Transakční zpracování dat

přednáška č. 4

EVROPSKÁ UNIE esf MŠMT

Transakce

- ***Def.:***
  - Transakce je série příkazů čtení či zápisu na databázových objektech
- Čtení: pro čtení se musí db objekt přenést do paměti z HDD a následně je přenesena do proměnné programu
- Zápis: db objekt je modifikován v paměti a následně zapsán na disk
- DB objekty: Jednotky se kterými pracují programy – např. stránky, záznamy, …

Příklad

```
BEGIN TRANSACTION;
UPDATE Human.JobCandidate SET jobs =
 11 where jobs IS NULL;

DELETE FROM Human.JobCandidate WHERE
 JobCandidateID = 13;

COMMIT TRANSACTION;
```

Základní vlastnosti transakcí - ACID

- Atomicity
- Consistency
- Isolation
- Durability
- ACID

Atomicity

- Buď proběhnout všechny operace transakce nebo neproběhnou žádné
- Uživatel se nemusí starat o výsledek (co ovlivnila) nekompletní transakce
- Například v případě selhání systému

Consistency

- Transakce musí zachovávat konzistentní stav databáze
- Tedy při spuštění transakce v db, která je v konzistentním stavu, ji tato transakce musí po svém ukončení opustit v konzistentním stavu
- Tuto vlastnost musí zaručit uživatel (tedy autor transakce)
- DBMS se pak stará o to, aby to platilo i u paralelně pobíhajících transakcí

Isolation

- V případě že v systému probíhá více transakcí paralelně, DBMS se musí zaručit, že transakce jsou izolované jedna od druhé, tedy že jedna nebude ovlivňovat druhou
- Pro uživatele to znamená, že pro něj DB vypadá jako, že aktuálně probíhá pouze jedna (jeho) transakce

Durability

- Pokud byla transakce jednou dokončena a potvrzena, pak DBMS musí zaručit, že změny, která tato transakce provedla zůstanou zachovány v DB i při případném selhání systému

Současné provádění transakcí

- Motivace:
  - Jak bylo řečeno pro zpracování db objektu, je nejprve nutné jej nahrát do paměti
  - IO operace jsou však velmi „drahé“
  - Proto by bylo nejlépe v době, kdy dochází k nahrání objektu do paměti, umožnit CPU zpracovávat jiné požadavky transakcí.

Anomálie při provádění transakcí

- Tedy případ, kdy by došlo k tomu, že transakce spuštěná nad konzistentní DB, by ji opustila v nekonzistentním stavu
- Co je to za vlastnost z ACID?

Typy konfliktů

- Reading Uncommitted data – WR conflicts
- Unrepeatable reads – RW conflicts
- Overwriting Uncommitted Data – WW conflicts

Reading Uncommitted data – WR conflicts

- T2 může číst data, která byla změněna nepotvrzenou (commit) T1.

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

Reading Uncommitted data – WR conflicts - příklad

- T1 převádí 1000 Kč z účtu A na účet B, T2 zvyšuje zůstatky na obou účtech o 6%.
- Máme následující plán:
- T1 1000 odečte z A, T2 čte hodnoty z A i B a zvyšuje o 6%, T1 přenese 1000 na B (viz minulý slide).
- Je to v pořádku?

Unrepeatable reads – RW conflicts

- Problémy:
  - T1 bude číst objekt A dvakrát a pokaždé bude mít jinou hodnotu, přesto že ho T1 neměnila -> unrepeatable read
  - Mějme T1 co inc(A) a T2 dec(A).
  - Pokud A=5 pak jakýkoli seriový plán bude končit A=5.
  - Náš plán však skončí jinak

T1	T2
R(A)	
	W(A)
R(B)	
R(A)	
	Commit
Commit	

Overwriting Uncommitted Data – WW conflicts

- T1 R(A), T1 W(A), T2 W(A), T1 commit, T2 commit

T1	T2
R(A)	
W(A)	
	W(A)
	Commit
Commit	

Overwriting Uncommitted Data – WW conflicts - příklad

- Podmínka konzistence: platy lidí A a B musí být stejné.
- T1 nastaví platy na 1000
- T2 nastaví platy na 2000
- T1 T2 -> 2000 ; T2 T1 -> 1000
- T1 A=1000, T2 B=2000, T1 B=1000, T2 A=2000 -> A !=B

Concurrency Control

- 2PL (Two-Phase Locking) a Strict 2PL
- Přidělování sdílených zámků
- Atomicita zamykání
  - Musíme zaručit, že příkazy lock a unlock budou atomické operace - semafory
- Lock Upgrade
  - Pokud T má zámek typu S pak může požádat o jeho update na X

