- Models based on **graphs**
  - Structure
    - Vertices, edges, attributes
  - Operations
    - **Navigation**
  - Examples
    - Network model (one of the very first database models)
    - **Resource Description Framework** (RDF)
    - **Neo4j**, InfiniteGraph, OrientDB, FlockDB, …

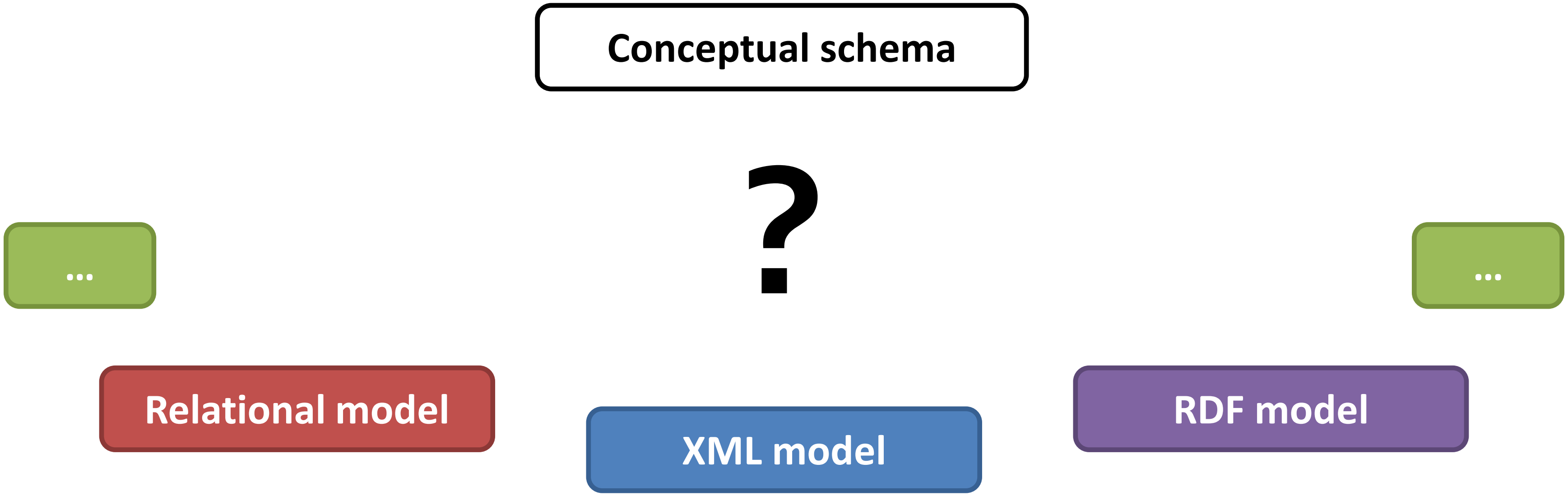


Overview of Logical Models

- **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**, ...
  - Note that
    - **They are suitable for different purposes**
    - Standards are often not strictly followed
    - Proprietary extensions are often available

