Jak to přišlo, že na podzim roku 1992 Československo nebylo zaplaveno produktem zabezpečení transakčního zpracování pro FoxPro 2.0

Petr Bílý, Jitka Majetičová

Motto: *Věc byla připravena s obezřetností přímo úžasnou.*
Jan Neruda, Povídky malostranské.

1 Úvod

Toto krátké vyprávění shrnuje naši drobnou práci, tápání i výboje na poli ošetření transakcí z hlediska bezpečnosti zpracování dat. Uzavíráme tak několikaleté období, které, byť zdaleka ne kontinuálně a systematicky, bylo poznamenáno permanentním pocitem, že s bezpečností dat při jejich zpracovávání databázovými systémy na počítačích třídy PC to zdaleka není všechno v pohodě. Jistě k tomu přispěla i určitá zhýčkanost, kterou jsme si přinesli z práce na sálových počítačích, kde se operační systém a IDMS téměř o vše postaraly za uživatele i programátora. V prostředí PC jsme náhle cítili, že programátoři, tvůrci software a konec konců i samotní uživatelé to s péčí o zpracovávaná data nemyslí příliš vážně. Tento stav nám ovšem krajně nevyhovoval, neboť jsme velmi záhy začali řešit úlohy náročné na objem dat, dobu jednoho „sezení" (session) u počítače i míru tolerantnosti k laickosti obsluhy. Než ovšem prozradíme „jak to přišlo...", zopakujeme a doplníme několik poznatků.

2 Opakování o transakcích

Nejprve se snadno shodneme na tom, že každý ze zainteresovaných chce mít v paměti počítače uložena ta „ správná data". Předpokládejme, že naše data máme uložena na počítači v **databázi**, což je **datová základna** modelu určitého informačního systému. „Správnost" dat uložených v databázi lze nyní hodnotit z několika hledisek. Především je to vztah ke skutečnostem v reálném informačním systému, který pomocí výpočetní techniky popisujeme, simulujeme nebo řídíme. Z toho vyplývají požadavky na **věrohodnost** a **aktuálnost** uložených dat, které sice není zdaleka jednoduché dodržet a uhlídat, ale zatím je ponecháme stranou našeho zájmu.

Zde se více zaměříme na otázky **konzistence** databáze. Zjednodušeně řečeno, databáze obsahuje nekonzistentní data tehdy, jestliže uživatel může dojít ke zjištění (vždy nebo jen při některém způsobu přístupu k datům), že data, jejichž existenci oprávněně předpokládá,

chybějí, mají nesprávné hodnoty, jsou špatně uspořádaná, případně se vyskytují nežádoucí. Příčinami vzniku nekonzistencí jsou chyby technického nebo programového vybavení, případně chyby obsluhy. Nekonzistence databáze se nejčastěji projeví ve vzájemném vztahu více databázových souborů, může však vzniknout i v úloze pracující s jedním souborem z jediného místa. S tím, jak přistupuje do hry zpracování více souborů, zpracování v síti (tj. distribuované zpracování) či dokonce zpracování distribuovaných dat, zvyšuje se obtížnost programátorského ošetření vzniku nekonzistencí resp. nápravy stavu databáze.

Jednoduchým příkladem vzniku nekonzistence je tato situace: přijmeme peníze do pokladny, ale opomeneme o tom učinit zápis do pokladního deníku. Výsledkem je nesoulad informací, které lze vytěžit z pokladny a z deníku. Je zřejmé, že vznikl nedokončením akce skládající se ze dvou na sebe nutně vázaných dílčích kroků.

