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**.

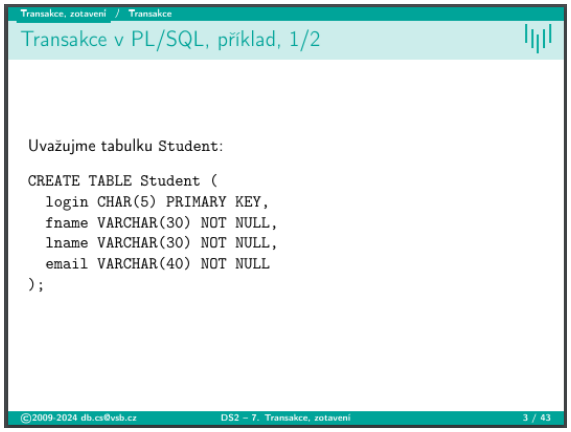
Neopakovatelné čtení 1/2

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

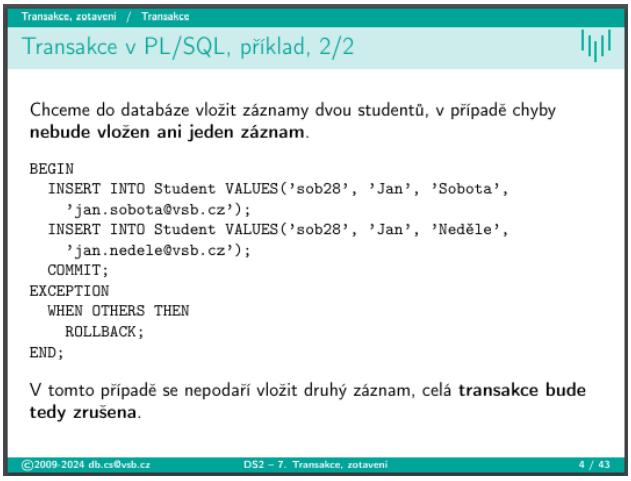


- dvě operace JMD = INSERT, jeden dotaz = SELECT
  - bez transakce:

```sql
INSERT INTO Student VALUES ('sob28', 'Jan', 'Sobota', 'jan.sobota@vsb.cz');
-- Předpokládejme, 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:



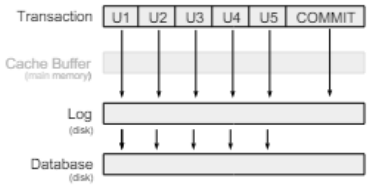
    - případný SELECT by vrátil prázdnou množinu

- 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 *okamžitá aktualizace*. 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[4].
  - Po úspěšném ukončení transakce, je do logu zapsán **COMMIT záznam**.



[4] 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
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
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.

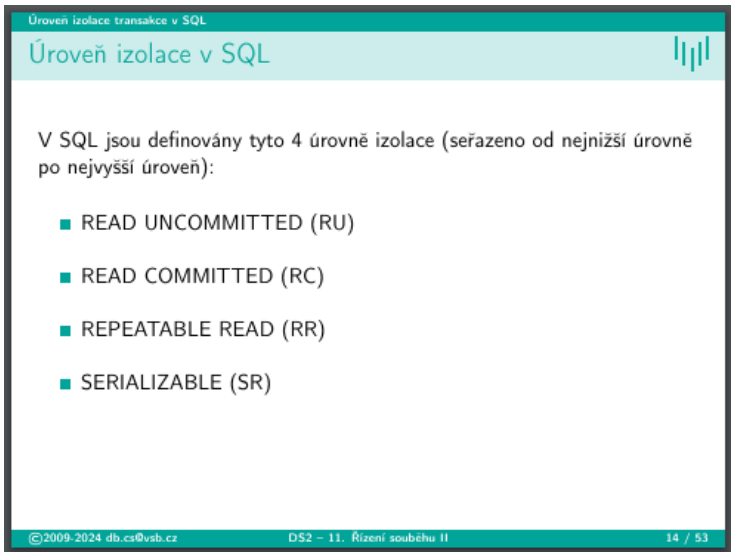
Statické vytvoření tabulky:

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

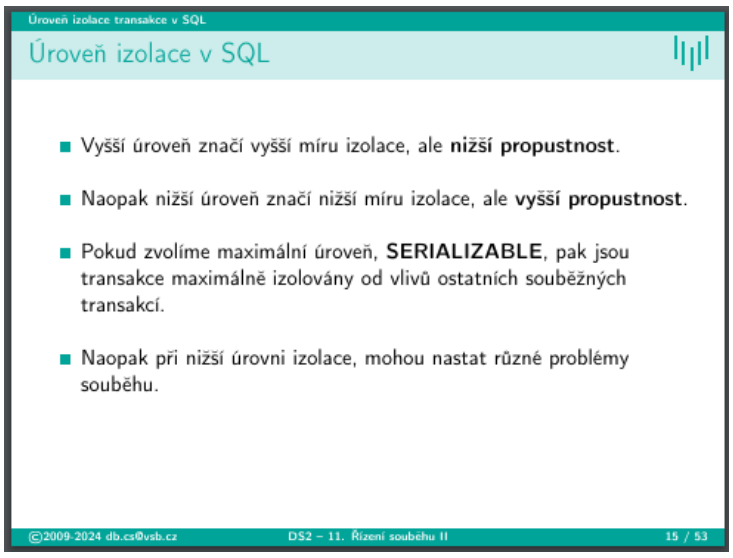
Statický výběr z tabulky:

