# Transakce, zámky a izolační úrovně v Microsoft SQL Serveru

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

# Motivace

- Databázové systémy jsou častým úložištěm dat pro vaše aplikace
- Jejich správné použití je klíčové pro bezchybný chod aplikace

- Celá řada vývojářů pracuje s relační databází a není si plně vědoma:
  - Vlastností transakcí
  - Negativních důsledků souběžného zpracování
  - Možností, jak se problémům při souběžném zpracování bránit

# Osnova

1. Transakce
2. Zámky
3. Izolační úrovně a problémy souběžného zpracování
4. Trochu optimizmu na závěr, aneb nešlo by to bez zámků?

# Osnova

1. **Transakce**
2. Zámky
3. Izolační úrovně a problémy souběžného zpracování
4. Trochu optimizmu na závěr, aneb nešlo by to bez zámků?

# Transakce

- Základní vlastnosti databázové transakce:

**A**tomicity (atomičnost)

**C**onsistency (konzistence)

**I**solation (izolace)

**D**urability (trvalost)

# Atomicity

- Všechny operace v rámci databázové transakce musí být provedeny **kompletně** nebo **vůbec**
- Každá operace modifikující data probíhá automaticky v transakci
  - I operace jako je vložení jednoho řádku do tabulky není jedna operace, ale zápis do více indexů, které musí vždy vracet stejný výsledek dotazu nehledě na to, přes který index tabulky data čteme
- Typy transakcí
  - Autocommit transakce
  - Explicitní transakce
    - Parametr spojení XACT_ABORT
  - Implicitní transakce

# Consistency

- Každá transakce může měnit data jen **povoleným způsobem**
- Databáze **musí být v platném** stavu před provedením transakce a i po skončení transakce
- Platný stav je určen integritními omezeními definovanými nad daty
  - Primary key, Unique key, Foreign key, Check constraint
- Interní datové struktury databáze musí být také v platném stavu
  - Datové stránky neobsahují ukazatele na neexistující stránky
  - Všechny indexy nad tabulkou jsou synchronizované a nehledě na to, přes který index čteme, tak dotaz dojde ke stejnému výsledku
- V průběhu transakce může být konzistence krátkodobě narušena

# Isolation

- Souběžně zpracovávané transakce **se ovlivňují pouze povoleným způsobem**, který je určen izolační úrovní transakce
  - Např.: Nepotvrzené změny v rámci transakce by neměly být viditelné jiným transakcím v dané databázi
- Výchozí úroveň izolace v Microsoft SQL Serveru je **read committed**
  - Neřeší celou řadu **nebezpečných problémů** při souběžném zpracování
- Je nutné chápat rozdíly mezi chováním jednotlivých izolačních úrovní a **správně nastavit izolační úroveň** transakce dle potřeb
  - Čím silnější je ochrana a izolace, tím ale vznikají silnější a déle trvající zámky, což snižuje možnosti souběžného zpracování

# Durability

- Jakmile klient obdrží informaci, že transakce byla potvrzena (committed), je **garantováno**, že změny jsou trvale uloženy
- Zpracování transakcí na SQL Serveru je postaveno na přístupu **ARIES** (**Algorithms for Recovery and Isolation Exploiting Semantics**)
- Jeho hlavní principy jsou:
  - **Write-ahead logging** – Jakákoliv změna je nejdříve zapsána do logu, před tím, než je zapsána na disk do datového souboru (garance atomičnosti a trvalosti změn)
  - **Repeating history during Redo** – Při restartu po havárii jsou identifikovány a zopakovány operace, jejichž změny se ztratily, aby byly zapsány potvrzené změny
  - **Logging changes during Undo** – Při rušení změn nepotvrzených transakcí tyto změny logujeme, aby při opakovaném restartu se akce neopakovaly

Blíže vysvětleno na přednášce Microsoft SQL Server Database Recovery Internals
Zdroj: https://cs.stanford.edu/people/chrismre/cs345/rl/aries.pdf

# Proces modifikace dat v databázi

1. Uživatel odesílá UPDATE příkaz
2. Je otevřena transakce a tato informace je zapsána do transakčního logu
3. Jsou načteny ty datové stránky z disku do paměti, které budou modifikovány
4. Jsou postupně provedeny změny datových stránek v paměti a popis změn je zapsán do transakčního logu
5. Po dokončení všech změn v datových stránkách je do transakčního logu zapsán COMMIT transakce
6. Klientovi je odesláno potvrzení o dokončení transakce, což mu garantuje, že změny jsou trvale uloženy
7. Všechny změněné datové stránky jsou později propsány operací CHECKPOINT do datového souboru na disk (i nepotvrzené změny)

# Osnova

1. Transakce
2. **Zámky**
3. Izolační úrovně a problémy souběžného zpracování
4. Trochu optimizmu na závěr, aneb nešlo by to bez zámků?

