

## **Zkratky**

DBS - Databázový systém

JMD - Jazyk manipulačních dat (DML)

DML - Data Manipulation Language (Jazyk pro manipulaci s daty, JMD)

DDL - Data Definition Language (Jazyk pro definici dat)

ORŠRBD - Objektově-relační Systém Řízení Báze Dat

SŘBD - Systém Řízení Báze Dat

ACID - Atomicity, Consistency, Isolation, Durability (Atributy transakcí)

JDD – Jazyk pro Definici Dat (just doing drugs)

RW - Read-Write (čtení-zápis)

WR - Write-Read (zápis-čtení)

WW - Write-Write (zápis-zápis)

SQL - Structured Query Language (Strukturovaný dotazovací jazyk)

PL/SQL - Procedural Language/Structured Query Language

T-SQL - Transact-Structured Query Language, je to PL/SQL pro Microsoft database

TCL - Transaction Control Language (Jazyk pro řízení transakcí)

RDBMS - Relational Database Management System (Relační databázový systém)

MVCC - Multi-Version Concurrency Control (Víceverzí řízení souběžnosti)

WAL - Write-Ahead Logging (Zápis dopředu logování)

S-lock - Shared Lock (Sdílený zámek)

X-lock - Exclusive Lock (Výlučný zámek)

### Příklad [1] (9 bodů)

Na konkrétním paralelním plánu dvou transakcí ukažte *výjimku* uzamykacího protokolu, která vede k problému souběhu *neopakovatelné čtení*. V transakcích použijte jen operace READ a WRITE pro jeden záznam.

#### 4. Neopakovatelné čtení



Transakce A	Čas	Transakce B
READ $t$	$t_1$	
	$t_2$	WRITE $t$
READ $t$	$t_3$	

V čase  $t_1$  a  $t_3$  získá transakce A odlišnou hodnotu záznamu  $t$ , mluvíme o **neopakovatelné čtení**.

#### Neopakovatelné čtení 1/2



- V případě úrovně **RC** (a nižší) je umožněno tzv. **neopakovatelné čtení**.
- V tomto případě příkaz SELECT požaduje **sdílený zámek** na záznam, SRBD ale nedodrží dvoufázový zamykací protokol a zámky mohou být **uvolněny před** ukončením transakce.
- **Výlučné zámky** jsou ovšem **uvolněny až na konci** transakce.
- V případě úrovně SR a RR k této výjimce nedochází, v případě úrovně RC a RU se může tento problém objevit.

#### Neopakovatelné čtení 2/2



Transakce A	Čas	Transakce B
SELECT * from student WHERE id='jan001'	$t_1$	-
-	$t_2$	UPDATE student SET rocnik=1 WHERE id='jan001' COMMIT
SELECT * from student WHERE id='jan001'	$t_3$	-
COMMIT	$t_4$	

### Příklad [2] (9 bodů)

Na příkladu konkrétní posloupnosti dvou operací JMD a jednoho dotazu, ukažte rozdíl mezi provedením operací v transakci a provedením operací bez použití transakce. Jak oba způsoby ovlivní výsledek posloupnosti operací?

```
Transakce, zotavení / Transakce
Transakce v PL/SQL, příklad, 1/2

Uvažujme tabulku Student:

CREATE TABLE Student (
  login CHAR(5) PRIMARY KEY,
  fname VARCHAR(30) NOT NULL,
  lname VARCHAR(30) NOT NULL,
  email VARCHAR(40) NOT NULL
);

© 2009-2024 db.cs@vsb.cz DS2 - 7. Transakce, zotavení 3 / 43
```

- dvě operace JMD = INSERT, jeden dotaz = SELECT
- bez transakce:

```
sql Zkopírovat kód

INSERT INTO Student VALUES ('sob28', 'Jan', 'Sobota', 'jan.sobota@vsb.cz');
-- Předpokládáme, že tento příkaz je úspěšný

INSERT INTO Student VALUES ('sob28', 'Jan', 'Neděle', 'jan.nedele@vsb.cz');
-- Tento příkaz selže kvůli duplicitnímu primárnímu klíči

SELECT * FROM Student;
-- Dotaz vrátí všechny záznamy, které byly úspěšně vloženy před selháním
```

- SELECT vrátí první záznam

- s transakcí:

```
Transakce, zotavení / Transakce
Transakce v PL/SQL, příklad, 2/2

Chceme do databáze vložit záznamy dvou studentů, v případě chyby
nebude vložen ani jeden záznam.

BEGIN
  INSERT INTO Student VALUES ('sob28', 'Jan', 'Sobota',
    'jan.sobota@vsb.cz');
  INSERT INTO Student VALUES ('sob28', 'Jan', 'Neděle',
    'jan.nedele@vsb.cz');
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
END;

V tomto případě se nepodaří vložit druhý záznam, celá transakce bude
tedy zrušena.

© 2009-2024 db.cs@vsb.cz DS2 - 7. Transakce, zotavení 4 / 43
```

- případný SELECT by vrátil prázdnou množinu

- bez transakce: Všechny úspěšné operace jsou trvale uloženy, i když některé následující operace selžou. Částečně provedené operace mohou vést k nekonzistentnímu stavu databáze.
- s transakcí: Buď jsou všechny operace úspěšně provedeny a potvrzeny (COMMIT), nebo žádná z operací není uložena (ROLLBACK). Zajišťuje atomičnost operací a udržuje konzistentní stav databáze.

### Příklad [3] (10 bodů)

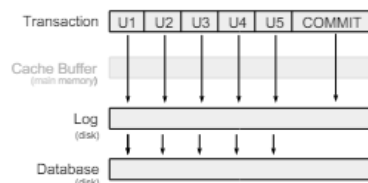
Popište co a kam se ukládá při vykonávání transakce v databázových systémech používajících techniku zotavení okamžitou aktualizací. Která z operací UNDO a REDO se provádí během zotavení databáze?

## 2. Zotavení okamžitou aktualizací 1/2



### ■ Zotavení okamžitou aktualizací (immediate update):

- Po každé aktualizaci, jsou do logu zapsány původní hodnoty záznamů, poté jsou do databáze zapsány nové hodnoty<sup>4</sup>.
- Po úspěšném ukončení transakce, je do logu zapsán **COMMIT** záznam.



<sup>4</sup>Pravidlo dopředného zápisu do logu.

## 2. Zotavení okamžitou aktualizací 2/2



### ■ Zotavení okamžitou aktualizací, vlastnosti:

- **UNDO/NO-REDO:**
  - Pokud transakce nebyla potvrzena, bude při zotavení zrušena operací **UNDO** (v logu jsou původní hodnoty, ale v databázi nové).
  - **NO-REDO:** nové hodnoty jsou v databázi po provedení aktualizace, REDO není nutné.
- **Výkon techniky je spíše nižší:** maximální počet diskových operací.

#### Příklad [4] (9 bodů)

Pro jaké příkazy JMD je nutné použít dynamické SQL namísto statického? Uveďte také dva konkrétní příklady takových operací.

