Optimistic concurrency control in Microsoft SQL Server

RNDr. David Gešvindr, Ph.D.
MVP: Data Platform | MCSE: Data Platform | MCT
david@wug.cz
@gesvindr

Osnova

1. Pessimistic vs. Optimistic concurrency control
2. Základní princip fungování zámků
3. Vlastní implementace optimistic concurrency control v aplikaci
4. Row-versioning na SQL Serveru
5. Multi-version optimistic concurrency v InMemory OLTP

Osnova

1. **Pessimistic vs. optimistic concurrency control**
2. Základní princip fungování zámků
3. Vlastní implementace optimistic concurrency control v aplikaci
4. Row-versioning na SQL Serveru
5. Multi-version optimistic concurrency v InMemory OLTP

Základní vlastnosti databázové transakce

- Základní vlastnosti transakce:

Atomicity (atomičnost)

Consistency (konzistence)

Isolation (izolace)

Durability (trvalost)

Další informace: https://www.wug.cz/zaznamy/331-SQL-Server-Bootcamp-2016-Transakce-zamky-a-izolacni-urovne-v-SQL-Serveru

Pessimistic vs. optimistic concurrency

- Při souběžném přístupu může chtít více spojení **aktualizovat stejná data**
  - Aktualizace dat zahrnuje jejich přečtení a následnou změnu

- **Pessimistic concurrency**
  - Data jsou vhodně uzamykána, aby nemohla být změněna

- **Optimistic concurrency**
  - Data nejsou uzamykána, ale před každou aktualizací se kontroluje, že od načtení data nikdo nezměnil

Osnova

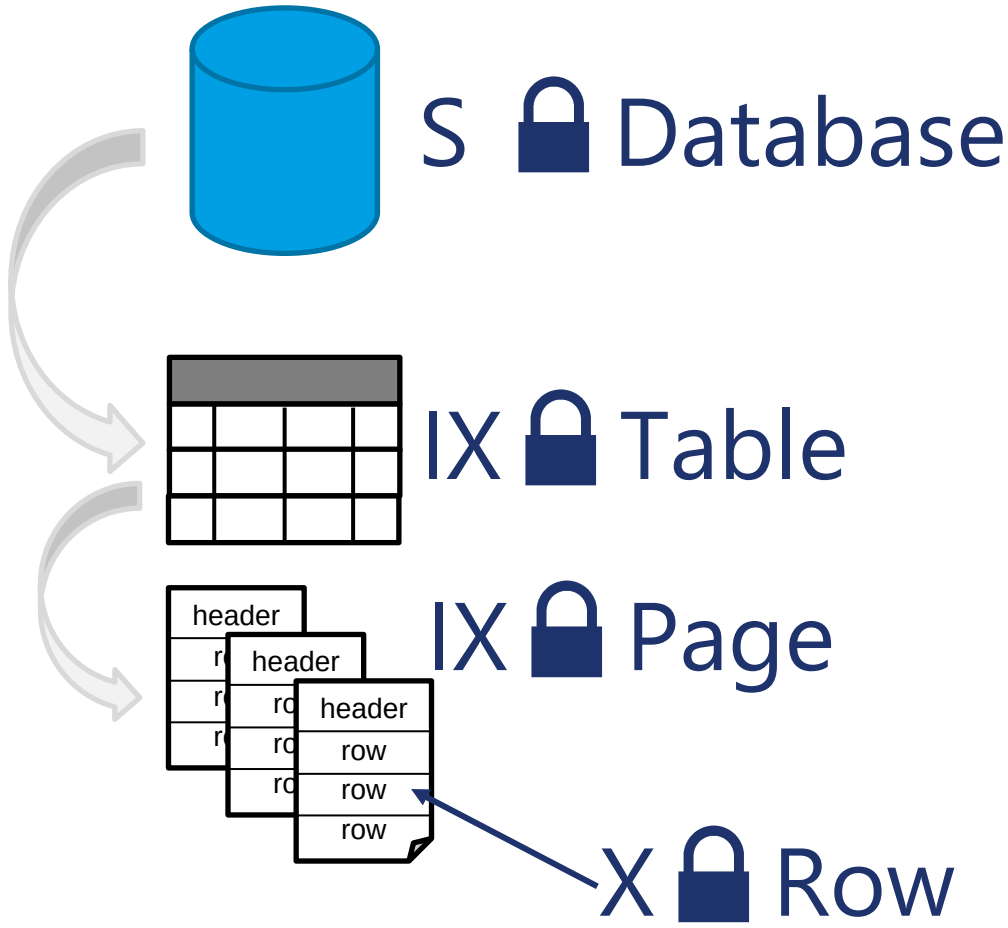
1. Pessimistic vs. optimistic concurrency control
2. **Základní princip fungování zámků a izolačních úrovní**
3. Vlastní implementace optimistic concurrency control v aplikaci
4. Row-versioning na SQL Serveru
5. Multi-version optimistic concurrency v InMemory OLTP

