Relační model (2)

Logické databázové modely

- mezi konceptuální a fyzickou vrstvou
- specifikuje, jak jsou konceptuální komponenty (entitní typy, vztahové typy apod.) reprezentovány v logických datových strukturách (které jsou interpretovatelné stroji)
- logické struktury = množiny, relace, funkce, grafy, stromy,…
    - ale také tabulky, objekty, kolekce atd.

Typy logických modelů

Modely založené na tabulkách

- **struktura**:
    - řádky pro entity
    - sloupce pro atributy

- **operace**:
    - výběr, projekce, join

- **příklady**:
    - relační model
    - tabulkové modely ze SQL

Modely založené na objektech

- OOP, zapouzdření, dědičnost,….

- **struktura**:
    - objekty s atributy
    - ukazatelé mezi objekty

- **operace**:
    - navigace

Modely založené na grafech

- **struktura**:
    - vrcholy, hrany, atributy

- **operace**:
    - průchody, pattern matching, grafové algoritmy

- **příklady**:
    - síťové modely
    - RDF
    - Neo4j apod.

Modely založené na stromech

- hierarchie, kategorizace, semi-strukturovaná data
- **struktura**:
  - vrcholy s atributy
  - hrany mezi vrcholy
- **příklady**:
  - XML dokumenty
  - JSON dokumenty

Logické modelování

Krok 1: Výběr správného logického modelu (relační, XML, RDF,…)

- dle charakteristiky dat
- dle možností dotazování
- dle užití (úložiště - JSON, výměna - XML, publikace - RDF,…)
- dle identifikovaných požadavků

Krok 2: Tvorba logického schématu (relační, XML, OWL ontologie,…)

- cíl: transformace konceptuálního schématu do logického
- aplikace často potřebují vícero schémat
  - zaměřují se na různé části reálného světa
  - slouží různým komponentám systému
  - jsou vyjádřeny v jiném logickém modelu
- Může toho být dosaženo (polo)automaticky? → Model-Driven Development

Model driven development (MDD)

- přístup k vývoji SW
- executable schémata místo executable kódu
  - tzn. schémata, která mohou být (polo)automaticky převedena na spustitelný kód
- bohužel zatím pouze teoreticky, není to zatím plně aplikovatelné
- MDD principy mohou být použity i pro databázové modelování

Terminologie

- úrovně abstrakce:
  - platformě-nezávislá
    - skrývá detaily pro specifickou platformu

konceptuální vrstva

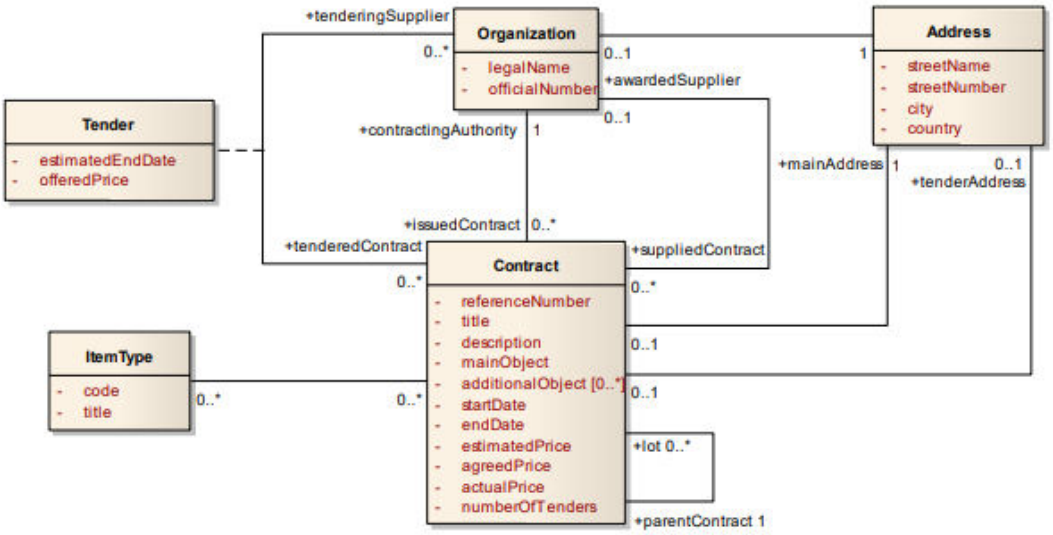
  - platformě-specifická
    - mapuje konceptuální schéma na daný logický model
      - přidává platformě-specifické detaily
  - úroveň kódu
    - vyjádření schématu v daném strojově-interpretovatelném logickém jazyce
    - SQL, XML schéma, OWL,…

