Monitoring a optimalizace databázových dotazů v Microsoft SQL Serveru

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**

Osnova

1. Aplikace jede pomalu, co s tím?
2. Jak nám pomohou indexy
3. Jak systematicky přistupovat k optimalizaci
4. Další zabijáci výkonu dotazů

Osnova

1. **Aplikace jede pomalu, co s tím?**
2. Jak nám pomohou indexy
3. Jak systematicky přistupovat k optimalizaci
4. Další zabijáci výkonu dotazů

Metody monitorování T-SQL dotazů a serveru

- SQL Server Profiler
- SQL Trace
- Extended Events

- Dynamic Management Views and Functions
- SQL Server Management Studio
  - Activity Monitor
  - Reports
- Performance Monitor
- Data Collector
- Query Store

SQL Server Profiler

- Grafický nástroj pro **trasování**
  - Zachytávání událostí, které se stanou v SQL Serveru
  - Interně využívá SQL Trace

- Kdy se používá
  - Identifikace náročných databázových dotazů
  - Sběr informací o dalších událostech SQL Serveru (deadlock)
  - Zachycení zátěže pro SQL Server Tuning Advisor
  - Zachycení zátěže pro „replay“
  - Sledování bezpečnostních událostí

- **V SQL Serveru 2016 je deprecated, používejte Extended Events**

Porovnání SQL Server Profileru a SQL Trace

SQL Server Profiler

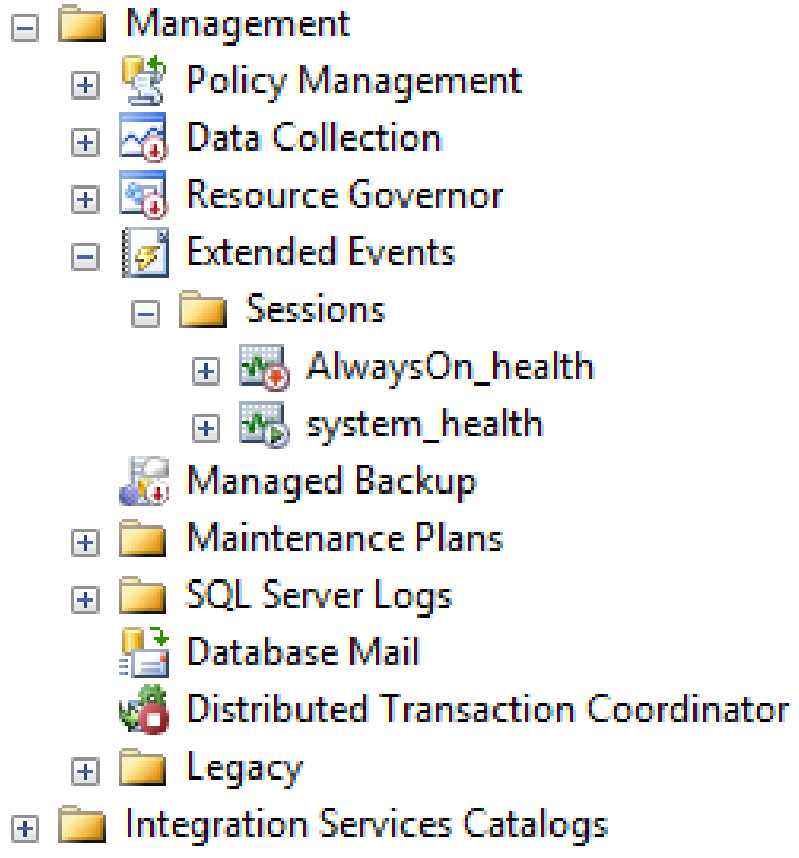
- Externí utilita
- Určeno pro **krátkodobé sledování** a časté změny konfigurace sledování
- Primárně pro výpis do UI, případně uložení do souboru nebo databáze

SQL Trace

- Interní služba SQL Serveru
- Určeno na **dlouhodobé sledování** s minimálním dopadem na výkon
  - Je nutné zapisovat jen co nejmenší množství událostí – dobře nastavit filtry
- Primárně pro zápis do souboru