Takovouto akci označujeme jako **transakce**. Omezíme-li se na oblast práce s daty na počítači, pak můžeme transakci definovat jako logicky uzavřenou posloupnost zásahů do databáze. Základním požadavkem transakčního zpracování je, abychom při jakémkoli využití dat z databáze získali data platná před zahájením provedení transakce nebo po jejím úplném dokončení. V žádném případě není přípustné získat data z nějakého stavu transakce rozpracované. Tento rys transakce se nazývá **atomičností**, to znamená, že celá posloupnost operací patřících k jedné transakci může být provedena jen jako celek. Z toho vyplývá důležitost pojmů **začátek transakce** a **konec transakce** jakožto definovaných časových okamžiků zpracování. Ne vždy je již započatá transakce úplně dokončena, ať již programově řízeným **zrušením transakce** (abort) nebo technickým přerušením provozu počítače. V takovýchto případech je nutné provést **vrácení transakce** (rollback) : všechny prováděné nebo již provedené změny je třeba vrátit do stavu před zahájením transakce. Aby takové vrácení bylo možné, je třeba zajistit uchování **stavu dat před transakcí** (before image), které jsou využity při vracení transakce. Pokud se uchovává i **stav dat po transakci** (after image), je možno provést rekonstrukci databáze do zvoleného časového okamžiku (rollforward) – např. při ztrátě datového souboru. Stavy dat před (po) transakci se uchovávají zpravidla ve speciálním souboru nazývaném **žurnál**.

Pokud přibereme do úvahy víceuživatelské zpracování, musíme zmínit i vlastnost **izolované vratnosti**. To jest požadavek, aby vrácením jedné z probíhajících transakcí nebyla postižena žádná jiná transakce – pokud by to nebylo splnitelné, musí být také vrácena.

3 Ošetření transakcí databázovými systémy na počítačích PC

V nekompletním zhodnocení se omezíme na třídu těch databázových systémů, které jsou určeny především mikropočítačům (nikoli hanlivě jim někdy říkáme „kancelářské databáze") s operačním systémem DOS. Ekonomické důvody a míra rozšíření zde zatím váží nejvíce. Mnoho dalších databázových systémů provozovatelných na PC se proble-

matice ošetření transakcí věnují důkladněji, ale jsou to systémy, které svým charakterem patří spíše na střední třídu počítačů (minipočítače) a PC využívají převážně jako terminály.

Nejrozšířenějšími DB systémy jsou ty, které používají jazyka typu xBASE. Z nich **DBASEIV** disponuje sice základními funkcemi a příkazy k ošetření transakcí, ale v době, kdy jsme je prověřovali, jsme narazili na některé problémy i chyby.

Největší zájem, daný mírou rozšíření u nás, je zaměřen na výrobky firmy FoxSoftware (nyní Microsoft). Jinak velmi úspěšná FoxBASE 2.10 potřebné mechanismy zcela postrádala, totéž platí i pro FoxPro 1.02. Populární **FoxPro 2.0** sama rovněž nic neřeší, ale při provozu pod síťovým operačním systémem Novell NetWare lze použít knihovnu procedur s funkcemi, které pro zabezpečení používají Transaction Tracking System (TTS). I zde stále ještě mnoho „manuální" práce na dotažení dokonalého zabezpečení zůstává na programátorovi.

Databázový systém **Advanced Revelation** je koncepčně velmi moderně (snad v této třídě dokonce bezkonkurenčně nejlépe) navržen, včetně systému práce s transakcemi. Nepříliš velký počet aplikací u nás neumožňuje učinit jednoznačné závěry.

Ambiciózní (zejména cenově) systém **PARADOX** verze 4.0 udává existenci zabudovaných mechanismů pro zvýšení bezpečnosti datových souborů, potřebné funkce pro zajištění transakčního zpracování mu však chybí.

Transakční zpracování podporuje i v mnoha směrech pozoruhodný **db_VISTA**, produkt firmy RAIMA.

Je jisté, že existují další databázové systémy této třídy, které se s problematikou vyrovnávají, nebo se o to alespoň snaží (za mnohé například SEZAM z domácí produkce). Vzhledem k jejich malému rozšíření či naší neinformovanosti se s nimi nezabýváme.

Pro úplnost je třeba se zmínit i o možnosti zabezpečit transakční zpracování aparátem nezávislým na použitém databázovém systému. Víme o řešení, které rovněž ale vyžaduje provoz pod Novell Netware 3.11.