logická vrstva

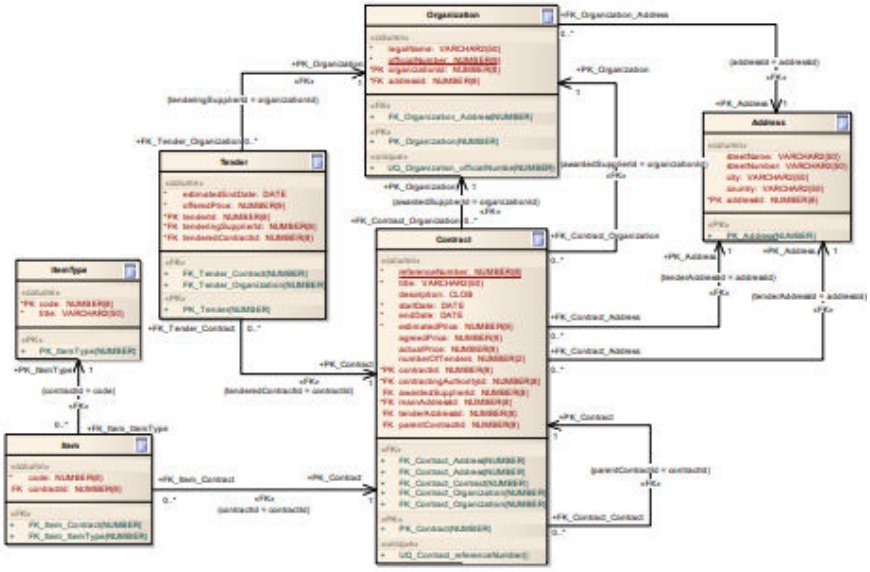
Real-word příklad

- informační systém pro veřejné zakázky
- jsou použity různé logické modely:
    - relační data model
        - pro ukládání dat
    - XML data model
        - pro výměnu dat s IT systémy veřejných autorit
    - RDF data model
        - pro publikaci dat na Web of Linked Data ve strojově čitelné podobě

- **Platform-independent schema**



- dále pak platformě specifické schéma (relační model)
    - je to UML class diagram rozšířený o možnost modelování logických schémat v objektově-relačním modelu
    - stereotypy nám umožní přidat specifickou sémantiku k základním konstruktům
        - <<table>> znamená, že třída reprezentuje tabulku
        - <<PK>> znamená, že atribut tvoří primární klíč
        - <<FK>> znamená, že atribut/asociace tvoří cizí klíč



Code level: **SQL** (snippet)

