# Datové modelování

Modely a reprezentace strukturovaných dat

**doc. Ing. Radek Burget, Ph.D.**
burgetr@fit.vutbr.cz

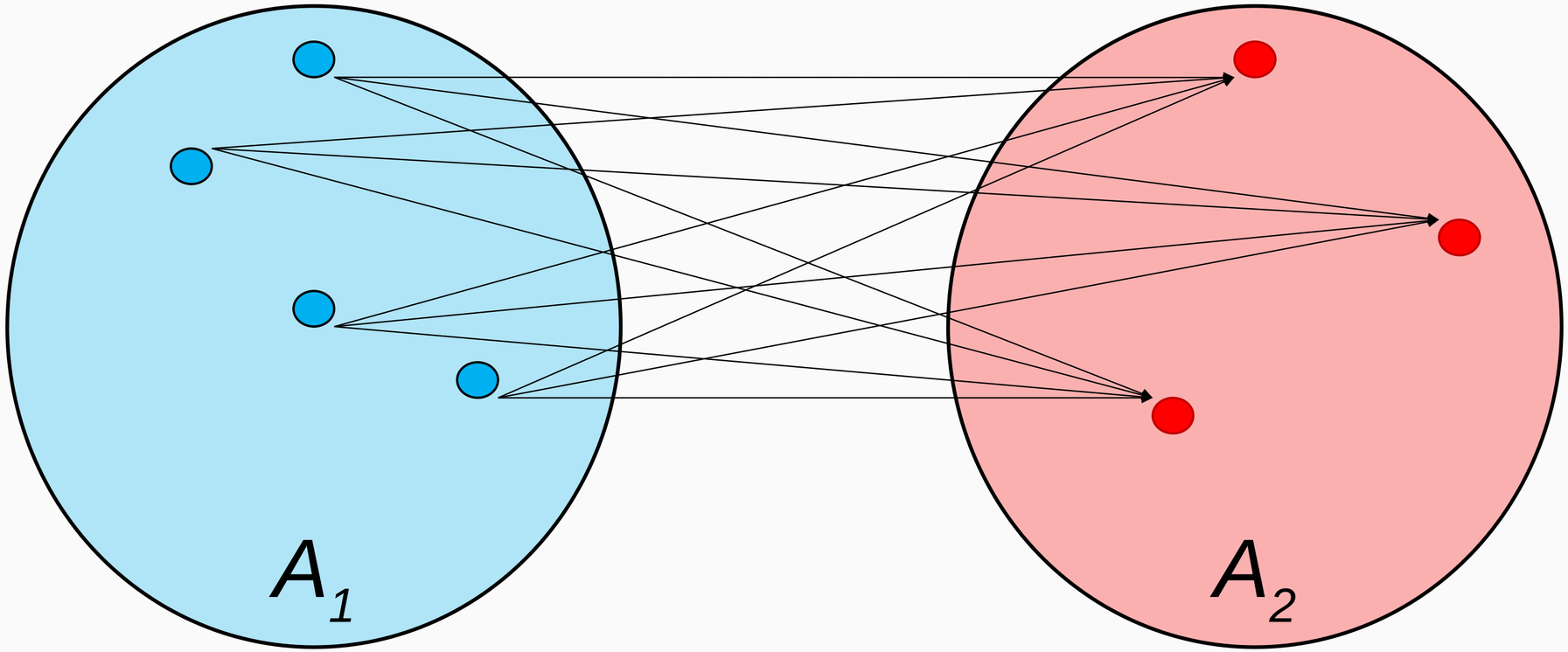
# Strukturovaná data

- Strukturovaná data se v IS vyskytují v různých kontextech
  - Persistentní úložiště – **databáze**
  - Soubory, HTTP komunikace – **serializace**
  - Uživatelské rozhraní – **vizualizace**
- Způsob reprezentace dat vychází z **modelování**
  - Abstrakce modelované reality
- Jak lze *obecně popsat* a následně *reprezentovat* strukturovaná data?

# Strukturovaná data

Jak vyrobit z jednoduchých údajů složitý?

# Kartézský součin A1 x A2




Výsledkem je **množina dvojic** – celkem $4 \times 3 = 12$ dvojic.

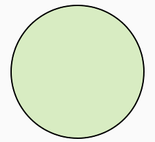
# Kartézský součin

- *Uspořádaná n-tice* $(a_1, a_2, \ldots, a_n)$
- *Kartézský součin množin* $A_1 \times A_2 \times \cdots \times A_n$ je množina všech uspořádaných n-tic takových, že $a_1 \in A_1, a_2 \in A_2, \ldots, a_n \in A_n$
- Podstatné je, že v uspořádané n-tici je každá hodnota prvkem jediné z množin v kartézském součinu a to té, která jí indexem odpovídá

# Struktura a kolekce

# Základní typy

- Základní nestrukturované datové typy:
  - Celočíselné
  - Reálná čísla
  - Znaky / řetězce
  - Datum / čas
  - Výčtové typy apod.



