Kapitola 13: Transakce

- Koncept transakce
- Stavy transakce
- Implementace atomičnosti a trvanlivosti
- Souběžné spouštění
- Serializovatelnost

Koncept transakce

- Transakce je posloupnost operací (část programu), která přistupuje a aktualizuje (mění) data.
- Transakce pracuje s konzistentní databází.
- Během spouštění transakce může být databáze v nekonzistentním stavu.
- Ve chvíli, kdy je transakce úspěšně ukončena, databáze musí být konzistentní.
- Dva hlavní problémy:
  - Různé výpadky, např. chyba hardware nebo pád systému
  - Souběžné spouštění více transakcí

ACID vlastnosti

Pro zajištění integrity dat musí databázový systém zaručovat:

- **Atomičnost** (Atomicity) – buď všechny operace v transakci jsou provedeny nad databází nebo žádná z nich.
- **Konzistence** (Consistency) – běh jediné (izolované) transakce zachovává konzistenci databáze.
- **Izolovanost** (Isolation) – ačkoli může být více transakcí spouštěno současně, každá transakce nesmí vědět o ostatních současně běžících transakcích. Dočasné mezivýsledky transakce musí být skryté pro ostatní transakce, tj. pro každou dvojici transakcí T_i a T_j platí, že z pohledu transakce T_i je buď T_j dokončena před spuštěním T_i, nebo je T_j spuštěna až po dokončení T_i.
- **Trvanlivost** (Durability) – Po úspěšném dokončení transakce jsou všechny v databázi provedené změny uchovány i při případném výpadku systému.

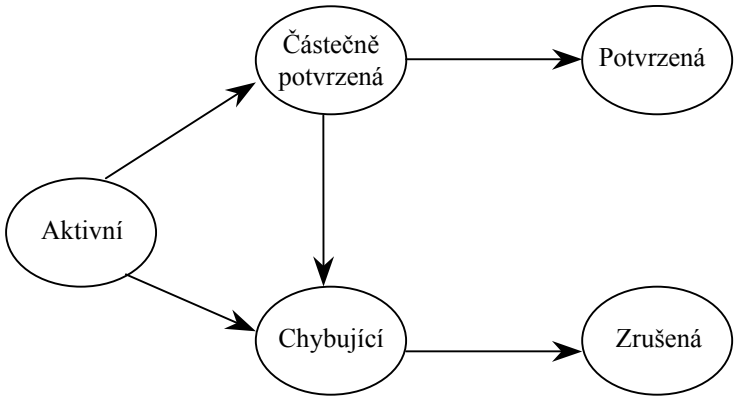
Příklad převodu finančních prostředků

- Transakce převádí \$50 z účtu *A* na účet *B*
  1. **čti(***A***)**
  2. *A := A – 50*
  3. **zapiš(***A***)**
  4. **čti(***B***)**
  5. *B := B + 50*
  6. **zapiš(***B***)**
- Požadavek konzistence – součet zůstatků na účtech *A* a *B* je nezměněn provedením transakce.
- Požadavek atomičnosti – pokud transakce skončí chybou po kroku 3 a před krokem 6, systém by měl zajistit, že provedené změny nebudou uloženy do databáze, jinak by byla v nekonzistentním stavu.

- Požadavek trvanlivosti – pokud byla uživateli již vrácena odpověď o úspěšném provedení transakce, všech provedené změny jsou stálé a musí přežít i výpadek databáze.
- Požadavek izolovanosti – pokud mezi kroky 3 a 6 je dovoleno jiné transakci číst částečně aktualizovanou databázi, bude mít k dispozici nekonzistentní databázi (součet *A+B* bude menší než by měl být).
  Izolovanosti můžeme velmi jednoduše dosáhnout použitím sériového spouštění transakcí, tj. jedna po druhé. Avšak současný běh více transakcí má významné výhody, což uvidíme později.