### Statické PL/SQL



- V PL/SQL bloku nemůžeme přímo volat všechny dostupné SQL příkazy. Příkazy, které lze volat v PL/SQL přímo, nazýváme **statické příkazy PL/SQL**. Mezi statické příkazy patří:
  - SELECT, INSERT, UPDATE, DELETE, MERGE
  - LOCK TABLE, COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION
- Je zřejmé, že mezi příkazy, které nemůžeme volat přímo, jsou všechny příkazy **JDD (DDL)** a příkazy, které **v době překladu nejsou známy** (tedy např. i select pro tabulku, jejíž jméno předáváme jako parametr procedury).

### Dynamické PL/SQL



- **Dynamické PL/SQL** umožňuje sestavit a volat jakýkoli SQL příkaz (na který má uživatel právo) za běhu aplikace.
- Nevýhodou je, že nelze jednoduše ověřit syntaktickou správnost (správné datové typy, počet parametrů atd.). **Navíc se vystavujeme nebezpečí SQL injection!**
- Dynamické PL/SQL spouštíme příkazem `EXECUTE IMMEDIATE`.
- **Upozornění:** Dynamické PL/SQL používáme jen v případě, kdy není možné použít statické PL/SQL!

### Dynamické PL/SQL, Příklad



V tomto příkladě vytvoříme a odstraníme tabulku `Book` s použitím příkazu `EXECUTE IMMEDIATE`.

```
DECLARE
  v_command VARCHAR2(50);
BEGIN
  EXECUTE IMMEDIATE 'CREATE TABLE Book ' ||
    '(id INT UNIQUE, name VARCHAR2(50), ' ||
    'author INT REFERENCES author(author_id))';

  v_command := 'DROP TABLE Book';
  EXECUTE IMMEDIATE v_command;
END;
```

- dva příklady operací:

#### Vytvoření tabulky s dynamickým názvem:

```
sql Zkopírovat kód  
  
DECLARE  
    table_name VARCHAR2(50) := 'DYNAMIC_TABLE';  
BEGIN  
    EXECUTE IMMEDIATE 'CREATE TABLE ' || table_name || ' (id INT, name VARCHAR2(50))';  
END;
```

Tento příkaz nelze provést staticky, protože jméno tabulky je známo až v době běhu programu.

#### Výběr z tabulky s dynamickým názvem:

```
sql Zkopírovat kód  
  
DECLARE  
    table_name VARCHAR2(50) := 'DYNAMIC_TABLE';  
    result VARCHAR2(50);  
BEGIN  
    EXECUTE IMMEDIATE 'SELECT name FROM ' || table_name || ' WHERE id = 1' INTO result;  
    DBMS_OUTPUT.PUT_LINE('Name: ' || result);  
END;
```

Tento příkaz nelze provést staticky, protože jméno tabulky je známo až v době běhu programu.

#### Statické vytvoření tabulky:

```
sql Zkopírovat kód  
  
CREATE TABLE STATIC_TABLE (  
    id INT PRIMARY KEY,  
    name VARCHAR2(50)  
);
```

#### Statický výběr z tabulky:

```
sql Zkopírovat kód  
  
SELECT name  
FROM STATIC_TABLE  
WHERE id = 1;
```

**Příklad [5] (0 bodů)**  
Popište jednotlivé úrovně izolace transakce dle specifikace SQL.

Úroveň izolace transakce v SQL

Úroveň izolace v SQL

V SQL jsou definovány tyto 4 úrovně izolace (seřazeno od nejnižší úrovně po nejvyšší úroveň):

- READ UNCOMMITTED (RU)
- READ COMMITTED (RC)
- REPEATABLE READ (RR)
- SERIALIZABLE (SR)

©2009-2024 db.cs@vsb.cz DS2 – 11. Řízení souběhu II 14 / 53

- **propustnost** = přenosová rychlost, rychlost přenosu dat

Úroveň izolace transakce v SQL

Úroveň izolace v SQL

- Vyšší úroveň značí vyšší míru izolace, ale **nižší propustnost**.
- Naopak nižší úroveň značí nižší míru izolace, ale **vyšší propustnost**.
- Pokud zvolíme maximální úroveň, **SERIALIZABLE**, pak jsou transakce maximálně izolovány od vlivů ostatních souběžných transakcí.
- Naopak při nižší úrovni izolace, mohou nastat různé problémy souběhu.

©2009-2024 db.cs@vsb.cz DS2 – 11. Řízení souběhu II 15 / 53

Úroveň izolace transakce v SQL

Výjimky souběhu pro různé úrovně izolace

V následující tabulce vidíme různé výjimky souběhu, které mohou či nemohou nastat pro různé úrovně izolace transakcí:

Úroveň izolace	Špinavé čtení	Neopakovatelné čtení	Výskyt fantomů
READ UNCOMMITTED	Ano	Ano	Ano
READ COMMITTED	Ne	Ano	Ano
REPEATABLE READ	Ne	Ne	Ano
SERIALIZABLE	Ne	Ne	Ne

©2009-2024 db.cs@vsb.cz DS2 – 11. Řízení souběhu II 16 / 53

**Příklad [6] (9 bodů)**

Uvažujte složený index `indx_student(prijmeni, mesto, rocnik)` vytvořený pro tabulku `Student`. Napište příklad posloupnosti 5 klíčů indexu, která je uspořádána tak, jak jsou uspořádány klíče v tomto indexu. Napište jeden SQL dotaz se selekcí pro minimálně dva ze tří atributů indexu, které budou pravděpodobně provedeny bodovým nebo rozsahovým dotazem v tomto indexu, a jeden SQL dotaz se selekcí na dva atributy tohoto indexu, kdy DBS tento index nevyužije a provede sekvenční průchod haldou.

**Příklad posloupnosti 5 klíčů v indexu `indx_student(prijmeni, mesto, rocnik)`**

Předpokládejme, že máme následující záznamy v tabulce `Student`:

1. (``Novák``, ``Praha``, 3)
2. (``Svoboda``, ``Brno``, 2)
3. (``Dvořák``, ``Ostrava``, 1)
4. (``Černý``, ``Brno``, 3)
5. (``Procházka``, ``Praha``, 2)

**Klíče v indexu budou uspořádány lexikograficky podle ``prijmeni``, ``mesto``, a ``rocnik`` následovně:**

1. ``Černý, Brno, 3``
2. ``Dvořák, Ostrava, 1``
3. ``Novák, Praha, 3``
4. ``Procházka, Praha, 2``
5. ``Svoboda, Brno, 2``

**SQL dotaz využívající index**

Dotaz na dva ze tří atributů, který pravděpodobně využije index:

```
sql                                                                    Zkopírovat kód
SELECT * FROM Student
WHERE prijmeni = 'Novák' AND mesto = 'Praha';
```

Vysvětlení: Tento dotaz specifikuje dva atributy indexu v lexikografickém pořadí (`prijmeni`, `mesto`). Proto může být využit složený index `indx_student`.

**SQL dotaz, který nevyužije index**

Dotaz na dva atributy, který nevyužije složený index a pravděpodobně provede sekvenční průchod haldou:

```
sql                                                                    Zkopírovat kód
SELECT * FROM Student
WHERE mesto = 'Brno' AND rocnik = 3;
```

Vysvětlení: Tento dotaz nezahrnuje první atribut indexu (`prijmeni`). Vzhledem k tomu, že dotaz neodpovídá lexikografickému pořadí složeného klíče, index nebude využit a databázový systém pravděpodobně provede sekvenční průchod haldou.



