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. OLTP
    - 2.0. Charakteristika transakčních systémů, OLTP a OLAP
    - 2.1. SQL
    - 2.2. Datový model
    - 2.3. Transakční zpracování
    - 2.4. Centralizované a distribuované databáze

3. OLAP
    - 3.0. OLAP systémy a datové krychle
    - 3.1. Analytické nástroje databází
    - 3.2. Datové sklady
    - 3.3. Dolování dat

4. Úvod do In-MemoryDataManagement

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
System Integration 2001. Sborník 9. ročníku konference. (si.vse.cz)
L.Lacko, Business Inteligence v SQL Serveru 2005,Reportovací, analytické a další datové služby, Computer Press 2006
Plattner H., Zeier A., In-Memory Data Management, Springer-Verlag, 2011

1 Úvod

1.1 DBS

- Jazyk pro definici dat
    - SQL
    - Datové slovníky

- Jazyk pro manipulaci s daty
    - SQL
    - 4 GL
    - Objektové jazyky ve vazbě na SQL

- Databázový systém
    - Základní vlastnosti – uchování dat a práce s daty
    - Požadavky na zabezpečení a ochranu dat
    - Rychlost zpracování a množství dat
    - Více-uživatelský přístup k datům (transakčnost)
    - Portabilita a otevřenost

- Komerční databázové systémy
    - DBS – praktický nástroj
    - Free a ShareWare
    - Malé DBS
    - Střední a velké DBS
    - Zabezpečení a ochrana dat – záruka za data

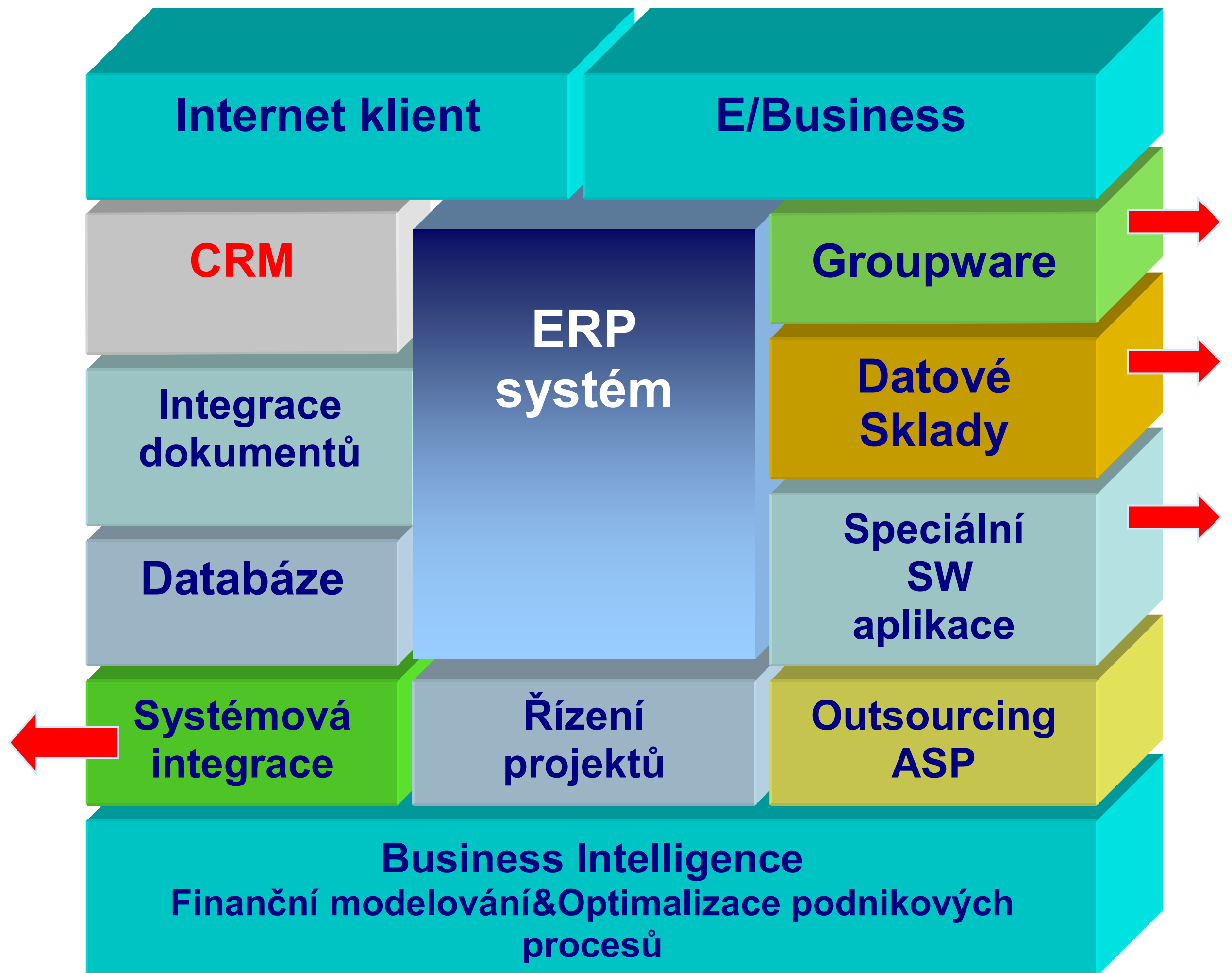
1.2 Co je to IS, vazba na DBS?

Struktura IS

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.


Internet klient
E/Business
CRM
ERP systém
Groupware
Integrace dokumentů
Datové Sklady
Databáze
Speciální SW aplikace
Systémová integrace
Řízení projektů
Outsourcing ASP
Business Intelligence
Finanční modelování&Optimalizace podnikových procesů

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.*

1.3 Základní pojmy databází

1.3.1 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.

1.3.2 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 $R(A_1{:}D_1, A_2{:}D_2, \ldots, A_n{:}D_n)$

1.3.3 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.

1.3.4 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).

1.3.5 Normální formy relací
1.NF, 2.NF, 3.NF, BCNF

2 OLTP

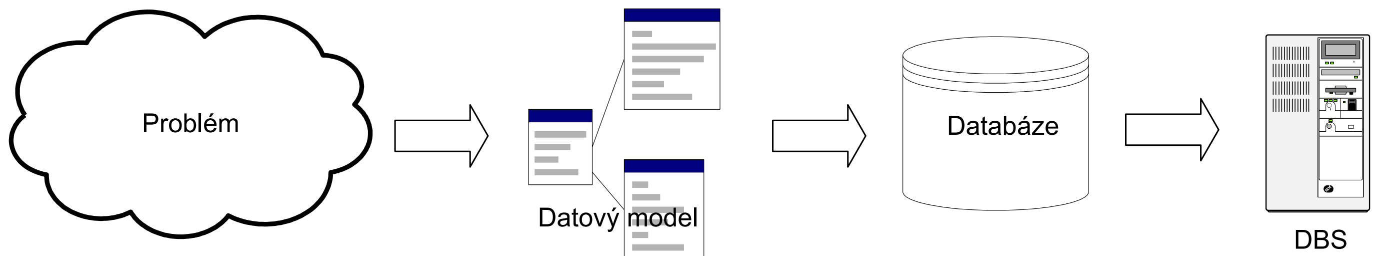
OLTP systémy uchovávají záznamy o jednotlivých uskutečněných transakcích a jsou obvykle realizovány pomocí dnes nejběžnější, relační databázové technologie. Data uchovávaná v OLTP databázovém systému jsou (zpravidla periodicky) agregována (typicky sumarizována) a poté ukládána do da-tového skladu, nad nímž se posléze podle potřeb provádí okamžité zpracování analýz pomocí vrstvy OLAP.
Shrnující přehled veškerých rozdílů je uveden v následující tabulce:

Provozní databáze – OLTP	Datový sklad - OLAP
Koncepční rozdíly	
Dostat data do systému	Dostat informace ze systému
Uživatelé mají možnost zadávat data, měnit, rušit a číst data	Uživatelé mají možnost pouze číst data
Zajišťují automatizaci rutinních činností	Umožňují kreativitu uživatelů při práci s daty
Aplikace jsou v podstatě statické	Aplikace jsou dynamické
Podporují každodenní firemní aktivity	Podporují dlouhodobé strategie
Orientované na výkonnost	Poskytují konkurenční výhodu
Proces implementace a využívání je poháněn technologií	Proces implementace a využívání je poháněn potřebami organizace
Technologické rozdíly	
Zpracovávají velké objemy malých transakcí	Zpracovávají malý počet komplexních dotazů
Transakce neustále přidávají a aktualizují data	Data se načítají dávkově
Důležitým hlediskem je omezení redundance dat	Důležitým hlediskem je rychlý přístup k datům pro účely analýz a prezentací
Integrita dat se zajišťuje datovým modelem a aplikacemi	Integrita dat se zajišťuje při dávkových procesech transformací dat
Datové modely jsou optimalizované pro online aktualizace a rychlé zpracování transakcí	Datové modely jsou optimalizované pro rychlé zpracování výstupů
Používá se převážně normalizované relační datové modely	Používá se kombinace datových modelů (normalizované a denormalizované relační modely, sumarizované tabulky, star schéma)

2.1 Datové modelování, databázové systémy

- Obsah a techniky DM
- Metodologie

2.1.1 Obsah a techniky DM



Datový model:

- **Entity**
- **Atributy**
- **Vazby mezi entitami**
- **Integritní omezení**
- **Triggery**
- **Uložené procedury**

Nástroje pro DM:

- **Grafické zobrazení**
- **Kontrolní mechanismy (data dictionary, normalita)**
- **Vazba na konkrétní databáze**

Datové modely:

- **Konceptuální – ERD**
- **Fyzický – struktura databáze**

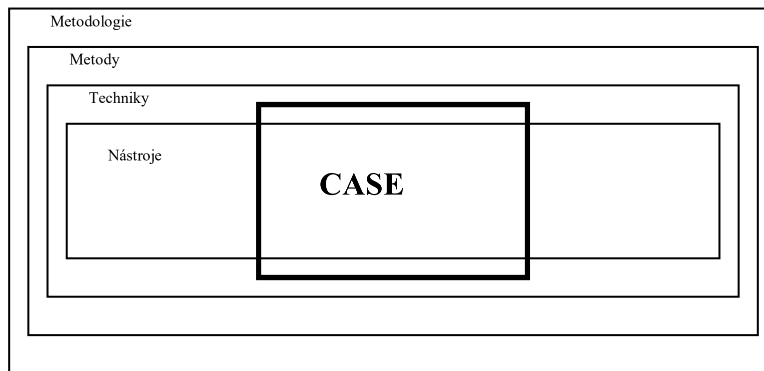
2.1.2 Metodologie – datový model CDM, PDM

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

2.1.3 Datové modelování v nástroji PowerDesigner

2.2 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**

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.

2.2.1 Kontrola souběžnosti (concurrency control)

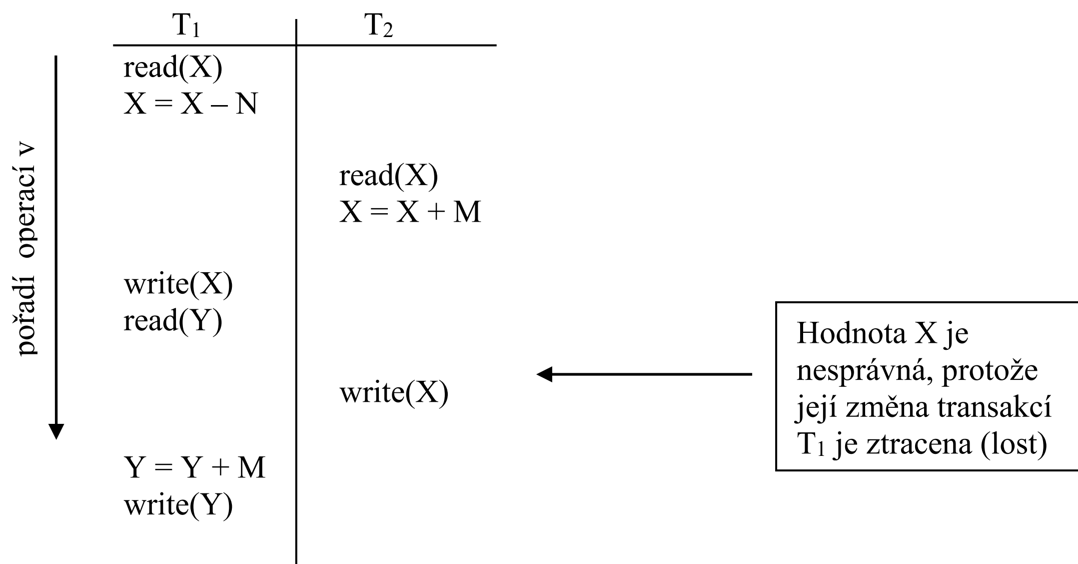
Proč vlastně kontrola souběžnosti transakcí? V případě kdy běží několik transakcí současně, může se stát, že tyto transakce mění tentýž databázový záznam. Pokud by tento souběžný běh transakcí nebyl kontrolován, mohl by nastat problém, jakým je *nekonzistentní databáze*. Proto v další části popíšeme 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:		T2:	
	read(x)		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 záznamuv 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.

