Databázové systémy

Tomáš Skopal

– transakce

* uzamykací protokoly
* alternativní protokoly
* zotavení

Osnova

- uzamykací protokoly
  - 2PL
  - striktní 2PL
  - uváznutí, prevence
  - fantom
- alternativní protokoly
  - optimistické řízení
  - časová razítka
- zotavení po havárii systému

Protokoly současného vykonávání transakcí (concurrency control protocols)

- rozvrhovač transakcí volí nějaký protokol (soubor pravidel pro rozvrhování) tak, aby byly zachovány požadované vlastnosti zpracování, jako
  - vysoký stupeň paralelizace (konkurence)
  - uspořádatelnost (konzistence, izolace)
  - zotavitelnost
  - atd.
- uzamykací protokoly (pesimistické řízení)
  - využívá se zámků, aby se předešlo konfliktům, nezotavitelnosti, ...
- alternativní protokoly
  - optimistické řízení
  - časová razítka

Proč protokol?

Rozvrhovač „nic dopředu neví“:
- požadavky na spuštění transakcí přicházejí různě v čase
- jelikož jedna transakce obsahuje řídící (programové) konstrukce, neví se dopředu ani sekvence skutečně vykonaných operací v rámci jediné transakce
- tj. nelze připravit rozvrh „dopředu“ a pak ho jen vykonat

Rozvrhovač tvoří rozvrh dynamicky, tj. může se řídit pouze lokálními podmínkami → PROTOKOL

DB klient(i)	T1	T2	T3
	R(A)	**R(C)**	**R(A)**
	W(B)	**W(C)**	**R(B)**
	W(A)		**R(C)**

Rozvrh

Proč uzamykat?

- zamykání entit může rozvrhovači sloužit jako nástroj pro zajištění konfliktové uspořádatelnosti tím, že potenciálně konfliktním dvojicí akcí je zámky nastaveno pevné pořadí vykonání
  - krátce, transakce nelze prokládat (vytvářet rozvrh) libovolně
  - uzamykací protokoly zjednodušují úlohu zajistit konfliktově uspořádatelný rozvrh

Uzamykání databázových entit

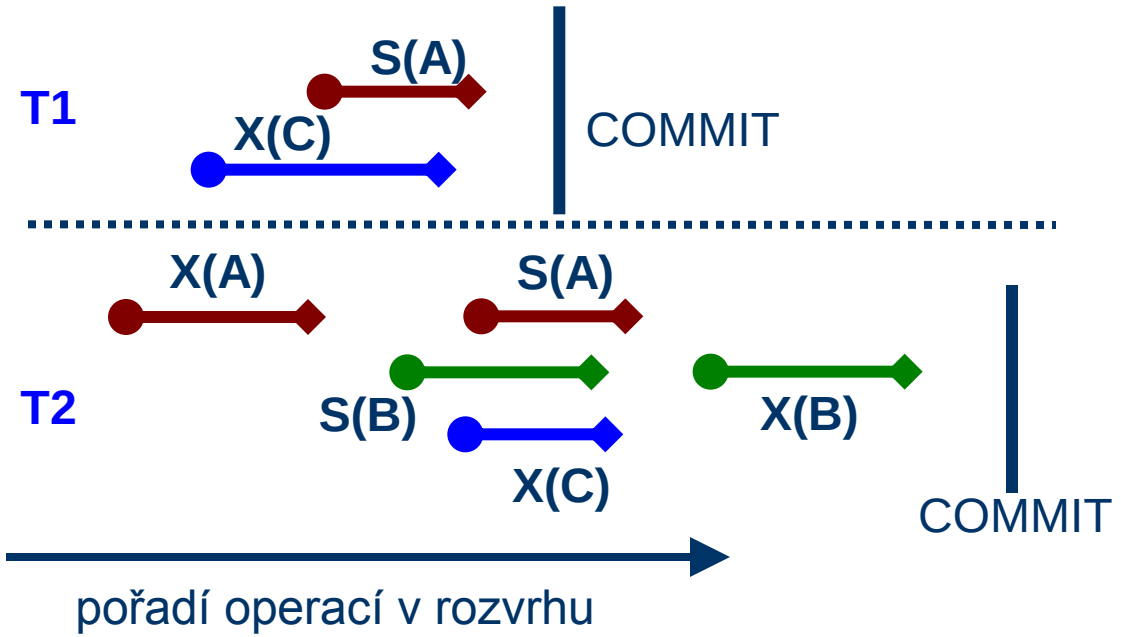
- **exkluzivní (exclusive) zámky**
  - **X(A)**, zcela uzamknou entitu A – číst i zapisovat do A může pouze vlastník zámku (ten, kdo uzamknul)
  - může být přidělen pouze jedné transakci
- **sdílené (shared) zámky**
  - **S(A)**, uzamknou entitu pro zápis – číst A může vlastník zámku – ten má jistotu, že do A nikdo nezapisuje
  - může být přidělen (sdílen) více transakcím (pokud již na A neexistuje exkluzivní zámek)
- odemyká se požadavkem **U(A)**
- pokud transakce požaduje zámek, který není k dispozici (je přidělen jiné transakci), je suspendována a čeká na přidělení
- uzamykání je uživateli (kódu transakce) skryto, používá ho rozvrhovač podle zvoleného uzamykacího protokolu
  - tj. použití zámků se objevuje pouze v rozvrhu (kam jsou přidány), ne v samotných (neproložených) transakcích