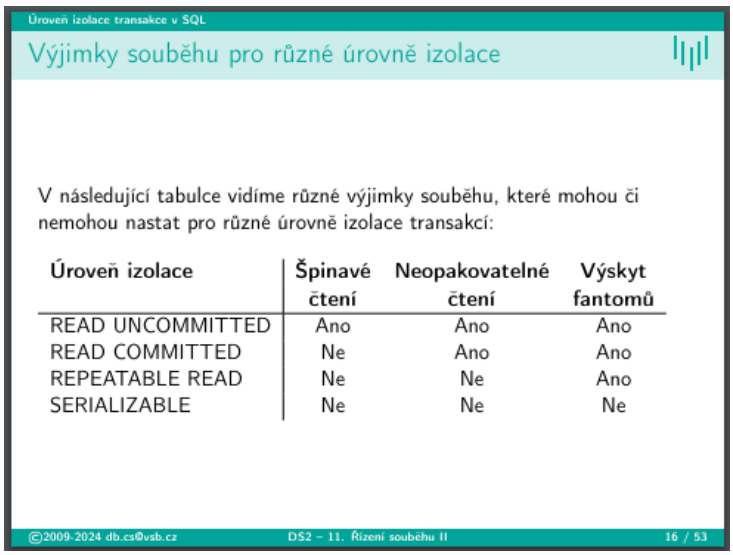
```sql
SELECT name
FROM STATIC_TABLE
WHERE id = 1;
```

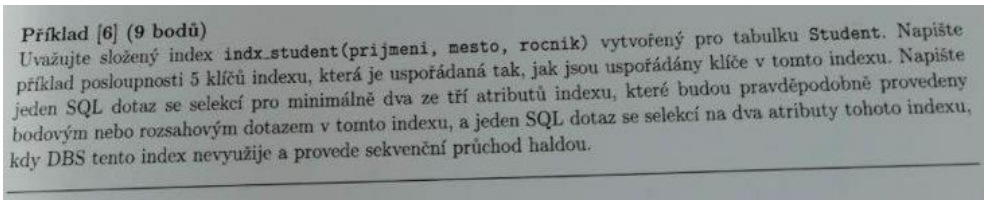
Příklad [5] (9 bodů)
Popište jednotlivé úrovně izolace transakce dle specifikace SQL.



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







Příklad [6] (9 bodů)
Uvažujte složený index `indx_student(prijmeni, mesto, rocnik)` vytvořený pro tabulku `Student`. Napište příklad posloupnosti 5 klíčů indexu, která je uspořádaná 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
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*.

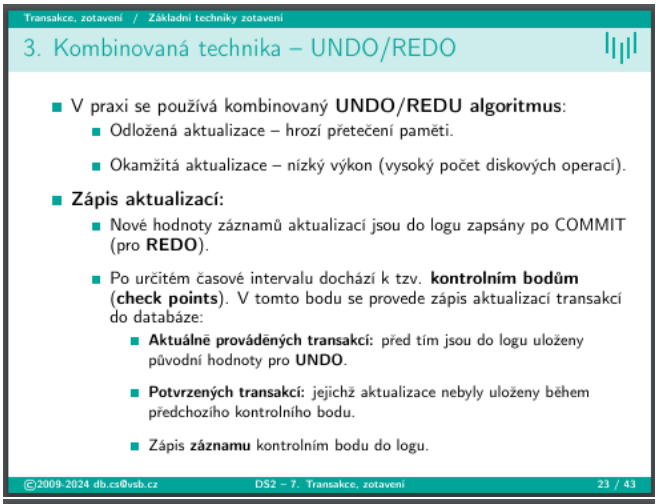
SQL dotaz, který nevyužije index

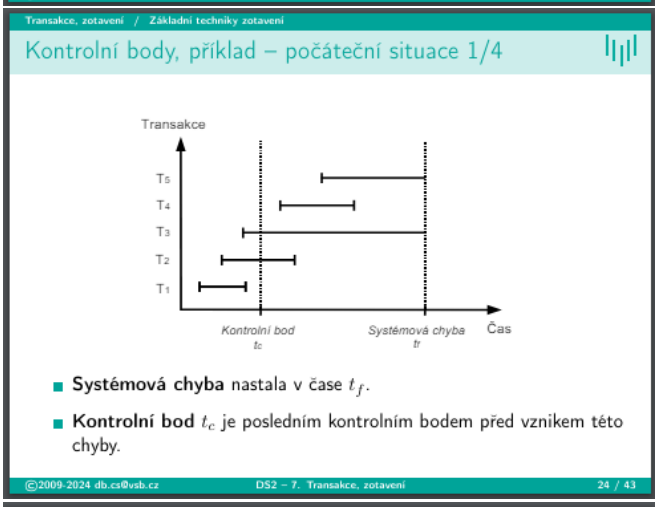
Dotaz na dva atributy, který nevyužije složený index a pravděpodobně provede sekvenční průchod haldou:

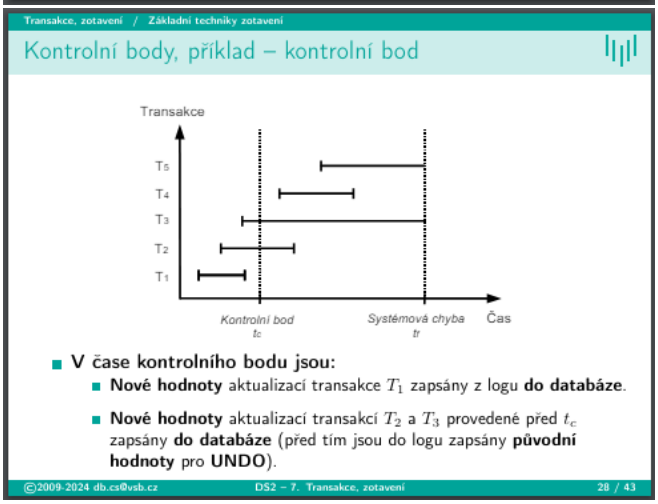
```sql
SELECT * FROM Student
WHERE mesto = 'Brno' AND rocnik = 3;
```

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**







- zbytek viz prezentace 7 (slide 23 – 32)

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

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.