# Strukturované datové typy

- Strukturovaný datový typ = datová struktura = *metadata*
  - Jak z jednodušších datových typů (ať už základních nebo i jednodušších strukturovaných) budovat složitější.
- Existují základní dva způsoby, jak strukturované datové typy vytvářet:
  - *struktura* a
  - *kolekce*.
- Vše je definováno předem, před vznikem hodnoty

# Struktura

- Strukturované hodnoty vytvářené:
  - **Pevným počtem** dílčích hodnot obecně **různých** typů
  - Tedy uspořádané n-tice, které jsou prvky kartézského součinu množin dílčích datových typů
  - Hodnoty jsou pojmenované, tzn. přistupuje se k nim přes jejich unikátní jméno
- Jako synonymum pro uspořádanou n-tici (tedy hodnotu) je velmi často užíván termín *struktura* nebo *záznam*. Jako synonymum pro kartézský součin (tedy datový typ) budeme často používat *typ struktura* nebo *typ záznam*.

# Schéma struktury

# Příklad datového typu struktura

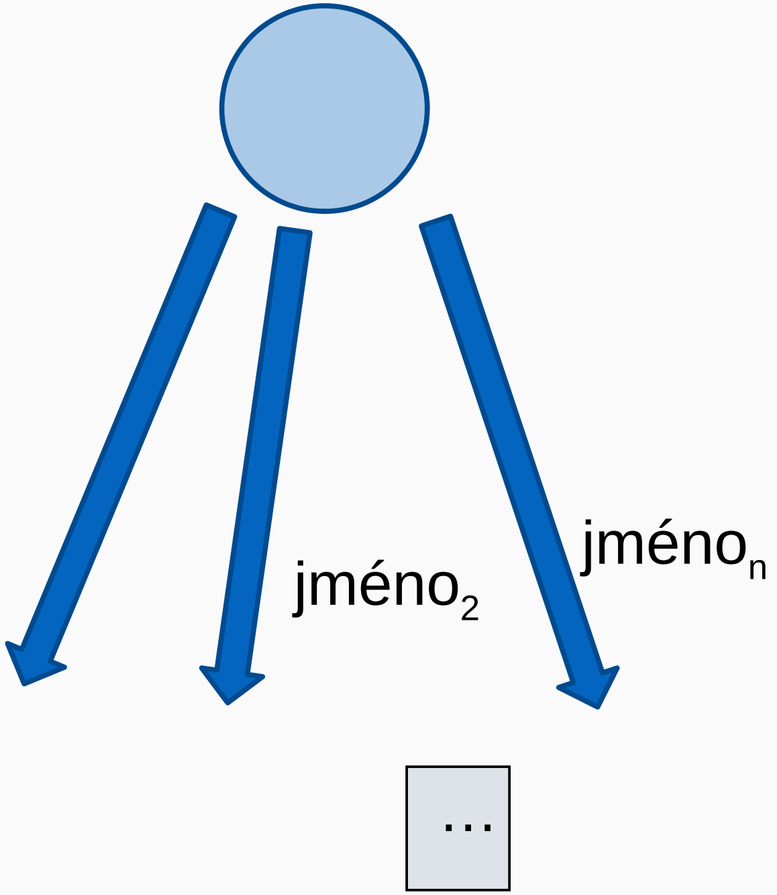
```
structure FyzOsoba
   properties
      UplneJmeno:    string
      Jmeno:         string
      Prijmeni:      string
      DatumNaroz:    date
   end structure
```

Definujeme *metadata*.

# Hodnota struktury

| **jména vlastností** | **hodnoty vlastností** |
|---|---|
| UplneJmeno: | "Prof. Ing. Jan Novák, CSc." |
| Jmeno: | "Jan" |
| Prijmeni: | "Novák" |
| DatumNaroz: | 24.5.1954 |

jméno $vlastnosti_1$



# Kolekce

- *Kolekce* (synonyma jsou *řetězec*, *posloupnost*, *seznam*, *soubor*) je, na rozdíl od struktur, tvořena
  - ***Předem neomezeným počtem*** *hodnot* ***stejných*** *datových typů.*

# Kolekce

- Množina obsahuje obvykle každý prvek pouze jednou. Pokud je povoleno, aby daný prvek byl v množině vícekrát, mluvíme o *multimnožině*
- Tradiční seznam je *uspořádanou multimnožinou*
- Obecně lze vytvářet kolekce s prvky libovolných datových typů.
- Časté omezení je vytvářet *pouze kolekce s prvky datového typu struktura*

# Operace nad množinou

- Vkládání prvku do kolekce (*add*),
- Získání prvku z kolekce (*item*),
- Určení počtu prvků kolekce (*count*) a
- Rušení prvku kolekce (*remove*)

případně

- Provádění operací nad všemi prvky (*forall*)