Implementace uzamykání

- akce X(A), S(A) lze chápat volněji – jako vznesení požadavku na zamknutí, tj. na těchto akcích „kurzor“ v rozvrhu „nečeká“, nicméně nelze pokračovat v rozvrhování následujících akcích dané transakce, dokud není zámek přidělen tzv. správcem zámků
- správce zámků (lock manager)
  - tabulka zámků
  - záznam v tabulce pro daný zámek: počet transakcí sdílejících zámek, typ zámku, ukazatel do fronty uzamykacích požadavků
- atomicita
  - je zcela nezbytné, aby zamykání a odemykání byly atomické operace (prostředky systému)

Příklad: rozvrh s uzamykáním

T1	T2
	X(A)
X(C)	
	W(A)
R(C)	
W(C)	
	U(A)
S(A)	
	S(B)
R(A)	
	R(B)
U(C)	
	X(C)
U(A)	
	S(A)
COMMIT	
	W(C)
	R(A)
	U(B)
	U(C)
	U(A)
	X(B)
	W(B)
	U(B)
	COMMIT



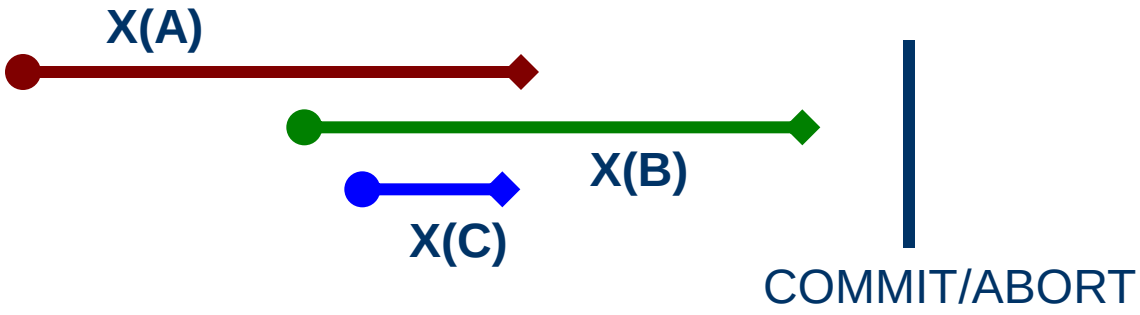
Dvoufázový uzamykací protokol (2PL)

2PL uplatňuje dvě pravidla pro sestavení rozvrhu:

1) pokud chce transakce číst (resp. modifikovat) entitu A, nejprve musí obdržet sdílený (resp. exkluzivní) zámek na A
2) transakce nemůže požadovat **žádný zámek**, pokud už nějaký zámek měla a uvolnila ho – na libovolné entitě

Jsou zřejmé dvě fáze – zamykání a odemykání

Příklad: upravení druhé transakce v předchozím rozvrhu na **2PL**

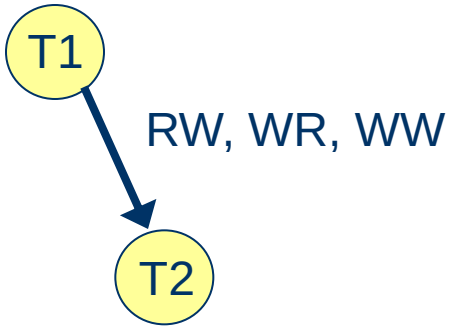


Vlastnosti 2PL

- omezení rozvrhu tak, aby splňoval 2PL zaručuje, že precedenční graf rozvrhu je **acyklický**, tj. rozvrh je **konfliktově uspořádatelný**
- 2PL **negarantuje zotavitelnost** rozvrhu

Příklad: rozvrh splňující 2PL, ale nezotavitelný, pokud T1 transakci zruší

T1	T2
X(A)	
R(A)	
W(A)	
U(A)	
	X(A)
	R(A)
	A := A *1.01
	W(A)
	S(B)
	U(A)
	R(B)
	B := B *1.01
	W(B)
	U(B)
	COMMIT
ABORT / COMMIT	

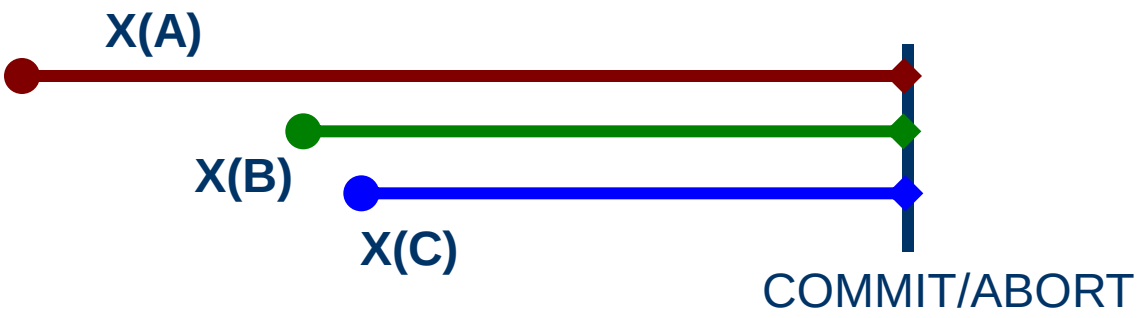


Striktní 2PL

Striktní 2PL „zostřuje“ druhé pravidlo 2PL, obě tedy zní:

1) pokud chce transakce číst (resp. modifikovat) entitu A, nejprve musí obdržet sdílený (resp. exkluzivní) zámek na A
2) **všechny zámky jsou uvolněny při ukončení transakce**

Příklad: upravení druhé transakce v předchozím rozvrhu na **striktní 2PL**



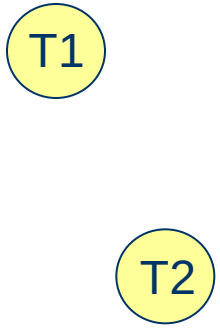
Vkládat požadavky U(A) do rozvrhu není potřeba, provedou se implicitně při COMMIT nebo ABORT.

