Aplikace DBS

Osnova:

1. Úvod - Co je to informační systém (IS), vazba na DBS?
    1.0. Obsah přednášky
    1.1. Základní pojmy databází
    1.2. Komerční databázové systémy
    1.3. Realizace IS

2. Datové modelování, databázové systémy
    2.0. Datový model CDM, PDM
    2.1. Ostatní prvky datové analýzy (triggery, uložené procedury, …)
    2.2. Transakční zpracování
    2.3. Centralizované a distribuované databáze

3. Životní cyklus projektu IS
    3.0. Průzkum trhu
    3.1. Předběžná analýza
    3.2. Analýza systému
    3.3. Projektová studie
    3.4. Implementace
    3.5. Testování
    3.6. Zavádění systému
    3.7. Zkušební provoz
    3.8. Rutinní provoz
    3.9. Reengineering

4. Hodnocení IS, metodiky, cetrifikace

1 Úvod

1.1 Co je to IS, vazba na DBS?

Z hlediska softwarových produktů jich můžeme na trhu najít celou škálu, od operačních systémů, různých podpůrných systémů (textové editory, tabulkové procesory, antivirové produkty, ...), databázových a jiných aplikací až po hry.

Nám však nepůjde o takto izolované produkty. Budeme se zabývat softwarovým vybavením, které je schopné pomoci řízení resp. řídit velké podniky komplexně. Samozřejmě základem takovýchto produktů budou systémy pro zpracování dat - *Databázové systémy.* Toto jádro bude pak obklopeno dalšími komponentami - systémy pošty a podpůrné systémy řízení týmové práce, knihovní systémy, systémy automatizovaného řízení výroby, grafické systémy a mapy, a další potřebné části. Takovýto celek nazýváme *Informačním systémem*.

Realizace takového celku je velmi náročným úkolem. Nejde již o softwarové dílo jednoho nebo skupinky programátorů, zde musí spolupracovat organizovaný tým manažerů, analytiků, specialistů na HW a systémový SW a samozřejmě programátorů. Mnohé z velkých systémů se neobejdou bez spolupráce více firem.

Co by mělo být hlavními kritérii pro tvorbu IS. Na prvním místě je pravidlo: *Systém musí sloužit a pomáhat jeho uživatelům v jejich práci.* Jakékoliv odchýlení od tohoto pravidla vede k nesouladu mezi dodavatelem a uživatelem IS. Každý nesoulad pak vede ke zpomalení a někdy dokonce k úplnému zastavení práce.

Samotné uvedení systému do provozu pak často bývá i otázkou psychologického přístupu než vlastní kvality systému. Odmítání práce se systémem uživateli bývá často příčinou neúspěchu při zavádění IS.

Vlastní systém též závisí na kvalitě do něj zadávaných dat. Je pravidlem, že nejvyšší kvalita dat je tam, kde uživatel je závislý na datech, která do systému vložil. Jinak řečeno, s daty dále pracuje, počítač se stává jeho pracovním nástrojem.

Velmi důležitá je i přívětivost systému a snadné ovládání. Zde opět lze uvést pravidlo: *Aby se systém mohl uživateli jevit jako jednoduchý, bude pravděpodobně uvnitř velmi složitý.*

1.2 Základní pojmy databází

1.2.1 Entitně relační model

E-R model je konceptuálním modelem, jde o popis na úrovni konceptů, nikoliv dat. V souvislosti s informačními systémy slouží k popisu reálného světa a dále se na jeho základě odvíjí popis systému na nižší (např. databázové) úrovni. Z konceptuálního E-R modelu se odvozuje relační schéma databáze.

Je založena na dvou pojmech *entita* a *vztah*. Entitou (v databázové mluvě) se rozumí popsatelný a jednoznačně identifikovatelný objekt. Entity jsou pouhé abstrakce skutečných objektů a popisují se pomocí atributů. Atribut daného typu přiřazuje každé entitě hodnotu, o které se předpokládá, že je již přímo reprezentována pomocí dat. Vztahy mezi entitami mohou být také typy a mohou mít své atributy.

1.2.2 Relace

Formálně lze relaci R zapsat jako n-tici (pro $n \geq 2$) atributů $(A_1{:}D_1, A_2{:}D_2, \ldots, A_n{:}D_n)$, kde A_i je jméno atributu a D_i je doména atributu (množina všech možných hodnot pro daný atribut).

Protože je relace množina, musí být její prvky různé. Jména atributů jsou v rámci jedné relace různá, ale domény se mohou opakovat.

Relaci popisujeme schématem relace $R(A_1:D_1,A_2:D_2,\ldots,A_n:D_n)$

1.2.3 Tabulka

Tabulka je používána jako reprezentace relace v databázi. Jednotlivé záznamy (řádky) v dané tabulce reprezentují n-tici relace a sloupce atributy relace (ovšem atribut zahrnuje celou doménu, kdežto sloupec pouze hodnoty v dané tabulce). Počet sloupců je ekvivalentní aritě (stupni) relace. Na rozdíl od relace může tabulka obsahovat stejné řádky (což je v realitě někdy výhodné), ale to je v rozporu s relací, kde dvě stejné n-tice nejsou dovoleny.

1.2.4 Přirozené spojení

Je operace nad relacemi a jejím výsledkem je nová relace. Prvky nové relace vznikají spojením n-tic z obou relací přes rovnost hodnot na maximální množině společných atributů. Výsledná relace bude mít schéma obsahující atributy z obou relací (včetně duplicitních atributů).

1.2.5 Polospojení

Velmi užitečnou operací používanou speciálně v distribuovaných databázových systémech je tzv. *levé Θ-polospojení*. Výsledkem operace polospojení dvou relací R(A) a S(B) zapsané výrazem $(R[t_1\Theta t_2]S)[Atr(R)]$ (kde $t_1 \in A$, $t_2 \in B$) je relace obsahující pouze ty n-tice z relace R, které jsou spojitelné pomocí podmínky $t_1\Theta t_2$. Analogicky je možné definovat i *pravé Θ-polospojení*. Je-li směr patrný z kontextu, mluvíme pouze o *polospojení*. Speciálním případem je *polospojení přes rovnost* ($t_1\Theta t_2$ je rovností). Možné je také *přirozené polospojení* definované jako polospojení přes rovnost, kde rovnosti jsou mezi všemi společnými atributy. Toto polospojení se značí R<* S.

1.2.6 Transakce

Je posloupnost logicky souvisejících akcí, jejichž provedením přechází databáze z jednoho konzistentního stavu do druhého, i když v průběhu provádění transakce může databáze být v nekonzistentním stavu.

1.2.7 Distribuovaná transakce

Jedná se o transakci v distribuovaných databázích. Tato transakce je rozdělena na jednotlivé části (podtransakce), které jsou prováděny na různých místech distribuovaného systému paralelně.

1.2.8 Sériové spouštění transakcí

Sériové spouštění transakcí je takové, kdy pro každé dvě transakce platí, že všechny operace jedné z nich jsou spuštěny před jakoukoliv operací druhé.

1.2.9 Serializovatelné spouštění transakcí

Serializovatelným spouštěním transakcí se nazývá takové spouštění, které dává stejné výsledky jako sériové.

1.2.10 Rozvrh transakcí

Rozvrhem je chápáno stanovení pořadí provádění jednotlivých operací více transakcí v čase.

1.3 INFORMAČNÍ SYSTÉM

Informace vyvolává změnu stavu nebo chování příjemce.
Data (to, s čím přichází příjemce do styku) zobrazují stavy objektů či probíhající procesy v realitě kolem nás. V závislosti na způsobu a okolnostech prezentace dat buď představují tato data pro příjemce informaci, nebo nikoli.
Znalosti představují zobecněné poznání (určité části) reality. Znalosti souvisejí s vymezováním pojmů, s kategorizací, s definováním, s odvozováním závěrů z dostupných faktů na základě abstraktních schémat (hypotéz) a s vymezováním mechanismů (postupů) odvozování závěrů.

Je systém sběru, uchovávání , analýzy a prezentace dat určený pro poskytování informací mnoha uživatelům různých profesí.

- může / nemusí být podporován počítačem (při návrhu IS zkoumáme optimální kombinaci automatizovaných a neautomatizovaných činností)
- musí disponovat prostředky sběru, kontroly a uchování dat
- data x informace
  Informace jsou jen ta data, která dokážeme využít, přiřadit jim význam či smysl. (Při návrhu IS nutno umožnit získávání odlišných informací pro různé zaměstnance – skladník, ředitel. Při tvorbě IS často konkrétní potřeby uživatelů nejsou známy – nejasnost a proměnlivost potřeb)
- IS ovlivňují pracovní procesy i organizační struktura podniků (při návrhu IS se často řeší změny v organizaci zákazníka)
- IS je vždy společným dílem dodavatele a zákazníka.

- vývoj a nasazení IS (nelze přímo koupit a použít)
  a) IS vyvíjen od začátku – návrh..., modely
  b) IS koupen a přizpůsoben potřebám zákazníka (customizace)
  (nutno provádět analýzu potřeb a formulovat požadavky zákazníka)

- počáteční etapy vývoje - stanovení cílů, formulace požadavků - velmi dlouhé a pracné

Převzít nebo vyvíjet?
Uživatelé dnes IS obvykle nevyvíjejí. Vývoj IS i customizace prováděné specializovanými firmami a nikoliv přímo uživatelem jsou dnes standardem.
Výhody a nevýhody customizovaného IS:
(+) menší nebezpečí, že dodavatel opustí trh, customizovaný IS bývá obvykle podporován větší firmou
(-) neodpovídá přesně potřebám. To obvykle znamená menší účinnost a také vyšší náklady na reorganizace, které by jinak nemusely být nutné
(-) IS má i konkurence, takže neposkytuje podstatnou výhodu před konkurencí
(*) vyšší nabídka funkcí, které však nemusí být vždy potřebné a pak zbytečně zvyšují nároky na obsluhu systému a také na hardware
(+) obsahuje know-how mnoha instalací, dodavatel většinou poskytuje přesné postupy pro zjišťování požadavku, instalaci, školení koncových uživatelů a oživování systému na místě
(+) ověřeno na více instalacích (reference, lze převzít zkušenosti)

(+) úspora nákladů na vývoj a především údržbu
(-) vyšší nebezpečí, že je IS založen na zastaralých technologiích
(-) u cizích systémů nedostatečná lokalizace (potíže s českou legislativou a abecedou)
(-) obtíže s integrací produktů třetích stran a existujících aplikací (např. u SW pro řízení technologií)

Shrnutí problémů počátečních etap vývoje a customizace IS

1. podceňování počátečních etap
2. vedení projektu
3. organizační problémy (neujasněné cíle)
4. syndrom dortu pejska a kočičky (chceme všechno)
5. nadbytečná přesnost
6. předčasné řešení technických problémů
7. zamlčené předpoklady, opominuté souvislosti
8. nedostatečná analýza dat
9. měřitelnost výsledků
10. volba termínů

1.3.1 Analýza

Stanovení cílů projektu (SCP – projektový záměr)

Cíle projektu by měly být stanoveny ve formě písemného dokumentu. Účelem dokumentu je rámcově stanovit funkce a další vlastnosti projektu. Dokument budou posuzovat pracovníci různých profesí a proto má být formulován srozumitelně, bez zbytečných podrobností, avšak dostatečně přesně. Odtud vyplývá. že dokument stanoveni cílu (SCP) musí být ve většině případů spíš intuitivní než formálně přesný .Dokument je rozpracováván (a zpřesňován) v etapě specifikace požadavků. Nemá přitom docházet ke změně podstatných prvků cílu. SCP má obvykle následující strukturu:

1. Název projektu (případně identifikační kód projektu).
2. Shrnutí cílů (formulace problému): Formulace celkového úkolu systému formou srozumitelnou i nečlenům týmu. Tato část má být stručná a vystihnout podstatu.
3. Vymezení uživatelů (kdo. kdy a za jakých okolností bude systém využívat, případně pro koho systém není určen).
4. Seznam nejdůležitějších funkcí spolu se stručnými popisy funkcí. Popis je formulován z hlediska uživatele.
5. Zásady pro dokumentování použité normy.
6. Vazby na jiné projekty a systémy.
7. Rámcové požadavky na hardware (konfigurace, spolehlivost. ...) a požadavky na efektivnost zpracování.
8. Metody ochrany dat, žádoucí způsoby využívání dodávaného softwaru.
9. Požadavky na spolehlivost systému jako celku (doba mezi chybami, vzpamatování po chybě, ochrana dat, funkce nutné pro detekci chyb).
10. Předpokládané termíny realizace a náklady na realizaci.
11. Způsob předání

12. Perspektivy rea1izovaného systému, jeho další rozvoj a zajištění údržby, pravidla šíření

Specifikace požadavků

- **Formální specifikace -** Písemné zadání návrhu informačního systému. Používá se i zobrazení pomocí DFD a tabulek.
- **Neformální specifikace -** Návrh informačního systému zapsaný pomocí jednoduchého programovacího jazyka, buď existujícího nebo smyšleného, na základě využití procedur a zápisu jednotlivých vztahů mezi nimi. Často jsou součástí i požadavky na zobrazované položky a vzhled výstupů informačního systému.

Vychází z dokumentu „Stanovení cílů“:

1. Název projektu a identifikátor projektu.
2. Úvod – shrnutí úkolů systému v obecně srozumitelné formě, shrnutí má být krátké.
3. Vymezení uživatelů (kdo, kdy, jak bude systém používat) a způsob využití produktu.
4. Perspektivy realizovaného systému (doba života, možnosti předání dalším uživatelům).
5. Způsoby vedení dokumentace.
6. Zajištění spolupráce mezi dodavatelem a uživatelem.
7. Dokumenty odkazované v textu.
8. Použité zkratky a slovník všech termínů používaných v textu, u nichž je nebezpečí, že nebudou správně chápány.
9. Vazby na jiné projekty.
10. Požadavky na hardware, efektivnost a spolehlivost (doba mezi výpadky, ochrana dat, metody detekce chyb...).
11. Rozpis dat a funkcí.
12. Plán testů (specifikace testů, testových procedur a testových případů).
13. Vymezení obsahu dokumentace předávané uživateli.
14. Termíny realizace, plán realizace.
15. Ekonomické a organizační zajištění (odhad nákladů, kdo jsou řešitelé...).
16. Vymezení způsobu údržby a způsobu prodeje produktu dalším uživatelům.

