

Boj se zavlečeným impedančním nesouladem na úrovni databáze

ABSTRACT: Impedanční nesoulad může být zmírněn správnou volbou databázové technologie. Článek vysvětluje, co to impedanční nesoulad je a uvádí jednoduché příklady jeho dopadu na vývoj aplikace. S ohledem na tento impedanční nesoulad dále analyzuje klady a zápory tří typů databází: *relační, objektové a Caché*, vícerozměrné databáze společnosti InterSystems.

Mary A. Finn
marketingový manažer produktu

Úvod

Objektově orientované programování se díky vyzrálosti jazyka Java a jeho širokému přijetí dostalo do popředí oblasti vývoje aplikací. Současní vývojáři dávají přednost objektovým technologiím, jako jsou Java, C++ a COM, pro jejich bohaté datové modely a podporu konceptů zvyšujících produktivitu, jako jsou například zapouzdření, dědění a polymorfismus.

Na celém světě je většina dat dosud uchovávána v relačních databázích. Vývojáři databázových aplikací (aplikací, které přistupují k uloženým datům) často zjišťují, že bojují s **impedančním nesouladem**, tedy s *vrozenou odlišností mezi objektovým a relačním datovým modelem*.

Snaha omezit problémy, plynoucí z impedančního nesouladu, vede programátory k různým řešením. Nejčastěji jde o využití tzv. „*objektového mapování relačních dat*“. Ani toto řešení však není ideální. Zpravidla totiž vede k snížení produktivity programování i výkonu aplikace.

Objasnění termínu impedanční nesoulad

Impedanční nesoulad je termín původně z elektrotechniky, který v softwarovém světě označuje rozdíl plynoucí z podstatné odlišností mezi relačním a objektovým modelem databáze.

Jednoduše řečeno: **relační model** uspořádá data do řádků a sloupců. Každý řádek představuje záznam, sloupce představují různé datové položky v záznamu. Jsou-li data příliš složitá, než aby se dala reprezentovat v dvourozměrné mřížce, musí se vytvořit dodatečné tabulky, které uchovají vztahové informace. Každá tabulka v takovém relačním schématu pak obsahuje některé, ale ne všechny, datové položky pro velmi mnoho záznamů.

Objektový model není omezen na uchovávání dat v řádcích a sloupcích. Místo toho návrhář vytvoří definici (šablonu), která úplně popisuje určitou třídu informací. Každý objekt (záznam) je specifickou instancí této třídy. Každý objekt tedy obsahuje všechny datové položky právě jednoho záznamu. To ale není všechno. Definice tříd mohou také obsahovat úseky kódu (tzv. metody), které pracují s daty popsány třídou. V relačním modelu podobný způsob záznamu neexistuje.

Jednoduchý příklad

Pro ilustraci rozdílu mezi těmito dvěma datovými modely předpokládejme, že vyvíjíte aplikaci pro pohledávky. Tato aplikace bude bezpochyby muset sledovat mnoho faktur, z nichž každá bude obsahovat informace záhlaví (např. datum a číslo faktury) a jednu nebo více řádkových položek. Každá řádková položka bude kromě jiného obsahovat informace o objednaném výrobku a o jeho množství.

Jednou z možností modelování faktury v relační databázi je vytvoření dvou tabulek. Jedna tabulka, nazvaná `Faktura`, obsahuje informace záhlaví, které se na každé faktuře objeví jen jednou. Druhá tabulka, nazvaná `RadkovePolozky`, obsahuje sloupce `Zdrojova_Faktura`, `Kod_Vyroбку_Radkove_Polozky` a `Mnozstvi_Radkove_Polozky`. První z těchto položek je zvláště důležitá, protože obsahuje tu hodnotu, která „svazuje“ řádkové položky s informacemi obsaženými v tabulce `Faktura`.

Jak vidíme, ani jedna tabulka neobsahuje všechny informace o dané faktuře. Každá tabulka však obsahuje některé informace o mnoha fakturách. Pokud navrhujete aplikaci, která má např. tisknout faktury, musí přistupovat k oběma tabulkám, k tabulce `Faktury` pro informace záhlaví a k tabulce `RadkovePolozky` pro podrobné informace. Uvědomte si také, že tyto tabulky neobsahují žádné pokyny o formátování dat pro tisk. Tyto pokyny budou uloženy mimo samotnou databázi.

