Microsoft SQL Server Index Internals

RNDr. David Gešvindr

MVP: Data Platform | MCSE: Data Management and Analytics | MCT

david@wug.cz

@gesvindr

Motivace

- Výpočetní prostředky počítačů neustále rostou, proto vzniká **mylná představa**, že většinu operací lze zpracovat **hrubou výpočetní silou** bez nutnosti optimalizovat samotný dotaz nebo algoritmus

- Mnohdy i triviální optimalizace ve zpracování dotazu může:
  - Řádově snížit dobu zpracování dotazu → **zkrácení odezvy**
  - Podstatně snížit jeho zátěž na SQL Server → **zvýšení propustnosti**

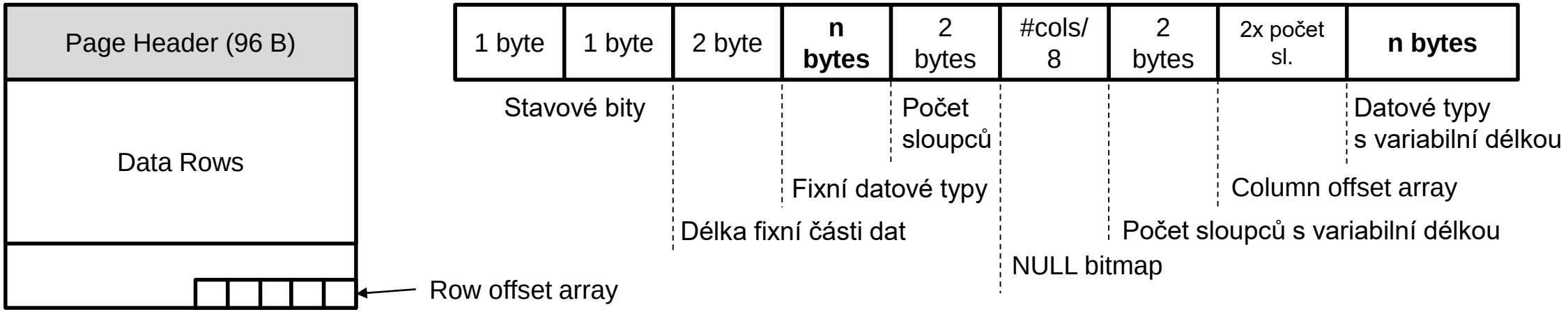
Jak jsou uložena data v SQL Serveru

- SQL Server ukládá data do **8 kB datových stránek (8192 bajtů)**
- Každá stránka je identifikována číslem souboru a svým sériovým číslem
  - Např.: 1:854
- Volné místo v datových souborech je spravováno ne po stránkách, ale po **64 kB extentech** – 8 po sobě jdoucích stránek
  - Pokud extent obsahuje jen stránky jednoho objektu, jedná se o **uniform extent**, naopak se jedná o **mixed extent**
- Existuje celá řada typů datových stránek, které se liší svým účelem

Zdroj: Kalen Delaney: Microsoft SQL Server 2008 Internals

Struktura datové stránky a řádku

- Index Page a Data Page mají velmi podobnou strukturu a liší se hlavně obsahem
  - **Index Page** – obsahuje data indexu
  - **Data Page** – obsahuje kompletní řádky tabulky
- Struktura stránky a řádku:



Zdroj: Kalen Delaney: Microsoft SQL Server 2008 Internals, str. 254 a 261

Jak zjistit složitost dotazu

- **Exekuční plán** popisuje, jaké fyzické operace SQL Server musel realizovat pro načtení dat
  - Je vyčíslena cena dotazu
  - **Pozor – Tato cena je vypočtena na základě odhadovaného exekučního plánu a nemusí odpovídat skutečnosti!**
  - Aktuální SSMS umí zobrazit skutečné složitosti vybraných operací (SQL Server 2016 a SQL Server 2014 SP2; SQL Server 2016 SP1 i wait statistiky na úrovni dotazu)
- **SET STATISTICS IO ON**
  - SQL Server ukáže počet I/O operací spojených s exekucí dotazu
- **SET STATISTICS TIME ON**
  - SQL Server ukáže pro každý příkaz v dávce dobu kompilace a exekuce

Zdroj: [1] https://blogs.msdn.microsoft.com/sql_server_team/extended-per-operator-level-performance-stats-for-query-processing/
[2] https://blogs.msdn.microsoft.com/sql_server_team/new-showplan-xml-properties-in-ssms-october-release/

Kde jsou uloženy řádky tabulky?

