Metody návrhu a implementace databázového stroje OO databáze.

Karel Obluk

1. Úvod

V současnosti nejrozšířenější a teoreticky nejlépe propracovaný databázový model je bezpochyby model relační. Přesto, nebo možná právě proto, jsou velmi dobře známy i některé jeho slabiny. Jedním z možných směrů dalšího vývoje, který se v poslední době stále více prosazuje, je objektově orientovaný model. Pokouší se řešit některé problémy spojené s výkonnosti a zejména je snahou o zvýšení míry abstrakce. Prezentovaná práce popisuje některé metody použité při návrhu databázového stroje objektově orientované databáze VEMA, které přinášejí poněkud netradiční řešení známých problémů.

2. Objektový model

Přestože ve světě existuje již celá řada OO databází a ještě více prací na stejné téma, prakticky žádní dva autoři se neshodnou na detailech popisu OO modelu. Na rozdíl od relačního modelu, který je teoreticky velmi detailně propracován a podložen standardy, různí autoři chápou různě i některé základní aspekty OO modelu. Je také nutno odlišit OO model databáze od nejrůznějších objektových prostředků pro práci s relační databází (například objektové rysy systémů PowerBuilder, SQL Windows nebo nejnověji CA-Windows). Rozsah příspěvku mi nedovoluje popsat detailně chápání vlastností objektového modelu databázového systému VEMA, zmíním se proto jen o jeho základních rysech - tak jak se na nich shodne většina autorů. Zájemce o podrobnější rozbor problematiky nalezne více informací například v [1] nebo [2]. Při návrhu databázového modelu jsme v první fázi vycházeli zejména z návrhu **Object Database Management Group** (ODMG) ve verzi 1.1 [2], který jsme obohatili o některé další prvky.

Objektový model vychází ze snahy o posunutí úrovně abstrakce na co nejvyšší úroveň. Objektová databáze se skládá ze samostatných objektů, každý objekt má svou jednoznačnou identifikaci (Object ID - **OID**), prostřednictvím které se na něj odkazují jiné objekty. V čistě objektovém modelu je vazba přes OID jediným prostředkem pro vyjádření vztahů (relací). Každý objekt obsahuje atributy (t.j. samotná data) a metody (t.j. popis způsobu práce s objektem). Dalším významným prvkem objektového modelu je zapouzdření a dědičnost. Naše rozšíření OO modelu jak jej prezentuje ODMG zahrnovala zejména vícenásobnou dědičnost, virtuální metody a možnost zanořování objektů (t.j. položkou objektu může být objekt). Databázový systém VEMA také zcela nezavrhuje relační vazby a zejména databázový stroj a databázový server musí být navrženy s ohledem na časté vyhodnocování relačních operací, které mohou být výhodnější z hlediska duplicity dat a operací jako je např. JOIN.

3. Požadavky na databázový stroj VEMA (DS VEMA)

Databázový stroj je jednou z klíčových částí databázového systému a jeho návrh proto musí být specifikován již v samotných začátcích projektu. Nezávisle na modelu databáze jsou však některé požadavky zcela zřejmé a problémem zůstává jen jejich implementace. Kromě samozřejmého předpokladu spolehlivosti a co nejvyšší výkonnosti se v úvodní studii zmiňují i možnosti údržby databáze za provozu, což zejména u operací jako je reorganizace není vůbec triviální problém. Striktní byl i požadavek podpory zanořených transakcí, co nejvyšší míry paralelnosti a do budoucna i podpora distribuovaných databází. Z použití OO modelu vyplývá nutnost údržby jednoznačných OID, z požadavků na výkonnost pak údržba klíčů. Mezi další vlastnosti, které v tomto příspěvku nejsou blíže rozvedeny, patří například hierarchický podhled na data (datový typ "strom") nebo tzv. nálepky (krátké popisky objektů).

4. Objektový identifikátor a objektový prostor

Celá databáze je rozdělena do několika objektových prostorů. **Objektový prostor** je souvislá persistentní oblast, ve které je zajištěna unikátnost objektových identifikátorů všech jejich objektů. Přestože ve většině případů bude použit jen jeden objektový prostor, může rozčlenění databáze na více objektových prostorů přinést rychlejší přístup k objektům zejména ve velmi rozsáhlých databázích nebo ve složitějších konfiguracích databáze. Je vhodné zdůraznit, že vnitřní rozdělení databáze na objektové prostory ještě neznamená distribuovanost databáze jako takové. Rozdělení na více objektových prostorů je jen implementační nebo optimalizační problém.