- **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**.

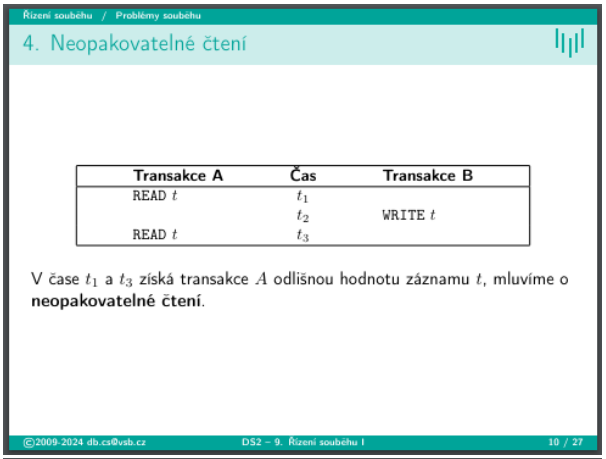
2. na souběžných plánech ukažte, jak se řeší problém nepotvrzené závislosti

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)
opakuj: `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.**

3. popište rozdíl mezi neopakovatelným čtením a výskytem fantomů

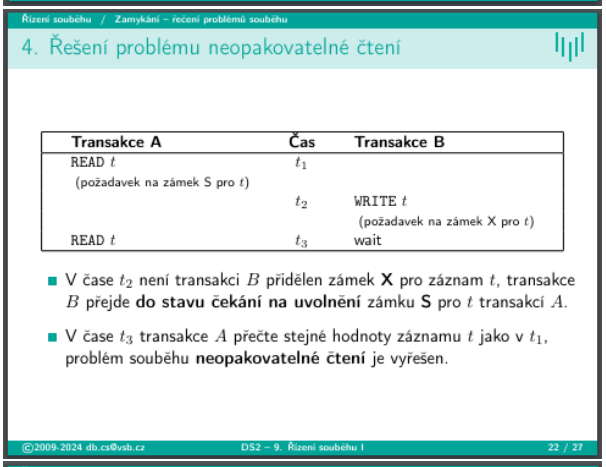


Řízení souběhu / Problémy souběhu

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**.

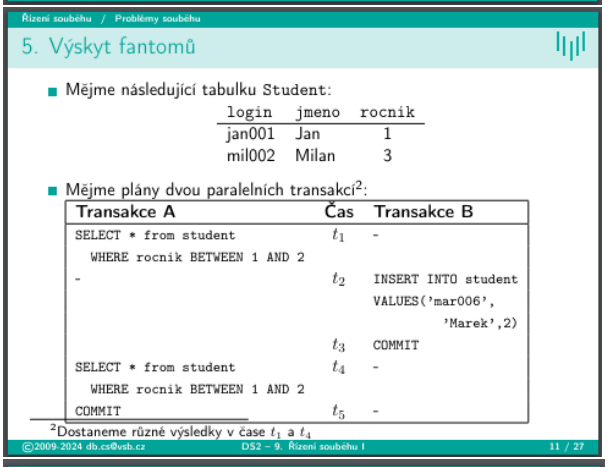


Řízení souběhu / Zamykání – řečení problémů souběhu

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 transakcí A.
- V čase t_3 transakce A přečte stejné hodnoty záznamu t jako v t_1, problém souběhu **neopakovatelné čtení** je vyřešen.