Heap

- Datové stránky **nemají definované pořadí** záznamů ani pořadí stránek
- IAM pouze udržuje seznam stránek, které tvoří tabulku
- Výhody:
  - Velmi jednoduché vložení záznamu – vkládáme kamkoliv, kde je místo
- Nevýhody:
  - Bez dodatečného indexu musíme vždy číst všechny řádky

Clustered Index

- Pořadí řádků ve stránce a pořadí datových stránek je určeno **klíčem indexu**
- Jedná se o B+ strom, který v listech (level 0) obsahuje **kompletní řádky tabulky**
  - Neexistuje jiná kompletní kopie řádku
- Výhody
  - Rychlé načítaní dat dle klíče
- Nevýhody:
  - Vyšší cena modifikace dat – musí být dodrženo pořadí záznamů a stránek

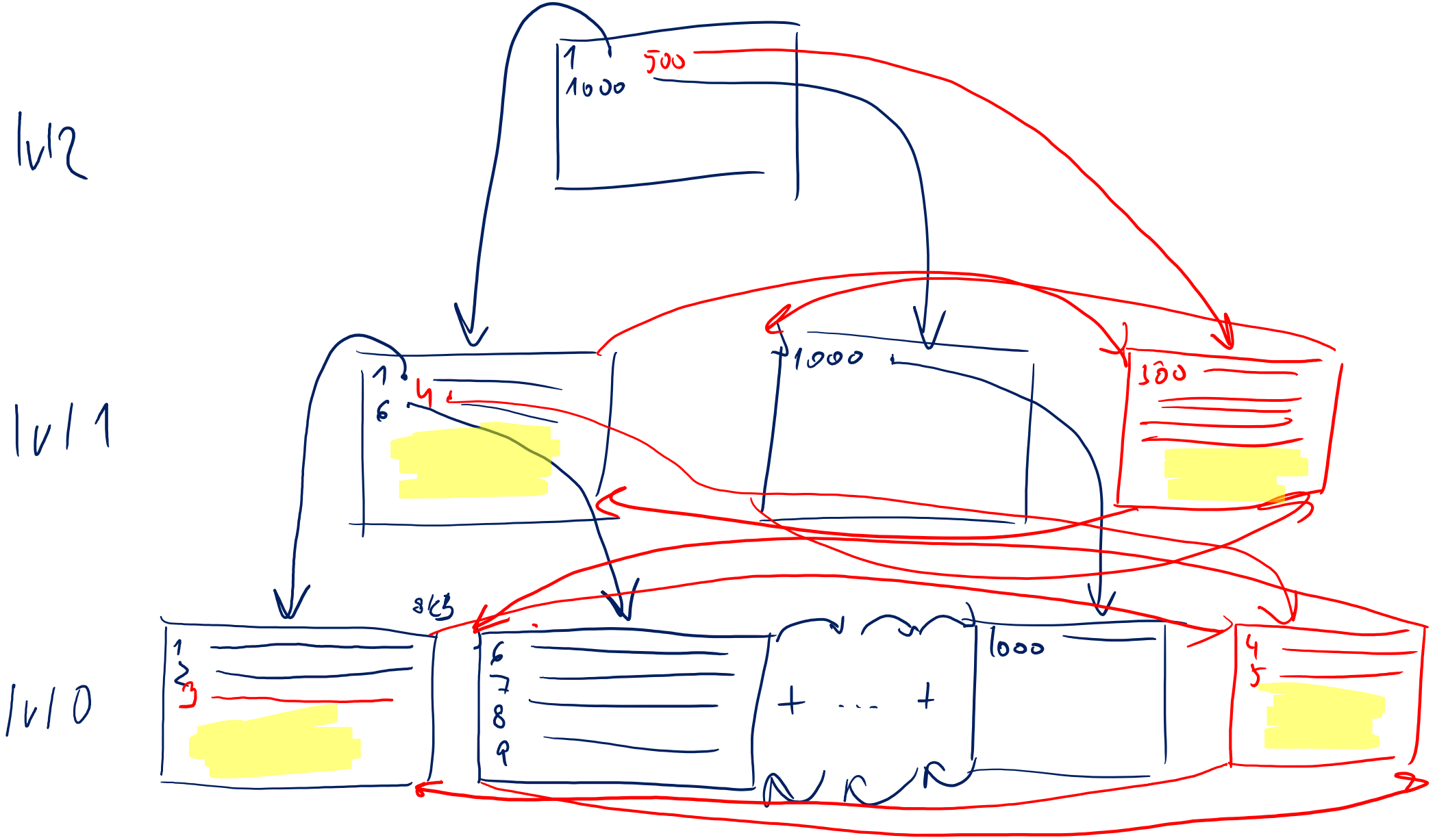
Doporučení pro použití clustered indexů

- V 99% případů se vyplatí použít clustered index
  - Heap se vyplatí použít v případech, kdy potřebujeme rychle uložit neseřazené záznamy, co zase budeme všechny číst (staging tabulky)
- Primární klíč a clusterovaný index spolu nesouvisí
  - **Primary Key constraint** je implementován pomocí **jedinečného indexu**
  - Při založení SQL Server automaticky volí clusterovaný index (nemusí být)
  - Primární klíč je většinou vhodným kandidátem na clusterovaný klíč
- Vytvoření clusterovaného indexu je velmi náročná operace
  - Dochází k rebuildu všech non-clustered indexů
- **Volba clusterovaného klíče je klíčová pro efektivitu indexu**

Pravidla výběru clusterovaného klíče

- Vhodný clusterovaný klíč má následující vlastnosti:
- **Unikátní**
  - Musí jednoznačně identifikovat záznam, jinak SQL Server doplní interně další 4 bajtový sloupec uniqueifier (který se propisuje do všech ostatních indexů)
- **Malá datová velikost**
  - Propisuje se do všech non-clustered indexů
- **Neměnný**
  - Změna klíče vede k přemístění řádku (rozdělení stránky, fragmentace) a aktualizaci všech neclusterovaných indexů nad tabulkou
- **Vzrůstající**
  - Nové hodnoty je výhodné zapisovat na konec indexu kvůli fragmentaci