2 Datové modelování

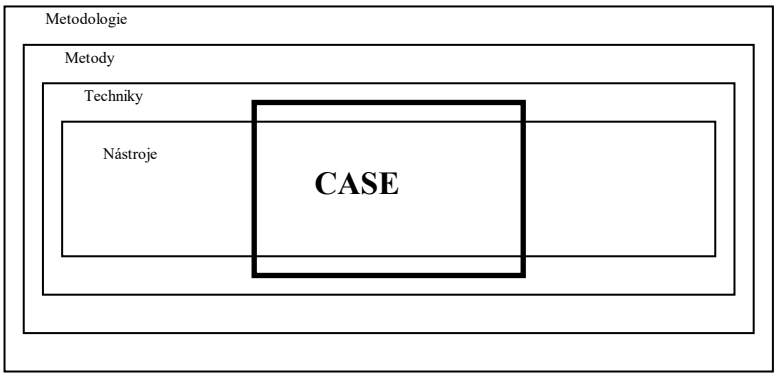
2.1 Datový model CDM, PDM

Metodologie je souhrn postupů a metod, CO, KDO a KDY se má dělat při projektování, implementaci a řízení IS.

Metoda stanovuje, CO je potřeba dělat v určité fázi projektu - např. model dat, funkční model atd. (př. Yourdon Structured Analisis Design).

Technikou rozumíme způsob, JAK vytvořit to, co je dáno metodou. (př. ERA model pro datový model, TopDown pro funkční analýzu).

Nástrojem je to, ČÍM realizujeme techniku, čím vyjádříme výsledek, např. data flow diagram, diagram entit a relací. Speciálním nástrojem je i CASE.



Yourdon Structured Method

Nejpoužívanější metodologie. Podstatou je modelování, vytváření obrazu reality specifickými prostředky. Základní charakteristiky YSM:

1. Grafičnost - grafická podpora návrhu software, větší vypovídací schopnost

2. Podpora TOPDOWN principu
   - dělení na podsystémy DISKRÉTNĚ
   - struktura lze vyjádřit STROMOVOU STRUKTUROU
   - vše společné jednotlivým subsystémům je obsaženo ve VYŠŠÍ ÚROVNI

3. Minimalizace redundance
   - použití DATA DICTIONARY

4. Poskytovat možnost předvídat chování systému

- ideální je tvorba prototypu
- dostačuje tvorba uživatelského rozhraní

5. Být snadno čitelný
   - používat jednoduché symboly

YSM pokrývá fáze systému:

- analýza požadavků
- specifikace (popis) systému
- konstrukce (design) systému
- implementace

YSM vytváří čtyři nezávislé modely:

- datový model systému
- funkční model systému
- model řízení systému
- model struktury programového systému

Datový model

Statický model systému. Zachycuje objekty a vztahy mezi nimi, např. entitně relační model.

Funkční model

Dynamický pohled na systém. Popisuje místa kde dochází k transfirmaci dat. Diagramy struktury funkcí, diagramy datových toků, slovní popisy funkcí.

Model řízení

Diagram stavů a přechodů. Diagram řídících toků.

Model struktury programového systému

Souhrn modulů a vazeb mezi nimi.

CASE

CASE - Computer Aided Software Engeneering. Jedná se o prostředky pro podporu projektování software. Podle funkčního záběru se produkty CASE dělí do tří skupin:

UPPER CASE

"makro" úroveň projektu
popisy a plánování systémů řízení a organizace
implementace teoretické metody pro návrh systémů
silné prostředky pro formální prezentaci výsledků

MIDDLE CASE
analýza a návrhy datových struktur
analýza a návrhy, případně i optimalizace funkčních struktur
podpora prototypového řešení
prostředky pro tvorbu dokumentace

LOWER CASE
automatizace kódování programů do zdrojového kódu
dokumentace k programům

2.1.1 Datové modelování v nástroji PowerDesigner

2.2 Ostatní prvky datové analýzy (triggery, uložené procedury, …)

3 Databázové systémy

Vývojáři informačních systémů tak mohou využít řady standardních zabezpečovacích mechanismů, kterými SŘBD (zpravidla pak databázový server) disponuje. Navíc lze konkrétní SŘBD většinou získat v různých provedeních lišících se úrovní zabezpečení, případně je možné využít nadstaveb nezávislých dodavatelů (například pro zajištění zabezpečeného přenosu dat). Různé varianty jsou výhodné jak pro zákazníky, kteří nejsou nuceni platit za vlastnosti, které nevyužijí, tak pro dodavatele, který může zvýšit úroveň zabezpečení pouhou změnou varianty databázového serveru (tvrzení je však nutno chápat v intencích konkrétního nasazení daného informačního systému – někdy toto přímočaré pravidlo nemusí být pravdivé).

Co vše by měl nabídnout aneb autentizace ...
Každý SŘBD by měl zajistit autentizaci uživatele přihlašovaného k databázi. Zpravidla se tak děje pomocí vnitřních mechanismů – v databázi jsou vytvořeni uživatelé s přidělenými (i jimi samými) hesly, která bývají šifrována. Takový uživatel se pak k databázi přihlašuje pomocí svého jména a odpovídajícího hesla. Není to však zdaleka jediná možnost – k autentizaci mohou být použity i speciální aplikace (například bezpečnostní či adresářové servery), hardwarová zařízení (čipové karty) či služby operačního systému (je-li uživatel platně přihlášen do operačního systému, má umožněn přístup i k databázi).

Bezpečnostní správce (administrátor) by měl mít možnost právo (oprávnění) pro připojení k databázi nejen přidělit, ale také odebrat či časově (případně jinak – například počtem přihlášení za aktuální měsíc) omezit jeho platnost. Výhodou dále je, umožní-li SŘBD definovat akce, které se mají vykonat při nesplnění autentizačních podmínek (kontaktovat bezpečnostního správce – například zasláním elektronické pošty či zprávy na operátor, zablokovat uživatelský účet nebo odebrat oprávnění přístupu ke chráněným datům).

Samozřejmostí by mělo být vygenerování záznamu do sledovacích protokolů (stejně jako v případě porušení ostatních bezpečnostních mechanismů). Nelze jednoznačně konstatovat, který způsob autentizace a reakce na selhání je nejlepší – vždy záleží na konkrétní situaci.

autorizace ...
Další oblastí je zajištění zabezpečení provádění systémových akcí v databázi – vytváření, modifikace a rušení uživatelů, úložných prostorů, databázových objektů (tabulek, indexů atd.), spouštění a zastavování databáze či nastavování vlastností bezpečnostních mechanismů. Hovoříme o tzv. autorizaci. V návaznosti na zvolenou bezpečnostní strategii bývají oprávnění na provedení těchto akcí rozdělena mezi více subjektů (nejčastěji mezi databázového administrátora, bezpečnostního správce a uživatele – v případě databázových objektů, jako jsou zmiňované tabulky, indexy).

Úlohu bezpečnostního správce mnohdy supluje administrátor aplikace, který rozhoduje o přiřazení přístupových práv ke konkrétním databázovým objektům (zpravidla tabulkám, pohledům a uloženým programům – procedurám či funkcím) a jednotlivým datům (některé SŘBD umožňují omezit přístup až na úroveň konkrétního záznamu či atributu). Možná oprávnění lze rozlišit na vkládání, změnu a mazání dat, případně vykonání programového kódu. Pokud konkrétní SŘBD neposkytuje dostatečné možnosti v řízení přístupu k datům, bývá nutné tento handicap řešit programově – například pomocí triggerů.

audit ...
Při vyjmenovávání vlastností zabezpečení se poměrně často zapomíná na sledování jednotlivých událostí a jejich následnou analýzu (hovoříme o tzv. auditu). Do speciálních souborů (případně tabulek) – deníku auditu (auditního deníku, protokolu událostí, sledovacího protokolu) jsou ukládány tzv. auditní záznamy.

Auditní záznamy by měly obsahovat co možná nejpřesnější informace charakterizující danou událost – například v případě neoprávněného přístupu k datům uživatele, datum a čas, terminál (klientský počítač), identifikaci připojení, údaje o dalších připojeních stejného uživatele atd. Mnohé SŘBD umožňují určit, které události a kdy (při úspěchu či selhání) se mají sledovat. Podstatné je, aby byl deník auditu průběžně kontrolován – jinak mnohé průniky do systému ani nemusí být odhaleny (naštěstí ne každý "profesionální škůdce" dokáže za sebou zamést všechny stopy).

Obdobně jako vlastní data je nutno i auditní záznamy považovat za velmi citlivé informace. SŘBD by měl také umožnit nadefinovat událost, která se vykoná při zapsání auditního záznamu daného typu (obdobně jako v případě nesplnění podmínek autorizace).

komunikace ...
Bez ohledu na použitou architekturu (klient/server, file/server atd.) by měl každý SŘBD nabídnou alespoň základní zabezpečení přenášených dat. Tento požadavek může být řešen pomocí standardních zabezpečovacích mechanismů v počítačových sítích (za využití některého z tzv. bezpečných protokolů), mnohdy v kombinaci s využitím vlastních síťových rozhraní. Možných variant je celá řada a záleží nejen na schopnostech serveru, ale také na možnostech použitých klientů.

Práva a role
Vlastní zabezpečení přístupu k datům bývá v současných SŘBD realizováno především na základě tzv. volitelného řízení přístupu (DAC - Discretionary Access Control). To vychází z myšlenky, že oprávněný (jak z hlediska přístupu, tak z hlediska poskytování práv) uživatel přiřazuje (přiděluje) konkrétní přístupová práva dalším uživatelům či jejich skupinám. Shodného mechanismu se používá i u zabezpečení systémových akcí (viz odstavec o autorizaci). Součástí přiřazení práva jinému uživateli může být také delegování pravomoci přidělovat daná práva (z obdarovaného se tak stává oprávněný uživatel) někomu dalšímu.

V běžné praxi to ale vypadá tak, že jednotlivá práva jsou sdružována do skupin (tzv. rolí), a teprve ty jsou přiřazovány konkrétním uživatelům (případně jiným rolím). Výhoda je zřejmá – zjednodušená administrace (právo je přiděleno či odebráno pouze jednou a změna se promítne u všech uživatelů, kterým byla měněná role přiřazena). Pouze v případech, kdy by použití rolí situaci spíše komplikovalo, jsou práva uživatelům přiřazována jednotlivě (někdy se také hovoří o přímém a nepřímém – u rolí – přidělování práv).

Některé SŘBD umožňují vytvořit veřejnou roli (většinou pojmenovanou public), kterou mají automaticky přiřazenou všichni uživatelé. Přidělování práv této roli by mělo být zváženo vždy velmi důkladně. Nad rolemi a právy je nezbytně nutné přemýšlet v hlubších souvislostech - odebrání konkrétního práva uživateli či z role public ještě nemusí znamenat, že uživatel o dané právo opravdu přijde. Může jej mít například přiřazeno zprostředkovaně jinou rolí. Stejně tak je nutno nastudovat, jak v konkrétním SŘBD jsou práva vyhodnocována –

například u uložené procedury mohou být primárně brána do úvahy práva (k procedurou používaným objektům) vlastníka procedury, nikoli práva toho, kdo proceduru spouští.

Na co omezení zdrojů?
Důležitou vlastností, kterou by měl nabídnout každý vyspělý SŘBD, je tzv. omezení zdrojů. Ve své podstatě jde o to, že uživateli (případně procesu serveru) jsou přiděleny kvóty na využití systémových zdrojů, jako je doba připojení, zatížení CPU, rozsah přenášených dat, maximální počet konkurenčních připojení (nemusí přitom jít vždy o jednoho uživatele) či umožnění připojení k databázi pouze v určitou dobu.

Tato pravidla sice nemají na první pohled přímý vztah k zabezpečení uložených dat, na druhou stranu mohou být v realizaci bezpečnostní strategie užitečná. Například snadněji se odhalí neoprávněný pokus o připojení k databázi tehdy, má-li daný uživatel možnost pouze jednoho (celkem) konkurenčního připojení a v okamžiku pokusu o průlom je z jiného místa připojen. Navíc cílem průlomu nemusí být pouze získání či změna dat, ale také zahlcení přenosových linek, zablokování možnosti změn dat či úplné vyřazení databázového serveru z provozu. A právě v těchto případech je vhodné využít i omezení zdrojů.

Záleží i na fyzickém uložení
Všechna data v databázi jsou fyzicky uložena na nějakém paměťovém médiu, v dnešní době nejčastěji na pevných discích. Z tohoto pohledu může být zabezpečení ovlivněno celou řadou dalších faktorů – od možnosti šifrování přes zabezpečení ze strany operačního systému (vyřešené zabezpečení přístupu k datům nám příliš nepomůže, když si nezašifrovaný soubor může běžný uživatel zkopírovat a doma analyzovat) až po speciální typy instalací, například do samostatných oddílů pevného disku.

Může být také účelné neshlukovat citlivá data do jednoho místa (záleží ovšem na konkrétní situaci – rozdělené umístění totiž může přinést zvýšené náklady z důvodů opakovaného zabezpečení). Stejně jako v případě dalších mechanismů zabezpečení je i zde nutno uvážit možný pokles výkonu.

Každý program obsahuje nejméně jednu chybu!
Přes odpovědný přístup tvůrců SŘBD je zarážející nedodržování některých základních opatření. Je možné se tak setkat s nenahrazováním hesla zástupnými znaky (či náhodnou změnou jejich počtu) v přihlašovacím dialogu, s umožněním přístupu k seznamu všech uživatelů (již samotná znalost uživatelova jména je potenciální hrozbou) nebo s chybnou identifikací uživatele v záznamech deníku auditu.

Přestože se jedná spíše o náhodné případy, je vhodné se v okamžiku požadavku na kvalitně zabezpečený systém obrátit na dodavatele, který využívá SŘBD osvědčených firem a má s touto problematikou dostatečné zkušenosti. Uvedené nedostatky mohou být mnohdy odstraněny během pár minut instalací opravného patche. Pokud však dodavatel o této možnosti neví, můžete se v budoucnu dočkat nemilých překvapení.

Je to vše?
Zdaleka není – uvedli jsme si zde pouze základní možnosti a oblasti zabezpečení databází. Součástí bezpečnostní strategie (politiky) by mělo být například i zálohování a archivace dat (zálohování a archivy mohou být cílem útoku také) či definování vlastností hesel (složitost, délka platnosti či možnost opakování). Není možné zapomínat ani na další oblasti

informačního systému – tedy na již zmiňované operační systémy, hardware či organizační opatření.

Zabezpečení dat je dnes ve vyspělých SŘBD řešeno kvalitně a v kombinaci s dalšími prvky nemusí být vytvoření bezpečného informačního systému až takovým problémem, jak by se na první pohled mohlo zdát. Na úplný závěr připomeňme, že k průniku může dojít i do sebelépe zabezpečeného systému – zejména při selhání lidského faktoru a nedodržení bezpečnostních opatření na organizační úrovni.

3.1 Transakční zpracování

V aplikacích se používá pojem *transakce*, což je skupina operací prováděných nad databází.

Transakce musí splňovat vlastnosti ACID:

- **Atomicity (nedělitelnost) – transakce se tváří jako jeden celek, tedy je vykonána celá, nebo vůbec**
- **Consistency (konzistence) – transakce může měnit databázi pouze z jednoho konzistentního stavu do druhého konzistentního stavu**
- **Isolation (izolace) – transakce je izolovaná od probíhajících změn (jsou viditelné pouze potvrzené (commited) změny), tj. dílčí efekty transakce nejsou viditelné ostatním**
- **Durability (trvanlivost) – změny v databázi provedené potvrzenou transakcí musí být trvanlivé**

Podívejme se teď na jednotlivé vlastnosti podrobněji z hlediska praxe.