4 Pravděpobné nároky běžného uživatele

vyplývají v první řadě z možností jeho technického a v druhé řadě i programového vybavení. Stále ještě mnoho uživatelů nepřechází na nákladné sítě typu Novell. Nákladnost tkví nejen v pořizovací ceně, ale i v náročnosti na nastavení a udržování síťového systému. Stále ještě mnoho systémů je jednouživatelských, což ovšem neznamená, že hledisko bezpečnosti dat zde není aktuální. Velká skupina uživatelů disponujících skupinou řádově jednotek počítačů si uvědomuje životní nutnost jejich propojení,

použijí však pro to s největší pravděpodobností jednodušší, ale ještě efektivní sítě (např. LANTASTIC).

Tato skupina uživatelů se v drtivé většině orientuje na již zmíněné „kancelářské" databáze, u nichž by uvítala možnosti zabezpečení jak více– tak i jednouživatelských aplikací.

5 Řešení pro FoxBASE 2.10

není již v tomto okamžiku aktuální, uvádíme je pouze pro ilustraci výše uvedených principů transakčního zpracování a posouzení historicky vývojového stupně.

Pro dané období (rok 1989) bylo charakteristické, že současné systémy prakticky neumožňovaly použít pro vytváření procedur jiných nástrojů, než vlastní jazyk. Přes toto závažné omezení jsme vytvořili jednoduchý systém, který umožňuje vrátit stav databáze na počátek transakce, nebo na počátek celé „session".

Filosofie návrhu je v těchto hlavních myšlenkách:

- registrovat každý zásah typu Append, Update, Delete na databázovou větu
- ovládat celý systém řídícím souborem zabezpečení, jedinečným pro celou aplikaci (DIARY)
- registrovat stavy vět jednotlivých databázových souborů před zásahem do databáze (soubor JOURNAL)

Realizace zabezpečení spočívá ve volání obslužných procedur z programu. Definování rozsahu transakce je zajištěno voláním procedury ENDTRAN. Začátek transakce je implicitně dán začátkem programu nebo koncem předchozí transakce. Zápis do žurnálních souborů zajišťuje procedura JOURNAL, kterou je nutno vyvolat při každém zásahu do databáze. Návrat ke stavu před zahájením transakce obstará vyvolání procedury ABOTRAN. Procedura ROLLBACK provedená na začátku session uskuteční vrácení, pokud předchozí zpracování nebylo regulérně ukončeno.

Řešení provází některá **omezení** (na př. jsou zablokovány 2 pracovní oblasti) přesto byl systém úspěšně využit v několika denně provozovaných aplikacích.

6 Možnosti FoxPro 2.0

Již povrchní seznámení se s první verzí FoxPro 2.0 nám přineslo zklamání. Produkt opět ignoroval ošetření transakcí. Pro aplikace pracující v síťovém prostředí je tato starost přenechána službám sítě, což nevyhovuje v případech mnoha sítí, které služby tohoto charakteru nenabízejí. Navíc nastává často nežádoucí vázanost aplikace na konkrétní síť. Aplikace zpracovávaná na samostatném PC nemá šanci transakce ošetřit.

Zpřístupnění balíku „Library Construction Kit" otevřelo cestu k vytváření procedur v jazyce C spolupracujících s FoxPro. Zlepšila se situace pro uživatele sítě Novell NetWare, díky již zmíněné knihovně procedur (NETLIB.PLB), i když ani toto řešení není zcela vyhovující. Pro ostatní je tu šance na vytvoření vlastního systému.

7 Vymezení požadavků na systém podpory transakčního zpracování ve FoxPro 2.0

Základní vlastnosti, které musí být implementovány, jsou dány charakteristikami transakcí, jak jsou uvedeny v kap. 2. Podrobněji se budeme zabývat požadavky a omezeními, které vyplývají z charakteru FoxPro a reálných možností použitelných nástrojů.

Systém je pochopitelně navrhován pro **síťový provoz**, ale měl by pomoci i při izolovaném zpracování na jednom PC. Rozhodli jsme se pro dvě samostané verze – důvodem je výrazně nižší spotřeba zdrojů pro verzi jednouživatelskou. Začali jsme verzí síťovou s perspektivou na následnou zjednodušující transformaci.