Zdroj: Kalen Delaney: Microsoft SQL Server 2008 Internals, str. 312-313


lvl2
lvl1
lvl0
1
1000
500
1
6
4
1000
100
1
2
3
6
7
8
9
+ ... +
1000
4
5

Důležitá terminologie

- **Density**
  - Popisuje data ve sloupci a říká, jak často se opakují duplicitní záznamy
  - **Density = 1/[počet jedinečných hodnot ve sloupci]**
  - Vysoká hustota → méně jedinečná data

- **Selectivity**
  - Popisuje predikát/množinu
  - Podíl, kolik záznamů z tabulky vyhovuje predikátu
  - Vysoká selektivita → málo vracených záznamů

- **Kardinalita**
  - Počet řádků vracených operátorem při zpracování dotazu

Zdroj: https://blogs.msdn.microsoft.com/bartd/2011/01/25/query-tuning-fundamentals-density-predicates-selectivity-and-cardinality/

Důležitá terminologie

- **Index Depth**
  - Hloubka indexu
  - Level 0 jsou vždy listy
  - Nejvyšší index level je kořen; vždy pouze 1 stránka

- **Index Key**
  - Klíč indexu je sloupec popř. několik sloupců, které jsou zahrnuty do indexu

- **Maximum size of Index Keys**
  - 32 sloupců, 900 bajtů clustered index, 1700 bajtů non-clustered index

Zdroj: [1] https://technet.microsoft.com/en-us/library/ms191241(v=sql.105).aspx
[2] https://docs.microsoft.com/en-us/sql/sql-server/maximum-capacity-specifications-for-sql-server?view=sql-server-2017

Non-clustered Index

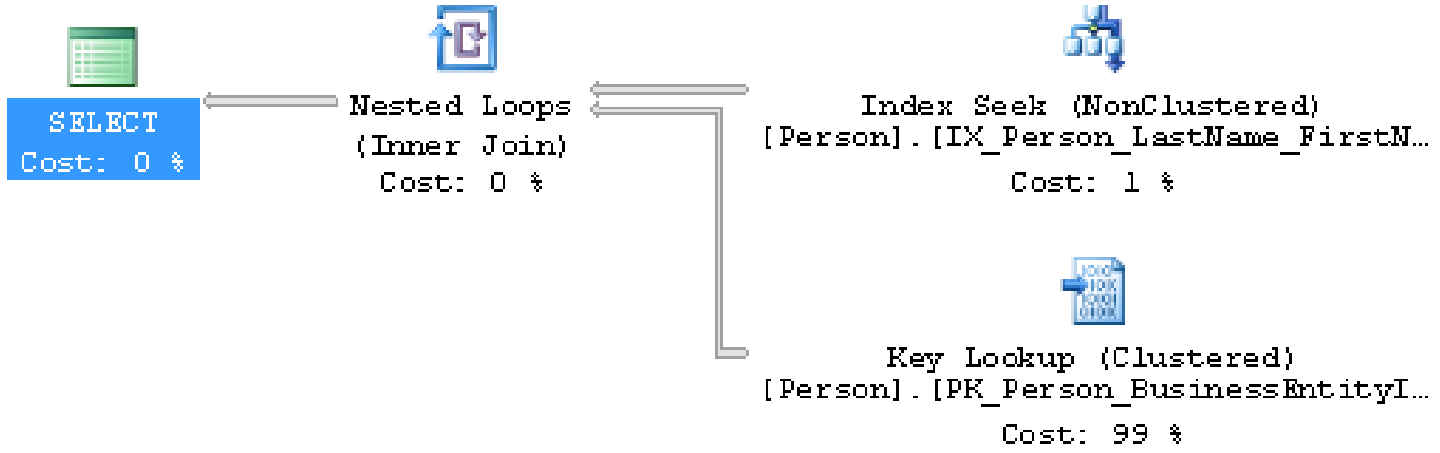
- Jedná se opět o B+ strom, jehož účelem je zefektivnit přístup k datům v tabulce

- Indexovací strategie:
  - Provádíme hledání podle vybraného sloupce
  - Načítáme pouze vybrané sloupce
  - Načítáme data seřazená podle daného sloupce

- Pokud požadujeme načtení dat, která nejsou v indexu, musí se načíst i zbytky řádků z clustered indexu nebo heap pomocí operace **Key Lookup (RID Lookup)**

