

DBS – transakční zpracování

Michal Valenta

Katedra softwarového inženýrství FIT
České vysoké učení technické v Praze
©Michal Valenta, 2014

BI-DBS, ZS 2014/15

<https://edux.fit.cvut.cz/courses/BI-DBS/>



Transakční zpracování

- Dva základní požadavky na DBMS :
 - ▶ chránit data – ve smyslu odolnosti vůči různým haváriím serveru
 - ▶ poskytnout korektní, rychlý a asynchronní přístup většímu množství současně pracujících uživatelů.
- Řešení
 - ▶ komponenta řízení soubežného (paralelního) zpracování (concurrency control)
 - ▶ komponenta zotavení z chyb (recovery)

Transakční zpracování

Řízení souběžného zpracování

Zajišťuje uživatelům, že každý vidí konzistentní stav databáze bez ohledu na to, že více uživatelů přistupuje asynchronně ke stejným údajům.

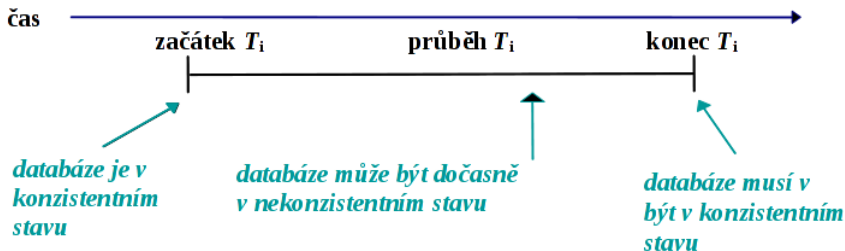
Zotavení z chyb

Zajišťuje, že stav databáze není narušen v případě chyby software, systému, nebo fyzického média v průběhu zpracování úlohy měnící data v databázi. Důsledkem takového incidentu nesmí být nekonzistence databáze.

Transakční zpracování

Řešením je transakce:

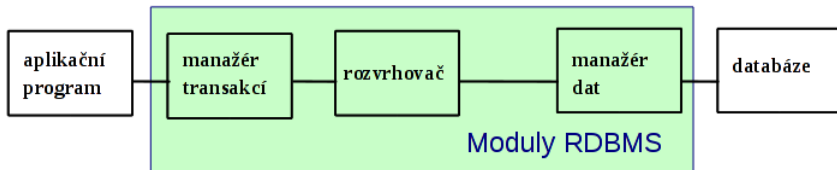
vhodná programová jednotka a vhodné mechanismy, které zabezpečí, že po skončení akce (korektním i nekorektním) zůstane databáze konzistentní (platí všechna IO definovaná ve schématu).



Moduly RDBMS

RDBMS musí zaručit:

- Transparentnost paralelního zpracování transakce.
- Atomicitu vzhledem k chybám.



Co je transakce?

Vhodná “logická” jednotka práce.

Obvykle se skládá z několika (i mnoha) dílčích operací.

Příklady transakcí

- Převod peněz z jednoho účtu na druhý.
- Přehlášení na termín zkoušky.

Začátek a konec transakce

Hranice transakce :

konec transakce

- Explicitní
 - ▶ COMMIT (potvrzení)
 - ▶ ROLLBACK (zrušení)
- Implicitní
 - ukončení session (záleží na klientovi zda commit nebo rollback).

začátek transakce

Je obvykle vymezen skončením transakce předchozí nebo vznikem session.

Pozor na nastavení klienta! Často bývá použit režim Autocommit On.

Obnova databáze po pádu

Využívá se transakční žurnál (log).

V transakčním logu jsou “změnové vektory”.

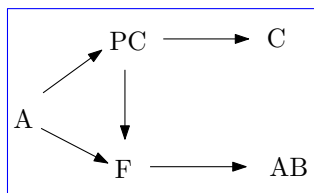
Operace použité při obnově

- UNDO
- REDO

Informace z transakčního žurnálu se používají **pouze** pro obnovu databáze po chybě.

Pro operaci **ROLLBACK** a zajištění tzv. **read consistency** se používají jiné datové struktury.