Process of Logical Modeling

- Problem 1: **Choosing the right logical model/s**



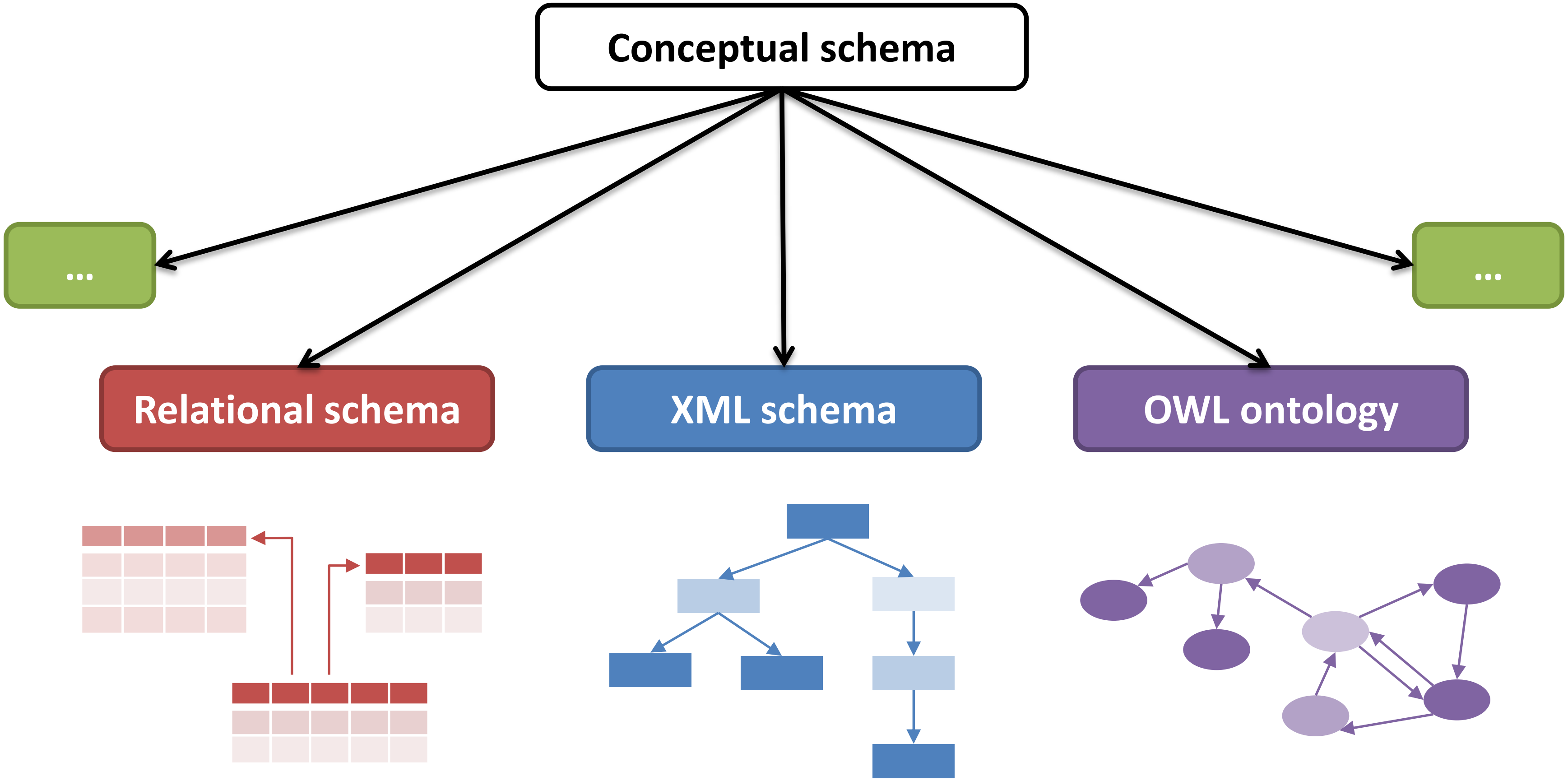
- Note that…
  - **Relational model is not always the best solution!**

Process of Logical Modeling

- Problem 1: **Choosing the right logical model/s**
  - According to…
    - **Data features**
      - True nature of real-world entities and their relationships
    - **Intended usage**
      - Storage (JSON data in document-oriented databases, …)
      - Exchange (XML documents sent by Web Service, …)
      - Publication (RDF triples forming the Web of Data, …)
      - …
    - **Query possibilities** – available expressive power
    - **Requirements** of stakeholders

Process of Logical Modeling

- Problem 2: **Designing logical schema/s**



Process of Logical Modeling

- Problem 2: **Designing logical schema/s**
  - Having a given conceptual schema
  - Working with different logical models
  - Covering different parts of the reality
  - Serving for different purposes
  - Challenge: **can this be achieved automatically?**
    - Or at least semi-automatically?
  - Answer: **Model-Driven Development**

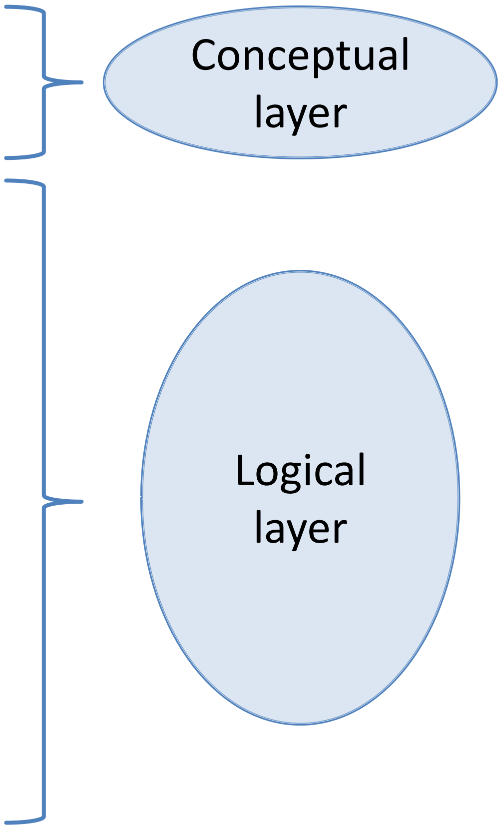
Model-Driven Development (MDD)