1. co je checkpoint a jak probíhá zotavení při undo/redo technice + popsat, jak checkpoint ovlivňuje transakce

Transakce, zotavení / Základní techniky zotavení

### 3. Kombinovaná technika – UNDO/REDO

- V praxi se používá kombinovaný UNDO/REDO algoritmus:
  - Odložená aktualizace – hrozí přetečení paměti.
  - Okamžitá aktualizace – nízký výkon (vysoký počet diskových operací).
- **Zápis aktualizací:**
  - Nové hodnoty záznamů aktualizací jsou do logu zapsány po COMMIT (pro REDO).
  - Po určitém časové intervalu dochází k tzv. **kontrolním bodům (check points)**. V tomto bodu se provede zápis aktualizací transakcí do databáze:
    - **Aktuálně prováděných transakcí:** před tím jsou do logu uloženy původní hodnoty pro UNDO.
    - **Potvrzených transakcí:** jejichž aktualizace nebyly uloženy během předchozího kontrolního bodu.
    - Zápis záznamu kontrolního bodu do logu.

©2009-2024 db.cs@vsb.cz DS2 – 7. Transakce, zotavení 23 / 43

Transakce, zotavení / Základní techniky zotavení

### Kontrolní body, příklad – počáteční situace 1/4

■ **Systémová chyba** nastala v čase  $t_f$ .

■ **Kontrolní bod  $t_c$**  je posledním kontrolním bodem před vznikem této chyby.

©2009-2024 db.cs@vsb.cz DS2 – 7. Transakce, zotavení 24 / 43

Transakce, zotavení / Základní techniky zotavení

### Kontrolní body, příklad – kontrolní bod

■ V čase kontrolního bodu jsou:

- **Nové hodnoty** aktualizací transakce  $T_1$  zapsány z logu **do databáze**.
- **Nové hodnoty** aktualizací transakcí  $T_2$  a  $T_3$  provedené před  $t_c$  zapsány **do databáze** (před tím jsou do logu zapsány **původní hodnoty** pro UNDO).

©2009-2024 db.cs@vsb.cz DS2 – 7. Transakce, zotavení 28 / 43

- zbytek viz [prezentace 7](#) (slide 23 – 32)

**Příklad [2] (9 bodů)**  
 Popište jednotlivé vlastnosti **ACID**.

Transakce, zotavení / Transakce

Vlastnosti ACID

Každá transakce musí splňovat **vlastnosti ACID**:

- **A - atomičnost (*atomicity*)** – transakce musí být atomická: jsou provedeny všechny operace transakce nebo žádná.
- **C - korektnost (*correctness*)** – transakce převádí korektní stav databáze do jiného korektního stavu databáze, mezi začátkem a koncem transakce nemusí být databáze v korektním stavu.
- **I - izolovanost (*isolation*)** – transakce jsou navzájem izolovány: změny provedené jednou transakcí jsou pro ostatní transakce viditelné až po provedení COMMIT.
- **D - trvalost (*durability*)** – jakmile je transakce potvrzena, změny v databázi se stávají trvalými i po globální chybě.

©2009-2024 db.cs@vsb.cz
DS2 – 7. Transakce, zotavení
12 / 43

- **Transakce je atomická** ⇒ Transakce se nemohou zanořovat.
- **Izolovanost v ACID** říká, že transakce se navzájem **neovlivňují**, tzn. transakce nesmí vidět **žádné aktualizace** ostatních transakcí, které nebyly potvrzeny před začátek této transakce.
- Serializovatelnost garantuje izolaci transakcí ve smyslu podmínky **ACID**.

2. na souběžných plánech ukažte, jak se řeší problém nepotvrzené závislosti

Řízení souběhu / Zamykání – řešení problémů souběhu

2. Řešení problému nepotvrzené závislosti

Transakce A	Čas	Transakce B
-	$t_1$	WRITE $t$ (získán zámek X na $t$ )
READ $t$ (požadavek na zámek S na $t$ )	$t_2$	-
wait	$t_3$	COMMIT/ROLLBACK (uvolnění zámku X na $t$ )
opakuji: READ $t$ (získán zámek S na $t$ )	$t_4$	

- V čase  $t_2$  není transakci A přidělen zámek S, A přejde do stavu čekání na uvolnění zámku X transakcí B.
- V čase  $t_3$ , při ukončení transakce B, je automaticky uvolněn zámek X pro  $t$ .
- V čase  $t_4$  pokračuje A v činnosti, hodnota záznamu  $t$  je hodnota z času  $t_1$  (při COMMIT transakce B) nebo před  $t_1$  (při ROLLBACK). Problém souběhu je vyřešen.

©2009-2024 db.cs@vsb.cz
DS2 – 9. Řízení souběhu I
19 / 27

### 3. popište rozdíl mezi neopakovatelným čtením a výskytem fantomů

Rízení souběhu / Problémy souběhu

#### 4. Neopakovatelné čtení

Transakce A	Čas	Transakce B
READ $t$	$t_1$	
	$t_2$	WRITE $t$
READ $t$	$t_3$	

V čase  $t_1$  a  $t_3$  získá transakce A odlišnou hodnotu záznamu  $t$ , mluvíme o **neopakovatelné čtení**.

©2009-2024 db.cs@vsb.cz DS2 – 9. Řízení souběhu I 10 / 27

Rízení souběhu / Zamykání – řešení problémů souběhu

#### 4. Řešení problému neopakovatelné čtení

Transakce A	Čas	Transakce B
READ $t$ (požadavek na zámek S pro $t$ )	$t_1$	
	$t_2$	WRITE $t$ (požadavek na zámek X pro $t$ )
READ $t$	$t_3$	wait

- V čase  $t_2$  není transakci B přidělen zámek X pro záznam  $t$ , transakce B přejde do stavu čekání na uvolnění zámku S pro  $t$  transakci A.
- V čase  $t_3$  transakce A přečte stejné hodnoty záznamu  $t$  jako v  $t_1$ , problém souběhu **neopakovatelné čtení** je vyřešen.

©2009-2024 db.cs@vsb.cz DS2 – 9. Řízení souběhu I 22 / 27

Rízení souběhu / Problémy souběhu

#### 5. Výskyt fantomů

- Mějme následující tabulku Student:

login	jmeno	rocnik
jan001	Jan	1
mil002	Milan	3

- Mějme plány dvou paralelních transakcí<sup>2</sup>:

Transakce A	Čas	Transakce B
SELECT * from student WHERE rocnik BETWEEN 1 AND 2	$t_1$	-
-	$t_2$	INSERT INTO student VALUES('mar006', 'Marek',2)
	$t_3$	COMMIT
SELECT * from student WHERE rocnik BETWEEN 1 AND 2	$t_4$	-
COMMIT	$t_5$	-

<sup>2</sup>Dostaneme různé výsledky v čase  $t_1$  a  $t_4$ .