Za stěžejní pokládáme požadavek provozu **na libovolném typu sítě**, tedy vedle sítí typu server–stanice i sítě typu peer–to–peer. Na to navazuje požadavek umožnit zabezpečení dat nejen v síti sdílených, ale i lokálních (přesněji aktuálně nedeklarovaných jako sdílená) na libovolném uzlu sítě.

Běžnou filosofií vracení transakcí na sebe vázaných (v čase se překrývajících) je **důsledné zamykání** všech změněných vět po celou dobu trvání transakce. Toto je vyhovující pro aplikace, které zpracovávají převážně „krátké" transakce (do několika sekund). Respektování **„dlouhých" transakcí** (minuty – např. použití BROWSE) vyžaduje jiné řešení, které si vynucuje požadavek zablokování výstupu dat do okamžiku ukončení skupiny vázaných transakcí.

Zachování dokumentace změn databáze z celého průběhu zpracování umožní vrácení nejen jedné transakce nebo skupiny vázaných transakcí, ale volitelně celé „session" nebo její specifikované části.

Transakce nemusí pokrýt celé zpracování na stanici, na druhé straně se v rámci transakce zabezpečují všechny v ní užité databázové soubory. Každá zpracovávaná úloha má své vlastní zabezpečení (a tedy i vracení) nezávislé na provozu a stavu ostatních úloh.

Konečně pochopitelným požadavkem je **minimalizace zásahů do zdroje programu**, který systém zabezpečení transakcí použije.

8 Poznámky ke konkrétnímu řešení

Použité nástroje: Balík „Library Construction Kit", který podporuje tvorbu rozhraní mezi FoxPro a jazykem C – tzv. Application Program Interface (dále API). Umožňuje nejen vyvolávat procedury v jazyce C z FoxPro, navíc je většina příkazů a funkcí FoxPro přístupná v prostředí jazyka C. Kromě toho lze ovlivňovat mechanismus, který zpracovává události ve FoxPro (event handling), což je podstatné pro řízení průběhu zpracování programu.

Základní problémy, které bylo nutno vyřešit, jsou tyto:

- indikace změn v databázi
- záznam změn v databázi
- vrácení databáze do konzistentního stavu

Pro **zjištění změny v databázi** je nutno nejprve vymezit, ve kterých situacích může dojít ke změně. Jde o efekt více než 20 příkazů a funkcí FoxPro, mnohé z nich v četných variantách dle použitých frází. Nejpřirozenější způsob zjištění změny – modifikace funkce těchto příkazů – není možný, protože nemáme přístup k jejich zdrojovému kódu. Pomůžeme si ale nástrojem API, kterým vytvoříme proceduru, reagující na události FoxPro, v konkrétním případě na změnu ukazatele na aktuální větu v databázovém souboru. Dočasně ukládáme obsah věty před zpracováním a po jejím opuštění zjistíme, zda došlo ke změně. Protože tato procedura pracuje „v pozadí" provádění příkazů FoxPro, lze mnoho příkazů (např. na ošetření velice citlivý BROWSE) ponechat zcela bez úprav. Tam, kde tento postup nelze uplatnit, použijeme doplňkové příkazy, které je třeba vložit před a za příkazy FoxPro (např. XAPPEND APPEND.... YAPPEND)

Záznam změn provádí většina transakčních systémů kopírováním relevantních bloků paměti. V našem řešení, které je velmi úsporné z hlediska spotřebované paměti, zaznamenáváme pouze ty položky databázové věty, které doznaly změny.

K zaznamenávání průběhu transakcí jsou použity 2 typy souborů :– **řídící soubor** (LOG), ve kterém jsou záznamy o začátcích a úspěšných ukončeních transakcí na všech zúčastněných stanicích ;– **žurnální soubor** s informacemi o původním stavu věty, ve které došlo k jakékoli změně dat, která byla zrušena nebo přidána do souboru.

Tyto soubory jsou sdíleny všemi úlohami pracujícími nad jednou aplikací. Každá úloha po svém spuštění tyto soubory otevře (nejsou–li již otevřeny jinou úlohou) a při svém ukončení je uzavře (nepoužívá–li je ještě úloha jiná).