Stavový diagram transakce



- **aktivní (Active)** - od začátku (probíhají DML příkazy)
- **částečně potvrzený (Partially Committed)** - po provedení poslední operace transakce
- **potvrzený (Committed)** - po úspěšném zakončení, tj. po potvrzení operace COMMIT
- **chybný (Failed)** - v normálním průběhu transakce nelze pokračovat
- **zrušený (ABorted)** - po skončení operace ROLLBACK, tj. uvedení databáze do stavu před započítím transakce

Vlastnosti transakcí – ACID

ACID vlastnosti transakce:

- atomicita – (**Atomicity**) - transakce musí buď proběhnout celá nebo vůbec,
- konzistence – (**Consistency**) - transakce transformuje databázi z konzist. stavu do jiného konzist. stavu,
- nezávislost – (**Independence**) - dílčí efekty jedné transakce nejsou viditelné jiným transakcím,
- trvanlivost – (**Durability**) - efekty úspěšně ukončené (potvrzené) transakce jsou trvale uloženy(persistence).

Žurnál a spol.

- database buffer cache
- žurnál obsahuje sekvenci změnových vektorů <XID, pageID, offset, length, old data, new data>
- další informace v žurnálu - commit, rollback, checkpoint
- LSN (Log Sequence Number) - id změnového vektoru
- checkpoint, checkpoint number (System Change Number - SCN)
- WAL (Write Ahead Log) princip

Žurnál a přidružená infrastruktura umožňuje implementaci Atomicity a Durability u transakčního zpracování

Zotavení z chyby - třídy možných chyb

Globální chyby (mají vliv na více transakcí)

- Spadnutí systému serveru (např. výpadek proudu), důsledkem je obecně ztráta obsahu vyrov. pamětí.
- Chyby systémové, které ovlivňují transakce, avšak nikoli celou databázi (např. uváznutí, odumření komunikace klienta se serverem).
- Chyby médií (např. incident na disku), které zapříčiní ohrožení databáze, nebo její části,

Lokální chyby (v jedné transakci).

- Logické chyby, které by měly být “odchyceny” a ošetřeny na úrovni transakce explicitním vyvoláním operace ROLLBACK (pokus o porušení IO při zápisu do DB, dělení nulou při výpočtu).

Zotavení z chyby - po restartu systému

synchronizační body

časové známky (v žurnálu a v hlavičkách db souborů); slouží k nalezení místa (v žurnálu) odkud je třeba začít s rekonstrukcí databáze

Požadavky:

- Na transakce, jejichž stav bude v důsledku incidentu nedefinovaný je nutné použít ROLLBACK.
- Transakce, které byly potvrzené před tím než nastala chyba systému, avšak které nebyly dokončeny fyzicky (např. odeslání vyrovnávacích pamětí na disk), je nutné zopakovat a uložit do datových souborů.

Technicky má obnova systému po pádu dvě fáze:

- 1 Roll Forward – přehrání transakčního žurnálu (a obnova vyrovnávací paměti)
- 2 Roll Back – odvolání transakcí, které nebyly v době pádu dokončeny.

Zotavení z chyby médií

Oracle: záleží na módu v jakém databázi provozujeme (archivní / nearchivní):

archivní mód:

- Vystavení celé databáze (nebo jen chybějících částí) ze záložní kopie (Backup).
- Použití žurnálu k REDO všech ukončených transakcí až do té chvíle, kdy ještě žurnál poskytuje potřebné informace.
- Tento postup umožňuje i tzv PITR (Point In Time Recovery)

nearchivní mód:

- Buď se můžeme vrátit k poslední plné záloze systému (tedy zpátky v čase).
- Nebo obětujeme data, která byla poškozena.

Paralení zpracování transakcí

- Předpoklady: ploché transakce, objekty atomické, FETCH, resp. OUTPUT přesunou stránku vyrovnávací paměti z/na disk

READ(X,x)

přiřazuje hodnotu atributu X lokální proměnné x. Není-li stránka s danou hodnotou ve VP, provede se FETCH(X), následuje přiřazení hodnoty X z VP proměnné x.

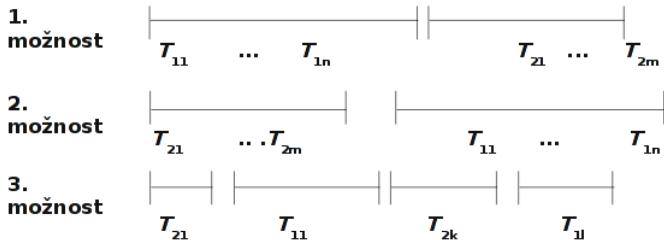
WRITE(X,x)

přiřazuje hodnotu lokální proměnné x atributu X ve vyrovnávací paměti. Operace má opět dvě fáze - není-li žádaná stránka ve vyrovnávací paměti, provede se FETCH(X), následuje přiřazení proměnné x atributu X ve vyrovnávací paměti.

