Databázové systémy 2

Michal Krátký, Radim Bača

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – Technická univerzita Ostrava

2024/2025

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Obsah přednášky

Sériový a serializovatelný plán

- Pokud jsou transakce provedeny za sebou mluvíme o **sériovém plánu**.
- **Sériový plán** často zapisujeme jako entici uspořádanou dle pořadí vykonávání jednotlivých transakcí; např. když A je vykonána před B, pak zapisujeme (A, B).
- Jak poznáme, zda souběžně prováděné transakce byly provedeny korektně?
- Musíme zavést nějakou 'míru korektnosti', používáme: **serializovatelnost** (angl. **serializability**).

Sériový a serializovatelný plán

Ekvivalentní plány

Dva plány pro stejné transakce jsou ekvivalentní pokud dávají stejné výsledky.

Serializovatelný plán

Plán vykonávání dvou transakcí je korektní tehdy a jen tehdy pokud je serializovatelný: plán je ekvivalentní s libovolným sériovým plánem.

Sériový plán, příklad (A, B)

$acc_1 = 30$	$acc_2 = 20$	$acc_1 = 50$
Transakce A	**Čas**	**Transakce B**
`READ` acc_1	t_1	-
$suma = 30$		
`READ` acc_2	t_2	-
$suma = 50$		
`READ` acc_3	t_3	-
$suma = 100$		
-	t_4	`READ` acc_3
-	t_5	`WRITE` $acc_3 = 60$
-	t_6	`READ` acc_1
-	t_7	`WRITE` $acc_1 = 20$
-	t_8	`COMMIT`
		$acc_1 = 20$
		$acc_2 = 20$
		$acc_3 = 60$

Sériový plán, příklad (B, A)

$acc_1 = 30$	$acc_2 = 20$	$acc_3 = 50$
Transakce A	**Čas**	**Transakce B**
-	t_1	`READ` acc_3
-	t_2	`WRITE` $acc_3 = 60$
-	t_3	`READ` acc_1
-	t_4	`WRITE` $acc_1 = 20$
-	t_5	`COMMIT`
		$acc_1 = 20$
		$acc_2 = 20$
		$acc_3 = 60$
`READ` acc_1	t_6	-
$suma = 20$		
`READ` acc_2	t_7	-
$suma = 40$		
`READ` acc_3	t_8	-
$suma = 100$		

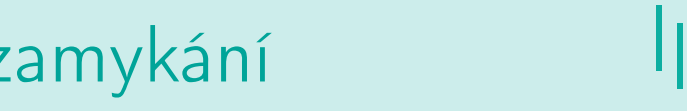
Poznámka: (A, B) a (B, A) mohou zjevně dávat různé výsledky.

Problém nekonzistentní analýzy

Plán prezentovaný při popisu problému nekonzistentní analýzy není korektní - není serializovatelný (tedy není ekvivalentní s (A, B) ani (B, A)).

$acc_1 = 30$	$acc_2 = 20$	$acc_3 = 50$
Transakce A	**Čas**	**Transakce B**
`READ` acc_1	t_1	-
$suma = 30$		
`READ` acc_2	t_2	-
$suma = 50$		
-	t_3	`READ` acc_3
-	t_4	`WRITE` $acc_3 = 60$
-	t_5	`READ` acc_1
-	t_6	`WRITE` $acc_1 = 20$
-	t_7	`COMMIT`
`READ` acc_3	t_8	-
$suma = 110$ ne 100		

Serializovatelný plán a dvoufázové uzamykání



- Dvoufázové uzamykání zaručuje, že plán bude vždy serializovatelný.
- Např. uzamykací protokol řešící problém nepotvrzené závislosti je ekvivalentní s plánem (B, A).

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` (uvolněný zámku X na t)
opakuj: `READ` t (získán zámek S na t)	t_4	

Serializovatelný plán a dvoufázové uzamykání

Uzamykací protokol řešící problém ztráty aktualizace skončí uváznutím, které bude detekováno a pokud bude zrušena transakce A, pak je plán ekvivalentní s plánem (B, A).

Transakce A	**Čas**	**Transakce B**
`READ` t	t_1	-
(získán zámek S na t)		
-	t_2	`READ` t
		(získán zámek S na t)
`WRITE` t	t_3	-
(požadavek na zámek X na t)		
wait	t_4	`WRITE` t
		(požadavek na zámek X na t)
wait		wait
wait		wait

Věta o dvoufázovém uzamykání

V článku:

Kapali P. Eswaran, Jim Gray, Raymond A. Lorie, and Irving L. Traiger: *The Notions of Consistency and Predicate Locks in a Database System*. In Commun. ACM 19(11), 1976, pages 624-633.

autoři dokázali následující větu:

Věta o dvoufázovém uzamykání

Pokud transakce dodržující přísné dvoufázové uzamykání, pak všechny možné souběžné plány jsou serializovatelné.