Atomicita je základním a nejvíce viditelným pilířem transakčního zpracování. Díky této vlastnosti je vždy zajištěno korektní provádění sdružených změn, jako je například přesun peněz mezi účty (odečet z jednoho, přičtení k druhému). Neúplné sdružené zápisy jsou noční můrou všech aplikací postavených na souborových databázích bez využití transakcí; již z tohoto důvodu je dobré přejít na některý z moderních databázových systémů s podporou TS.

Atomicita zajišťuje, že při chybě nebo zrušení transakce v průběhu jakékoliv operace dojde k obnově původního stavu před zahájením transakce. Důležitým faktorem z hlediska praxe je právě možnost kdykoliv zrušit provádění transakce z vůle klienta (operace ROLLBACK), k čemuž většinou dochází z důvodu odhalení nesrovnalostí v datech až v průběhu jejich zpracování. Protože některé transakce mohou být velmi složité a časově náročné a návrat zpracování na úplný začátek z důvodu jediné nefatální chyby je velice nepraktický, umožňují některé databázové platformy detailnější rozdělení operací v rámci transakce na menší bloky a následný návrat pouze na začátek aktuálně prováděného bloku. Tato vlastnost může být prezentována různě, buď jako vnořené transakce (nested transactions) nebo jako body návratu (check points); přináší také řadu omezení v závislosti na konkrétní platformě. Body návratu mohou mít navíc u některých systémů význam návratu do časového okamžiku bez vazby na konkrétní transakci. Je tedy nutné před jejich použitím vždy pečlivě prostudovat dokumentaci. Navíc je použití těchto vlastností spojeno s vyššími nároky na databázovou platformu a má

vliv na její výkon a spotřebu zdrojů – především paměti a diskového prostoru. Proto je vhodné tyto vlastnosti používat jen v opravdu nezbytných případech.

Konzistence je poměrně problematická vlastnost. Většina databázových platforem (především systémů založených na jazyce SQL) nabízí více či méně komplexní aparát pro zajištění konzistence dat v databázi. Tento aparát je typicky tvořen možností definovat kontrolní pravidla pro jednotlivé sloupce tabulek, pro vztahy dat v rámci tabulky a základní vztahy mezi tabulkami. Navíc lze definovat i velmi složitá pravidla s pomocí uložených procedur a triggerů. Tato pravidla jsou následně databázovými platformami automaticky průběžně kontrolována při každé operaci a jejich porušení je indikováno jako chyba. Tím je ovšem porušeno pravidlo, které dovoluje nekonzistence dat v rámci probíhající transakce, což může v praxi přinést řadu problémů. Navíc má automatická kontrola konzistence dat nemalý negativní vliv na výkon a na nároky dané platformy. Proto je v praxi často voleno kompromisní řešení, které částečně nebo zcela zajišťuje konzistenci dat pomocí kontroly dat v klientské aplikaci a pečlivě zpracovaných transakcích. Automatická kontrola konzistence dat je vhodná díky svému centrálnímu zpracování na serveru především v těch případech, kdy k databázi přistupují klienti různými způsoby. Ať už v praxi zvolíte jakýkoliv způsob pro zajištění konzistence dat, bude vaše databáze konzistentní jen do té míry, jak dobře navrhnete a naimplementujete svůj systém.

Konzistence tedy není skutečná, ale pouze předpokládaná vlastnost transakcí. Její konkrétní naplnění záleží vždy jen na vás.

Izolovanost je nejvíce matoucí vlastností transakcí a v praxi je často zdrojem závažných problémů. Cílem vzájemného izolování transakcí je zamezit interferencím mezi transakcemi a následné ztrátě dat nebo jejich chybné interpretaci (čtení nepotvrzených údajů). Vzájemnému ovlivňování transakcí lze zcela zamezit pouze jejich serializací, což má ovšem negativní a zpravidla neakceptovatelný dopad na výkon systému. Protože jsou oba protichůdné požadavky na korektnost zpracování dat i co nejvyšší propustnost systému kritickými faktory, musí databázové platformy k uspokojení obou požadavků implementovat nějakou formu synchronizace současně zpracovávaných transakcí. Usmířit vodu a oheň ovšem nelze bez kompromisu. Aby byla celá, již tak dost komplikovaná situace ještě zábavnější, nabízejí databázové platformy různé varianty tohoto kompromisu v podobě tzv. úrovní izolace (isolation level).

Problematika izolace a synchronizace současně prováděných transakcí je obsáhlá a budeme se jí v plné míře věnovat později.

Trvalost změn úspěšně ukončené transakce je nejlépe srozumitelnou vlastností transakčního zpracování. Přesto má jeden ne zcela zřejmý aspekt, kterým je zvláštní význam potvrzení transakce pro databázový systém. Potvrzení transakce (operace COMMIT) totiž hraje klíčovou roli v práci každé databázové platformy a je s ním spojena řada činností a vnějších projevů systému.

Souběžné transakce

Při souběžném zpracování transakcí se samozřejmě může stát, že různé transakce zpracovávají v jeden okamžik stejná data. Pokud se různé transakce pokusí aktualizovat stejné údaje, může dojít ke ztrátě informace v případě, že změněná data jsou opětovně změněna jinou transakcí dříve než je původní změna potvrzena. K problémům ovšem může dojít i v

případě že transakce data pouze čte, pokud se je jiná transakce zároveň pokouší změnit nebo nová data přidává. V takovém případě může transakce načíst doposud nepotvrzené změny, případně změny sice potvrzené, ovšem nekonzistentní s daty transakce (fantómové řádky, nereprodukovatelné čtení). To může způsobit problémy především u dlouhotrvajících transakcí, které zpracovávají velké množství dat.
Aby k těmto problémům nedocházelo, řídí server přístup transakcí k databázi obdobným způsobem, jako je třeba řídit přístup k sdíleným prostředkům u vícevláknových aplikací. Řízení přístupu je typicky realizováno pomocí různých typů zámků. Protože soupeření transakcí o data má negativní vliv na propustnost systému, je definováno několik úrovní izolace transakcí, které představují různou míru kompromisu mezi volným a tudíž nechráněným přístupem, a exkluzivním přístupem blokujícím práci ostatních transakcí. Úroveň izolace je vlastnost transakce a různé transakce tedy mohou pracovat s odlišnou úrovní izolace.

Standard SQL92 definuje následující izolační úrovně:

Read Uncommitted
Tato izolační úroveň dovoluje nejvyšší propustnost ze všech standardem definovaných možností. Souběžné aktualizace jsou sice blokovány, ale transakce může číst i dosud nepotvrzené změny provedené jinými transakcemi. To ovšem staví na hlavu samotný princip (ACID) transakcí, protože ačkoliv nemůže dojít ke ztrátě informace z důvodu přepsání změn jinou transakcí, může dojít k celé škále anomálií při čtení nepotvrzených dat.

Z výše uvedených důvodů lze tuto izolační úroveň použít jen pro transakce, které buď pouze data mění, nebo u kterých nezáleží na korektnosti a konzistenci přečtených dat. Protože jen málokterá transakce splňuje tyto podmínky (nutno vzít v úvahu i automatické zpracování dat na serveru pomocí triggerů), je vhodné tuto izolační úroveň vůbec nepoužívat. Některé databázové platformy (jako například InterBase/Firebird a PostgreSQL) tuto úroveň izolace z bezpečnostních důvodů vůbec nepřipouští.

Read Committed
Jak již název napovídá, blokuje tato izolační úroveň nejen souběžné aktualizace, ale navíc dovoluje čtení pouze potvrzených změn provedených jinými transakcemi. Pokud jsou transakce které provádějí změny krátké, dovoluje tato izolační úroveň velmi vysokou propustnost srovnatelnou s propustností úrovně Read Uncommitted. Ačkoliv nelze číst nepotvrzená data, může stále docházet k anomáliím při čtení dat (nereprodukovatelné čtení, fantómové řádky). Úroveň Read Committed je vhodná pro transakce které provádějí především změny dat, a které nevyžadují stabilní pohled na data. Není vhodná pro transakce, které provádějí zpracování velkého množství dat (např. generování sestav), případně mění data na základě dříve přečtených údajů (přepočty cen apod.)

Repeatable Read
Mimo vlastností úrovně Read Committed zajišťuje tato úroveň izolace navíc stabilní pohled na čtená data tak, aby bylo zajištěno že jednou přečtené údaje nebudou změněny. Mohou ovšem vznikat anomálie z přidání nových dat (fantomové záznamy), kdy opakované čtení téže množiny může vrátit odlišný počet záznamů.

U mnoha databázových platforem vyvolá pouhé čtení dat jejich uzamčení proti změnám. Tuto izolační úroveň je tedy vhodné používat pouze u takových transakcí, které ze své podstaty

vyžadují stabilní pohled na data (např. generování sestav). Možnost vzniku fantomových záznamů ovšem vylučuje použít tuto izolační úroveň pro transakce, které na základě načtených údajů přidávají nebo mění data (např. přecenění skladu, účetní uzávěrka).

Serializable
Úroveň Serializable zabraňuje všem interferencím mezi transakcemi stejným způsobem, jako by transakce byly prováděny postupně a nikoliv souběžně. Ostatní transakce sice mohou číst data ze zpracovávaných tabulek, ale nemohou je měnit ani přidávat nové údaje.

Z důvodu značného blokování ostatních transakcí je vhodné používat tuto úroveň izolace pouze v nezbytných případech (např. přecenění skladu, účetní uzávěrka).

Různé databázové platformy mohou pro jednotlivé izolační úrovně používat odlišné názvy (nebo i stejné ale v jiném významu), případně nemusí podporovat všechny úrovně definované standardem, a lišit se může i dopad jednotlivých úrovní na propustnost transakcí. Je tedy vždy nezbytné prostudovat dokumentaci konkrétního produktu.

Protože různé transakce mohou používat odlišné izolační úrovně, liší se i míra vzájemného blokování mezi souběžnými transakcemi podle konkrétní míry izolace soupeřících transakcí. Protože jsou úrovně Repeatable Read a Serializable u většiny systémů pohromou pro propustnost souběžného zpracování dat, je vhodné je používat jen v nezbytných případech a pro "běžnou" práci s databází používat úroveň Read Committed.

Jedním z důvodů izolace transakcí je zabránění tzv. *dominovému efektu*. Problémy s paralelním zpracováním transakcí je nutné kombinovat s problémy týkajícími se zotavení z chyb. Řeší se otázka, které transakce spustit znovu v případě jejich vzájemné závislosti podle sdílených objektů. Například pokud transakce A, B, C pracují současně s objektem Z a některá z transakcí měnící objekt Z se dostane do chybného stavu a je zrušena, musí být zrušeny všechny transakce pracující se Z, protože používaly nesprávnou hodnotu objektu Z. Tomuto jevu se říká *kaskádové rušení transakcí* neboli *dominový efekt*. Je zřejmé, že pojem izolace je přímo vztažen ke konzistenci databáze a paralelnímu zpracování.

Druhá vlastnost (Consistency) je v rukou aplikačních programátorů, ostatní tři vlastnosti musí být zajištěny DDBS.

Jestliže počáteční stav databáze je konzistentní a jestliže každý transakční program je navržen tak, aby udržel konzistentní databázi pokud je spouštěn izolovaně, pak spouštění ekvivalentní sériovému neobsahuje transakci, která zachovává nekonzistentní databázi.

3.1.1 Kontrola souběžnosti (concurrency control)
Proč vlastně kontrola souběžnosti transakcí? V případě kdy běží několik transakcí současně, může se stát, že tyto transakce mění tentýž databázový záznam. Pokud by tento souběžný běh transakcí nebyl kontrolován, mohl by nastat problém, jakým je *nekonzistentní databáze*. Proto v další části popíšeme tři problémy, které mohou nastat při nekontrolovaném běhu souběžných transakcí. Všechny tyto problémy budou ilustrovány na příkladu dvou transakcí T_1 a T_2, jejichž operace jsou znázorněny na obrázku.

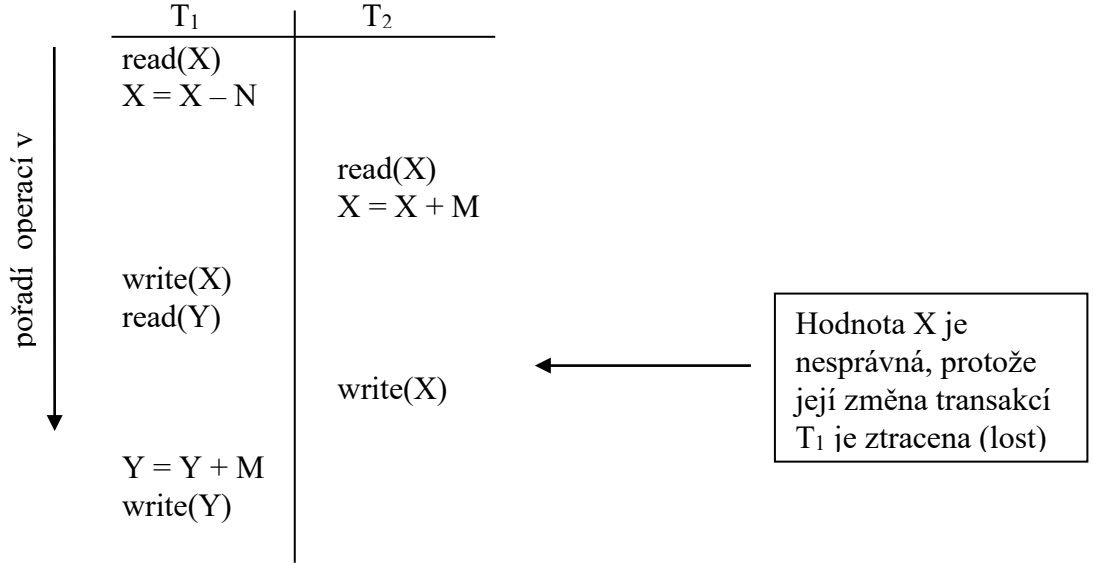
T_1:	read(x)	T2:	read(X)
	X = X-N		X = X+M
	write(X)		write(X)
	read(Y)		
	Y = Y+N		
	write(Y)		

Obrázek: Příklad transakcí T_1 a T_2

Problém zvaný „Lost Update“

Tento problém nastane v případě, kdy dvě transakce přistupující ke stejnému záznamu v databázi a po jejich ukončení je hodnota v záznamu nesprávná. Předpokládejme, že transakce T_1 a T_2 s operacemi jako na obrázku č. 2.4 jsou spuštěny současně a jejich operace jsou prováděny v pořadí jak je zobrazeno na obrázku č. 2.5. Potom hodnota X po ukončení transakcí bude nesprávná, protože transakce T_2 četla hodnotu X ještě před tím, než ji transakce T_1 změnila.

