Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Semestrální práce z předmětu Objektové a relační databáze

Databáze seriálů

Čtvrtek lichý, 14:00

David Pocar

2020

Obsah

Seznam diagramů

1 Úvod

Cílem projektu je evidovat seriály a s nimi související informace. Problematika byla modelována dvěma způsoby – objektovým a relačním. První způsob byl realizován pomocí velice zajímavého jazyka Smalltalk prostřednictvím programu *Daskalos*. Druhým způsobem bylo provedení pomocí relační databáze MariaDB (MySQL).

2 Popis modelu

V rámci dané problematiky byly definovány následující entity: *Adresa, Epizoda, Herec, Lokalita, Ocenění, Osoba, Postava, Režisér, Scénárista, Seriál, Série, Společnost, Žánr*.

Společnost může vlastnit *Seriály*. Každá *Společnost* má právě jednu *Adresu* (sídlo). Metodou této třídy je zjištění nejlépe hodnoceného *Seriálu* (vlastní).

Lokalita slouží ke spojení *Epizody* a *Adresy*, přičemž této vazbě přidává nové atributy v podobě počasí během natáčení a vynaložené ceny za možnost využití *Lokality*.

Adresa obsahuje údaje o poloze a je přiřazována k *Lokalitě* a *Společnosti*.

Epizoda obsahuje informace o konkrétním odvysílaném dílu *Seriálu*. Každá *Epizoda* spadá do konkrétní *Série*. Každá *Epizoda* má přesně jednoho *Režiséra*. Dále také jednoho a více *Herců*, *Scénáristů* a *Lokalit* natáčení. *Epizoda* obsahuje rovněž metodu pro zjištění umístění *Epizody* v žebříčku (dle hodnocení) všech *Epizod*, napříč všemi *Seriály*.

Herec dědí z třídy *Osoba*. V každé *Epizodě* se nachází alespoň jedna instance *Herce*, ale ne každý *Herec* hraje v každé *Epizodě*. Disponuje metodou pro zjištění, pod kterými režiséry *Herec* pracoval nebo pracuje.

Ocenění je forma odměny za vynaložené úsilí. Instance této třídy se mohou vyskytovat u instancí *Osoby* a *Seriálu*. Samotné ocenění má atributy pro rozlišnosti, jako název, popis a rok.

Osoba je třída, z níž dědí *Režiséři*, *Herci* a *Scénáristi*. Obsahuje základní informace o každé evidované osobě (jméno, příjmení, datum narození, ...). Disponuje také metodou pro zjištění počtu výskytu dané osoby v *Epizodách* a výpočet celkové mzdy.

Režisér, stejně jako *Herec* dědí veškeré vlastnosti *Osoby*. Přidává však jednu metodu s funkcí pro získání všech *Osob*, které kdy *Režisér* vedl při režírování *Epizody*. Právě jedna instance

Režiséra se může nacházet u *Epizody*, přičemž ne každý *Režisér* musí ve skutečnosti něco režírovat.

Scénárista je další třídou dědící vlastnosti z *Osoby*. Obsahuje atribut pro průměrný počet stran, které napíše pro jednu *Epizodu*. Dále má tento objekt metodu pro zjištění celkového počtu stránek, které daný *Scénárista* průměrně napsal (za všechny *Epizody*).

Seriál obsahuje atributy název, popis a datum zahájení. Každý *Seriál* má jednu a více *Sérií* a *Žánrů*. *Seriál* může spadat právě pod jednu *Společnost*. Stejně jako *Osoba*, může i *Seriál* mít nějaké *Ocenění*. *Seriál* disponuje čtyřmi metodami. První metoda slouží k zjištění celkového počtu *Epizod* seriálu. Druhá metoda (doba trvání) slouží k zjištění celkového času, potřebného pro zhlédnutí celého *Seriálu* v aktuální podobě. Další metoda vrací všechny země, kde byl *Seriál* natáčen a poslední metoda vrací hodnocení celého *Seriálu* dle hodnocení konkrétních *Epizod*.

Série obsahuje atributy pro označení čísla *Série* a datum zahájení vysílání. Každá *série* spadá právě pod jeden seriál, přičemž každá *Série* obsahuje jednu a více *Epizod*. Obsahuje dvě metody jako *Seriál*, a to doba trvání a hodnocení. Rozdílem je, že zahrnuje pouze epizody dané *Série*, a nikoliv celého *Seriálu*. Obsahuje také metodu, pro zjištění nejlepší *Epizody*.