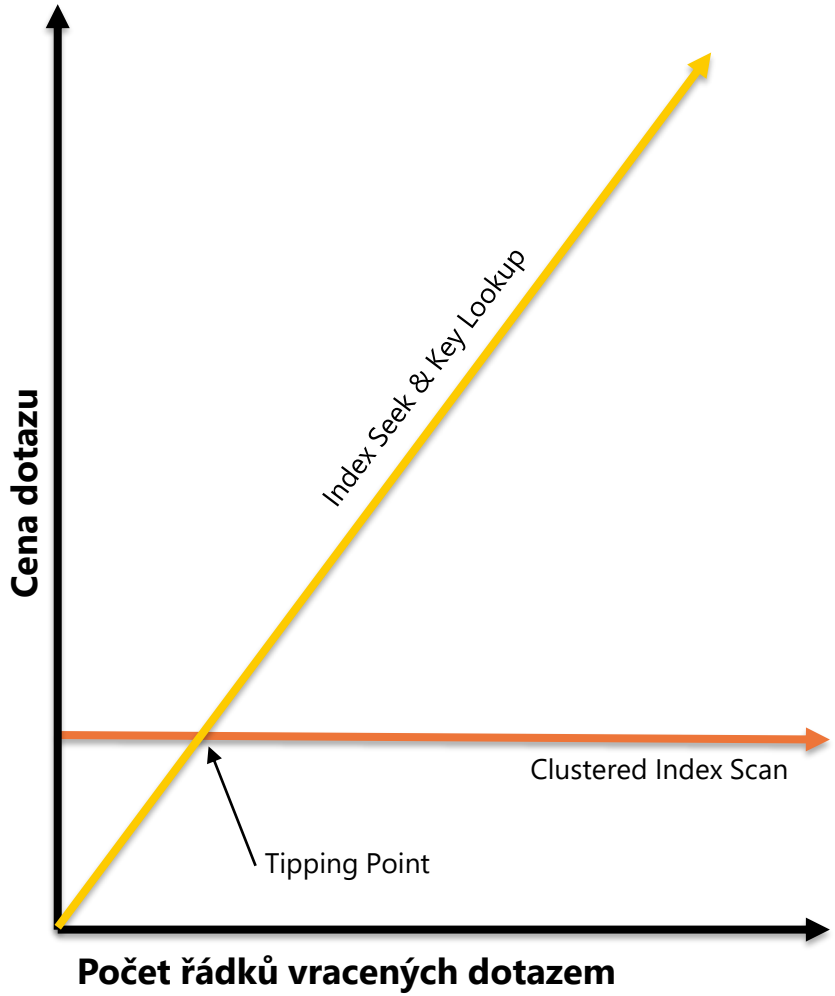
Key Lookup

- V neclusterovaném indexu je nalezen clusterovaný klíč nebo Row ID
- Chybějící sloupce jsou načteny s pomocí operací:
  - Key Lookup (clustered index)
  - RID Lookup (heap)



Kdy se vyplatí použít non-clustered index

- SQL Server **porovnává** náročnost hledání v non-clustered indexu a Key-Lookup operace proti náročnosti kompletního průchodu indexu (Clustered Index Scan)
- Hranice od které není dotaz dostatečně selektivní se označuje **Tipping Point**
- Je-li počet řádků vracených dotazem **menší než 25% datových stránek** clustered indexu, index se určitě použije
- Je-li počet řádků vracených dotazem **větší než 33% datových stránek** clustered indexu, index se určitě nepoužije



Zdroj: http://www.sqlskills.com/blogs/kimberly/the-tipping-point-query-answers/

Tipping Point – Příklad 1

- Tabulka
  - 500 000 datových stránek
  - 1 000 000 záznamů

- 25% datových stránek – 125 000 záznamů (12,5%)
- 33% datových stránek – 166 000 záznamů (16,6%)

- Index Seek v kombinaci s operací Key-Lookup se použije, pokud dotaz vrací méně, než **12,5% – 16,6%** řádků v závislosti na dalším nastavení SQL Serveru

Zdroj: http://www.sqlskills.com/blogs/kimberly/the-tipping-point-query-answers/

Tipping Point – Příklad 2

- Tabulka
  - 10 000 datových stránek
  - 1 000 000 záznamů

- 25% datových stránek – 2 500 záznamů (0,25%)
- 33% datových stránek – 3 333 záznamů (0,33%)

- Index Seek v kombinaci s operací Key-Lookup se použije, pokud dotaz vrací méně, než **0,25% – 0,33%** řádků v závislosti na dalším nastavení SQL Serveru

Zdroj: http://www.sqlskills.com/blogs/kimberly/the-tipping-point-query-answers/

Nevhodný návrh dotazu

- Efektivní dotaz načítá z databáze jen **nutné minimum informací**, které dostačuje aplikaci pro daný účel
  - Např.: Pro vypsání seznamu objednávek nemusím načítat 50 dalších sloupců v tabulce objednávek, když nejsou zobrazeny v UI

- Je důležité:
  - Vracet jen sloupce, co skutečně využijeme (pozor na SELECT * FROM tabulka)
  - Filtrovat a stránkovat záznamy na serveru
  - Zbytečně neřadit záznamy, pokud to opravdu nepotřebujeme