# Vlastnosti kolekce

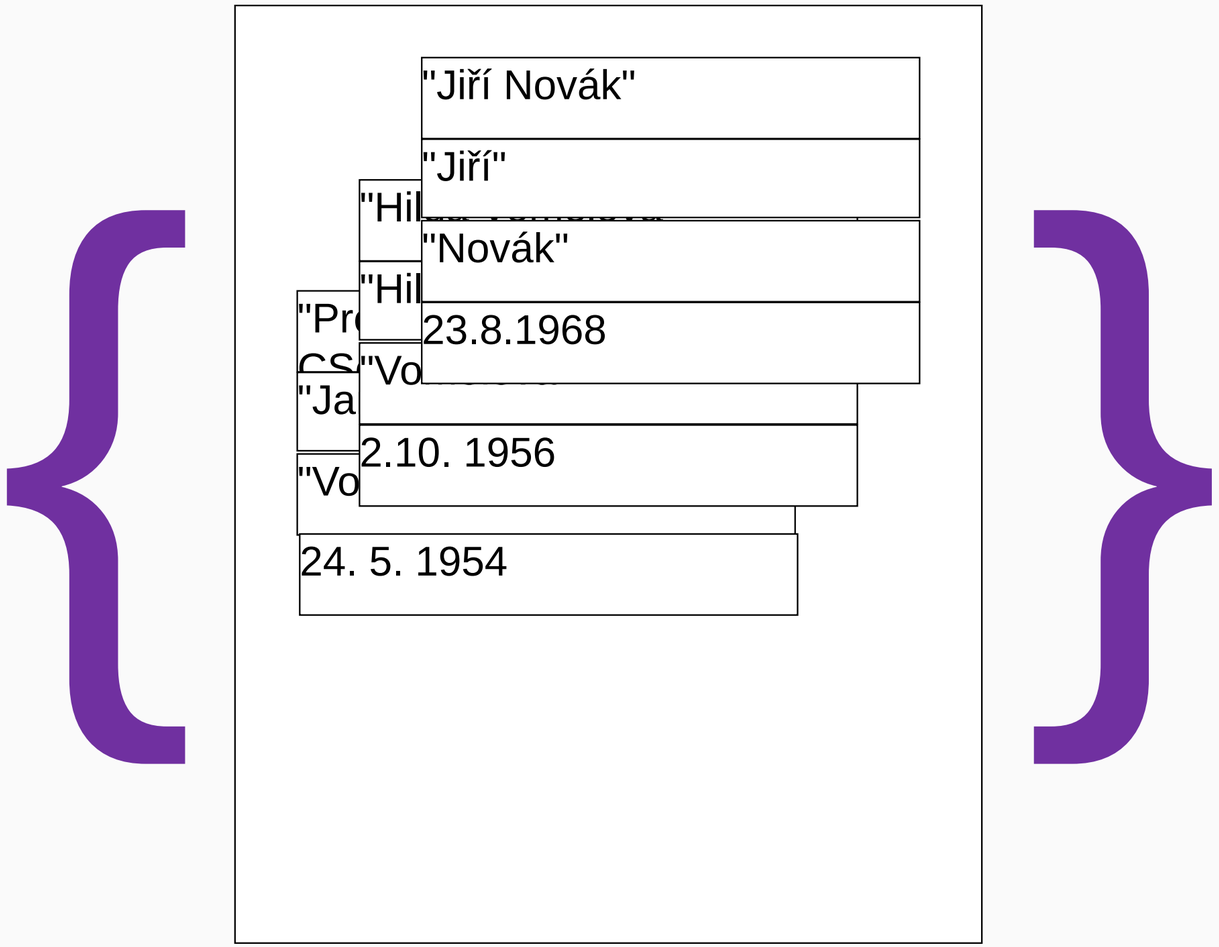
- *Kurzor (**iterator**)*, což je ukazovátko do kolekce, kterým lze posunovat oběma směry a nastavovat je do různých pozic v kolekci podle různých kriterií.
- Protože v průběhu práce s kurzorem se může kolekce měnit co do obsahu i počtu prvků, dělíme kurzory na *stabilní*, které na tuto skutečnost neberou zřetel a *nestabilní*, které reflektují změny
- Nad kolekcí může existovat jedno nebo více definovaných *uspořádání* jejich prvků podle různých klíčů.

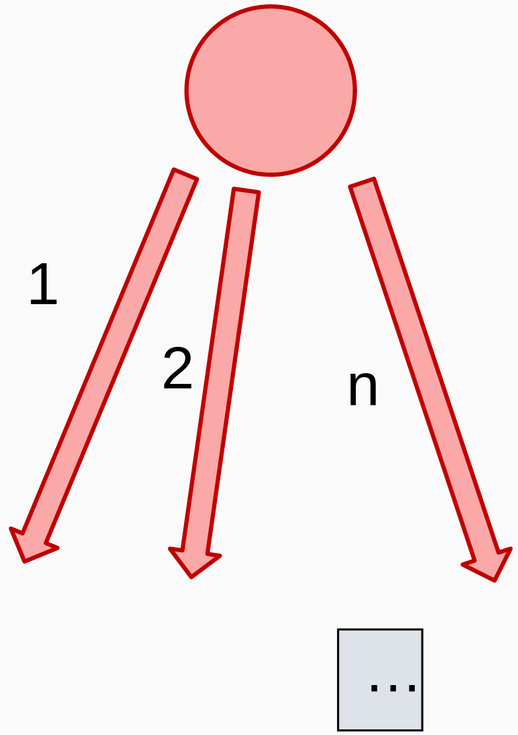
# Schéma kolekce

# Příklad datového typu kolekce struktur

```
collection FyzickeOsoby of
    structure FyzOsoba
       properties
        UplneJmeno: string
        Jmeno:      string
        Prijmeni:   string
        DatumNaroz: date
    end structure
```

