ÚVOD DO DOTAZOVACÍHO JAZYKA OQL

Vojtěch Merunka
PEF ČZU v Praze, merunka@pef.czu.cz

ABSTRAKT
Příspěvek popisuje databázový jazyk OQL navržený mezinárodním sdružením ODMG. Jazyk OQL je doporučeným standardem pro práci s objektově-orientovanými datovými zdroji, mezi které patří také nerelační objektové databáze a distribuovaná objektově orientovaná technologie CORBA. Jde o slibný nový standard, který vzniká postupně od roku 1993. Dnes se však již dostává do praxe, protože jeho varianty se postupně objevují v některých dostupných databázových systémech a v prakticky používaných technologiích, které propojují prostředí univerzálních objektových programovacích jazyků s datovými zdroji.

KLÍČOVÁ SLOVA
Objektové databáze, OQL, SQL, ODMG, OCL, ODL, EyeDB, HQL, LINQ.

OBJEKTOVÉ 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 jsou 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 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 dnes zatím objektové databáze méně v praxi rozšířené, než relační a objektově relační, jsou pravděpodobně následující:

1. Malá praktická znalost objektových databází v komunitě tvůrců softwaru.
2. Malá dostupnost systémů na trhu a jejich vysoká cena. Velké relačně objektové systémy jsou levnější než objektové. (například komerční cena systému Gemstone je asi dvojnásobná než cena 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 není výrobci respektován. Modelovací jazyk UML přímo nepodporuje všechny konstrukce potřebné k modelování objektové databáze. Lze je ale do UML doplnit pomocí stereotypů a specifikací. Metody používané pro návrh relačních databází nejsou vhodné k plnému využití možností objektových databází.

Přes uvedené problémy stále existují důvody se domnívat, že význam objektových databází v blízké budoucnosti poroste. 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, protože v nich potřebujeme klást dotazy nad množinami obsahující prvky různých druhů (čili instance různých objektových tříd). Navíc 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 neustále vznikajících různorodých dat. Jsou to systémy 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 nerelačními 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. 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 ale situace taková, že na univerzitách v ČR se až na výjimky objektové databáze neobjevují ani ve výuce.

Ve světě je několik univerzitních pracovišť, která 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).

POTŘEBA DOTAZOVACÍHO JAZYKA PRO OBJEKTOVÉ DATABÁZE

Problematika objektových databázích je od poloviny 90. let diskutována na odborných konferencích a v odborných publikacích. Smysluplné využití objektově orientovaného datového modelu v databázových systémech proto není třeba zvlášť dokazovat. Logickým důsledkem rozdílů relačního a objektově orientovaného datového modelu je, že v objektových databázích není možné efektivně používat jazyk SQL. Přesněji jde o to, že jazyk SQL nepodporuje všechny vlastnosti, které jsou součástí objektového datového modelu.

K manipulaci s objekty se zpočátku používaly imperativní programovací jazyky C++, Smalltalk a Java. Kromě toho dlouhodobě existují snahy uvnitř samotné komunity SQL o podporu některých objektově orientovaných rysů. Je to záležitost návrhů SQL 1999 a SQL 2003. Tyto snahy jsou částečně implementovány v nejnovějších verzích objektově relačních databázových systémů. Ale ani v tomto případě nelze hovořit o plně objektově-orientovaném dotazovacím jazyku pro nerelační objektové databáze.