Například pokud původně bylo X = 50, N = 10 a M = 5, tak konečná hodnota X by měla být 45, ale protože se ztratí změny provedené transakcí T_1, bude výsledná hodnota X = 55.

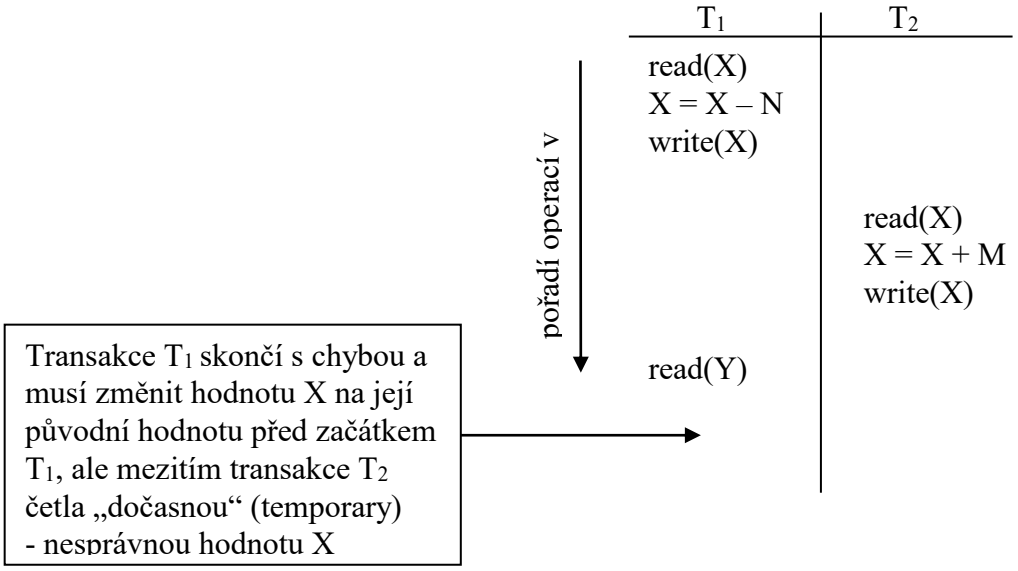


Obrázek Příklad návaznosti operací transakcí T_1 a T_2 pro problém Lost Update

Problém dočasné změny „Temporary Update“

Ten nastane v případě, kdy jedna transakce mění záznam v databázi a poté skončí chybou. Ke změněnému záznamu přistupuje jiná transakce dřív než dojde k vrácení původní hodnoty.

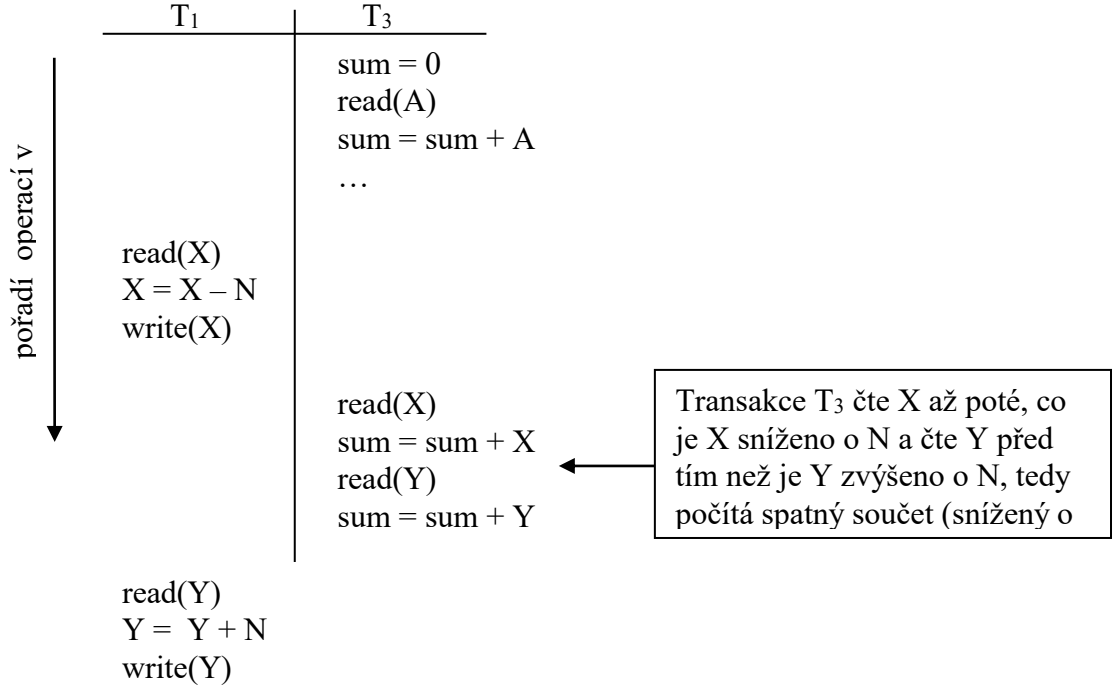
Obrázek ukazuje příklad, kdy transakce T_1 mění hodnotu X a skončí před dokončením. Tedy systém musí vrátit změny doposud provedené transakcí T_1, ale než se tak stane, transakce T2 přečte „dočasnou“ (temporary) hodnotu X, která nebyla permanentně uložena v databázi.



Obrázek Příklad návaznosti operací transakcí T_1 a T_2 pro problém Temporary Update

Poblém nesprávného součtu „Incorrect Summary“

Dalším problémem, který může nastat při nekontrolovaném spouštění souběžných transakcí je, pokud jedna transakce počítá součet hodnot a během tohoto výpočtu druhá transakce provádí změny některých těchto hodnot používaných ve výpočtu transakce první. Funkce může počítat s některými hodnotami před tím než jsou změněny a s některými až po změně,



tedy výsledek funkce není správný. Příklad takových transakcí je zobrazen na obrázku

Obrázek Příklad návaznosti operací transakcí T1 a T2 pro problém Incorrect Summary

Metody používané pro řízení běhu souběžných transakcí.

Chování transakcí v jednotlivých izolačních úrovních je značně ovlivněno způsobem jejich implementace. Znalost vnitřní práce databázového serveru tedy rozhodně není na škodu, protože vám umožní vytvářet aplikace, které s databázovým prostředím tvoří harmonický celek. Proto se na základní metody implementace transakcí podíváme podrobněji. Transakce lze implementovat různými způsoby, ale nejčastěji je používána implementace využívající zámků a transakčních protokolů. Nejdříve se podíváme, jak je v této architektuře řešena aktualizace dat.

Protože v praxi končí drtivá většina transakcí potvrzením změn (commit), je každá změna okamžitě uložena přímo do databáze a změněné nebo vymazané řádky (záznamy) jsou pro ostatní transakce označeny jako uzamčené proti zápisu. Zároveň je pro každý změněný, přidaný nebo vymazaný řádek vytvořen záznam o změně, který obsahuje identifikaci řádku a druh změny (v případě aktualizace i původní hodnoty změněných sloupců). Každý takový záznam je uložen do seznamu změn dané transakce uloženém v paměti a zároveň je uložen spolu s informací o transakci do souboru transakčního protokolu pro případ neočekávaného selhání systému.

Transakční protokol tvoří jeden nebo více souborů na disku a z bezpečnostních důvodů je dobré pro něj vyhradit prostor na jiném diskovém zařízení, než na kterém je uložena databáze. Rovněž je nutné pro protokol rezervovat dostatečně velký prostor, protože se neustále rozrůstá a u velmi exponovaných databází může přibývat i o několik megabajtů denně.

Informace o změnách jsou většinou uchovávány v paměti po celou dobu běhu transakce, takže při operaci rollback nebo selhání klienta lze databázi rychle uvést do původního stavu bez nutnosti zpracovávat soubory na disku. Na druhou stranu to ovšem zvyšuje nároky databázového serveru na paměť – zejména pokud je množství změn provedených jedinou transakcí příliš velké nebo je najednou zpracováváno velké množství transakcí provádějících změny.

Transakční protokol na disku je používán pro odstranění změn nepotvrzených transakcí pouze v případě selhání samotného databázového serveru. U některých databázových platforem lze s pomocí protokolu obnovit stav databáze k libovolnému časovému okamžiku, respektive k ukončené transakci. Výhodou transakčního protokolu je snadná realizace přírustkového zálohování (pokud obsahuje i data přidaných řádků), kdy postačí vytvořit záložní kopii původního stavu databáze a poté vytvářet záložní kopie transakčního protokolu. Nevýhodou je velká časová náročnost při rekonstrukci databáze po selhání systému nebo při obnově z takovéto zálohy.

Značný vliv na chování transakcí má rovněž použití zámků. Změněné nebo vymazané řádky jsou až do potvrzení pro ostatní transakce uzamčeny nejen pro zápis, ale často i pro pouhé čtení. Databáze totiž obsahuje již změněná data,a transakce s jinou izolační úrovní než Read Uncommitted je nesmějí číst. Trasankce s izolací Read Committed musí s čtením počkat až do potvrzení (nebo odvolání) změn, transakce s přísnější izolací mohou data číst pouze pokud budou změny odvolány (což je ovšem nepravděpodobné). Některé databázové platformy, jako je např. Oracle, proto umožňují transakcím s potřebou stabilního pohledu na data načíst

původní hodnoty řádku z transakčního protokolu. U platforem, které tuto schopnost nemají, jsou takové transakce blokovány a mohou skončit i chybovým hlášením o kolizi transakcí.

Stejně jako změna dat způsobí jejich uzamčení pro zápis i čtení, uzamkne i pouhé čtení dat trasankcemi s izolační úrovní Repeatable Read nebo Serializable přečtená data proti zápisu. V případě úrovně Serializable je proti zápisu typicky uzamčena celá tabulka.

Problém blokování čtení změněných dat nebo zápisu u přečtených dat může být i velmi závažný, pokud databázový server nerealizuje zámky na úrovni jednotlivých řádků (dnes již většina systémů), ale na úrovni databázových stránek (např. MS SQLServer 6.5) nebo celých tabulek (MySQL). V takovém případě mohou být uzamčena i taková data, která nebyla přímo změněna nebo přečtena.

Jak je vidět, použití zámků a transakčního protokolu má u většiny databázových platforem značný vliv na propustnost a jediným způsobem jak dosáhnout dobrých výsledků je pečlivý výběr vhodné izolační úrovně pro jednotlivé transakce.
V této kapitole popíšeme podrobněji některé základní metody pro řízení běhu souběžných transakcí. Cílem těchto metod je především zachování základních vlastností transakcí a zajištění serializovatelného spouštění.

Zamykání
Jedna z nejzákladnějších technik pro řízení běhu souběžných transakcí je založená na zamykání databázového záznamu.

Zámek je proměnná asociovaná s datovým záznamem, která vyjadřuje status záznamu s ohledem na možné operace, která mohou být nad záznamem použity. Pro zamykání je možné použít několik typů zámků. Nejprve uvedeme jednoduchý, ale omezeně použitelný typ jakým jsou binární zámky a dále sdílené a výhradní zámky, které poskytují obecnější použití.

Binární zámek

Tento typ zámku může nabývat dvě hodnoty 0 (false) a 1 (true). Hodnota 1 vyjadřuje, že asociovaný databázový záznam je *uzamčen*, tedy že data nemohou být zpřístupněna pro databázové operace, které tyto data vyžadují. Naopak hodnota 0 udává, že asociovaný záznam není uzamčen.

Na obrázku č. 2.8 je znázorněno, jak typicky vypadají operace *Zamkni* (Lock) a *Odemkni* (UnLock) záznam X v databázovém systému.

```
Zamkni(X):

START:   IF  „odemčeno“ X THEN „zamkni„ X
           ELSE
                BEGIN
                   čekej dokud („odemčeno“ X a transakční manažer nevzbudil
transakci)
                   jdi na START
                END

Odemkni(X):

        „odemkni“ X
        jestliže nějaká transakce čeká, tak ji vzbuď

Vysvětlivky:
        „odemčeno“ X          … test zda hodnota zámku asociovaného s X je 0
```

Obrázek Operace zamykání a odemykání záznamu pro binární zámek

Základní pravidla pro používání binárních zámků:

1. zamknutí záznamu X v transakci musí být před operacemi *čtení* nebo *zápisu* X
2. odemknutí X se provede až po provedení operací *čtení* a zápisu nad záznamem X
3. transakce nesmí požadovat zamknutí záznamu, pokud již tento záznam uzamknula
4. transakce nesmí požadovat odemknutí záznamu, pokud nemá opravdu záznam uzamčen

Binární zámky se snadno implementují jako přidaná položka *ZÁMEK* záznamu v databázi.

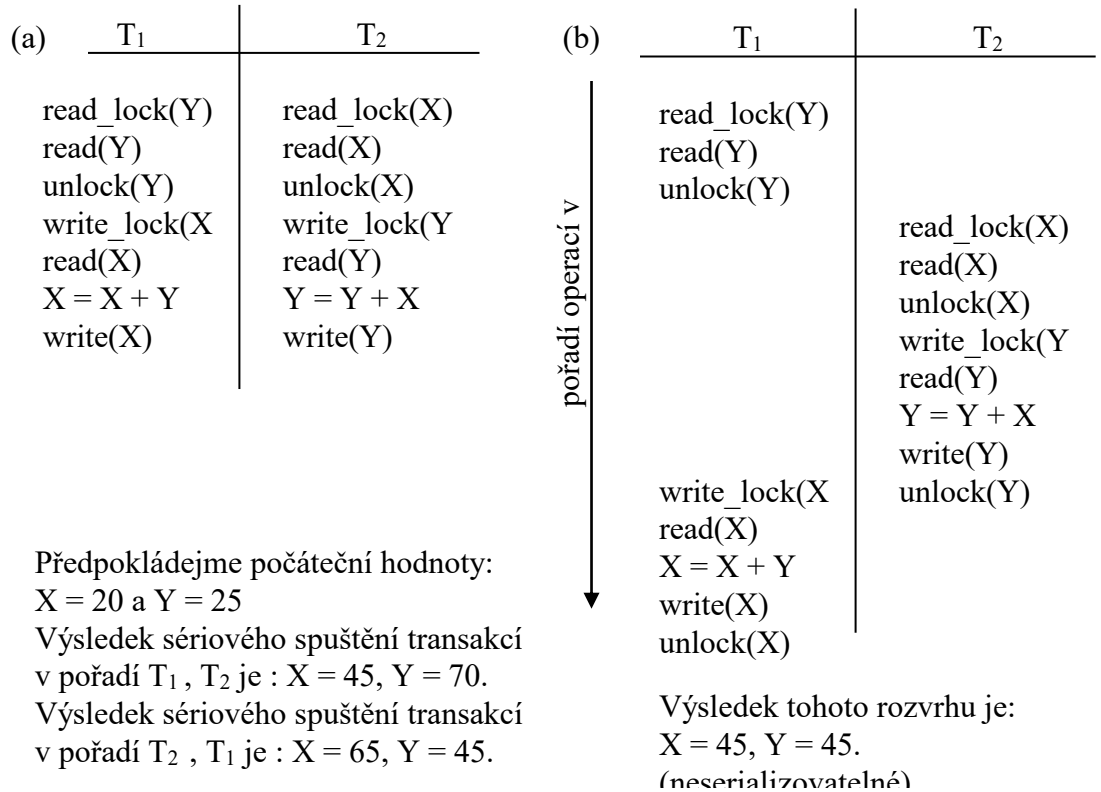
Sdílený a výhradní zámek (Shared and exclusive lock)

Chceme, aby transakce, které pouze čtou měly přístup pro čtení k záznamu a transakce, které vyžadují zápis, aby záznam výhradně uzamkly, tedy aby ostatní transakce nemohly zapisovat daný záznam. Tento typ zámku se nazývá „multiple_mode lock“ a nabývá tří hodnot: *zamčeno pro čtení* (read lock), *zamčeno pro zápis* (write lock) a *odemknuto* (unlock). Zámek pro čtení se nazývá *sdílený zámek*, ostatní transakce mohou číst záznam, a zámek pro zápis se nazývá *výhradní zámek*, pouze jedna transakce má mimořádný přístup k záznamu.