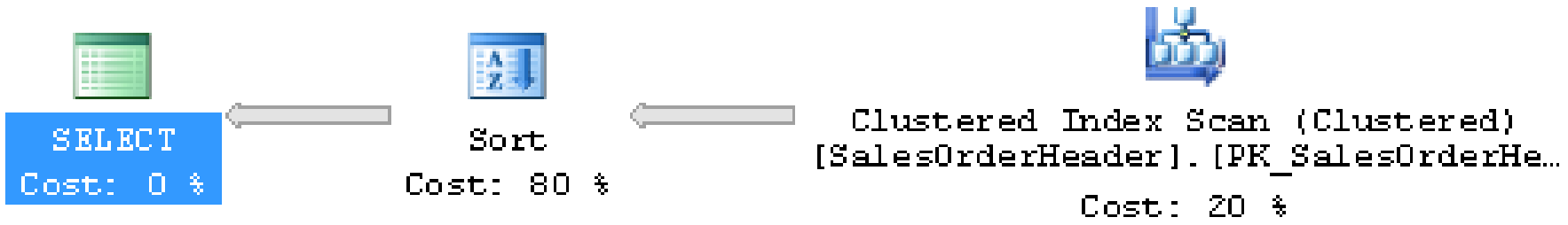
Covering Index

- Non-clustered index, který obsahuje všechna data, která požaduje dotaz
  - Není třeba načítat data z clusterovaného indexu
  - Odpadá drahá operace Key Lookup
  - **Velice efektivní přístup k datům**
- Protože existují omezení na velikost klíče indexu, je možné do listů indexu přidat další sloupce – **Included Columns**
- Snažíme se vytvořit index, který poskytne co nejvíce dat
  - Pozor – čím je index větší, tím je méně efektivní, protože způsobuje víc čtení

Řazení klíčů v indexu

- U každého klíče v indexu definujeme, jestli data mají být řazena vzestupně nebo sestupně

- Vhodným řazením dat v indexu je možné nahradit **velmi drahou** operaci Sort při zpracování dotazu



Filtrovaný index

- Do non-clustered indexu je možné přidat predikát, který definuje podmnožinu řádků, které budou indexovány

- Vhodné zejména pro sloupce s velmi vysokou density (datový typ **bit**), pokud jedna z hodnot má dobrou selektivitu a druhá nikoliv
  - Např.: IsOrderCompleted – **99% záznamů = 1**; **1% záznamů = 0**
  - Hledáme a indexujeme pouze rozpracované objednávky

Indexované computed columns

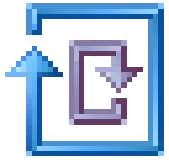
- Používáte-li v tabulce **computed column** a využitý výraz je deterministický – potom můžete pro zvýšení výkonu tento sloupec zaindexovat

- Možné použití:
  - V SQL Serveru 2016 je přidána podpora JSON dokumentů, ale na rozdíl od XML je není možné přímo indexovat
  - Pokud bychom chtěli efektivně vyhledávat podle hodnoty v nějaké části dokumentu, vytvoříme si computed column s funkcí JSON_VALUE, kam extrahujeme hodnotu z vybrané cesty v JSON dokumentu
  - Daný sloupec necháme indexovat

Indexovaný pohled

- Běžný databázový pohled vloží při volání svoji definici do dotazu, jenž jej volá
  - Databázový pohled nepřináší žádný výkonnostní benefit
- Indexovaný (materializovaný) pohled je **pohled, nad kterým je vytvořen clusterovaný index**
  - Celý pohled je vyhodnocen a výsledky jsou uloženy do clusterovaného indexu
  - V případě čtení dat se již nevyhodnocuje původní dotaz
  - **Pozor: Každá aktualizace souvisejících tabulek musí aktualizovat i tento pohled** (nevhodné pro měněné tabulky v rámci OLTP)
- Využití: Složité výpočty nad daty, co se mění jen dávkově

Načítání dat z více tabulek

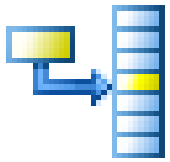


Nested Loops

- Pro každý načtený řádek se volá Index Seek v druhé tabulce

Merge Join

- Spojení záznamů pokud jsou seřazeny podle stejného sloupce

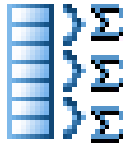


Hash Match

- Efektivní spojení většího množství záznamů, pokud nejsou seřazeny dle stejného slupce
- Je v prvním průchodu v paměti spočítána hashovací tabulka
- Při průchodu druhého zdroje se hashuje hodnota klíče a dohledávají se záznamy z první tabulky

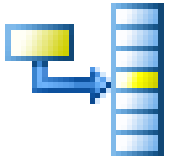
Zdroj: https://msdn.microsoft.com/en-us/library/ms191158.aspx

Operátory pro seskupení dat



Stream Aggregate

- Používán pro: GROUP BY
- Je možné jej použít, pokud záznamy jsou seřazené podle stejných sloupců jako jsou v GROUP BY



Hash Match (Aggregate)

- Čte záznamy na vstupu a s pomocí hash funkce je zařazuje do skupin

Zdroj: https://msdn.microsoft.com/en-us/library/ms191158.aspx

Paralelní zpracování

Distribute Streams

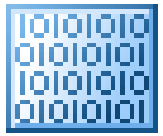
- Záznamy rozdělí na více paralelně zpracovávaných výstupů

Gather Streams

- Sloučí se vstupy z několika paralelně zpracovávaných vstupů do jednoho výstupu

Repartition Streams

- Záznamy z několika vstupů rozdělí na více paralelně zpracovávaných výstupů



Bitmap Filter

- Zavčasu rychle odstraní záznamy, které je zbytečné zpracovávat a tím zvýší výkon paralelních operací