©2009-2024 db.cs@vsb.cz DS2 – 9. Řízení souběhu I 11 / 27

Rízení souběhu / Zamykání – řešení problémů souběhu

#### 5. Řešení problému výskyt fantomů

Transakce A	Čas	Transakce B
SELECT * from student WHERE rocnik BETWEEN 1 AND 2	$t_1$	-
-	$t_2$	INSERT INTO student VALUES('mar006', 'Marek',2)
	$t_3$	COMMIT
SELECT * from student WHERE rocnik BETWEEN 1 AND 2	$t_4$	-
COMMIT	$t_5$	-

- Pokud je v čase  $t_1$  zamčen jen záznam pro ročník 1 (záznam pro ročník 2 v tabulce neexistuje), pak problém **vyřešení není**.
- Musím dojít k **zamykání rozsahu dotazu**, nikoli zamykání existujících záznamů v tabulce.

©2009-2024 db.cs@vsb.cz DS2 – 9. Řízení souběhu I 23 / 27

4. **chci udělat update většího počtu řádků, ukažte, jak se to dělá kurzorem (špatně), a jak se to dělá jedním dotazem**

- explicitní kurzor:

```
sql Zkopírovat kód

DECLARE
  CURSOR c_students IS SELECT id, grade FROM Student WHERE grade < 50;
  v_student Student%ROWTYPE;
BEGIN
  OPEN c_students;
  LOOP
    FETCH c_students INTO v_student;
    EXIT WHEN c_students%NOTFOUND;

    -- Aktualizace jednotlivého řádku
    UPDATE Student
    SET grade = grade + 10
    WHERE id = v_student.id;

  END LOOP;
  CLOSE c_students;
END;
```

- jeden SQL dotaz:

```
sql Zkopírovat kód

UPDATE Student
SET grade = grade + 10
WHERE grade < 50;
```

- Použití kurzoru pro aktualizaci většího počtu řádků je neefektivní a náročné na systémové zdroje. Aktualizace jedním SQL dotazem je mnohem rychlejší a efektivnější způsob, jak provést hromadné změny v databázi.

5. napište exekuční plán dotazu na primární klíč (tabulka halda, automatický index na PK)
- o z Discordu, podle MoonSoD:

Index pro primární klíč

### Index pro primární klíč, Oracle

- Uvažujme nyní dotaz na hodnotu primárního klíče, například:  
`select * from Customer where idCustomer=100001;`
- Plán v Oracle:

Operation	Object
-----	-----
SELECT STATEMENT ()	
TABLE ACCESS (BY INDEX ROWID)	CUSTOMER
INDEX (UNIQUE SCAN)	SYS_C00469561
- buffer gets je 3 namísto 1 958 jako u dotazu, který prováděl sekvenční průchod haldou.
- Ačkoli jsme žádný index nevytvořili pomocí `create index`, DBS automaticky vytvořil index pro primární klíč.

©2009-2024 db.cs@vsb.cz DS2 - 12. Vykonávání dotazů v DBS 27 / 96

`SELECT idCustomer FROM Customer WHERE idCustomer = 1`

Operation

SELECT STATEMENT ()  
TABLE ACCESS (BY INDEX ROWID)  
INDEX (UNIQUE SCAN)

**Příklad [1] (9 bodů)**

Napište minispecifikaci transakce z vašeho semestrálního projektu (nepište: převod částky z účtu, kaskádové mazání atd.). Popište případné vstupní parametry i výstup funkce.

- opovažte se tam někdo napsat moji minispecifikaci, naučte se svoji

Transakce **AddCondition** (**p\_cat\_id**, **p\_condition\_id**):

Vstupy:

- **p\_cat\_id** – ID kočky přidávané ve formuláři
- **p\_condition\_id** – ID nemoci/zranění, kterou kočka má

Popis:

Ke zdravotní historii kočky přidá nový záznam o nemoci. Pokud už je kočka přiřazená do místnosti, zkontroluje, jestli jsou v místnosti další nemocné kočky. Pokud ano, a mají odlišnou nemoc, přeřadí kočku do místnosti, kde jsou pouze kočky se stejnou nemocí, nebo která je prázdná. V případě úspěšného provedení vrátí *true*, v opačném případě *false*.

1. Ulož si ID místnosti do proměnné *room\_id*, do které je kočka aktuálně přiřazena.  

```
SELECT room_id
FROM CatRoom
WHERE cat_id = p_cat_id;
```
2. Zkontroluj, jestli jsou v této místnosti jiné kočky, které mají odlišnou nemoc.  

```
SELECT COUNT(*)
FROM CatRoom cr
JOIN MedicalHistory mh ON cr.cat_id = mh.cat_id
WHERE cr.room_id = room_id
AND cr.cat_id != p_cat_id
AND mh.condition_id != p_condition_id;
```
3. Pokud předchozí count vrátil číslo větší, než 0, najdi místnost, ve které jsou buď kočky se stejnou nemocí, nebo je prázdná. Ulož výsledek do *new\_room\_id*.  

```
SELECT TOP 1 room_id
FROM CatRoom cr
JOIN MedicalHistory mh ON cr.cat_id = mh.cat_id
JOIN Room r ON cr.room_id = r.room_id
WHERE mh.condition_id = p_condition_id
ORDER BY r.capacity ASC
UNION
SELECT room_id
FROM Room
WHERE room_id NOT IN (
    SELECT room_id
    FROM CatRoom
)
ORDER BY r.capacity ASC;
```
4. Pokud nebyla nalezena vhodná místnost, ukonči transakci.
5. Přemísti kočku do nalezené vhodné místnosti.  

```
UPDATE CatRoom
SET room_id = new_room_id
WHERE cat_id = p_cat_id;
```
6. Přiřaď kočce danou nemoc.  

```
INSERT INTO MedicalHistory (cat_id, condition_id, treatment_date)
VALUES (p_cat_id, p_condition_id, CURRENT_TIMESTAMP);
```
7. Pokud všechny kroky proběhly v pořádku, proved *commit* transakce, vrať *true*.  
 Pokud se v průběhu vyskytly chyby, proved *rollback*, vrať *false*.

**Příklad [2] (9 bodů)**

Popište jednotlivé vlastnosti **ACID**.

- už tady jednou je

### Příklad [3] (10 bodů)

Napište, jakým způsobem **není** při úrovni izolace transakce **READ UNCOMMITTED** dodrženo dvoufázové uzamykání a k jakému problému souběhu tak může dojít (který řeší hned následující úroveň izolace transakcí). Ukažte na příkladu dvou konkrétních souběžných transakcí (použijte operace READ/WRITE pro jeden záznam).

- výlučné zámky by měly být uvolněny až po ukončení transakce, ale při read uncommitted jsou uvolněny už před ním, a proto může dojít ke špinavému čtení

Úroveň izolace transakce v SQL

### Výjimky souběhu pro různé úrovně izolace

V následující tabulce vidíme různé výjimky souběhu, které mohou či nemohou nastat pro různé úrovně izolace transakcí:

Úroveň izolace	Špinavé čtení	Neopakovatelné čtení	Výskyt fantomů
READ UNCOMMITTED	Ano	Ano	Ano
READ COMMITTED	Ne	Ano	Ano
REPEATABLE READ	Ne	Ne	Ano
SERIALIZABLE	Ne	Ne	Ne

©2009-2024 db.cs@vsb.cz DS2 – 11. Řízení souběhu II 16 / 53

Úroveň izolace transakce v SQL

### Špinavé čtení

V případě úrovně RU může nastat tzv. *špinavé čtení*, tedy transakce může načíst data změněná a dosud nepotvrzená jinou transakcí, viz:

Transakce A	Čas	Transakce B
-	$t_2$	UPDATE student SET rocnik=1 WHERE id='jan001'
SELECT * from student WHERE id='jan001'	$t_3$	-
COMMIT	$t_4$	-
-	$t_5$	COMMIT/ROLLBACK

V tomto případě jsou tedy před ukončením transakce **uvolněny i výlučné zámky**.