Vlastnosti striktního 2PL

- omezení rozvrhu tak, aby splňoval 2PL zaručuje, že precedenční graf rozvrhu je **acyklický**, tj. rozvrh je **konfliktově uspořádatelný**
- striktní 2PL navíc **garantuje**
  - **zotavitelnost** rozvrhu (transakci lze stornovat a obnovit původní hodnoty)
  - zabezpečení proti **kaskádovému rušení** transakcí

Příklad: rozvrh splňující striktní 2PL

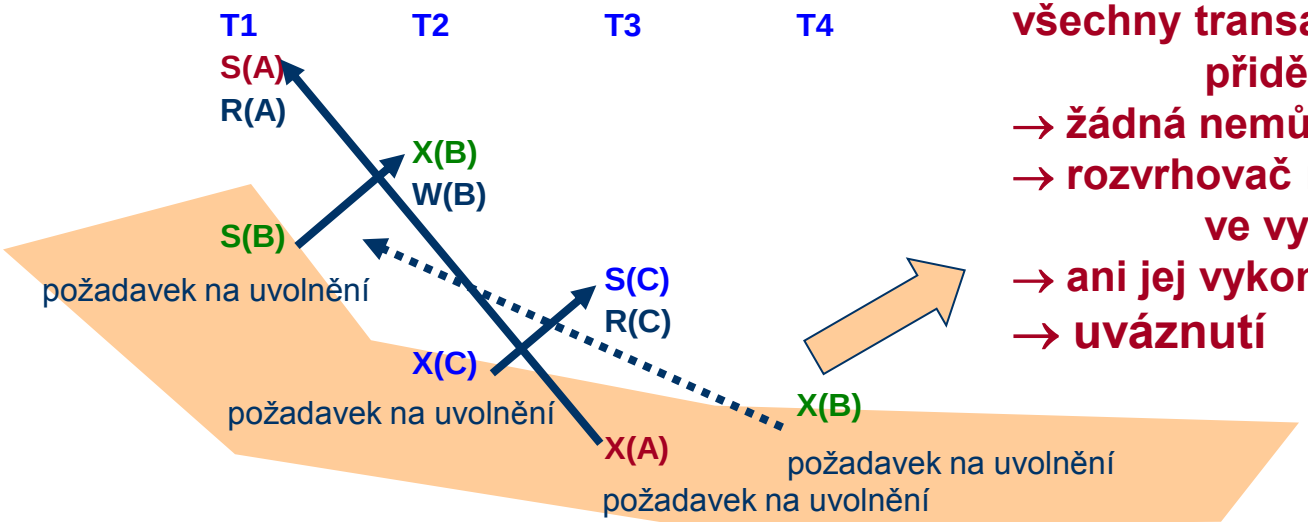
T1	T2
S(A)	
R(A)	
	S(A)
	R(A)
	X(B)
X(C)	
R(C)	
	R(B)
	W(B)
	COMMIT
W(C)	
ABORT / COMMIT	



Uváznutí (deadlock)

- při zpracovávání rozvrhu může dojít k situaci, kdy transakce T1 žádá zámek, který má přidělený T2, ale ta ho nemohla uvolnit, protože zase čeká na zámek neuvolněný T1
  - situaci lze zobecnit na více transakcí,
    T1 čeká na T2, T2 čeká na T3, ..., Tn čeká na T1
- může nastat i při použití striktního 2PL (nemluvě o slabších protokolech)

Příklad:



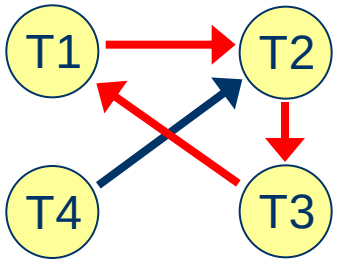
všechny transakce čekají na přidělení zámku
→ žádná nemůže zámek uvolnit
→ rozvrhovač nemůže pokračovat ve vytváření rozvrhu
→ ani jej vykonávat
→ uváznutí