Model-Driven Development

- **MDD**
  - Software development approach
    - Can be used as a database design methodology as well
  - It enables us to create **executable schemas instead of executable code**
    - I.e. to create schemas that can be automatically (or at least semi-automatically) converted to executable code
    - **Unfortunately, just in theory**… recent ideas, **not yet applicable in practice today** (lack of suitable tools)
    - But we will show how to apply its principles in order to deal with multiple and different logical schemas we need to apply in our database system

MDD for Logical Database Schemas

- 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, …

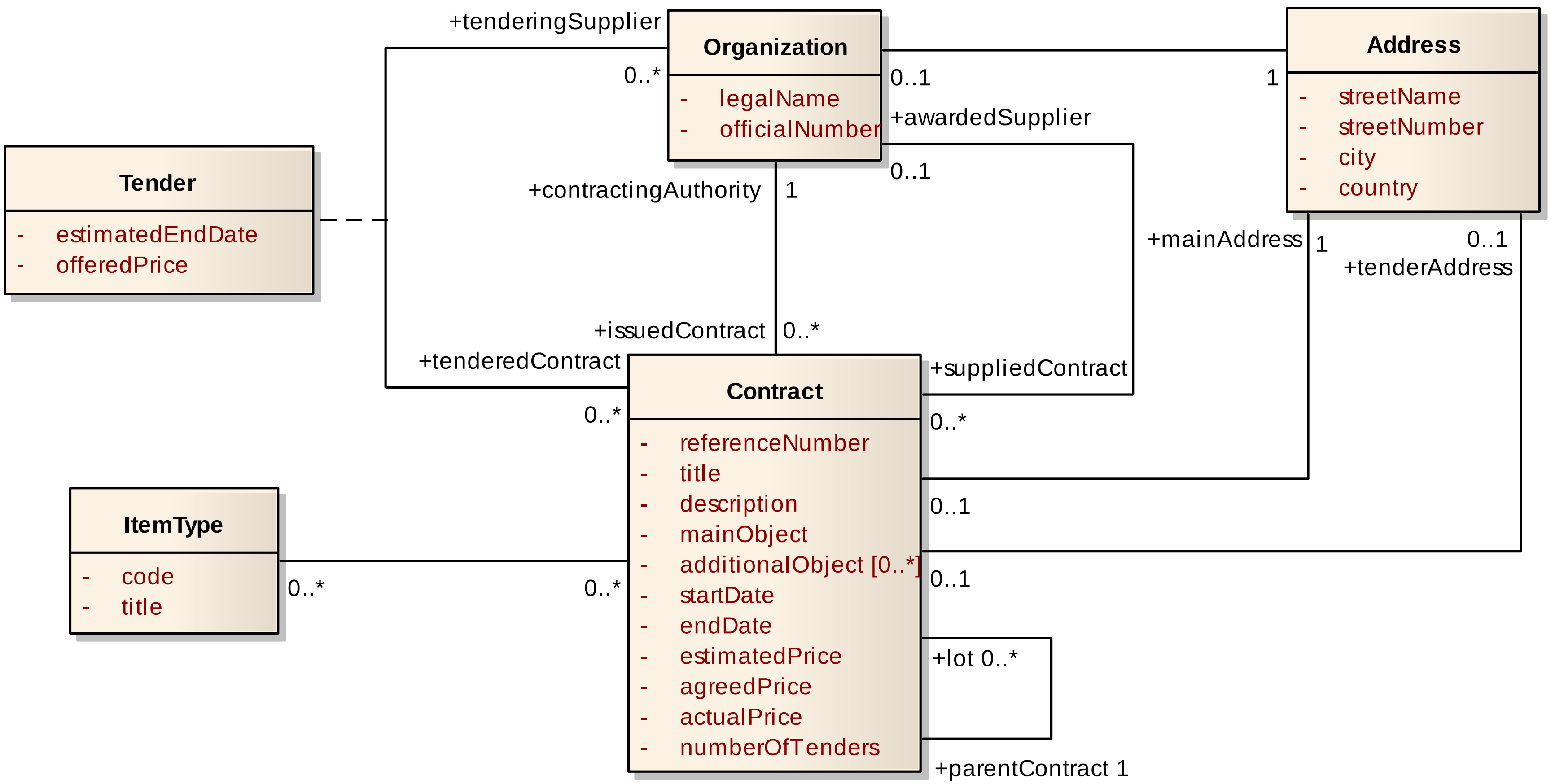


Practical Example

- Information System for Public Procurement
  - http://www.isvzus.cz/
  - There are many logical models to deal with:
    - **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…)