Každý *Seriál* má jeden a více *Žánrů*, přičemž třída *Žánr* obsahuje pouze atribut název pro označení.

2.1 UML diagram tříd

Na diagramu tříd jsou znázorněny všechny vztahy mezi zmíněnými entitami. V diagramu se například vyskytuje generalizace mezi třídou Osobou a třemi podtřídami Herec, Režisér a Scénárista, což naznačuje dědění. Dále je využita kompozice mezi Seriálem, Sérií a Epizodou.

UML diagram tříd byl navržen prostřednictvím Lucidchart pod studentskou licencí.

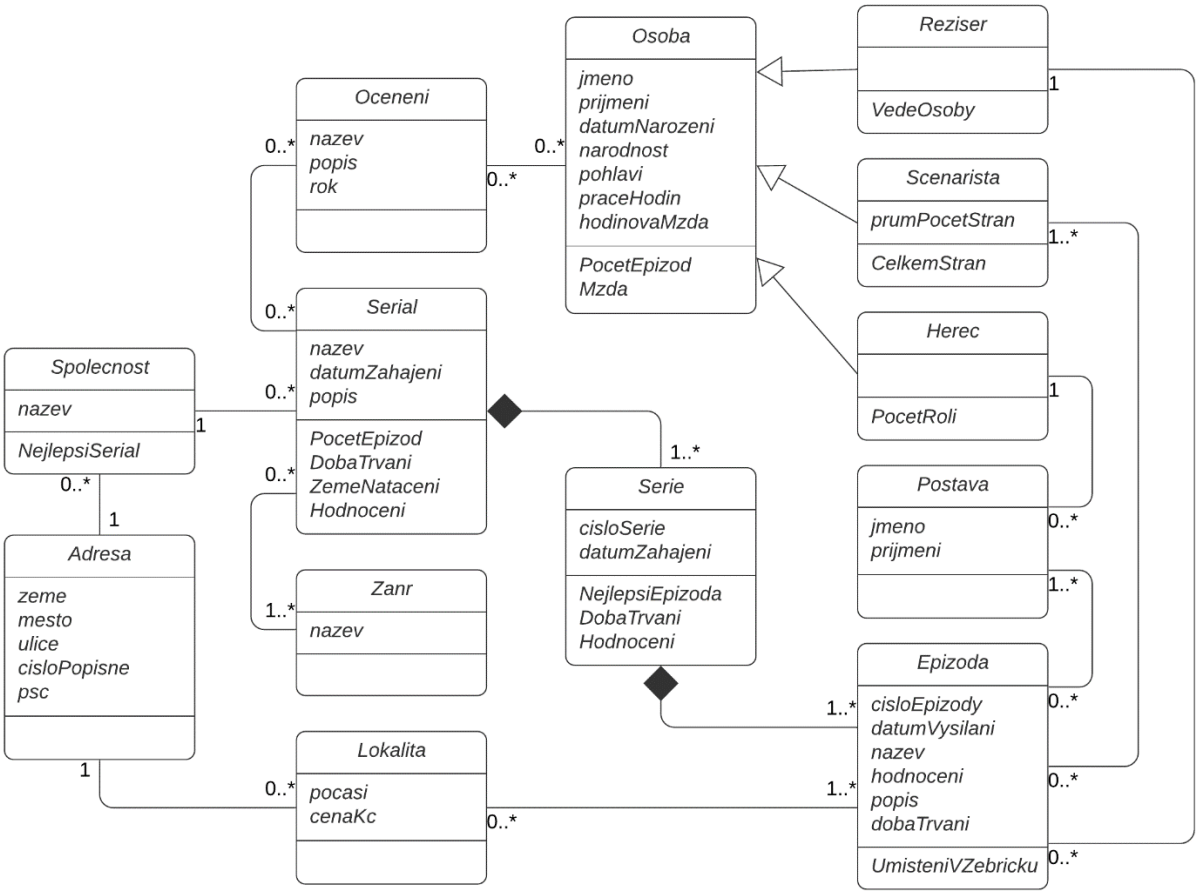

Diagram 1 – UML diagram tříd

3 Implementace

Implementace dané problematiky byla provedena dvěma způsoby – objektově a relačně.