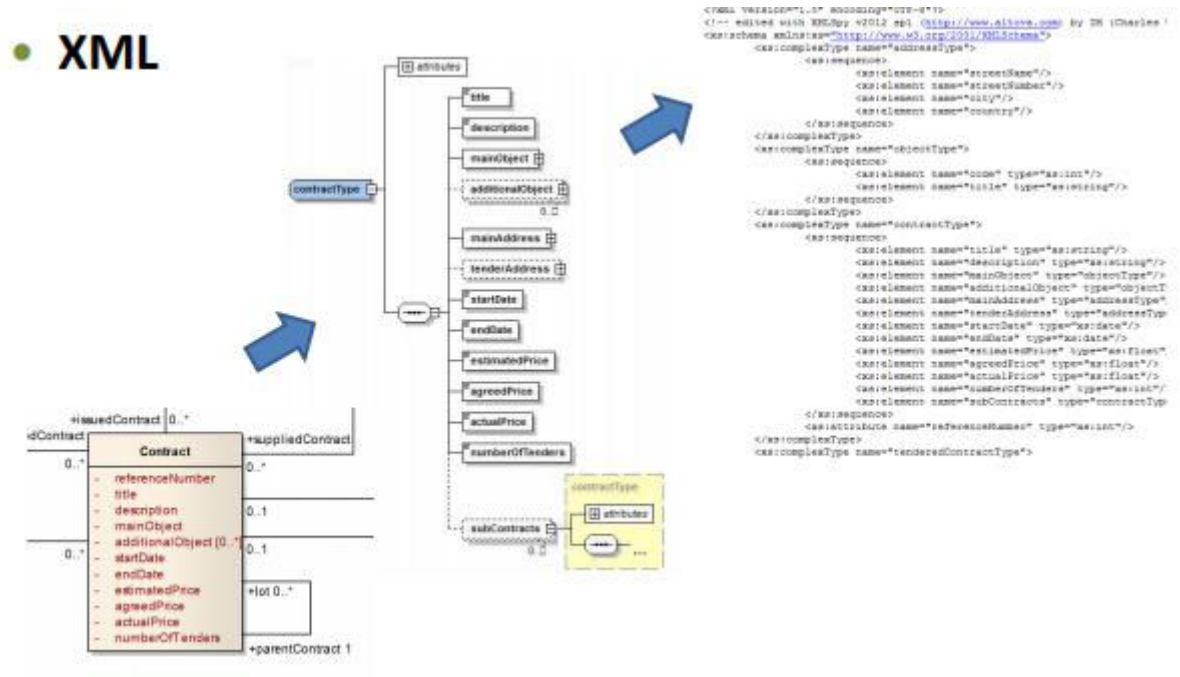
```
CREATE TABLE Contract (
  referenceNumber NUMBER(8) NOT NULL,
  title VARCHAR2(50) NOT NULL,
  description CLOB,
  startDate DATE NOT NULL,
  endDate DATE NOT NULL,
  estimatedPrice NUMBER(9) NOT NULL,
  ...
);

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

- kód byl vytvořen zcela automaticky z platformně-specifického diagramu

- **XML**



Relační model

- umožňuje ukládat entity, vztahy a jejich atributy v relacích

Definice a terminologie

Schéma relace

- popis relační struktury (vše kromě dat)
- ***S(A1:T1, A2:T2, ..., An:Tn)***
  - **S** = jméno schématu
  - **Ai** = jména atributů, **Ti** = jejich typy
  - specifikace typů se často vynechává
- **příklad**: Person(personalID, firstName, lastName)

Schéma relační databáze

- množina relačních schémat (+ integritní omezení,…)

Relace

- podmnožina kartézského součinu atributních domén Ti
- tzn. relace = množina
- prvky jsou nazývány tuples

Relační databáze

- množina relací

Základní požadavky

- **Atomicita** atributů
  - pro domény atributů mohou být použity jen jednoduché typy

- **Unikátnost** tuplů
  - relace je množinou, tzn. dva identické tuply nemohou existovat

- **Nedefinované pořadí**
  - relace je množinou, tuply tedy nemohou být seřazovány

- **Kompletnost** hodnot
  - v tuplech nejsou žádné díry, tzn. všechny hodnoty jsou specifikovány
  - mohou však být přidány NULL hodnoty

Integritní omezení

- **Identifikace** (každý tuple je identifikován jedním či více atributy)
  - **superklíč** = množina daných atributů (např. množina všech relačních atributů)

  - **klíč** = superklíč s minimálním počtem atributů
    - tzn. žádný atribut nemůže být odstraněn, aniž by byla narušena schopnost identifikace
    - v jedné relace může být vícero klíčů (klidně i s různým počtem atributů)
    - klíče jsou podtrhovány
    - Relation(<u>Key</u>, <u>CompositeKeyPart1, CompositeKeyPart2</u>,…)

- **Referenční integrita**
  - **cizí klíč** = množina atributů odkazující relace, které korespondují se (super)klíči odkazované relace

    $ReferencingTable.foreignKey \subseteq ReferencedTable.Key$

    $foreignKey \subseteq ReferencedTable.Key$

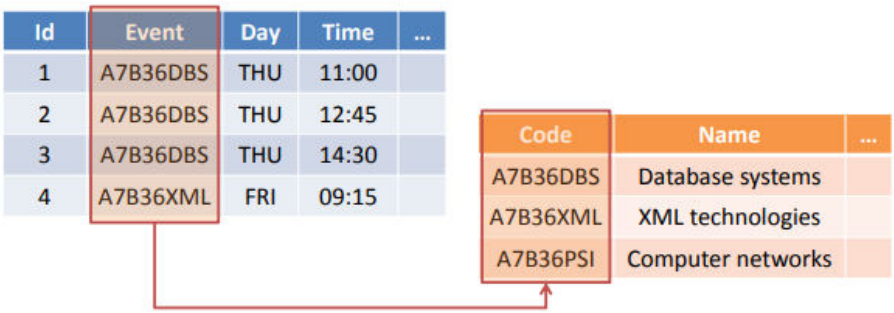
Příklad relační databáze

- Schema

  Course(Code, Name, ...)

  Schedule(Id, 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	

Relace vs. tabulky

- záhlaví tabulky ~ relační schéma
- řádka ~ tuple
- sloupec ~ atribut

- nicméně…
  - tabulky nejsou množiny, takže…
    - v tabulkách mohou být duplicitní řádky
    - v tabulkách mohou být řádky uspořádané
  - sQL a existující RDBMS nedodržují vždy striktně formální relační model

Objekt vs. (Objektově-)Relační Model

Relační model

- data ukládána do plochých tabulek
- vhodné pro datově-intensivní dávkové operace

Objektový model

- data ukládána jako grafy objektů
- vhodné pro individuální navigační přístup k entitám

Objektově-relační model

- relační model obohacen o objektové elementy
- atributy mohou být komplexních datových typů
- metody mohou být definovány i na datových typech

Transformace UML/ER na relační model

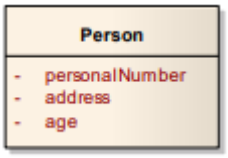
- Co máme:
  - ER: entitní typy, atributy, identifikátory, vztahové typy, ISA hierarchie
  - UML: třídy, atributy, asociace

- Co potřebujeme:
    - schéma relací s atributy, klíči a cizími klíči

- Jak toho dosáhnout:
    - třídy s atributy => relační schémata
