OBJEKTOVÝ PŘÍSTUP V DATABÁZOVÉ TECHNOLOGII

Vojtěch Merunka

Katedra informačního inženýrství, PEF, ČZU Praha
merunka@pef.czu.cz

Abstrakt

Příspěvek seznamuje s objektovým datovým modelem a popisuje, jaký má vztah k síťovému a relačnímu datovému modelu. Na konkrétním příkladu systému Gemstone je ukázán postup tvorby objektové databázové aplikace a jsou demonstrovány dotazy v jazyce Smalltalk a OQL.

Klíčová slova: Gemstone, IDMS, objektová databáze, objektový datový model, ODMG, OQL, Oracle, relační datový model, síťový datový model, Smalltalk, tvorba objektové databáze, VisualWorks.

1. Úvod

První objektově orientované databáze se objevily již ve druhé polovině 80. let a vznikly na základě potřeby uchovávat a databázově zpracovávat v pokud možno nezměněné podobě data z programů napsaných v tehdy se rozvíjejících objektově orientovaných programovacích jazycích. Ve srovnání s relačními databázemi, které v té době byly na vrcholu vývoje, to byly systémy velmi neefektivní a málo výkonné, protože se jednalo o experimentální programy psané jako aplikace v nějakém objektovém programovacím jazyce.

Po více než deseti letech vývoje je však situace jiná. Z praxe již známe případy, kdy nasazení objektové databáze vyřešilo problémy, které relační systém nedokázal zvládnout. Objektové databázové aplikace se objevují již i v ČR. Dnešní objektové databáze mají srovnatelný výkon s velkými relačními systémy – zvládají stovky transakcí za sekundu a tisíce současně připojených uživatelů. S objektovými databázovými aplikacemi se můžeme setkat například v informačních systémech letového provozu (např. Finair, Air France), rezervačních systémech osobní letecké dopravy (např. Fractal s.r.o. u nás v Praze, TourisNet Gran Canaria), informačních systémech pro řízení dopravy zboží (např. Orient Overseas Container Line – jeden z 3 největších dopravců mezi USA a Evropou), informačních systémech dodavatelů elektřiny (např. Floria Power & Light), rezervačních hotelových služeb (např. Navigant International Northwest Travel), systémů pro pojištění (např. povinné ručení automobilů v Argentině) a další. Objektové databáze také používá pro speciální aplikace mnoho firem v kombinaci s relačními. Jsou to například firmy Texas Instruments, BMW, Opel, Ford, JP Morgan, IBM, Hewlett Packard, AT&T a další.

2. Objektově orientované a objektově relační databáze

Databázové systémy jsou založené na různých datových modelech. Jde o datový model síťový (a jeho variantu hierarchický datový model), relační, objektově relační a objektově orientovaný. (Někteří autoři také považují za databázové datové modely ještě modely fulltextové, hypertextové a modely založené na sémantických sítích).

Dnes dominuje relační datový model, který ve většině aplikačních oblastí postupně nahradil databáze založené na síťovém datovém modelu. Dnešní praxe však ukazuje, že relační databáze začínají být postupně nahrazovány databázemi objektovými. Pod obecným označením „objektové databáze“ se však skrývají dva vzájemně odlišné datové modely:

1. Objektově relační datový model představuje evoluční trend vývoje. Jde o doplnění relačního datového modelu o možnost práce s některými datovými strukturami, které známe z oblasti objektově orientovaných programovacích jazyků. Většina výrobců velkých relačních databázových systémů (např. Oracle) zvolila tuto variantu. Objektově relační datový model ale ve svých principech zůstává původním relačním datovým modelem.
2. Objektově orientovaný datový model, který představuje revoluční trend vývoje. Jde o nový datový model, který není postaven jako rozšíření relačního datového modelu. Do jisté míry zde jde o renesanci původního síťového datového modelu, který je doplněn o možnost práce s objekty tak, jak je známe z objektového programování.

Relačně objektová technologie je dnes rozšířenější. „Nerelační“ objektově orientovaný datový model má však následující přednosti:

1. Lépe podporuje datové struktury, které známe z objektově orientovaných programovacích jazyků. Není třeba datové struktury tolik transformovat, aby byly uložitelné do databáze. Existují již prakticky použitelné systémy (např. Gemstone, ObjectStore, O2, Versant, …), které dovolují v databázi zpracovávat objekty ve stejném tvaru, jak se s nimi nakládá v objektových programovacích jazycích.
2. Protože navazuje na síťový datový model, tak má předpoklady pro efektivnější způsoby zpracování dotazů ve srovnání s relačním datovým modelem. Tato vlastnost se projevuje hlavně u složitých datových struktur, které by se podle relačního datového modelu musely rozkládat do mnoha vzájemně provázaných relačních tabulek.