Detekce uváznutí

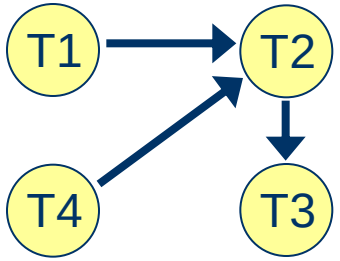
- uváznutí lze detekovat pravidelným zkoumáním tzv. waits-for grafu
- **waits-for** graf je dynamicky se měnící graf, podle toho jak jsou přidělovány/uvolňovány zámky
  - uzly tvoří aktivní transakce rozvrhu
  - orientovaná hrana znázorňuje čekání transakce T1 na (nějaký) zámek v současnosti přidělený transakci T2
  - pokud je v grafu cyklus, došlo k uváznutí (alespoň dvou transakcí)

Příklad: graf k předchozímu příkladu

(a) s požadavkem T3 na X(A)



(b) bez požadavku T3 na X(A)



Konverze zámků

- **lock upgrade** – změna již uděleného sdíleného zámku na exkluzivní
  - může pomoci zmenšit riziko uváznutí
  - Příklad: když transakce T1 již má S(A) a později požaduje i X(A), měla by dostat přednost před T2, která rovněž požaduje X(A), ale stejně ho nemůže dostat, protože S(A) vlastní T1

- **lock downgrade** – změna již uděleného exkluzivného zámku na sdílený
  - rovněž zmenšuje riziko uváznutí, i když snižuje paralelismus díky (potenciálně bezúčelně) přidělovaným exkluzivním zámkům
  - 2PL lze rozšířit tak, aby zahrnoval i lock downgrade
  - používá většina komerčních systémů

Řešení uváznutí

- k uváznutí zpravidla dochází zřídka, proto lze případné uváznutí řešit jednoduše
  - pokud transakce čeká na zámek moc dlouho, asi uvázla – transakci zrušíme
- pro lepší diagnostiku se použije waits-for graf, který je periodicky testován na cyklus
  - pokud nastane uváznutí, zrušíme jednu transakci v cyklu
  - zruší se ta transakce v cyklu, která (jedna z možností)
    - má nejmenší počtu zámků
    - zatím vykonala nejméně práce
    - má nejdále k dokončení
  - zrušená transakce může být znovu restartována a při dalším případném uváznutí upřednostněna (aby nebyla zrušena znovu)

Prevence uváznutí

- **prioritní upřednostňování**
  - každá transakce má prioritu (danou např. časovým razítkem, čím starší transakce, tím vyšší priorita)
  - pokud transakce T1 požaduje zámek a T2 již tento zámek drží, správce zámků volí mezi dvěma strategiemi
    - **wait-die** – pokud T1 má vyšší prioritu, může čekat, pokud ne, je zrušena
    - **wound-wait** – pokud T1 má vyšší prioritu, zruší se T2, jinak T1 čeká
  - restartované (zrušené) transakci je potřeba přiřadit původní prioritu, aby měla vyšší šanci na dokončení při případném dalším uváznutí
- **konzervativní 2PL protokol**
  - protokol, který požaduje, aby **všechny zámky**, které transakce bude v celém svém průběhu potřebovat, byly žádány na začátku transakce
  - pokud nelze přidělit všechny, je alespoň požádáno o jejich blokování (rezervaci) tak, že jakmile dojde k jejich uvolnění, dostane je transakce najednou a začne pracovat
  - protokol není v praxi používán, protože
    - se těžko předem odhaduje, které zámky budou potřeba v celém průběhu transakce (dynamicita a větvení)
    - vysoká režie uzamykání

Fantom

- nyní uvažujme dynamickou databázi, tj. do databáze lze vkládat a z databáze mazat (tj. ne pouze číst a aktualizovat stávající data)
- pokud jedna transakce pracuje s množinou entit a druhá tuto množinu LOGICKY mění (přidává nebo odebírá), může to mít za následek nekonzistenci databáze (neuspořádatelný rozvrh)
  - proč: T1 uzamkne všechny entity, které v dané chvíli odpovídají jisté vlastnosti (např. podmínce WHERE příkazu SELECT)
  - T2 v průběhu zpracování T1 může tuto množinu LOGICKY rozšířit (tj. v této chvíli by množina zámků definovaná podmínkou WHERE byla větší), což má za následek, že některé nově přidané entity budou uzamknuté (a tudíž i zpracované), kdežto jiné ne
- platí i pro striktní 2PL

Příklad – fantom

T1: najdi nejstaršího zaměstnance – muže a nejstarší zaměstnankyni – ženu
(**SELECT * FROM** Zaměstnanci ...) + **INSERT INTO** Statistika ...