# Hodnota kolekce



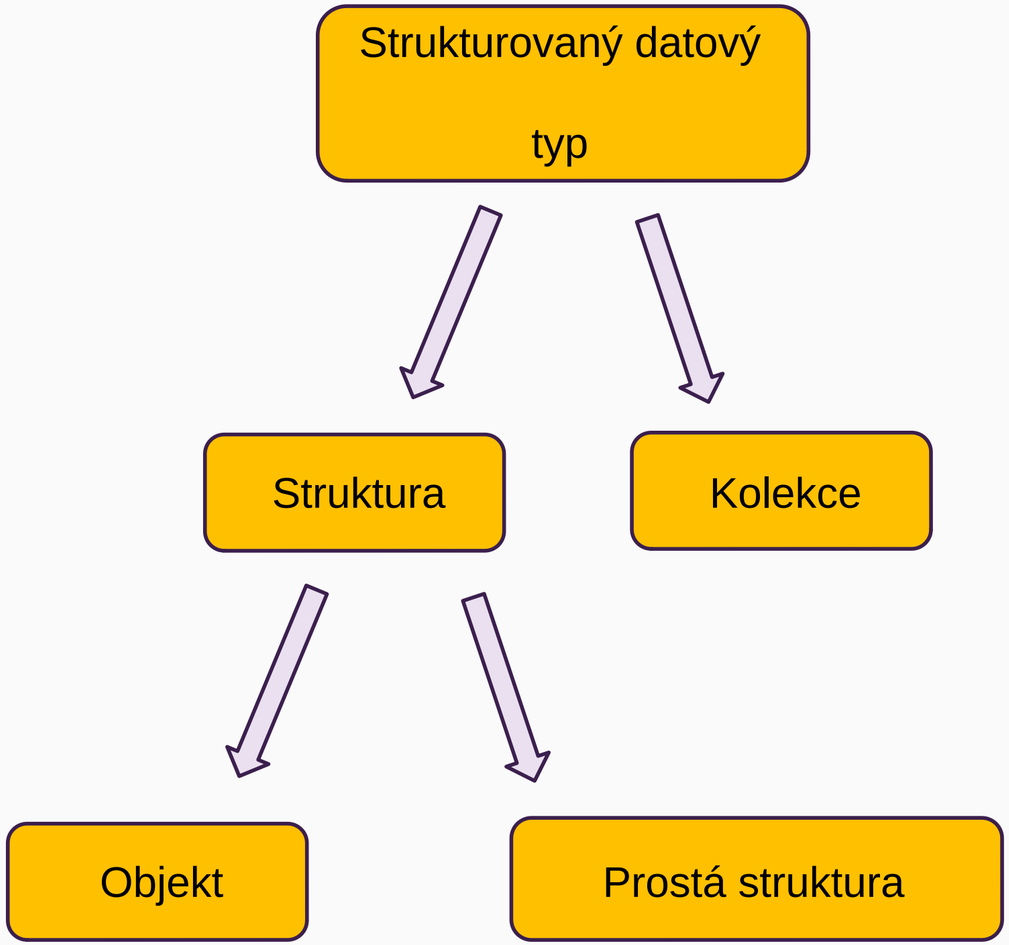
# Agregáty

- Vlastnostmi kolekce jsou nejčastěji *agregáty (agregované hodnoty)*, což jsou hodnoty statisticky popisující prvky *kolekce* nejčastěji *číselných hodnot*.
  - *počet prvků,*
  - *maximum*,
  - *minimum*,
  - *součet hodnot*,
  - *průměr* atd.

# Objekt a prostá struktura

- *Objekt je struktura s identifikací.*
- Každému objektu v systému přiřazena *jednoznačná identifikace* nazývaná *OID* (***object identification***).
- Objekt je tedy *struktura, jejíž systémovou a obvykle první vlastností je OID*. Hodnotu OID generuje databázový systém při vzniku objektu a po celou dobu činnosti ji nemění.
- Tím, že má objekt OID, je *identifikovatelný* a tudíž i *odkazovatelný*. Má to za následek, že může figurovat jako *člen* ve *vztazích*. To struktura bez identifikace nemůže. Takovou strukturu bez OID budeme nadále nazývat *prostou strukturou*.