Základní pravidla pro používání sdílených a výhradních zámků:

1. transakce musí uzamknout záznam X pro čtení nebo pro zápis před jakoukoliv operací čtení záznamu X
2. transakce musí uzamknout záznam X pro zápis před jakoukoliv operací zápisu X
3. transakce musí odemknout záznam X až po provedení všech operací čtení či zápisu nad záznamem X
4. transakce nesmí požadovat zamknutí záznamu pro čtení pokud ho má již uzamčen pro čtení nebo zápis
5. transakce nesmí požadovat zamknutí záznamu pro zápis pokud ho má již uzamčen pro čtení nebo zápis
6. transakce nesní požadovat odemknutí záznamu X pokud opravdu nedrží zámek pro čtení či zápis nad X

Oba tyto typy zámků (binární, sdílený a výhradní) ovšem nezaručují serializovatelnost rozvrhu transakcí. Příkladem jsou transakce uvedené na obrázku č. 2.11a, kdy záznam Y v transakci T1 a záznam X v transakci T2 jsou odemčeny příliš brzy. To umožňuje souběžné spuštění transakcí tak, jak je znázorněno na obrázku č. 2.11b, které není serializovatelné a proto dává nesprávné výsledky. Abychom zajistili serializovatelnost, musíme použít rozšiřující protokol, který je založen na určení pozic operací zamykání a odemykání v každé transakci.



Obrázek Příklad neserializovatelného rozvrhu transakcí

Dvoufázový zamykací protokol (Two Phase Locking)

Řekneme, že transakce splňuje dvoufázový potvrzovací protokol, pokud všechny operace zamykání záznamů v transakci předchází první operaci odemykání. Tedy transakci je možné rozdělit do dvou fází:

a) **expanze (expanding phase) – v této fázi mohou být získávány nové zámky, ale ne uvolněny**
b) **uvolňování (shrinking phase) – zámky se pouze uvolňují, ale nezískávají nové**

nPokud každá transakce splňuje požadavky na dvoufázový zamykací protokol, je zajištěna serializovatelnost rozvrhu. Limitem tohoto protokolu je, že transakce musí zamykat záznam dřív než ho doopravdy potřebuje nebo nemůže uvolnit zámek na záznam X neboť později potřebuje zamknout záznam Y, nebo opačně, T musí zamknout záznam Y dříve než ho opravdu potřebuje, aby mohla uvolnit X. Tedy X musí být držena transakcí T dokud nejsou všechny potřebné záznamy zamčeny. Pouze potom může být X uvolněn a zatím musí každá transakce používající záznam X čekat.

Ačkoliv dvoufázový zamykací protokol zaručuje serializovatelnost, použitím zámků mohou nastat další problémy: *deadlock* a *livelock* (viz níže).

Na obrázku č. 2.12 je uvedeno jak se musí upravit transakce z příkladu na obrázku č. 2.11a, aby splňovaly dvoufázový zamykací protokol.

T_1	X a Y hodnoty pro iniciální X=20 a Y=25	T_2	X a Y hodnoty pro iniciální X=20 a Y=25
read_lock(Y)		read_lock(X)	
read(Y)	Y = 25	read(X)	X = 20
write_lock(X		write_lock(Y	
unlock(Y)		unlock(X)	
read(X)	X = 20	read(Y)	Y = 25
X = X + Y		Y = Y + X	Y = 45
write(X)	X = 45	write(Y)	

Obrázek Příklad upravení transakcí T_1 a T_2, aby splňovaly dvoufázové zamykání

Deadlock a Livelock

Při použití zámků mohou nastat tyto dva problémy:

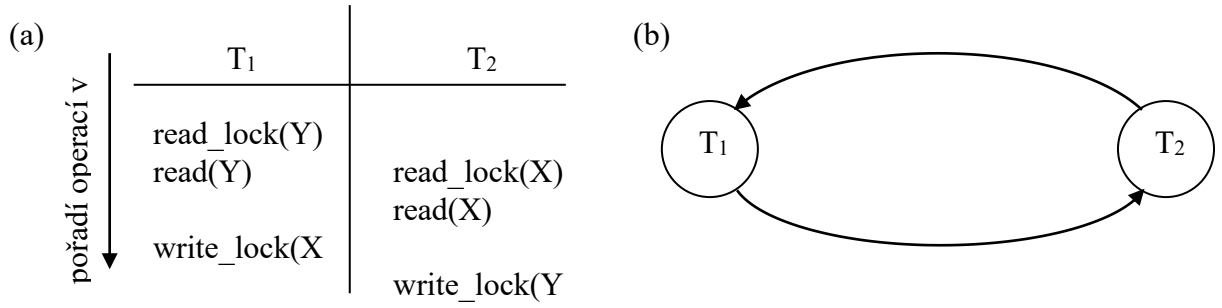
a) Deadlock – nastane pokud transakce čeká na uvolnění zámku, který drží druhá transakce a ta čeká na uvolnění zámku, který drží první transakce (i při cyklickém čekání více než dvou transakcí). Tento případ je zobrazen na obrázku, kde transakce T_1 čeká na uzamčení záznamu X, který má uzamčena transakce T_2 a T_2 čeká na záznam Y uzamčený transakcí T_1.

b) Livelock – nastane pokud transakce nemůže pokračovat po neurčitou dobu, zatím co jiné transakce v systému pokračují normálně.

Jedno z řešení, jak předcházet deadlocku je, že každá transakce se pokusí získat všechny zámky napřed a pokud se jí to nepodaří, všechny získané uvolní a zkusí to po nějakém čase znovu. Dalším řešením je uspořádat všechny záznamy v databázi a následně zajistit, aby každá transakce jí používané záznamy zamykala v pořadí odpovídající danému uspořádání v databázi.

Druhým typem přístupu je detekce deadlocku. Necháme transakce běžet a až poté kontrolujeme, zda nedošlo k deadlocku. To lze jednoduše zjistit například konstrukcí „grafu čekání“ (wait for graph). Uzly tohoto grafu jsou právě běžící transakce a hrana z uzlu T1 do uzlu T2 znamená, že transakce T1 čeká na zámek záznamu, který vlastní transakce T2. Potom pro detekci deadlocku stačí detekovat cykly v grafu viz obrázek. Pokud nalezneme cyklus, některou z transakcí v cyklu ukončíme (uvolní se zámky) a změny provedené touto transakcí vrátíme.

Livelock může nastat pokud čekací schéma pro uzamykání je neférové, dává prioritu nějakým transakcím na úkor druhým. Standardním řešením livelocku je schéma používající kritérium pro přidělování zámků v pořadí: „kdo první přijde, je první obsloužen“ (first come first serve). Jiná schémata povolují některým transakcím, aby měly vyšší prioritu než ostatní, ale zvyšuje se priorita transakce, která dlouho čeká, dokud eventuelně nedosáhne nejvyšší priority.



Obrázek a) Příklad *deadlocku* b) odpovídající čekací graf (Wait-For-Graph)

Časová razítka (Time Stamps)

Další technikou pro řízení souběžného běhu transakcí a zajištění serializovatelnosti transakcí je používání časových razítek.

Časové razítko je jedinečný identifikátor, který je přiřazen transakci. Typicky je časovým razítkem hodnota, která odpovídá pořadí spuštění transakce v systému, nebo čas nastartování transakce. Při použití této metody nemůže nastat *deadlock*, protože se nepoužívá zamykání.

Uvedeme dvě techniky používání časových razítek *timestamp ordering* a *multiversion concurrency control.*

Timestamp ordering (uspořádání podle časových razítek)

Tato metoda je založena na uspořádání transakcí podle časových razítek. Rozvrh, ve kterém transakce se účastní, je pak serializovatelný a ekvivalentní sériový rozvrh má transakce uspořádány podle časových razítek. Na rozdíl od dvoufázového zamykání, kde je rozvrh serializovatelný tím, že je ekvivalentní **nějakému** sériovému rozvrhu, který povoluje zamykání, je u uspořádání časových razítek rozvrh ekvivalentní **konkrétnímu** sériovému uspořádání odpovídajícímu uspořádání podle časových razítek.

Abychom dosáhli co největší konkurence, spouštíme transakce zcela volně. Ovšem musíme zajistit, aby pořadí přístupu ke každému záznamu, ke kterému přistupuje více jak jedna transakce v rozvrhu, nebylo v rozporu se serializovatelností rozvrhu. Abychom toto zajistili, musíme ke každému záznamu v databázi přiřadit dvě časová razítka (TS):

1. **read_TS(X) – obsahuje nejvyšší hodnotu časového razítka ze všech razítek transakcí, které úspěšně četly záznam X.**
2. **write_TS(X) - obsahuje nejvyšší hodnotu časového razítka ze všech razítek transakcí, které úspěšně zapsaly záznam X.**

Pokud se například transakce T snaží číst či zapsat záznam X, stačí porovnat příslušné hodnoty časových razítek, abychom zjistili, zda pořadí spuštění transakce není v rozporu s ekvivalentním sériovým rozvrhem. Pokud je T v rozporu, musí být ukončena a všechny změny, které měla na databázové záznamy musí být vráceny zpět. Později je tato transakce znovu spuštěna s novým časovým razítkem. Poznamenejme, že pokud transakce T_1 použila hodnotu, kterou zapsala T, musí být také ukončena a její změny vráceny zpět (rollback). Stejně tak jakákoliv transakce T_2 používající hodnotu zapsanou T_1. Tento efekt je znám jako kaskádový rollback (cascading rollback) a je jedním z problémů spojených s používáním uspořádání pomocí časových razítek.

Algoritmus pro souběžný běh transakcí musí zjistit, jestli uspořádání transakcí podle časových razítek je v rozporu s následujícími dvěma případy:

1. **transakce T zapisuje záznam X (obsahuje operaci zápisu):**

a) jestliže read_TS(X) > TS(T) tak musíme ukončit T a vrátit změny provedené T zpět. To je v případě, kdy jiná transakce s vyšším časovým razítkem = pozdější v uspořádání podle časových razítek opravdu četla hodnotu X před tím, než T změnila X, tedy v rozporu s uspořádáním.

b) jestliže write_TS(X) > TS(T) tak nepovolíme operaci zápisu a pokračujeme v transakci T, protože jiná, novější transakce již tuto hodnotu zapsala a nemůžeme ji přepsat. Jinak by došlo ke ztrátě předcházející správné hodnoty.

c) Jestliže žádná z podmínek a) i b) nenastane, tak provedeme operaci zápisu X a nastavíme časové razítko zápisu pro X na hodnotu TS(T).

2. transakce T čte záznam X (obsahuje operaci čtení):

a) jestliže write_TS(X) > TS(T), tak musíme ukončit T a vrátit změny provedené T zpět. To protože nějaká jiná transakce s větším časovým razítkem než TS(T) (a tedy v uspořádání časových razítek až po T) skutečně zapsala hodnotu X. A to před tím, než transakce T měla šanci číst X, proto došlo k porušení uspořádání.

b) jestliže write_TS(X) <= TS(T), tak spustíme operaci čtení X a nastavíme hodnotu časového razítka čtení pro X na maximální hodnotu z read_TS(X) a TS(T).

Protokol uspořádání časových razítek, stejně jako dvoufázový zamykací protokol zaručují serializovatelnost rozvrhů, i když některé z rozvrhů jsou povoleny jedním protokolem a ne druhým a naopak. Při použití uspořádání podle časových razítek nemůže nastat *deadlock*, ale může nastat stálým ukončováním a restartováním *livelock*. Tento problém je znám jako *problém cyklického restartu* (cyclic restart problem).

Multiversion concurrency control

Dalším protokolem pro řízení konkurenčního běhu transakcí, který může také používat koncept časových razítek, udržuje staré hodnoty dat, když jsou tato data měněna. Tento způsob je znám jako Multiversion conciurency control., protože je udržováno několik verzí hodnot dat. Když transakce požaduje přístup k záznamu, časové razítko transakce je porovnáno s časovými razítky různých verzí záznamu. Příslušná hodnota je vybrána, aby byla udržena serializovatelnost právě spuštěného rozvrhu, pokud je možná.

Je zde několik navrhovaných schémat Multiversion concurency control. Zaměříme se na jeden z nich, jako na příklad. V této technice jsou systémem udržovány verze X1, X2, …, Xn každého záznamu X v databázi. Pro každou verzi je uchována jeho hodnota a dvě časová razítka:

1. read_TS(Xi) – časové razítko čtení Xi, určující nejposlednější transakci, která úspěšně četla verzi Xi (nejvyšší hodnota ze všech takových transakcí)
2. write_TS(Xi) – časové razítko zápisu Xi, obsahující časové razítko transakce, která tuto hodnotu zapsala

V tomto schématu kdykoliv transakce T zapíše hodnotu X, vytvoří se nová verze Xn+1 záznamu X s časovými razítky obsahujícími hodnotu TS(T). Podobně pokud transakce T může číst číst verzi záznamu Xi, tak hodnota časového razítka read_TS(Xi) je nastavena na větší z hodnot TS(T) a read_TS(Xi).

Abychom zajistili serializovatelnost, použijeme následující dvě pravidla na kontrolu čtení a zápisu dat:

1. pokud transakce T zapisuje záznam X: verze záznamu Xi má nejvyšší hodnotu časového razítka zápisu (ze všech verzí X), které je menší nebo rovné TS(T) a zároveň

TS(T) < read_TS(Xi), potom ukončíme transakci T. Jinak vytvoříme novou verzi Xj, která bude mít read_TS(T) = write_TS(T) = TS(T).