Strict 2PL

- Definuje pravidla, kdy lze přidělovat zámky transakcím:
  1. Pokud transakce T chce číst (modifikovat) objekt DB, musí nejprve požádat o sdílený (exklusivní) zámek
  2. Všechny zámky držené transakcí T jsou uvolněny po skončení T

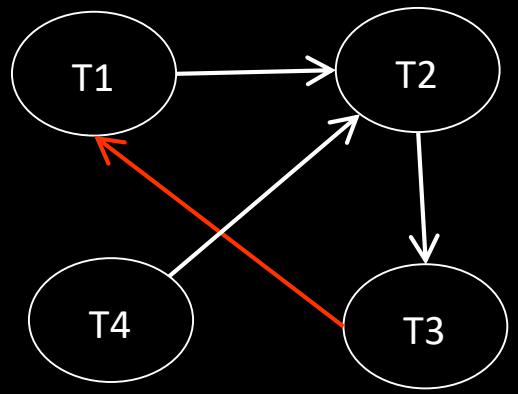
2PL

1. Pokud transakce T chce číst (modifikovat) objekt db, nejprve musí požádat o sdílený (exklusivní) zámek – zůstává jako u Strict 2PL
2. Transakce nemůže požádat o nový zámek, pokud již nějaký uvolnila

DeadLock

- Prevence (pomocí priorit T)
  - Wait-Die
  - Wound-Die
- Detekce
  - Wait-For Graph
  - Time-out
- Prevence x Detekce
- Kterou T abortovat?

T1	T2	T3	T4
S(A)			
R(A)			
	X(B)		
	W(B)		
S(B)			
		S(C)	
		R(C)	
	X(C)		
			X(B)
		X(A)	



Dynamické databáze

- Phantom problém:
  - Máme tabulku obsahující platy zaměstnanců.
  - Dále uvažujme dotaz na všechny zam., kteří mají plat mezi 10 tis. a 30 tis. korun
  - Pro tento dotaz dojde k zamčení všech záznamů obsahují informace o platech
  - Problém je, že tento zámek nemůže zabránit jiné transakci vložit do DB nový záznam, který bude odpovídat dotazu (nový pracovník s platem např. 15 tis. korun)
  - Může to nastat např. při vkládání nebo také pro Roll-back jiné transakce
  - Pokud tedy budeme počítat např. průměrný plat dvakrát za sebou, pokaždé může obdržet jiný výsledek .
- Index Locking
- T musí projít všechny stránky a zamknout
- B+ stromy
- Multiple-Granularity Locking
  - Soubor->Stránka->Záznam

Concurrency without Locking

- Optimistic Concurrency Control
  - READ: T načte požadované objekty do svého pracovní prostoru v paměti
  - VALIDATION: pokud T je připravena končit DBMS zkontroluje, zda-li T je v konfliktu s jinou Tx (pomocí časových značek); pokud ano pak je T abortována a restartována
  - WRITE: pokud nejsou konflikty pak změny v pracovním prostoru T jsou zapsány na disk
- Funguje dobře pokud je pouze několik konfliktů v DB

Concurrency without Locking

- Timestamp-Based C.C.
  - RTS(O) read time stamp
  - WTS(O) write time stamp
  - TS(O) time stamp transakce

Stupně izolace transakcí

```
SET TRANSACTION ISOLATION LEVEL
{
READ UNCOMMITTED      |
READ COMMITTED        |
REPEATABLE READ       |
SNAPSHOT              |
SERIALIZABLE          }
[ ; ]
```

READ UNCOMMITTED

- Specifies that statements can read rows that have been modified by other transactions but not yet committed.
- Transactions running at the READ UNCOMMITTED level do not issue shared locks to prevent other transactions from modifying data read by the current transaction.
- READ UNCOMMITTED transactions are also not blocked by exclusive locks that would prevent the current transaction from reading rows that have been modified but not committed by other transactions.
- When this option is set, it is possible to read uncommitted modifications, which are called dirty reads.
- Values in the data can be changed and rows can appear or disappear in the data set before the end of the transaction.

READ COMMITTED

- Specifies that statements cannot read data that has been modified but not committed by other transactions.
- This prevents dirty reads. Data can be changed by other transactions between individual statements within the current transaction, resulting in nonrepeatable reads or phantom data.

