Předpoklady využití prostředí klient/server při tvorbě objektově orientovaných aplikací

Jiří Schindler, Jan Ministr

EkF VŠB-TU Ostrava, Sokolská 33, 701 21 Ostrava, Česká Republika

Abstract

The choice of information system architecture has fundamental influence on the level of the software applications, implementation ability DB systems and user access to data. Three types of architecture came into existence during historical development of information systems (IS) Host/Terminal (H/T), File/Server (F/S) and Client/Server. Information system with architecture C/S represent the most perspective information technology from viewpoint of speed, cardinality and „comfort" of satisfy user information needs. Object-oriented development (OO) of applications in the C/S environment arose on the transformation base of structured to object-oriented methods and it can be anticipate, that it will be just the C/S architecture, which will enforced through the medium of its characteristics the object-oriented approach into system analysis and design of IS. C/S information technology continues in previous development stage as all range other technologies and now it represent qualitative top of construction IS, which is established on the base of dynamic modelling, which come out from event-oriented approach, event driven programming, message polymorphism, transaction processing, SQL communications and user GUI.

1. CHARAKTERISTIKA PROSTŘEDÍ C/S

Hlavní odlišnosti mezi metodami strukturovanými a objektově orientovanými spočívají v tom, že ve strukturovaných metodách se připouští oddělování procesů zpracování dat od dat samotných a obojí se koncipuje v podobě statických modelů funkčních a datových. Tím se rozdělují prvky systému na dva typy. Na funkce (které jsou aktivním nositelem děje) a data (která jsou v systému pasivována). To vede k několika důsledkům:

- prvním z nich je sepjetí funkcí IS s výpočetním systémem. Tento vztah aktivuje výpočetní systém v těch časových okamžicích, kdy to systém dovoluje, nikoliv kdy vzniká potřeba zpracování informací,
- druhým důsledkem je modelování datových struktur na bázi relačního modelu, v němž entity nemají vztah ke svým vlastním operacím. Z toho vyplývá, že sémantika programového vybavení neumožňuje pružně měnit vztahy mezi entitami,
- třetím důsledkem je to, že programový design je vytvářen na bázi takových událostí, které dostatečně neřeší dynamičnost informačního systému v reálném čase.

Vyjmenované nedůslednosti strukturovaných metod vyplňují metody objektově orientované, jejichž podstata spočívá v pojetí dynamických objektů, které na sebe v systému přebírají nositelství děje. Průběh programové aplikace je na základě jejich dějů řízen výskytem příslušných událostí a následnou aktivací počítačového zpracování.

Pro vymezení pojmu *architektura*, užívaného v souvislosti s vývojem C/S aplikací, stačí architekturu chápat jako technickou a softwarovou podporu provozovaného informačního systému vyvinutou objektově orientovanými postupy a s takovou technickou a softwarovou výbavou, jejíž koncepce je založena na myšlence, že data by se měla zpracovávat tam, kde jsou uložena. Centrální část výpočetního systému - server vlastní data a slouží celému systému, uživatelská část, která využívá služeb serveru a je nositelem uživatelského rozhraní, se nazývá klient. Architektura informačního systému založena na nezávislých softwarových procesech klienta a serveru, z nichž první odpovídá za zpracování zadaného úkolu a druhý mu poskytuje specifické služby, se nazývá architekturou klient/server.

Proces klienta je interaktivní front-end proces, jehož úkolem je řešit problémy rozhraní mezi uživatelem a výpočetním systémem tím, že klient předává uživatelské požadavky serveru a obdržené výsledky vhodně presentuje. Proces serveru je dávkový back-end proces, který klientovi poskytuje nezbytně nutná data a stará se o integritu a konzistenci databáze. Výměna dat mezi klientem a serverem se děje prostřednictvím přeložených SQL výrazů vyhovujících požadavkům na úspornost a rychlost přenosů. Zatímco server spravuje SQL tabulky a poskytuje datové služby různým aplikacím, je klient soustředěn na požadavky uživatele a vyhovuje jim prostřednictvím oken, která dnes ztotožňujeme především s grafickým uživatelským rozhraním GUI.