Jak fungují zámky

1. Připojíme se k databázi
   - Vznikne sdílený zámek nad databází
2. Spustíme dotaz:

```
UPDATE table1
SET Name = ‘David’
WHERE Id = 10
```

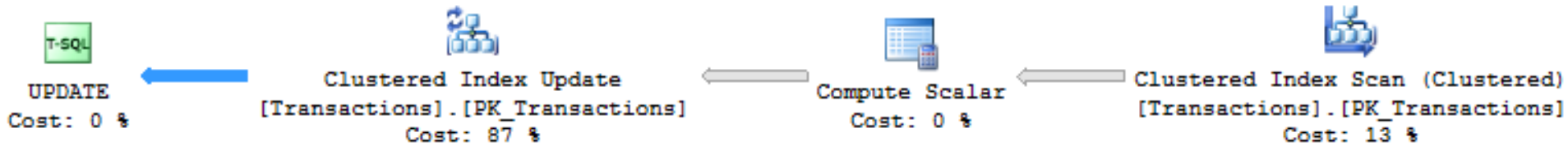
   - Je třeba exkluzivně uzamknout řádek
   - Zamykat je však třeba shora
   - Intent eXclusive zámek



Typy zámků v SQL Serveru

- **Shared Lock**
  - Je možné zdroj číst, ale neměnit
- **Update Lock**
  - Signalizace, že budeme měnit, ale zatím neměníme
  - Kvůli snížení blokování čtení

Požadovaný zámek \ Aktivní zámek	Shared	Update	Exclusive
Shared	GRANT	GRANT	WAIT
Update	GRANT	WAIT	WAIT
Exclusive	WAIT	WAIT	WAIT



- **Exclusive Lock**
  - Je možné zdroj modifikovat

Level 0 – Read Uncommitted

Problém: Dirty Read

- Při čtení dat nejsou vytvářeny sdílené zámky
  - Transakce může číst i nepotvrzená data
- **Modifikace dat vytváří exkluzivní zámky vždy nehledě na izolační úroveň**

- Vhodné použití:
  - Čtení dat, kde nezáleží na přesném stavu
  - Není blokováno čtení a není blokován ani další zápis
    - Jsou pouze blokovány modifikace schématu (SCH_S)
  - **Používat velmi opatrně**

Level 1 – Read Committed

Double Read + Non-repeatable Reads + Phantoms

- Pro čtení dat jsou vyžadovány sdílené zámky
  - Není možné číst nepotvrzená data (exkluzivní zámky uvolněny na konci transakce)
- Neposkytuje absolutní přesnost při čtení dat
  - Při rozsáhlém čtení je i v rámci jedné tabulky možné narazit na nekonzistence
- Sdílené zámky jsou uvolňovány hned po přečtení dat

Level 2 – Repeatable Read

Phantoms

- Sdílené zámky jsou uvolněny až na konci transakce
- Ostatní transakce mohou data číst, ale nemohou je měnit
  - Jakákoliv přečtená data jsou po celou dobu života transakce stejná

- Nevýhody:
  - Jakákoliv data, která přečteme, uzamkneme až do konce transakce
  - Rizika rozsáhlého blokování
  - Problém s fantomy