# Pessimistic vs. optimistic concurrency

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

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

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

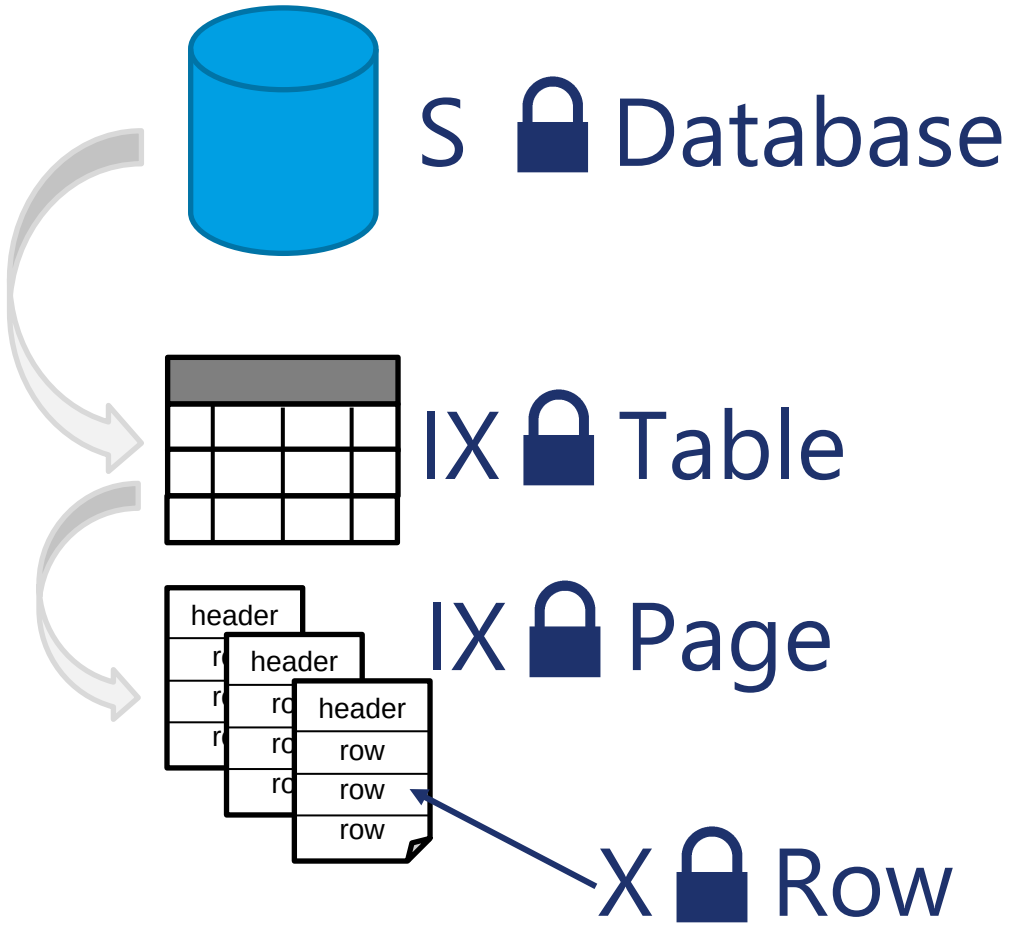
Blíže vysvětleno na přednášce Optimistic concurrency control in Microsoft SQL Server

# Jak fungují zámky

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

```
UPDATE Users
SET UserName = ‘David’
WHERE UserID = 10
```