Prokládání transakcí, rozvrh, rozvrhovač

Transakce T_1 a T_2 se skládají z dílčích operací $T_1 = \{T_{11} \dots T_{1n}\}$ a $T_2 = \{T_{21} \dots T_{2m}\}$

Možnosti prokládání:



- Stanovení pořadí provádění dílčích akcí více transakcí v čase nazveme **rozvrhem**. Maximalizace paralelismu zpracování, tj. vytváření rozvrhu, je věcí **rozvrhovače**.

Ztráta aktualizace

- nebezpečí **ztráty aktualizace** (při “prokládání” transakcí)

T1	T2	stav
READ(x) $x := x - 5$		$X = 80$ zrušení 5 rezervací
	READ(x) $x := x + 4$	$x = 80$ přidání 4 rezervací
WRITE(x)		$X = 75$
	WRITE(x)	$X = 84$

Ztráta aktualizace

Přestože by hodnota X v databázi měla být 79, je 84

Ztráta aktualizace

- nebezpečí **dočasně aktualizace** (při chybě systému)

T1	T2
READ(x) x:=x-5 WRITE(x)	
	READ(x) x:=x+4 WRITE(x)
READ(z) –chyba transakce	

Chyba transakce

Po provedení operace ROLLBACK u transakce T1 bude aktualizace provedená transakcí T2 založena na posléze odvolaných změnách takže výsledek bude chybný.

Problémy s paralelním zpracováním

- nebezpečí **nekorektního použití agregační funkce**

T1	T3	stav
	sum:=0 READ(a) sum:=sum+a	čítač počtu rezervací
READ(x) x:=x-a WRITE(x)		
WRITE(z)	READ(x) sum:=sum + x READ(z) sum:=sum + z	čte po změně X čte před změnou X

Problémy s paralelním zpracováním

neopakovatelné čtení

- Transakce T1 provede SELECT a použije načtené hodnoty.
- Transakce T2 poté změní hodnoty některých řádků.
- Když transakce T1 později provede tentýž select, dostane jiné hodnoty.

fantóm

Jako v předchozím případě, pouze s tím rozdílem, že transakce T2 smaže nebo přidá řádky. Ve druhém dotazu obdrží tedy transakce T1 jinou sadu dat.

Stupně izolace

	Úroveň izolace			
Anomálie	0	1	2	3
dočasná aktualizace	x	-	-	-
neopakovatelné čtení	x	x	-	-
fantóm	x	x	x	-

- 0 ... read uncommitted
- 1 ... read committed
- 2 ... repeatable read
- 3 ... serializable

Uspořádatelnost rozvrhu

- Sériové rozvrhy zachovávají operace každé transakce pohromadě. Pro N transakcí existuje $N!$ různých sériových rozvrhů.
- Lze vytvořit i rozvrh, kde jsou operace navzájem prokládány – **paralelní rozvrh**.
- Efekt paralelního zpracování transakcí má být stejný jako podle nějakého sériového rozvrhu.
- Rozvrh je **korektní**, když je v nějakém smyslu ekvivalentní kterémukoliv sériovému rozvrhu.
- O transakčním zpracování, které zaručuje tuto vlastnost se říká, že **zaručuje uspořádatelnost**.

Uspořádatelnost rozvrhu

- Definici uspořádatelnosti založíme na kompatibilitě operací READ a WRITE:
- Dvě operace jsou konfliktní, jestliže výsledky jejich různého sériového volání nejsou ekvivalentní.
- Operace, které nejsou konfliktní nazýváme kompatibilní.

kompatibilita	$READ_j(A)$	$WRITE_j(A)$
$READ_i(A)$	+	-
$WRITE_i(A)$	-	-

Uspořádatelnost rozvrhu

Uspořádatelnost rozvrhu

Máme-li rozvrhy $S1$ a $S2$ pro množinu transakcí $T = T1, \dots, TN$, pak $S1$ s $S2$ jsou ekvivalentní (vzhledem ke konfliktům), jsou-li splněny dvě podmínky:

- Jestliže se v prvním rozvrhu vyskytuje $READ(A)$ v T_i a tato hodnota vznikla z $WRITE(A)$ v transakci T_j , potom totéž musí být zachováno v druhém rozvrhu.
- Jestliže se v prvním rozvrhu vyvolá poslední operace $WRITE(A)$ v T_i , pak totéž musí být i v druhém rozvrhu.