Příčiny, proč jsou zatím objektové databáze méně v praxi rozšířené, než relační, jsou pravděpodobně následující:

1. Malá praktická znalost objektových databází v komunitě tvůrců softwaru.
2. Dostupnost systémů na trhu a jejich cena. Velké relačně objektové systémy jsou levnější než objektové. (například komerční cena systému Gemstone je asi 2x větší než Oracle).
3. Konservativní myšlení potenciálních uživatelů a samozřejmě také potřeba zpracovávat již vytvořené báze dat v relačních systémech.
4. Chybějící standardy a nedostatečná podpora metod analýzy a návrhu. Návrh standardu ODMG 3.0 [5] není všemi výrobci respektován. Modelovací jazyk UML nepodporuje všechny konstrukce potřebné k modelování objektové databáze. Metody používané pro návrh relačních databází nejsou vhodné k plnému využití možností objektových databází.

Přes uvedené problémy však existují důvody se domnívat, že význam objektových databází v blízké budoucnosti poroste, protože již dnes existuje celá řada aplikací, kde objektové databáze prakticky prokazují svoje přednosti. Společnou vlastností těchto aplikací je velké množství komplexních datových struktur a jejich proměnlivost za chodu systému, které způsobují problémy relačním databázím. Takové systémy mohou pracovat až se stovkami a tisíci různých vzájemně poskládaných datových typů reprezentovaných třídami objektů. Dotazy nad takovými objekty ještě navíc vyžadují vysokou míru vzájemného polymorfismu. (V takových systémech kupříkladu potřebujeme klást dotazy nad množinami obsahující prvky

různého typu. A zároveň očekáváme, že při přidání nového datového typu se nebudou muset přepisovat již hotové dotazy.) Typickým příkladem takových systémů jsou datové sklady, které jsou charakteristické dlouhodobým shromažďováním velkého množství nově vznikajících různorodých dat. Takové systémy jsou charakteristické nejen pro řízení velkých podniků, ale také v různých evidenčních systémech státní správy, zdravotnických systémech, informačních systémech obsahujících ekologické informace, zemědělských informačních systémech, historiografických informačních systémech atp.

Na druhou stranu je třeba poznamenat, že relační databáze fungují velmi dobře v oblastech, kde během života systému nedochází k požadavku na změnu struktury databáze a na přidávání dalších datových typů. Relační systém může být výkonný i pokud se databáze skládá z velkého množství záznamů, ale uložených v malém počtu jednoduše strukturovaných relačních tabulek.

Bohužel se v ČR dnes objektovými databázemi žádné pracoviště soustavně nezabývá. Dílčí práce byly vykonány v první polovině 90. let na Fakultě elektrotechniky a informatiky VUT Brno a na Matematicko fyzikální fakultě Komenského univerzity v Bratislavě. [9,11,12] Práce obou týmů vedly ke konstrukci experimentálních databázových systémů. Slovenský systém byl později dopracován ve finském projektu tvorby metamodelovacího nástroje na universitě v Jyväskylä – nyní jedním z celosvětově používaným CASE nástrojem Metaedit™. Dnes je situace taková, že na univerzitách v ČR se až na výjimky objektové databáze neobjevují ani ve výuce.

Ve světě je několik univerzitních pracovišť, které se objektovými databázemi zabývají (např. CERN, Université de Genéve, Vrije Universiteit Brusel a celá řada v USA jako např. MIT a Stanford University). Výsledky jejich práce jsou využívány v praxi při konstrukci objektových databází. Na internetu existuje mezinárodní sdružení ODMG – Object Database Management Group (www.odmg.org).

Problematika objektových databázích je od poloviny 90. let diskutována na odborných konferencích. Od konce 90. let vycházejí v zahraničí odborné publikace, které se zabývají především vlastnostmi vybraných objektových databází. [1,3,6,7,10,13] Přestože již existuje mnoho dílčích teoretických prací, které jednotlivě dokazují účelnost objektově orientovaného datového modelu v databázových systémech, tak se zatím v oblasti metod analýzy a návrhu objektových databázových aplikací používají jen postupy původně určené pro práci s relačními systémy a nebo jen intuitivní přístupy založené na zkušenosti s imperativními objektově orientovanými programovacími jazyky.