2. pokud transakce čte záznam X: najdeme verzi tohoto záznamu Xi, která má nejvyšší write_TS(Xi) ze všech verzí X, a které je menší nebo rovno TS(T). Potom vrátíme hodnotu Xi transakci a nastavíme read_TS(Xi) na hodnotu větší z read_TS(Xi) a TS(T).

V bodu 1 je transakce T ukončena, jestliže se pokouší zapsat verzi záznamu X (podle bodu 1), který byl přečten jinou pozdější transakcí s časovým razítkem rovným read_TS(Xi).

Problémy při řízení souběžných transakcí

Při řízení souběžných transakcí v distribuovaných databázích vznikají další problémy, které nejsou v centralizovaném prostředí. Některé z těchto problémů jsou následující:

- **počítání s násobnými kopiemi dat. Řízení souběžných transakcí je odpovědné za udržení konzistence těchto kopií.**
- **výpadek některého z uzlů v síti. Databáze by měla fungovat i v případě, kdy některé z uzlů vypadnou a následně znovu obnovit konzistenci dat ve všech uzlech sítě.**
- **chyba komunikační sítě. Databáze musí být připravena na výpadek komunikační sítě, kdy dochází k rozdělení celé sítě na jednotlivé samostatně pracující podsítě.**
- **distribuované potvrzování transakcí. Problém vznikající při potvrzování transakce, která přistupuje k datům uloženým v různých uzlech sítě. Tento problém se často řeší pomocí *Dvoufázového potvrzovacího protokolu* (viz níže).**
- **distribuovaný *daedlock*, nastane v případě cyklického čekání mezi několika uzly v síti.**

Pro řízení souběžných transakcí v distribuovaných databázích lze použít metody používané v centralizovaných databázích s pomocí dalšího protokolu, který koordinuje spolupráci jednotlivých uzlů.

3.1.2 Protokoly kontroly souběžnosti

Protokoly kontroly souběžnosti se používají k omezení spouštění souběžných transakcí v centralizovaném databázovém systému a v distribuovaném pouze k serializovatelnému spouštění.

Protokoly kontroly souběžnosti se dělí na dvě kategorie:

- **pesimistické**
- **optimistické**

Pesimistické protokoly

U těchto protokolů dochází ke kontrole před tím, než je databázová operace provedena. Tedy předchází nekonzistencím zamítnutím potenciálních neserializovatelných spouštění a zajištěním, že výsledek potvrzených transakcí musí být zaznamenán do databáze nebo anulován.

Příkladem tohoto protokolu je komerčně široce používaný *Dvoufázový uzamykací protokol* a dále *Uspořádání podle časových razítek*.

Optimistické protokoly

U těchto protokolů nedochází ke kontrolám v době běhu transakce a tedy dovolují neserializovatené spouštění. Prováděné změny nejsou hned aplikovány přímo v databázi, ale

v lokání paměti. Na konci běhu transakce validační fáze zkontroluje, zda změny prováděné transakcí nejsou v rozporu se serializovatlností. Pokud ne, transakce se potvrdí a změny se promítnou do databáze, jinak je ukončena a spuštěna později. Tedy transakce s detekovanými anomáliemi jsou ukončeny během validační fáze před tím, než je výsledek transakce zviditelněn.

Optimistické protokoly pro řízení souběžných transakcí jsou rozděleny do tří fází:

1. **Fáze čtení: transakce může číst záznamy z databáze, ale změny jsou prováděny pouze do lokálních kopií záznamů udržovaných v pracovním prostoru transakce.**
2. **Validační fáze: provádí se testování, aby byla zajištěna serializovatelnost, pokud by transakce změny promítla do databáze.**
3. **Fáze zápisu: pokud je validační fáze úspěšná, transakce provede změny do databáze, jinak jsou změny zrušeny a transakce je spuštěny znovu.**

Ideou optimistických protokolů je provádět všechno testování najednou, aby transakce běžela s minimální režií až do validační fáze. Pokud je malá závislost mezi transakcemi, je hodně transakcí validováno úspěšně, v opačném případě je hodně transakcí zamítnuto a znovu restartováno. V prostředí, kde je velká závislost mezi transakcemi, optimistické protokoly nejsou vhodné. Tyto techniky se nazývají optimistické, protože předpokládají malou závislost mezi transakcemi, a tedy že není nutné provádět testování během provádění transakce.

Příkladem je *Certifikace*, která provádí validaci v čase potvrzování transakce.

3.1.3 Kontrola atomičnosti

Protokol kontroly atomičnosti zaručuje, že každá transakce je atomická (buď byly všechny operace transakce provedeny nebo žádná). Nejpoužívanějším takovým protokolem je *Dvoufázový potvrzovací protokol*.

Transakce je považována za potvrzenou, pokud koordinátor a všichni účastníci transakčního zpracování souhlasí s jejím potvrzením (nejsou na žádné ze zúčastněných stran konflikty).

Nicméně nějaké selhání koordinátora nebo komunikace mezi ním a ostatními účastníky (dotčenými uzly) může přinutit Dvoufázový potvrzovací protokol, dokonce s pomocí *spolupracujícího ukončovacího protokolu*, zablokovat transakci dokud není chyba odstraněna. K vyřešení prvního případu, selhání koordinátora, byl navržen *Třífázový potvrzovací protokol*. Chyby vedou k používání bezpečnostních záznamů, které zaznamenávají každou akci. Tyto záznamy jsou používány *obnovovacími protokoly* (recovery protocols) pro navrácení databáze do konzistentního stavu.

Život transakce

Život transakce začíná jejím zahájením, při kterém jsou specifikovány základní parametry které dále vymezují její činnost a chování. Mezi tyto parametry patří:

Příslušnost k uživateli a databázi

Pokud je transakce iniciována klientem, podléhá aktuálnímu připojení klienta k serveru a pracovní databázi. Některé servery nedovolují nastartovat více jak jednu transakci v rámci jednoho připojení, případně nedovolují zahrnout do jediné transakce operace nad více jak jednou databází.

Některé servery dovolují programům prováděným na serveru (uživatelské funkce, uložené procedury a spouště) zahajovat vlastní transakce. Kontext transakce je pak dán buď databází a

uživatelem, který daný kód vyvolal, případně je dovoleno aby si kód na serveru vytvořil své vlastní spojení k serveru a databázi a určil tak kontext pro novou transakci.

Izolační úroveň
Definuje způsob interakce s ostatními transakcemi.

Způsob řešení konfliktů
Pokud dojde ke kolizi mezi transakcemi, může transakce buď čekat na výsledek blokující transakce, nebo okamžitě skončit chybou. V praxi je pravděpodobnost odvolání transakce velmi malá a čekání na její výsledek tedy většinou stejně končí chybou kolize transakcí. Čekání na výsledek rovněž zvyšuje pravděpodobnost vzájemného zablokování transakcí (deadlock).

Pracovní podmínky
Některé databázové platformy umožňují blíže specifikovat budoucí potřeby transakce, což serveru umožní dopředu vytvořit transakci vhodné podmínky pro práci, případně zvolit jiný, optimálnější postup. Do této kategorie např. spadá možnost definovat transakce pouze pro čtení, případně dopředu specifikovat potřebný režim přístupu k jednotlivým tabulkám. Například transakce pouze pro čtení v izolaci Read Committed mohou být chápány jako předem potvrzené a režie serveru na jejich práci je výrazně menší a blokování ostatních transakcí je menší nebo i nulové.

Zahájení transakce je spojeno s vytvořením kontextu a alokací zdrojů serveru pro její práci, a je tedy vhodné ji zahájit až v okamžiku skutečné potřeby.

S každým příkazem vykonaným v rámci transakce narůstá spotřeba zdrojů serveru a zvyšuje se pravděpodobnost kolize s dalšími transakcemi. Proto je obecně doporučováno, aby každá transakce zahrnovala pouze nezbytně nutné množství operací a trvala co nejkratší dobu. U serverů s multigenerační architekturou nebo u transakcí s úrovní izolace Read Uncommitted a Read Committed jsou kritické pouze oprace vložení, změny nebo výmazu dat, které podle architektury serveru blokují řádek, databázovou stránku nebo celou tabulku. U serverů používající zámky a transakční protokol nebo u transakcí s izolační úrovní Serializable jsou kritické i operace čtení dat, protože rovněž blokují možnost zápisu dat z ostatních transakcí.

Většina transakcí je ukončena potvrzením (operace commit), a práce databázových systémů je pro tento scénář optimalizována. U serverů s MGA představuje potvrzení transakce pouhou změnu příznaku o jejím stavu v tabulce transakcí, a uvolnění paměťových struktur alokovaných v průběhu její práce jako jsou otevřené kurzory a pracovní soubory. U serverů používajících zámky a transakční protokol je navíc ještě nutné uvolnit všechny zámky, a zapsat do protokolu záznam o úspěšném ukončení transakce.

Pokud byla transakce prováděna nad více databázemi, postupuje server při jejím potvrzování podle tzv. dvoufázového potvrzovacího protokolu. V první fázi je realizováno potvrzení změn v jednotlivých databázích bez uvolnění datových struktur transakce. Teprve po úspěšném provedení první fáze na všech databázích je provedena druhá fáze - skutečné uvolnění všech struktur spojených s transakcí a její formální ukončení. V průběhu první fáze se transakce nachází ve speciálním přechodném stavu, a pokud nedojde k jejímu úspěšnému zakončení na všech databázích (např. z důvodu přerušení spojení se vzdálenou databází), může transakce v

tomto stavu "zamrznout" (tzv. limbo transakce) a může být zapotřebí intervence administrátora.

Některé operace nad databází nejsou provedeny ihned po vyžádání, ale jsou serverem pouze zaznamenány a skutečné provedení je odloženo až na dobu potvrzení transakce. Jedná se většínou o operace měnící strukturu databáze (SQL příkazy z rodiny CREATE, ALTER nebo DROP) nebo některé speciální kontroly dat. U serveru InterBase/Firebird jsou například události (events) rozesílány klientským aplikacím právě až při potvrzení transakce.

Odvolání transakce (Rollback) je vždy spojeno se značnou zátěží pro server přímo úměrnou množství změn které transakce provedla. Proto by aktivní odvolání transakce z vůle klienta mělo být používáno pouze jako záchrana v nouzi, a nikoliv jako běžný prvek práce s databází. U serverů se zámky a transakčním protokolem je při odvolání transakce nezbytné obnovit původní stav databáze, u serverů s MGA tato nutnost odpadá a rollback je proto rychlý a relativně levný (odstranění nepotřebných verzí řádků je odloženo na později). Pokud transakce neprovedla žádné změny, je většina serverů schopna převést rollback na mnohem "levnější" operaci commit. Protože se riziku odvolání transakce nelze zcela vyhnout (např. z důvodu pádu klienta), je vhodné držet množství změn provedených v rámci jediné transakce (především u dávkových změn a importů) v rozumných mezích.

Většina databázových platforem umožňuje při ukončení transakce ihned zahájit novou transakci se stejnými parametry, a přenést kontext a zdroje původní transakce do této nově zahájené transakce (tzv. commit/rollback retaining). Hlavním důvodem je zachování otevřených kurzorů a předpřipravených příkazů při současném potvrzení (a tudíž zviditelnění pro ostatní) doposud provedených změn. Zachování kontextu ovšem znamená i zachování všech zámků a viditelnosti dat, což je obvzlášť důležité u izolační úrovně Repeatable Read a Serializable. Všechny doposud přečtené řádky (nebo tabulky) jsou stále zamčené proti zápisu, změny provedené a potvrzené jinými transakcemi po zahájení prvotní transakce stále nejsou viditelné. Dlouhý řetěz transakcí se stejným kontextem značně zatěžuje zdroje serveru, snižuje propustnost a může blokovat některé "ozdravné" činnosti serveru. Z výše uvedených důvodů je nutné tuto možnost využívat s rozvahou, a čas od času přerušit řetěz transakcí s přenášením kontextu plnohodnotným ukončením transakce.

3.2 Kriteria návrhu distribuovaných databází

Důležitými kriterii pří návrhu a klasifikaci distribuovaných databázových systémů jsou stupeň centralizace, modely dat, těsnost spojení a rozmístění dat.

3.2.1 Stupeň centralizace

Stupeň centralizace distribuovaných databází je jedním z nejdůležitějších kritérií. Vypovídá o tom, do jaké míry je řízení databáze soustředěno na jedno místo. Existuje mnoho možností řešení stupně centralizace distribuovaných databází. Pro názornost zde uvedeme dva krajní případy, a to *Centralizované DDBS* a *Decentralizované DDBS*.

U centralizovaných DDBS je řízení soustředěné na jeden centrální uzel (počítač) v síti. Jsou zde i kopie všech dat v databázi, které se zde centrálně řídí přístup a provádění změn ve struktuře distribuované databáze, synchronizace transakcí v DDBS a všechny další činnosti systému. Výhodou takovýchto systémů je relativní jednoduchost řízení všech činností. Správa systému má stálý přehled o aktuálních stavech všech složek systému a může kdykoliv zasáhnout podle potřeby. Nevýhodou je vysoká zátěž komunikačních prostředků, každá změna v datech musí být řízena a povolena z centra. Dále musí být dostatečně zabezpečeno centrum proti výpadku či poruše, protože by to znamenalo výpadek celého systému jako celku.

U decentralizovaných systémů nemá žádný uzel zvláštní privilegia. Všechny uzly mají stejné informace týkající se DDBS a také stejnou zodpovědnost za zachování integrity v systému. Algoritmy zajišťující integritu dat bez centrálního řízení jsou složitější než u centralizovaných systémů. Avšak decentralizované systémy oproti centralizovaným vynikají svojí robustností, výpadek jakéhokoliv uzlu v síti nemá za následek výpadek celého systému, ale pouze může znamenat ztrátu dat které daný uzel spravuje. Pomocí duplikování dat v jiných uzlech lze tuto možnou ztrátu zmírnit.

3.2.2 Modely dat

Jednotlivé lokální databáze mohou být organizované s použitím jednotného nebo různých datových modelů. Použití jednotného datového modelu značně snižuje složitost architektury DDBS.

3.2.3 Těsnost spojení

V nedistribuovaných systémech běžný a přirozený požadavek, aby každá aktualizace dat byla okamžitě provedena a tyto změny hned zpřístupněny ostatním uživatelům, se v distribuovaných systémech zajišťuje podstatně hůře a je příčinou složitosti programového vybavení DDBS a i zvýšení nákladů na provoz systému. Ale existuje mnoho aplikačních oblastí, kde lze z tohoto požadavku alespoň částečně upustit.

Jako příklad uvedeme podnik s výrobou, centrálním skladem a s se sítí prodejen. Každá prodejna uchovává informace o obratech a stavech výrobků na skladě. Tyto lokální databáze jsou součástí celopodnikového systému a obsahují kopie potřebných dat. Aby se zjednodušilo řešení a provoz systému, v průběhu dne se aktualizace dat pouze zaznamenávají a vykonávají se jednorázově v době, kdy jsou počítače a komunikační linky méně vytížené, např. v nočních hodinách. Samozřejmě se při návrhu systému muselo počítat s tím, že v průběhu dne systém pracuje s dočasně nepřesnými daty.