GUI usnadňuje komunikaci mezi aplikací a uživatelem pomocí grafiky a textu. Pokud je použito rozumným způsobem, pak

- snižuje množství informace, které si uživatel musí při své práci pamatovat,
- urychluje komunikaci mezi aplikací a uživatelem, protože uživatel má možnost visuálně porovnat množství, tendence a vztahy sledovaných veličin,
- presentuje informaci v konsistentní formě,
- grafika omezuje chybovost, snižuje nároky na školení a zacvičování obsluhy, přičemž vede k vyšší úrovni řešeného problému a podporuje zapamatovatelnost obsluhování aplikace.

Nejběžnějším grafickým uživatelským rozhraním (v současnosti standardizovaným), ze kterého se vychází při tvorbě aplikací, je Microsoft Windows. Hlavní příčinou této obliby je konsistence mezi Windows aplikacemi, což napomáhá uživateli naučit se používat tyto aplikace rychleji a snadněji než aplikace znakově založené. Výhodou GUI je možnost individuální logiky postupu uživatele při práci s aplikací, která však klade vyšší nároky na přístup k paměti, aby byla zajištěna přiměřená škála odezev. Na druhé straně je třeba podotknout, že aplikace nevyužívající GUI a SQL databázové technologie nemohou efektivně využívat možnosti stále dostupnějších systémových zdrojů.

Návrh GUI je složitější než návrh znakově orientovaného rozhraní, protože je ve své podstatě rozsáhlejší a má k dispozici daleko širší možnosti presentace dat. Nabídka barev, tlačítek, textových oken a podobně někdy svádí k přílišnému používání zbytečných efektů v aplikacích s GUI, které uživatele rozptylují a zdržují. Proto je nutné využívat platné standardy GUI, nebo je v dané organizaci zavést.

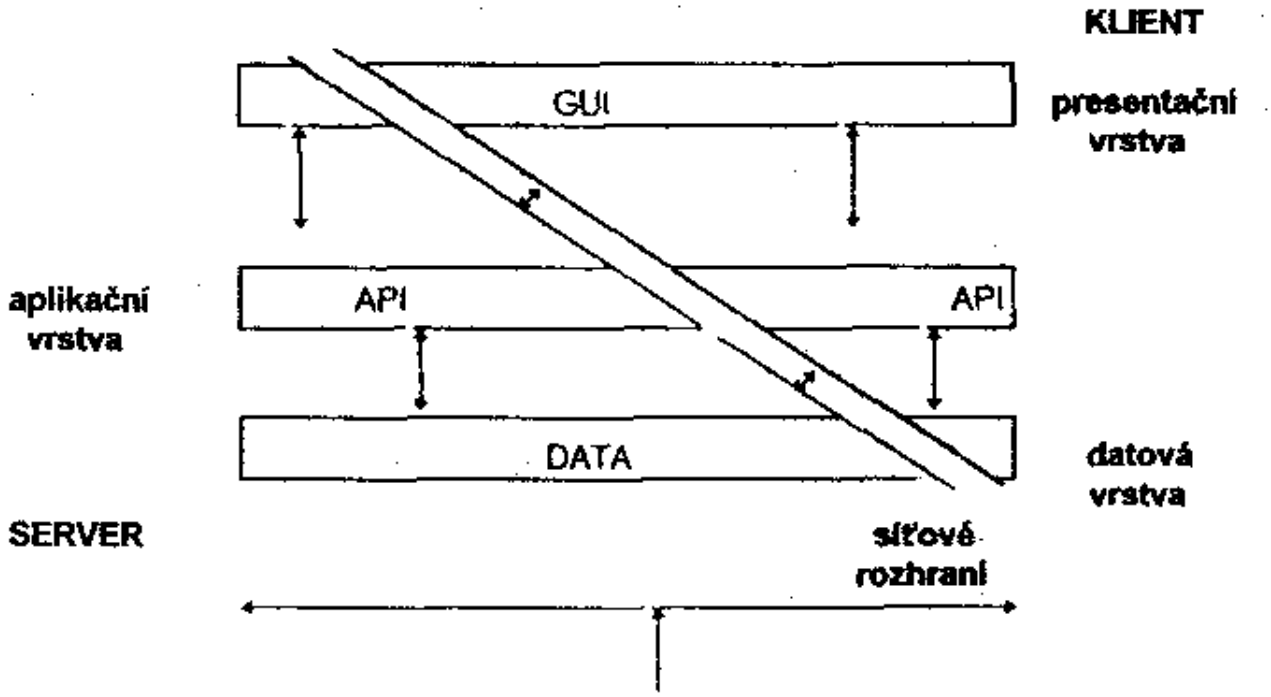
Aby informační systém s architekturou C/S mohl být navržen, implementován a spolehlivě provozován, museji všechny jeho komponenty splňovat podmínky, které