V objektovém modelu nemusejí být data uspořádána do řádků a sloupců, takže definice třídy `Faktura` bude představovat seznam všech datových položek, které dohromady vytvářejí fakturu. Budou zde vlastnosti obsahující informace záhlaví, jako je např. `Datum_Faktury`, `Cislo_Faktury`, atd. a kolekce jedné nebo více instancí třídy `Radkova_Polozka`. Třída `Radkova_Polozka` zahrnuje vlastnosti `Kod_Vyroбку_Radkove_Polozky` a `Mnozstvi_Radkove_Polozky`.

Definice tříd jsou pouhými podrobnými návrhy formátu dat. Každá jednotlivá faktura je určitou instancí třídy `Faktura` a obsahuje určité instance třídy `RadkovaPolozka`, které do ní náleží. Každý objekt `Faktura` tedy obsahuje veškeré informace pro danou fakturu a pouze pro tuto fakturu.

Definice tříd mohou ale také obsahovat metody, které pracují s daty popsanými třídou. Třída `Faktura` může obsahovat například metodu `Tisk()`, která určuje, jak naformátovat údaje faktury pro tisk. Trvalé objekty budou zahrnovat nějakou metodu `Ulozit()`, která určuje, jak jsou objekty do databáze ukládány. Výchozí nastavení metody `Ulozit()` bude určeno strukturou databázového stroje a zajištěno dodavatelem databáze.

Impedanční nesoulad při práci s databází

Předpokládejme, že v aplikaci pro pohledávky vytváříme novou fakturu s jednou řádkovou položkou. Pokud byste programovali v prostředí **relační databáze**, byl by kód podobný kódu uvedenému v příkladu č.1. Obsahoval by dva příkazy `Insert` (Vložit): jeden pro přidání informací záhlaví do tabulky `Faktury` a druhý pro přidání podrobných informací do tabulky `RadkovePolozky`. Příkaz `Insert` je standardním příkazem jazyka SQL, jehož implementaci poskytne dodavatel relační databáze.

Příklad č.1 : Vytvoření nové Faktury s použitím relačního modelu

```
If (flag="Nove") {  
  Insert Into Faktura  
    (Datum_Faktury, Cislo_Faktury)  
  Values(Today, :NoveCisloFaktury)  
  Insert Into RadkovePolozky  
    (Zdrojova_Faktura, Mnozstvi_Radkove_Polozky, Kod_Vyroбку_Polozky_Radku)  
  Values (:NoveCisloFaktury, :Mnozstvi, :ObjednanyVyrobek)  
}
```

Kód pro uložení faktury s jednou řádkovou položkou pomocí **objektového modelu** je uveden v příkladu č.2. Až na syntaktické podrobnosti vypadá podobně jako kód v příkladu pro relační prostředí. Hlavní rozdíl je v tom, že metoda `Ulozit()` je volána pouze jednou.

Příklad č.2: Vytvoření nové Faktury s použitím objektového modelu

```
If (flag="Nove") {  
  objInv=nova Faktura()  
  objInv.DatumFaktury=Today  
  objInv.CisloFaktury=NoveCisloFaktury  
  objLI=nova ObjInv.RadkovaPolozka()  
  objLI.MnozstviRadkovePolozky=Mnozstvi  
  objLI.KodVyroбку=ObjednanyVyrobek  
  objInv.Ulozit()  
}
```

Představte si, že obchodní logiku pro aplikaci chcete napsat v objektově orientovaném jazyku, jako je například Java nebo C++, ale data potřebujete uložit v relační databázi. Toho u faktury dosáhnete tak, že musíte naprogramovat příkazy `Insert` jazyka SQL v rámci metody `Ulozit()` v definici třídy `Faktura`. Toto je jeden z projevů impedančního nesouladu. Třída objektu s kolekcí musí být převedena do nesourodých tabulek stroje relační databáze.

Impedanční nesoulad při návrhu

Jiná forma impedančního nesouladu se může neočekávaně objevit v průběhu návrhu. Kromě možnosti bohatšího a intuitivnějšího způsobu modelování dat zahrnuje totiž objektová technologie řadu konceptů, které významně zvyšují programátorskou produktivitu. Objektová technologie podporuje zejména koncepty dědičnosti a polymorfismu.