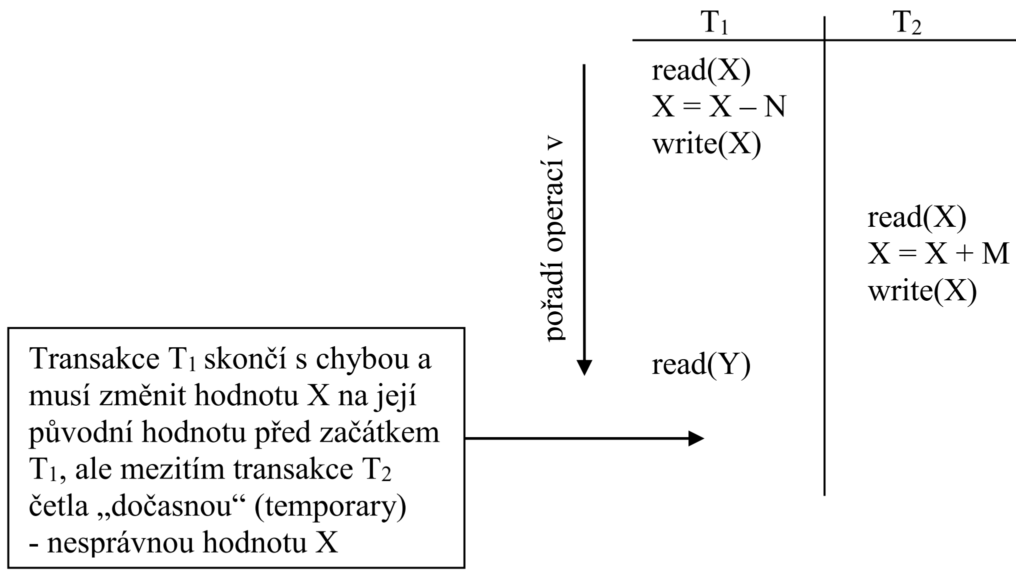


Obrázek Příklad návaznosti operací transakcí T_1 a T_2 pro problém Lost Update

Obrázek zobrazuje případ, kdy transakce T_1 přesouvá rezervaci pro N sedadel v letadle X do letadla Y a transakce T_2 rezervuje M sedadel v letadle X. Pro provedení transakcí v pořadí, jka je znázorněno na obrázku zůstane N + M rezervovaných sedadel v letadle X místo pouhých M.

Problém dočasné změny „Temporary Update“

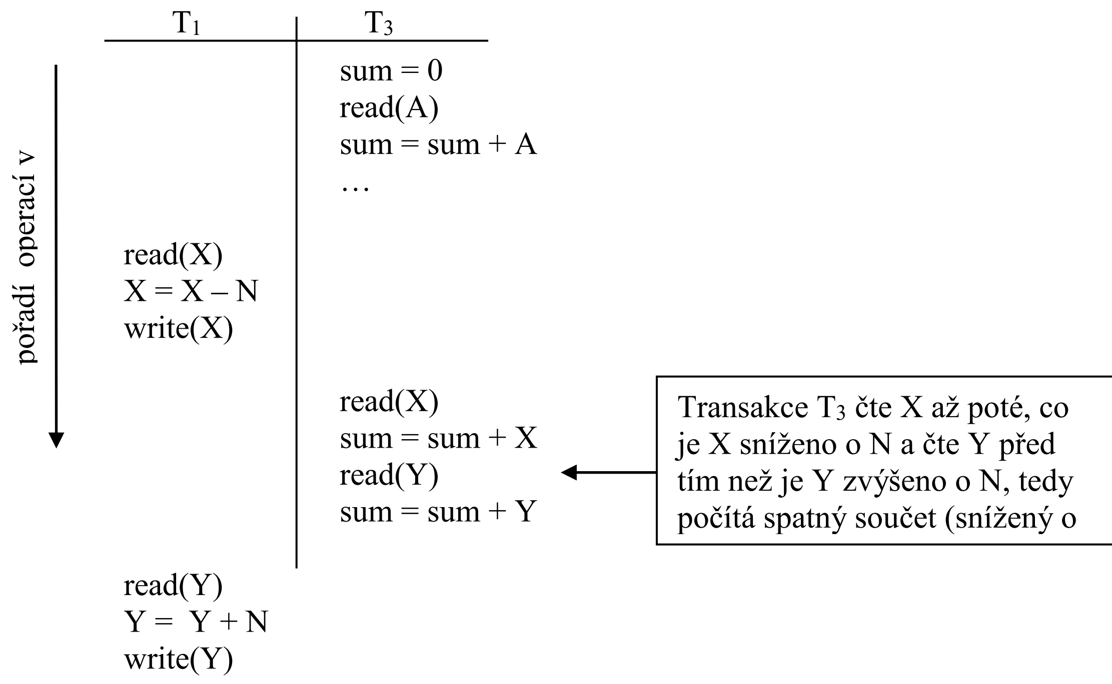
Ten nastane v případě, kdy jedna transakce mění záznam v databázi a poté skončí chybou. Ke změněnému záznamu přistupuje jiná transakce dřív než dojde k vrácení původní hodnoty. Obrázek ukazuje příklad, kdy transakce T_1 mění hodnotu X a skončí před dokončením. Tedy systém musí vrátit změny doposud provedené 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

Problém 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.

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.

<u>*Zamykání*</u>

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.

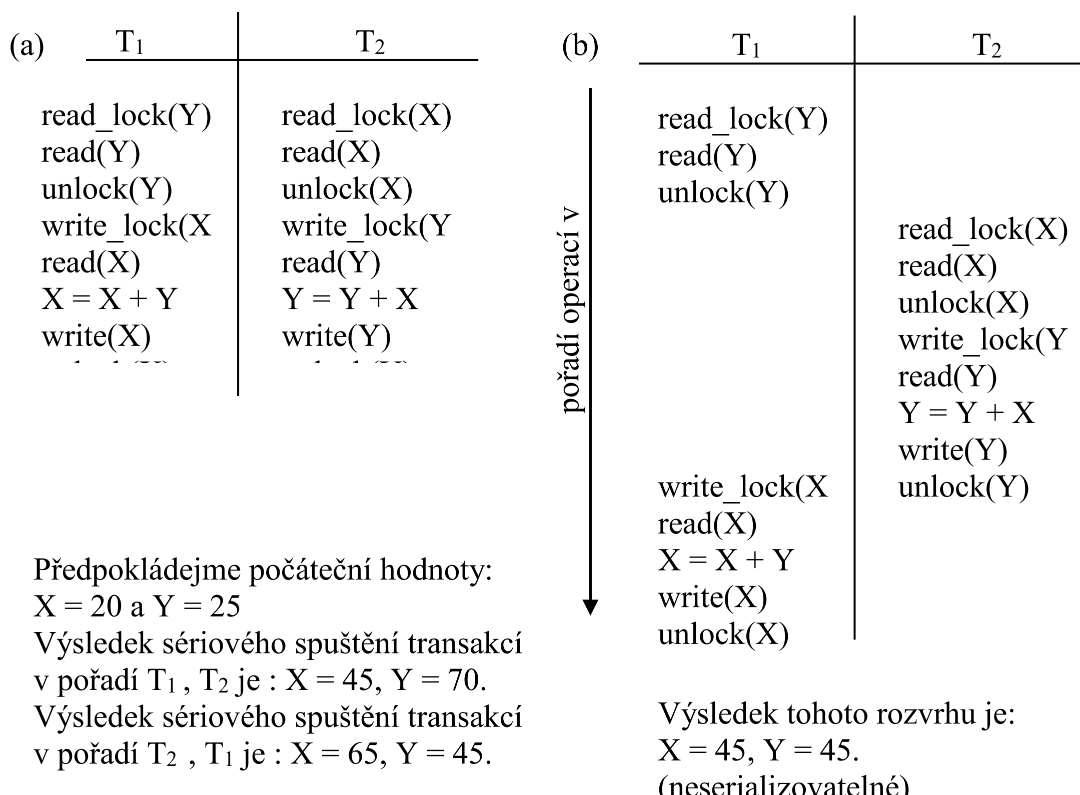
Sdílený a výhradní zámek (Shared and exclusive lock)