Level 3 – Serializable

- Využívá navíc **range-locks**
  - Uzamknou určitý rozsah hodnot
  - Nemůže být přidán žádný nový záznam

- Efektivní provedení **vyžaduje vhodný index**
  - Uzamčení vhodných uzlů ve stromu
  - **Pokud není vhodný index – vznikne zámek nad celou tabulkou**

Nastavení izolační úrovně

- Pro celé spojení pomocí příkazu:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

  - Izolační úroveň se od daného okamžiku aplikuje se na všechny operace prováděné **v daném spojení**

- Pro individuální operace s pomocí HINTů:

```
SELECT * FROM Users WITH(NOLOCK)
```

  - Velmi dobře si rozmyslete jaký bude dopad, když budete míchat jednotlivé izolační úrovně uvnitř transakce
  - Seznam HINTů: https://docs.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-table

Osnova

1. Pessimistic vs. optimistic concurrency control
2. Základní princip fungování zámků
3. **Vlastní implementace optimistic concurrency control v aplikaci**
4. Row-versioning na SQL Serveru
5. Multi-version optimistic concurrency v InMemory OLTP

Vlastní implementace optimistic concurrency control

- Řada frameworků poskytuje podporu pro ochranu dat s pomocí optimistic concurrency
- Aktualizace dat je podmíněna testem, jestli se data od načtení do aplikace nezměnila:
  1. Formulář je vyplněn daty načtenými z databáze
  2. Uživatel neurčitou dobu provádí změny (zámky by měly velmi negativní dopad na ostatní uživatele)
  3. Uživatel ukládá celý formulář, aplikace testuje, jestli nepřepisujeme cizí změny v datech

Detekce provedených změn

- Aplikace může podmínit příkaz UPDATE shodou v původních hodnotách všech sloupců
  - Vyžaduje, aby si aplikace u všech atributů entity sledovala původní i nové hodnoty a uvede je ve WHERE podmínce
  - Pokud operace vrátí 0 změněných řádků, je třeba řešit konflikt v datech

- U řádku je možné si ukládat čas jeho poslední změny, případně jeho verzi
  - Porovnává se pouze čas poslední změny
  - Hodnotu aktualizuje aplikace nebo trigger
  - Budoucí hodnota v porovnání s původně načtenou hodnotou značí, že řádek byl od načtení změněn a nastal konflikt

Zdroje: https://docs.microsoft.com/en-us/ef/core/saving/concurrency
https://docs.microsoft.com/en-us/ef/core/modeling/concurrency?tabs=data-annotations

Timestamp / Rowversion

- Datový typ **rowversion** obsahuje verzi záznamu
  - Datový typ **timestamp** je synonymum, nejde proto přetypovat na čas
  - Interně se jedná i binary(8)/varbinary(8)
  - Aktuální hodnotu v databázi lze zjistit přes funkci @@DBTS
- Takto vytvořený sloupec v tabulce je pouze pro čtení a inkrementuje svoji hodnotu spolu s každou změnou řádku
- Aplikace může aktualizaci dat podmínit hodnotou rowversion sloupce v čase načtení řádku
  - Aktualizace žádného řádku značí konflikt

Zdroj: https://docs.microsoft.com/en-us/sql/t-sql/data-types/rowversion-transact-sql?view=sql-server-ver15

Osnova

1. Pessimistic vs. optimistic concurrency control
2. Základní princip fungování zámků
3. Vlastní implementace optimistic concurrency control v aplikaci
4. **Row-versioning na SQL Serveru**
5. Multi-version optimistic concurrency v InMemory OLTP

Row-versioning na SQL Serveru

- Od **Microsoft SQL Serveru 2005** je možné aktivovat izolační úrovně, které nativně verzují řádky při jejich změnách
  - Původní kopie řádku je přístupná ke čtení bez zámků, zatím co v datových stránkách je nepotvrzená hodnota probíhající transakce