DOTAZOVACÍ JAZYK OQL

Dotazovací jazyk OQL (*Object Query Language*) je součástí širšího standardu sdružení ODMG, který se postupně vytváří již od roku 1993. V poslední době se již myšlenky OQL začínají prakticky prosazovat v řadě prakticky používaných technologií. Můžeme se tak setkat například s

1. jazykem OQL.NET, který je variantou jazyka OQL podle představ firmy Microsoft,
2. jazykem komponenty LINQ, která je součástí systému Microsoft .NET,
3. jazykem HQL (Hibernate Query Language), který se používá pro propojení objektů v programovacím jazyce Java s různými databázemi,
4. různými implementacemi část jazyka OQL v nerelačních objektových databázových systémech jakými je například EyeDB (použitá pro projekt mapování lidského genomu), ODABA, O2, Versant, Gemstone
5. různými implementacemi standardu CORBA, kde jazyk OQL slouží k manipulaci s distribuovanými objekty, a
6. jazyk OQL je využíván jako dotazovací jazyk v různých softwarových aplikacích jako například generátory reportů (ReportWeaver, Delphi FastReport) aplikace GIS, genové databáze, a aplikace pro Data Mining.

VLASTNOSTI OQL

Jazyk OQL je záměrně navržen tak, aby byl velmi podobný jazyku SQL-92. Dokonce platí, že část syntaxe SQL, která se týká manipulace s daty, je kompletně obsažena v OQL. Proto je například řada dotazů konstrukce `select-from-where` zcela shodná v SQL i OQL.

Na druhou stranu se jazyk OQL nezabývá tvorbou datových struktur. To znamená, že příkazy SQL týkající se definice dat (např. `create-table`, `create-index` atd.) nemají v OQL žádný ekvivalent. K definici *tříd*, *kolekcí* a *metod* jazyk OQL nejde použít. Počítá se zde buď

s nějakým běžným objektovým programovacím jazykem jako např. Java, C#, C++, Smalltalk a nebo s jiným speciálně navrženým jazykem, který se jmenuje ODL (*Object Definition Language*). Jazyk ODL je již řadu let používán pro definici objektových datových struktur standardu CORBA pro práci s distribuovanými objekty a také pochází z dílny stejného sdružení ODMG. Je proto zřejmé, že do jazyka OQL nebylo třeba přidávat konstrukce pro definici dat, když to již bylo vyřešené jiným způsobem.

Protože je OQL velmi podobný jazyky SQL, tak se jeho autoři snažili napravit některé chyby nebo nedostatky jazyka SQL. Jde o vlastnosti, které jazyk OQL má, v jazyce SQL by mohly být také, ale z historických důvodů tam nejsou:

1. OQL dovoluje používat proměnné a přiřazovací operátor (`:=`).
2. Databázovým dotazem je nejen konstrukce `select-from-where`, ale každý výraz, který reprezentuje nějaká data. Proto například prostý výraz v OQL `zaměstanci` se v SQL musí napsat jako `select * from zaměstnanci;`
3. Každý výraz v OQL je kombinovatelný s matematickými i logickými výrazy stejným způsobem jako v univerzálních programovacích jazycích. To znamená, že na rozdíl od SQL je možné na jazyk OQL nahlížet jako na funkcionální jazyk. Proto můžeme v OQL napsat například `max(select věk from osoby)`; na rozdíl od v SQL jedině možného byť nelogického `select max(věk) from osoby;`. Rovněž tak můžeme v OQL napsat sice logicky zřejmé, ale v SQL nepřijatelné výrazy jako například `select ... from první_množina union druhá_množina;` nebo `select ... from (select ...);` atp.