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 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:

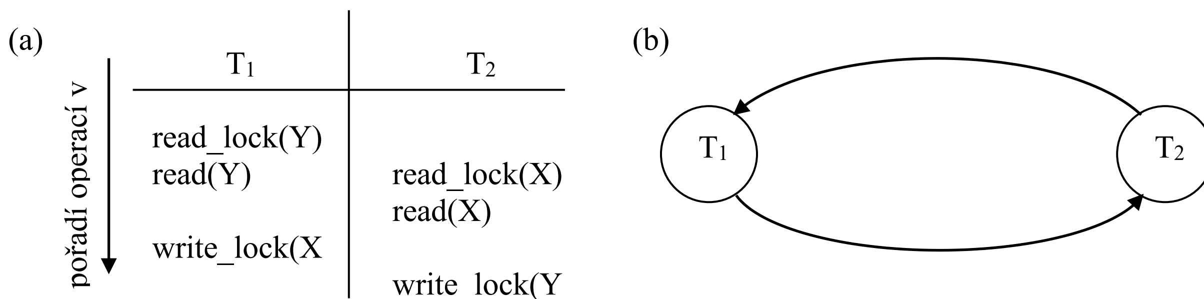
a) Deadlock – nastane pokud transakce čeká na uvolnění zámku, který drží druhá transakce a ta čeká na uvolnění zámku, který drží první transakce (i při cyklickém čekání více než dvou transakcí). Tento případ je zobrazen na obrázku, kde transakce T_1 čeká na uzamčení záznamu X, který má uzamčena transakce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

2.2.2 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.

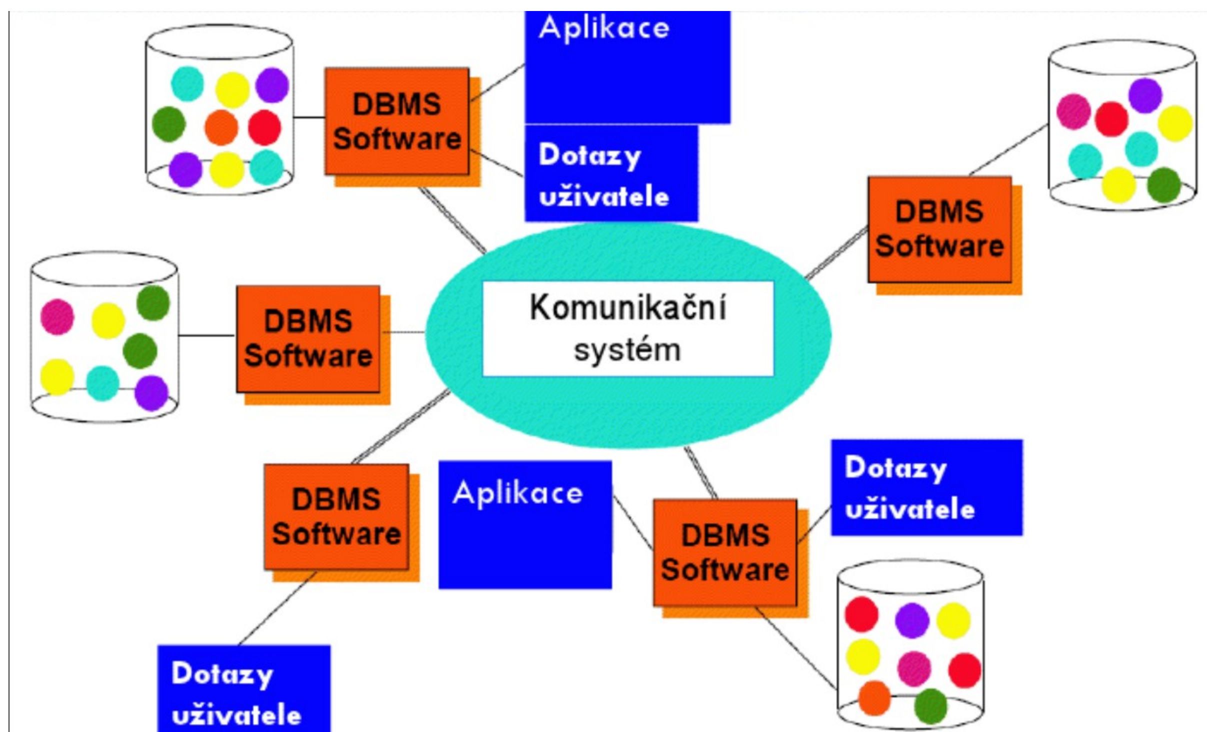
2.2.3 Kontrola atomičnosti

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.

2.3 Centralizované a distribuované databáze



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

2.3.1 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.

2.3.2 Modely dat

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

2.3.3 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.

2.3.4 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.

2.3.5 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.

2.3.6 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.

2.3.7 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*.

Příklad

ZAMĚSTNANEC	CisloZ	PrijmeniZ	JmenoZ	AdresaZ
	DatNarZ	PohlaviZ	OddZ	PlatZ

ODDĚLENÍ	CisloO	NazevO	MistoO	VedouciO

PROJEKT	CisloP	OddP	NazevP

PRACUJE_NA	Zam	Proj	Hodiny

DÍTĚ	CisloZ	JmenoD	DatNarD

Fragmentace dat

V DDBS se mimo jiné jedná o to, jak optimálně rozdělit data na jednotlivá místa. Pro jednoduchost zatím předpokládejme, že data nejsou replikovaná tj. že každý soubor či část souboru je uložena na právě jediném místě.
Dříve než se rozhodneme, jak distribuovat data, je třeba určit logickou jednotku (část databáze), která se bude distribuovat. Nejjednodušší logickou jednotkou je tabulka (entita, relace). V řadě případů je relace v rámci distribuce rozdělena na menší logické jednotky.

Horizontální a vertikální fragmentace
Horizontální fragment relace je podmnožina záznamů v tabulce (relaci). N-tice patřící do horizontálního fragmentu je dána podmínkou na jeden nebo více atributů v tabulce (selekce). V našem příkladu můžeme definovat tři podmínky pro fragmentaci relace ZAMĚSTNANEC: (OddZ = 10), (OddZ = 20), (OddZ = 30), každý fragment obsahuje záznamy pracovníků zařazených na příslušné oddělení. Podobně samozřejmě můžeme definovat fragmenty relace PROJEKT podmínkami : (CisloO = 10), (CisloO = 20), (CisloO = 30). Horizontální fragmentace rozděluje relaci horizontálně – vznikají tak skupiny řádků (podmnožiny relace), které k sobě logicky patří. Tyto fragmenty pak mohou být umístěny každý do jiného uzlu.
Vertikální fragment relace obsahuje pouze vybrané atributy relace (projekce). Mohli bychom například rozdělit relaci ZAMĚSTNANEC na dva vertikální fragmenty, z nichž první by obsahoval personální informace (JmenoZ, PrijmeniZ, AdresaZ, DatNarZ, PohlaviZ) a druhý pracovní informace (CisloZ, OddZ, PlatZ). Toto rozdělení ovšem není správné, protože pokud jsou tyto fragmenty uloženy samostatně, již nikdy nezískáme původní informace, protože v nich není obsažen rozumný společný (spojující) atribut. Do každého vertikálního fragmentu je tedy nezbytné zahrnout atributy primárního klíče relace, jen tak je možné z fragmentů rekonstruovat původní relaci.
Kombinovaná (mixed) fragmentace vznikne jako kombinace obojího výše uvedeného. Původní relaci pak získáme pomocí operací sjednocení a vnější sjednocení (vnější spojení) v odpovídajícím pořadí. Obecně takovýto fragment relace se dá získat pomocí kombinace selekce a projekce.
Fragmentační schéma databáze je definice množiny fragmentů, které zahrnují všechny atributy a n-tice (záznamy) databáze a splňují podmínku, že celá databáze může být rekonstruována z fragmentů aplikováním posloupnosti operací OUTER UNION (nebo OUTER JOIN) a UNION. Je užitečné, ale nikoliv nezbytné, aby všechny fragmenty byly disjunktní (s výjimkou atributů primárního klíče).
Alokační schéma popisuje uložení fragmentů v jednotlivých místech (uzlech) databáze tj. pro každý fragment je specifikován uzel, kde je uložen. Je-li fragment uložen na více než jednom místě, hovoříme o replikaci