- Jsou dostupné 2 nové izolační úrovně:
  - Read Committed Snapshot Isolation (RCSI)
  - Snapshot Isolation (SI)

Read Committed Snapshot Isolation

- Změna výchozí izolační úrovně v SQL Serveru
  - Tam kde bychom byli bývali čekali na zámcích při čtení exkluzivně uzamknutého řádku nyní přečteme bez zámku původní hodnotu
  - Nebezpečné z důvodu změny chování – tam, kde bychom čekali s rozhodnutím na novou hodnotu „na cestě“ se nyní rozhodneme hned podle staré hodnoty
- Konzistentní pohled na data je zajištěn na úrovni příkazu
  - Verze řádků se udržují jen na dobu potřebnou pro provedení příkazu
  - Pokud budeme opakovaně číst v transakci, nedosáhneme chování REPEATABLE READ či SERIALIZABLE

Snapshot Isolation

- Nová izolační úroveň v Microsoft SQL Serveru
  - Musíme ji explicitně aktivovat v rámci spojení
  - Povolení této funkcionality neovlivní stávající kód

- Poskytuje plně konzistentní pohled na data po celý čas existence transakce ve stavu jako při zahájení transakce
  - Zahájením transakce není BEGIN TRANSACTION ale první operace
  - Verze řádků se musí uchovávat po celý čas transakce navíc pro všechny tabulky v databázi

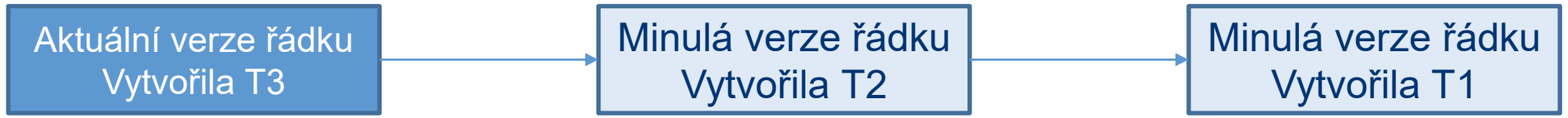
Konflikt v Snapshot Isolation

- U transakcí v Snaphot Isolation levelu může docházet při změnám ke konfliktům
- Konflikt nastává, pokud změníme data, která od našeho prvního čtení stihl někdo změnit a změnu potvrdit
  - Pokud by nenastal konflikt, tak bychom způsobili Lost Updates a přepsali cizí změnu

Version Store

- Při aktivaci SI nebo RCSI na úrovni databáze musí příkazy UPDATE a DELETE generovat verze řádků v **tempdb** databázi
  - Dochází ke snížení výkonu modifikací dat a vzroste zátěž na tempdb

- Řádky musí být v úložišti **version store** databáze tempdb uloženy tak dlouho, dokud existuje transakce, která by je mohla potřebovat
  - Verze se generují i když neběží zrovna žádná transakce, co čte data
  - Původní hodnoty musí být připraveny, kdyby čtení bylo spuštěno



Persistent Version Store

- Spolu s technologií **Accelerated Database Recovery (ADR)** v Microsoft SQL Serveru 2019 je uvedeno úložiště **persistent version store**
  - Původní verze řádků nejsou již ukládány v tempdb, ale jako součást databáze samotné (lze vybrat v rámci které File Group)
  - Primárně je tento mechanizmus navržený pro rychlý rollback a recovery proces databáze (původní hodnoty nemusí být vyčítány z transakčního logu)
  - Tyto verze záznamů využívají následně i Read Committed Snapshot a Snapshot izolační úrovně

Osnova

1. Pessimistic vs. optimistic concurrency control
2. Základní princip fungování zámků
3. Vlastní implementace optimistic concurrency control v aplikaci
4. Row-versioning na SQL Serveru
5. **Multi-version optimistic concurrency v InMemory OLTP**