tato technologie na architekturu C/S klade. Mezi základní komponenty vytvářející informační systém, včetně jejich vlastností, patří:

1. Uživatel s ujasněnou informační potřebou, schopností pracovat s výpočetní technikou a ovládající svou aplikaci.
2. Technické prostředky výpočetního systému, k nimž zejména náleží server (servery), klientské stanice a síťové komunikační vybavení.
3. Softwarové vybavení tvořící síťové prostředí, operační systém, aplikační programové vybavení a systém řízení báze dat.
4. Databáze.
5. Metody používané při navrhování, implementaci a provozování IS.

Pokud je některá z vyjmenovaných komponent v rozporu s požadavky architektury C/S a působí heterogenně, je stabilita a efektivnost informačního systému ohrožena. Zatímco technické prostředky, softwarové vybavení a používaný databázový systém patří k technologické části informačního systému, náleží úroveň uživatele a používané metody k jeho „mentální" části.

Procesy probíhající při výměně dat v architektuře C/S se odehrávají ve třech vrstvách. V datové, aplikační a presentační. Datová vrstva je vlastní procesům na straně serveru, presentační na straně klienta. Softwarové jádro mezi těmito vrstvami tvoří aplikační vrstva s procedurálním návrhem na výměnu dat prostřednictvím výrazů SQL (viz obr. 1).



Obr. 1: Schéma vyjadřující míru distribuce dat a procesů mezi klientem a serverem.

Součástí technologického návrhu architektury C/S je její dvojúrovňová, tříúrovňová nebo obecně n-úrovňová koncepce. V dvojúrovňové architektuře se jedná o míru distribuce programů a dat mezi klientem a serverem vyjádřenou polohou ukazovátka na obr. č. 1. Vlastností této architektury je nízká cena technických zařízení, rychlý vývoj aplikace a použití uživatelsky přívětivého GUI. V tříúrovňové architektuře je mezi klienta a datový server zařazen navíc server aplikační. Výhodou tříúrovňové architektury proti dvojúrovňové je otevřenost systému vůči komponentám od různých

výrobců, ale její nevýhodou je, že vývoj v tříúrovňové architektuře je oproti dvojúrovňové pomalejší a dražší.

Informační toky v obr. 1 ukazují na integritu komponent technologické části architektury C/S, v níž se jakákoliv generační technická heterogennost systému projevuje zpomalováním procesů zpracování a může vést až k nespolehlivosti informačního systému. Aplikační interface (API) v současnosti uživateli zpřístupňuje informace od více aplikací najednou a výměnu dat (DDE) i výměnu objektů (OLE) mezi aplikacemi.

2. CYKLUS VÝVOJE OBJEKTOVĚ ORIENTOVANÝCH APLIKACÍ

Přechod od strukturovaného k objektově orientovanému vývoji aplikací v architektuře C/S znamená uvést do vzájemného souladu přístup uživatele k údajům o objektech rozmístěných v počítačové síti, jejich presentaci na monitorech a operace prováděné s objekty. Jinými slovy ve stadiu analýzy jde o specifikaci uživatelských požadavků a ve stadiu designu o návrh aplikace, která těmto požadavkům odpovídá. Objektově orientovaný přístup v oblasti systémové analýzy rozšiřuje potřebu registrace uživatelských požadavků, diagramů datových toků a modelů entit (objektů), vyplývající ze strukturovaného přístupu, o podrobný popis realizace informačních potřeb a pracovních postupů (kontrolních scénářů) koncového uživatele včetně tvorby modelu uživatelských objektů.