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.

2.3.8 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.

2.3.9 Formy replikace

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

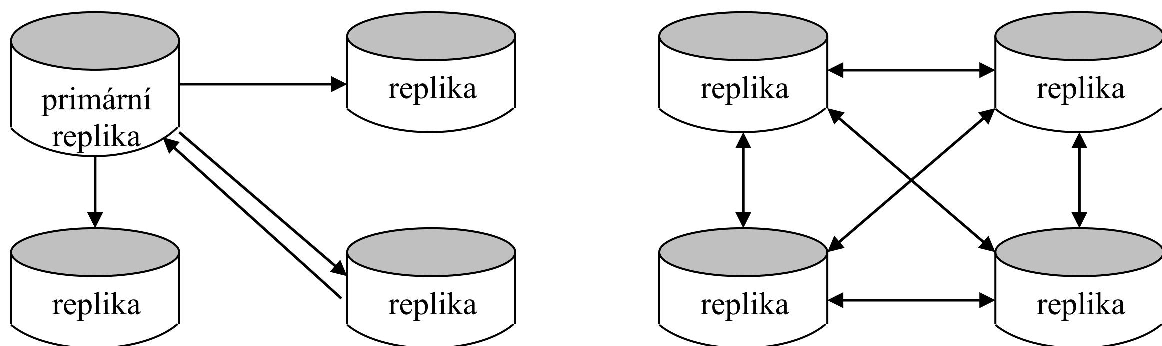
- **Jednosměrné systémy – data aktualizuje pouze jeden účastník**
- **Obousměrné (symetrické) systémy – aktualizovat data může několik účastníků**

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.

2.3.10 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.

2.3.11 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)**

2.3.12 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 .**

2.3.13 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.

2.3.14 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)**

2.3.15 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.

3 OLAP

analytické databáze slouží jako podklad pro získání sumarizovaných a agregovaných údajů a jsou známé pod pojmem OLAP (*Online Analytical Procesing*). Tato zkratka zahrnuje nejen struktury údajů ale i analytické služby, které slouží pro analýzu velkého množství údajů.

3.1 Datové sklady

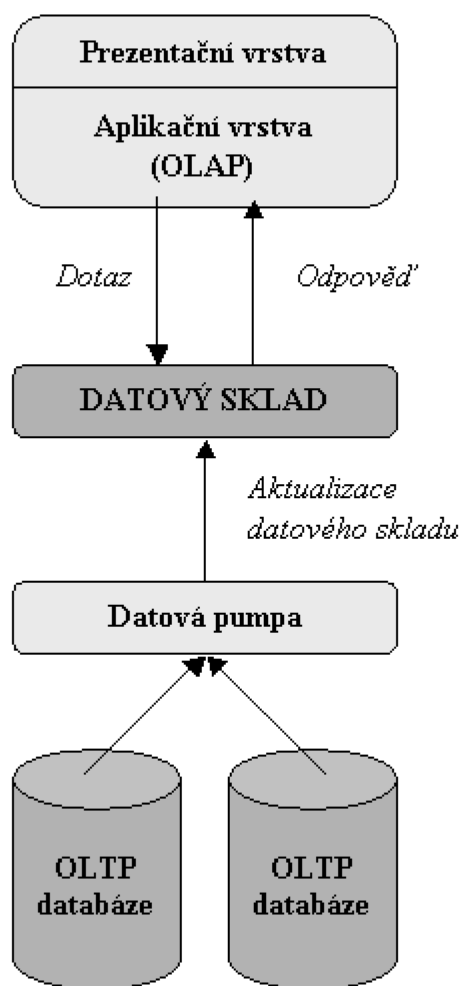
Co to je vlastně datový sklad (*data warehouse*)? Datový sklad je podnikově strukturovaný depozitář subjektově orientovaných, integrovaných, časově proměnlivých, historických dat použitých na získávání informací a podporu rozhodování.

Tolik definice. Co vše je v ní skryto, je potřeba vysvětlit podrobněji:

- **Subjektová orientace** - údaje do datového skladu se zapisují spíše podle předmětu zájmu než podle aplikace, ve které byly vytvořeny. Předmětem zájmu, nebo subjektem, je zde myšlena orientace např. na zákazníka, zaměstnance, výrobek.
- **Integrovanost** – datový sklad musí být jednotný a integrovaný, tzn. údaje týkající se jednoho předmětu jsou ukládána pouze jednou. Znamená to zavést jednotnou terminologii a konzistentní jednotky veličin.
- **Časová variabilita** – údaje jsou ukládány v časové posloupnosti, údaje jsou platné pro určitý časový moment.
- **Neměnnost** – údaje se obvykle nemění ani neodstraňují, jen se v pravidelných intervalech přidávají nové.
  Připouští se pouze dva typy operací - zavedení do skladu,
  - přístup k těmto údajům.

V datových skladech nejsou „živá“ data, která uživatelé přímo aktualizují. Do datových skladů se data přesunují z jiných zdrojů dat (většinou provozních systémů) a transformují se do datového modelu navrženého specificky pro analýzy velkého množství dat. Díky zaměření datových skladů na sledování trendů a vzájemné porovnávání výsledků v mnohaleté historii jsou analýzy prováděné nad tabulkami se stamiliony záznamů běžnou praxí. Právě kvalita návrhu datového modelu datového skladu ovlivňuje nejvyšší měrou výslednou použitelnost řešení. Sebelepší nadstavba v podobě nástroje BI (business intelligence) či jakékoliv aplikace pro analýzy nemůže eliminovat handicap vzniklý nekvalitním návrhem datového modelu.

Celý systém datového hospodaření lze rozdělit tedy na dvě základní části. První z nich je OLAP. Na druhé straně stojí klasické databázové systémy, které se označují jako OLTP, což je zkratka „on-line transaction processing“ neboli „okamžité zpracování transakcí“.