3. Objektově orientovaný datový model

Hlavním motivem pro vznik objektového datového modelu (ODM) byly problémy s ukládáním a zpracováním objektů v relačních databázích. Relační datový model (RDM) objektovému programování nevyhovuje, protože je příliš jednoduchý. Z tohoto důvodu vznikl tlak na konstrukci nových databázových systémů, které by lépe dokázaly pracovat s objekty.

Objektový a relační datový model se od sebe výrazně liší. Tabulky jsou v ODM pouze jedna z možných forem výstupní prezentace uložených dat. ODM se nicméně může podobat strukturám síťových databází, jak jsme je znali v systémech IDMS. Na ODM můžeme nahlížet jako na renesanci síťového datového modelu. Při určité míře zjednodušení lze připustit vztah:

síťový datový model + objektové typy dat + polymorfismus = objektový datový model.

Vyjmenujme si nyní základní charakteristiky objektového datového modelu:

1. Objektová databáze podporuje více typů množin objektů. (na rozdíl od relačního datového modelu, kde je relační tabulka jediným „druhem množiny“). Společně se označují termínem *collection* (české označení zatím chybí, většina autorů používá termín „sada“ či „kolekce“). V konkrétních databázových systémech to může být až několik desítek různých typů různých vlastností tak, jak je známe z knihoven objektových programovacích jazyků. (Je to například Array, List, OrderedCollection, SortedCollection, Set, Bag, Dictionary, …)
2. Objektová databáze rozlišuje mezi pojmem třída objektů a množina (kolekce) objektů. Třída je jen realizace datového typu objektů a množina je jen úložiště pro objekty. Na rozdíl od tabulek v RDM, kde role třídy a množiny splývají dohromady, v ODM nemusíme pracovat pouze s množinami, které obsahují jen všechny objekty jedné třídy. Můžeme mít například více množin objektů stejného typu i množinu obsahující objekty z různých tříd. (Pokud takové objekty mají díky polymorfismu nějaké společné atributy, tak nám nic nebrání je držet pohromadě a nad takovou množinou provádět například selekci.)
3. Objekty se skládají z vnitřních datových složek (což mohou být opět jiné objekty) a z metod, které představují funkční stránku každého objektu. Známe nejen tzv. přístupové metody, které jen přímo manipulují s datovými složkami objektu (zapisují nové hodnoty datových složek a nebo čtou hodnoty datových složek), ale i metody složitější, které vypočítávají z objektů taková data, jenž v objektu nebyla jednoduše uložena jako jedna z jeho datových složek. Toto známe z objektového programování. Pro ODM je ale důležité si uvědomit, že mezi atributy objektů patří nejen jejich datové složky (jako v RDM), ale i metody poskytující další data. (Dále v textu je jeden takový příklad atributu „věk“ osoby.)
4. Polymorfismus objektů nevzniká pouze děděním tříd. Pokud mají objekty společné atributy, tak jsou polymorfní i když jejich třídy mezi sebou nedědí.
5. Každý objekt má svoji vlastní identitu, což v objektové databázi znamená, že v rámci jednoho paměťového prostoru má každý objekt systémem přidělen jednoznačný identifikátor obvykle označovaný jako OID (Object IDentifier). OID plní úlohu ukazatele do virtuální paměti. OID každého objektu zůstává stejný, i když se v objektu změní všechny jeho datové složky nebo metody. OID se také samozřejmě nemění při změnách objektu na fyzické úrovni, jako např. změna jeho umístění v operační paměti nebo na disku. Vzhledem k existenci konceptu OID můžeme rozlišovat mezi pojmem rovnost dat objektu a totožnost objektu (Dva objekty se shodnými daty ještě nemusejí být totožné). Praktický důsledek konceptu OID je ten, že v ODM není potřeba objektům vytvářet primární klíče. Toto RDM nezná, neboť identita relačních záznamů je dána jen hodnotami atributů. V některých objektových databázích (např. Gemstone) mohou OID zůstat skryté pod aplikačním rozhraním SŘBD. V takové databázi potom její uživatel vidí objekty, které se přímo propojují a skládají mezi sebou.
6. V ODM lze v bázi dat pracovat i s takovou soustavou objektů, která je sama o sobě aplikací. Objektová databáze nemusí sloužit jen jako úložiště dat, se kterým manipuluje externí program. Algoritmy programu lze „rozpustit“ v metodách objektů přímo uložených v objektové databázi. Tvorba databázové aplikace na straně klienta je potom velmi zjednodušená, protože v extrémním případě se může jednat jen o prezentační rozhraní výpočetního systému, který celý pracuje „uvnitř“ objektového databázového serveru.