Design technologické části informačního systému je závislý jak na předpokládané úrovni počítačového zpracování, tak na velikosti systému a na specifikách uživatelů (na mentální části). Mezi známé typové postupy charakteristické pro životní cyklus vývoje systému patří:

- přírůstkový model pro velké projekty,
- spirálový model pro OO vývoj menších aplikací,
- dodávka SW balíku.

Pro libovolně zvolený vývojový cyklus platí, že musí být vyjasněné jeho aplikační oblasti, které rozhodují o architektuře informačního systému. Nejužívanějšími způsoby identifikace těchto oblastí jsou

- strukturalizace subsystémů založená na analýze organizačních celků. Použití této metody je vhodné při navrhování nového IS při založeni organizace nebo po takové restrukturalizaci organizace, která vyžaduje novou koncepci informačního systému (na př. při přechodu z centralizovaného na decentralizované řízení nebo při nahrazování architektury H/T architekturou C/S),
- datová analýza stávajícího modelu databáze, který v důsledku vývoje organizace zastarává, je překonán a vzniká potřeba jeho reengineeringu na podmínky nových předmětných oblastí,
- analýza časových událostí, která je často spojována s uživatelskými informačními potřebami a objekty. Tato metoda je použitelná buď při vývoji menší aplikace, v níž jsou události vznikající na rozhraní IS s okolím jednoznačně identifikovatelné, nebo představuje doplněk předcházejících metod, v nichž jsou aplikační oblasti definované a jde o provádění analýzy časových událostí v každé z nich.

Závěr vyplývající z předcházející identifikace je nejbliže objektově orientovanému přístupu. V podstatě jde o analýzu prováděnou v prostředí rozděleném do

aplikačních oblastí, v nichž jsou identifikovány objekty a jejichž údaje jsou předmětem počítačového zpracování. Analýzou událostí, jež tyto objekty na rozhraní s informačním systémem vyvolávají, vzniká obrázek o jejich dynamických vlastnostech a tím pádem i o způsobech jejich zpracování. Předmětem OO systémové analýzy dynamiky objektů jsou přechodové procesy mezi jednotlivými stavy objektů, které se v OO terminologii nazývají „operacemi". Jak bylo výše zmíněno, při aplikaci OO přístupu není nositelem děje v IS výpočetní systém, ale objekt, který svou interakci s výpočetním systémem iniciuje událost a uvádí do vzájemného vztahu některou ze svých operací s operacemi (funkcemi) výpočetního systému.

3. ANALÝZA ČASOVÝCH UDÁLOSTÍ JAKO VÝCHODISKO NÁVRHU TRANSAKČNÍHO ZPRACOVÁNÍ V ARCHITEKTUŘE C/S

Při interakci objektů vnějšího světa s výpočetním systémem je možné na nejhrubší rozlišovací úrovni tyto objekty rozdělit do čtyř tříd.

1. Objekty, jejichž vlastnosti jsou předmětem počítačového zpracování (objekty evidenčního charakteru).
2. Objekty, jejichž stav se výpočetním systémem sleduje.
3. Objekty generované samotným dynamickým procesem (vyplývající z povahy dynamického procesu).
4. Objekty, které počítačové zpracování ovlivňují (řídí).

Každý z těchto objektů může být zdrojem specifických událostí. V uvedeném pořadí jsou to události [3]

Flow	(F) : tokové	(signál přinesl informaci, kterou je nutné zpracovat)
State	(S) : stavové	(nastal stav, který je nutné ošetřit)
Temporal	(T) : časové	(nastal okamžik spuštění dávky)
Control	(C) : řídicí	(zásah operátora nebo správce systému)

Řídicí události souvisí s bezpečností informačního systému, ochranou dat a správou databáze. Neiniciuje je běžný uživatel, ale administrátor, který má k tomu zvlášť vyhrazená práva. Časové události zpravidla souvisí s projektovaným dávkovým zpracováním a jsou odvozeny od proměnné času. Vznik těchto událostí je podmíněn tím, že nastala konjunkce časového okamžiku s předpokládaným spuštěním dávkového procesu. Stavové události vznikají při výskytu některého sledovaného stavu objektu reálného světa, na nějž výpočetní systém příslušným způsobem reaguje.

