Transakce a transakční zpracování

Michal Valenta

Katedra softwarového inženýrství
Fakulta informačních technologií
České vysoké učení technické v Praze

BI-DBS, LS 2021/2022

https://courses.fit.cvut.cz/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.
- V architektuře DB stroje jsou moduly/komponenty:
  - řízení souběžného (paralelního) zpracování (concurency control)
  - zotavení z chyb (recovery)

Transakční zpracování

Modul řízení souběžného zpracování (concurency control)

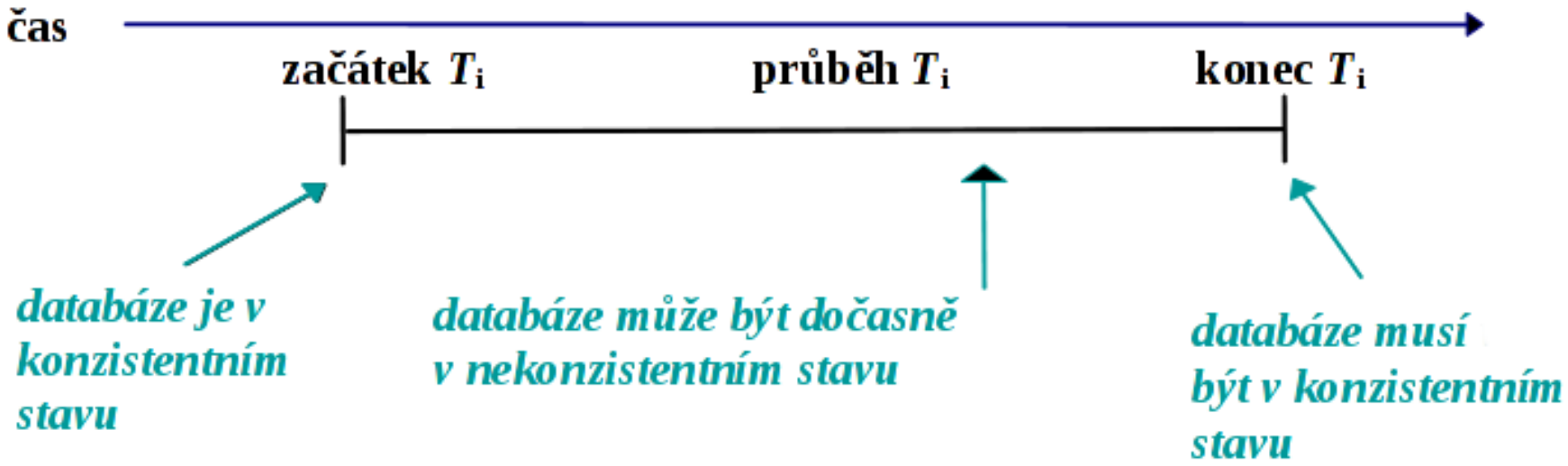
Zajišťuje uživatelům, že každý vidí konzistentní stav databáze bez ohledu na to, že více uživatelů přistupuje asynchronně ke stejným údajům.

Modul zotavení z chyb (recovery)

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.

Transakce

Transakce je
sekvence akcí, které spolu „logicky souvisí“ (tj. je vhodné je vnímat jako jeden celek), měnících stav databáze.



Transakční zpracování (dodržení vlastností ACID) zajistí, že po skončení transakce (úspěšném i neúspěšném) zůstane databáze konzistentní (platí všechna IO definovaná ve schématu).