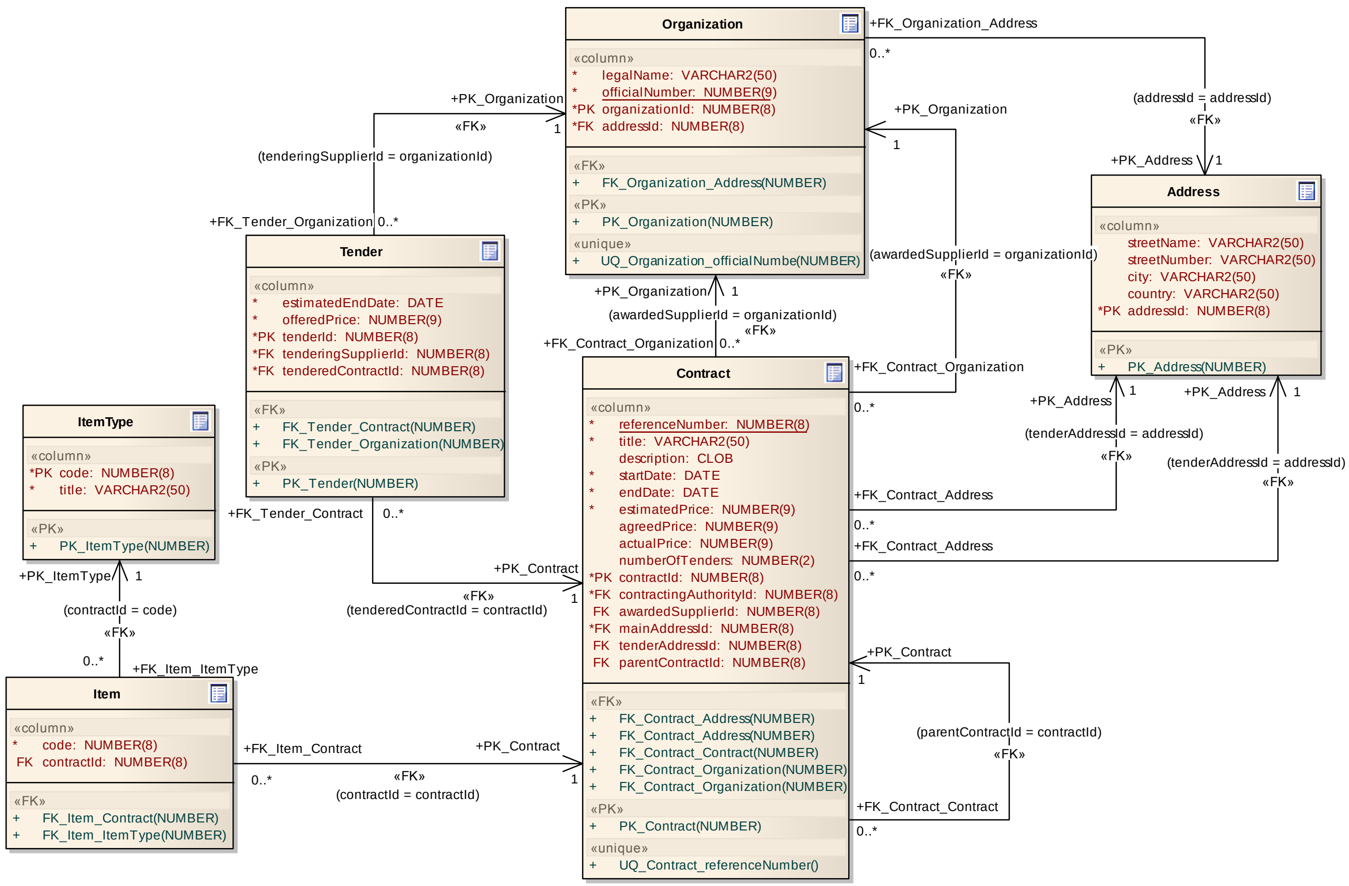
Practical Example

- Platform-independent schema



Practical Example

- Platform-specific schema: **relational model**



Practical 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 models a table
      - <<PK>> specifies that an attribute models a primary key
      - <<FK>> specifies that an attribute/association models a foreign key
      - etc.