Nejčastějšími událostmi jsou tokové události. Vznikají náhodně, když se vyskytuje objekt (signál) s novou informací, kterou je nutné zpracovávat, nebo když se vyskytne informační potřeba, kterou je třeba uspokojit na základě evidovaných údajů o objektech. Takové události se týkají objektů evidenčního charakteru, které v interakci s výpočetním systémem (v dialogu) iniciují následující množinu operací: *Insert* (I), *Modify* (M), *Delete* (D) a *Read* (R).

Výpočetní systém samotný ve vzájemném vztahu s objekty evidence představuje objekt, který má vedle svých skalárních atributů množství atributů dynamických (vlastních operací), jež jej vybavují nemalým počtem způsobů interakcí s objekty evidenčního charakteru při realizaci tokových událostí.

Povaha objektů zúčastněných na dynamice informačního systému je dvojí. Komponenty výpočetního systému představují objekty, jež jsou v systému trvale přítomné a poskytují servis (službu, obsluhu, zpracování) objektům, které tento servis využívají. První z nich jsou trvale přítomné prvky v systému (stabilní objekty, které tvoří technologickou část IS), druhé jsou nestabilními objekty (signály, jakožto nositelé informací), které jsou stabilními objekty zpracovávány.

Každý z objektů tvořících součást IS se vyznačuje vlastní dynamikou, která mu je předurčena jak funkcí celého systému, tak jeho vlastními operacemi. Vzhledem ke konečnosti stavů, jichž může každý objekt v daném čase a v daném místě IS nabývat, je možné jeho dynamiku modelovat stavovým diagramem.

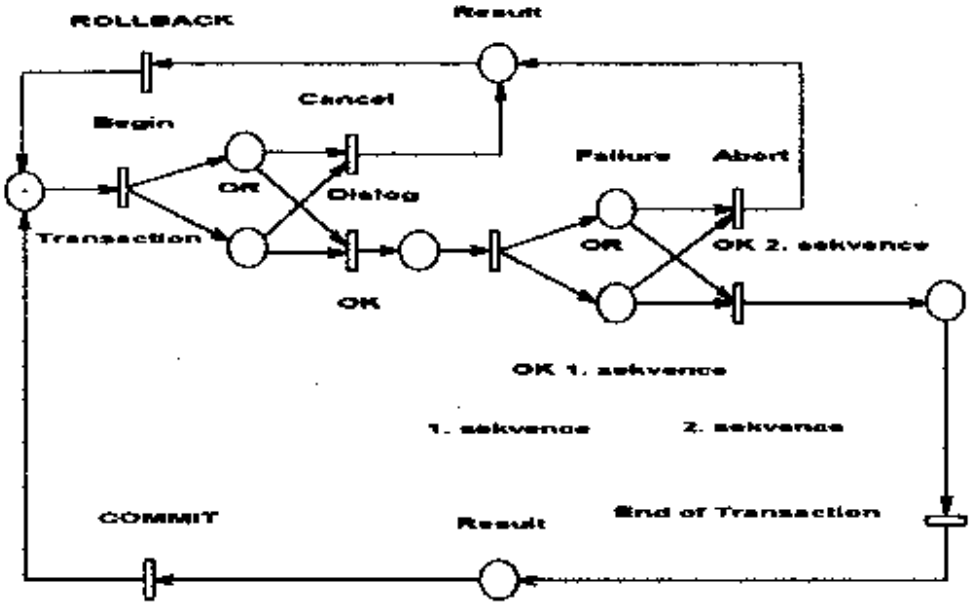
Při interakci dvou objektů je důležité to, aby se jejich operace vzájemně doplňovaly (jeden objekt poskytuje servis, druhý objekt jeho služeb využívá). Z komplementárnosti operací obou interagujících objektů automaticky nevyplývá nositelství děje toho či onoho objektu. Služba může být jedním objektem vnucována druhému objektu stejně tak, jako si může jeden objekt na druhém objektu zpracování vynucovat. Při analýze vztahů mezi objekty je takřka vždy možné určit, který z nich má ve vzájemné interakci dominantní postavení a je tak zvaně aktivní, zatímco druhý objekt sdílí roli podřízeného prvku a je v interakci pasivní. Posloupnost takovýchto interakcí, které zpracovávají událost od počátku do konce, se nazývá transakce.