Objektový identifikátor (**OID**) určuje jednoznačně každý objekt. Část OID určuje objektový prostor, druhá část identifikuje objekt v rámci objektového prostoru. Je zřejmé, že algoritmus mapování OID na adresu objektu je kritický pro výkonnost celé databáze.

Zvláštním druhem objektového prostoru je tzv. **velký objekt**. Velký objekt je objekt, jehož součástí jsou další objekty. Tyto objekty se mohou odkazovat vzájemně jeden na druhý (v rámci velkého objektu) a bylo by velmi neefektivní je ukládat (a odkazovat se na ně) samostatně. Proto jsou vnitřní malé objekty uloženy jako součást velkého objektu a jejich OID jsou lokální vzhledem k němu. Při načítání velkého objektu do paměti jsou pak všechny odkazy na lokální OID převedeny na paměťové reference; naopak při ukládání na disk jsou paměťové reference převáděny na lokální OID. Tyto převody jsou velmi složité zejména ve vazbě na vícenásobnou dědičnost a zanořování objektů; kvůli těmto rozšířením výchozího ODMG modelu nelze použít žádnou z metod známých zejména ze síťového modelu databází. Ve chvíli kdy vzniká tento příspěvek ještě zmíněný problém není uspokojivě vyřešen.

5. Verze objektů

Jedním z významných prvků DS VEMA z hlediska řešení zanořených transakcí a vysoké míry souběžného zpracování je **verzování objektů**. Verzování objektů může být v zásadě dvojího typu. Podle toho, zda dovoluje existenci více paralelních verzí (tedy jakési větvení) nebo zda předpokládá jen jednu linii existence objektu. První varianta je velmi podstatná pro distribuované databáze, její řešení však přináší řadu problémů spojených s ošetřením slučování paralelních verzí. Při sloučení vznikají komplikace zejména při ošetření chyb (slučování probíhá často až mnohem později, než vznikly jednotlivé verze objektu a při chybě a případně požadovaném rollbacku některé verze je problematická hlavně vazba na reálný svět), nezanedbatelné jsou problémy s implementací metod pro sloučení verzí bez uživatelské interakce (častý požadavek, který je však někdy prakticky neřešitelný). První verze DS VEMA implementuje druhou zmíněnou metodu verzování objektů. V daném okamžiku existuje jen jedna "nejnovější" verze objektu, podstatná je však současná existence této nové verze a také několika starších verzí téhož objektu. O tom, která verze objektu je viditelná pro kterého klienta rozhoduje DS podle kontextu. Tak může DS implementovat velmi efektivně zanořené transakce, navíc tato metoda umožňuje dosahovat vysokou míru souběžného zpracování i při dodržení nejvyššího stupně izolace (SERIALIZABLE jak ji definuje SQL). Tento princip je velmi blízký metodě tzv. *shadow pages* (např. viz. [3]). Samozřejmě má i své nevýhody, mezi které patří složitější implementace a zejména nutnost pravidelné reorganizace pro odstraňování starých (a nepotřebných) verzí objektů. Koncept samostatných objektových prostorů však umožňuje implementaci reorganizace databáze za běhu, protože lze reorganizovat objektový prostor kopírováním do nového. Stačí jen zajistit překlad OID ze starého na nový objektový prostor a vše ostatní je jen ošetření běžných odkazů na databázi. Za cenu nepatrného snížení výkonu při reorganizaci tak uživatel získává mocné nástroje v podobě zanořených transakcí a možnost nejvyšší míry izolace klienta dokonce i po celou životnost procesu (nejen transakce).

6. Transakce

Transakce je dostatečně známý prostředek pro zajištění konzistence databáze. Jednou z modifikací transakčního schematu je princip zanořených transakcí. Na jedné straně představuje velmi mocný prostředek v rukou vývojáře, na druhou stranu však řada autorů označuje impelementaci za velmi složitou nebo neefektivní. Některé systémy řeší problém dlouhých transakcí implementací tzv. bodů návratu (v anglickém originále *savepoint*), t.j. bodů do kterých se lze při chybě vrátit, ovšem které na rozdíl od operace COMMIT nezviditelňují změny provedené transakcí pro ostatní procesy. Naproti tomu je řešení pomocí verzování objektů velmi elegantní a nepotřebuje žádné podobné speciální operace; zůstává splněna podmínka, že verze objektu je pro ostatní procesy viditelná, jen pokud nad daným objektem není rozpracována žádná transakce.