T2: vlož zaměstnance Františka a smaž zaměstnankyni Evu (náhrada zaměstnance)
(**INSERT INTO** Zaměstnanci ..., **DELETE FROM** Zaměstnanci ...)

Výchozí stav databáze: **{[Pepa, 52, m], [Jaroslav, 46, m], [Eva, 55, ž], [Dáša, 30, ž]}**

T1	T2
uzamkni muže, tj.	
S(Pepa)	
S(Jaroslav)	
M = max{**R(Pepa)**, **R(Jaroslav)**}	
	Insert(František, 72, m)
	uzamkni ženy, tj.
	X(Eva)
	X(Dáša)
	Delete(Eva)
	COMMIT
uzamkni ženy, tj.	
S(Dáša)	
Ž = max{**R(Dáša)**}	
Insert(M, Ž) // výsledek se vloží do tabulky Statistika	
COMMIT	

fantom
lze vložit zaměstance muže, ačkoliv by (logicky) měli být uzamknuti **všichni** muži během celé transakce T1

Ačkoliv rozvrh splňuje **striktní 2PL**, výsledek **[Pepa, Dáša]** je nekorektní, protože neodpovídá ani sériovému rozvrhu T1, T2, který vrátí **[Pepa, Eva]**, ani T2, T1, který vrátí **[František, Dáša]**.

Fantom – prevence

- pokud neexistují indexy (např. B+-stromy) na entitách „uzamykací podmínky“, je potřeba zamknout vše, co by mohlo být fantomem negativně ovlivněno
  - např. celou tabulku, nebo i více tabulek
- pokud existují indexy na entitách „uzamykací podmíky“, je možné „hlídat fantoma“ na úrovni indexu (**index locking**) tak, že při vnějším pokusu o modifikaci množiny vyhovujících záznamů je „potenciálně fantomová transakce“ pozdržena stejně, jako by byla nová/modifikovaná/mazaná entita uzamčena
- zobecněním indexového uzamykání je predikátové uzamykání (predicate locking), kdy se požadují zámky přímo na úrovni logické podmínky a ne na úrovni fyzické
  - těžko se implementuje, proto se používá zřídka

Úrovně izolace

- čím striktnější uzamykací protokol, tím horší možnosti souběžného zpracování transakcí
  - pro různé účely jsou vhodné různé protokoly tak, aby se dosáhlo co nejvyššího výkonu a dostatečné „iluze“ izolace transakcí
- proto SQL-92 charakterizuje úrovně izolace pro různé účely

Úroveň	Protokol	WR	RW	Fantom
READ UNCOMMITTED (read only transakce)	žádný	možná	možná	možná
READ COMMITTED	S2PL na X() + 2PL na S()	Ne	možná	možná
REPEATABLE READ	S2PL	Ne	Ne	možná
SERIALIZABLE	S2PL + prevence fantoma	Ne	Ne	Ne

Alternativní („nezámkové“) protokoly

- mějme DB situaci, v níž transakce mohou přijít do konfliktu jen zřídka, tehdy
  - jsou uzamykací protokoly „kanón na vrabce“ – zbytečně pesimistické řízení – pořád se něco zamyká/odemyká
  - režie uzamykání neúměrně vysoká skutečnému využití zámků při prevenci konfliktů
- lepší použít
  - optimistické řešení
  - řízení pomocí časových razítek