Věta o dvoufázovém uzamykání

Nyní si uveďme zjednodušenou variantu dříve uvedeného dvoufázového uzamykacího protokolu:

1. Transakce musí požádat o zámek na objekt (např. záznam) před tím než chce nad tímto objektem provést nějakou operaci.
2. Po uvolnění libovolného zámku nesmí transakce již požadovat další zámky.

Transakce dodržující tento protokol tak pracují ve dvou fázích: v první fázi požadují zámky, ve druhé fázi zámky uvolňují. Prakticky je druhá fáze realizována jedinou operací COMMIT nebo ROLLBACK, při které jsou uvolněny všechny držené zámky.

Transaction Processing Performance[1]

- Z předchozích kapitol je patrné, že implementace řízení transakcí (především uzamykacího protokolu) výrazně ovlivňuje výkon DBS.
- Z tohoto důvodu se stále vyvíjejí uzamykací protokoly, které umožňují dosáhnout co nejvyšší propustnosti (myšleno počtem transakcí za vteřinu) při zachování korektnosti.
- Na stránkách organizace Transaction Processing Performance Council najdeme žebříčky aktuálně nejvýkonnějších systémů pro jednotlivé testy.
- Výrobci DBS pak tyto výsledky používají při vyzdvihování výhod svých systémů.

[1] `http://www.tpc.org/`

Úroveň izolace transakcí

- Serializovatelnost garantuje izolaci transakcí ve smyslu podmínky **ACID**.
- Pokud je plán transakcí serializovatelný, pak se neprojevují negativní vlivy souběhu.
- Za izolovanost transakcí musíme zaplatit – v tomto případě **nižším výkonem** souběhu (tedy nižším počtem vykonaných transakcí – propustností).
- SŘBD proto umožňují nastavit **úroveň izolace**, která zvýší propustnost, ale sníží míru izolace transakce.

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)

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.

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

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`	t_3	-
`WHERE id='jan001'`		
`COMMIT`	t_4	
	t_5	`COMMIT/ROLLBACK`

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

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	

Výskyt fantomů 1/3

Mějme následující tabulku `Student`:

`login`	`jmeno`	`rocnik`
jan001	Jan	1
mil002	Milan	3

Pokud je povolena úroveň RR (a nižší) a v transakci se vyskytují dva stejné dotazy, pak může nastat výjimka souběhu zvaná **výskyt fantomů**. Mějme plány těchto dvou transakcí.

Výskyt fantomů 2/3

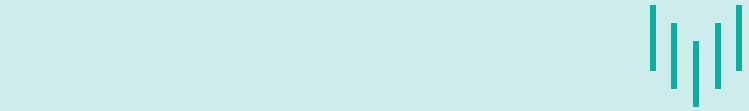
Mějme plány těchto dvou transakcí:

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	-

Výskyt fantomů 3/3



- V tomto případě systém vrátí pro stejný dotaz v transakci A v čase t_1 resp. t_4 různé výsledky: SŘBD neprovádí zamykání rozsahu záznamů z tabulky `Student` (tedy záznamů s hodnotou atributu `rocnik` ≤ 1 a ≥ 2).
- V případě úrovně SR oba dotazy vrátí stejný počet záznamů.

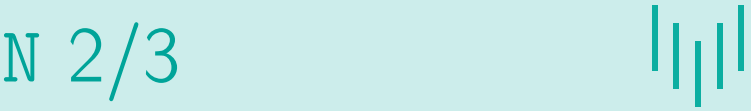
Syntaxe příkazu START TRANSACTION 1/3

```
START TRANSACTION <volitelné parametry>;
```

Kde <*volitelné parametry*> specifikují:

- *režim přístupu* (angl. *access mode*),
- *úroveň izolace* (angl. *isolation level*),
- *velikost diagnostikované oblasti* (angl. *diagnostics area size*).

Syntaxe příkazu START TRANSACTION 2/3



- Úroveň izolace zadáváme ve tvaru `ISOLATION LEVEL` <*izolace*>, kde <*izolace*> je `READ UNCOMMITED`, `READ COMMITED`, `REPEATABLE READ` a `SERIALIZABLE`.
- Režim přístupu může být `READ ONLY` nebo `READ WRITE`. Pokud nespecifikujeme žádný režim, je nastaven `READ WRITE`. Pokud ovšem zvolíme úroveň izolace `READ UNCOMMITED`, pak je nastaven režim `READ ONLY`. Pokud tedy zvolíme režim `READ WRITE`, pak úroveň izolace nesmí být `READ UNCOMMITED`.