Obr.č.4 Struktura datového skladu

Rozdílnost mezi OLAP a OLTP spočívá v tom, že OLTP systémy uchovávají záznamy o jedno-tlivých uskutečněných (typicky obchodních) transakcích a jsou obvykle realizovány pomocí dnes nejběžnější, relační databázové technologie. Data uchovávaná v OLTP databázovém systému jsou (zpravidla periodicky) agregována (typicky sumarizována) a poté ukládána do da-tového skladu, nad nímž se posléze podle potřeb provádí okamžité zpracování analýz pomocí vrstvy OLAP.
Datový sklad je na rozdíl od OLTP databáze určen výhradně ke čtení dat pro potřeby nejrůznějších analýz. Jedinou výjimkou jsou (obvykle periodické) aktualizace datového skladu, tj. přidávání nových datových agregátů či odstraňování již neaktuálních datových agregátů, které probíhají obvykle periodicky každý týden, měsíc, atp. Tyto akce je ovšem možno chápat za součást údržby datového skladu, která probíhá ve speciálním režimu při momentálním vyloučení zpracování OLAP požadavků uživatelů datového skladu. V běžném režimu práce (tzn. při provádění dotazů a analýz) není obsah datového skladu modifikován. Tento zásadní rozdíl mezi OLTP systémy a datovými sklady má rozsáhlé důsledky pro způsob jeho implementace, návrhu a tvorby konceptuálního modelu, který je orientován na dosažení co nejrychlejšího zpracování dotazů kladených datovému skladu vrstvou OLAP.

Shrnující přehled veškerých rozdílů je uveden v následující tabulce:

Provozní databáze	Datový sklad
Koncepční rozdíly	
Dostat data do systému	Dostat informace ze systému

Uživatelé mají možnost zadávat data, měnit, rušit a číst data	Uživatelé mají možnost pouze číst data
Zajišťují automatizaci rutinních činností	Umožňují kreativitu uživatelů při práci s daty
Aplikace jsou v podstatě statické	Aplikace jsou dynamické
Podporují každodenní firemní aktivity	Podporují dlouhodobé strategie
Orientované na výkonnost	Poskytují konkurenční výhodu
Proces implementace a využívání je poháněn technologií	Proces implementace a využívání je poháněn potřebami organizace
Technologické rozdíly	
Zpracovávají velké objemy malých transakcí	Zpracovávají malý počet komplexních dotazů
Transakce neustále přidávají a aktualizují data	Data se načítají dávkově
Důležitým hlediskem je omezení redundance dat	Důležitým hlediskem je rychlý přístup k datům pro účely analýz a prezentací
Integrita dat se zajišťuje datovým modelem a aplikacemi	Integrita dat se zajišťuje při dávkových procesech transformací dat
Datové modely jsou optimalizované pro online aktualizace a rychlé zpracování transakcí	Datové modely jsou optimalizované pro rychlé zpracování výstupů
Používá se převážně normalizované relační datové modely	Používá se kombinace datových modelů (normalizované a denormalizované relační modely, sumarizované tabulky, star schéma)

Tab.č.1 Rozdíly technologií provozních databází (OLTP) a datových skladů

Výše uvedený popis datového skladu je však pro mnoho plně nezasvěcených uživatelů příliš odborný. Mnoho uživatelů chce raději slyšet jednoduché odpovědi na položenou otázku.

Co je to datový sklad a proč ho potřebujeme?

Odpověď IT Manažera:
Firma používá řadu nepropojených systémů, je velice obtížné je udržovat, pro podporu rozhodování nepracují efektivně. Fyzicky tedy oddělíme naše provozní systémy od systémů pro podporu rozhodování a vytvoříme úložiště informací, které je organizované pro efektivní přístup.

Uživatel slyší:
Víme, že údaje z provozních databází skrývají mnoho informací, ale je velice časově náročné je získat. Potřebuje tento čas využít jinak než čekat na výstupy, které musí definovat dopředu bez možnosti operativní reakce.

3.1.1 Architektura datových skladů

V průběhu realizace se prosadily dva základní koncepty architektury datových skladů:

- Nezávislé datové trhy (datamarty)

- Integrovaný datový sklad

Nezávislé datové trhy - tato koncepce přináší řešení potřeb jednotlivých útvarů či aplikací odděleně od sebe a tím se vytváří samostatná datová úložiště. Sjednocením těchto úložišť pak vytváříme datový sklad. Toto řešení je velmi výhodné z hlediska rychlého zavádění a nižších počátečních investic. Přináší ale také nevýhody, a to pokud datové trhy nejsou prvotně budovány s výhledem na sjednocení do jediného datového skladu, pak obsahují nekonzistentní data mezi jednotlivými částmi a tím jsou komplikované načítací a sjednocovací procesy.

Integrovaný datový sklad - při této koncepci se data ukládají do centrálního datového úložiště, ze kterého se následně odvozují datové trhy pro potřeby jednotlivých útvarů. Jde tedy o kon-zistentní obsah, který sebou nese jednodušší správu načítání dat. Nevýhodou je však složitější a pomalejší implementace. Definice všeobecně přijatých dimenzí ze všech následně generovaných částí přináší komplikace ve sjednocení všech pohledů na uložené údaje, ale pouze takto jsme schopni vytvářet konsolidované údaje napříč celým podnikem.

3.1.2 Metody budování datového skladu

Metody budování datových skladů jsou úzce spojeny se zvolenou koncepcí architektury datového skladu. Mezi nejvíce používané metody patří:

- Metoda velkého třesku
- Metoda přírůstková

Metoda velkého třesku znamená zavedení datového skladu pomocí jediného projektu.

Skládá se ze tří etap:

- analýza požadavků,
- vytvoření podnikového datového skladu,
- vytvoření přímého přístupu nebo přístupu přes datové trhy.

Nevýhodou je velká komplikovanost analýzy. Málokdy se totiž podaří vyřešit všechny problémy spojené s analýzou dat na začátku a během realizace dochází k různým změnám.

Metoda přírůstková předpokládá budování datového skladu po jednotlivých etapách, postupně podle jednotlivých předmětných oblastí. Toto částečné řešení pomáhá uživateli postupně se seznamovat s možnostmi, které datový sklad přináší. Dále umožňuje vytvářet jasnější představy o možných výstupech a jejich využití. Jedná se tedy o iterativní proces, který udržuje neustálou spojitost mezi již existujícími částmi datového skladu a potřebami uživatelů.

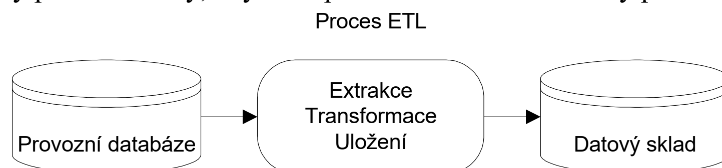
Etapy přírůstkové metody:

- strategie, cíle při budování datového skladu,
- definice architektury datového skladu a technických prostředků,
- analýza získávání dat a požadavků na přístup k datům,
- návrh a transformace požadavků do detailních podmínek,
- sestavení a testování databázových struktur,
- produkce a instalace datového skladu se zajištění provozu a údržby.