Řízení souběhu / Problémy souběhu

5. Výskyt fantomů

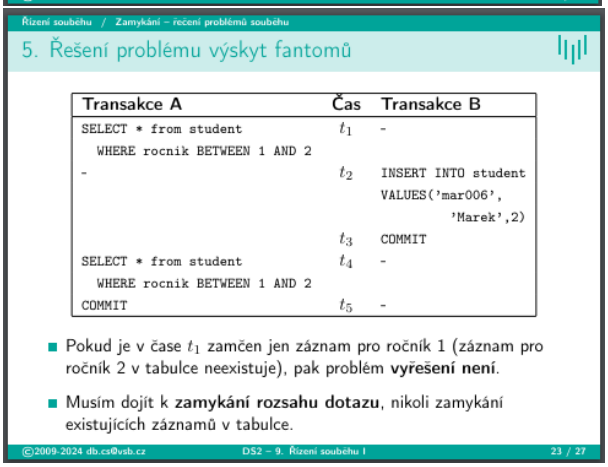
- Mějme následující tabulku `Student`:

login	jmeno	rocnik
jan001	Jan	1
mil002	Milan	3

- Mějme plány dvou paralelních transakcí[2]:

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	-

[2]Dostaneme různé výsledky v čase t_1 a t_4



Řízení souběhu / Zamykání – řečení 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.

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

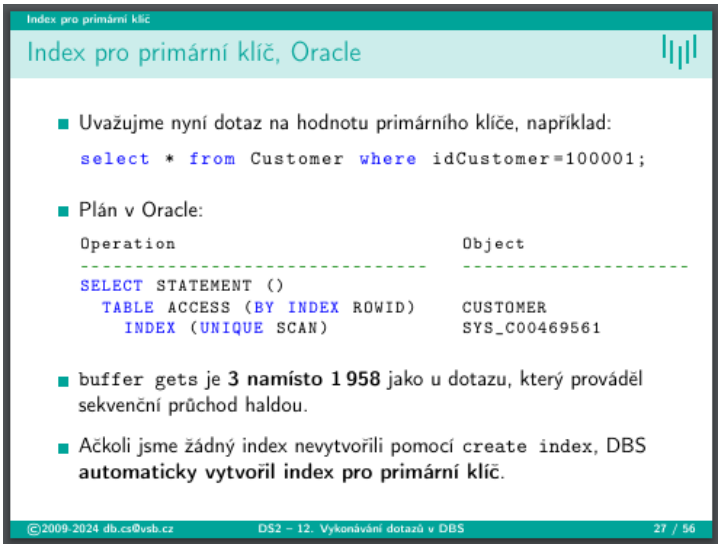
```sql
DECLARE
  CURSOR c_students IS SELECT id, grade FROM Student WHERE grade < 50;
  v_student Student%ROWTYPE;
BEGIN
  OPEN c_students;
  LOOP
    FETCH c_students INTO v_student;
    EXIT WHEN c_students%NOTFOUND;

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

  END LOOP;
  CLOSE c_students;
END;
```

    - jeden SQL dotaz:

```sql
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)**
   - z Discordu, podle MoonSoD:



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);
```

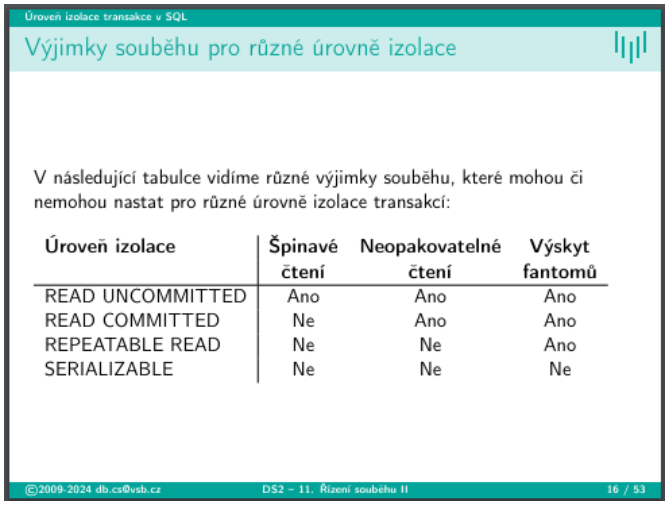
7. Pokud všechny kroky proběhly v pořádku, proveď *commit* transakce, vrať *true*.
Pokud se v průběhu vyskytly chyby, proveď *rollback*, vrať *false*.

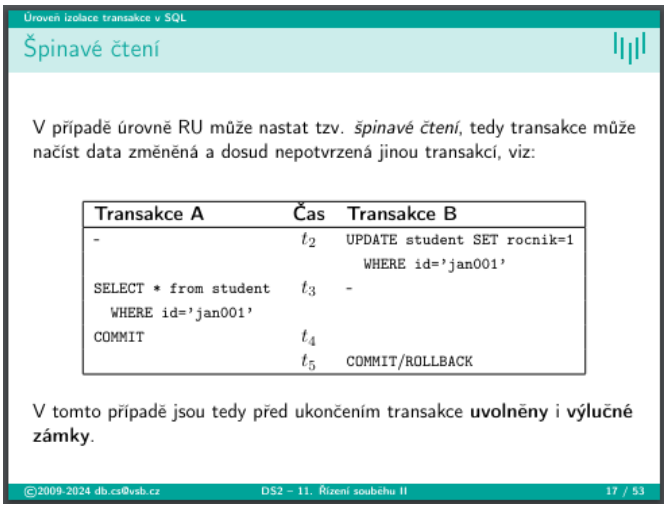
Příklad [2] (9 bodů)
Popište jednotlivé vlastnosti **ACID**.