Nejjednodušším prostředkem pro formální popis transakce (celého sledu vzájemných interakcí výpočetního systému s údaji o objektech reálného světa) je modifikovaná přechodová matice stavů (obr. 2) a grafickým znázorněním Petriho síť (obr. 3). Řádky i sloupce přechodové matice obsahují uspořádaný seznam stabilních prvků systému (na př. proces klienta, dialog, proces serveru, rezultat API a pod.), který je dán sledem jejich účastí na zpracování transakce. Podíl každého prvku systému na provádění transakce je vyjádřen přechodovými procesy umístěnými v přechodové matici na příslušných průsečících řádků a sloupců hlavní diagonály (viz obr. 2), přičemž seznam prvků systému v matici odpovídá řazení procesů při provádění transakce. Procesy umístěné na hlavní diagonále matice jsou chápány jako specifické úspěšně zakončené transformace. Řízení těchto procesů ve výpočetním systému je záležitostí transakčního monitoru.

	Okolí syst.	Klient	Dialog	1. sekv.	2. sekv.	Result	Okolí syst.
Okolí syst	Signál						
Klient		Begin tr.					
Dialog		begin dial.	OK dialog			Cancel	Rollback
1. sekv.			beg. 1. s.	OK 1. s.		Failure	Rollback
2. sekv.				beg. 2. s.	OK 2. s.		
Result					beg. res.	End of tr.	Commit
Okolí syst							Výsledek

Obr.: 2. Přechodová matice znázorňující procesy systému účastnící se na provádění transakce

Jednoduchý příklad na obrázku č. 2 slouží jako ilustrace návrhu transakčního zpracování, v němž jeden abort způsobil uživatel v dialogu (Cancel) a druhý abort je vyvolán systémovou chybou (Failure). Úspěšně zpracovanou transakci je sled přechodových procesů seřazených na hlavní diagonále přechodové matice. Odbočky vyvolané kontrolou průběhu transakce (Cancel a Failure v Result) znamenají zrušení transakce.



Obr.: 3. Značená Petriho síť znázorňující průběh transakce podle tabulky na obr. č. 2

Petriho síť na obr. č. 3 daleko výstižněji presentuje dynamiku transakce než přechodová matice na obr. č. 2. Navíc v komplikovaných případech slouží jako simulátor pro odzkoušení spolehlivosti kauzality procesů, z nichž se transakce skládá. Takto zkonstruovaný dynamický model transakce se stává kvalifikovanou předlohou pro konstrukci programové sekvence řešící dané transakční zpracování.

4. KOMUNIKACE APLIKAČNÍ A DATOVÉ VRSTVY

Softwarovým prostředkem komunikace klienta a serveru je deklarativní programovací jazyk SQL, který umožňuje standardní formu připojení uživatelské aplikace k různým typům databází.

Z hlediska uživatele poskytuje SQL tyto výhody
- nabízí možnost aktualizovat databázi přímo v dané aplikaci,
- minimalizuje potřebu uživatele znát strukturu databáze,
- usnadňuje výměnu informací mezi databázemi, které jsou alokovány na různých počítačích,
- umožňuje komunikaci mezi rozdílnými typy systémů řízení báze dat,
- zmenšuje nároky na školení personálu při přechodu na jiný produkt.

Pomocí SQL lze
- vytvořit databázi,
- řídit přístup k datovým zdrojům v multiuživatelském prostředí,
- manipulovat s daty.

Výše uvedené funkce zajišťují následující komponenty jazyka SQL:
DDL (Data Definition Language), DCL (DataControl Language) a DML (Data Manipulation Language). Příkazy jazyka DDL definují a udržují integritu databází a jejich vnitřních tabulek. Soubor příkazů jazyka DDL se liší mezi jednotlivými databázovými produkty rozsahem funkcí jednotlivých příkazů. Mezi základní příkazy SQL patří

- CREATE, který definuje
  - jméno tabulky
  - jméno každého sloupce uvnitř tabulky
  - datový typ každého sloupce
  - nenulovou hodnotu
  - inicializační hodnotu
  - primární a cizí klíč
  - unikátní sloupec
  - fyzické uložení tabulky.