Takovéto DDBS se někdy nazývají volně spojené systémy.

Umožněním dočasné, řízené nekonzistence systému se dají podstatně ovlivnit provozní charakteristiky systému a zjednodušit jejich návrh.

3.2.4 Rozmístění dat v DDBS

Vzhledem na omezení komunikačních linek (kapacitní i nákladové) je otázka geografického rozmístění dat velmi důležitá.

Data by se měla rozmisťovat co nejblíže k místu jejich vzniku nebo využívání. Pro urychlení přístupu k datům a snížení komunikační zátěže se některá data používají v několika aktuálních kopiích na různých uzlech, kde jsou často používána. Tím se zajistí, že potřebná data jsou ihned k dispozici bez nutnosti jejich přenosu po síti.

Toto urychlení výběru je na úkor stížené aktualizace dat, protože všechny kopie musí být identické. Proto je zapotřebí aktualizaci dat provést na všech kopiích a zabránit přístupu k těmto datům během jejich aktualizace. Na druhé straně replikace dat zvyšuje odolnost DDBS proti poruchám. Na jiných uzlech se duplikují především data, ke kterým je potřebný rychlý přístup, která nejsou často aktualizované, respektive která jsou mimořádně důležitá.

Klasifikace distribuovaných databázových systémů

Distribuované databázové systémy je možné klasifikovat podle různých kritérií. Nejzákladnějším kriteriem je vlastní distribuce dat.

3.2.5 Typy distribucí

Je-li několik databází spojených dohromady, mluvíme o „nepravé distribuci“, má-li několik databází společně organizovaná data, mluvíme o „pravé distribuci“. Mohou existovat i smíšené formy distribuce.

3.2.6 Organizace dat

Klasifikace podle rovnocennosti uzlů. V prvním případě, kdy jsou si uzly rovnocenné, se vyskytuje celé programové vybavení na obou uzlech. V druhém případě, kdy si uzly nejsou rovnocenní, hraje jeden uzel vedoucí roli a obsahuje komponenty programového vybavení i dat, které nejsou na žádném z ostatních uzlů.

Druhý typ představují databázové systémy typu stanice – server, kde na stanicích jsou lokální SŘBD (systémy řízení báze dat), které využívají data ze serveru a na serveru je globální SŘBD. Na stanicích jsou data ze serveru, která uživatel využívá ke svým transakcím.

3.2.7 Distribuce relací

V obecné architektuře distribuovaných SŘBD se objevují dva základní moduly:

- modul řízení transakcí
- modul řízení dat.

Modul řízení transakcí (MŘT) je částí distribuovaného systému řízení báze dat (SŘBD), který se zabývá aspekty vlastní distribuce dat. Tedy za prvé lokalizací všech případů redundance dat a za druhé zpracováním uživatelských transakcí.

To ve skutečnosti znamená:

1. **provést analýzu požadavků transakce**
2. **rozdělit transakci na jednotlivé podtransakce odpovídající fyzickým částem distribuované databáze a realizovat je jako provedení paralelních programů ve spolupráci s několika moduly řízení dat**
3. **z jednotlivých mezivýsledků sestavit konečný výsledek a ten pak předat uživateli**

Modul řízení dat (MŘD) je ta část distribuovaného SŘBD, která pro lokální databáze plní běžné funkce SŘBD.

Distribuovaný systém, který má ve všech uzlech stejný modul řízení dat, se nazývá *homogenní*, v opačném případě *heterogenní*.

Co znamená replikace dat

Replikace znamená zkopírování dat z databáze do více (geograficky vzdálených) míst za účelem podpory distribuovaných aplikací. Kopie dat, resp. Jejich odpovídající datové struktuře se nazývají *repliky*.

V roce 1995 se objevily u známých výrobců SŘBD tzv. *replikační servery*, které se staly zatím nejefektivnějším přístupem k implementaci distribuovaných databází. Firma Sybase přišla se svým replikačním serverem z rodiny SYBASE System 10 jako novou technologií, kerá optimalizuje sdílení dat na úrovni rozlehlého podniku. Tento software zajišťuje řízení aktualizace kopií pro jistou část sítě, ve které se replikovaná data vyskytují. Obecně může být několik replikačních serverů s rozdělenými pravomocemi, které spolu komunikují. Replikační server firmy Sybase může být použit ve spojení s jinými servery založenými na jiných ať relačních či nerelačních SŘBD, takže ho lze použít i v heterogenním DDBS [8].

Heterogennost prostředí je pro replikace speciální problém, který komplikuje vlastní replikační mechanizmy. Přidělává další časovou režii pro potřebné transformace dat apod. Ale práce v heterogenním prostředí je nutností, neboť typické velké podniky využívají několik různých databázových produktů.

databázový systém založený na replikaci musí zajistit následující :

- **dopravit replikovaná data do místa, kde jsou potřebná**
- **automaticky synchronizovat všechny repliky po chybě, která nastala nad některou z replik**
- **provést změny do všech replik v co nejkratším časovém intervalu po jejich provedení nad některou z replik**
- ***replikovat data z (do) heterogenních databázových systémů, které běží na platformách od různých výrobců**

Technologii replikací na úrovni transakčního zpracování lze chápat jako ... dvoufázového potvrzovacího protokolu (2PC), který v DDBS při 50 a více různých místech s distribuovanými daty vede k některým problémům. Jedná se především o problém dostupnosti zdrojů. Řeší se, co dělat v případě je-li jediná replika nedostupná, nebo na ni čeká velké množství požadavků. Dalším problémem je pomalost provozu sítě, který může nastat při velkých přenosech dat. Třetím problémem je různorodost úloh v distribuovaném prostředí, kde vyladění jednoho typu vede k problémům s druhým a naopak (viz níže). Replikační přístup uvolňuje těsná spojení míst, která jsou typická pro klasické DDBS.

Replikační přístup pokrývá v nejhlubším dělení dvě třídy aplikací. Prvním z nich jsou tzv. velkosklady dat (Data Warehouses) či systémy podporující rozhodování (Decision Support Systems - DSS), kdy v místě uživatele jsou vyžadována data konzistentní v jistém minulém časovém okamžiku nebo i více okamžicích (historická data). Tento přístup lze nejspíše použít v aplikacích s kopiemi pouze pro čtení (read-only). Příkladem softwaru založeného na tomto přístupu je např. Information Warehouse firmy IBM.

Naopak replikace dat vhodná pro použití v ..., k On-line transakčnímu zpracování směřují k extrémálnímu přístupu DDBS (globální schéma databáze + 2PC, tj. synchronně aktualizované repliky) nebo k mírnější variantě s asynchronně aktualizovanými replikami dat. V tomto případě asynchronnost znamená, že data nejsou aktualizované v rámci aplikační transakce, ale v co nejkratším časovém úseku (nejde o periodické či odkládané aktualizace).

Použitelnost jednotlivých přístupů závisí na požadavcích na systém a globální situaci v síti. Synchronní (On-line) přístup se používá tam, kde jsou zapotřebí nejaktuálnější data (úplná konzistence). Musí však být k dispozici rychlá síť a vysoká dostupnost serverů. Asynchronní systémy by neměly být na těchto faktorech příliš závislé.

3.2.8 Typy replik

Replikovat lze celou databázi nebo pouze její část. Pokud vznikne replika jako výsledek dotazu nad databází, rozlišuje se *read-only* replika, resp. *momentka* (snapshot). Momentka je výsledkem dotazu nad relacemi jednoho (několika) míst, do které se periodicky promítají změny řízené z jednoho místa. Jednotlivá místa mohou momentku pouze číst.

Dotaz, který definuje momentku není zcela obecný. Z hlediska terminologie DDBS lze využít vertikální i horizontální fragmentace. Nejčastěji se používají momentky pouze nad jednou relací databáze, i když by bylo vhodnější připustit spojení tabulek (přístup ORACLE 7) a tím tak předem připravit pro uživatele data vyhovující např. preferovaným dotazům.

Replika může být umístěna na stejném serveru jako primární kopie, nebo na jiném serveru v lokální síti či dokonce na vzdáleném serveru sítě.

3.2.9 Formy replikace

Replikace lze provádět různými způsoby, z nichž nejrozšířenější jsou následující dva typy (obrázek č. 3.1):

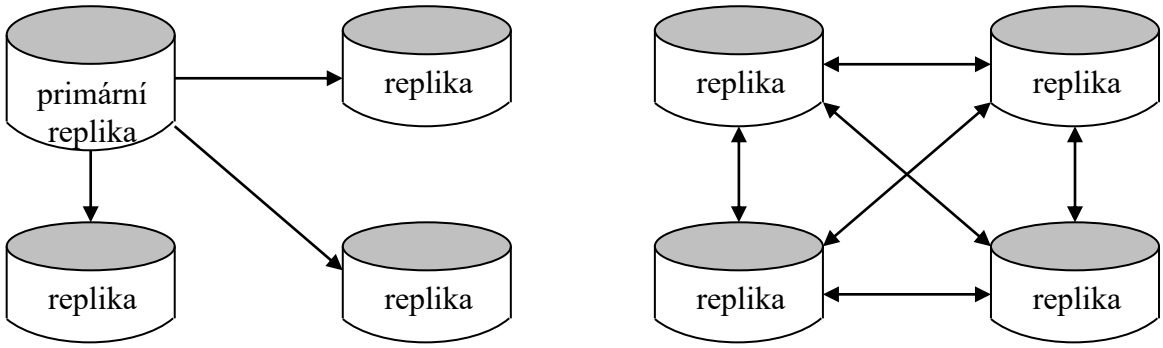
- **Jednosměrné systémy – data aktualizuje pouze jeden účastník**
- **Obousměrné (symetrické) systémy – aktualizovat data může několik účastníků**

Jednosměrné systémy

Možnost aktualizovat data mají pouze uživatelé na jednom místě. Provoz může vést k běžnému zpracování paralelních transakcí, kde hraje roli uspořádatelnost. Replikace je realizována pomocí read-only replik nebo momentek, tedy v tomto případě nenastávají problémy s konflikty, které mohou zapříčinit nezávislé aktualizace týchž dat na různých replikách. Mohou však nastat problémy s výkonem systému, jsou-li pro aktualizace replik nutné nějaké transformace, např. v heterogenním prostředí. Pro urychlení aktualizace momentek je možně promítnout pouze provedené změny namísto celé momentky.

Obousměrné systémy

U těchto systémů, kdy aktualizovat repliky mohou uživatelé na několika místech, je situace podobná běžnému On-line transakčnímu zpracování, pouze s tím rozdílem, že u asynchronního replikačního systému není k dispozici žádný blokovací (zamykací) mechanizmus. Tedy vznikají konflikty, které je nutné řešit.



Obrázek Jednosměrný a obousměrný replikační systém

Z hlediska organizace aktualizace replik se rozlišuje následující dva přístupy:

- **peer-to-peer – aktualizace repliky může být provedena v libovolném místě a poté je propagována do dalších míst. V tomto případě se využívají různé algoritmy s distribuovaným řízením aktualizace replik. Většina synchronizačních algoritmů je založena na hlasování (voting) všech uzlů obsahujících daná replikovaná data.**
- **master-slave – každá aktualizace je provedena nejprve v primární replice na místě označeném jako master a tuto změna je poté propagována do ostatních míst. Dle požadavku na aplikaci lze tyto změny promítnout buď okamžitě nebo po uplynutí časového intervalu. Slabým místem této architektury je možnost chyby nebo nedostupnosti primární repliky na masteru. To vadí v případě, když chceme aktualizovat data v lokálním místě, ale přes primární repliku, která není dostupná.**

Představitelem obou těchto přístupů byl především CA-Ingres a Sybase. Je zřejmé, že architektura master-slave je snadnější na implementaci, dává lepší předpoklady na zotavení z chyb a vede k rychlejšímu provozu. Ovšem jde o méně obecné řešení. V dnešní době jak replikační server Sybase, tak Enterprise server Oracle, ale i SQL Server umožňují obousměrnou replikaci. Informix používá pro obousměrnou replikaci produkt Praxis International – OmniReplicator.

3.2.10 Způsoby aktualizace replik

Dalším důležitým krokem při návrhu systému je určení způsobu, jakým se budou replikovaná data aktualizovat. Máme k dispozici následující dva typy replikace:

- **datová replikace – jedná se o propagování změn dat pomocí změn řádků relací do míst s odpovídajícími replikami**
- **transakční replikace – aktualizace se provádí spuštěním transakce, která provedla změny nad některou z replik, na ostatních místech obsahujících odpovídající repliku**

V prvním případě se přenášejí stará i nová data pro eventuální řešení konfliktů. Ve druhém případě řeší konflikty sama lokální transakce. Pro asynchronní řízení aktualizace replik je vhodné požadavky řadit do fronty (ta se nazývá distribuční). Není-li požadované místo dostupné, zůstává požadavek ve frontě, v opačném případě se „protlačí“ na dané místo.

Vlastnosti replikovaných data

V této kapitole uvedeme nejprve výhody a nevýhody použití replikovaných dat v distribuovaném systému. V další části kapitoly prodiskutujeme formy používaných dat v DDBS, problémy a náklady na udržování replik ve shodném stavu. Nakonec si uvedeme prostředky používané k synchronizaci replik.

3.2.11 Výhody a nevýhody replikovaných dat:

Výhody:

- **umožňují více uživatelům lokální přístup k datům**
- **umožňují souběžný přístup několika uživatelů k daným datům umístěným v různých uzlech, což vede ke zvýšení výkonu DS**
- **umožňují přístup k datům, i když jsou některé (ne všechny) uzly obsahující kopie těchto dat mimo provoz**
- **zvyšují spolehlivost tím, že máme k dispozici několik archivních kopií dat**

Nevýhody:

- **obtížná údržba shodného obsahu všech kopií stejných dat**
- **vyšší náklady na programové vybavení**
- **náklady na uložení dat v jednotlivých uzlech (v současné době již není moc důležité)**

3.2.12 Formy a distribuce dat v DDBS

Z hlediska distribuce dat rozeznáváme tři základní formy použitých dat:

- **centralizovaná data, kdy všechna data jsou umístěna v jednom centrálním uzlu**
- **rozdělená dat, kdy jsou data rozmístěna v několika různých uzlech a zároveň žádná stejná data nejsou ve více uzlech**
- **replikovaná data, kdy jsou data rozmístěna v několika různých uzlech (stejná data se mohou vyskytovat na více různých uzlech)**

Při návrhu distribuce dat se sledují zejména následující cíle:

a) **Dosažení místního zpracování – data jsou umístěna co nejblíže aplikaci, která je využívá, tedy ideálně jsou data i příslušná aplikace umístěny na stejném uzlu. Předností takovýchto aplikací není jen vyloučení přístupu k jiným uzlům, což ovlivňuje dobu odezvy aplikace, ale i výrazně nižší a jednodušší řízení aplikace.**

b) **Dosažení vyšší spolehlivosti při zpracování – umístěním kopie stejných dat do několika různých uzlů (tzv. replikovaných dat). V případě poruchy počítače v některém z těchto uzlů nebo poškozením kopie, lze využít některou jinou kopii stejných dat.**

c) **Dosažení rozdělení pracovní zátěže – provádí se zejména za účelem dosažení co nejvyššího stupně paralelního zpracování .**

3.2.13 Náročnost systému pro replikaci

Obtížnost, nákladnost a vynaložení systémových zdrojů na údržbu replik ve shodném stavu je závislá na několika faktorech, které jsou dány především:

a) **požadavkem aplikace na to, jak přísně z časového hlediska musí být jednotlivé repliky udržovány ve shodném stavu, tj. synchronizovány. Pouze u extrémních aplikací se vyžaduje, aby byly repliky vždy v daném okamžiku shodné. Toto se označováno jako *shodnost v reálném čase*. U většiny aplikací postačuje, pokud jsou repliky shodné po uplynutí nějakého časového intervalu (většinou jeden den).**

b) **četností změn prováděných na daných datech. Data, která podléhají často změnám, označujeme jako *dynamická data* a data, která jsou málo měněna označujeme jako *statická data*.**

c) **rozsahem uvažovaných dat**

Vzhledem k vysokým nákladům na synchronizaci replik není vhodné provádět synchronizaci nad rozsáhlými daty v reálném čase, s výjimkou dat statických.