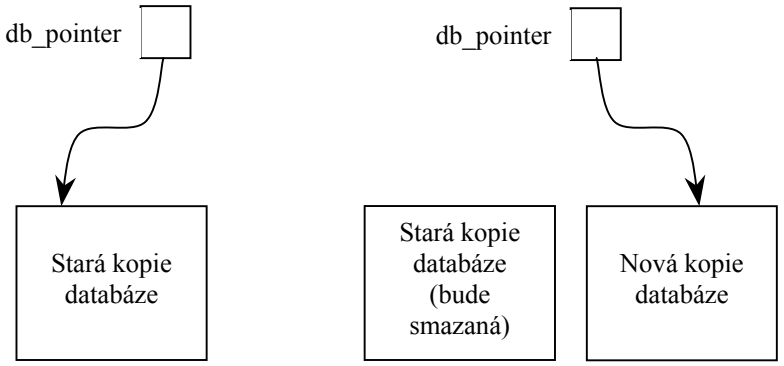
Stavy transakce

- **Aktivní** – počáteční stav; transakce zůstává v tomto stavu, dokud běží
- **Částečně potvrzená** (Partially Committed) – jakmile byla provedena poslední operace transakce
- **Chybující** (Failed) – po zjištění, že normální běh transakce nemůže pokračovat
- **Zrušená** (Aborted) – poté, co byla transakce vrácena (rolled back) a databáze byla vrácena do stavu před spuštěním transakce. Dvě možnosti po zrušení transakce:
  - Znovu spustit transakci – pouze pokud nedošlo k logické chybě
  - Zamítnout transakci
- **Potvrzená** (Committed) – po úspěšném dokončení



Implementace atomičnosti a trvanlivosti

- Databázový podsystém správy obnovy (recovery-management) implementuje podporu pro atomičnost a trvanlivost transakcí.
- Schéma *stínové databáze*:
  - Předpokládejme, že v jednu chvíli je aktivní pouze jedna transakce.
  - Ukazatel **db_pointer** vždy ukazuje na současnou konzistentní kopii databáze.
  - Všechny změny jsou prováděny ve *stínové kopii* databáze a **db_pointer** bude ukazovat na změněnou stínovou kopii, až transakce dosáhne stavu „částečně potvrzená“ a všechny změny byly uloženy na disk.
  - V případě chyby transakce je stará konzistentní kopie (ukazuje na ni **db_pointer**) použita a stínová kopie může být smazána.



(a) před změnou (b) po změně

- Toto schéma předpokládá, že disk nemůže selhat.
- Užitečné pro textové editory, ale velmi neefektivní pro velké databáze: spuštění transakce vyžaduje vytvoření kopie celé databáze.

Souběžné spouštění

- Více transakcí může být spuštěno současně. Výhody jsou:
  - Zvýšené využití procesoru a disku, které vede k vyšší transakční propustnosti: jedna transakce může používat procesor, zatímco jiná disk.
  - Snížená průměrná doba odezvy: krátké transakce nemusí čekat na dokončení dlouhých.
- Schémata pro řízení souběžnosti (Concurrency Control) – mechanizmy pro řízení interakcí (vzájemného působení) souběžných transakcí, které zamezují porušení konzistence databáze.

Plány

- **Plány** jsou posloupnosti, které určují časové pořadí provádění instrukcí souběžných transakcí
  - plán pro množinu transakcí musí obsahovat všechny operace prováděné těmito transakcemi
  - musí zachovávat pořadí instrukcí stejné jako v každé jednotlivé transakci

Příklady plánů

- Nechť T_1 převádí \$50 z účtu *A* na účet *B* a T_2 převádí 10% zůstatku *A* na *B*. Následující plán č.1 je sériové spuštění transakce T_2 po transakci T_1.

T_1	T_2
čti(*A*)	
A := A – 50	
zapiš(*A*)	
čti(*B*)	
B := B + 50	
zapiš(*B*)	
	čti(*A*)
	*temp := A * 0.1*
	A := A – temp
	zapiš(*A*)
	čti(*B*)
	B := B + temp
	zapiš(*B*)

- Nechť T_1 a T_2 jsou stejné transakce. Následující plán č.3 není seriový plán, ale je ekvivalentní plánu č.1.

T_1	T_2
čti(*A*)	
A := A – 50	
zapiš(*A*)	
	čti(*A*)
	*temp := A * 0.1*
	A := A – temp
	zapiš(*A*)
čti(*B*)	
B := B + 50	
zapiš(*B*)	
	čti(*B*)
	B := B + temp
	zapiš(*B*)

- V obou plánech je součet *A+B* zachovaný.
- Následující plán č.4 nezachovává součet *A+B*

T_1	T_2
čti(*A*)	
A := A – 50	
	čti(*A*)
	*temp := A * 0.1*
	A := A – temp
	zapiš(*A*)
	čti(*B*)
zapiš(*A*)	
čti(*B*)	
B := B + 50	
zapiš(*B*)	
	B := B + temp
	zapiš(*B*)

Serializovatelnost

- Základní předpoklady – každá transakce zachovává konzistenci databáze
- Tedy sériový plán zachovává konzistenci databáze
- Plán je serializovatelný, když je ekvivalentní sériovému plánu. Různé formy ekvivalence plánů vedou k následujícím pojmům:
  - Konfliktní serializovatelnost
  - Pohledová serializovatelnost
- Ignorujeme všechny instrukce kromě **čtení** a **zápisu** a předpokládáme, že transakce mohou provádět libovolné výpočty na datech v lokálních vyrovnávacích pamětech mezi čteními a zápisy. Naše zjednodušené plány se skládají pouze z operací **čtení** a **zápisu.**