7. Na rozdíl od běžných objektových programovacích jazyků mohou objekty v ODM migrovat mezi různými třídami, v systému může existovat současně více verzí jedné třídy. Různí uživatelé podle svých přístupových práv mohou mít dostupné různé atributy na stejných objektech.

Na závěr si sobě odpovídající pojmy relačního a objektového datového modelu porovnáme v následující tabulce:

Tab. 1. Porovnání relačního a objektového datového modelu

RDM	ODM
záznam (řádek tabulky)	objekt (prvek množiny)
tabulka	1) třída objektů (jako datový typ) 2) množina objektů (i z různých tříd)
atribut (položka řádku tabulky)	1) datová složka objektu 2) metoda objektu, která poskytuje data
primární klíč (není ukazatelem do paměti)	OID (je ukazatelem do paměti)

4. Jak vytvořit objektovou databázovou aplikaci

Objektový datový model není nadstavbou relačního datového modelu. Je tedy otázkou, jaké metody návrhu použít. Pro relační datový model je k dispozici datová normalizace, metoda syntézy atributů a metoda datové dekompozice podle funkčních závislostí atributů. Bohužel pro objektový datový model zatím není žádná všeobecně uznávaná a používaná technika nebo metoda návrhu. Je možné převzít relační techniky, ale potom dostaneme jen „relační databázi v objektovém prostředí“ a nevyužijeme všechny vlastnosti, které ODM má. Jinou možností je převzít schéma objektů a tříd tak, jak jsou navrženy v aplikaci, která má s databází pracovat. To už je lepší, protože právě proto byl objektový datový model vyvinut. Jenomže struktura objektů výhodná pro aplikaci může komplikovat jejich efektivní databázové zpracování.

Čtenáři se mohou v různých pramenech setkat s různými tvrzeními o objektových databázích, které tuto problematiku zjednodušují a prohlašují například, že objektovou databázi není třeba normalizovat a že objektová databáze je mnohonásobně rychlejší než relační.

Tvrzení o rychlosti platí, ale jen pro případ, kdy se podaří navrhnout objektové schéma tak, aby obsahovalo přímo propojené objekty mezi sebou na rozdíl od relační databáze, která musí používat spojení od cizího klíče z jedné tabulky na primární klíč druhé tabulky. Tedy jinými slovy pokud se podaří objektové schéma navrhnout v duchu zásad síťového datového modelu.

S normalizací to je ještě složitější. Uvažuje se o využití refaktoringu a návrhových vzorů. Určitý pokrok učinil Kent Beck, jeden z pionýrů agilních metodik, ve své knize [2], kde prezentuje první tři „objektové normální formy“, které se týkají správného návrhu struktury tříd objektů a jsou odvozeny z relačních normálních forem. V relačním datovém modelu jsou jednotkou funkční závislosti samotné atributy – tedy datové složky v záznamech a ne celé záznamy. Proto se bez normalizace v RDM neobejdeme. Objektový datový model pracuje s objekty, které navenek vystupují jako nedělitelné jednotky. Proto není problém normalizace tak naléhavý jako v RDM, ale neznamená to, že neexistuje.

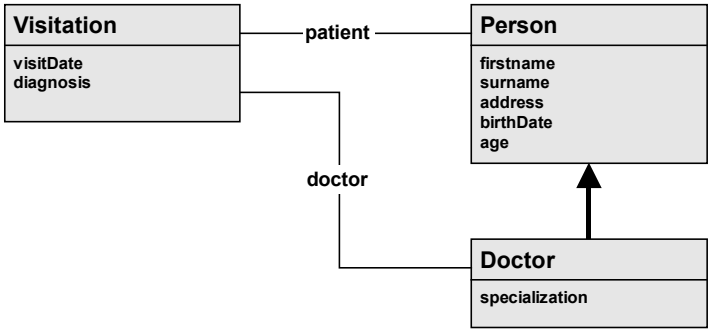
Při tvorbě objektové databáze jsme tedy zatím odkázáni jen na zkušenost. Lze však popsat postup, jak objektovou databázi vytvořit [4,14,15]:

1. Sestavit konceptuální model úlohy. Zde je možné použít diagram tříd UML nebo Chenův konceptuální ER diagram s rozšířením na vazby generalizace-specializace. Zatím ale modelujeme jen množiny. Atributy zde rozpoznané budou později implementovány nejen jako datové složky ale také jako metody objektů. V tomto kroku se to ještě nerozlišuje.
2. K prvkům množin najít potřebné třídy. Bohužel diagram tříd UML nemá zvláštní symboly pro množiny a pro třídy. Pokud máme množiny z objektů jen jedné třídy, tak to nevadí, ale jinak musíme použít stereotypy nebo přidat nový symbol.
3. Rozhodnout, které atributy budou implementovány datovými složkami a které metodami. Tyto metody pak naprogramovat.
4. Naplnit databázi daty. Tedy vytvářet objekty (instance) tříd a ukládat je do vybraných množin. Vytvořit objekt jako instanci třídy nestačí. Takový objekt totiž ještě není prvkem žádné množiny v databázi.