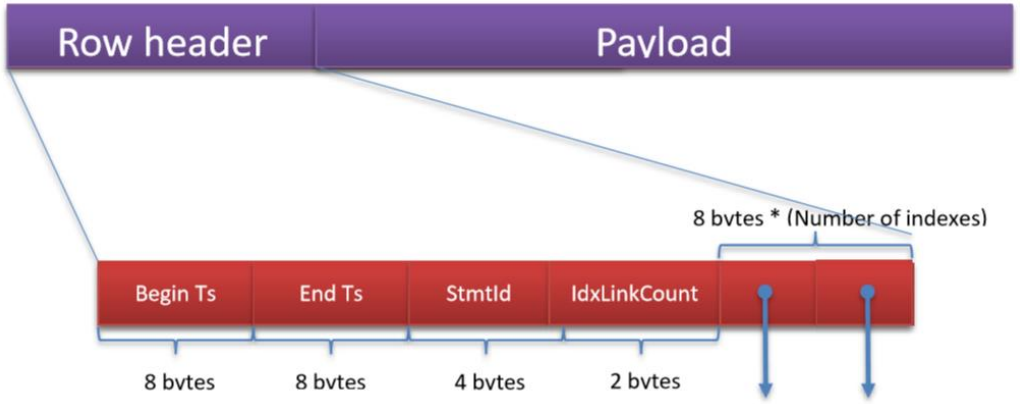
In-memory OLTP

- SQL Server 2014 přidává podporu pro **Memory Optimized Tabulky** a **nativně kompilované uložené procedury**
- Jedná se o nový engine pro uložení a zpracování dat přímo v operační paměti
  - Nevyužívá zámky, ale multi-version optimistic concurrency
  - Je zaručena persistence dat při výpadku díky využití transakčního logu
- Dochází až k **30 násobnému zvýšení počtu zpracovaných transakcí** za vteřinu

Další informace: https://www.wug.cz/zaznamy/503-SQL-Server-Bootcamp-2018-Zpracovavejte-velka-data-v-SQL-Serveru-rychlosti-blesku

Struktura řádku

- Řádek je uložený v nové datové struktuře:



- Begin-Ts – časové razítko transakce, co vložila řádek
- End-Ts – časové razítko transakce, co smazala řádek
- StmtId – jedinečné ID příkazu, co řádek založil
  - Ochrana před Halloween Problem (UPDATE neaktualizuje řádek 2x)
- IdxLinkCount – počet referencí na řádek