Možnosti tohoto příkazu je vytvoření virtuální podtabulky dané tabulky pomocí SQL query, kterým je ukládána pouze definice této tabulky a vlastní podtabulka je naplněna daty až při provedení dotazu. Rovněž je možné pomocí tohoto příkazu provést spojení dvou i více tabulek.

- ALTER, který umožňuje změnu definice tabulky, tzn.
  - přidání nového sloupce,
  - zrušení nevyužívaného sloupce,
  - změnu primárního a cizího klíče.
- DROP, který odstraňuje tabulku z databáze, kdy je vymazána definice a obsah tabulky.

Příkazy jazyka DCL se soustřeďují na bezpečnost databáze z hlediska přístupu jednotlivých uživatelů nebo uživatelských skupin k datům. Příkazy primárně tvoří

- GRANT, který určuje privilegia přístupu uživatelů k datům jednotlivých tabulek (čtení, modifikace, výmaz, vložení),
- REVOKE, který odstraňuje nadefinovaná privilegia.

Dané příkazy lze uchovávat jako soubory DOS, tzv. Script SQL soubory, které lze následovně použít při definici tabulky a přesunu dat mezi databázemi.

Příkazy jazyka DML provádějí vlastní práci s daty (vyhledávání a aktualizace) v tabulce databáze. Mezi základní příkazy patří

- SELECT, který hledá data podle jmen tabulek a sloupců. Vyhledávání ve sloupcích lze omezit bližším určením hodnot pomocí konstanty nebo vzoru. Pro hledaná data lze také určit kritéria
  - pořadí,
  - podmínku vyhledávání,
  - masku,
  - agregované funkce (AVG, COUNT, MAX, MIN, SUM),
  - potlačení duplicitních řádků,
  - seskupení dle daných kritérií,
  - spojení více tabulek

Jednotlivé příkazy SELECT lze spojovat a vytvářet tak UNIONY, které kombinují řádky daných SELECT a odstraňují z nich duplicity.

- INSERT, který přidává řádek do tabulky.

- UPDATE, který provádí změny hodnot jednoho nebo více sloupců celé tabulky nebo jen těch řádků, které vyhovují zadané podmínce.
- DELETE, který odstraňuje jeden nebo více řádků z tabulky.

4.1 Charakteristiky implementace SQL

Příkazy jazyka SQL je možné používat interaktivně nebo uvnitř aplikačního programu. Pokud je použit interaktivní způsob, je daný požadavek okamžitě proveden. Při použití zabudovaného SQL v hostitelské aplikaci výsledek požadavku obdrží nejdříve program, který pak podle své logiky rozhodne o způsobu jeho dalšího zpracování (zobrazení, tisk). Zabudované SQL je rozšířením hostitelských programovacích jazyků a představuje nedílnou součást programové aplikace. Distributoři databází obvykle nabízejí své prekompilátory pro dané programovací jazyky. Prekompilátor přeloží zabudované SQL do volání funkcí, které jsou uzpůsobeny syntaxi hostitelského jazyka. Takto vytvořený výstup prekompilátoru je následně zkompilován do object kódu.

Důležitou vlastností SQL je jeho přenositelnost. Oficiálními standardy jsou normy ANSI (American National Standards Institute) a ISO (International Standards Organization), ale v praxi prodávané produkty vykazují od těchto norem mírné odchylky většinou v rozšíření schopností jednotlivých produktů oproti daným normám. Mezi hlavní rozdíly jednotlivých dialektů producentů SQL patří

- Datové typy
- Kódy chyb
- Interaktivní SQL
- Programovací rozhraní
- Systémové tabulky
- Dynamické SQL
- Databázové struktury
- Pořadí srovnávání.

Z existence dialektů SQL plyne nebezpečí, že při použití zabudovaného SQL do aplikace bude nutné při přechodu k jinému typu databáze upravovat příkazy SQL. Pokud se chce projektant vyhnout problémům, použije SQL, které se co nejvíce blíží daným standardům.