Syntaxe příkazu `START TRANSACTION` 2/3

- Velikost diagnostikované oblasti se nastavuje jako celé číslo uvedené za klíčovým slovem `DIAGNOSTICS SIZE` a specifikuje kolik výjimek bude systém ukládat na zásobníku.

Příklad

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT last_name, salary FROM employees
  WHERE last_name IN ('Pondělí','Středa','Pátek');

UPDATE employees SET salary = 9900
  WHERE last_name = 'Pondělí';

COMMIT;
```

Správa verzí (versioning)

- **Zamykání** se nazývá pesimistický přístup k souběžnému zpracování: předpokládáme, že paralelní transakce se budou pravděpodobně navzájem ovlivňovat.
- **Správa verzí** je další přístup pro řízení souběžného zpracování – optimistický přístup: předpokládáme, že paralelní transakce se ovlivňovat nebudou.
- Při uzamykání spravuje systém jednu kopii dat a jednotlivým transakcím přiděluje zámky.
- Při správě verzí systém vytváří při aktualizaci kopie dat a systém sleduje, která z verzí má být viditelná pro ostatní transakce (v závislosti na úrovni izolace).

Správa verzí, příklad

Transakce čtou/aktualizují záznam z tabulky `Produkty(id, jmeno_produktu, pocet)`.

t_1	Transakce A čte záznam (48501, 'Osa - ocel', 45). V databázi existuje jedna kopie - označme jako c_1.
t_2	Transakce B aktualizuje záznam, systém vytvoří kopii (48501, 'Osa - ocel', 44), označme jako c_2. Pokud B bude chtít získat tento záznam, získá kopii c_2.
t_3	Transakce C chce získat tento záznam. Transakce B neprovedla COMMIT, proto C získá kopii c_1.
t_4	Transakce B provede COMMIT. Systém zpravuje pouze jednu kopii záznamu.
t_5	Transakce E chce získat tento záznam, získá aktualizovaný záznam.

Zamykání, příklad

Transakce čtou/aktualizují záznam z tabulky `Produkty(id, jmeno_produktu, pocet)`.

t_1	Transakce A čte záznam (48501, 'Osa - ocel', 45), systém přidělí transakci S zámek.
t_2	Transakce B se pokusí aktualizovat záznam, pokud bude úroveň izolace SERIALIZABLE nebo REPEATABLE READ je transakce pozastavena. Pokud A pracuje na nižší úrovni izolace, systém může umožnit transakci B pokračovat, přidělí X zámek a B aktualizuje záznam: (48501, 'Osa - ocel', 44).
t_3	Transakce C chce získat tento záznam. Pokud B drží X zámek, C je pozastavena.
t_4	...

Příklad, porovnání

Pozorování 1:

- Správa verzí umožňuje paralelní zpracování více souběžných transakcí.
- Zamykání bude ve většině případů způsobovat, že dvě nebo více transakcí budou čekat na ukončení jiné transakce (a uvolnění zámků).

Pozorování 2:

- Plány jsou odlišné - což je v pořádku. Plán musí splňovat 'pouze' podmínku serializovatelnosti.

Správa verzí, výhody, nevýhody

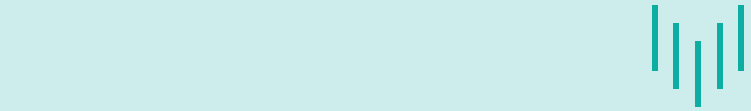
- **Nevýhodou správy verzí** je vyšší režie - zvýšený požadavek na paměť při zprávě kopií dat.
- Pokud budou **převažovat READ** operace, je správa verzí efektivnější než zamykání.
- Pokud budou převažovat aktualizace stejných záznamů, bude režie správy verzí převyšovat popsané výhody.
- SŘBD proto často používají kombinaci obou přístupů.

Implementace řízení souběhu v Oracle

- **Oracle používá uzamykání a správu verzí**. Správa verzí neumožňuje vznik špinavého čtení (READ UNCOMMITTED tedy není podporován).

- Oracle podporuje dvě úrovně izolace SQL a jednu vlastní:
  - READ COMMITTED
  - READ ONLY
  - SERIALIZABLE

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**.

Úroveň izolace SERIALIZABLE 1/2

- Každý dotaz spuštěný transakcí vidí změny **potvrzená před začátkem transakce** (tedy ne dotazu jako v případě READ COMMITTED) a změny provedené samotnou transakcí.
- Tato úroveň izolace je vhodná:
  - pro velké databáze a krátké transakce, které aktualizují málo záznamů,
  - pokud pravděpodobnost, že dvě paralelní transakce budou aktualizovat stejný záznam je relativně nízká,
  - pokud dlouhé transakce používají především čtení.

Poznámka: Všimněte si, že největším problém jsou transakce, které provádí ve větší míře aktualizace stejných záznamů.