3.2.14 Data vhodná pro replikace

Pro replikaci jsou vhodná především data s těmito vlastnostmi:

- **s daty pracuje velký počet transakcí z různých uzlů**
- **data se neaktualizují často, tj. jsou v zásadě statická**
- **aktualizace jsou jednoduché, tedy lze snadno udržet integritu celého systému**
- **aktualizace nejsou časově kritické, tj. nemusí se provádět v reálném čase**

- **rozsah případných replikovaných dat není příliš velký – cenové náklady na vnější paměti (v současné době ztrácí tato podmínka na důležitosti)**

3.2.15 Prostředky k zajištění synchronizace replik

Nyní uvedeme základní techniky a jejich stručný popis pro řízení synchronizace replik v distribuovaném systému.

- zámek**, složí k řízení přístupu ke zdroji, který lze využít pouze jedním procesem, který drží zámek, tj. nemůže být současně sdílen více procesy. Použití zámků zajišťuje vzájemné vyloučení procesů.**
- hlasování (voting)**, využívá se v případě, kdy více procesů se potřebuje dohodnout na jednoznačném společném postupu. Lze využívat různé algoritmy, které jsou často založené na jednom řídící uzlu (koordinátorovi), který vyhodnocuje hlasy zúčastněných uzlů a na jejich základě určí další postup. Jedním z použití tohoto mechanismu je dvoufázový potvrzovací protokol.**
- time-out**, se využívá k tomu, aby určitý proces nečekal nekonečně dlouho na vznik nějaké události, která podmiňuje jeho další chování. Každý proces má určen svůj časový interval, po jehož uplynutí (time-out) přestane na danou událost čekat. Např. když transakce nezíská po určitou dobu přístup k datům, přestane čekat a provede *abort***
- časové razítko**, je jednoznačné číslo spojené s objektem nebo událostí v DS. (většinou vyjadřuje čas vzniku transakce) Tuto hodnotu lze považovat za čas, i když nemusí být použit skutečný čas, ale hodnota z monotónně rostoucí posloupnosti. Nejmladší transakce má nejvyšší hodnotu časového razítka. Toho se využívá k uspořádání transakcí a detekci konfliktu, který vznikne pokusem transakce provést operaci mimo pořadí v tomto uspořádání.**

Lze říci, že pro obsluhu soutěžení procesů (typicky pro souběžný přístup k datům) se používají zámky, časová razítka, případně time-out. Pro řízení spolupráce procesů se obvykle používá hlasování, time-out, případně časová razítka.

4 Životní cyklus projektu IS

4.1 Vlastnosti kvalitního IS:

1. Je užitečný a účinný, efekt z používání je prokazatelný
2. Je uživatelsky příjemný, lehce ovladatelný a estetický
3. Je odolný vůči chybám uživatele
vůči technickým poruchám
4. Působí důvěryhodně
5. Má integrovánu nápovědu
6. Jde o otevřený systém přístupný změnám a úpravám
7. Je založen na v čase stálých principech.

4.2 Životní cyklus projektu IS – shrnutí

Úvodem se zamysleme proč tvořit IS. Nejjednodušší odpovědí je naučit počítač něco co usnadní nám nebo jiným život. Samozřejmě pro tvůrce zde přibývá i zájem vydělat.

To jakým postupem systém tvořit bude obsahem této kapitoly. Budeme se zabývat jednotlivými prvky životního cyklu projektu IS.

Průzkum trhu
Předběžná analýza
Analýza systému
Projektová studie
Implementace
Testování
Zavádění systému
Zkušební provoz
Rutinní provoz
Reengineering

II.0. Průzkum trhu

Pojďme k otázce co tvořit. Prvotní etapou projektu je zformulování vlastního obsahu IS a jeho zaměření. Samozřejmě při opakované tvorbě či nasazování systému stejného typu je tento prvek prováděn pouze jednou (resp. při opakovaném nasazení je značně omezen).

Cílem průzkumu trhu je najít oblast, ve které lze nejen IS vytvořit (to lze celkem snadno), ale i úspěšně nabídnout a prodat.

Problémy současného trhu:

levná konkurence malých dodavatelů jednoduchých programů a samostatných programátorů
existence firem, které zdědily různé systémy a zejména kontakty z doby předrevoluční

silná marketingová činnost zahraničních firem
stále určitá nedůvěra zákazníků vůči výpočetní technice.

Možnosti realizace průzkumu trhu:

nahrazení průzkumu trhu pouze tvorbou subdodávek
„zázračná“ intuice
odhad dle zkušeností a názoru dalších odborníků
postup kopírováním stavu, který nastal v zahraničí
vlastní průzkum u potencionálních zákazníků
použití specializované firmy pro vytvoření průzkumu.

Výsledky průzkumu trhu:

oblast působnosti pro tvorbu IS
odhad zázemí (počtu a velikosti) zákazníků
finanční síla potencionálních zákazníků
informace o konkurenci a jejích produktech
rizika při selhání produktu
zdroje nutné pro dokončení
časový plán a časové rezervy
studie o postupu etablování se v nové oblasti trhu
pokud možno pilotní zákazník (i pod cenou).

II.1. Předběžná analýza

Prvotním impulsem pro tvorbu IS je požadavek (požadavky) zákazníka. Je nutné tyto základní požadavky v hrubých rysech rozpracovat, odhadnout náklady a čas realizace.

V této etapě realizace nejsme schopni sestavit detailní postup dalších prací, naším cílem je vytvořit rámcový projekt, který bude obsahovat jen základní, nejdůležitější funkce. Cílem této etapy je posouzení proveditelnosti projektu, hrubý odhad jeho ekonomické efektivnosti, určení možných rizik a hrubý návrh řešení. Rámcový projekt pak má asi tuto strukturu:

časový plán projektu
zdroje požadované pro řešení (finance, personál, technika, ...)
odhad investiční náročnosti
odhad funkčnosti
odhad ekonomické efektivnosti

Pro vytvoření projektu máme následující nástroje:

Analýza současného stavu. Cílem je získat znalosti o současném stavu, identifikovat nedostatky, navrhnout změny.

Katalog uživatelských požadavků. Cílem specifikace požadovaných výstupních informací, informace o zvyklostech uživatele, požadavky obsluhy.
Problémový katalog. Obsahem katalogu je popis problému, popis možných důsledků, nástin možného řešení.
Návrh řešení systému. Je závěrečným dokumentem vzniklým na základě předchozích nástrojů. Obsahuje popis účelu systému, identifikace uživatelů a hranic subsystémů, závěry z analýzy katalogu požadavků uživatele, návrh hlavních funkčních celků, seznam rozhodujících událostí, odhad velikosti datové základny, představa o technickém řešení.

II.2. Analýza systému

Jedná se o klíčový úsek vývoje produktu. Chyby v návrzích struktury dat, funkčnosti ale i systémového SW (databázový stroj) a HW jsou později jen velmi obtížně odstranitelné. Strukturu analýzy systému je možno rozčlenit takto:

funkční popis systému
datová analýza
modulární konstrukce
formy vstupu a výstupu
organizace a ochrana dat
odhad velikosti systému
návrh koncepce testování
hardwarová studie
ekonomické hodnocení

Funkční popis systému

Model toho, co má systém dělat, za jakých podmínek činnosti vykonává, odkud kam posílá výsledky. Popis by měl obsahovat:

- určení vstupních a výstupních datových toků
- činnost jednotlivých funkcí, tj. způsob transformace vstupních datových toků na výstupní (jedná se o rozpracování katalogu uživatelských požadavků)
- formulace podmínek pro výkon jednotlivých činností.

Datová analýza

Cílem je vytvořit statický model reality, tj. popsat objekty, jejich vlastnosti a vzájemné vztahy. Pro realizaci tohoto cíle existují různé techniky (popíšeme později tzv. ER model).

Modulární konstrukce

Funkce vzniklé z funkčního popisu systému budou rozděleny do modulů dle svého obsahu. Systém by měl být navržen po vrstvách dle modulů přibližně stejné složitosti.

Formy vstupu a výstupu

Návrh vnějšího tvaru systému (návrh obrazovek, dialogu s uživatelem, tiskových výstupů).

Organizace a ochrana dat

Jde o transformaci ER diagramů z funkční analýzy do relační struktury, normalizace relačních schémat (3NF, BCNF), vnitřní rozdělení a optimalizace datové základny.

Odhad velikosti systému

Zjištění přibližné velikosti datových struktur, odhad mohutnosti datových toků mezi systémem a okolím. Podklad pro volbu technického vybavení.

Návrh koncepce testování

Návrh plánu testů pro jednotlivé moduly, seznamy „tvrzení“ o funkčnosti systému, které se budou experimentálně ověřovat při testování.

Hardwarová studie

Po dokončení funkční a datové analýzy jsou k dispozici potřebné informace pro specifikaci technických prostředků. Jedná se o velice důležitou součást analýzy.

Ekonomické zhodnocení

Veškeré předchozí součásti dávají k dispozici informace pro upřesnění odhadu ekonomické efektivity.

Výsledkem analýzy systému je nejdůležitější dokument předrealizační etapy, projektová dokumentace.Tento dokument je určující pro konečné rozhodnutí zákazníka o volbě systému, pro uzavření smlouvy, pro určení časového harmonogramu a ceny. Projektová dokumentace je též základním podkladem pro konečné předání systému resp. pro řešení sporných momentů.

II.2. Projektová studie

Je výsledkem analýzy systému. Jedná se o jeden z nejdůležitějších dokumentů celé fáze realizace. Projektová studie je podkladem pro:

obsah obchodní smlouvy, časový harmonogram a cenu
vlastní implementaci
zavádění u odběratele
způsob dílčích a závěrečné zkoušky
předání u odběratele.

Z hlediska obsahu by měla projektová studie obsahovat veškeré informace zjištěné ve fázi analýzy. Základní členění projektové studie by mělo obsahovat:

základní informace o dodavateli systému včetně možných subdodavatelů
základní informace o odběrateli včetně týmu pro nasazení systému
popis současného stavu u odběratele
globální návrh systému
detailní popis systému v členění
funkční analýza systému
datová analýza systému
popis datových toků
popis událostmi řízených funkcí
popis modulární konstrukce
popis způsobu nasazování systému
popis funkčních zkoušek
hardwarová studie
časový harmonogram dodávek
ceny a platební kalendář

Základním prvkem při tvorbě projektové studie by měla být maximální podrobnost, která je v rámci času, znalostí a finančních prostředků možná. Důvodem je důležitost tohoto dokumentu jak z hlediska smluvního (buď přímo součást smlouvy nebo podklad pro její tvorbu), z hlediska implementačního (na základě smlouvy je realizován systém), z hlediska doložení správnosti realizovaného systému (způsob funkčních zkoušek, popis úplnosti systému).

Projektová studie musí být psána i s ohledem na to, že na jejím základě provádí zákazník závěrečné rozhodnutí o realizaci systému, resp. o výběru dodavatele pro systém. Připomínky zákazníka k projektové studii, které jsou zaznamenány v období oponentního řízení), jsou shrnuty v takzvaném zápisu z oponentury. Na základě tohoto zápisu buď zákazník projektovou studii nepřijme. V opačném případě je buď požadováno, aby připomínky byly do studie zapracovány a ta předložena znovu, nebo se připomínky včetně vyjádření dodavatele stanou přílohou projektové studie.

II.3. Implementace

Jedná se o etapu, ve které dochází k vlastnímu „programování“. Pro některé až zde začíná a vlastně i končí práce, což jak jsme si již říkali je obrovský omyl. Nelze však i tuto etapu podceňovat, protože je základní etapou vlastního vývoje systému.

Základem implementace jsou poznatky získané při analýze systému a při tvorbě projektové studie. Vychází se z datové analýzy a z funkční analýzy. U větších systémů je vhodné vytvořit i tzv. programátorskou příručku, tj. rozšíření funkční analýzy o informace potřebné pro programátory.

Postup prací je možné shrnout do následujících kroků:

Zadání pro jednotlivé programy

Na základě funkční analýzy resp. programátorské příručky je vytvořeno zadání, které určuje vždy data a jejich transformace v konkrétním modulu.

Programová realizace

Na základě provedené analýzy probíhá fáze programování. Čím více je věnováno fázi přípravy podkladů v předchozích etapách tím kratší a méně „kvalifikovaná“ může být tato fáze. Všeobecným trendem je přesunout těžiště prací do analytické části a fázi programování zkrátit na minimum.

Příprava testů

Fáze programové realizace nekončí pouze „naprogramováním“, je potřeba ověřit správnost realizovaných funkcí. Pro testování je nutné připravit vzorová testovací data. Této přípravě je nutné dbát na to, aby bylo pokryto maximum variant skutečných dat.

Provádění jednotlivých testů

Jedná se o odzkoušení reakcí programů na všechny běžné typy vstupních dat. Metody pro verifikaci programů (ověření, potvrzení správnosti) nejsou většinou k dispozici a bývají používány pouze mimořádně (v případech, kdy selhání programu může mít neodhadnutelné výsledky).

Pro realizaci této etapy je potřeba vytvořit fungující tým programátorů vedený vždy analytikem, který zodpovídá za správnost řešení. Správná volba a složení týmu je nutná pro optimální realizaci systému.

5 LITERATURA

Scheber A., Databázové systémy, Alfa 1988
Král J., Informační systémy, Science 1998, Veletiny
Pour J., Dohnal J. Architektury informačních systémů, EKOPRESS 1997
Pour J., Aplikační systémy, EKOPRESS 1997
Straka M., Vývoj databázových aplikací, Grada 1992