# Strukturované datové typy



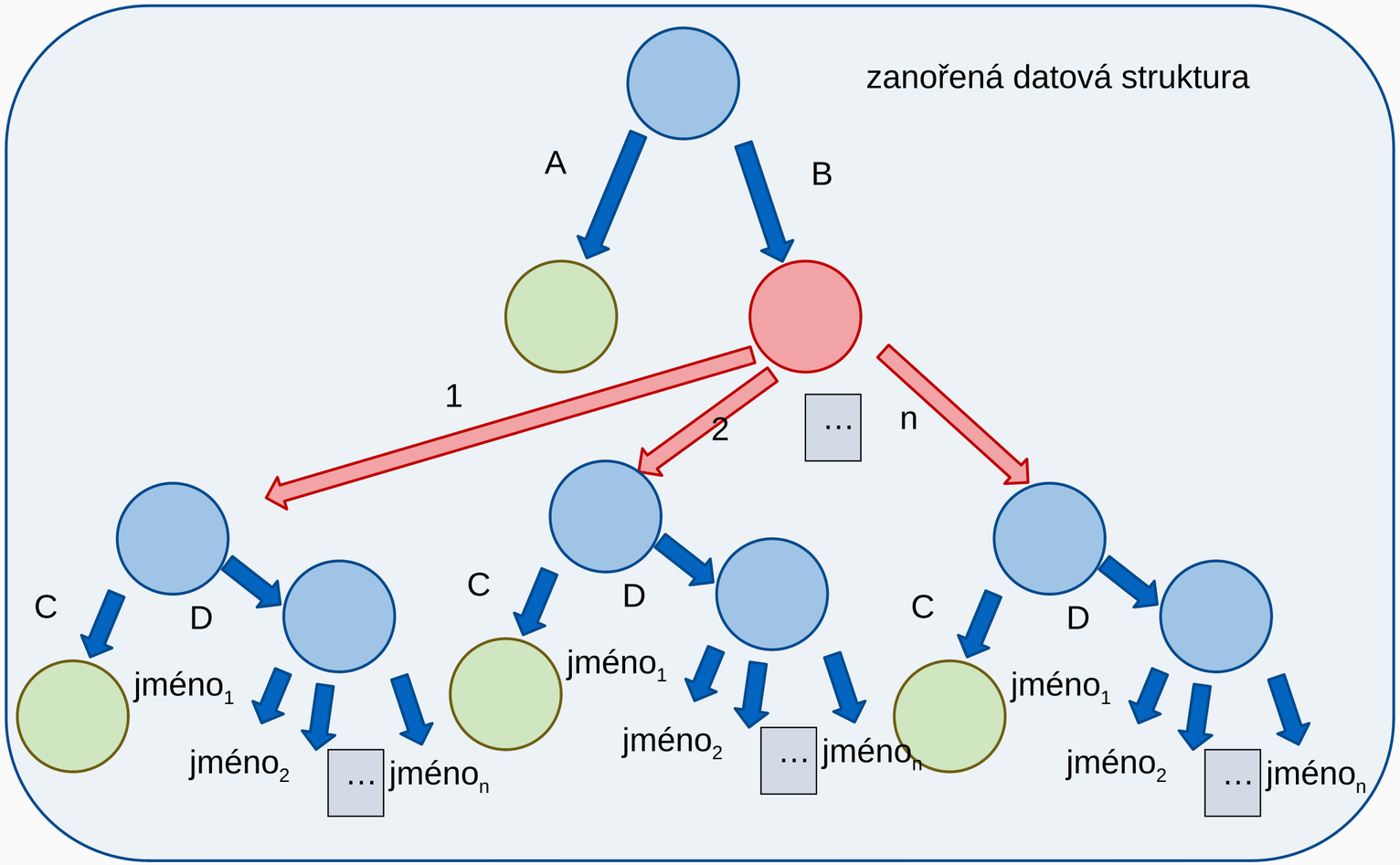
# Zanořené kolekce a struktury

- Obecně lze struktury a kolekce libovolně vzájemné vnořovat

```
structure ZANORENA
   properties
      A: integer
      B: collection of structure
            properties
               C: integer
               D: structure
               …
                  end structure

         end structure
end structure
```

# Graf hodnoty zanořených typů



# Datové modelování

Od struktury a kolekce k modelům

# Data, metadata, atd.

- **Data**
  - Konkrétní hodnota (*výskyt*)
  - Např. "Jan Novák", "1250 Kč", … (i strukturované hodnoty)
- **Metadata** = jak vypadají data
  - Formální popis struktury dat
  - Záleží na datovém modelu:
    - Relační: definice struktury tabulek (relací)
    - Objektový: definice tříd
    - …

# Další úrovně

- Každou úroveň metadat lze opět popsat
- **meta-metadata = meta$^2$data** = jak vypadají metadata
  - Popis datového modelu
  - Např. relační: relace je kolekce struktur, SQL jako prostředek definice metadat, …
- **meta$^3$data**
  - Jak se popisuje databázový model – kolekce, struktura, …
- … atd.

# Cíle modelování

- Zobrazit modelovanou realitu
  - Zjednodušení reality pro potřeby návrhu IS – abstrakce
  - Konzultace s odborníky na cílovou doménu – spolupráce
- Vytvořit popis pomocí prostředků příslušného datového modelu
  - Vytvořit **metadata** – implementace
  - Různé prostředky podle způsobu použití

# Databázové modely

- Modely, které je schopen interpretovat systém pro řízení databázového systému SŘBD
- Jinak též zvané **produkční modely**
- V jejich definičním jazyku musejí být zapsána **metadata** pro všechny datové struktury uložené v databázi
- Prozatím budeme uvažovat jako produkční **relační a objektový datový model**.