Příklady transakcí – 1/2

Převod peněz (částka X) z účtu A na účet B

1. Z účtu A odečtu částku X

```
UPDATE ACCOUNT
SET AMOUNT = AMOUNT - X
WHERE ACCOUNT_ID = A;
```

2. Na účet B přičtu částku X

```
UPDATE ACCOUNT
SET AMOUNT = AMOUNT + X
WHERE ACCOUNT_ID = B;
```

3. Ukončení transakce: `COMMIT / ROLLBACK`.

Příklady transakcí – 2/2

Student S, předmět P – přehlášení z termínu T1 na T2

- IO: student nemůže být přihlášen na dva budoucí termíny z jednoho předmětu.
- Připustíme-li, že databáze může být v rámci transakce **dočasně** nekonsistentní, pak lze:
  1. Zapsat studenta na T2.
     Student bude zapsaný na oba termíny, tedy **dočasně a pouze v rámci transakce** porušuje IO.
  2. Pokud se zápis na T2 podaří, škrtnout studenta z T1.
- Tento postup je „bezpečný“ (změna termínu se podaří) i v případě souběžné práce více studentů snažících si zapsat zkoušky z předmětu P.
- Pokud nepřipustíme (dočasné) porušení IO v rámci transakce, mohlo by se stát, že student při pokusu o změnu termínů skončí tak, že nebude mít zapsaný žádný!

Začátek a konec transakce

Hranice transakce :

konec transakce

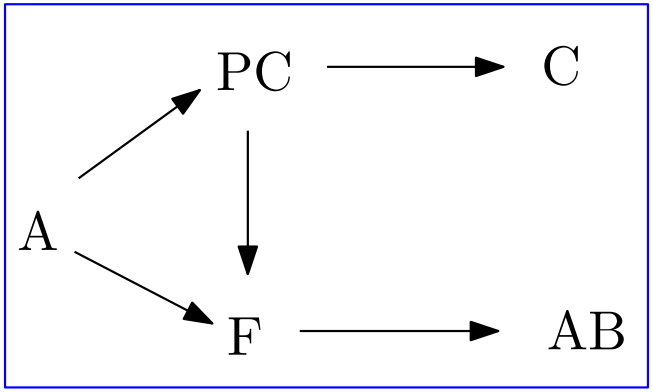
- Explicitní
  - COMMIT (potvrzení)
  - ROLLBACK (zrušení)
- Implicitní
  ukončení session (záleží na klientovi zda commit nebo rollback).

začátek transakce

Je obvykle vymezen skončením transakce předchozí nebo vznikem session.

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

Stavový diagram transakce



- aktivní (Active) - od začátku (probíhají DML příkazy)
- částečně potvrzený (Partially Commited) - po provedení poslední operace transakce
- potvrzený (Commited) - 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 započetím 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).

Obnova databáze po pádu

Využívá se transakční žurnál (log soubor).
V transakčním logu jsou „změnové vektory“.

Operace použité při obnově

- UNDO
- REDO

Poznámka: 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.

Moduly a struktury potřebné pro recovery

- database buffer cache (bloky stejné velikosti)
- žurnál obsahuje sekvenci **změnových vektorů** <transID, blockID, old data, new data>
- speciální změnové vektory pro operace commit, rollback a checkpoint
- checkpoint (synchronizace database buffer cache a DB bloků na disku) má jednoznačné checkpoint number (System Change Number - SCN)
- v Oracle je žurnál implementován formou (minimálně) dvou souborů pevné velikosti, které se cyklicky přepisují (událost log switch)

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:

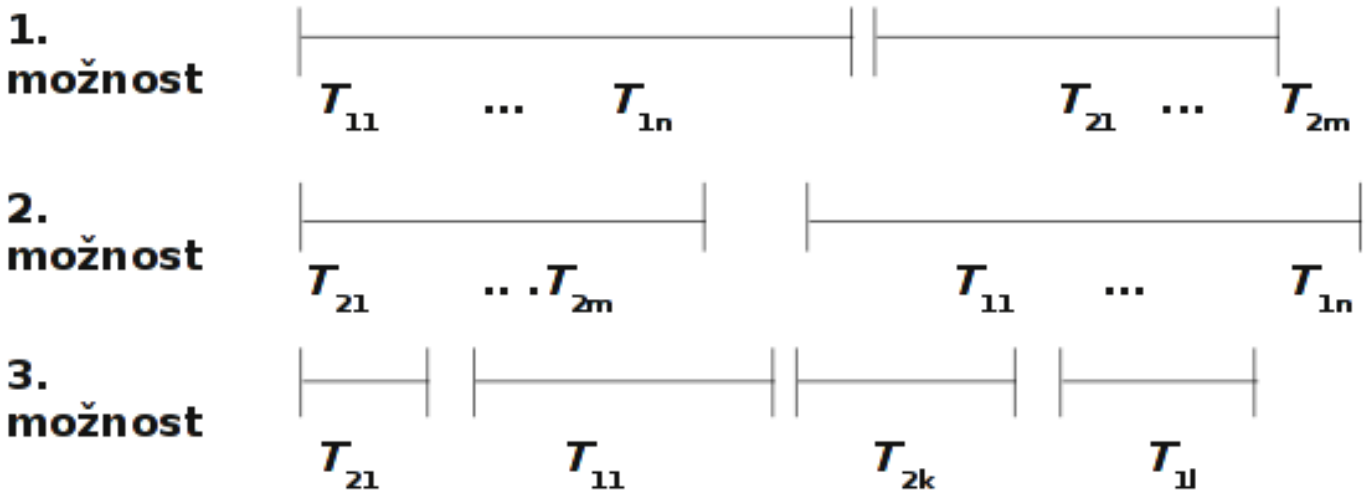
- Vystavení celé databáze (nebo jen chybějících částí) ze záložní kopie (Backup).
- Použití žurnálu k REDO všech ukončených transakcí až do té chvíle, kdy ještě žurnál poskytuje potřebné informace.
- Tento postup umožňuje i tzv PITR (Point In Time Recovery)

nearchivní mód:

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

Prokládání transakcí, rozvrh, rozvrhovač

Transakce T_1 a T_2 se skládají z dílčích operací $T_1 = \{T_{11} \ldots T_{1n}\}$ a $T_2 = \{T_{21} \ldots T_{2m}\}$
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 "prokládaní" 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 **dočasné 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

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 trasakce T1 jinou sadu dat.

Stupně izolace

Anomálie	Úroveň izolace			
	read uncommited	read commited	repeatible read	serializable
	0	1	2	3
Dočasná aktualizace	povoleno	zakázáno	zakázáno	zakázáno
Neopakovatelné čtení	povoleno	povoleno	zakázáno	zakázáno
Fantóm	povoleno	povoleno	povoleno	zakázáno

- standard nařizuje implicitně úroveň **serializable**
- v praxi se ovšem potkáte nejčastěji s úrovní **read commited**

Poznámka: PostgreSQL ani Oracle neumožní nižší úroveň než **read commited**, MySQL s použitím MyISAM storage engine umí jen **read uncommited**, engine InnoDB umí totéž, co PostgreSQL či Oracle.

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,...,TN, pak S1 s S2 jsou ekvivalentní (vzhledem ke konfliktům), jsou-li splněny dvě podmínky:

- Jestliže se v prvním rozvrhu vyskytuje READ(A) v Ti a tato hodnota vznikla z WRITE(A) v transakci Tj, potom totéž musí být zachováno v druhém rozvrhu.
- Jestliže se v prvním rozvrhu vyvolá poslední operace WRITE(A) v Ti, 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 $\{U, H\}$
- $U = \{T_i \mid i = 1, 2, ..., n\}$
- $H = \{h_{ik}(A)\}$
- $h_{ik}(A)$ ... 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 hraha 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