Statickým SQL se rozumí tvorba aplikací, které jsou orientovány na procesní zpracování dat, kdy se *parametry* příkazu SQL nemění a příkaz je programovým kódem. Dynamickým SQL se rozumí tvorba aplikací, kdy je příkaz SQL tvořen programem na základě měnících se uživatelských vstupů v okamžiku dokončení jejich zadání. Takový příkaz není kódem aplikace a umožňuje vytváření nových tabulek a indexů.

4.2 Rysy SQL při práci s databázemi

Před připojením aplikačního programu k databázi je nutné identifikovat danou databázi pomocí transakčního objektu. Transakční objekt, který umožňuje komunikaci mezi aplikací a databází, určuje jméno typu databáze, jméno databáze, identifikaci uživatele, heslo, jméno databázového serveru a další speciální

parametry. Pokud aplikace vyžaduje spojení s více databázemi, je nutné vytvořit více transakčních objektů.

Transakce SQL je posloupnost příkazů SQL, která je provedena jako celek. Příkazy, které podporují toto transakční zpracování, jsou

- COMMIT indikující úspěšný konec transakce
- ROLLBACK indikující naopak neúspěšný konec transakce, což si vynucuje návrat k původnímu stavu po provedení dílčích změn v průběhu zpracování transakce.

V případě práce více uživatelů s jednou databází je třeba ošetřit konkurenční prostředí pomocí zámků na úrovních

- databáze
- tabulky
- stránky
- řádku.

Vlastní realizace zámků bývá prováděna jako sdílené čtení nebo exkluzívní psaní. Mnoho databázových produktů má své vlastní techniky provádění zámků a je třeba se s nimi před použitím databáze seznámit.

Většina databází podporuje uložení procedur, jež jsou tvořeny posloupností SQL příkazů. Procedura je pojmenována, zkompilována a uložena v databázi. V této podobě je umožněno její volání z aplikace. Způsob práce s SQL příkazy poskytuje možnost odděleného ověřování, optimalizaci a kompilaci úplné posloupnosti SQL příkazů předem. Triggery jsou požadavky na akci s databází při předem určených podmínkách. Triggery umožňují spustit akci, jakmile nastane událost podmiňující provedení změn v tabulce. Některé databáze poskytují možnost implementace pravidel ověřování platnosti dat v několika úrovních, z čehož vyplývá schopnost databáze udržovat se v konsistentním stavu.

5. ZÁVĚR

Od víceuživatelského přístupu k databázím v informačních systémech budovaných na bázi architektury C/S uživatelé vyžadují jednoduchost ovládání systému, rychlou presentaci informací, přívětivé prostředí a hlavně spolehlivost systému a bezporuchový provoz. Efekt vyplývající z využívání takto navrženého informačního systému je závislý na vyváženosti obou částí IS, technologické i mentální. Jejich koexistence a vzájemná spolupráce mohou vést až k synergickému efektu, pokud se přístup víceuživatelský chápe také jako přístup multiagentový. V referátu jsou k tomuto přístupu položeny základy pouze z hlediska metodicko technologického, nikoliv metodicko mentálního. Druhé hledisko bude dále zpracováváno až do dosažení kritérií optimality a vysoké efektivnosti provozování IS.

Referát je výsledkem ověřování jednoho z dílčích směrů řešení výzkumného úkolu GAČR 201/95/0134 „Gramatické systémy a multiagentové systémy".

LITERATURA

[1] Češka, M.: Petriho sítě. Akademické nakladatelství CERM, Brno 1994

[2] Pokorný, J.: Transakční zpracování. Datasem '96, MFF UK, 1996

[3] Schindler,J.:Událostně orientovaný přístup v simulačních modelech a projektech interaktivních IS. Sborník XVII-th InternationalColloquium-Workshop "Advanced Simulation of Systems", VŠB-TU Ostrava, 1995, vol. 1, p. 96-102, ISBN 80-901751-1-2

[4] Schindler, J.: Životní cyklus objektu - podstata objektově orientovaného modelování logistických a informačních procesů. Konference MME '96, VŠE Praha 1996