Objektový model byl implementován prostřednictvím objektově orientovaného jazyka Smalltalk v programu *Daskalos*.

Implementace relačního modelu byla provedena za pomocí dotazovacího jazyku SQL nad MariaDB (MySQL) databází, a to prostřednictvím vývojářských nástrojů *JetBrains DataGrip*, *MySQL Workbench* a *Adminer*.

Oproti objektovému modelu má struktura relačního modelu drobné změny mezi entitami – většinou se jednalo o přidání spojovacích tabulek pro realizaci N:M vztahu. Podstatnější změnou byla menší úprava datového typu atributu *dobaTrvani* (u všech entit). V objektovém modelu byl použit datový typ *number* (pro zjednodušení) a u relačního naopak *time*.

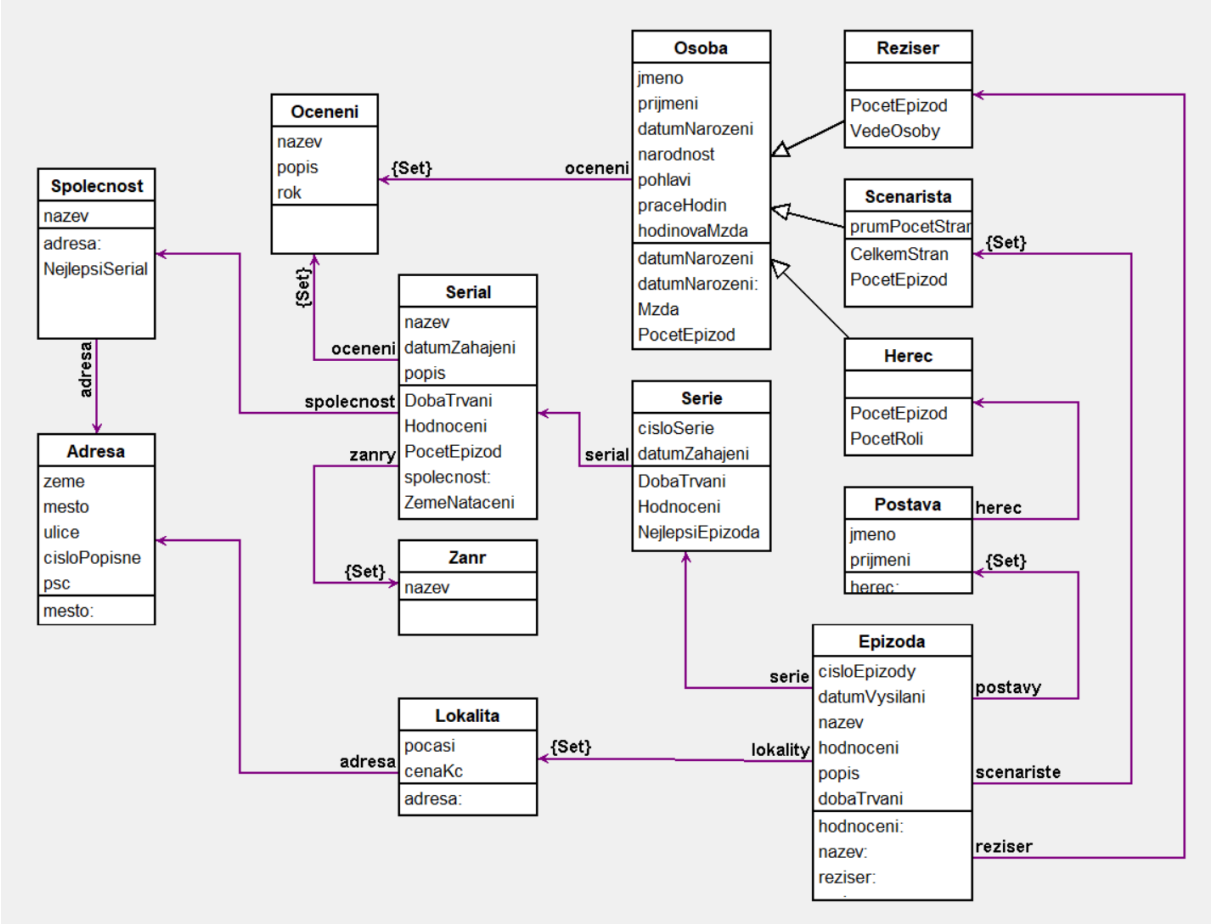

Diagram 2 – Daskalos diagram

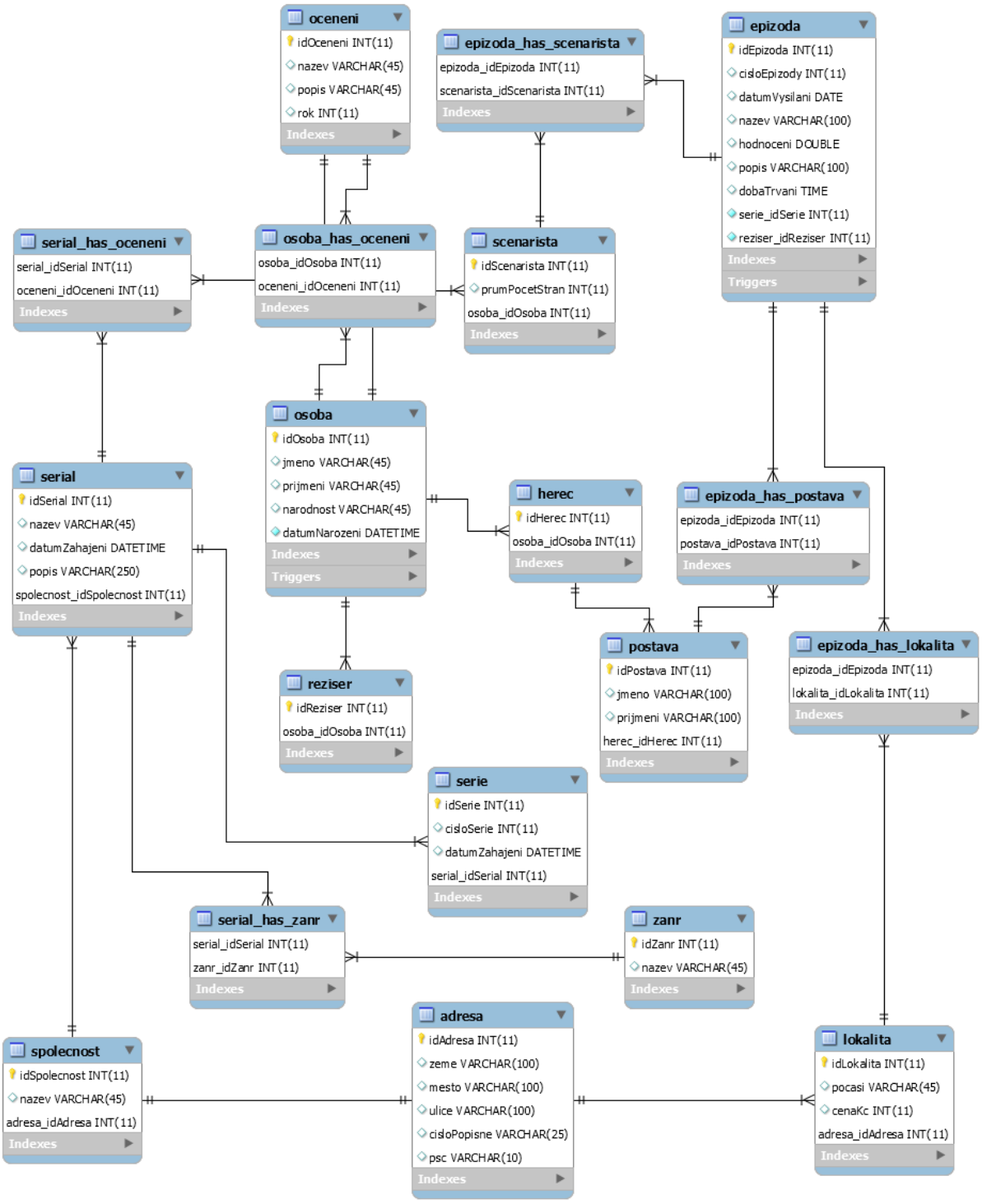

Diagram 3 – ER diagram

3.1 Metody

U objektového přístupu byly metody řešeny v příslušných objektech (začínají velkým písmenem). U relačního přístupu byly vytvořeny dotazy, u kterých lze pomocí klauzule WHERE upřesnit konkrétní entitu.