Optimistické řízení

- základní předpoklad je, že ke souběžně vykonávané transakce mohou přijít do konfliktu jen zřídka (tj. průnik dat, na kterých každé dvě transakce pracují, je malý nebo nulový)
- třífázový optimistický protokol
  - **Read**: transakce reálně čte data z DB, nicméně zapisuje do svého lokálního datového prostoru
  - **Validation**: jakmile chce transakce potvrdit, předloží SŘBD svůj datový prostor (tj. požadavek na změnu databáze)
    - SŘBD rozhodne, zda takový požadavek není v konfliktu s jinou transakcí – pokud ano, transakce je zrušena a restartována
    - pokud je vše nekonfliktní, nastance poslední fáze:
  - **Write**: obsah lokálních dat se zkopíruje do databáze
- pokud naopak nastává mnoho konfliktů (transakce hodně „soutěží“ o společná data),
  - je optimistické řízení nevýhodné, často dochází k rušení/restartu transakcí
  - lépe je použít uzamykací protokol

Časová razítka

- uzamykací protokoly tím, že vynucují čekání transakce na zámek, konfliktově uspořádávají akce (a tím vlastně i celé transakce), aby odpovídaly nějakému sériovému rozvrhu
- využití uspořádání konfliktů lze zařídit i bez uzamykání, pomocí **časových razítek**
  - každá transakce Ti obdrží na začátku vykonávání časové razítko TS(Ti) (logický nebo fyzický čas v okamžiku startu transakce)
  - během zpracování/rozvrhování je kontrolováno pořadí konfliktních operací, tj. pokud akce A1 transakce T1 je v konfliktu s akcí A2 transakce T2, musí A1 nastat před A2 pokud TS(T1) < TS(T2)
    - pokud tato podmínka neplatí, je transakce T1 zrušena a restartována
  - implementuje se pomocí časových razítek pro čtení i zápis každé entity v databázi
  - pro dosažení zotavitelnosti rozvrhu se musí navíc implementovat „bufferování“ všech zápisů dokud transakce nepotvrdí

Zotavení po havárii systému

- správce zotavení (recovery manager) zajišťuje
  - **atomicitu** – odvolání (undoing) všech akcí, pokud je transakce zrušena
  - **trvanlivost** – promítnutí všech potvrzený akcí do databáze, i když systém zhavaruje
- jedna z nesložitějších a nejhůře implementovatelných součástí SŘBD
- ARIES – algoritmus pro zotavení
  - používaný mnoha systémy (např. IBM DB2, MS SQL, Informix, Sybase, Oracle)
  - algoritmus se spustí po restartu systému, který předtím zhavaroval)
- tři fáze ARIES:
  - **Analysis**: identifikují se „dirty pages“ (modifikované, ale nezapsané stránky) v bufferu a transakce, které byly aktivní v okamžiku havárie
  - **Redo**: zopakují se všechny akce od příslušného místa zapsané v tzv. **logu** a databáze se obnoví do stavu před havárií
  - **Undo**: odvolají se všechny akce těch transakcí, které nestačily potvrdit, tj. databáze je nakonec ve stavu, který odráží pouze potvrzené transakce

Principy algoritmu ARIES

- Write-Ahead Logging (WAL)
  - každá změna databáze je nejdříve zaznamenána do logu (který musí být na stabilním médiu)
- Repeating History During Redo
  - při restartu systému ARIES pomocí záznamů v logu vystopuje všechny akce před havárií a znovu je promítne (redo) tak, aby databáze byla ve stavu před havárií
  - potom jsou všechny nepotvrzené transakce zrušeny
- Logging Changes During Undo
  - při procesu rušení transakce (undo) jsou změny rovněž logovány, pro případ, že by systém (opět) zhavaroval ve fázi ABORT

Log

- někdy také nazývaný, **trail** nebo **journal**
- uchovává **historii akcí** provedených SŘBD
  - např. na úrovni stránek
- záznam v logu je identifikován (log sequence number - LSN)

Příklad – zotavení pomocí ARIES