Zdroj: https://msdn.microsoft.com/en-us/library/ms191158.aspx

Údržba indexů

- Pro odstranění fragmentace je možné provést 2 různé operace údržby indexů:

- **Reorganize**
  - Přeskupení dat v datových stránkách na úrovni listů B+ stromu
  - Menší zátěž ale i menší efektivita
  - Online operace

- **Rebuild**
  - Index je kompletně zrušen a znovu vygenerován
  - Velmi náročná operace, zátěž i na transakční log
  - Neprobíhá online (kromě edice Enterprise)

Zdroj: https://msdn.microsoft.com/en-us/library/ms189858.aspx

Fill Factor

- Je možné SQL Server instruovat k tomu, aby při vytvoření indexu nebo jeho REBUILDu nevyužil 100% volného místa v datových stránkách
  - Ponechání volného místa dává smysl u často měněných tabulek, kde vlivem updatů dochází k častým page splitům
  - Hodnotu lze modifikovat na úrovni serverového nastavení nebo na úrovni indexu
  - **Je nutné otestovat a nastavit tak, aby na daném systému posléze došlo ke snížení počtu page splitů**
- Doporučené hodnoty
  - 100 nebo 0 – statická data nebo ideální clusterovaný klíč
  - 90-70% - často měněná data

Zdroj: http://sqlmag.com/blog/what-best-value-fill-factor-index-fill-factor-and-performance-part-2