Konfliktní serializovatelnost

- Instrukce I_i a I_j transakcí T_i a T_j jsou v konfliktu, když existuje nějaké Q, které je přistupované oběma instrukcemi I_i a I_j, a nejméně jedna z nich zapisuje Q.
- 1. I_i = ***čti(Q),*** I_j = ***čti(Q).*** I_i a I_j nejsou v konfliktu.
  2. I_i = ***čti(Q),*** I_j = ***zapiš(Q).*** I_i a I_j jsou v konfliktu.
  3. I_i = ***zapiš(Q),*** I_j = ***čti(Q).*** I_i a I_j jsou v konfliktu.
  4. I_i = ***zapiš(Q),*** I_j = ***zapiš(Q).*** I_i a I_j jsou v konfliktu.
- Intuitivně, konflikt mezi I_i a I_j vynucuje časové pořadí mezi nimi. Pokud I_i a I_j následují za sebou v plánu a nejsou v konfliktu, jejich výsledky zůstanou stejné, ikdyž byly v plánu prohozeny.
- Pokud plán S může být převeden na plán S' pomocí posloupnosti prohození nekonfliktních instrukcí, říkáme, že plány S a S' jsou *ekvivalentní podle konfliktu.*
- Říkáme, že plán S je *serializovatelný podle konfliktu*, jestliže je ekvivalentní podle konfliktu se sériovým plánem.
- Příklad plánu, který není serializovatelný podle konfliktu:

T_3	T_4
čti(Q)	
	zapiš(Q)
zapiš(Q)	

  V uvedeném plánu nejsme schopni vyměnit instrukce tak, že dostaneme sériový plán $<T_3,T_4>$ nebo $<T_4,T_3>$.
- Plán č.3 uvedený níže lze převést na plán č.1 (sériový plán), ve kterém T_2 následuje T_1. Převod je proveden posloupností výměn nekonfliktních instrukcí. Výsledně je plán č.3 serializovatelný podle konfliktu.

T_1	T_2
čti(A)	
zapiš(A)	
	čti(A)
	zapiš(A)
čti(B)	
zapiš(B)	
	čti(B)
	zapiš(B)

Pohledová serializovatelnost

- Nechť *S* a *S'* jsou dva plány pro stejnou množinu transakcí. *S* a *S'* jsou *pohledově ekvivalentní*, jestliže platí následující tři podmínky:
  1. pro každou datovou položku *Q*, když transakce T_i čte počáteční hodnotu *Q* v plánu *S*, potom transakce T_i také musí číst počáteční hodnotu *Q* v plánu *S'.*
  2. pro každou datovou položku *Q*, když transakce T_i provede **čti(**Q**)** v plánu *S* a hodnota je výsledkem transakce T_j, potom transakce T_i musí v plánu *S'* také číst hodnotu *Q*, která je výsledkem transakce T_j.
  3. pro každou datovou položku *Q*, když nějaká transakce provede poslední (závěrečné) **zapiš(**Q**)** v plánu *S*, pak také musí stejná transakce provést poslední **zapiš(**Q**)** v plánu *S'.*
- pohledová ekvivalence je také založena pouze na operacích **čti()** a **zapiš()**.
- Plán *S* je *pohledově serializovatelný*, když je pohledově ekvivalentní sériovému plánu.
- Každý plán, který je serializovatelný podle konfliktu, je také pohledově serializovatelný.
- Následující plán je pohledově serializovatelný, ale ne podle konfliktu:

T_3	T_4	T_6
čti(Q)		
	zapiš(Q)	
zapiš(Q)		
		zapiš(Q)

- Každý plán, který je pohledově serializovatelný, ale není serializovatelný podle konfliktu, používá slepé zápisy (blind writes) – zápis bez předchozího čtení.

Další definice serializovatelnosti

- Následující plán dává stejný výsledek jako sériový plán $<T_1,T_5>$, ale není ekvivalentní pohledově ani podle konfliktu se sériovým plánem.

T_1	T_2
čti(A)	
A := A – 50	
zapiš(A)	
	čti(B)
	B := B – 10
	zapiš(B)
čti(B)	
B := B + 50	
zapiš(B)	
	čti(A)
	A := A + 10
	zapiš(A)

- Rozhodnutí této ekvivalence vyžaduje analýzu výpočetních operací – jiných než čtení a zápis.