3.1.3 Proces přípravy a plnění datových skladů

Do datového skladu se data nezadávají, ale načítají se z provozních systémů. Zdroje mohou být různorodé. Je běžné, že vznikne požadavek na sjednocení a vytěžování informací z řady datových zdrojů, ale tyto zdroje jsou naprosto nekonzistentní, tzn. jsou uloženy ve zcela odlišných strukturách, formátech, některé mohou být i zcela nestrukturované, mají odlišnou filozofii záznamu, jsou uloženy na různých médiích atd. V souvislosti s touto problematikou se objevuje termín ETL. Je to zkratka, která se skládá z prvních tří písmen slov *Extraction, Transformation, Load*. Tato tři slova definují jednotlivé fáze aktivního procesu v datovém skladu.

Celý proces ETL je poměrně časově náročný. Načítání se většinou provádí v čase, kdy nejsou provozní systémy příliš zatíženy, aby se neprodlužovala doba odezvy pro uživatele těchto systémů.



Obr.č.5 Obecné schéma datových skladů

Extrakce dat je prvním a zároveň nejkritičtějším krokem ke správnému a informační hodnotu přinášejícímu využití datového skladu. Jedná se o schopnost převzít data z co nejširšího spektra datových zdrojů nejrůznějšího charakteru, mezi které se řadí nesčetné databázové standardy, nedatabázové strukturované formáty, textové soubory, standardy elektronické pošty apod. Souhrnně tedy můžeme extrakci chápat jako tu pracovní etapu, kdy usilujeme o přesné, rychlé, bezpečné, lehce kontrolovatelné a dobře řiditelné načtení dat z co nejvíce externích datových zdrojů. Tato fáze se opakuje s periodicitou, která souvisí s využíváním daných informací. Po jejím skončení budou potřebná data načtena přímo do připravených zdrojových struktur pro extrahovaná data.

Transformace je postupná řada operací, které extrahovaná data připraví pro vlastní načtení do datového skladu. Drtivá většina informací získaných extrakcí totiž ještě není zdaleka připravena vydat svoje skryté bohatství. Mezi příčiny patří zejména nesoulad mezi daty z jednotlivých zdrojů a jejich neúplnost – viz. následující příklady:

- Nejednoznačnost údajů
- Chybějící záznamy a duplicity
- Formáty čísel a textových řetězců

Základem transformace je vytvoření programové logiky, která provede převod mezi zdrojovými strukturami naplněnými syrovými daty a cílovými strukturami, které jsou zdrojem pro pozdější vytěžování dat. Definice zdrojových a cílových struktur celého procesu transformace a pravidel, která zkontrolují, doplní nebo změní data, pokud nejsou korektní, je velmi náročným a pro každý projekt specifickým úkolem.
Další nedílnou součástí transformace je validace, tzn. ověření správnosti extrahovaných dat, případně odhalení rozporů v těchto datech. Transformace je tedy chápána jako proces získání co nejkvalitnějších dat, protože informace jsou jen tak dobré, jak dobrá je kvalita dat. Tak, jak je kvalitní každý jednotlivý záznam, je kvalitní i datový sklad. Kvalita dat je kritickým faktorem pro úspěch celého projektu datového skladu.

Přenos je poslední částí celého procesu, kdy jsou transformovaná data natažena do vlastního fyzického prostoru datového skladu a jsou přístupna pro vytěžování - pokládání dotazů. Data mohou být natahována ve stejném tvaru jaké mají cílové struktury, nebo mohou být natažena v předzpracovaném tvaru do takzvaných multidimenzionálních tabulek, které obsahují předpřipravené podklady pro rychlé odezvy na dotazy zpracované podle jednotlivých dimenzí.

Na vykonání procesu ETL je možné používat specializované nástroje od externích dodavatelů nebo interně vyvinuté nástroje. Z konkrétních nástrojů lze uvést Microsoft DTS (*Data Transformation Services*), Oracle Warehouse Builder a mnoho dalších.

Plnění datových skladů probíhá ve dvou fázích:

- prvotní naplnění datového skladu v období implementace,
- cyklické plnění v období provozu datového skladu.

V prvotní fázi jsou do datového skladu ukládána i historická data získaná z provozních informačních systémů. Rozhodnutí o přesunu a využívání těchto dat je závislé na minulých změnách v provozních systémech a vhodnosti těchto dat k analýzám. V cyklickém plnění jsou data do datového skladu již pouze doplňována.

3.1.4 Uložení dat v OLAP systémech

V oblasti provozních systémů převažuje v oblasti uložení dat relační databázová technologie, v případě OLAP systémů však tato technologie již není tak jednoznačná.

Existují tři základní varianty uložení dat:
- relační databázový OLAP,
- multidimenzionální OLAP,
- hybridní OLAP.

Relační online analytické zpracování (ROLAP) získává údaje pro analýzy z relačního datového skladu. Tyto údaje z relačních databází se po zpracování předkládají uživatelům jako multidimenzionální pohled. Výhodou tohoto způsobu zpracování je uložení dat v relačních databázích, takže nevzniká problém s redundancí.

Pro **multidimenzionální online zpracování** (MOLAP) se získávají data buď z datového skladu nebo z operačních zdrojů. Mechanismus OLAP potom uloží analytická data ve vlastních datových strukturách a sumářích. Během tohoto procesu se ukládá tolik předběžných výsledků, kolik je z technického a časového hlediska možné. Hlavní výhodou je maximální výkon vzhledem k dotazům uživatelů, nevýhodou je redundance údajů, neboť tyto údaje jsou uloženy jednak v relační databázi, jednak v multidimenzionální databázi.

Hybridní online zpracování (HOLAP) je kombinací úložišť MOLAP a ROLAP, přičemž se využívají výhody jednotlivých typů úložišť a do značné míry se eliminují jejich nevýhody. Údaje zůstávají v relačních databázích a spočítané agregace se ukládají do multidimen-zionálních struktur.

3.1.5 Metadata

Metadata jsou data o datech, která popisují strukturu a obsah datového skladu. Metadata jsou pro technologie datových skladů důležitá, pomáhají administrátorům a uživatelům určit a po-chopit datové položky. Zároveň popisují transformaci zdrojových údajů do datového skladu. Na základě metadat musí být jasné, jaký konkrétní obsah datového skladu byl odvozen z provozních systémů. Metadata zvyšují a udržují kvalitu dat v datovém skladě a lze je rozdělit do několika typů:

- administrativní metadata,
- metadata koncových uživatelů,

- metadata pro optimalizaci.

Administrativní metadata popisují zdrojové databáze a jejich obsah, obchodní pravidla určující transformaci dat ze zdrojových systémů do datového skladu a objekty datového skladu.

Metadata koncových uživatelů pomáhají ve vytváření dotazů a se správnou interpretací výsledků. Pro uživatele může být také užitečná znalost definic a popis dat a hierarchie, které mohou existovat v rámci různých dimenzí.

Metadata pro optimalizaci jsou spravována za účelem usnadnění optimalizace návrhu a výko-nu datového skladu např. definice agregací, sběr statistik apod.

3.1.6 Analýza OLAP

Analýza OLAP slouží ke zpracování údajů uložených v datovém skladu do podoby vhodné pro koncového uživatele. OLAP pracuje s multidimenzionálním prostorem, který je definován metadaty. OLAP typicky poskytuje uživateli tyto analytické postupy:

- sestavení dotazu v multidimenzionálním prostoru,
- volba zobrazení výsledku dotazu v kontingenční tabulce, grafu,
- zobrazení různé úrovně detailu podle hierarchie v dimenzích nebo relace mezi dimenzemi,
- zvýraznění výjimek,
- umožňuje použít aritmetický, množinový aparát v multidimenzionálním prostoru.