Jinak řečeno: Relativní pořadí konfliktních operací nad stejnými objekty je v obou rozvrzích stejné.

Uspořádatelnost – příklad 1

T4: {READ(A), akce1(A), WRITE(A), READ(B), akce2(B), WRITE(B)}

T5: {READ(A), akce3(A), WRITE(A), READ(B), akce4(B), WRITE(B)}

S3	
T4	T5
READ(A) AKCE1(A) WRITE(A)	
	READ(A) AKCE3(A) WRITE(A)
READ(B) AKCE2(B) WRITE(B)	
	READ(B) AKCE4(B) WRITE(B)

S4	
T4	T5
READ(A) AKCE1(A)	
	READ(A) AKCE3(A) WRITE(A) READ(B)
WRITE(A) READ(B) AKCE2(B) WRITE(B)	
	AKCE4(B) WRITE(B)

$S1:\{T4,T5\}$ a $S2:\{T5,T4\}$.

jsou sériové rozvrhy.

S3 a S4 nejsou sériové.

Zřejmě $S3 \equiv S1$, $S3 \neq S2$.

Rozvrh je uspořádatelný, jestliže existuje sériový rozvrh s ním ekvivalentní.

Uspořádatelnost rozvrhu

- O rozvrhu jsme řekli, že je **uspořádatelný**, jestliže existuje sériový rozvrh s ním ekvivalentní.
- Mohou však existovat rozvrhy, které nejsou ekvivalentní se žádným sériovým rozvrhem podle této definice a přesto produkují stejný výsledek jako nějaký sériový rozvrh
- Existují možnosti definovat smysluplná kritéria korektnosti paralelních rozvrhů, která vůbec nejsou založena na pojmu uspořádatelnost.

Precedenční graf

Precedenční graf rozvrhu

- Jde o orientovaný graf $\{U, H\}$
- $U = \{T_i \mid i = 1, 2, \dots, n\}$
- $H = \{h_{ik}(A)\}$
- $h_{ik}(A)$... vzhledem k manipulacím s objektem A vede hrana od uzlu T_i k uzlu T_k , čímž říkáme, že rozvrh může být ekvivalentní pouze s takovým sériovým rozvrhem, kde T_i předchází T_k .

Konstrukce precedenčního grafu

Konstrukce precedenčního grafu rozvrhu S pro $\{T_i, T_k\}$

Od uzlu T_j povede hrana k uzlu T_k , jestliže:

- T_j volá $WRITE(A)$ před tím, než T_k volá $READ(A)$
(v T_k se čte z databáze hodnota A , vzniklá v T_j)
- T_j volá $READ(A)$ před tím, než T_k volá $WRITE(A)$
(v T_j se čte z Db hodnota A , dříve, než se v T_k změní)
- poslední $WRITE(A)$ v T_j předchází
poslednímu volání $WRITE(A)$ v T_k

Precedenční grafy – příklad 1

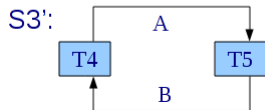
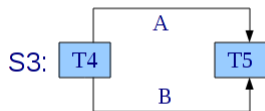
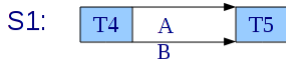
T4: {READ(A),akce1(A),WRITE(A),READ(B),akce2(B),WRITE(B)}

T5: {READ(A),akce3(A),WRITE(A),READ(B),akce4(B),WRITE(B)}

S1:{T4,T5} a S2:{T5,T4} jsou sériové rozvrhy.

S3	
T4	T5
READ(A) AKCE1(A) WRITE(A)	
	READ(A) AKCE3(A) WRITE(A)
READ(B) AKCE2(B) WRITE(B)	
	READ(B) AKCE4(B) WRITE(B)

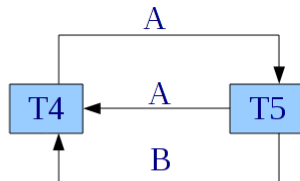
S3'	
T4	T5
READ(A) AKCE1(A) WRITE(A)	
	READ(A) AKCE3(A) WRITE(A) READ(B) AKCE4(B) WRITE(B)
READ(B) AKCE2(B) WRITE(B)	