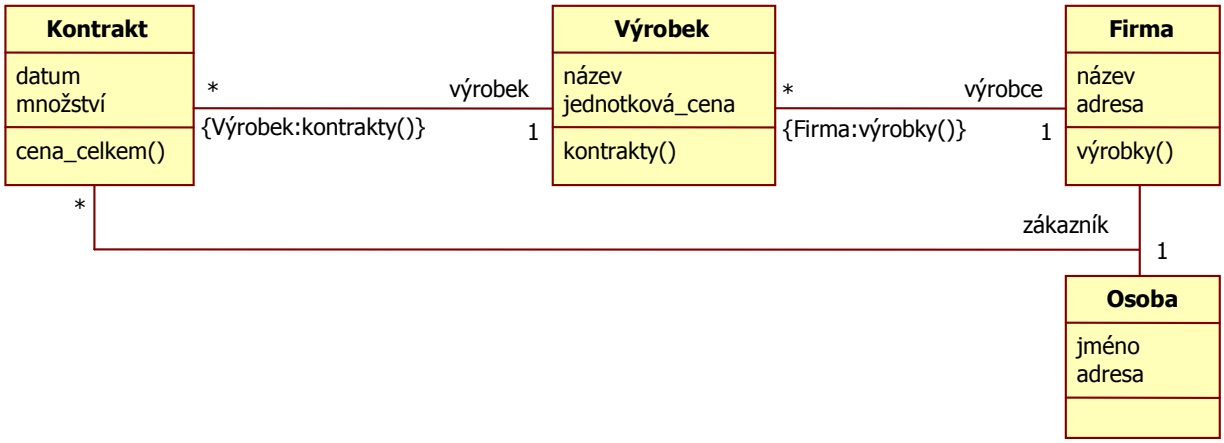
PODPORA OBJEKTOVÉHO DATOVÉHO MODELU

Vylepšení naznačená v předchozí kapitole se netýkala specifik objektového datového modelu. Hlavní síla OQL však spočívá v podpoře práce s objektovým datovým modelem způsobem, který v SQL nemá přímý ekvivalent. Pro prezentaci několika takových vlastností použijeme příklad konkrétní databáze.

příklad objektové databáze

Budeme mít objektovou datovou strukturu tvořenou čtyřmi třídami: `Kontrakt`, `Výrobek`, `Firma` a `Osoba`. Extenze těchto tříd (množiny instancí těchto tříd) si můžeme pojmenovat `kontrakty`, `výrobky`, `firmy` a `osoby`. Objekty těchto tříd jsou vzájemně propojeny, jak ukazuje obrázek č. 1.

Tato databáze může sloužit k uchovávání informací o provedených kontraktech na nějaké výrobky nějakých firem. Kontrakty mohou být sjednány jednotlivými osobami a nebo firemními zákazníky. Tato struktura může bez jakékoli další úpravy popisovat implementaci v nějaké nerelační objektové databázi, jako je například EyeDB, O2, ODABA nebo Gemstone.



Obr. 1 – příklad databáze.

Od relačního řešení se tato databáze liší v následujícím:

1. Objekty nepotřebují mít explicitně zadané *primární klíče*. Identita objektů bude totiž zajištěna interními mechanismy správy paměti naší databáze a tak nemusíme přidávat objektům žádné identifikační datové atributy, pokud je v zadání nepotřebujeme k uživatelskému zpracování. Proto jsme se bez nich v této ukázce záměrně obešli.
2. *Vazby mezi objekty* jsou zajištěny dvojím způsobem:
   a. *Přímým skládáním objektů*, kdy hodnotou atributu jednoho objektu je celý skládaný objekt a nebo kolekce skládaných objektů. Na obrázku to je vyznačeno vazbami `výrobek` a `výrobce` mezi třídami. Nejde zde v žádném případě o spojení pomocí hodnoty cizího klíče, která by se odkazovala na hodnotu primárního klíče v druhé tabulce. Například instance třídy `výrobky` mají svoje *výrobce* přímo ve svých proměnných s hodnotami celých instancí třídy `Firma`.
   b. *Odvozeným způsobem pomocí metod*, které ve svém programovém kódu vyhledávají objekty „z druhé strany vazby“. Na obrázku to jsou metody `kontrakty()` a `výrobky()`. Například instance třídy `výrobky` znají související kontrakty díky kódu metody `kontrakty()`.
3. *Podporou polymorfismu*, kdy hodnotou atributu (či jinak řečeno „objektem na druhé straně vazby“) mohou být objekty různých tříd. Na obrázku to je informace o zákaznících kontraktů, kde takovým zákazníkem může být jak nějaká osoba, tak i nějaká firma.