©2009-2024 db.cs@vsb.cz DS2 – 11. Řízení souběhu II 17 / 53

- příklad dvou konkrétních souběžných transakcí (používající READ/WRITE pro jeden záznam  $t$ ):

Řízení souběhu / Problémy souběhu

### 2. Problém nepotvrzené závislosti

Transakce A	Čas	Transakce B
-	$t_1$	WRITE $t$
READ $t$	$t_2$	-
-	$t_3$	ROLLBACK

- Transakce A se čtením záznamu v čase  $t_2$  stala **závislou na nepotvrzeném zápisu** transakce B v čase  $t_1$ . A provedla **špinavé čtení** (dirty read).
- Transakce A pracuje po čase  $t_3$  s **neplatnými hodnotami záznamu**, tedy s hodnotami získanými v čase  $t_2$ , ačkoli platné hodnoty jsou hodnoty z času před  $t_1$ .

©2009-2024 db.cs@vsb.cz DS2 – 9. Řízení souběhu I 7 / 27





#### Příklad [4] (9 bodů)

Popište jaké datové struktury se ve vytvoří po zadání příkazu `CREATE TABLE ...` v databázové systému pro uložení záznamů tabulky. Popište také, jaké informace tyto datové struktury budou obsahovat.

Tabulka typu haldy – heap table

### Fyzická implementace relačního datového modelu

- Základní úložiště pro tabulku relačního datového modelu je **tabulka typu haldy (heap table)**:
  - stránkované pole.
- **Stránky/bloky**:
  - Záznamy jsou uloženy ve stránkách/blocích o velikosti nejčastěji 8kB (násobky alokačních jednotek souborového systému - nejčastěji 2kB).
  - Stránky slouží pro efektivní výměnu dat mezi pamětí (*cache buffer*) a diskem.
- **Vyhledávání je sekvenční**: složitost  $O(n)$ : DBS prochází všechny stránky a všechny záznamy.

©2009-2024 db.cs@vsb.cz

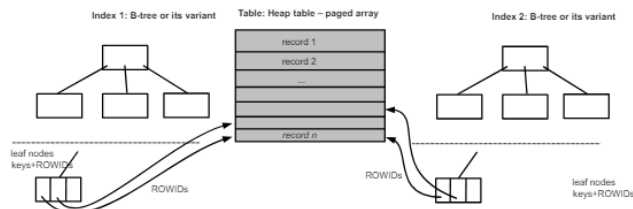
DS2 – 12. Vykonávání dotazů v DBS

4 / 56

Základní úložiště pro relační datový model

### Základní úložiště pro relační datový model

- Ke každé tabulce je vytvořena haldy a index typu B-strom pro primární klíč a jedinečné atributy (*unique*).
- Index obsahuje v položkách stránek dvojice (hodnota primárního klíče, **ROWID**), kde ROWID (v SQL Server **RID**) je odkaz na záznam do haldy.



©2009-2024 db.cs@vsb.cz

DS2 – 12. Vykonávání dotazů v DBS

32 / 56

Základní úložiště pro relační datový model

### ROWID

- Pomocí **ROWID** identifikujeme záznamy v haldě.
- ROWID je 8-10B hodnota (v Oracle 10B, v SQL Server 8B) skládající se z:
  - čísla bloku,
  - pozice záznamu v haldě.
- Proč DBS nepoužívá paměťový ukazatel?
  - protože bloky mohou být umístěny na disku.

©2009-2024 db.cs@vsb.cz

DS2 – 12. Vykonávání dotazů v DBS

33 / 56

## 1. odložená aktualizace

Transakce, zotavení / Základní techniky zotavení

1. Zotavení odloženou aktualizací 1/2

■ Zotavení odloženou aktualizací (angl. **deferred update**):

■ Aktualizace záznamů jsou během transakce ukládány **do paměti**.

■ Po potvrzení transakce, jsou nové hodnoty uloženy do logu a poté do databáze<sup>2</sup> (na disku).

Transaction

U1 U2 U3 U4 U5 COMMIT

Cache Buffer  
(main memory)

Log  
(disk)

Database  
(disk)

↓ U1-U5, COMMIT

↓ U1-U5

<sup>2</sup>Dopředný zápis do logu.

©2009-2024 db.cs@vsb.cz DS2 – 7. Transakce, zotavení 19 / 43

Transakce, zotavení / Základní techniky zotavení

1. Zotavení odloženou aktualizací 2/2

■ Zotavení odloženou aktualizací, vlastnosti:

■ **NO-UNDO/REDO:**

■ Pokud transakce selže, **není nutné provést UNDO**: databáze nebyla aktualizovaná před **COMMIT**.

■ **REDO** bude provedeno v případě (pomocí nových hodnot v logu), kdy systém zapsal aktualizace do logu, ale k zapsání změn do databáze nedošlo.

■ **Výkon** techniky je spíše **vyšší** (minimalizace diskových operací)<sup>3</sup>. V případě dlouhých transakcí hrozí **přetečení paměti**.

<sup>3</sup>Používá se například pro In-Memory DBS.

©2009-2024 db.cs@vsb.cz DS2 – 7. Transakce, zotavení 20 / 43

**Příklad [6] (9 bodů)**

Napište příklad kurzoru, který **není** možné zapsat jedním SQL příkazem. Uveďte konkrétní příklad kurzoru i SQL dotazu.

```
sql Zkopírovat kód

DECLARE
  CURSOR cur_students IS
    SELECT student_id, grade
    FROM Students;

  v_student_id Students.student_id%TYPE;
  v_grade Students.grade%TYPE;
BEGIN
  OPEN cur_students;
  LOOP
    FETCH cur_students INTO v_student_id, v_grade;
    EXIT WHEN cur_students%NOTFOUND;

    IF v_grade < 60 THEN
      UPDATE Students
      SET status = 'Fail'
      WHERE student_id = v_student_id;
    ELSE
      UPDATE Students
      SET status = 'Pass'
      WHERE student_id = v_student_id;
    END IF;
  END LOOP;
  CLOSE cur_students;
END;
```

- SQL příkazy pro dosažení stejného výsledku bez kurzoru:

Aktualizace studentů, kteří neprošli

```
sql Zkopírovat kód

UPDATE Students
SET status = 'Fail'
WHERE grade < 60;
```

Aktualizace studentů, kteří prošli

```
sql Zkopírovat kód

UPDATE Students
SET status = 'Pass'
WHERE grade >= 60;
```

- Tento příklad ukazuje situaci, kdy je kurzor užitečný pro provádění více kroků logiky na jednotlivé řádky, což by bylo obtížné nebo neefektivní provádět pomocí jednoho SQL příkazu. Kurzor umožňuje iterativní zpracování každého řádku výsledku dotazu s možností provádět složitější operace a podmínky než jen prosté aktualizace dat.