Při zahájení transakce se do tabulky transakcí procesu zapíše nová položka. Každá položka transakční tabulky potom obsahuje malé pomocné pole, do kterého se ukládá několik prvních OID objektů změněných v transakci. Každý objekt změněný v transakci má navíc v mapě objektů (viz. dále) nastaven příznak rozpracované transakce. Operace COMMIT resp. ROLLBACK potom projde všechny objekty registrované v položce transakční tabulky a buď jim nastaví příznak platnosti (COMMIT) nebo naopak neplatnosti (ROLLBACK). Pokud bylo v rámci transakce změněno více objektů než byla kapacita pole v položce transakční tabulky, je zjistit které objekty byly rozpracovány v ukončované transakci sekvenčním průchodem mapy objektů. Tato varianta je sice časově náročenější, nicméně nepředpokládá se, že by byla příliš běžná. Typický počet objektů změněných v transakci by neměl přesáhnout 80 až 100.

Problémy s ošetřením zanořených transakcí tak zůstávají jen v oblasti indexů (klíčů). Pokud má být koncept zanořených transakcí důsledný a pokud má být současně funkční jeden ze základních funkčních rysů klíčů - t.j. zajištění jednoznačnosti, objevuje se požadavek zavedení verzování i v klíčích. Jiné metody by neúměrně zatížily buď vyhodnocování viditelnosti objektu nebo operaci COMMIT a ROLLBACK resp. obojí.

7. Mapování OID

Jak jsem se již zmínil výše, je algoritmus mapování OID na adresu objektu jedním z kritických míst z hlediska výkonnosti celého systému. Musí zajistit jednak nalezení objektu, ale navíc ošetřit výběr správné verze objektu z hlediska izolace procesu a rozpracovaných transakcí. Objekt jiného procesu je viditelný pokud nad ním není rozpracována žádná transakce a samozřejmě pokud k němu má čtecí proces povolen přístup. Tento základní požadavek je podle požadované míry izolace procesu případně rozšířen o další podmínku. Pokud je míra izolace "READ COMMITTED" (jak je definována v SQL), žádná další omezení neexistují. Pokud je však míra izolace nejvyšší ("SERIALIZABLE" dle SQL), je v dalším kroku porovnán čas vytvoření verze objektu s časem přihlášení čtecího procesu. Verze objektu je potom viditelná jen pokud byla vytvořena před přihlášením čtecího procesu. Přihlášením zde může být jednak otevření databáze procesem nebo okamžik požadavku "občerstvení dat". Tyto časové údaje by se daly chápat jako okamžiky zahájení a dokončení speciálního typu "čtecí" transakce. Hledání správné verze objektu pokračuje, dokud není nalezena viditelná verze objektu nebo dokud nejsou porovnány všechny existující verze objektu.

Z dosavadního popisu je zřejmé, že v každém objektovém prostoru existuje tabulka, ve které jsou shromážděny všechny údaje nutné pro nalezení správné verze objektu dle OID a kontextu práce procesu. Přestože v objektových databázích je většinou počet objektů výrazně menší než kardinalita relací v typických relačních databázích, může i u zde dojít k rychlému růstu množství jednotlivých objektů a jejich verzí. Proto není možné spoléhat na nějaký typ jednoduché, paměťově rezidentní tabulky, ve které by i operace INSERT bylo možné implementovat prostým odsunutím konce tabulky. Zejména požadavek verzování si vynucuje některé jiné implementační metody. V daném okamžiku se jako nejslibnější jeví kombinace tabulky pro větší

konstantní část databáze a tabulky s rozptýlenými položkami pro přidávání nových verzí. Experimentálně jsme si totiž ověřili, že převážná část databáze je od určitého objemu dat konstantní a přístupy k ní se omezují na čtení. Jen malé procento dat je měněno, přibývající data (t.j. nové objekty) mohou být přidávány do "tabulkové" části mapy. Staré a nepotřebné položky z části TRP mohou být rušeny i automaticky při vytváření nových a není nutné čekat až na reorganizaci.

8. Žurnál

Žurnál (v angličtině nazývaný **log file**), je nutnou součástí databázového systému. Na rozdíl od většiny současných systémů, které jej využívají hlavně pro implementaci transakcí a ošetření chyb, systém VEMA zajišťuje tyto operace pomocí verzování objektů. Žurnál si tak uchovává převážně svou funkci protokolu o zásazích do databáze. Lze z něj jednak rekonstruovat stav databáze v libovolném okamžiku, jednak může být použit pro zjištění odpovědnosti za změny v databázi (tedy téma v angličtině nazývané Security). Vzhledem k obrovskému objemu dat, která jsou do žurnálu ukládána, bude pravděpodobně žurnál DS VEMA komprimován. Odkazy na něj totiž budou spíše výjimečné a hlavním problémem se zřejmě stane kapacita paměťových médií. Díky verzování objektů také není nutné ukládat do žurnálu všechny změny okamžitě, ale je docela dobře možné odložit aktualizaci žurnálu až na okamžiky ukončování transakcí.