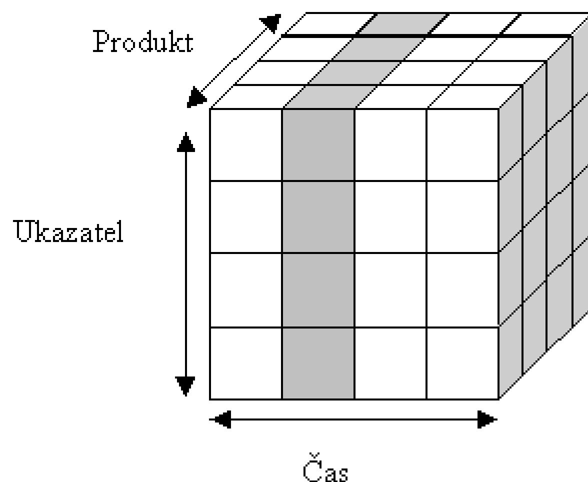
Výsledkem analýzy údajů bývá obvykle multidimenzionální datová struktura – *kostka*. Pojem datová kostka byl zaveden proto, že na data spravovaná OLAP serverem se lze dívat jako na určitou datovou krychli/kostku. Jednotlivé rozměry (dimenze) odpovídají různým úhlům pohledu na data. Data určité organizace můžeme například zkoumat z pohledu zákazníků, zaměstnanců, poboček či času. Na průsečíku rovin pohledu pak nalezneme konkrétní čísla.

Pro výpočet krychlí je potřebné vykonat velké množství výpočtů a agregací a to v reálném čase. Každá krychle má několik dimenzí, na rozdíl od geometrické krychle může mít multidimenzionální databázový model i více dimenzí než tři.

Krychle OLAP jsou vytvořeny na základě dvou druhu údajů:

- faktů,
- dimenzí.

Fakta jsou numerické měrné jednotky obchodování. Tabulka faktů je největší tabulka v databázi a obsahuje velký objem dat.
Dimenze obsahují logicky nebo organizačně hierarchicky uspořádané údaje. Jsou to vlastně textové popisy obchodování. Tabulky dimenzí jsou menší než tabulky faktů a data v nich se nemění tak často. Příkladem používaných dimenzí je dimenze časová, produktová, geografická atd. Tabulky dimenzí obvykle obsahují stromovou strukturu. Například dimenze vytvořená na základě organizační struktury se člení na jednotlivé úrovně podle konkrétního uspořádání v organizaci.



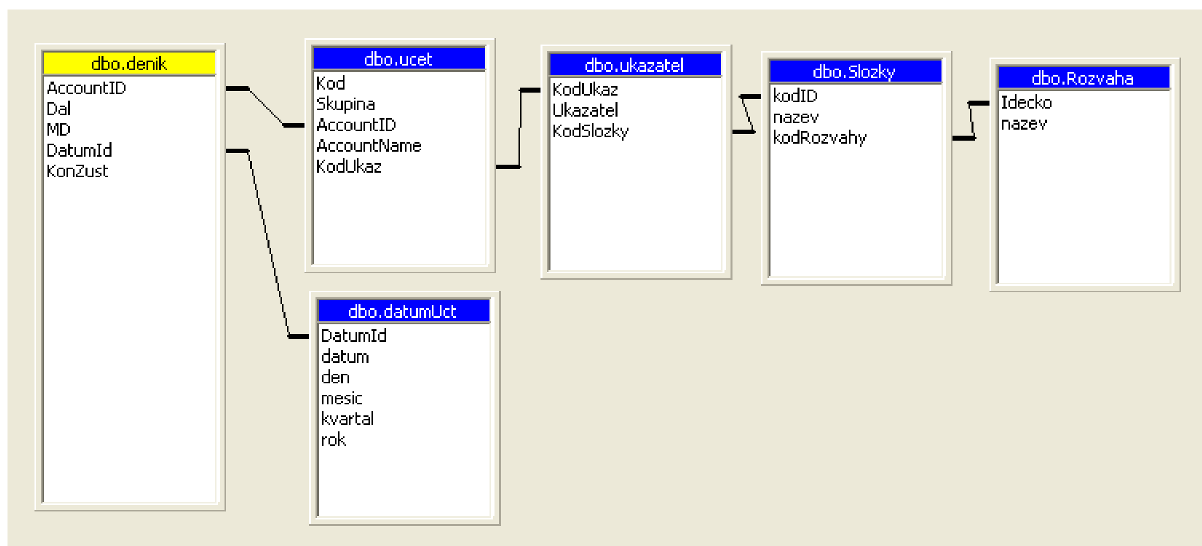
Obr.č.6 Obecné schéma datové kostky

Tabulky faktů a dimenzí mohou vytvářet různá schémata. Nejčastěji používané je hvězdicové (*star schema*) nebo schéma sněhové vločky (*snoflake schema*).

Hvězdicové schéma se skládá z tabulky faktů obsahující vazby na tabulky dimenzí, mezi kterými neexistuje relační propojení. Toto schéma je velice jednoduché a pochopitelné a posky-tuje vysoký dotazovací výkon.

Obr.č.7 Příklad hvězdicového schématu (žlutě tabulka faktů)

Schéma **sněhové vločky** obsahuje některé dimenze složené z mnoha relačně svázaných tabulek. Tento model umožňuje rychlejší zavedení údajů oproti předchozímu schématu, ale má podstatně nižší dotazovací výkon.



Obr.č.8 Příklad schématu sněhová vločka (žlutě tabulka faktů)

Základní operace umožňující analýzy v OLAP systémech:

- *drill-down* – umožňuje uživateli ve zvolených instancích jisté agregační úrovně nastavit nejnižší agregační úroveň
- *roll-up* – ve zvolených instancích agregační úrovně nastavuje vyšší agregační úroveň
- *pivoting* – umožňuje otáčet datovou krychlí, tj.měnit úhel pohledu na data na úrovni prezentace obsahu datového skladu
- *slicing* – dovoluje provádět řezy datovou kostkou, tj. nalézt pohled, v němž je jedna dimenza fixována v jisté instanci na jisté agregační úrovni
- *dicing* – obdoba slicingu, umožňuje nastavit takový filtr pro více dimenzí

3.2 Využití datových skladů

Datový sklad představuje především velký objem skrytých informací a jeho využití se neomezuje pouze na jednu oblast manažerských informačních systémů.

Mezi nejdůležitější oblasti využití datových skladů jsou:

- operativní dotazy - předem nepřipravené dotazy na určité hodnoty,
- sestavy - standardní generované dávkově nebo operativní vytvářené podle potřeby,
- multidimenzionální analýza - rychlé prohlížení sumarizovaných dat z různých pohledů
- statistické analýzy,
- finanční analýzy,
- analýzy časových řad a tvorba předpovědí,
- vizualizace dat - prohlížení dat v dynamicky provázaných grafech,
- data mining - specializované techniky pro zpracování velkých objemů dat a hledání skrytých závislostí.

4 Úvod do In-Memory Data Management