Počet epizod, ve kterých herec hrál bez ohledu na počet postav.

PocetEpizod

```
"Pocet epizod, ve kterych herec hral bez ohledu na pocet postav"

^(Epizoda allInstances select: [:ep | (ep postavy collect: [:p | p herec]) includes: self]) size
```

```sql
SELECT herec.idHerec,
       osoba.jmeno,
       osoba.prijmeni,
       COUNT(epizoda_idEpizoda) AS pocetEpizod
FROM epizoda_has_postava spoj
    LEFT JOIN epizoda ON spoj.epizoda_idEpizoda = epizoda.idEpizoda
    LEFT JOIN postava ON spoj.postava_idPostava = postava.idPostava
    LEFT JOIN herec ON postava.herec_idHerec = herec.idHerec
    LEFT JOIN osoba ON herec.osoba_idOsoba = osoba.idOsoba
#WHERE herec.idHerec = 1
GROUP BY idHerec;
```

Výpis zemí, kde byl daný seriál natáčen

ZemeNataceni

```
"Vsechny zeme nataceni daneho serialu"

^(((Epizoda allInstances select: [:ep | ep serie serial = self])
        collect: [:e | e lokality]) flatten
        collect: [:l | l adresa zeme]) asSet
```

```sql
SELECT serial.idSerial,
       adresa.zeme
FROM epizoda_has_lokalita spoj
    LEFT JOIN epizoda ON spoj.epizoda_idEpizoda = epizoda.idEpizoda
    LEFT JOIN serie ON serie.idSerie = epizoda.serie_idSerie
    LEFT JOIN serial ON serial.idSerial = serie.serial_idSerial
    LEFT JOIN lokalita ON spoj.lokalita_idLokalita = lokalita.idLokalita
    LEFT JOIN adresa ON lokalita.adresa_idAdresa = adresa.idAdresa
# WHERE serial.idSerial = 1
GROUP BY serial.idSerial, adresa.zeme
ORDER BY epizoda.idEpizoda;
```

Počet epizod všech sérií daného seriálu

PocetEpizod

```
"Pocet epizod ze vsech serii daneho serialu"

^(Epizoda allInstances select: [:ep | ep serie serial = self]) size
```

```sql
SELECT serial.idSerial,
       epizoda.idEpizoda
FROM epizoda
     LEFT JOIN serie ON serie.idSerie = epizoda.serie_idSerie
     LEFT JOIN serial ON serie.serial_idSerial = serial.idSerial
# WHERE serial.idSerial = 1
GROUP BY serial.idSerial;
```

Nejlépe hodnocená epizoda dané série

```
NejlepsiEpizoda

        "Nejlepsi epizoda cele serie"


        | epizody |

        epizody := Epizoda allInstances select: [:ep | ep serie = self].

        ^epizody inject: nil into:

                [:a :b | (a isNil ifTrue: [0] ifFalse: [a hodnoceni]) < b hodnoceni

                        ifFalse: [a]

                        ifTrue: [b]].
```

```sql
SELECT subselect.cisloSerie,
       subselect.nazev,
       subselect.hodnoceni,
       subselect.idEpizoda,
       subselect.idSerie
FROM (
         SELECT serie.cisloSerie,
                epizoda.idEpizoda,
                epizoda.nazev,
                epizoda.hodnoceni,
                serie.idSerie,
                RANK() OVER
                    (PARTITION BY serie.idSerie
                    ORDER BY epizoda.hodnoceni DESC) epizodaRank
         FROM epizoda
              LEFT JOIN serie ON serie.idSerie = epizoda.serie_idSerie
     ) subselect
WHERE subselect.epizodaRank = 1
# AND subselect.idSerie = 1
GROUP BY subselect.idSerie;
```

Nejlépe hodnocený seriál dané společnosti

NejlepsiSerial

```
"Nejlepsi serial spolecnosti podle prumerneho hodnoceni serialu"

| serialy nejlepsiSerial |
serialy := Serial allInstances select: [:se | se spolecnost = self].
^serialy inject: nil into:
        [:a :b | (a isNil ifTrue: [0] ifFalse: [a Hodnoceni]) < b Hodnoceni
                ifFalse: [a]
                ifTrue: [b]].
```