# Konceptuální modely

- Slouží pro komunikaci mezi návrháři, případně se zákazníky
- Jsou formálně přesné a **převoditelné** na produkční modely
- Často jsou grafické pro větší přehlednost
- Nejběžnější konceptuální modely **diagram tříd (UML)** a **E-R diagram**

# Transformace mezi datovými modely

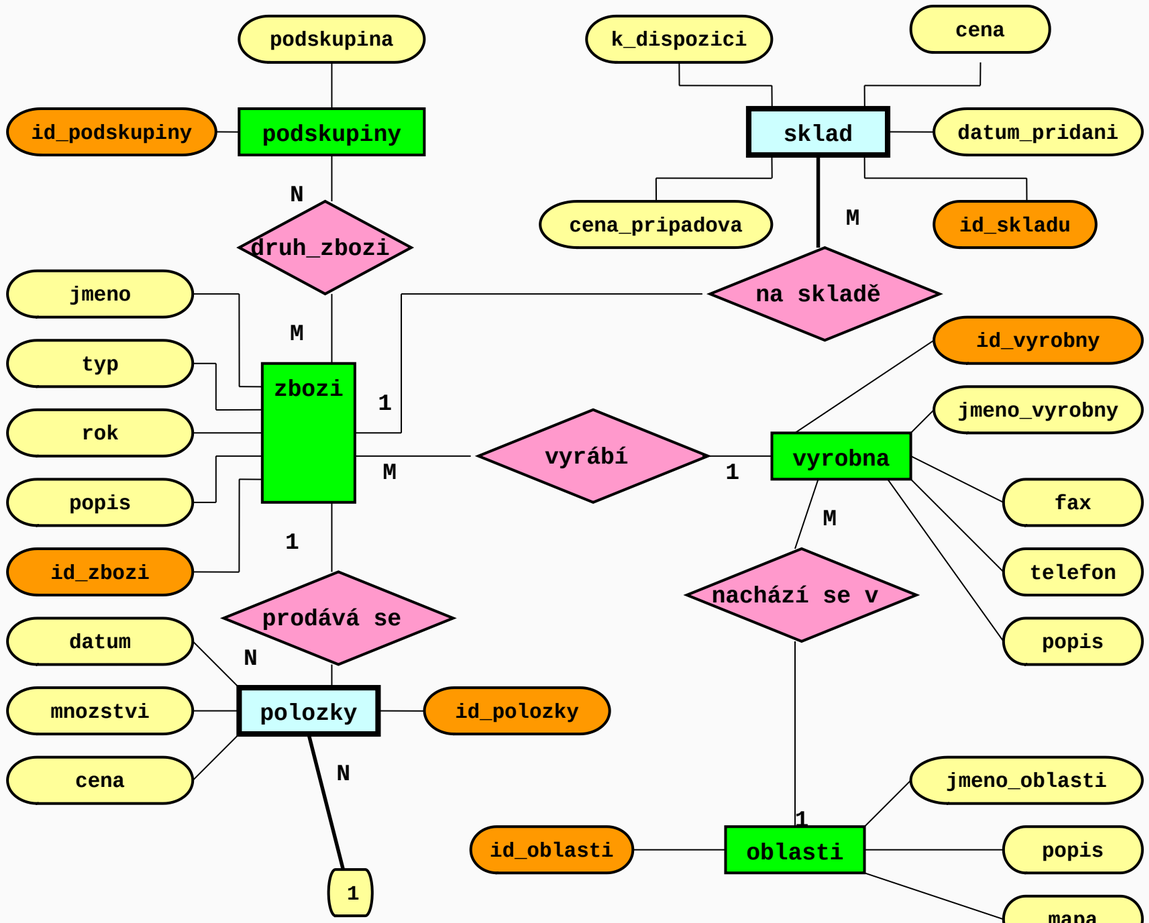
- Slouží nejčastěji pro transformaci konceptuálních modelů na produkční.
- Transformace je tím složitější, čím jsou modely více sémanticky odlišné.
- Nejčastěji se uvažuje **transformace E-R diagramu na relační datový model**.

# Konceptuální modely

# Entity-relationship (E-R) diagram

- Hlavní komponenty E-R diagramu jsou:
  - **entita** a
  - **vztah** (***relationship*** nikoliv ***relation***).
- **Entity** modelují objekty, které se vyskytují v modelovaném fyzickém systému
  - např. studenti, profesoři, předměty na vysoké škole
  - a jejich **atributy**
- **Vztahy** modelují spojení mezi entitami – například profesoři *učí* předměty.
- Navíc tvoří důležitou část E-R specifikace **integritních omezení** na entitách a vztazích, např. profesor učí **pouze jeden předmět** v daném čase.