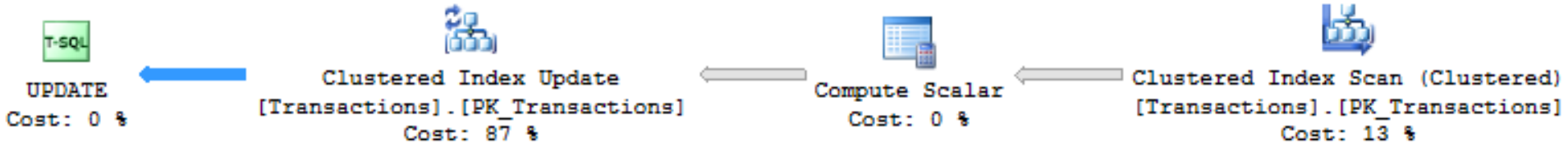
   - Pro provedení samotné modifikace dat je třeba mít exkluzivně uzamknutý řádek
   - Zamykat je však třeba směrem od tabulky s pomocí Intent eXclusive zámku, abychom předešli deadlocku



# Typy zámků v SQL Serveru

- Shared Lock
  - Je možné zdroj číst, ale neměnit
- Update Lock
  - Signalizace, že budeme měnit, ale zatím neměníme
  - Nevytváří se rovnou exkluzivní zámek kvůli snížení blokování čtení
  - Sdílený zámek ve fázi čtení by naopak vedl ke vzniku deadlocků

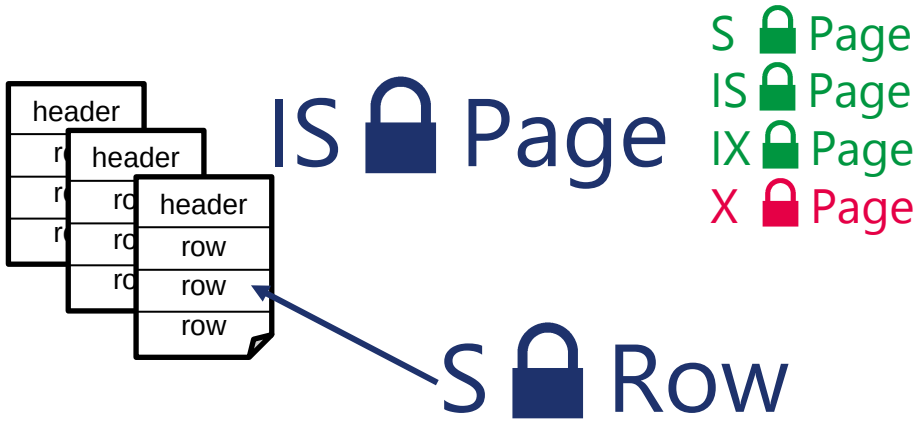
| Požadovaný zámek \ Aktivní zámek | Shared | Update | Exclusive |
|---|---|---|---|
| **Shared** | GRANT | GRANT | WAIT |
| **Update** | GRANT | WAIT | WAIT |
| **Exclusive** | WAIT | WAIT | WAIT |

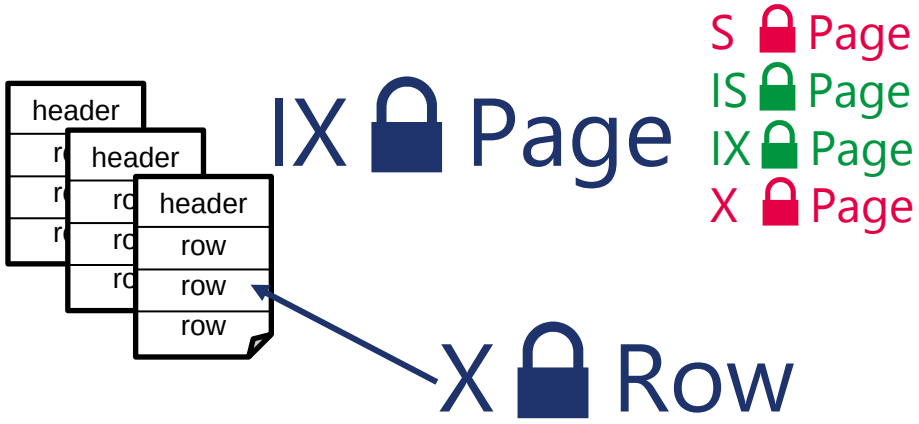



- Exclusive Lock
  - Je možné zdroj modifikovat

# Typy zámků v SQL Serveru

- Intent Lock
  - Chrání před vytvořením nekompatibilního zámku nad nadřazeným objektem