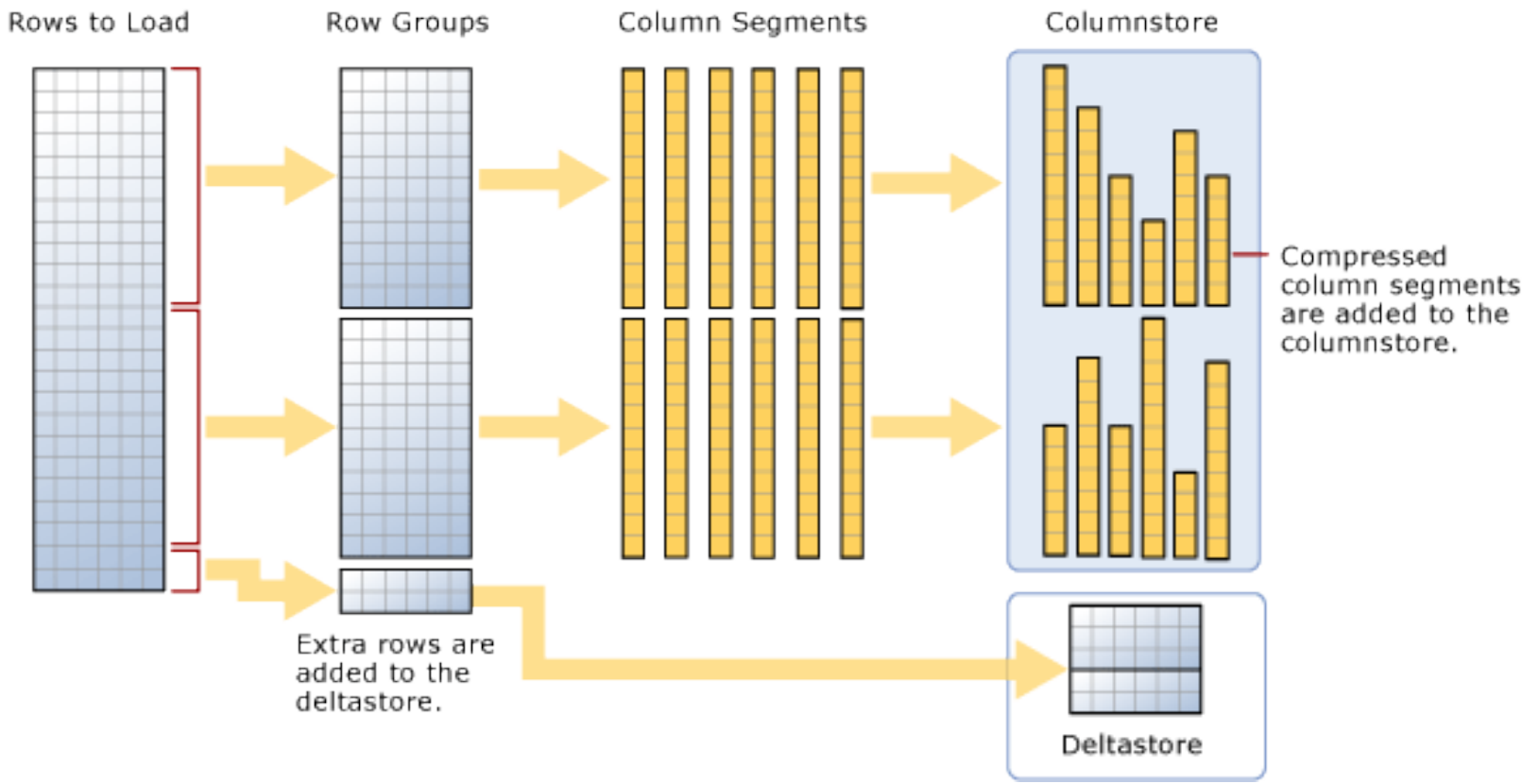
Columnstore Indexy

Columnstore Index

- V SQL Serveru běžně platí, že data jsou uložena **po řádcích** v clustered indexu nebo heap
- **Columnstore Index ukládá data komprimovaná po sloupcích**
- S pomocí Columnstore Indexů lze dosáhnout až 100x zvýšení výkonu analytických dotazů
  - Rozsáhlé agregace nad datovým skladem, které čtou veliké rozsahy řádků
- Velmi silná komprese umožňuje dosáhnout až 10 násobného zmenšení uložených dat – rozsáhlé tabulky faktů

Terminologie Columnstore Indexů

- Data jsou rozdělena do RowGroups (až 1 048 576 řádků)



Zdroj: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-data-loading-guidance

Různé verze Columnstore Indexů

- SQL Server 2012
  - Přinesl podporu pro **Read-only nonclustered columnstore index**
  - Využití pro Data Warehouse, složité vkládání nových dat
- SQL Server 2014
  - Uvedl **Updateable clustered columnstore index**, ale nebylo možné vytvořit jiné nonclustered indexy
  - Využití pro Data Warehouse, nevhodné pro OLTP zátěž
- SQL Server 2016 uvádí 2 novinky:
  - **Updateable nonclustered columnstore index**
  - Btree index on a clustered columnstore index

Vhodné použití Column Store indexů

- Dotazy, které **čtou celou tabulku**
  - Agregace dat
  - Rozsáhlé tabulky faktů v datových skladech

- Cloumn Store Index **nepodporuje operaci seek**
  - **Nevhodný pro vyhledávání individuálních řádků**

- Rozšíření OLTP tabulek s B+ stromovým clustered indexem o **aktualizovatelný non-clustered column store index**

Jak Column Store indexy zvyšují výkon

- **Data Compression**
  - Až 10x menší objem data uložený na disku a načítaný z disku (menší I/O)
  - Data se uchovávají komprimovaná i v operační paměti
- **Column Elimination**
  - Díky uložení po sloupcích se nemusí vůbec načítat data sloupců, se kterými nepracujeme
- **Rowgroup Elimination**
  - U každé rowgroup si SQL ukládá metadata o minimální a maximální hodnotě v každém sloupci
  - Filtrování na úrovni celých rowgroups

Zdroj: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-query-performance

Jak Column Store indexy zvyšují výkon

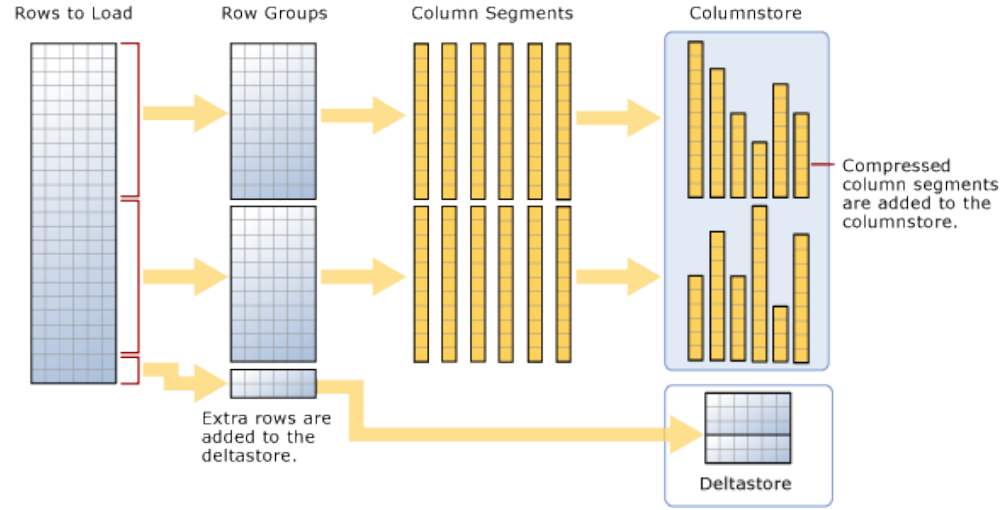
- **Batch Mode Execution**
  - Vybrané operátory se spouští v dávkovém režimu (batch mode), kdy pracují až s 900 řádky najednou, místo spouštění řádek po řádku (row mode)
  - Umí pracovat nad komprimovanými daty a snižují zbytečnou zátěž

- **Aggregate Pushdown**
  - Vybrané operace na výpočet agregačních funkcí jsou přeneseny až na operaci SCAN, která načítá data
  - Pouze funkce MIN, MAX, SUM, COUNT, AVG nad datovými typy <= 64 bit

Zdroj: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-query-performance

Efektivní načítání dat do Columnstore Indexu

- Při načítání dat do **clustered columnstore indexu** je důležité se vyhnout použití Delta Store
  - Neplatí pro non-clustered columnstore indexy
- Dosáhneme toho dávkovým načítáním dat
- Velikost dávek musí být alespoň 102 400 řádků, aby se zapisovaly rovnou do CI
  - Netřeba data řadit
  - Nepoužívat TABLOCK
  - Minimální logování