9. V/V operace

Posledním problémem, kterému se chci ve svém příspěvku věnovat, je problematika V/V operací. V databázové literatuře toto téma dosti opomíjeno, s odkazem na rozdílnou povahu V/V v různých operačních systémech. Podle mého názoru je však výkonnost V/V tak podstatná, že ji nelze přeskočit pouhým odkazem na operační systém.

DS VEMA bude implementován již v první fázi na několika platformách, zejména v prostředí OS SCO Unix, UnixWare a Windows (NT); v blízké budoucnosti by měl přibýt OS/2 a další varianty systémů Unix. Je zřejmé, že se nelze spolehnout na prostředky některé ze zmíněných platforem, protože co je k dispozici v jednom systému, nemusí být implementováno v jiném. Navíc i když dva dělají totéž, nemusí to být vždy totéž. Příkladem může být paměťově mapovaný soubor, který je ve Windows NT výslovně doporučován jako nejlepší (nejrychlejší, nejspolehlivější, nej...) prostředek pro V/V operace. Unix sice paměťově mapované soubory nabízí také, je však otázka zda jsou z hlediska výkonu systému jako celku skutečně tou nejlepší variantou.

Stejně tak se není možné spolehnout na implicitní vyrovnávací paměť implementovanou v jednotlivých systémech. Sebelepší obecný algoritmus může selhávat při nasazení na databázovém systému, který bude používat nějaký speciální (také "optimalizovaný") přístupový systém. V tomto případě se jeví jako jediná spolehlivá varianta vlastní vyrovnávací paměť, kterou může databázový systém řídit se znalostí věci. Z testů totiž vyplývá, že nejcennější pro správce vyrovnávací paměti je (pro mnohé možná paradoxně) informace "co lze zapomenout". Teprve daleko v závěsu za ní se objevuje výhoda predikce (tzn. "co budu potřebovat příště") a další případné znalosti o použití dat. Tuto informaci však správci nemůže poskytnout nikdo jiný než databázový stroj.

Samozřejmě kombinace vlastní vyrovnávací paměti společně s vyrovnávací pamětí systému může přinášet nejlepší výsledky. Tento předpoklad je však také velmi silně závislý na použité platformě a kapacitě operační paměti, která je k dispozici. Často se ukazuje, že je-li k dispozici dostatek vnitřní vyrovnávací paměti (např. pro MS-DOS 3MB a více), je výhodnější vypnout vyrovnávací paměť operačního systému (resp. ji zmenšit na úkor volné paměti pro DS).

10. Závěr

Pokusil jsem se nastínit některé rysy a metody návrhu OO databázového stroje, tak jak je postupně vyvíjen ve firmě VEMA. Přestože ve chvíli vzniku tohoto příspěvku je návrh již poměrně ucelený a probíhají první práce na implementaci, některé problémy zůstávají stále otevřené a některé zmíněné metody mohou být zcela nahrazeny jinými. Rozsah příspěvku mi bohužel nedovoluje popsat všechny vlastnosti a přístupy, které jsou v DS VEMA použity. Většina z nich je však založena na již známých postupech a algoritmech, jejichž modifikace, rozšíření a vzájemná kombinace se jeví jako ideální cesta pro další vývoj. V tak důkladně teoreticky zmapované oblasti, jakou je svět databázových aplikací, zřejmě nelze v nejbližší době očekávat zásadní změny technologií.

Literatura

1. Date, C.J. "An Introduction to Database Systems, 6th ed.", Reading, Mass.: Addison-Wesley (1994)
2. "The Object database standard, ODMG-93 / edited by R. G. G. Cattell, Release 1.1", San Francisco, CA" Morgan Kaufmann Publishers, Inc. (1994)
3. Lorie, Raymond A. "Physical Integrity in a Large Segmented Database." ACM TODS 2, No.1 (March 1977)

Ing. Karel Obluk
VEMA
Výstavní 17/19
603 00 Brno
e-mail: obluk@vema.cz
tel: +5 - 4321 5524
fax: +5 - 4321 1946