DĚDIČNOST spočívá v tom, že definice jedné třídy může být odvozena od jiné třídy. Například v aplikaci pro pohledávky můžete vytvořit obecnou třídu `SablonaFaktury` a mít specifické třídy `FakturaSoftwaru` a `FakturaHardwaru`, které dědí vlastnosti a metody ze třídy `SablonaFaktury` (Tyto třídy mohou také obsahovat nezděděné vlastnosti a metody, které jsou pro každou třídu charakteristické). Pokud vznikne během vývoje aplikace nutnost změnit třídu `SablonaFaktury`, dědičnost zajistí, že jsou tyto změny automaticky promítnuty do definicí tříd `FakturaSoftwaru` a `FakturaHardwaru`.

POLYMORFISMUS spočívá v tom, že různé implementace metody mohou sdílet společné rozhraní. Například, metoda `Tisk()` může ve třídách `FakturaSoftwaru` a `FakturaHardwaru` obsahovat různé příkazy pro formátování. Chcete-li však fakturu vytisknout, aplikace musí pouze načíst objekt do paměti a vyvolat jeho metodu `Tisk()`. Díky polymorfismu bude objekt „vědět“, jak má být naformátován pro tisk podle toho, ke které třídě patří.

V relačním modelu neexistuje ani dědičnost, ani polymorfismus. Někteří prodejci velkých relačních databází, jako jsou společnosti Oracle, Microsoft a IBM, se pokoušeli implementovat koncepty objektově orientovaného návrhu, ale výsledky nesplnily očekávání programátorů.

Metody zmírnění impedančního nesouladu

Oba výše uvedené příklady impedančního nesouladu jsou velmi jednoduché, ale poslouží pro předvedení problému. Práce, kterou vyžaduje „normalizace“ impedančního nesouladu, může být značná a prudce roste se zvětšující se složitostí aplikace. Dopad impedančního nesouladu ale může být podstatně snížen vhodnou volbou databázové technologie. Uvažujme tři možnosti ukládání dat: relační databáze, „čistá“ objektová databáze a vícerozměrná databáze společnosti Caché.

I. POUŽITÍ RELAČNÍ DATABÁZE

V tomto článku už bylo uvedeno, jak může pokus o použití relační databáze s aplikací založenou na objektové technologii vést k závažným problémům s impedančním nesouladem. Vývojáři ale někdy nemají jinou možnost. Potřebují přistupovat k datům, které jsou uloženy v nějaké relační databázi. V tomto případě bude jednou z možností použití nástroje „objektově relačního mapování“, buď jako nástroje samostatného nebo jako vestavěného do tzv. „objektově relačních“ databází.

Mapovací nástroje vytvářejí soubor (mapu), který obsahuje kód pro převod mezi objekty a relačními tabulkami. Vývojáři musí přesně určit, jak má být tento převod proveden, tj. které vlastnosti objektu odpovídají kterým sloupcům dat ve kterých tabulkách a naopak. Vytvořená mapa je uložena a používá se vždy, když aplikace přenáší data do databáze nebo z ní. Nástroje objektově relačního mapování spravují vyrovnávací paměť a pomáhají omezovat snížení výkonnosti související s převodem dat mezi objekty a relačními formuláři.

Objektově relační mapování však může významně zpomalit vývoj aplikace, nehledě na problémy s výkonem při zpracování. Většina mapovacích nástrojů neimplementuje koncepty objektového modelování, jako jsou dědičnost, polymorfismus atd., nebo je implementuje pouze částečně. Navíc, jestliže je aplikace přizpůsobována a upravována, je třeba vytvářet nové aktualizované objektově relační mapy.

Budou-li vývojáři, bojující s impedančním nesouladem mezi objektově orientovanými aplikacemi a relačními databázemi, uvažovat o přenesení dat do úložiště, které pracuje s objekty lépe, měli by bezesporu zvážit jednorázové úsilí potřebné pro přeformátování a přenesení dat a proti tomu nepřestávající aktualizační práci a ztráty výkonu při používání objektově relační mapy.

II. POUŽITÍ OBJEKTOVÉ DATABÁZE

Na první pohled by se mohlo zdát, že impedanční nesoulad lze zcela vyloučit uložením dat do „čisté“ objektové databáze. Objektově orientovaná aplikace snadno spolupracuje s objektovou databází. To ale platí jen částečně. I v tomto případě totiž vzniká impedanční nesoulad, budete-li chtít v této databázi spouštět dotazy SQL.