Je zřejmé, že kdybychom chtěli stejnou databázi realizovat v relačním nebo i objektově relačním prostředí, tak bychom se museli s těmito novými možnostmi nějak vypořádat – tedy nahradit je nějakými složitějšími relačními strukturami a nebo na ně zapomenout.

navigace, skládání objektů

Navigace je velmi silnou vlastností objektových databází, která dokonce čisté nerelační objektové databáze připodobňuje k historickým síťovým databázím (které podle mého názoru nebyly tak špatné, jak se dnes často zjednodušeně soudí). Protože zde máme objekty propojené přímo (na fyzické úrovni to samozřejmě jsou reference ve virtuální paměti), tak můžeme v OQL napsat například tento dotaz:

Najdi datumy kontraktů na výrobky vyrobené v Mladé Boleslavi.

```
select k.datum
from kontrakty k
where k.výrobek.výrobce.adresa = "Mladá Boleslav";
```

Některé verze OQL mají ještě alternativní operátor ->, který má úplně stejný význam jako běžnější tečka. Proto lze stejný dotaz napsat i tímto způsobem:

```
select k.datum
from kontrakty k
where k->výrobek->výrobce->adresa = "Mladá Boleslav";
```

metody v dotazech

Ve standardu OQL se za atributy považují i hodnoty, které vznikly výpočtem z metod. Pracuje se s nimi úplně stejně jako s hodnotami, které jsou v objektech přímo uloženy.

Budeme-li mít například v naší databázi ve třídě *Kontrakt* metodu `cena_celkem()`, která počítá celkovou cenu kontraktu jako násobek jednotkové ceny výrobku a jeho objednaného množství, tak si můžeme dovolit řadu dotazů podobných následujícímu:

Vypiš kontrakty, které byly od zákazníků z Brna a měly celkovou cenu vyšší než 10.000:

```
select *
from kontrakty k
where k.cena_celkem > 10000 and k.zákazník.adresa = "Brno";
```

Jak již bylo řečeno, tak i výsledky metod můžou tvořit datové vazby mezi objekty a také je lze považovat za atributy objektů. Proto lze například napsat následující dotaz:

Vypiš všechny výrobky, které byly alespoň 10krát objednány:

```
select *
from výrobky v
where count(v.kontrakty) >= 10;
```

tvorba objektů z výsledku dotazů

Tvorba nových objektů (instancí předem definovaných tříd) je velmi jednoduchá a vychází z běžně používaných objektových programovacích jazyků. Stačí napsat název třídy a do závorky napsat konkrétní hodnoty atributů pro novou instanci této třídy:

```
nová_firma  :=  Firma(název: "TPCA" , adresa: "Kolín");
```

Toto lze samozřejmě použít i v dotazech, pokud budeme chtít vytvářet nové objekty z dat, která vznikla jako výsledek dotazu. Můžeme tak třeba rovnou vyrobit kolekci nových faktur (máme-li předem naprogramovanou třídu *Faktura*, samozřejmě) pro brněnské zákazníky, kteří si něco objednali určitý den:

```
select Faktura(datum: k.datum , zákazník: k.zákazník, ...)
from kontrakty k
where k.datum = 10-03-2009 and k.zákazník.adresa = "Brno";
```

anonymní třída Struct

V některých případech potřebujeme vytvořit nové objekty (například z důvodu jejich exportu z databáze do datového prostředí na straně klienta), ale nemáme tam vytvořenou novou třídu pro takové nově vytvořené objekty. Proto standard OQL podporuje zvláštní univerzálně použitelnou třídu *Struct*, která se chová jako n-tice obsahující proměnlivou strukturu podle potřeby. Například dotaz

```
select struct(k.výrobek.výrobce.adresa , k.zákazník.adresa)
from kontrakty k
where k.datum = 10-03-2009;
```