- už tady jednou je

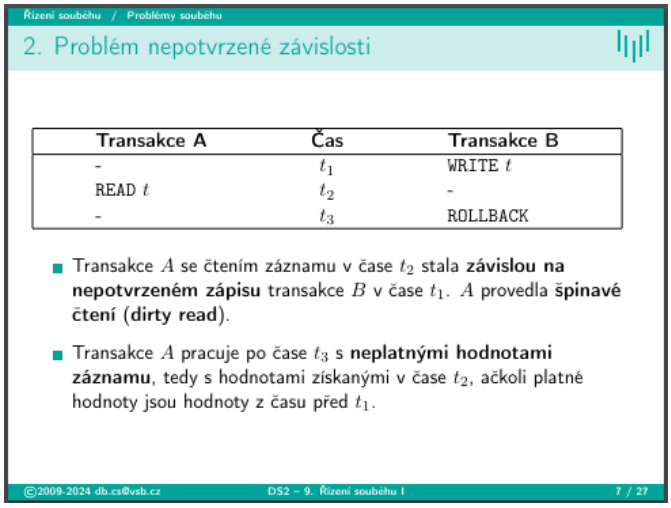
Příklad [3] (10 bodů)
Napište, jakým způsobem **není** při úrovni izolace transakce **READ UNCOMMITTED** dodrženo dvoufázové uzamykání a k jakému problému souběhu tak může dojít (který řeší hned následující úroveň izolace transakcí). Ukažte na příkladu dvou konkrétních souběžných transakcí (použijte 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





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



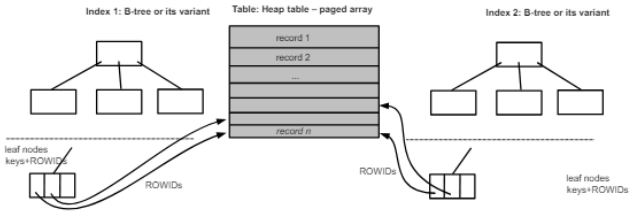
Příklad [4] (9 bodů)
Popište jaké datové struktury se ve vytvoří po zadání příkazu `CREATE TABLE ...` v databázové systému pro uložení záznamů tabulky. Popište také, jaké informace tyto datové struktury budou obsahovat.

Fyzická implementace relačního datového modelu

- Základní úložiště pro tabulku relačního datového modelu je **tabulka typu halda (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.

Základní úložiště pro relační datový model

- Ke každé tabulce je vytvořena halda 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.



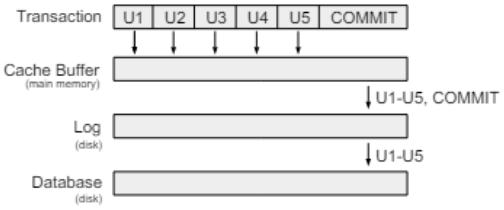
ROWID

- 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.

1. odložená aktualizace

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[2] (na disku).



[2] Dopředný zápis do logu.

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)[3]. V případě dlouhých transakcí hrozí **přetečení paměti**.

[3] Používá se například pro In-Memory DBS.

Příklad [6] (9 bodů)
Napište příklad kurzoru, který **není** možné zapsat jedním SQL příkazem. Uveďte konkrétní příklad kurzoru i SQL dotazu.

---

sql Zkopírovat kód

```sql
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

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

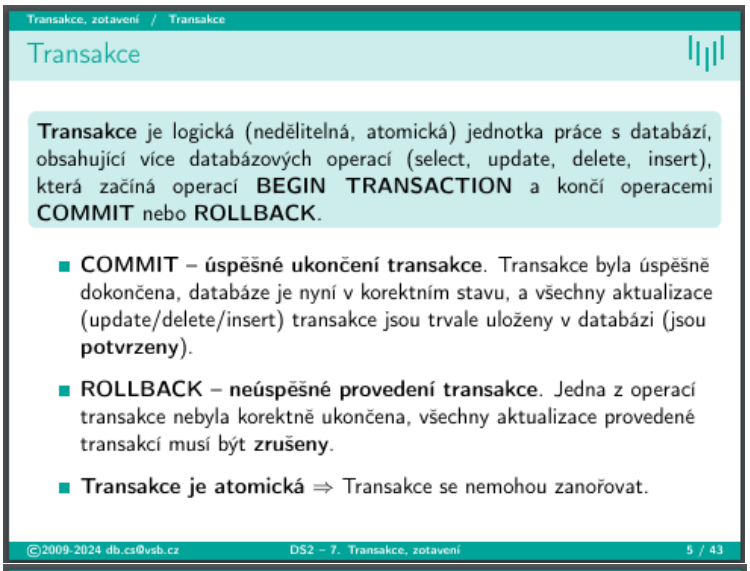
  Aktualizace studentů, kteří prošli

  sql Zkopírovat kód

  ```sql
  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?