Jazyk SQL je nejpoužívanější dotazovací jazyk na světě, který vychází z toho, že data jsou uložena v relačních tabulkách. Někteří dodavatelé objektových databází poskytují přístup k datům prostřednictvím některého z objektových dotazovacích jazyků (*object query language* – OQL), ale tyto jazyky nejsou příliš rozšířené. Aby byla zajištěna kompatibilita s běžnými aplikacemi pro analýzu dat a vytváření sestav, musí objektová databáze podporovat rozhraní ODBC a JDBC a z toho důvodu musí poskytovat určitý mechanismus pro projektování dat ve tvaru relačních tabulek.

Typickým řešením bývá opět mapování. Nevýhody mapování ale stále platí: ztráty výkonu a nedostatky podpory pro vývoj datového modelu. Naproti tomu je mapu nutno vyvolat pouze tehdy, když je v databázi spuštěn dotaz SQL.

III. POUŽITÍ VÍCEROZMĚRNÉ DATABÁZE CACHÉ S UNIFIKOVANOU DATOVOU ARCHITEKTUROU

Pro uložení dat existuje i třetí možnost. Caché, vícerozměrná databáze společnosti InterSystems. Ačkoliv se o vícerozměrných databázích často uvažuje v souvislosti s prostředím datových skladů, je systém Caché navržen tak, aby mohl být základem transakční aplikace. Kromě toho implementuje jedinečný prostředek snižující impedanční nesoulad - *unifikovanou datovou architekturu*.

Díky unifikované datové architektuře „sdílejí“ objektové a relační datové modely vícerozměrná data systému Caché. Vícerozměrná pole se snadno zobrazí jako tabulky, které nejsou vlastně nic jiného než dvourozměrná pole. Podobně mezi objekty a vícerozměrnými poli je jednoduchá korelace, protože ani objekty ani vícerozměrná pole nejsou omezena na řádky a sloupce, jako je tomu u relační technologie. Převod mezi datovými formáty je automatizován a je součástí kompilované definice dat. Z pohledu vývojáře je každá tabulka objektem a každý objekt je jednou nebo více tabulkami.

Některé další vlastnosti unifikované datové architektury Caché

- **Plný souběh**
Aktualizace dat, prováděné pomocí relačního rozhraní, jsou okamžitě přístupné prostřednictvím objektového rozhraní a naopak.
- **Podpora postupných změn datového modelu**
Automaticky se projevují změny definice datové struktury v objektové v relační části.
- **Plná podpora jazyka SQL**
Jsou podporovány veškeré příkazy DDL, DML a DCL jazyka SQL.

- **Plná podpora objektů**

Jsou podporovány všechny koncepty objektového modelování, jako jsou jednoduchá a vícenásobná dědičnost, polymorfismus, rozšířené datové typy a generátory metod.

- **Obsluha objektů**

Objekty definované v unifikované datové architektuře mohou být vystavovány jako objekty jazyků Java, C++ nebo rozhraní COM. Tím je zajištěna kompatibilita se širokou škálou objektově orientovaných technologií.

Unifikovaná datová architektura může významně snížit impedanční nesoulad, ale nemůže ho odstranit úplně. Některé koncepty není možno automaticky sdílet, například objektové metody a relační trigger. Přesto však zůstává systém Caché nejlepší volbou pro vývojáře, kteří chtějí kombinovat objektový a relační přístup.

Komentář k trendům v této oblasti

V posledních letech se stávají objektově orientované programovací jazyky, jako jsou Java, C++ a rozhraní COM, dominantními technologiemi pro vývoj aplikací. Proto potřebují vývojáři databázových aplikací stavět data jako objekty.

Jazyk SQL je však stále dominantní technologií pro analýzu dat a vytváření sestav. Aby byla databáze použitelná, musí umožňovat stavět data v relačním formátu tabulek, ke kterému lze přistupovat prostřednictvím rozhraní ODBC a JDBC.

Impedančnímu nesouladu, tj. rozdílu, který plyne z odlišností mezi objektovými a relačními datovými modely, se nelze vyhnout. Ale může být významně zmírněn volbou vhodné databázové technologie. Při vývoji nových aplikací je výhodné uložení dat do vícerozměrné databáze jako je *Caché*, která díky své unifikované datové struktuře umožňuje stavět data zároveň jako objekty i tabulky.

Pro budoucí vývoj aplikace, pro kterou jsou již existující data uložena v relační databázi, by měli vývojáři zvážit možnost převodu dat do vícerozměrné datové struktury. Jednorázovým úsilím při převodu dat se vyhnou ztrátám výkonu a potížím při vývoji a aktualizacích datového modelu, které jsou vlastní objektově relačnímu mapování.