5. Příklad objektové databáze

5.1 Popis úlohy

Vlastnosti objektové databáze budou prezentovány pomocí následující úlohy: Mějme jednoduchou databázi pro evidenci návštěv v ordinacích lékařů. U každého pacienta bude evidováno jeho jméno, příjmení, adresa, datum narození a věk. U každého doktora ještě navíc jeho specializace. U návštěvy pacienta u lékaře bude evidováno datum návštěvy a diagnóza. Není vyloučeno, aby se jeden lékař stal druhému lékaři pacientem. Úlohu znázorňuje následující obrázek:



Obr. 1. Analýza úlohy

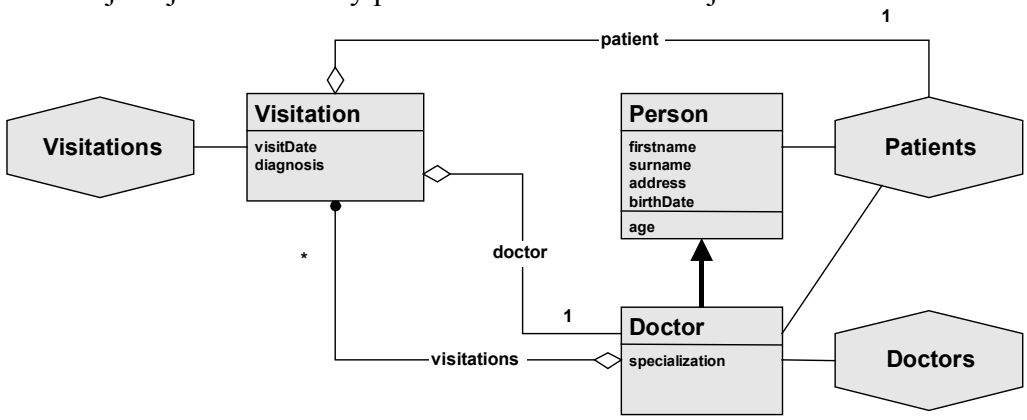
5.2 Implementace úlohy

1. Požadované atributy objektů je možné implementovat nejen jako jejich datové položky, což by odpovídalo možnostem relačních databází, ale také jako metody objektů. V našem příkladu může být takto řešen atribut `age`, který je vypočitatelný metodou z data narození.
2. V aplikaci lze navrhovat nejen třídy objektů, ale také množiny objektů, které mohou obsahovat jako prvky objekty z různých tříd. V našem příkladu to může být množina `Patients`, která bude obsahovat jako svoje prvky instance třídy `Person` i `Doctor`. Další množiny `Doctors` a `Visitations` budou obsahovat pouze instance jedné třídy a tím tedy budou podobné relačním tabulkám.
3. Protože objektové vývojáře tolik nesvazují požadavky normálních forem, tak si lze zvolit směr, kterým na sebe budou objekty odkazovat. Například je možné odkazovat objekt třídy `Doctor` z objektu třídy `Visitation` a nebo rovněž tak odkazovat z objektu třídy `Doctor` na podmnožinu objektů třídy `Visitation`. Záleží jen na analýze zadání úlohy, která zkušenému vývojáři napoví, jaký směr vazby je pro praktický chod databáze

přirozenější. Pokud je nějakým způsobem zabezpečena konzistence báze dat, tak je možné realizovat oba dva směry vazby současně. Tak tomu bude v našem příkladu u vazby mezi doktory a vyšetřeními.

4. Podobně jako směr vazby, tak na vývojáři záleží, jaké množiny objektů zveřejní uživatelům databáze. V naší úloze totiž není nezbytně potřeba zveřejňovat všechny množiny. Pokud například nebude třeba klást příliš často dotazy nad množinou všech doktorů, tak množina `Doctors` není nutná, protože data o doktorech jsou tranzitivně dosažitelná z objektů třídy `Visitation`. V našem příkladě jsou ale z didaktických důvodů všechny množiny ponechány.