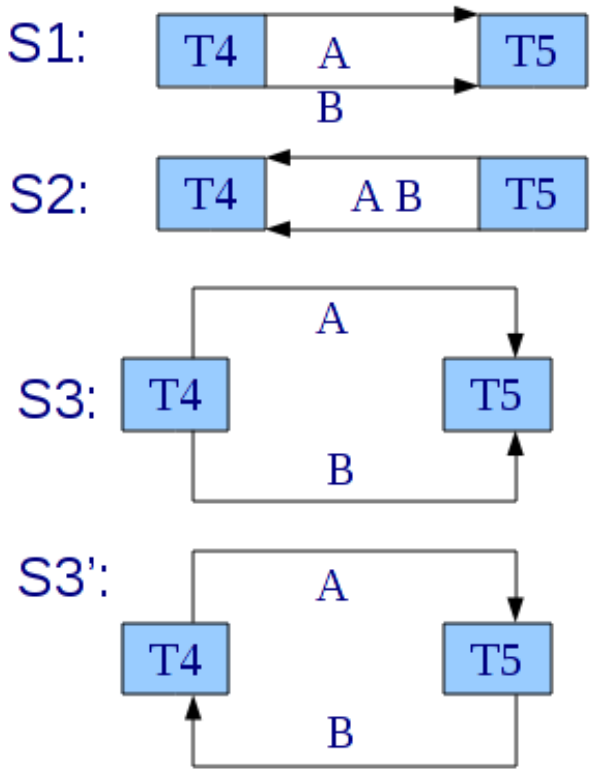
Precedenční grafy – příklad 1

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)

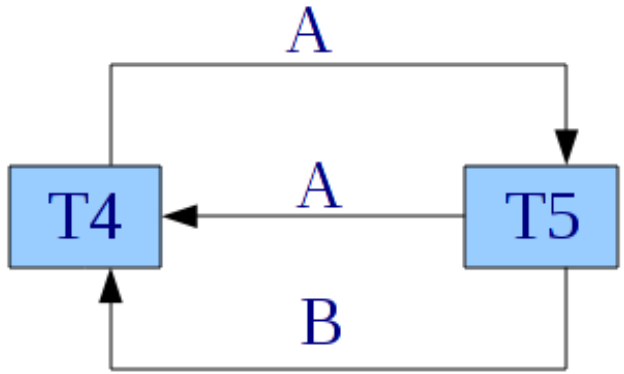
S3’	
T4	T5
READ(A) AKCE1(A) WRITE(A)	
	READ(A) AKCE3(A) WRITE(A) READ(B) AKCE4(B) WRITE(B)
READ(B) AKCE2(B) WRITE(B)	



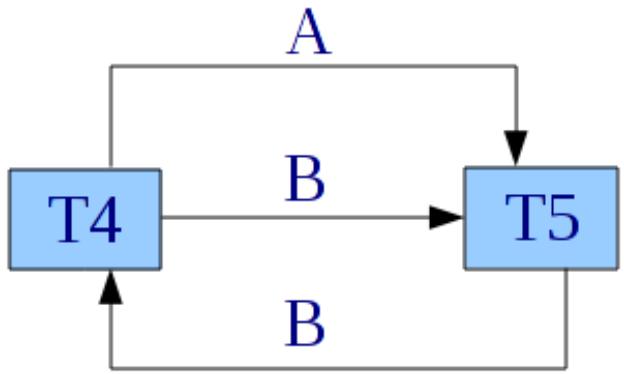
Precedenční grafy - příklad 2

S 4		*S 4'*	
T_4	T_5	T_4	T_5
READ(A)		READ(A)	
akce1(A)		akce1(A)	
	READ(A)		READ(A)
	akce3(A)		akce3(A)
	WRITE(A)		WRITE(A)
	READ(B)		READ(B)
WRITE(A)			akce4(B)
READ(B)		WRITE(A)	
akce2(B)		READ(B)	
WRITE(B)		Akce2(B)	
	akce4(B)		WRITE(B)
	WRITE(B)	WRITE(B)	

S4:



S4':



Testování uspořádatelnosti

Tvrzení 1

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

Podle tohoto Tvrzení

rozvrh S4 není uspořádatelný,
rozvrh S3 je uspořádatelný.

Tvrzení 2

Dva rozvrhy jsou ekvivalentní, jestliže mají stejný precendenční graf.