Úroveň izolace SERIALIZABLE 2/2

- Oracle vykoná serializovatelnou transakci **aktualizující záznam** pouze pokud aktualizace záznamu provedená jinou transakcí byla **potvrzena dříve než serializovatelná transakce začala**.
- V opačném případě SŘBD generuje chybu[2], tedy pokud se transakce pokouší aktualizovat data změněná jinou transakcí, která aktualizace potvrdila po začátku původní transakce.

Poznámka: **Oproti specifikaci SQL tedy transakce nepřejdou do stavu čekání.**

[2] ORA-08177: Cannot serialize access for this transaction

Úroveň izolace READ-ONLY

- Tato úroveň izolace je podobná úrovni SERIALIZABLE až na to, transakce **nesmí aktualizovat data**[3].
- Tato úroveň izolace je vhodná pro generování sestav u kterých požadujeme konzistentní obsah (obsah databáze během vykonávání transakce musí být stejný, jako před začátkem transakce).

[3] Uživatel SYS může aktualizovat data

Příklad, Serializable[4]

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

Poznámka: V čase t3 vidíme úspěšnou operaci delete T2, při zamykání by operace úspěšná nebyla - T2 by nezískala zámek X.

[4] `https://asktom.oracle.com/pls/apex/asktom.search?tag=transaction-isolation-level`

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`

Implementace řízení souběhu v SQL Server

- Úrovně izolace
  - READ UNCOMMITTED
  - READ COMMITTED
  - REPEATABLE READ
  - SERIALIZABLE
  - SNAPSHOT - stejné vlastnosti jako SERIALIZABLE vzhledem ke specifikaci SQL, ale ... (viz následující slide).

SNAPSHOT vs SERIALIZABLE

- **SERIALIZABLE**, popsané problémy souběhu se nevyskytují, ale transakce se mohou blokovat (jelikož jsou **použity zámky**):
  - Příkazy transakce nevidí nepotvrzené aktualizace ostatních transakcí.
  - Ostatní transakce nemohou měnit data přečtená aktuální transakcí dokud ta neskončí.
  - Ostatní transakce také nemohou vložit záznamy s hodnotami, které se objevily v rozsahu při čtení aktuální transakce dokud ta neskončí.

SNAPSHOT vs SERIALIZABLE

- **SNAPSHOT**:
  - Stejné jako SERIALIZABLE z pohledu specifikace SQL: transakce vidí data potvrzená před začátkem transakce.
  - Nejsou použity zámky, ale **verzování**: transakce čtoucí data neblokuje ostatní zapisující transakce, stejně tak transakce zapisující data neblokují čtoucí transakce.
  - Abychom mohli použít SNAPSHOT, musí být nastaven `ALLOW_SNAPSHOT_ISOLATION` na `ON`:

    ```
    ALTER DATABASE <DB Name>
      SET ALLOW_SNAPSHOT_ISOLATION ON
    ```

    `READ_COMMITTED_SNAPSHOT` pak dále nastavuje implicitní úroveň na READ COMMITTED (využívá se verzování).

Explicitní zamykání

- Je jasné, že problémem není ani tak zajištění podmínky ACID, ale spíše zajištění této podmínky při rozumné propustnosti.
- SŘBD proto podporují tzv. **explicitní zamykání** – záznamy nejsou zamykány automaticky, ale programátor o jejich přidělení explicitně žádá.
- Nejčastěji SŘBD podporující konstrukce `LOCK TABLE` a `SELECT FOR UPDATE`.

Zámky tabulek

- ROW SHARE (RS) – povoluje dalším transakcím dotazování, zamezuje získat na celou tabulku X zámek.
- ROW EXCLUSIVE (RX), zamezuje ostatním transakcím získat S zámek. RX je automaticky přidělován při aktualizaci záznamu.
- SHARE (S): povoluje dalším transakcím dotazování, transakce neaktualizuje tabulku.
- SHARE ROW EXCLUSIVE (SRX) – povoluje dalším transakcím dotazování, nepovoluje získání S zámku a aktualizaci tabulky danou transakcí.
- EXCLUSIVE (X): nepovoluje žádné operace paralelních transakcí.

Zámky tabulek

Dotaz	Tabulkový zámek
SELECT...FROM table ...	-
INSERT INTO table ...	RX
UPDATE table ...	RX
DELETE FROM table ...	RX
SELECT ... FROM table FOR UPDATE OF ...	RS
LOCK TABLE table IN ROW EXCLUSIVE MODE	RX
LOCK TABLE table IN SHARE MODE	S
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE	SRX
LOCK TABLE table IN EXCLUSIVE MODE	X

LOCK TABLE

- Syntaxe pro Oracle:

```
LOCK TABLE <názvy tabulek nebo pohledů oddělené čárkami>
IN <lock_type> MODE [NOWAIT];
```

  kde `lock_type` může být: `ROW SHARE`, `ROW EXCLUSIVE`, `SHARE UPDATE`[6], `SHARE`, `SHARE ROW EXCLUSIVE`, `EXCLUSIVE`.

- Pokud použijeme klíčové slovo `NOWAIT`, systém nečeká s přidělením zámku pokud je tabulka zamčena jiným uživatelem a vrátí řízení volajícímu programu.
- Podobně jako v případě implicitního zamykání, systém uvolní všechny zámky na konci transakce.

[6] Stejné jako `ROW SHARE`

LOCK TABLE, příklad

Příklad:

```
LOCK TABLE Student IN SHARE MODE;
```

Tímto příkazem požadujeme sdílený zámek na tabulku `Student`. Systém bude čekat na přidělení zámku, pokud jiná transakce získala dříve na tuto tabulku zámek.

SELECT FOR UPDATE 1/2

- Příkaz SELECT s klauzulí FOR UPDATE vybírá záznamy a zároveň je zamyká výlučným zámkem. Přesněji: transakce požaduje X zámek na záznamy a ROW SHARE tabulkový zámek.
- V tomto případě tedy počítáme s pozdější aktualizací záznamů – snažíme se vyhnout problémům, které mohou nastat při požadavku na výlučný zámek při dříve uděleném sdíleném zámku.

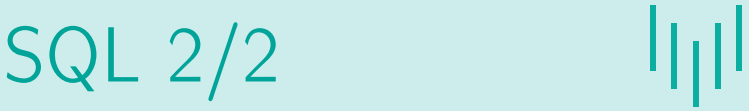
SELECT FOR UPDATE 2/2

- Implicitně systém čeká dokud požadovaný zámek není přidělen, toto chování můžeme změnit klauzulemi: NOWAIT, WAIT, SKIP LOCKED.
- Můžeme použít i v případě kdy nechceme aby vybrané záznamy byly aktualizovány jinou transakcí.
- Není ve standartu, najdeme např. v Oracle, Postgresql.

SELECT FOR UPDATE, příklad PL/SQL 1/2

- Pokud je `SELECT FOR UPDATE` asociován s explicitním kurzorem, pak tento kurzor nazýváme FOR UPDATE kurzor.
- Pouze FOR UPDATE kurzor se může objevit v klauzuli `CURRENT OF` příkazů UPDATE a DELETE.
- Klauzule CURRENT OF omezí platnost příkazů UPDATE a DELETE pouze na aktuální záznam kurzoru.

SELECT FOR UPDATE, příklad PL/SQL 2/2



```
DECLARE
  my_emp_id NUMBER(6);
  my_job_id VARCHAR2(10);
  my_sal    NUMBER(8,2);
  CURSOR c1 IS
    SELECT employee_id, job_id, salary
    FROM employees FOR UPDATE;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO my_emp_id, my_job_id, my_sal;
    IF my_job_id = 'SA_REP' THEN
      UPDATE employees SET salary = salary * 1.02
        WHERE CURRENT OF c1;
    END IF;
    EXIT WHEN c1%NOTFOUND;
  END LOOP; END;
/
```

Reference

- SQL Server:
  - `https://docs.microsoft.com/en-us/sql/t-sql/statements/set-transaction-isolation-level-transact-sql`
  - `https://www.sqlservercentral.com/articles/isolation-levels-in-sql-server`
- Oracle:
  - `https://docs.oracle.com/en/database/oracle/oracle-database/21/cncpt/data-concurrency-and-consistency.html`
  - `https://blogs.oracle.com/oraclemagazine/on-transaction-isolation-levels`