Practical Example

- Code level: **SQL** (snippet)

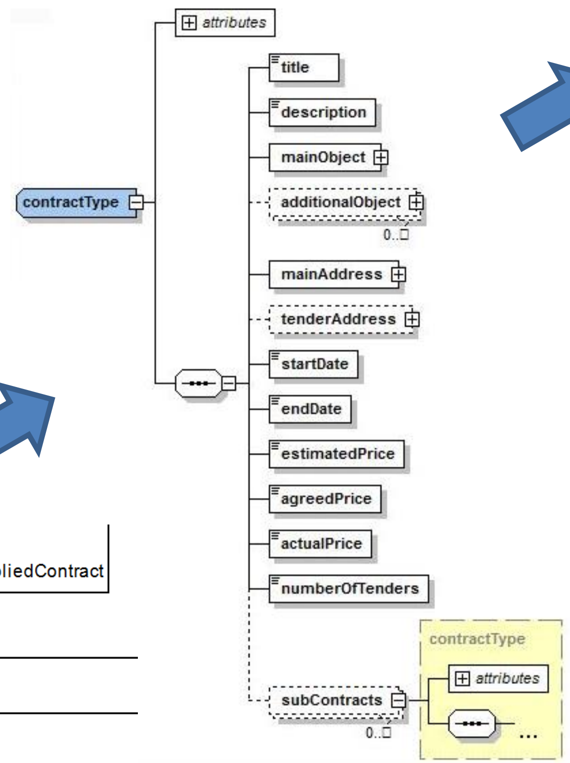
```
CREATE TABLE Contract (
  referenceNumber NUMBER(8) NOT NULL,
  title VARCHAR2(50) NOT NULL,
  description CLOB,
  startDate DATE NOT NULL,
  endDate DATE NOT NULL,
  estimatedPrice NUMBER(9) NOT NULL,
  ...
);

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(...);
...
```

Practical Example

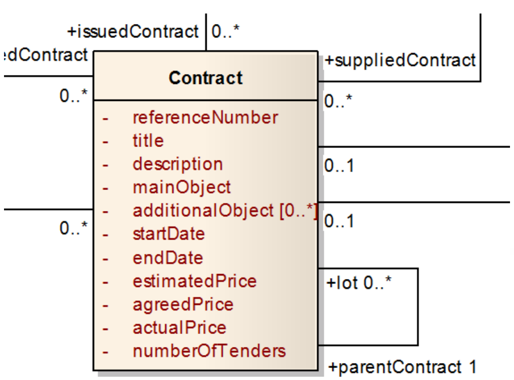
- Code level: **SQL** (snippet)
  - The previous code was generated **fully automatically**
    - from a platform-specific diagram
      - **It has to contain all the necessary information**
    - using a **CASE tool** (Computer-Aided Software Engineering)
      - Which can detect errors and
      -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