Popsaná implementace je znázorněna na následujícím diagramu, kde kromě standardních symbolů UML jsou ještě šestihrany pro znázornění množin objektů:



Obr. 2. Implementace úlohy

5.3 Program v jazyce Smalltalk DB databázového systému Gemstone

Program musí nejprve vytvořit příslušné třídy s metodami, potom vytvořit potřebné množiny a nakonec do nich uložit data, což budou instance vytvořených tříd. V praxi je výhodné samozřejmě v co největší míře využívat vizuální programovací nástroje prostředí (viz. obrázek 3.). [8,14] Zde budou ukázány jen zdrojové kódy v jazyce Smalltalk DB, který je databázovým rozšířením univerzálního objektového programovacího jazyka Smalltalk. Databázový systém Gemstone samozřejmě na straně klienta podporuje i jiné programovací jazyky (např. Java a C++) a dovoluje objekty v nich vytvořené konvertovat a mapovat na objekty ve své databázi. Gemstone má také relační interface dodržující standard SQL-92 a objektový interface kompatibilní s CORBA 2.0.

Nejprve vytvoříme třídy. Lze k tomu použít vizuální nástroje. Gemstone má ale také znakový příkazový řádek. Zde je ukázka kódu pro definici třídy `Person` a třídy `Doctor`, která z ní dědí:

```
Object subclass: 'Person'
   instVarNames:
          #(firstname surname
            address birthDate)
   inDictionary: UserClasses.
```

```
Person subclass: 'Doctor'
   instVarNames: #(specialization)
   inDictionary: UserClasses.
```

Dále je třeba napsat potřebné metody. Na ukázce je metoda `age` třídy `Person`, kterou je realizován atribut věk osob:

```
Method: Person
age
  ^birthDate isNil
```

```
   ifTrue:  [nil]
  ifFalse:  [((Date today subtractDate: birthDate) / 365.2422) truncated]
```

V kapitole 5.2 odstavci 3 byla popsána možnost oboustranného propojení mezi doktory a vyšetřeními. Pro udržení konzistentní databáze je potom výhodné využít možnosti objektového programování a definovat například k doktorům metody na přidávání a odebírání vyšetření tak, aby byla zajištěna konzistence i „z druhé strany“:

```
Method: Doctor
addVisitation: aVisitation
  visitations add: aVisitation.
  aVisitation doctor: self.
```

```
Method: Doctor
removeVisitation: aVisitation
  visitations remove: aVisitation.
  aVisitation doctor: nil.
```

Dále bude třeba vytvořit množiny objektů. Množiny objektů se ukládají do logické paměťové oblasti `UserGlobals`:

```
  UserGlobals at: #Doctors put: Set new.
  UserGlobals at: #Patients put: Set new.
  UserGlobals at: #Visitations put: Set new.
```

A nyní už zbývá jen naplnit data. Pro uživatele manipulace s daty (jako ostatně všechno) probíhá zcela podle zásad objektového programování, jak ukazuje následující ukázka:

```
d := Doctor new
  firstname: 'Jan';
  surname: 'Novak';
  address: 'Novakova 12';
  birthDate: '4-JAN-1950';
  specialization: 'obvodni';
  visitations: Set new.
```