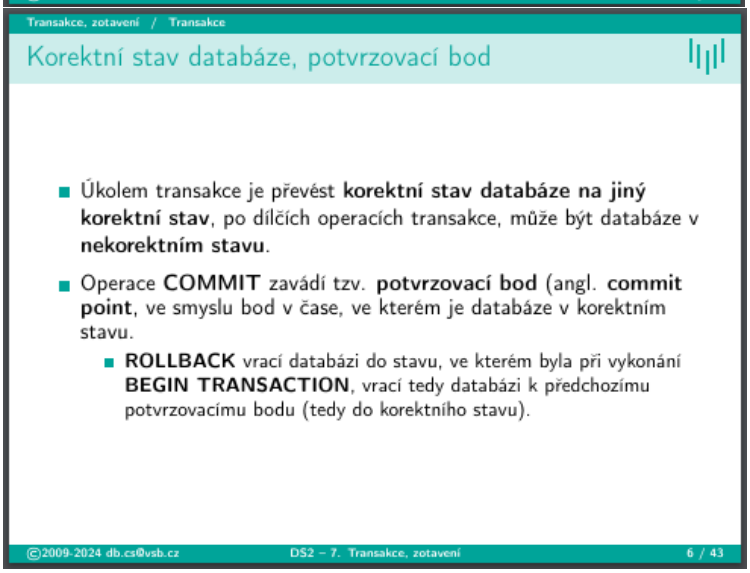
Transakce, zotavení / Transakce

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@vsb.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@vsb.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

- 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

- 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[2]:
  `http://server/stranka?vstup=X' or 1=1 --`

[2] (Hodnota v ukázce není zakódovaná: mezery → %20, atd.)

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 milionem 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, SŘBD ale nedodrží dvoufázový zamykací protokol a zámky mohou být **uvolněny před** ukončením transakce.
- **Výlučné zámky** jsou ovšem **uvolněny až na konci** transakce.
- V případě úrovní SR a RR k této výjimce nedochází, v případě úrovní RC a RU se může tento problém objevit.

Neopakovatelné čtení 2/2

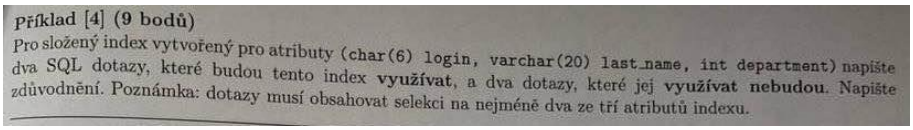
Transakce A	Čas	Transakce B
`SELECT * from student WHERE id='jan001'`	t_1	-
-	t_2	`UPDATE student SET rocnik=1 WHERE id='jan001' COMMIT`
`SELECT * from student WHERE id='jan001'`	t_3	-
`COMMIT`	t_4	

Příklad, Read Committed[5]

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;	t6	
select * from a;		
(result: 0 records)		

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

[5] `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 **využívat nebudou**. 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
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
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
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*:

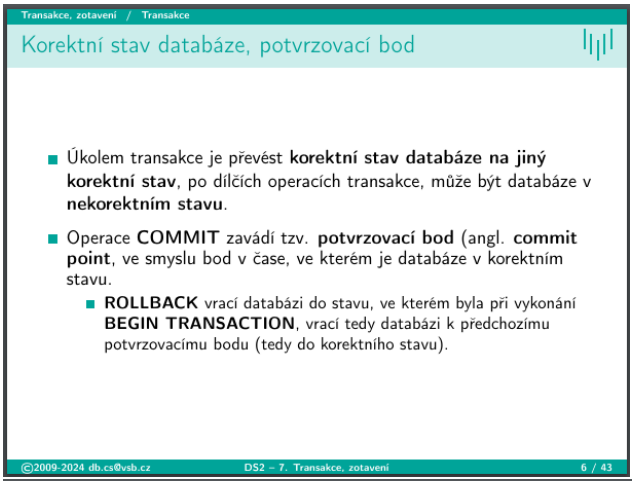
```sql
SELECT * FROM Student WHERE SUBSTR(login, 1, 3) = 'abc' AND department = 30;
```

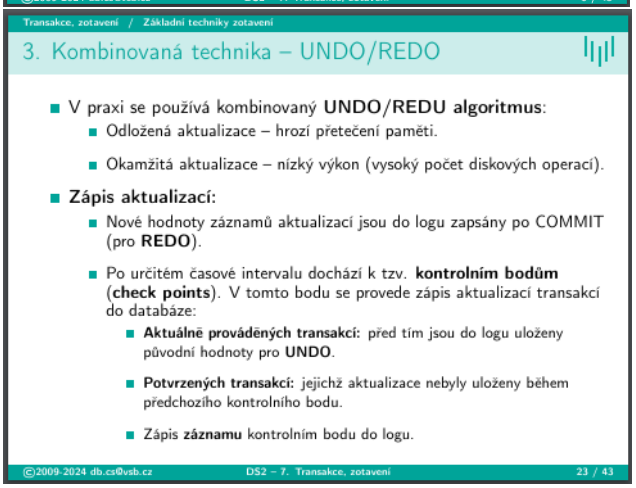
  - Použití funkce SUBSTR na indexovaném atributu (*login*) způsobí, že index nebude použit, protože indexované hodnoty jsou modifikovány funkcí, což znemožňuje efektivní vyhledávání pomocí indexu.

- 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.