Uzamykací protokoly

Testování uspořádatelnosti 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 uspořádatelný.
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)

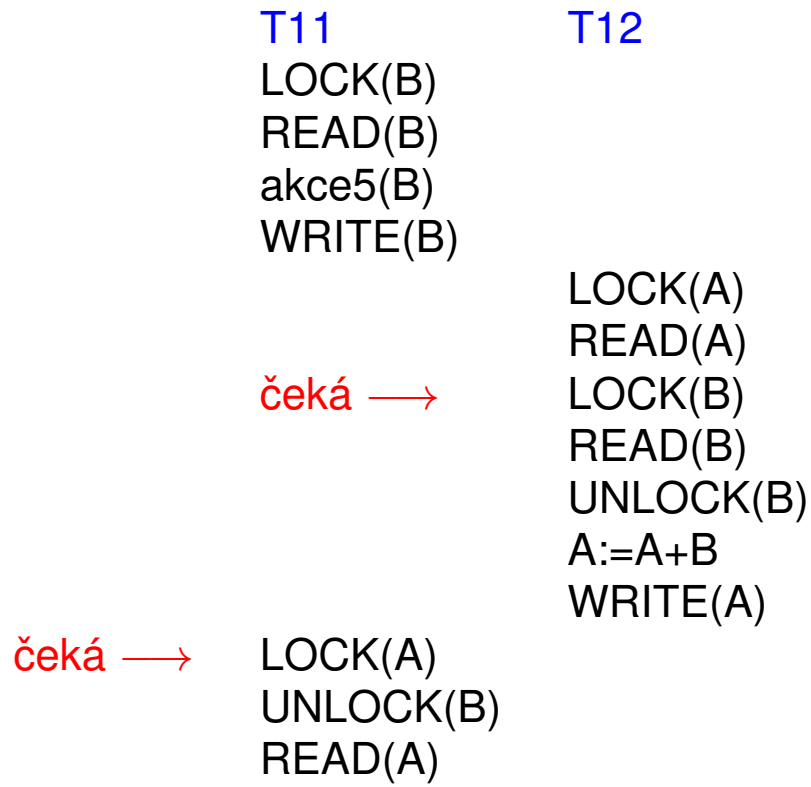
Poznámka:

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.



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

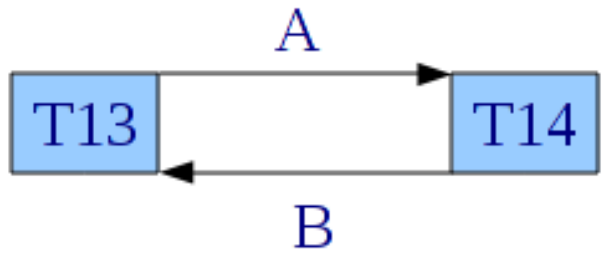
Stále ještě neznáme postačující podmínku pro to, aby všechny **legální rozvrhy pro dobře formované transakce** byly uspořádatelné. Konzistence databáze totiž není zaručena tím, že objekt odemkneme bezprostředně po manipulaci s ním. Zavedeme dvoufázové transakce.

Motivace pro dvoufázové transakce

T13	T14
LOCK(A) READ(A) WRITE(A) UNLOCK(A)	
	LOCK(A) READ(A) WRITE(A) UNLOCK(A) 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 uspořádatelný.

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

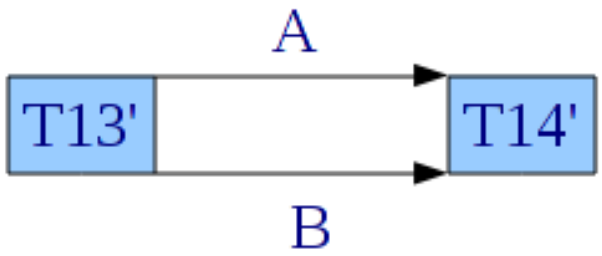
Tvrzení:

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

Zavedením dvoufázových transakcí se vzdáváme některých uspořádatelných 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 uspořádatelný, ač T13’ není dvoufázová.

Nevylučujeme existenci uspořádatelného legálního rozvrhu transakcí, které nejsou dvoufázové.