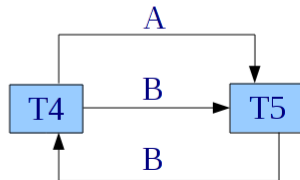
Precedenční grafy - příklad 2

S 4		S 4'	
T ₄	T ₅	T ₄	T ₅
READ(A) akce1(A)	READ(A) akce3(A)	READ(A) akce1(A)	READ(A) akce3(A)
WRITE(A) akce2(B)	WRITE(A) akce4(B)	WRITE(A) akce2(B)	WRITE(A) akce4(B)
WRITE(B)	READ(B)	WRITE(B)	READ(B)
	WRITE(B)		WRITE(B)

S4:



S4':



Testování uspořadatelnosti

Tvrzení 1

Rozvrh je uspořadatelný, (tedy ekvivalentní některému sériovému rozvrhu), jestliže v jeho precedenčním grafu neexistuje cyklus.

Podle tohoto Tvrzení

rozvrh S4 není uspořadatelný,
rozvrh S3 je uspořadatelný.

Tvrzení 2

Dva rozvrhy jsou ekvivalentní, jestliže mají stejný precedenční graf.

Uzamykací protokoly

Testování uspořadatelnosti jakéhokoliv rozvrhu není to nejlepší pro praxi. Časové nároky takového přístupu by zřejmě přesahovaly rozumnou míru.

Přístup z druhé strany:

Konstruovat transakce podle předem daných pravidel tak, že za určitých předpokladů bude každý jejich rozvrh uspořadatelný. Soustavě takových pravidel se obecně říká protokol.

Uzamykací protokoly

Nejznámější protokoly jsou založeny na dynamickém zamykání a odmykání objektů v databázi.

Jednoduchý model

Objekt může být v daném čase uzamčen nejvýše jednou transakcí - Lock(A), Unlock(A)

Transakce, která uzamkla objekt, má právo na něm provádět operace READ a WRITE (a další operace).

Legální rozvrh

Legální rozvrh

- objekt bude nutné mít v transakci uzamknutý, kdykoliv k němu chce tato transakce přistupovat
- transakce se nebudou pokoušet uzamknout objekt již uzamknutý jinou transakcí (čekají na uvolnění zámku)

Poznámka:

Samotná legálnost rozvrhu nezaručuje uspořádatelnost.

Uváznutí transakcí

V případech některých rozvrhů může nastat **uváznutí (deadlock)**.

Uváznutí lze řešit tak, že provedeme ROLLBACK jedné transakce. Co tato uzamkla, bude odemknuto, čímž se druhá transakce odblokuje.

T11

LOCK(B)
READ(B)
akce5(B)
WRITE(B)

čeká →

čeká →

LOCK(A)
UNLOCK(B)
READ(A)

T12

LOCK(A)
READ(A)
LOCK(B)
READ(B)
UNLOCK(B)
A:=A+B
WRITE(A)

Dobře formované transakce

Omezíme se dále na tzv. **dobře formované transakce**, které podporují přirozené požadavky na transakce :

Dobře formované transakce

- transakce zamyká objekt, chce-li k němu přistupovat
- transakce nezamyká objekt, když ho již zamkla
- transakce neodmyká objekt, který nezamkla
- na konci transakce nezůstane žádný objekt zamčený

Stále ještě neznáme postačující podmínku pro to, aby všechny **legální rozvrhy pro dobře formované transakce** byly uspořádatelné.

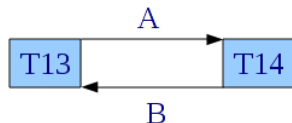
Konzistence databáze totiž není zaručena tím, že objekt odemkneme bezprostředně po manipulaci s ním. Zavedeme dvoufázové transakce.

Motivace pro dvoufázová transakce

T13	T14
LOCK(A) READ(A) WRITE(A) UNLOCK(A)	
	LOCK(A) READ(A) WRITE(A) UNLOCK(A) READ(B) LOCK(B) READ(B) WRITE(B) UNLOCK(B)
LOCK(B) READ(B) WRITE(B) UNLOCK(B)	

Rozvrh je legální, transakce jsou dobře formované.

Precedenční graf rozvrhu:



Rozvrh přesto není uspořadatelný.

Pomůže, když uvažované transakce budou **dvoufázové**.

Prohodíme-li UNLOCK(A) s LOCK(B) v T13 bude tato transakce dvoufázová.

Dvoufázový uzamykací protokol

Uzamykací protokol je množina pravidel, které udávají, kdy transakce bude uzamykat a odmykat objekty databáze.