Hodnoceni

```
"Prumerne hodnoceni vsech epizod ze vsech serii daneho serialu"

^((Epizoda allInstances select: [:ep | ep serie serial = self])
        collect: [:e | e hodnoceni]) avg
```

```sql
SELECT subselect.nazev,
       subselect.nazevSerialu,
       subselect.prumerneHodnoceni
FROM (
         SELECT spolecnost.nazev,
                serial.nazev           AS nazevSerialu,
                spolecnost.idSpolecnost,
                serial.idSerial,
                AVG(epizoda.hodnoceni) AS prumerneHodnoceni
         FROM epizoda
              LEFT JOIN serie
                    ON serie.idSerie = epizoda.serie_idSerie
              LEFT JOIN serial
                    ON serial.idSerial = serie.serial_idSerial
              LEFT JOIN spolecnost
                    ON serial.Spolecnost_idSpolecnost = spolecnost.idSpolecnost
         GROUP BY serial.idSerial
         ORDER BY prumerneHodnoceni DESC
     ) subselect
#WHERE subselect.idSpolecnost = 1
GROUP BY subselect.idSpolecnost;
```

3.2 Dotazy

K dotazům jednotlivých přístupů byl navíc přidán zápis Lambda kalkulu, který se syntaxí velice podobá Smalltalku.

Výběr jedné nejlépe hodnocené epizody

λ	max(inst(Epizoda) ›› (λe | e ◁ hodnoceni))
Smalltalk	Epizoda inject: nil into: [:a :b | (a isNil ifTrue: [0] ifFalse: [a hodnoceni]) < b hodnoceni ifFalse: [a] ifTrue: [b]]

SQL	`SELECT *`
	`FROM epizoda`
	`ORDER BY epizoda.hodnoceni DESC`
	`LIMIT 0,1;`

Výběr adres (města) všech společností

λ	inst(Spolecnost) ›› (λs | s ◁ adresa ◁ mesto)
Smalltalk	Spolecnost collect: [:s | s adresa mesto]

SQL	`SELECT adresa.mesto`
	`FROM spolecnost`
	`  LEFT JOIN adresa ON spolecnost.Adresa_idAdresa = adresa.idAdresa;`

Výběr prvních epizod prvních sérií

λ	inst(Epizoda) // (λe | e ◁ serie ◁ cisloSerie = 1 & e ◁ cisloEpizody = 1)
Smalltalk	Epizoda select: [:e | e serie cisloSerie = 1 & e cisloEpizody = 1]

SQL	`SELECT *`
	`FROM epizoda`
	`  LEFT JOIN serie ON serie.idSerie = epizoda.serie_idSerie`
	`WHERE serie.cisloSerie LIKE 1`
	`  AND epizoda.cisloEpizody LIKE 1;`

Výběr seriálu s žánrem komedie

λ	inst(Serial) // (λs | s ◁ zanr ›› list(zanr, λz | z ◁ nazev = 'Komedie'))
Smalltak	Serial select: [:s | (s zanry collect: [:z | z nazev]) includes: 'Komedie']

SQL	`SELECT serial.nazev, zanr.nazev`
	`FROM serial_has_zanr`
	`  LEFT JOIN serial ON serial.idSerial = serial_has_zanr.serial_idSerial`
	`  LEFT JOIN zanr ON serial_has_zanr.zanr_idZanr = zanr.idZanr`
	`WHERE zanr.nazev = 'Komedie';`

Výběr částky nejdražší lokality v České republice

λ	max((inst(Lokalita) // (λl | l ◁ adresa ◁ zeme = 'Czech Republic')) ›› (λl | l ◁ cenaKc))
Smalltalk	((Lokalita select: [:l | l adresa zeme = 'Czech Republic']) collect: [:l | l cenaKc]) max

SQL	`SELECT MAX(lokalita.cenaKc) AS max`
	`FROM adresa`
	`  LEFT JOIN lokalita ON adresa.idAdresa = lokalita.adresa_idAdresa`
	`WHERE adresa.zeme = 'Czech Republic';`

3.3 Pravidla

Pravidla, která se provádí během vkládání nebo čtení.