Practical Example

- **XML**



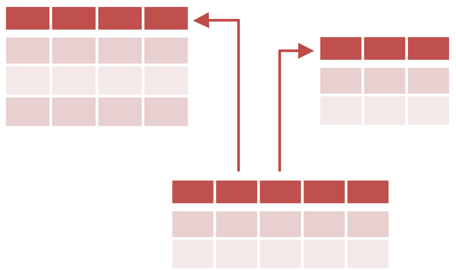

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2012 sp1 (http://www.altova.com) by IM (Charles
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:complexType name="addressType">
		<xs:sequence>
			<xs:element name="streetName"/>
			<xs:element name="streetNumber"/>
			<xs:element name="city"/>
			<xs:element name="country"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="objectType">
		<xs:sequence>
			<xs:element name="code" type="xs:int"/>
			<xs:element name="title" type="xs:string"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="contractType">
		<xs:sequence>
			<xs:element name="title" type="xs:string"/>
			<xs:element name="description" type="xs:string"/>
			<xs:element name="mainObject" type="objectType"/>
			<xs:element name="additionalObject" type="objectT
			<xs:element name="mainAddress" type="addressType"
			<xs:element name="tenderAddress" type="addressTyp
			<xs:element name="startDate" type="xs:date"/>
			<xs:element name="endDate" type="xs:date"/>
			<xs:element name="estimatedPrice" type="xs:float"
			<xs:element name="agreedPrice" type="xs:float"/>
			<xs:element name="actualPrice" type="xs:float"/>
			<xs:element name="numberOfTenders" type="xs:int"/
			<xs:element name="subContracts" type="contractTyp
		</xs:sequence>
		<xs:attribute name="referenceNumber" type="xs:int"/>
	</xs:complexType>
	<xs:complexType name="tenderedContractType">
```



Relational Model

Relational Model

- Relational model
  - Model for storage of objects and their relationships in **relations** (**tables**)
  - Founded by E. F. Codd in 1970



- Relations vs. tables
  - **Relation**
  - **Table** = structure with rows and columns
    - Tables are more intuitive,
      but hide important mathematical background!

Relational Model

- Definitions
  - **Relation schema**
    - Description of a relation 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 attributes and T_i their types (attribute domains)
      - Specification of types can be omitted
  - **Schema of a relational database**
    - Set of relation schemas (+ integrity constraints, …)

Relational Model

- Definitions
  - **Relation** = data
    - 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 (consequences?)
  - **Atomicity of attributes**
    - Only simple types can be used for domains of attributes
  - **Unique identification**
    - Relation is a set, and so **two identical tuples cannot exist**
  - **Undefined ordering**
    - Once again, relation is a set, and so **tuples are not ordered**
  - **Completeness of tuples**
    - There are no *holes* in tuples, i.e. **all values are specified**
      - However, `NULL` values (well-known from relational databases) can be added to attribute domains as special metavalues