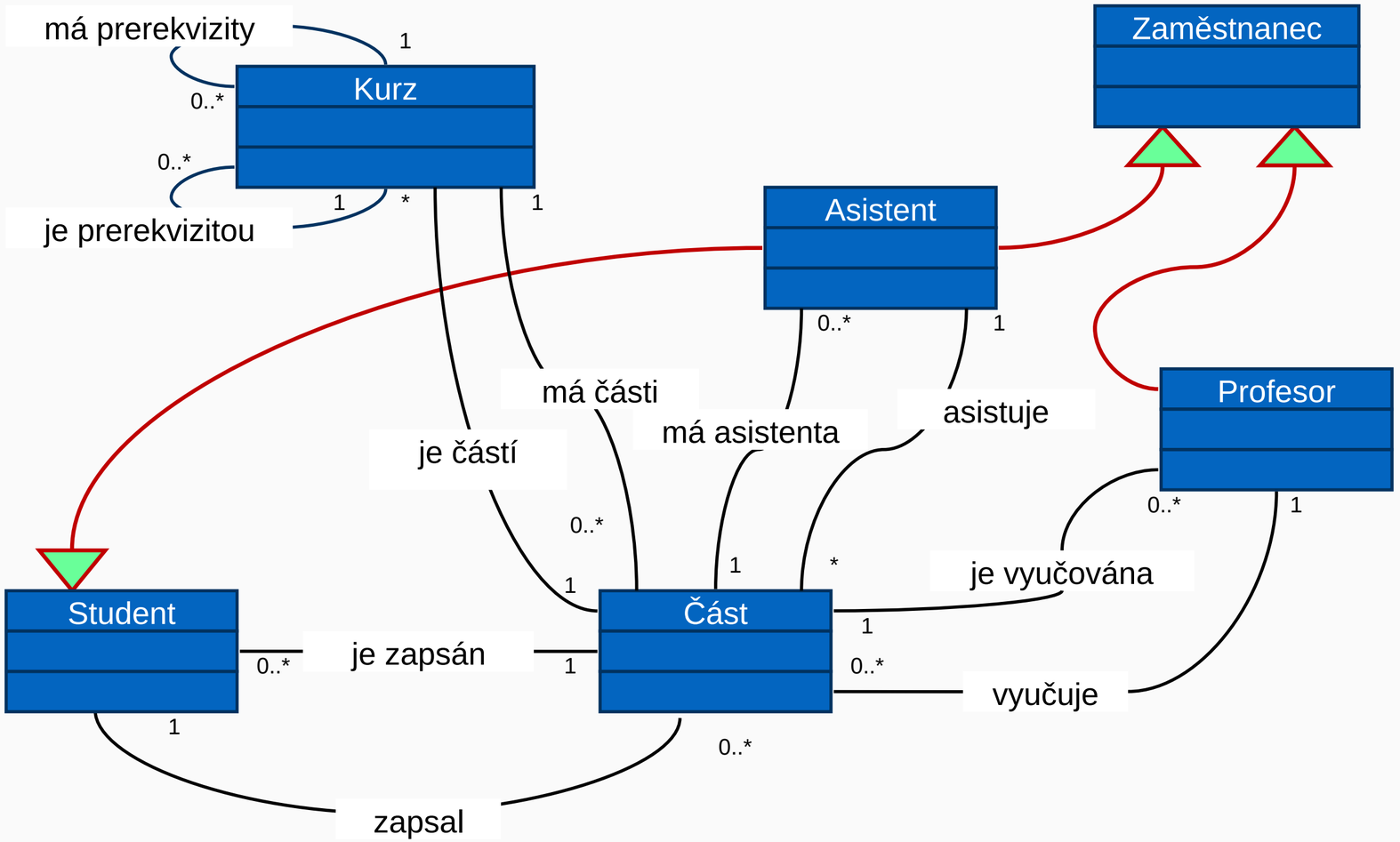
# Příklad E-R diagramu



# Diagram tříd

- **Třídy**
  - Jejich **vlastnosti** (jméno a datový typ)
- **Vztahy**
  - S různou kardinalitou
- **Dědičnost**
  - Generalizace – specializace

# Příklad diagramu tříd



# Databázové modely

# Relační model dat

- Tabulka (= **relace**) v relačním modelu je *kolekcí struktur*, přičemž datové typy vlastností jsou *jednoduché* (tedy především *ne odkazy/vztahy*)
- Srovnej: *Podmnožina kartézského součinu*

```
collection of
    structure
        properties
            jméno vlastnosti1: jednoduchý datový typ1
            jméno vlastnosti2: jednoduchý datový typ2
            …
            jméno vlastnostin: jednoduchý datový typn
    end structure
```

# Vztahy

- Umožňují odkazovat z jedné (strukturované) hodnoty (vlastníka) jinou (člen)
- Musí existovat datový typ *jednoznačné identifikující (odkazující) strukturovanou hodnotu* (např. OID)
- Vztah je definován prvkem vlastníka typu odkaz (reference) a členem, který je hodnotou odkazu identifikován.

# Vztahy

- Relační model dat vztahy přímo neobsahuje
  - Vytváří se až v okamžiku dotazování (JOIN apod.)
  - (Neplést s referenční integritou!)
- Objektový model
  - Vztahy lze tvořit pomocí OID