**Příklad [1] (9 bodů)**  
Popište co je to transakce, jak začíná a jak je ukončena. Jaký je rozdíl mezi více databázovými operacemi, které jsou uvedeny v transakci resp. jsou zapsány bez transakce?

Transakce, zotavení / Transakce

Transakce

Transakce je logická (nedělitelná, atomická) jednotka práce s databází, obsahující více databázových operací (select, update, delete, insert), která začíná operací **BEGIN TRANSACTION** a končí operacemi **COMMIT** nebo **ROLLBACK**.

- **COMMIT** – úspěšné ukončení transakce. Transakce byla úspěšně dokončena, databáze je nyní v korektním stavu, a všechny aktualizace (update/delete/insert) transakce jsou trvale uloženy v databázi (jsou potvrzeny).
- **ROLLBACK** – neúspěšné provedení transakce. Jedna z operací transakce nebyla korektně ukončena, všechny aktualizace provedené transakcí musí být zrušeny.
- Transakce je atomická ⇒ Transakce se nemohou zanořovat.

©2009-2024 db.cs@usb.cz DS2 – 7. Transakce, zotavení 5 / 43

Transakce, zotavení / Transakce

Korektní stav databáze, potvrzovací bod

- Úkolem transakce je převést korektní stav databáze na jiný korektní stav, po dílčích operacích transakce, může být databáze v nekorektním stavu.
- Operace **COMMIT** zavádí tzv. **potvrzovací bod** (angl. **commit point**, ve smyslu bod v čase, ve kterém je databáze v korektním stavu.
  - **ROLLBACK** vrací databázi do stavu, ve kterém byla při vykonání **BEGIN TRANSACTION**, vrací tedy databázi k předchozímu potvrzovacímu bodu (tedy do korektního stavu).

©2009-2024 db.cs@usb.cz DS2 – 7. Transakce, zotavení 6 / 43

### Operace s transakcí:

Transakce v databázových systémech je soubor operací, který se provádí jako celek. Pokud jedna z operací selže, celá transakce je vrácena zpět (rollback). Transakce zajišťuje dodržení ACID, umožňuje vrácení změn v případě chyby.

### Operace bez transakcí:

Když jsou databázové operace prováděny bez transakce, každá operace je provedena samostatně a okamžitě zapsána do databáze. To znamená, že:

- 1) pokud jedna operace selže, ostatní provedené operace zůstávají zapsány,
- 2) částečně provedené operace mohou vést k nekonzistentnímu stavu databáze,
- 3) současně prováděné operace mohou zasahovat do sebe navzájem a
- 4) každá úspěšná operace je ihned trvale zapsána do databáze - chybí možnost vrácení provedených změn.

**Příklad [2] (9 bodů)**

Popište bezpečností útok **SQL injection**, včetně kódu, který se pokusí útočník přidat do SQL dotazu. Jak je možné se takovému útoku bránit?

Útok SQL Injection

## Útok SQL Injection

- Text získaný od uživatele (resp. vně systému), např. z webového formuláře:
- Pomocí spojení řetězců pak vytvoříme dotaz, např.:

```
OPEN c FOR
'SELECT fname, lname, salary '
|| 'FROM employees '
|| 'WHERE lname=''' || p_lname
|| ''';
```

Útok SQL Injection

## Útok SQL Injection

- Pokud uživatel (v tomto případě útočník) zadá místo příjmení například: X' or 1=1 --
- Výsledkem je řetězec:

```
SELECT fname, lname, salary
FROM employees
WHERE lname='X' or 1=1 --'
```

- Combo boxy formuláře problém neřeší, útočník může vložit text přímo do URL<sup>2</sup>:  
`http://server/stranka?vstup=X' or 1=1 --`

<sup>2</sup>(Hodnota v ukázce není zakódovaná: mezery → %20, atd.)

Útok SQL Injection

## Jak se vyhnout SQL Injection v PL/SQL?

- Pokud to jde, používejte **statické dotazy**.
- V dynamických dotazech používejte **vázané proměnné** pro nastavení hodnoty atributu.
- Pod daným uživatelem v databázi nemějte vytvořené tabulky, které nenáleží k dané aplikaci (tj. nastavte správné **řízení přístupu**). Tímto opatřením budete minimalizovat následky útoku.

## Jak se vyhnout SQL Injection v PL/SQL?



- Tam kde hodnota od uživatele **ovlivňuje** řetězec dotazu (tj. nelze použít vázané proměnné):
  - **Nepoužíváme** hodnotu přímo (např. převod řetězce získaného od uživatele na číslo).
  - Pokud bude text od uživatele použit jako jméno objektu z databáze (např. tabulky), zkontrolujeme či upravíme text (může obsahovat potenciálně nebezpečné znaky ' nebo ") pomocí **DBMS\_ASSERT** nebo zkontrolujeme existenci objektu v **systémovém katalogu** (např. USER\_TABLES).

## Jak se vyhnout SQL Injection?



Používejte vázané proměnné.

■ **Nepoužívejte:**

```
'SELECT * FROM employees
  WHERE fname = ''' || p_fname ||
        ''' AND lname = ''' || p_lname || ''';
```

■ **Používejte:**

```
'SELECT * FROM employees
  WHERE fname = :1 AND lname = :2';
```

Zamezíte SQL Injection.

### Příklad [3] (10 bodů)

Napište, jakým způsobem **není** při úrovni izolace transakce **READ COMMITTED** dodrženo dvoufázové uzamykání a k jakému problému souběhu tak může dojít (který řeší vyšší úroveň izolace transakce). Ukažte na příkladu dvou souběžných transakcí (použijte operace READ/WRITE pro jeden záznam).

## Úroveň izolace READ COMMITTED

- Každý dotaz spuštěný transakcí vidí pouze **data potvrzená před začátkem dotazu (ne transakce!)**.
- Tato úroveň izolace je vhodná v případě prostředí s málo konflikty souběhu.
- Dotaz v takovéto transakci nemůže přečíst data potvrzená během vykonávání dotazu. Pokud např. dotaz prochází celou tabulkou s miliónem záznamů a další transakce aktualizuje záznam 150 000, pak původní transakce nevidí tuto změnu.
- Na druhou stranu, mezi jednotlivými dotazy může transakce vidět aktualizace provedené jinými transakcemi: může dojít k **neopakovatelnému čtení** (S zámky jsou uvolněny před ukončením transakce) a **výskytu fantomů**.

## Neopakovatelné čtení 1/2

- V případě úrovně **RC (a nižší)** je umožněno tzv. **neopakovatelné čtení**.
- V tomto případě příkaz **SELECT** požaduje **sdílený zámek** na záznam, **SRBD** ale nedodrží dvoufázový zamykací protokol a zámky mohou být **uvolněny před ukončením transakce**.
- **Výlučné zámky** jsou ovšem **uvolněny až na konci transakce**.
- V případě úrovně **SR** a **RR** k této výjimce nedochází, v případě úrovně **RC** a **RU** se může tento problém objevit.