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
    - More precisely: **no attribute can be removed so that the identification ability would still be preserved**
    - There can be more keys
      - Even with different numbers of attributes
    - Notation: keys are underlined
      - Relation(<u>Key</u>, <u>CompositeKeyPart1, CompositeKeyPart2</u>, …)
      - **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
      - ReferencingTable.foreignKey ⊆ ReferencedTable.Key
      - foreignKey ⊆ ReferencedTable.Key

Sample Relational Database

- Schema

  Course(<u>Code</u>, Name, …)

  Schedule(<u>Id</u>, Event, Day, Time, …), Event ⊆ Course.Code

- Data

Id	Event	Day	Time	…
1	A7B36DBS	THU	11:00	
2	A7B36DBS	THU	12:45	
3	A7B36DBS	THU	14:30	
4	A7B36XML	FRI	09:15	

Code	Name	…
A7B36DBS	Database systems	
A7B36XML	XML technologies	
A7B36PSI	Computer networks	

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 the existing RDBMS do not follow the relational model strictly

Object vs. (Object-)Relational Model

- **Object model**
  - Data stored as graphs of objects
  - Suitable for individual navigational access to entities
- **Relational model**
  - Data stored in flat tables
  - Suitable for data-intensive batch operations
- **Object-Relational model**
  - Relational model enriched by object elements
    - Attributes may be of complex types
    - Methods can be defined on attribute types

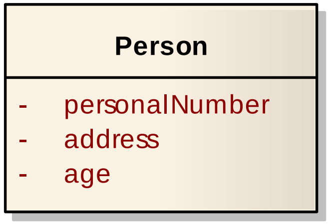
Transformation of UML / ER to RM

Conceptual Schema Transformation

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

Classes

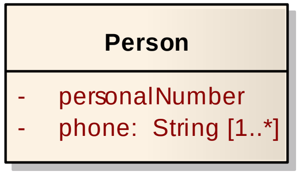
- **Class →**
  - Separate table
    - Person(personalNumber, address, age)



  - **Artificial keys**
    - Artificially introduced **integer identifiers**
      - with no correspondence in the real world
      - but with several efficiency and also design advantages
      - and usually automatically generated and assigned
    - Person(personId, personNumber, address, age)

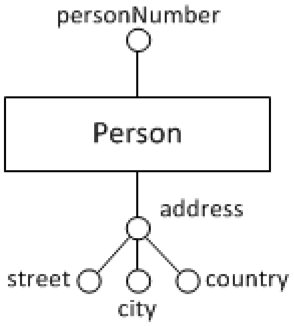
Attributes

- **Multivalued attribute →**
  - Separate table
    - Person(<u>personalNumber</u>)
      Phone(<u>personalNumber, phone</u>)
      Phone.personalNumber ⊆ Person.personalNumber



Attributes

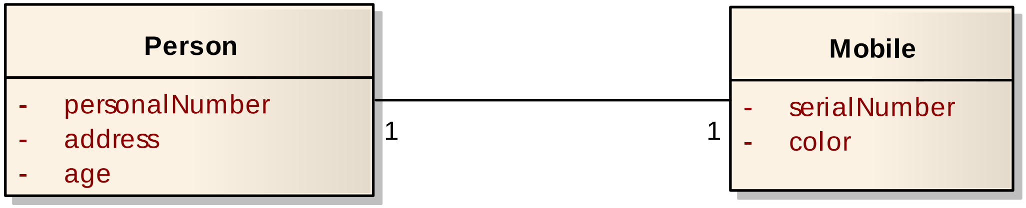
- **Composite attribute →**



  - 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

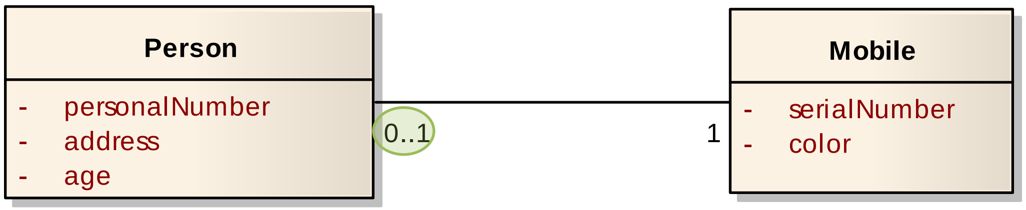
- **Multiplicity (1,1):(1,1) →**



  - Single table
    - Person(<u>personalNumber</u>, address, age, <u>serialNumber</u>, color)

Binary Associations

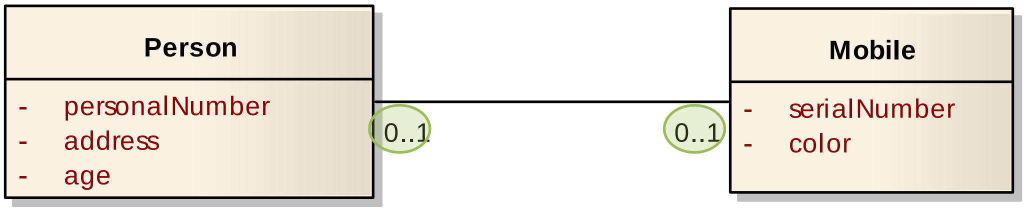
- **Multiplicity (1,1):(0,1) →**



  - Two tables
    - Person(<u>personalNumber</u>, address, age, <u>serialNumber</u>)
      Person.serialNumber ⊆ Mobile.serialNumber
      Mobile(<u>serialNumber</u>, color)
    - Why not just 1 table?