    - asociace => samostatná relační schémata nebo společně se třídami

Třídy

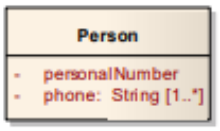
- separátní tabulka
    - Person(<u>personalNumber</u>, address, age)



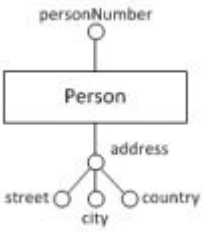
- **Artificial klíče**
    - uměle přidané integerové identifikátory
    - bez odkazu na reálný svět
    - … ale s pár efektivními a designovými výhodami
    - většinou automaticky generované a přiřazené
    - Person(<u>personID</u>, <u>personNumber</u>, address, age)

Atributy

- **multihodnotové atributy**
  => separátní tabulky
    - Person(<u>personalNumber</u>)
    - Phone(<u>personalNumber, phone</u>)
    - Phone.personalNumber ⊆ Person.personalNumber



- **složené atributy**
  => separátní tabulky
    - Person(<u>personalNumber</u>)
    - Address(<u>personalNumber</u>, street, city, country)
    - Address.personalNumber ⊆ Person.personalNumber
    - sub-atributy mohou být také inlineovány, ale jen s (1,1) kardinalitou
        - Person(<u>personNuber</u>, street, coty, country)



Multiplicity (1,1):(1,1) →

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

Multiplicity (1,1):(1,1) →

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

Multiplicity (1,1):(0,1) →

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

Multiplicity (0,1):(0,1) →

- Three tables
  - Person(<u>personalNumber</u>, address, age)
    Mobile(<u>serialNumber</u>, color)
    Ownership(<u>personalNumber</u>, <u>serialNumber</u>)
    Ownership.personalNumber ⊆ Person.personalNumber
    Ownership.serialNumber ⊆ Mobile.serialNumber
  - Note that a personal number and serial number are both independent keys in the Ownership table

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

Multiplicity (1,n)/(0,n):(0,1) →

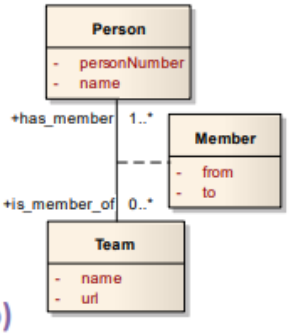
- Three tables
  - Person(<u>personalNumber</u>, address, age)
    Mobile(<u>serialNumber</u>, color)
    Ownership(personalNumber, <u>serialNumber</u>)
    Ownership.personalNumber ⊆ Person.personalNumber
    Ownership.serialNumber ⊆ Mobile.serialNumber
  - Why a personal number is not a key in the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

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

Atributy asociací

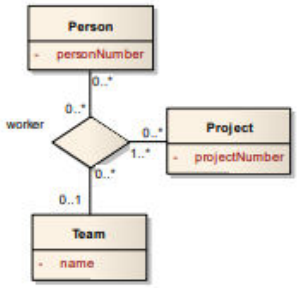
Attribute of an association →



- 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

Obecné asociace

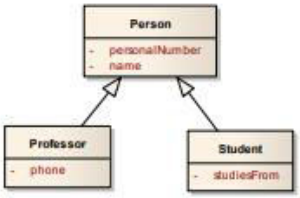
N-ary association →



- 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…

Hierarchie

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)

- **jen jedna tabulka pro zdroj hierarchie**
  - Person(personalNumber, name, phone, studiesFrom, type)
  - univerzální, ale ne vždy vhodné
    - typy instancí se rozlišují dle umělého atributu
  - **výhoda**: žádné joiny
  - **nevýhoda**: vyžadovány NULL hodnoty

- **separátní tabulky pro každý listový typ** (na konci hierarchie)
  - Professor(personalNumber, name, phone)
  - Student(personalNumber, name, studiesFrom)
  - není vždy aplikovatelné
  - **výhoda**: žádné joiny
  - **nevýhody**:
    - redundance (když failuje covering constraint)
    - integritní omezení (unikátnost personalNumber)

Slabé entitní typy

- separátní tabulka
  - Institution(name)
  - Team(code, name)
  - Team.name ⊆ Institution.name
- kardinalita musí být vždy (1,1)
  - klíč slabých entitních typů zahrnuje i klíč z entit, na nichž závisí