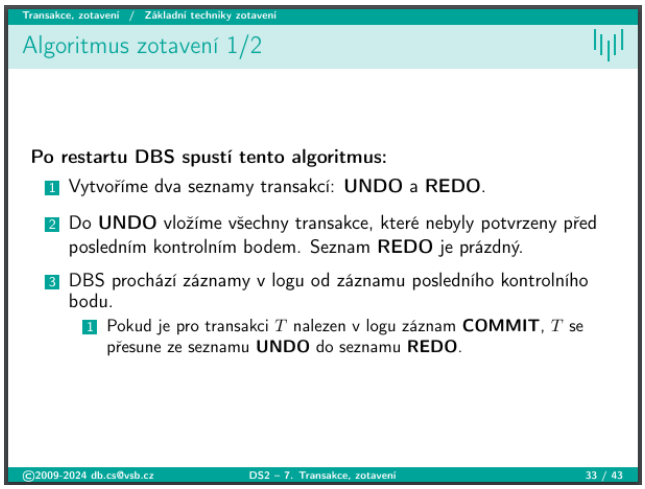
Příklad [5] (9 bodů)
Uvažujte techniku **zotavení UNDO/REDO**. Popište, co se ukládá do logu a databáze v okamžiku kontrolního bodu a co se ukládá do logu a databáze v okamžiku COMMIT konkrétní transakce. Jak jsou zapsané informace využity pro operace UNDO resp. REDO vykonávané při zotavení databáze?

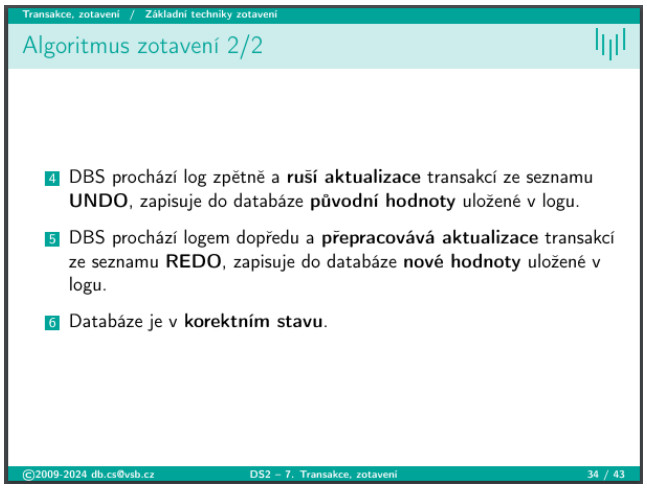
- **Pravidlo dopředného zápisu do logu (write-ahead log rule):**
  - Všechny aktualizace musí být zapsány **do logu před zápisem do databáze**. Před ukončením operace COMMIT, je do logu zapsán tzv. **COMMIT záznam**.

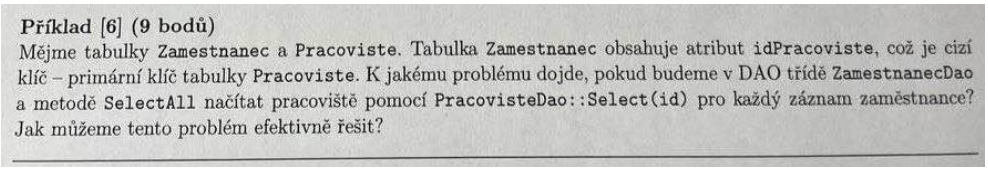




  - už tady je výše, příp. v prezentaci 7 (slide 23 – 32)

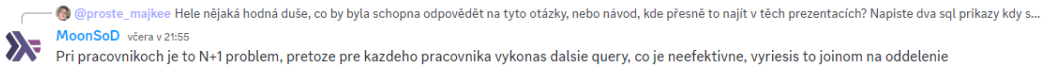






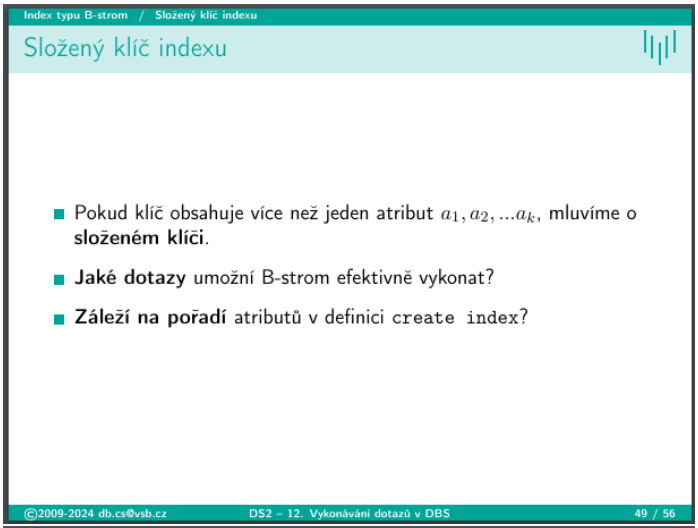
Příklad [6] (9 bodů)
Mějme tabulky `Zamestnanec` a `Pracoviste`. Tabulka `Zamestnanec` obsahuje atribut `idPracoviste`, což je cizí klíč – primární klíč tabulky `Pracoviste`. K jakému problému dojde, pokud budeme v DAO třídě `ZamestnanecDao` a metodě `SelectAll` načítat pracoviště pomocí `PracovisteDao::Select(id)` pro každý záznam zaměstnance? Jak můžeme tento problém efektivně řešit?