# Typy zámků v SQL Serveru

- **Schema modification (Sch-M) lock**
  - Při provádění DDL operací využívá SQL Server **schema modification (Sch-M) zámky**, které zablokují úplně všechny přístupy k danému objektu

- **Schema stability (Sch-S) lock**
  - SQL Server dále využívá **schema stability (Sch-S) zámky**, když kompiluje a spouští dotazy
  - Brání změnám ve struktuře databáze během kompilace a běhu dotazu
  - Tyto zámky neblokují žádné zámky vznikající při práci s daty (jsou kompatibilní i s exkluzivními zámky)

# Rozsah zámků

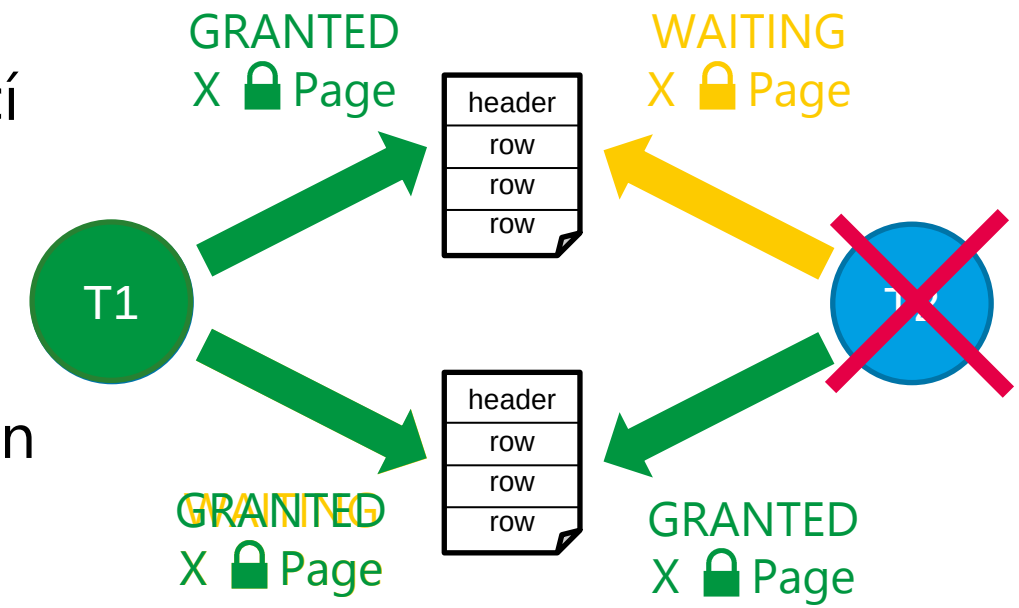
- SQL Server určí při kompilaci dotazu potřebné rozsahy zámků
  - Snaha je co nejméně blokovat ostatní

- Lock Escalation
  - Pokud je při běhu dotazu překročena hranice 5000 zámků nad jednou tabulkou nebo indexem
  - Zvýšena úroveň zámků, čímž jsou sníženy nároky na zdroje (paměť)


Blíže vysvětleno na přednášce Partitioning v Microsoft SQL Serveru

# Deadlock

- Deadlock nastává, když se dvě nebo více operací **trvale vzájemně zablokují zámky**
  - Transakce A se nemůže dokončit, dokud se nedokončí transakce B, ta je ale blokována transakcí A
  - Bez vnějšího zásahu nemá problém řešení
  - Deadlock monitor ukončí jednu z transakcí (Error 1205, Deadlock victim)
  - Ladění je mnohdy obtížné, protože se deadlocky objevují náhodně
    - System_health Extended Events Session
  - Doporučená úprava aplikace
    - Přístup ke zdrojům ve stejném pořadí



# Osnova

1. Transakce
2. Zámky
3. **Izolační úrovně a problémy souběžného zpracování**
4. Trochu optimizmu na závěr, aneb nešlo by to bez zámků?