Formát zobrazení data narození u osoby

Smalltalk	**datumNarozeni** "generated by Daskalos" ^datumNarozeni printFormat: #(1 2 3 $. 1 1)

SQL	`SELECT DATE_FORMAT(osoba.datumNarozeni, '%d.%m.%y') as datumNarozeni`
	`FROM osoba;`
	`# WHERE osoba.idOsoba = 1;`

Omezeni rozsahu hodnoty hodnocení pro epizodu (0 až 10)

Smalltalk	**hodnoceni: anObject** "generated by Daskalos" | tmpHodnoceni | tmpHodnoceni := self checkValue: anObject forVariable: #hodnoceni. (tmpHodnoceni > 10 or: [tmpHodnoceni < 0]) ifTrue: [self error: 'Hodnoceni musi byt cislo od 0 do 10']. hodnoceni := tmpHodnoceni

SQL	`CREATE TRIGGER `before_insert_on_epizoda ` BEFORE INSERT ON `epizoda` FOR EACH ROW`
	`BEGIN`
	`  IF NEW.hodnoceni > 10 OR NEW.hodnoceni < 0 THEN`
	`  SIGNAL SQLSTATE '45000'`
	`  SET MESSAGE_TEXT = 'Hodnoceni musi byt cislo od 0 do 10';`
	`  END IF;`
	`END;`

Omezení pro minimální hodnotu datumu narození na 1.1.1900

Smalltalk	**datumNarozeni: anObject** "generated by Daskalos" | tmpDatumNarozeni | tmpDatumNarozeni := self checkValue: **anObject** forVariable: #datumNarozeni. tmpDatumNarozeni < (Date readFromString: '1-1-1900') ifTrue: [self error: 'Minimalni povoleny datum narozeni je 1-1-1900']. datumNarozeni := tmpDatumNarozeni

SQL	`CREATE TRIGGER` `before_insert_on_osoba` `BEFORE INSERT ON` `osoba` `FOR EACH ROW`
	`BEGIN`
	`IF` NEW.datumNarozeni < '1900-01-01' `THEN`
	`SIGNAL SQLSTATE` '45000'
	`SET MESSAGE_TEXT` = 'Minimalni povoleny datum narozeni je 1900-1-1';
	`END IF;`
	`END;`

Omezení rozsahu názvu epizody na 100 znaků

Smalltalk	**nazev: anObject** "generated by Daskalos" | tmpNazev | tmpNazev := self checkValue: **anObject** forVariable: #nazev. tmpNazev size > 100 ifTrue: [self error: 'Nazev epizody muze mit maximalne 100 znaku']. nazev := tmpNazev
SQL	`create table epizoda` (# ... nazev `varchar(100)` # ...)

Integrita vztahu epizody se sérií (validace objektu, cizí klíč)

Smalltalk	**serie: anObject** "generated by Daskalos" | tmpSerie | tmpSerie := self checkValue: **anObject** forVariable: #serie. (tmpSerie isMemberOf: Serie) ifFalse: [self error: 'Zprava ''Epizoda serie:'' prijima pouze instanci tridy ''Serie''']. serie := tmpSerie

SQL	`create table epizoda`
	`(`
	`#  ...`
	`  constraint epizoda_serie_idSerie_fk`
	`  foreign key (serie_idSerie) references serie (idSerie)`
	`  on update cascade`
	`#  ...`
	`)`

4 Závěr

Výsledkem projektu jsou dva funkčně podobné modely realizované pomocí rozdílných technologií a přístupů. V první řadě byl vytvořen návrh modelu prostřednictvím UML diagramu tříd, který byl použit při realizaci objektové a relační databáze.

Pokud se zamyslíme nad podstatou věci, tak je již například z dotazů patrné, že se v objektovém prostředí lepé pohybuje mezi objekty a jejich asociacemi, než je tomu v relačním přístupu. Objekty mají v objektových databázových systémech tu vlastnost, že se s nimi pracuje jako s celky a můžeme tak jednoduše zjistit potřebné vlastnosti konkrétní instance objektu. V relačních databázových systémech, musíme pro tyto potřeby provádět různá propojení několika relací, abychom vytěžili potřebná data, přičemž tyto operace bývají velice náročné na zpracování (při velkém objemu dat).

Nevýhodou objektového přístupu Smalltalku je velice limitující dokumentace a nedostatek návodů. Za jednu z výhod by se dalo považovat přímé ukládání na místní disk, kde není potřeba spouštět žádný server, jako tomu je například u MariaDB (MySQL).

Přílohy

Zdrojové kódy: https://github.com/nocturne32/databaze-serialu