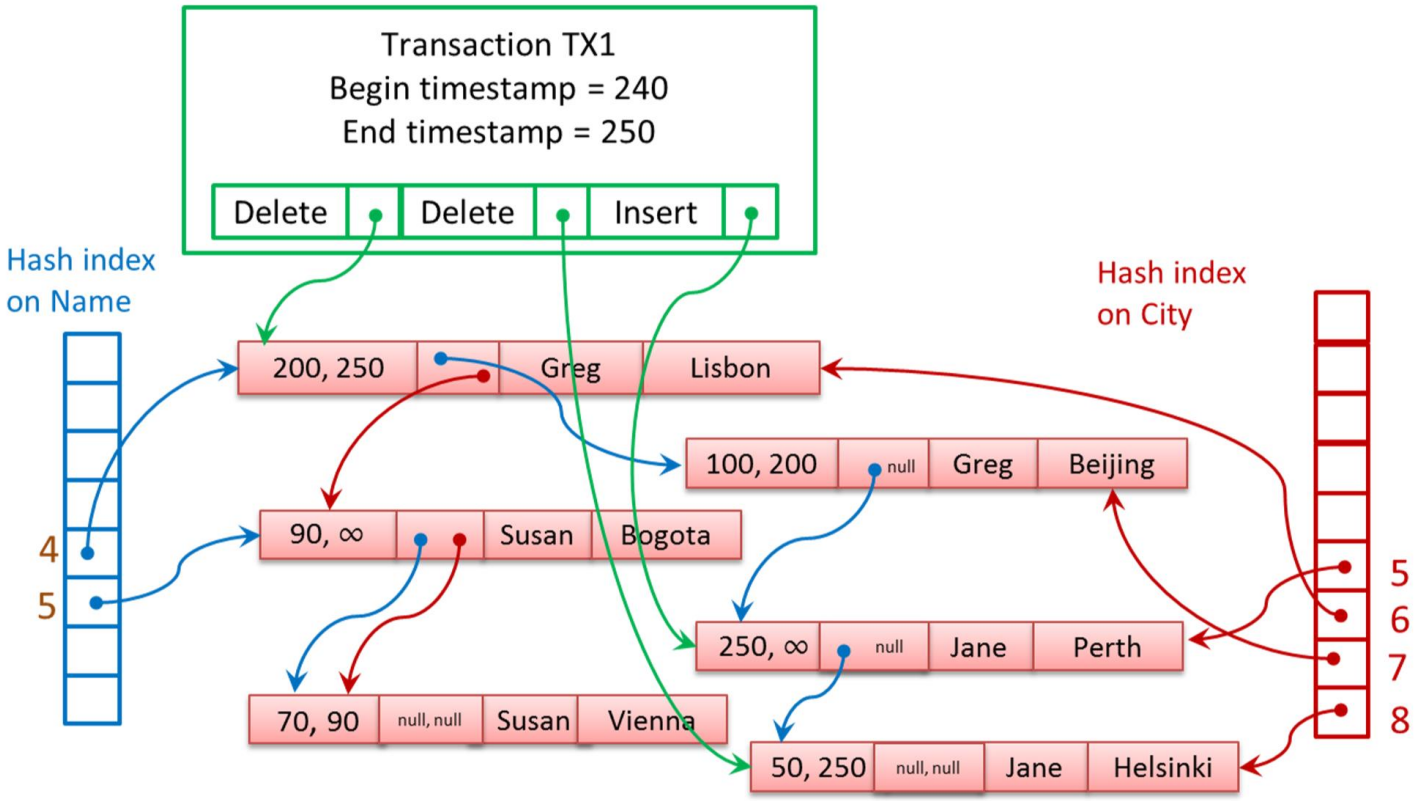
Zdroj: Kalen Delaney: SQL Server In-Memory OLTP Internals for SQL Server 2016, Technical White Paper

Podporované izolační úrovně

- SNAPSHOT
  - Transakce vidí všechna data tak, jak existovala v čase začátku transakce
  - Konzistentní pohled na všechny tabulky
  - Konflikt, pokud změníme hodnotu, kterou někdo mezi tím změnil
- REPEATABLE READ
  - Všechny garance co SNAPSHOT, navíc je garantováno, že všechna data co jsme přečetli nikdo do celou transakci nezměnil (pokud ano, chyba při validaci)
- SERIALIZABLE
  - Vše co REPEATABLE READ a navíc Phantom Avoidance

Zdroj: Kalen Delaney: SQL Server In-Memory OLTP Internals for SQL Server 2016, Technical White Paper

Ukázka souběžného zpracování



TX1: ID 100, začátek 240
Maže řádek <Greg, Lisbon> a současně aktualizuje <Jane, Helsinky→Perth>

TX2: začátek 243, autocommit SELECT
co čte celou tabulku

TX3: začátek 246

```
DECLARE @City nvarchar(32);
BEGIN TRAN TX3
   SELECT @City = City
   FROM T1 WITH (REPEATABLEREAD)
   WHERE Name = 'Jane';
   UPDATE T1 WITH (REPEATABLEREAD)
   SET City = @City
   WHERE Name = 'Susan';
COMMIT TRAN  -- commits at timestamp 255
```

Zdroj: Kalen Delaney: SQL Server In-Memory OLTP Internals for SQL Server 2016, Technical White Paper

Osnova

1. Pessimistic vs. optimistic concurrency control
2. Základní princip fungování zámků
3. Vlastní implementace optimistic concurrency control v aplikaci
4. Row-versioning na SQL Serveru
5. Multi-version optimistic concurrency v InMemory OLTP

Dotazy

Chcete si na toto téma společně povídat 4 dny? Přijďte na GOC 631

RNDr. David Gešvindr, Ph.D.

MVP: Data Platform | MCSE: Data Platform | MCT

david@wug.cz

@gesvindr