# Lost Updates

- Tento problém nastává, pokud dvě transakce načtou řádek a ten následně aktualizují na základě již načtené hodnoty
- Např.:
  - **Editor A** si v 8:00 stáhne článek a celý den na něm pracuje
  - **Editor B** si v 9:00 stáhne článek a celý den na něm pracuje
  - **Editor A** uloží novou verzi článku v 15:00
  - **Editor B** přepíše článek svou verzí v 15:30 a tím je práce **editora A** ztracena
- Řešení:
  - **Editor A** si článek **exkluzivně uzamkne** a znemožní tak ostatním článek vůbec načíst za účelem následné editace

# Lost Updates

- Microsoft SQL Server nás chrání před problémem ztracených změn na úrovni souběžných příkazů UPDATE
  - UPDATE příkaz vytváří UPDATE zámky, které jsou vzájemně nekompatibilní a tím znemožní ostatním příkazům UPDATE načíst data pro následnou modifikaci
- Tento problém ale **můžeme jednoduše vytvořit ve vlastním kódu**:
  1. Načteme zdrojová data do dočasné tabulky / proměnné (originální data zůstávají v tabulce bez zámků a mohou být ostatními měněna)
  2. Aktualizujeme data na základě naší kopie z dočasné tabulky nebo proměnné a tím můžeme přepsat změny, které byly nad daty mezitím potvrzeny

# Dirty Read (Uncommitted Dependency)

- Čtení nepotvrzených změn (mezivýsledků) jiné transakce

**Session 1**

```
SELECT UserName
FROM Users
-- Načte UserName = David
```

**Session 2**

```
BEGIN TRANSACTION
UPDATE Users
SET UserName = 'Gesvindr'
WHERE UserID = 1
-- Zatím nepotvrzená změna
```

```
SELECT UserName
FROM Users WITH(NOLOCK)
-- Načte UserName = Gesvindr
-- Nepotvrzená hodnota
```

```
ROLLBACK TRANSACTION
```

# Level 0 – Read Uncommitted

## Problém: Dirty Read (a všechny ostatní zmíněné později)

- Při čtení dat v této izolační úrovni **nejsou vytvářeny sdílené zámky**
  - Transakce může číst i nepotvrzená data a není blokována exkluzivními zámky

- **Modifikace dat vytváří exkluzivní zámky vždy nehledě na izolační úroveň**

- Izolační úroveň Read Uncommitted je **velmi nebezpečná**, protože kompletně narušujeme izolaci transakcí a **čteme tak data pochybné kvality**, která by nikdo neměl vidět
  - Tato izolační úroveň je zneužívána jako univerzální řešení problémů se zámky
  - **Používat ji pro potřeby reportingu v systému, kde probíhají změny je krajně nevhodné**

# Level 1 – Read Committed

## Missing and double Reads + Nonrepeatable Reads + Phantom Reads

- Pro přečtení záznamu **musí být vytvořen sdílený zámek**
  - Není tedy možné číst nepotvrzené změny cizích transakcí, protože jejich nekompatibilní exkluzivní zámky jsou uvolněny až na konci transakce po potvrzení změn
- **Sdílené zámky jsou uvolňovány hned po přečtení řádku**
  - Díky tomu může dojít k řadě problémů jako je neopakovatelné čtení nebo dvojité čtení
- Tato výchozí izolační úroveň neposkytuje garanci kvality čtených dat
  - Při čtení tabulky s probíhajícími změnami může dojít k načtení **nekonzistentních či neplatných výsledků dotazu**

# Missing and double reads

- Pokud dojde k takové aktualizaci řádku během probíhajícího čtení, kdy je řádek v indexu přesunut fyzicky z již přečtené oblasti do ještě nepřečtené oblasti indexu, tak je následně **přečten dvakrát**
  - Tento problém nastává, pokud jsou sdílené zámky uvolňovány hned po přečtení záznamu a tím může dojít k jeho přesunu během probíhajícího čtení
  - Obranou je izolační úroveň **repeatable read**, kdy se sdílené zámky uvolní až na konci čtení

- Podobně, pokud je řádek přesunut během čtení z nepřečtené oblasti do již přečtené oblasti indexu, **není přečten vůbec**
  - Zde nutné využít izolační úroveň **serializable**, aby nepřečtená oblast byla uzamčena range-zámky