Tvoří se tím nové objekty, které budou obsahovat uspořádané dvojice hodnot *adresa výrobce* a *adresa zákazníka*. Takové objekty se nám mohou mapovat v programovacím jazyce na straně klienta na dvouprvková pole.

hierarchičnost versus plochost dat

Relační databáze jsou na rozdíl od objektových vždy ploché. Objektová databáze ale může být jak plochá, tak i hierarchická. Představíme-li si například dotaz

```
select f.výrobky
from firmy f
where f.adresa = "Kolín";
```

tak dostaneme všechny výrobky všech firem z Kolína. Nebude to ale kolekce, ve které by všechny nalezené výrobky byly uloženy jako její prvky. To, co dostaneme, bude kolekce, která bude jako svoje prvky obsahovat další dílčí kolekce (každou za každou jednotlivou firmu) a požadované výrobky budou až prvky v těchto vnitřních dílčích kolekcích. Výsledkem je tedy hierarchická tříúrovňová struktura:

1. první úroveň je celý výsledek,
2. druhá úroveň vnitřních kolekcí odpovídá původním jednotlivým firmám a
3. třetí úroveň jsou požadované výrobky.

Tato „hierarchičnost“ může být někdy k užitku, protože zachovává podobu původní struktury ze které byla vytvořena. Na druhou stranu je ale zřejmé, že je v mnoha případech jen na obtíž, a tak v OQL musí existovat mechanismus, jak prvky z výsledku „sesypat“ do jedné velké kolekce. K tomu v OQL slouží funkce *flatten*:

```
flatten(select f.výrobky
        from firmy f
        where f.adresa = "Kolín");
```

ZÁVĚR

Jazyk OQL je slibně se rozvíjejícím standardem pro dotazovací jazyk nové generace databázových systémů. Jeho formální podobnost s jazykem SQL je mu velkou výhodou. Do blízké budoucnosti lze očekávat různé možné (a snad i vzájemně kombinovatelné) varianty vývoje:

1. Dojde ke konvergenci jazyka OQL s nějakou novou verzí jazyka SQL, jak již naznačuje vývoj nových verzí jazyka SQL.
2. Jazyk OQL zatím dal vzniknout řadě sobě podobných ale navzájem nekompatibilních proprietárních dotazovacích jazyků konkrétních objektových databází na trhu, což povede k potřebě standardizace mezi nimi.
3. Jak naznačuje projekt LINQ od Microsoftu, tak principy jazyka OQL se stanou standardně využívanou součástí nějaké široce používané technologie jako je například .NET, přičemž může dojít ke změně dosavadní syntaxe.
4. Může dojít ke konvergenci jazyka OQL s jazykem OCL, který je součástí standardu UML a slouží k definici pravidel a dotazů nad objektovým konceptuálním modelem. Je zde patrný velký průnik vlastností jazyků OQL a OCL, který ale tyto dva sémanticky velmi podobné jazyky řeší různými syntaxemi. Jazyk OQL má syntaxi téměř shodnou s jazykem SQL, ale jazyk OCL má syntaxi podobnou univerzálním objektovým programovacím jazykům (Smalltalk, C++, …) a lambda-kalkulu.

Tento článek obsahuje látku podporovanou grantem MSM6046070904 na výzkum v oblasti znalostních databázových systémů.

LITERATURA

ACE DB – *a genome database*, http://www.acedb.org.

CATELL, R.G. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, ISBN 9781558606470

EyeDB – *Open Source Object Database*, http://www.eyedb.org

HIBERNATE – *The Persistence for Java and .NET*, http://hibernate.org

LINQ – *.NET Language-Integrated Query*, http://msdn.microsoft.com/cs-cz/library/bb308959(en-us).aspx

MERUNKA, V. *Objektové Modelování*. Alfa Publishing. Praha 2008, ISBN 978-80-87197-04-2

ODMG, *The Object Management Group*. http://www.odmg.org