## Neopakovatelné čtení 2/2

Transakce A	Čas	Transakce B
SELECT * from student WHERE id='jan001'	$t_1$	-
-	$t_2$	UPDATE student SET rocnik=1 WHERE id='jan001'
SELECT * from student WHERE id='jan001'	$t_3$	COMMIT
COMMIT	$t_4$	-

## Příklad, Read Committed<sup>5</sup>

Transaction 1	Time	Transaction 2
set transaction isolation level read committed;	t1	
select * from a;	t2	
(result: 2 records)		
	t3	delete from a;
	t4	commit;
select * from a;	t5	
(result: 0 records)		
commit;		
select * from a;	t6	
(result: 0 records)		

*Poznámka:* T1 uvolní dříve zámky S, T2 smaže záznamy a po commit vidi T1 0 záznamů. V předchozím případě T1 viděla stejný výsledek obou dotazů.

<sup>5</sup><https://asktom.oracle.com/pls/apex/asktom.search?tag=transaction-isolation-level>

**příklad [4] (9 bodů)**

Pro složený index vytvořený pro atributy (char(6) login, varchar(20) last\_name, int department) napište dva SQL dotazy, které budou tento index **využívat**, a dva dotazy, které jej **nebudou** využívat. Napište zdůvodnění. Poznámka: dotazy musí obsahovat selekci na nejméně dva ze tří atributů indexu.

- SQL dotazy, které využívají složený index:

- a) Dotaz s podmínkami na *login* a *last\_name*:

```
sql Zkopírovat kód  
  
SELECT * FROM Student WHERE login = 'abc123' AND last_name = 'Novak';
```

- Tento dotaz využívá první dva atributy složeného indexu (*login* a *last\_name*), což umožňuje efektivní použití indexu.

- b) Dotaz s podmínkami na všechny tři atributy:

```
sql Zkopírovat kód ovát kód  
  
SELECT * FROM Student WHERE login = 'def456' AND last_name = 'Svoboda' AND department = 10;
```

- Tento dotaz využívá všechny tři atributy složeného indexu, což zajišťuje maximální využití indexu pro rychlé vyhledávání.

- SQL dotazy, které nevyužívají složený index:

- a) Dotaz s podmínkami na *last\_name* a *department*:

```
sql Zkopírovat kód  
  
SELECT * FROM Student WHERE last_name = 'Novak' AND department = 20;
```

- Tento dotaz nevyužívá první atribut složeného indexu (*login*), což znamená, že index nebude použit a databázový systém bude muset provést sekvenční skenování tabulky.

- b) Dotaz s funkcí na *login* a podmínkou na *department*:

```
sql Zkopírovat kód  
  
SELECT * FROM Student WHERE SUBSTR(login, 1, 3) = 'abc' AND department = 30;
```

- Použití funkce SUBSTR na indexovaném atributu (*login*) způsobí, že index nebude použit, protože indexované hodnoty jsou modifikovány funkcí, což znemožňuje efektivní vyhledávání pomocí indexu.

- Dotazy, které využívají index: využívají sekvenční pořadí atributů ve složeném indexu a zahrnují *login* jako první atribut.
- Dotazy, které nevyužívají index: buď nezahrnují první atribut složeného indexu (*login*), nebo na indexovaný atribut aplikují funkci, což znemožňuje použití indexu pro rychlé vyhledávání.



#### Příklad [5] (9 bodů)

Uvažujte techniku zotavení **UNDO/REDO**. Popište, co se ukládá do logu a databáze v okamžiku kontrolního bodu a co se ukládá do logu a databáze v okamžiku COMMIT konkrétní transakce. Jak jsou zapsané informace využity pro operace UNDO resp. REDO vykonávané při zotavení databáze?

#### ■ Pravidlo dopředného zápisu do logu (write-ahead log rule):

- Všechny aktualizace musí být zapsány **do logu před zápisem do databáze**. Před ukončením operace COMMIT, je do logu zapsán tzv. **COMMIT záznam**.

Transakce, zotavení / Transakce

### Korektní stav databáze, potvrzovací bod

- Úkolem transakce je převést **korektní stav databáze** na jiný **korektní stav**, po dílčích operacích transakce, může být databáze v **nekorektním stavu**.
- Operace **COMMIT** zavádí tzv. **potvrzovací bod** (angl. **commit point**), ve smyslu bod v čase, ve kterém je databáze v korektním stavu.
  - **ROLLBACK** vrací databázi do stavu, ve kterém byla při vykonání **BEGIN TRANSACTION**, vrací tedy databázi k předchozímu potvrzovacímu bodu (tedy do korektního stavu).

©2009-2024 db.cs@vsb.cz DS2 – 7. Transakce, zotavení 6 / 43

Transakce, zotavení / Základní techniky zotavení

### 3. Kombinovaná technika – UNDO/REDO

- V praxi se používá kombinovaný **UNDO/REDO algoritmus**:
  - Odložená aktualizace – hrozí přetečení paměti.
  - Okamžitá aktualizace – nízký výkon (vysoký počet diskových operací).
- **Zápis aktualizací**:
  - Nové hodnoty záznamů aktualizací jsou do logu zapsány po **COMMIT** (pro **REDO**).
  - Po určitém časové intervalu dochází k tzv. **kontrolním bodům** (**check points**). V tomto bodu se provede zápis aktualizací transakcí do databáze:
    - **Aktuálně prováděných transakcí**: před tím jsou do logu uloženy původní hodnoty pro **UNDO**.
    - **Potvrzených transakcí**: jejichž aktualizace nebyly uloženy během předchozího kontrolního bodu.
    - **Zápis záznamu kontrolního bodu** do logu.

©2009-2024 db.cs@vsb.cz DS2 – 7. Transakce, zotavení 23 / 43

- už tady je výše, příp. v [prezentaci 7](#) (slide 23 – 32)

Transakce, zotavení / Základní techniky zotavení

### Algoritmus zotavení 1/2


Po restartu DBS spustí tento algoritmus:

- 1 Vytvoříme dva seznamy transakcí: **UNDO** a **REDO**.
- 2 Do **UNDO** vložíme všechny transakce, které nebyly potvrzeny před posledním kontrolním bodem. Seznam **REDO** je prázdný.
- 3 DBS prochází záznamy v logu od záznamu posledního kontrolního bodu.
  - 1 Pokud je pro transakci *T* nalezen v logu záznam **COMMIT**, *T* se přesune ze seznamu **UNDO** do seznamu **REDO**.

©2009-2024 db.cs@vsb.cz DS2 – 7. Transakce, zotavení 33 / 43

Transakce, zotavení / Základní techniky zotavení

## Algoritmus zotavení 2/2





- DBS prochází log zpětně a **ruší aktualizace** transakcí ze seznamu **UNDO**, zapisuje do databáze **původní hodnoty** uložené v logu.
- DBS prochází logem dopředu a **přeprocovává aktualizace** transakcí ze seznamu **REDO**, zapisuje do databáze **nové hodnoty** uložené v logu.
- Databáze je v **korektním stavu**.

©2009-2024 dls.cs@vsb.cz DS2 - 7. Transakce, zotavení 34 / 43

### Příklad [6] (9 bodů)

Mějme tabulky `Zamestnanec` a `Pracoviste`. Tabulka `Zamestnanec` obsahuje atribut `idPracoviste`, což je cizí klíč – primární klíč tabulky `Pracoviste`. K jakému problému dojde, pokud budeme v DAO třídě `ZamestnanecDao` a metodě `SelectAll` načítat pracoviště pomocí `PracovisteDao::Select(id)` pro každý záznam zaměstnance? Jak můžeme tento problém efektivně řešit?