# Nonrepeatable Read

- Při opakovaném čtení dat v téže transakci přečteme jiná data
  - Sdílené zámky jsou uvolněny hned po přečtení řádku, který může být tím pádem aktualizován jinou transakcí

**Session 1**

```
BEGIN TRANSACTION
SELECT * FROM Users
```

**Session 2**

```
UPDATE Users
SET UserName = 'Gesvindr'
WHERE UserID = 1
-- Potvrzená změna
```

```
SELECT * FROM Users
/* Čte ve stejné transakci,
ale přečte jiný výsledek */
```

# Level 2 – Repeatable Read

## Phantom Read

- Sdílené zámky, které vzniknou kvůli čtení řádků, **jsou uvolněny až na konci transakce**

- Ostatní transakce mohou data číst, ale nemohou je měnit
  - Jakákoliv přečtená data jsou po celou dobu otevřené transakce stejná
  - Mohou ale přibývat nové záznamy (Phantom Read)

- V této izolační úrovni **hrozí rizika rozsáhlého blokování zápisu** do dat, která jsme kdykoliv během transakce přečetli
  - Přečtená data uzamkneme pro zápis až do konce transakce

# Phantom Read

- Při opakovaném čtení dat v rámci jedné aktivní transakce se objeví nový záznam

**Session 1**

```
BEGIN TRANSACTION
SELECT * FROM Users
```

---

**Session 2**

```
INSERT INTO Users
VALUES ('Gesvindr')
-- Potvrzená změna
```

---

```
SELECT * FROM Users
/* Čte ve stejné transakci,
ale přečte řádek navíc */
```

# Level 3 – Serializable

- Sdílené zámky, které vzniknou kvůli čtení řádků, **jsou uvolněny až na konci transakce**

- Transakce ale navíc uzamkne i rozsah čtených hodnot s pomocí zámků typu **range-locks**
  - Díky tomu nemůže být vložen do daného rozsahu 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 spouštěné operace 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 dostupných HINTů: https://docs.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-table?view=sql-server-ver15

# Když sdílené zámky při čtení nestačí

- V některých případech je nutné serializovat i spouštění příkazů SELECT, aby nedošlo k problémům při souběžném zpracování
  - Během čtení proto využíváme HINT **UPDLOCK**
  - Místo sdílených zámků vzniknou update zámky, které jsou vzájemně nekompatibilní, proto do naší „kritické sekce“ nevstoupí více transakcí současně

- Např.: Pokud neexistuje záznam, tak jej vlož, jinak jej aktualizuj
  - Fázi ověření existence záznamu by mohlo současně projít více souběžných procesů, které by následně všechny vložily nový řádek, místo toho, aby tak učinil jen první proces, a další řádek pouze aktualizovali

# Shrnutí doporučení pro vývojáře

- Nevyžadujte od uživatele žádné vstupy během transakce
  - Uživatel musí poskytnout všechny stupy před zahájením transakce, aby mohla proběhnout co nejrychleji a došlo tak brzkému uvolnění zámků
- Vytvářejte co nejkratší transakce a omezte objem zpracovaných dat
  - Nezahrnujte jako součást transakce zbytečné operace, které s ní nesouvisí
  - Pokud to jde, pošlete začátek transakce i její konec v jedné dávce
- Neotevírejte transakci na čtení dat, pokud to nepotřebujete
  - Čtení dat ve vyšších izolačních úrovních blokuje až do konce transakce zápis
- Zvažte využití optimistic concurrency
  - Díky tomu nebude docházet ke vzniku zámků

# Osnova

1. Transakce
2. Zámky
3. Izolační úrovně a problémy souběžného zpracování
4. **Trochu optimizmu na závěr, aneb nešlo by to bez zámků?**

# 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)

Blíže vysvětleno na přednášce Optimistic concurrency control in Microsoft SQL Server

# 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

# Negativní dopady na výkon

- 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

# Osnova

1. Transakce
2. Zámky
3. Izolační úrovně a problémy souběžného zpracování
4. Trochu optimizmu na závěr, aneb nešlo by to bez zámků?

# Dotazy

**Chcete si na toto téma společně povídat 5 dnů? Přijďte na GOC 631**

**RNDr. David Gešvindr, Ph.D.**

MVP: Data Platform | MCSE: Data Platform | MCT

david@wug.cz

@gesvindr