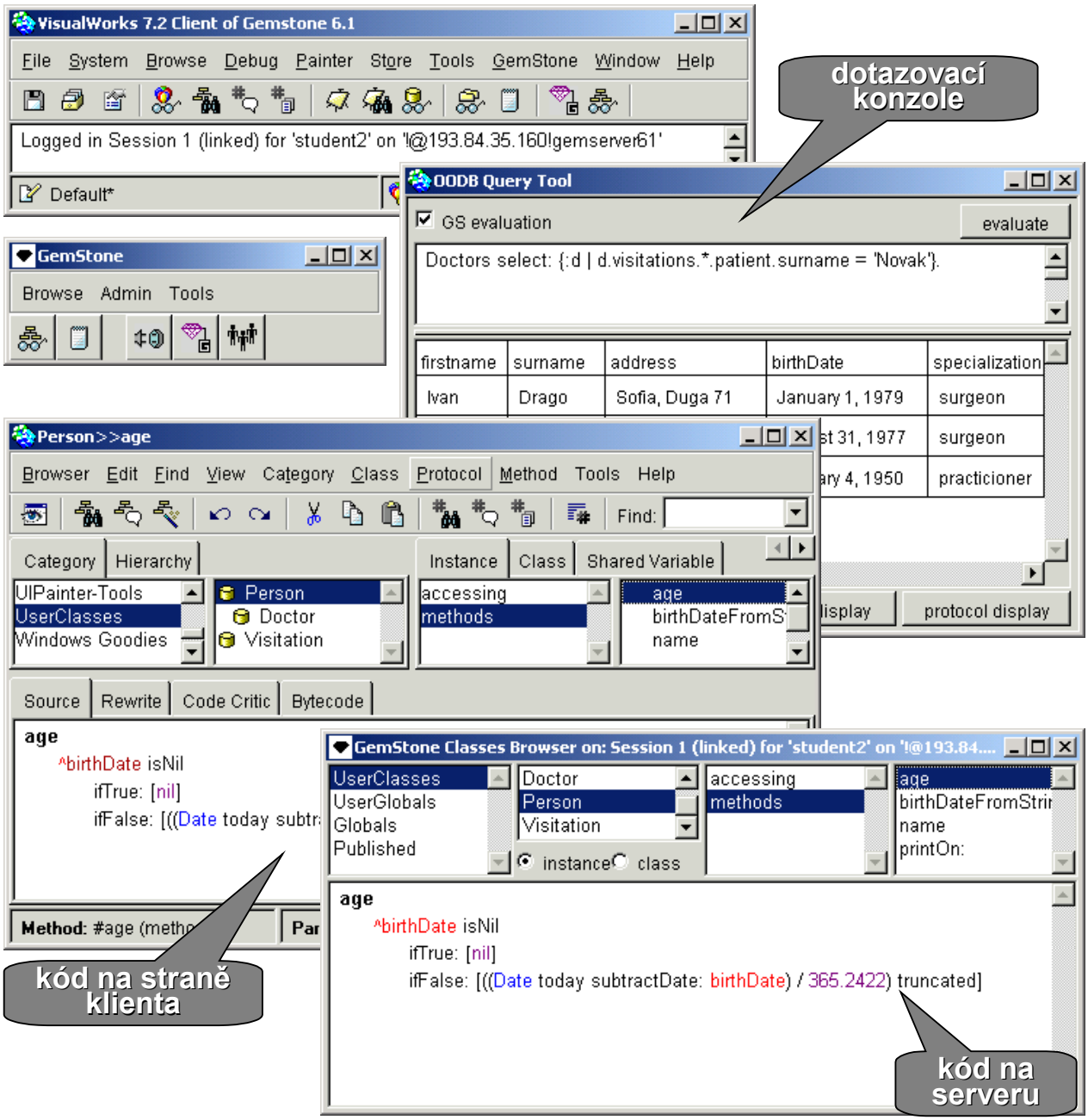
```
p := Person new
  firstname:
  'Karel';
   surname: 'Noha';
   address: 'Zikova 56';
  birthDate: '3-JAN-1954'.

v := Visitation new
  visitDate: '12-DEC-2000';
  diagnosis: 'angina'.
```

Objekty `d, p, v` je ještě třeba propojit (propojujeme je přímo – spojení odkazem z cizích na primární klíče nepotřebujeme):

```
  d addVisitation: v.    v patient: p.
```

a nakonec vložit do množin:

```
  Doctors add: d.          Patients add: p.          Visitations add: v.
```



Obr. 3. Ukázka grafického uživatelského prostředí VisualWorks/Gemstone

6. Příklady dotazů

Následujících několik ukázek dotazů předvede dotazovací možnosti jazyka Smalltalk DB [8,14] a také jazyka OQL [5,15], který je součástí standardu ODMG a podobá se jazyku SQL:

1) Najdi vyšetření, které prováděl doktor Dyba:

```
Visitations select: {:v | v.doctor.surname = 'Dyba'}.
```

```
SELECT *
  FROM v IN Visitations
  WHERE v.doctor.surname = 'Dyba';
```

Gemstone samozřejmě podporuje indexování. Pokud by bylo třeba zvýšit výkon právě předloženého dotazu, tak by správce databáze mohl vytvořit indexy například takto:

```
Visitations createEqualityIndexOn: 'doctor.surname'
    withLastElementClass: String.
```

2) Najdi všechny doktory, kteří léčili pacienta Nováka.

```
Doctors select: {:d | d.visitations.*.patient.surname = 'Novak'}.
```

```
SELECT *
  FROM d IN Doctors
  WHERE EXISTS
    (SELECT * FROM v IN d.visitations WHERE v.patient.surname = 'Novak');
```

Hvězdička v datové cestě parametru dotazu ve Smalltalku DB znamená, že bude třeba vyhledávat do šířky, protože atribut `visitations` doktora `d` není jedním objektem, ale celou množinou dalších objektů, z nichž každý má dále po cestě atributy `patient.surname`. OQL takovou schopnost nemá a tak musíme použít podmínku `EXISTS` na vnořený příkaz `SELECT`.