# Objektový model dat

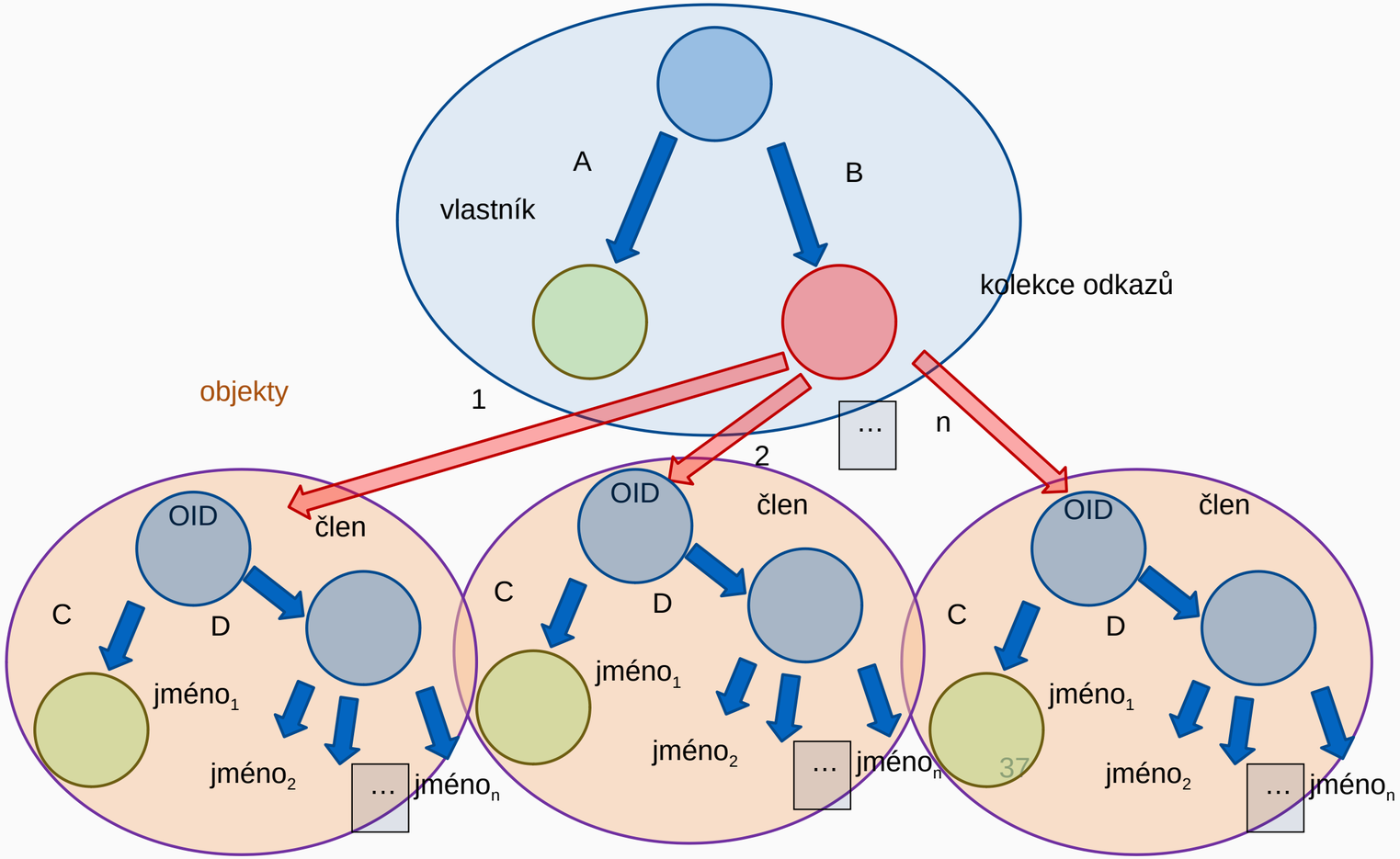
- Základní typy + datový typ OID
- Objekt je vždy strukturou na nejvyšší úrovni
- Dva druhy neomezeně zanořených struktur
  - Kolekce (někdy omezení pouze na kolekce prostých struktur a OID)
  - Prosté struktury (ostatní)
- Další vlastnosti zde neřešené (dědičnost apod.)
- **Odpadá nutnost transformace objektového modelu na schéma relační databáze**

# Odkazované struktury (objekty)

```
structure VLASTNIK
   properties
      A: integer
      B: CLEN
end structure

object CLEN
   properties
      C: integer
      D: structure
      …
end object
```

# Graf hodnoty odkazovaných typů



# Objektově-relační mapování (ORM)

- Čistě objektové databáze se v praxi vyskytují minimálně
- Relační databáze jsou oproti tomu rozšířené, výkonné a odladěné
- ORM je databázová vrstva zajišťující automatické mapování objektů na relace a zpět
  - Program pracuje s objekty (např. uživatelé, smlouvy, …)
  - ORM vrstva vytváří SQL dotazy, transformuje data
- Zajišťuje konzistenci aplikace a schématu databáze
  - Možnost automatické tvorby schématu databáze apod.
  - Snazší úpravy aplikace

# ORM řešení

- Java
  - Standardní aplikační rozhraní JPA, Hibernate, …
- PHP
  - Např. knihovna Doctrine
  - Použití do značné míry shodné s JPA
- Obdobně na dalších platformách
  - .NET, Python, JavaScript, …
- Více v předmětu Pokročilé informační systémy

# Transformace modelů

- Mezi modely mohou existovat transformace, zejména, pokud jsou si *sémanticky blízké*
- Velmi častou je transformace **E-R diagramu na relační datový model**
  - viz. postup z IDS

# Jiné transformace

- Diagram tříd lze transformovat na relační model obdobně jako E-R diagram
  - Je nutno řešit dědičnost – různé strategie
- Pro objektový databázový model lze přímo použít diagram tříd
  - V modelu se přímo definují třídy a jejich vlastnosti
  - Vztahy jsou reprezentovány vlastnostmi tříd
  - Odkazy na jiné objekty
    - např. `zbozi.vyrobna`

# Závěr

- **Ze struktur, kolekcí a základních typů můžeme vytvořit známé produkční i konceptuální modely**
- Reprezentace kolekcí a struktur:
  - Textová reprezentace ve formálních jazycích – **serializace** (1D) – komunikace mezi složkami systému
  - Grafická reprezentace - **vizualizace** (2D) – vstup a výstup systému

# A to je vše!

Dotazy?