REPEATABLE READ

- Specifies that statements cannot read data that has been modified but not yet committed by other transactions and that no other transactions can modify data that has been read by the current transaction until the current transaction completes.
- Shared locks are placed on all data read by each statement in the transaction and are held until the transaction completes.
- This prevents other transactions from modifying any rows that have been read by the current transaction.
- Other transactions can insert new rows that match the search conditions of statements issued by the current transaction. If the current transaction then retries the statement it will retrieve the new rows, which results in phantom reads.
- Because shared locks are held to the end of a transaction instead of being released at the end of each statement, concurrency is lower.

SERIALIZABLE

- Statements cannot read data that has been modified but not yet committed by other transactions.
- No other transactions can modify data that has been read by the current transaction until the current transaction completes.
- Other transactions cannot insert new rows with key values that would fall in the range of keys read by any statements in the current transaction until the current transaction completes.
- Range locks are placed in the range of key values that match the search conditions of each statement executed in a transaction. This blocks other transactions from updating or inserting any rows that would qualify for any of the statements executed by the current transaction.
- This means that if any of the statements in a transaction are executed a second time, they will read the same set of rows.
- The range locks are held until the transaction completes.
- This is the most restrictive of the isolation levels because it locks entire ranges of keys and holds the locks until the transaction completes.

Příklad Dirty Reads

- Dirty read nastává, pokud transakce čte data, která byla modifikována jinou transakcí, která není potvrzena.

```
Transakce A začátek.
UPDATE employee SET salary = 31650
WHERE empno = '000090'
Transakce B začátek.
SELECT * FROM employee
(Transakce B vidí data, která jsou
 upravena transakcí A. Tyto změny nejsou
 potvrzeny.)
```

Příklad Non-Repeatable Reads

- Non-repeatable reads nastává pokud dva dotazy v rámci jedné transakce vrací jiná data. Nastává tehdy pokud jiná transakce modifikuje data, která jsou čtena.

```
Transakce A začátek.
SELECT * FROM employee WHERE empno = '000090'
Transakce B začátek.
UPDATE employee SET salary = 30100 WHERE
  empno='000090'
(Transakce B upravuje data, která jsou čtena
  transakcí A, předtím než je transakce A
  potvrzena.
Transakce A pokračuje.
SELECT * FROM employee WHERE empno = '000090',
(v tuto chvíli dostává jiný výsledek)
```

Příklad Phantom Reads

- Nové záznamy odpovídající podmínkám dotazu se objevují během čtení transakcí. Records that appear in a set being read by another transaction.

```
Transakce A začátek.
SELECT * FROM employee  WHERE salary > 30000
Transakce B začátek.
INSERT INTO employee (empno, firstnme, midinit,lastname, job,salary) VALUES ('000350', 'NICK','A','GREEN','LEGAL' ,COUNSEL',35000)
(transacke B vkládá nový záznam, který odpovídá podmínce v dotazu transakce A.)
Transakce A pokračuje.
SELECT * FROM employee  WHERE salary > 30000
```

Transakce v SQL

TRANSACTION jméno_transakce

WHENEVER {ERROR|podmínka} ROLLBACK

příkazy

COMMIT END

- Příkazy z množiny manipulačních

```
TRANSACTION dalsi_rok
                    WHENEVER ERROR ROLLBACK
                                SELECT plat,vek FROM osoba  WHERE vek>18
                                UPDATE osoba  SET vek=vek+1
                                DELETE * FROM osoba WHERE plat = 0
COMMIT END
```

Příklad MS SQL

```
USE AdventureWorks;
GO
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
GO
BEGIN TRANSACTION;
GO
SELECT * FROM HumanResources.EmployeePayHistory;
GO
SELECT * FROM HumanResources.Department;
GO
COMMIT TRANSACTION;
GO
```

Postgres

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN;

UPDATE accounts SET balance = balance + 100.00
  WHERE acctnum = 12345;

UPDATE accounts SET balance = balance - 100.00
  WHERE acctnum = 7534;

COMMIT;
```

Transakce v aplikaci

- Hibernate:

```
// Non-managed environment idiom
Session sess = factory.openSession();
Transaction tx = null;
try {
    tx = sess.beginTransaction();

    // do some work
    ...

    tx.commit();
}
catch (RuntimeException e) {
    if (tx != null) tx.rollback();
    throw e; // or display error message
}
finally {
    sess.close();
}
```

Spring framework

- @Annotation driven

```
@Transactional(params)
public class DefaultFooService implements FooService {

    Foo getFoo(String fooName);