Soubor LOG sestává z posloupnosti záznamů jediného typu, kde je uloženo jméno stanice (v síti jednoznačné), zda jde o začátek nebo konec transakce a čas. Tuto dokumentaci prováděných transakcí uchováváme pro celý průběh jedné session.

Žurnální data jsou fyzicky rozdělena do dvou souborů: JNL a JNL_MEM (pro záznam změněných memo položek). Každý záznam žurnálního souboru nese informaci o stavu dat ve větě databázového souboru před její změnou. Na začátku každého záznamu v žurnálním souboru jsou údaje o tom, která stanice záznam ukládá, o který databázový soubor se jedná a o jaký typ zásahu jde. Pak následují v závislosti na typu zásahu další potřebné informace, typicky číslo věty a seznam dvojic číslo položky, obsah položky. V případě změny memo položky bude odkaz na umístění původního obsahu v souboru JNL_MEM.

Záznamy v žurnálním souboru mají proměnnou délku, zpětné zřetězení záznamu umožňuje uložení délky jako poslední hodnoty v záznamu. To má význam hlavně při rollbacku, kdy je třeba zpracovávat informace od konce souboru JNL směrem k začátku.

Vrácení stavu databáze může nastat buď vrácením jedné nebo skupiny vázaných transakcí. Dojde k tomu řízeně z programu nebo při restartu úlohy. Při jakémkoli startu úlohy zjistí stanice ze souboru LOG, zda zpracování poslední transakce bylo úspěšné nebo zda je třeba provést nejprve vrácení. Proces vracení pak vyžaduje spolupráci stanic, které byly účastny v poslední nedokončené skupině transakcí. Vracení probíhá zpětnou rekonstrukcí zásahů podle žurnálních dat.

Programování úlohy pro transakční zpracování samozřejmě vyžaduje vložení příkazů pro vymezení začátku a konce transakce, případně programově řízeného vrácení. Výše zmíněné úpravy některých příkazů ovlivňujících stav databáze jsou řešeny pomocí „předkompilátoru". Ten přetransformuje původní zdrojový tvar programu do nového zdrojového začleněním modifikovaných příkazů.

Omezení při použití tohoto transakčního systému nejsou příliš významná. Je třeba mít otevřen 1 databázový soubor, ve kterém jsou uloženy informace o strukturách používaných souborů (obsadí oblast 25). Soubory LOG, JNL a JNL_MEM, mají vyhražené jméno, což však nepřináší omezení, neboť jsou uloženy ve speciálním adresáři. V případě, že uvnitř transakce použijeme příkaz ZAP, dojde k uvolnění paměťového prostoru až po ukončení session.

9 Příčiny neúspěchu realizace

Každý, kdo absolvoval odladění alespoň jednoho programu, jistě shovívavě přijal nadsázku v mottu příspěvku. Přesto se musíme přiznat, že se nám dosti dlouho zdálo, že návrh řešení není zatížen žádnou hrubou chybou, a že (odhlédneme-li od pracnosti) jsme poměrně blízko cíle. Testy na jednoduchém příkladě probíhaly úspěšně.

Systém však v reálném prostředí nevyhověl. Hlavní příčinou je problém otevření souboru s přiřazeným číslem „handle" (identifikační číslo paměťového prostoru) větším

než 20. Funkcemi FoxPro lze jiné než databázové soubory otevírat pouze v nesdíleném módu. Pro otevírání sdílených souborů (žurnál a LOG) jsme proto použili funkce v jazyce C. Ale zde lze sdílenč otevřít soubor jen funkcí, která pro komunikaci se souborem používá číslo handle. Počet souborů, se kterými může DOS zacházet tímto způsobem, je omezen na 20. Zjistili jsme, že již po otevření programu ve FoxPro činí počet obsazených handle minimálně 15. Náš systém potřebuje 3 soubory otevírané sdílenč, čímž se dostáváme k hodnotě 18. Při otevření jiných než databázových souborů a překročení počtu 20 dochází ke kolizi informace o souboru pod tímto číslem vedeným programem FoxPro a operačním systémem DOS.

Autoři: ing Jitka Majetičová, ing Petr Bílý
PIKSOFT sdružení
Jablonecká 52
190 00 Praha 9