T15

LOCK(B)
READ(B)
AKCE5(B)
WRITE(B)
LOCK(A)
READ(A)
AKCE7(A)
WRITE(A)
UNLOCK(B)
UNLOCK(A)

Dvoufázová transakce :

- 1. fáze - uzamyká se , nic neodemyká
- 2. fáze - od prvního odemknutí, do konce se už nic nezamyká

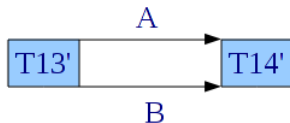
Tvrzení:

Jestliže všechny transakce v dané množině transakcí T jsou **dobře formované** a **dvoufázové**, pak každý jejich **legální rozvrh** je **uspořadatelný**.

Zavedením dvoufázových transakcí se vzdáváme některých uspořadatelných rozvrhů

T13'	T14'
LOCK(A) READ(A) Akce, WRITE(A) UNLOCK(A)	
	LOCK(A) READ(A) Akce, WRITE(A)
LOCK(B) READ(B) Akce, WRITE(B) UNLOCK(B)	
	LOCK(B) READ(B) UNLOCK(A) Akce, WRITE(B) UNLOCK(B)

Precedenční graf tohoto rozvrhu:



Je tedy uspořadatelný, ač T13' není dvoufázová.

Nevylučujeme existenci uspořadatelného legálního rozvrhu transakcí, které nejsou dvoufázové.

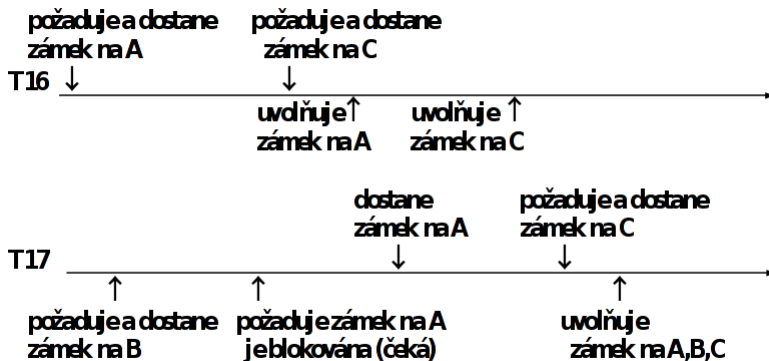
Rozvrh - praktická ukázka jak to funguje

T16	T17
LOCK(A) READ(A) v praxi často	LOCK(B) READ(B)
LOCK(C) READ(C) WRITE(A) UNLOCK(A)	
	LOCK(A) READ(A) WRITE(B)
WRITE(C) UNLOCK(C)	
	LOCK(C) READ(C) WRITE(C) UNLOCK(A) UNLOCK(B) UNLOCK(C)

Jeden z uspořádatelných
dvoufázových rozvrhů T16 a T17.

v praxi často Transakce T17 jednou čeká na
uvolnění zámku pro A, jednou na
uvolnění zámku C (viz obrázek na
dalším slide).

Příklad z předchozího slide



Opět možné uváznutí

	T19		T20
1	LOCK(A)	4	LOCK(B)
2	READ(A)	5	READ(B)
3	WRITE(A)	6	WRITE(B)
	LOCK(B) READ(B) UNLOCK(A) WRITE(B) UNLOCK(B)		LOCK(A) READ(A) UNLOCK(B) WRITE(A) UNLOCK(A)

Detekce a řešení v praxi? Graf závislostí nebo timeout pro získání zámku.
Abort (rollback) jedné transakce umožní druhé pokračovat.

Závěry

- Dvofázový uzamykací protokol, legální rozvrhy a dobře formované transakce řeší nezávislost (Independence) transakční a souběžnou práci více uživatelů nad stejnými daty.
- Stupně izolace? Různé druhy zámků.
- Striktní dvofázový uzamykací protokol - všechno se odmyká až na konci transakce. Kam zmizelo souběžné zpracování? Proč to celkem funguje? Zamykání (obvykle) na úrovni řádků (zamknout lze i celou tabulku). Složitější struktury v jiných DB modelech (stromy, grafy)?
- Atomicity a Durability vlastnosti jsou zajištěny pomocí žurnálu a přidružené infrastruktury.
- Consistency vlastnost je zajišťovaná přímo v interních algoritmech DML operací a tím, že transakce může obsahovat více DML a SELECT příkazů.