      - Because a mobile phone can exist independently of a person

Binary Associations

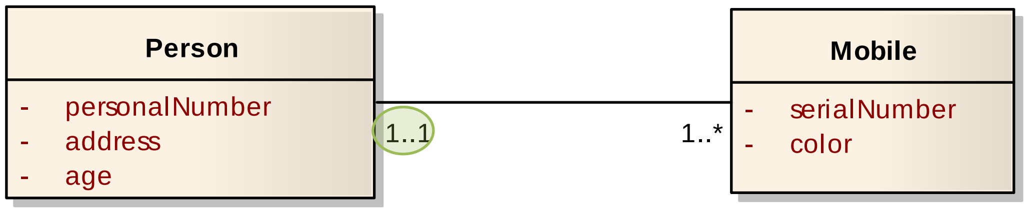
- **Multiplicity (0,1):(0,1) →**



  - 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

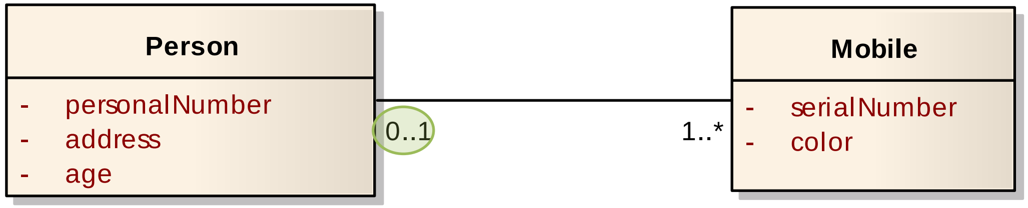
- **Multiplicity (1,n)/(0,n):(1,1) →**



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

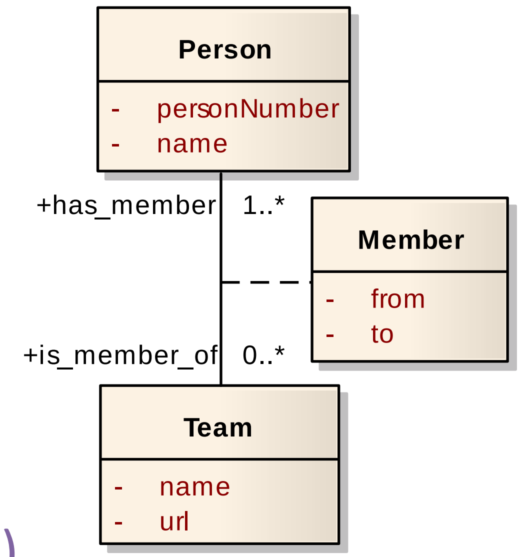
Binary Associations

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

  - Three tables
    - Person(<u>personalNumber</u>, address, age)
      Mobile(<u>serialNumber</u>, color)
      Ownership(<u>personalNumber, serialNumber</u>)
      Ownership.personalNumber ⊆ Person.personalNumber
      Ownership.serialNumber ⊆ Mobile.serialNumber
    - Note that there is a composite key in the Ownership table

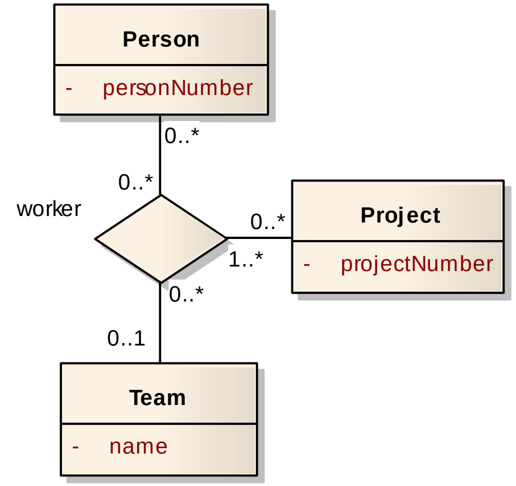
Attributes of Associations

- **Attribute of an association →**
  - Stored together with a given association table
    - Person(<u>personNumber</u>, name)
      Team(<u>name</u>, url)
      Member(<u>personNumber, name</u>, from, to)
      Member.personNumber ⊆ Person.personNumber
      Member.name ⊆ Team.name
  - Multivalued and composite attributes are transformed analogously to attributes of ordinary classes



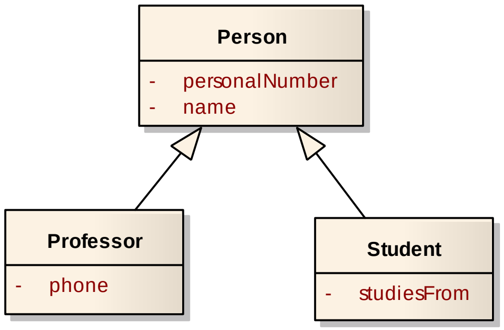
General Associations

- **N-ary association →**
  - Universal solution:
    N tables for classes +
    1 association table
    – Person(<u>personNumber</u>)
      Project(<u>projectNumber</u>)
      Team(<u>name</u>)
      Worker(<u>personNumber, projectNumber, name</u>)
      Worker.personNumber ⊆ Person.personNumber
      Worker.projectNumber ⊆ Project.projectNumber
      Worker.name ⊆ Team.name
  - **Less tables?** Yes, in case of (1,1) cardinalities…



Hierarchies

- **ISA hierarchy →**
  - Universal solution:
    separate table for each type
    - Person(<u>personalNumber</u>, name)
      Professor(<u>personalNumber</u>, phone)
      Student(<u>personalNumber</u>, studiesFrom)
      Professor.personalNumber ⊆ Person.personalNumber
      Student.personalNumber ⊆ Person.personalNumber
    - Applicable in any case (w.r.t. **covering / overlap constraints**)
    - Pros: flexibility (when altering attributes)
    - Cons: joins (when reconstructing entire persons)



Hierarchies

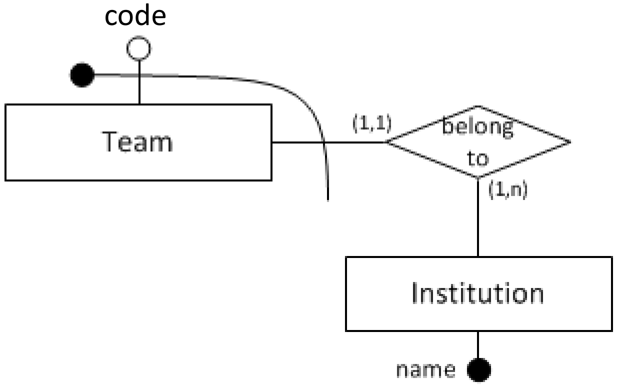
- **ISA hierarchy →**
  - Only one table for a hierarchy source
    - Person(personalNumber, name, phone, studiesFrom, type)
    - Universal once again, but **not always suitable**
      - Types of instances are distinguished by an artificial attribute
      - Should this attribute simulate an **enumeration** or even a **set**?
      - Depends 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 is always (1,1)
    - Key of the weak entity type involves also a key from the entity type it depends on