3) Najdi adresy pacientů, kteří měli angínu:

```
(Visitations select: {:v | v.diagnosis = 'angina'})
    collect: {:v | v.pacient.address}
```

```
SELECT v.patient.address
  FROM v IN Visitations
  WHERE v.diagnosis = 'angina';
```

4) Najdi pacienty starší 60 let.

```
Patients select: {:p | p.age > 60}.
```

```
SELECT *
  FROM p IN Patients
  WHERE p.age > 60;
```

V podmínce se odvolává na atribut, který je počítán metodou. Využívá se tu také polymorfismu, protože v množině `Patients` jsou instance třídy `Person` i `Doctor`.

Tyto příklady byly záměrně vybrány tak, aby demonstrovaly přednosti propojení objektů díky síťové datové struktuře v objektové databázi. Z dotazů ve Smalltalku i OQL je na první pohled patrné, že srovnatelná relační databáze by byla komplikovanější a odpovídající dotazy složitější.

7. Závěr

Je pravda, že objektově relační databáze dokáží implementovat naši úlohu také. Především poslední verze Oraklu dovolují využít hodně hybridních objektově-relačních vlastností. Cílem článku ale bylo prakticky ukázat čistý objektový datový model. Hybridní objektově relační technologie je totiž ještě méně standardizovaná, než objektová. Tvorba aplikací, která se o tento přístup opírá, se snadno dostane do vleku specifických funkcí posledních verzí konkrétního systému. Oproti tomu zde popsaný přístup je aplikovatelný nejen v Gemstone,

ale v jakékoliv objektově orientované databázi (např. ObjectStore, O_2, Cache, Ontos, Jasmine, …).

Současná praxe bohužel pod pojmem objektové databáze chápe většinou jen různá rozšíření relačních databází. To je velká chyba, protože o „nerelačních“ databázích u mnoha informatiků přetrvává představa, že to jsou systémy příliš exotické, málo výkonné a hlavně „nestandardní“. Analytici potom chybují v tom, že považují relační datový model za univerzálně platný přístup pro konceptuální modelování. [16] Znalost objektového datového modelování proto považuji v dnešní době za důležitou. Jestliže vývojáři z praxe budou znát jen relační datový model, byť jakkoli doplněný o hybridní přístupy, tak budou mít dojem, že „tabulky“ a vazby mezi nimi jsou jediný prakticky použitelný způsob analýzy, návrhu a implementace dat v informačních systémech.

Literatura:

1. Barry D.: The Object Database Handbook: How to Select, Implement, and Use Object-Oriented Databases, ISBN 0471147184
2. Beck K.: Agile Database Techniques - Effective Strategies for the Agile Software Developer, John Wiley & Sons; ISBN 0471202835
3. Bertino, E., Martino, L., Object-Oriented Database Systems, Concepts and Architectures, Addison-Wesley, 1993. ISBN 0-201-62439-7
4. Carda A., Merunka V., Polák J.: Umění systémového návrhu - objektově orientovaná tvorba informačních systémů pomocí původní metody BORM. Grada, Praha 2003. ISBN 80-247-0424-2
5. Catell R. G.: The Object Data Standard: ODMG 3.0, ISBN 1558606475
6. Delobel C., Lecluse C., Richard P.: Databases: from Relational to Object-Oriented Systems, International Thomson Computer Press, 1995. ISBN 1850321248
7. Embley D.: Object Database Development: Concepts and Principles, ISBN 0201258293
8. Gemstone Object Server - documentation & non-commercial version download, http://www.gemstone.com, http://smalltalk.gemstone.com
9. Hruška T.: Objektově orientované databázové technologie, sborník konference Tvorba softwaru 1995, ISBN 80-901751-3-9
10. Kroha P.: Objects and Databases, McGraw Hill, London 1995, ISBN 0-07-707790-3.
11. Lacko B.: Komparativní analýza strukturovaného a objektově orientovaného přístupu, sborník konference OBJEKTY 1996. ČZU Praha 1996 str.69-76
12. Lacko B.: Vademekum objektově orientované technologie, sborník konference Programování, Ostrava 1993
13. Loomis M., Chaundri A.: Object Databases in Practice, ISBN 013899725X
14. Merunka V.: Objektový databázový systém Gemstone, sborník konference OBJEKTY 2003. Ostrava 2003. ISBN 80-248-0274-0
15. Merunka V.: Objekty v databázových systémech, skriptum ČZU Praha, Praha 2002. ISBN 80-213-0318-2
16. Molhanec M.: Kritika některých chápání objektově orientovaného paradigmatu, Sborník 30. ročníku konference Tvorba softwaru, Ostrava 2004