Zdroj: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-data-loading-guidance

Real-Time Operational Analytics

- V současnosti se pro analýzu dat z OLTP systému data kopírují do datového skladu, kde se pak analyzují
  - S tím jsou spojené další náklady na nasazení a provoz
  - Komplexní řešení složené z několika propojených služeb
  - Toto řešení pracuje se zpožděnými daty

- **Real-Time Operational Analytics**
  - Pokud v OLTP databázi informačního systému vytvoříme **Updateable nonclustered columnstore index**
  - Minimální dopad na výkon OLTP databáze
  - Možnost spouštět analytické dotazy s vysokým výkonem

Real-Time Operational Analytics

- Columnstore index je možné vytvořit nad klasickou tabulkou na disku, přičemž i zde se dostaví znatelné zlepšení výkonu při dotazování

- Je také možné vytvořit **In-Memory Columnstore Index**
  - Vytváří se vždy nad všemi sloupci memory optimized tabulky
  - Další zvýšení výkonu, protože se index nenačítá z disku
  - Snížení režie při zápisu, protože i deltastore je jako memory optimized tabulka

In-Memory 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

Výkon In-memory OLTP

- In-memory technologie poskytuje vysoký výkon
- Microsoft v testech demonstroval [1]:
  - 1 milion batch requests/s při čtení a zápisu 4KB dat
  - Ukládání řádků rychlostí 10 milionů hodnot za vteřinu (100B na řádek)
  - Zpracování transakcí s objednávkami rychlostí 260 000 transakcí za vteřinu
- In-memory řešení a zkušenosti s ním popisuje i společnost bwin [2]:
  - Použití pro cachování session state webové aplikace
  - Ve verzi 2016 dosáhli přes 1 200 000 batch req/sec

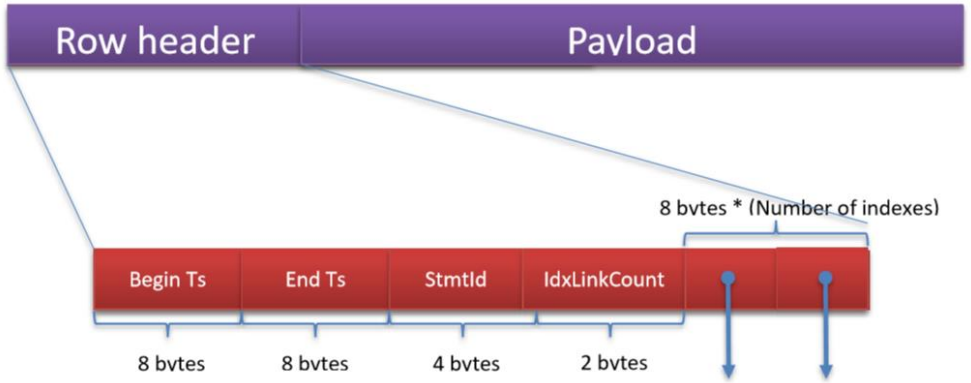
[1] Kalen Delaney: SQL Server In-Memory OLTP Internals for SQL Server 2016, Technical White Paper
[2] https://blogs.msdn.microsoft.com/sqlcat/2016/10/26/how-bwin-is-using-sql-server-2016-in-memory-oltp-to-achieve-unprecedented-performance-and-scale/

Memory-optimized tabulky

- Veškerá data jsou uložená v operační paměti
- Je možné si vybrat mezi:
  - Trvalým uložením dat (SCHEMA_AND_DATA)
  - Dočasným uložením dat (SCHEMA_ONLY)
- Transakce mají stejné vlastnosti (ACID)
- Pro trvalé uložení dat jsou využity checkpoint soubory
  - Používány jen pro zotavení po restartu serveru
- Je využíván stejný transakční log, ale efektivněji
  - Tabulky SCHEMA_ONLY do transakčního logu nezapisují vůbec a dosahují ještě lepšího výkonu

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

Nativně kompilované moduly

- Největšího výkonnostního nárůstu lze dosáhnout díky použití nativně kompilovaných procedur a funkcí
- T-SQL kód je přeložen do C a následně zkompilován do strojového kódu v podobě DLL, která je načtená do databáze (děje se automaticky)
- Vygenerovaný kód procedury pracuje v paměti přímo s datovými strukturami použitých tabulek
  - Funguje proto pouze pro memory-optimized tabulky
- Nejsou povolené určité jazykové konstrukce v jazyce T-SQL
  - https://msdn.microsoft.com/en-us/library/dn246937.aspx

Vhodné použití In-Memory OLTP

- Problémy se zámky (locks i latches)
- Vysoká I/O zátěž
- Veliké množství INSERTů a SELECTů
- Požadavky na velmi rychlé business transakce
- Ukládání session z webserverů

- Nevhodné použití:
  - Jiná zátěž než OLTP (datové sklady)
  - Závislost aplikace na mechanizmu zámků

Dotazy

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

RNDr. David Gešvindr

MVP: Data Platform | MCSE: Data Management and Analytics | MCT

david@wug.cz

@gesvindr