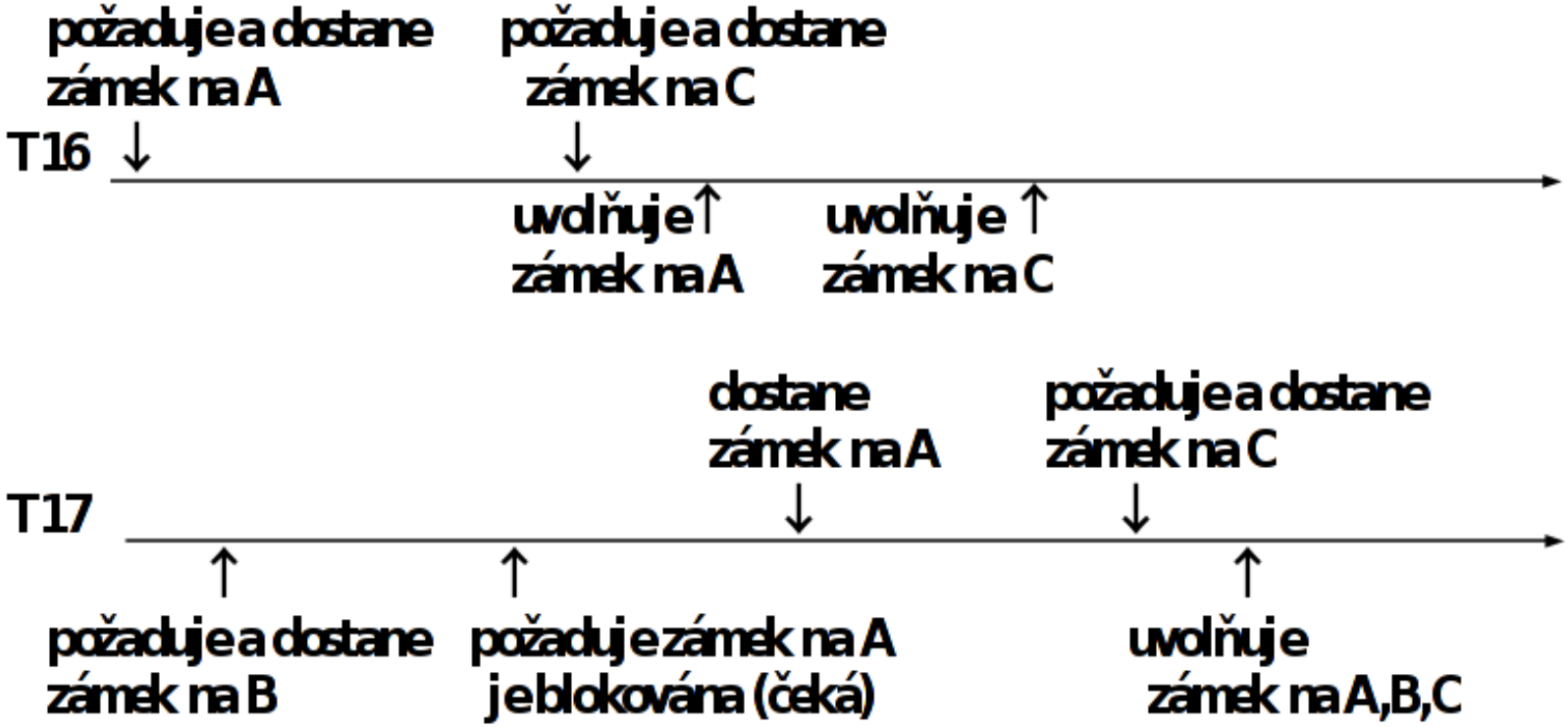
Rozvrh - praktická ukázka jak to funguje

T16	T17
LOCK(A) READ(A)	
	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.

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.

Důležité pojmy a koncepty

- transakce
- vlastnosti ACID
- stavový diagram transakce
- žurnál (co to je a k čemu slouží)
- úrovně izolace
- uzamykací protokol - co to je a k čemu slouží

Shrnující poznámky

- 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.
- Dvoufázový 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 dvoufázový 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)?