- večer vysvětlí na Discordu MoonSoD

 @proste\_majkee Hele nějaká hodná duše, co by byla schopna odpovědět na tyto otázky, nebo návod, kde přesně to najít v těch prezentacích? Napiste dva sql prikazy kdy s...  
 MoonSoD včera v 21:55  
Pri pracovníkoch je to N+1 problem, pretože pre každého pracovníka vykonas dalsie query, čo je neefektívne, vyriesis to joinom na oddelenie

### 1. kontrolní bod? UNDO/REDO, kam se to ukládá, jaká transakce

- už by tu mělo být

## 2. popsat složený index, pro jaký dotaz je vhodný/nevhodný

Index typu B-strom / Složený klíč indexu

Složený klíč indexu

- Pokud klíč obsahuje více než jeden atribut  $a_1, a_2, \dots, a_k$ , mluvíme o složeném klíči.
- Jaké dotazy umožní B-strom efektivně vykonat?
- Záleží na pořadí atributů v definici `create index`?

©2009-2024 db.cs@vsb.cz

DS2 – 12. Vykonávání dotazů v DBS

49 / 56

Index typu B-strom / Složený klíč indexu

Tabulka OrderItems

IDORDER	IDPRODUCT	UNIT_PRICE	QUANTITY
1	4320	1796023	1
1	7795	28533	9
1	24477	4157	9
1	25231	41566	6
1	34709	1497974067	1
2	19090	62625	8
2	24733	71542	10
2	42795	61306	5
2	48281	95708	10
2	51968	31302	5
2	86756	2274	10
3	35455	1710186106	2
3	36256	2928345	1
3	44360	22510816	3
3	58269	27514	7
3	75299	70638	2
3	81503	35979	6
...			

- Klíče jsou v listových uzlech B-stromu setříděny dle atributů  $a_1, a_2, \dots, a_k$ , tak jak byly uvedeny v definici `create index`.
- Mluvíme o tzv. **lexicografickém uspořádání** (viz třídění slov ve slovníku).
- V tomto případě jsou tedy nejprve uloženy klíče s `idOrder=1`, setříděné dle `idProduct`, pak klíče s `idOrder=2` atd.

©2009-2024 db.cs@vsb.cz

DS2 – 12. Vykonávání dotazů v DBS

50 / 56

- konkrétní příklady v [prezentaci 12](#) (slide 51, 52, 53, 55)

Index typu B-strom / Složený klíč indexu

Složený klíč – specifikace dotazu

- 1 Lexikografickému uspořádání pro klíč  $a_1, a_2, \dots, a_k$  **odpovídají dotazy** obsahující bodové dotazy pro atributy  $a_1, \dots, a_l$ ,  $l \leq k$ , pro atribut  $a_{l+1}$  může být specifikován rozsah, atributy  $a_{l+2}, \dots, a_k$  mohou zůstat nespecifikované.
- 2 Jakýkoli jiný dotaz znamená nevyužití indexu, tedy vykonání dotazu sekvenčním průchodem haldou.
- 3 Pokud DBS odhadne velikost výsledku dle bodu 1 za příliš vysokou, provede dotaz dle bodu 2.

©2009-2024 db.cs@vsb.cz

DS2 – 12. Vykonávání dotazů v DBS

54 / 56

## 3. napsat konkrétní kurzor chybně použitý s jednovětvým SQL, konkrétní kurzor a ekvivalentní SQL dotaz

- to už by tu taky někde mělo být

**4. popsat zamykání u problémů non repeatable, popsat konkrétní plán**

- u RR (to je asi myšleno tím non repeatable) může dojít pouze k výskytu fantomů
- je proto třeba uzamknout celý rozsah dotazu, ne pouze jednotlivé záznamy tabulky
- už je to tady u rozdílů mezi neopakovatelným čtením a výskytem fantomů

**5. něco JMD, dotaz s transakcí/bez transakce**

- to už by tady taky mělo být, rozdíl použití a nepoužití transakce

## 6. co je to DAO, DTO v ORM, příklad?

Objektově-relační mapování (ORM) / Entity a vazby

Návrh ORM 1/2

- Existuje celá řada postupů (**mapování**) jak navrhnout ORM, viz předmět **Vývoj informačních systémů**.
- V DSII budeme používat toto jednoduché mapování:
  - **Jedna třída reprezentuje (většinou) jeden entitní typ** (jeden objekt reprezentuje jeden záznam tabulky):
    - Součást aplikační vrstvy.
    - Názvy: **aplikační objekt, doménový objekt, Data Transfer Object - DTO**.
    - Například: vytvoříme třídu `Person` pro tabulku `Person`.
    - Obecně můžeme mapovat jeden entitní typ na více doménových objektů nebo více entitních typů na jeden doménový objekt.

©2009-2024 db.cs@vsh.cz

DS2 - 13. Objektově-relační mapování

11 / 49

Objektově-relační mapování (ORM) / Entity a vazby

Návrh ORM 2/2

- V DSII budeme používat toto jednoduché mapování:
  - **Jedna třída reprezentuje implementaci funkcí (většinou) nad jednou tabulkou**:
    - Součást datové vrstvy.
    - Název: **Data Access Object - DAO**.
    - Obecně může jedna třída zahrnovat operace pro více tabulek (tiskové sestavy, transakce, ...).
    - Například: vytvoříme třídu `PersonDao` se statickými metodami `Select`, `Insert`, ... pro tabulku `Person`.
    - Implementace funkcí pracujících s více tabulkami umísťujeme do společných tříd, např. `QueriesDao`, `TransactionsDao`.
  - **Pomocné třídy** – přístup do databáze, správa připojení, transakcí, ...
  - Složitější (v některých případech nutné) mapování, viz **Vývoj informačních systémů**.

©2009-2024 db.cs@vsh.cz

DS2 - 13. Objektově-relační mapování

12 / 49

- spousta příkladů k tématu je v [prezentaci 13](#)

Objektově-relační mapování (ORM) / Entity a vazby

Doménový objekt (entita, DTO)

- Entita v ORM slouží jako kontejner pro (nejčastěji) jeden záznam tabulky.
- Mějme tabulku `Order`:

```
create table "Order" (  
  id_order  INTEGER NOT NULL primary key identity ,  
  id_user   INTEGER FOREIGN KEY  
    REFERENCES "User" (id_user),  
  id_staff  INTEGER NOT NULL FOREIGN KEY  
    REFERENCES Staff(id_staff),  
  date_order DATE NOT NULL,  
  price     FLOAT  
);
```

©2009-2024 db.cs@vsh.cz

DS2 - 13. Objektově-relační mapování

13 / 49

Objektově-relační mapování (ORM) / Entity a vazby

Doménový objekt, příklad, implementace v C#

```
public class Order  
{  
  public int id_order { get; set; }  
  public int id_user { get; set; }  
  public int id_staff { get; set; }  
  public DateTime date_order { get; set; }  
  public float? price { get; set; }  
}
```

©2009-2024 db.cs@vsh.cz

DS2 - 13. Objektově-relační mapování

14 / 49