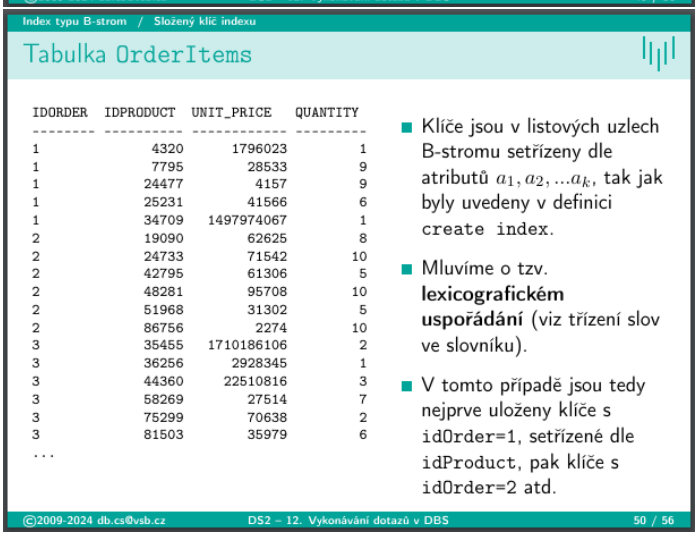
- večer vysvětlí na Discordu MoonSoD



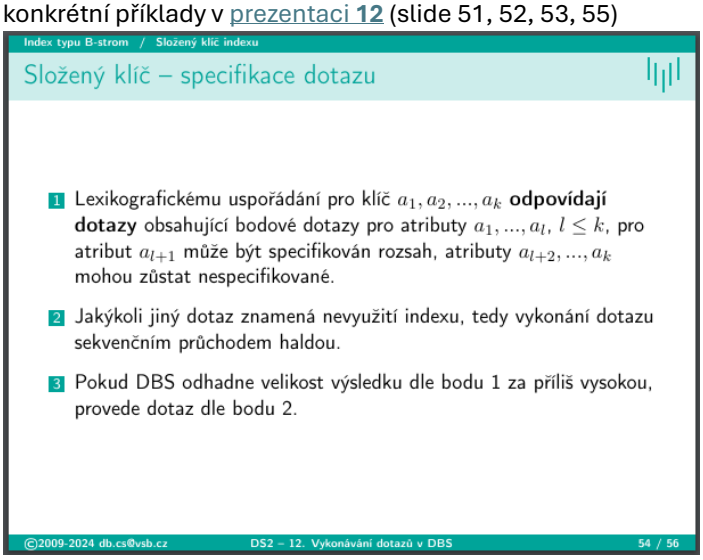
1. **kontrolní bod? UNDO/REDO, kam se to ukládá, jaká transakce**
   - už by tu mělo být

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





- konkrétní příklady v prezentaci **12** (slide 51, 52, 53, 55)

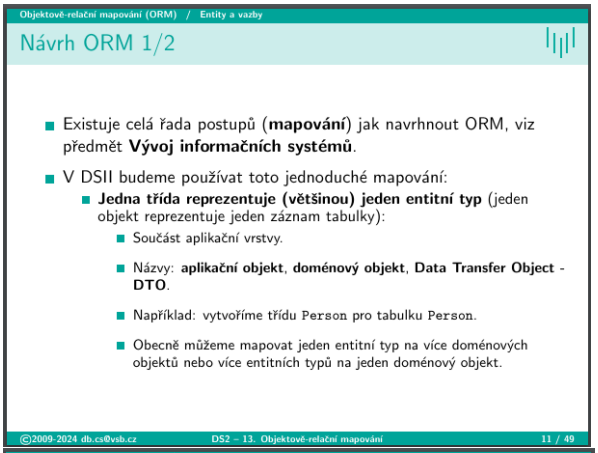


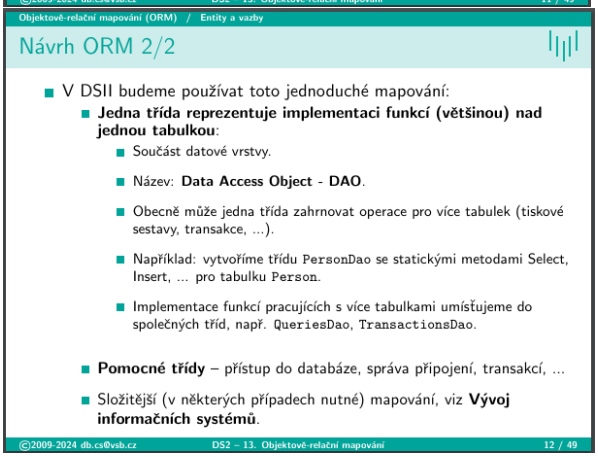
3. **napsat konkrétní kurzor chybně použitý s jednovětým SQL, konkrétní kurzor a ekvivalentní SQL dotaz**
    - to už by tu taky někde mělo být

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 transakci/bez transakce**
    - to už by tady taky mělo být, rozdíl použití a nepoužití transakce

6. co je to DAO, DTO v ORM, příklad?





- spousta příkladů k tématu je v prezentaci **13**