    Foo getFoo(String fooName, String barName);

    void insertFoo(Foo foo);

    void updateFoo(Foo foo);
}
```

Parameters

Property	Type	Description
value	String	Optional qualifier specifying the transaction manager to be used.
propagation	enum: Propagation	Optional propagation setting.
isolation	**enum: Isolation**	**Optional isolation level.**
readOnly	boolean	Read/write vs. read-only transaction
timeout	int (in seconds granularity)	Transaction timeout.
rollbackFor	Array of Class objects, which must be derived fromThrowable.	Optional array of exception classes that *must* cause rollback.
rollbackForClassName	Array of class names. Classes must be derived fromThrowable.	Optional array of names of exception classes that *must* cause rollback.
noRollbackFor	Array of Class objects, which must be derived fromThrowable.	Optional array of exception classes that *must not* cause rollback.
noRollbackForClassName	Array of String class names, which must be derived fromThrowable.	Optional array of names of exception classes that *must not*cause rollback.

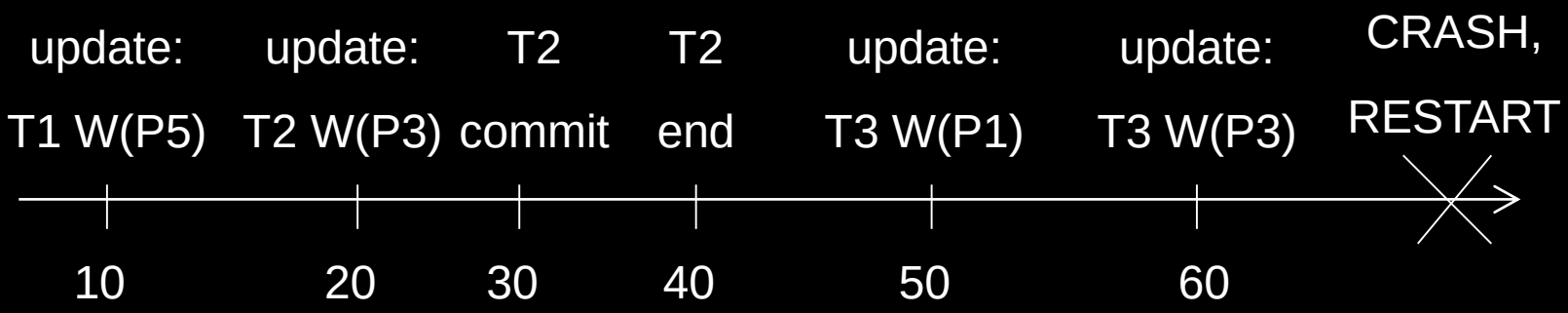
enum: Isolation

- READ_UNCOMMITTED
- READ_COMMITTED
- REPEATABLE_READ
- SERIALIZABLE

	dirty reads	non-repeatable reads	phantom reads
READ_UNCOMMITTED	yes	yes	yes
READ_COMMITTED	no	yes	yes
REPEATABLE_READ	no	no	yes
SERIALIZABLE	no	no	no

Recovery

- ***Atomicity*** and ***durability*** properties
- ARIES recovery algorithm
  - Analýza
  - REDO
  - UNDO

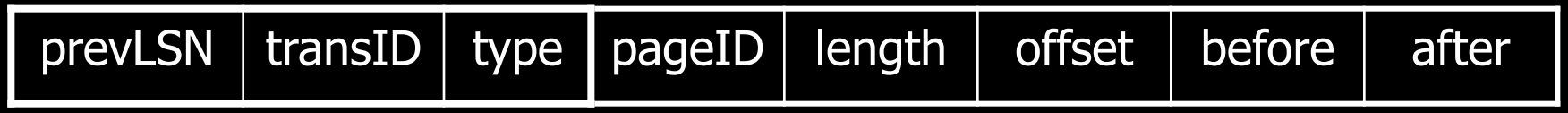


ARIES al. - principy

- Write-ahead logging
  - Nejdříve do logu (uložen na stabilním místě) a pak změny zapsány do databáze
- Repeating history during redo
  - Návrat do stavu v jakém byla DB v době pádu
- Logging changes during undo
  - Pro případ opakovaného pádu

Log (žurnál)

- Musí být zachován i při pádu (více kopií)
- Každý záznam má unikátní ID (LSN)
- Záznamy pro:
  - Update stránky, Commit, Abort, End, UNDO
- Pomocí prevLSN, transID, type (typ záznamu) je „nalinkována“ minulost



Děkuji za pozornost