Extended Events

- Technologie, která nahrazuje SQL Trace a dále rozšiřuje jeho možnosti
  - SQL Trace je označen jako deprecated
- Je k dispozici od SQL Serveru 2008
- Ale v Management Studiu je podpora až od **SQL Serveru 2012**
- Způsobuje minimální zátěž pro SQL Server
- Konfigurace s pomocí přehledného T-SQL kódu



system_health Session

- Extended Events session, která je automaticky aktivní po instalaci instance SQL Serveru

- Monitoruje závažné události v SQL Serveru:
  - https://msdn.microsoft.com/en-us/library/ff877955.aspx
  - Např:
    - Závažné chyby SQL Serveru (severity > 20)
    - Transakce čekající na získání zámku déle než 30 vteřin
    - Vznik deadlocku

Osnova

1. Aplikace jede pomalu, co s tím?
2. **Jak nám pomohou indexy**
3. Jak systematicky přistupovat k optimalizaci
4. Další zabijáci výkonu dotazů

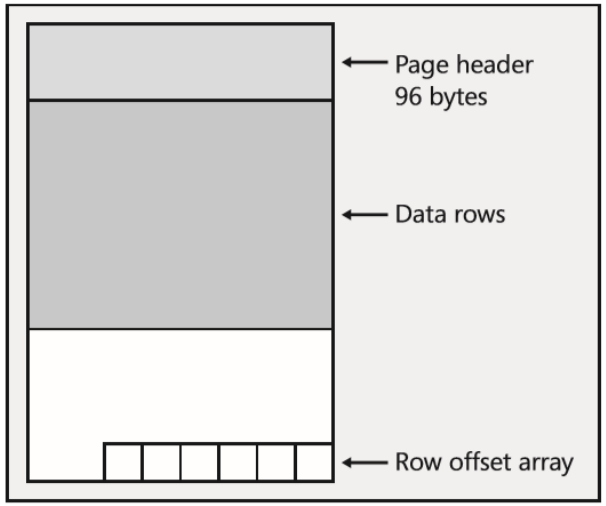
Jak jsou uložena data v SQL Serveru

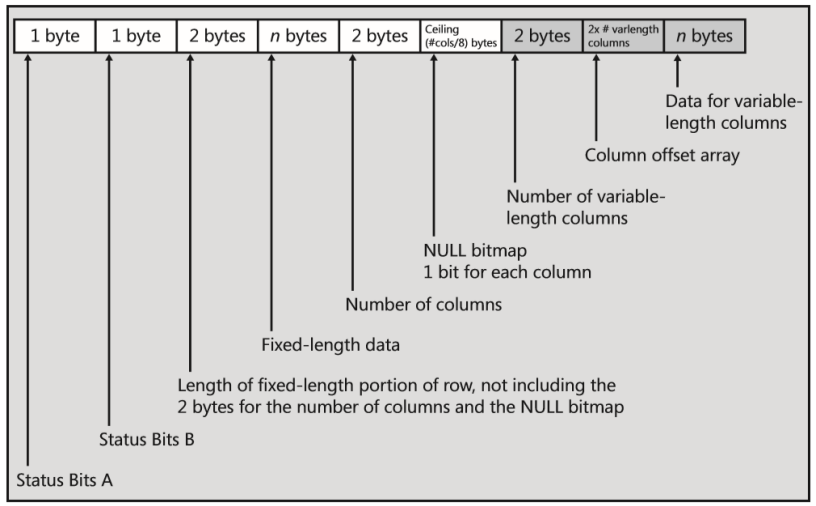
- SQL Server ukládá data do **8 kB datových stránek (8192 bajtů)**
- Každá stránka je identifikována číslem souboru a svým sériovým číslem
  - Např.: 1:854
- Volné místo v datových souborech je spravováno ne po stránkách, ale po **64 kB extentech** – 8 po sobě jdoucích stránek
  - Pokud extent obsahuje 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!**
  - SSMS 2016 umí zobrazit skutečné složitosti vybraných operací

- **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

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
- 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

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**
  - 16 sloupců, 900 bajtů

Zdroj: https://technet.microsoft.com/en-us/library/ms191241(v=sql.105).aspx

Non-clustered Index

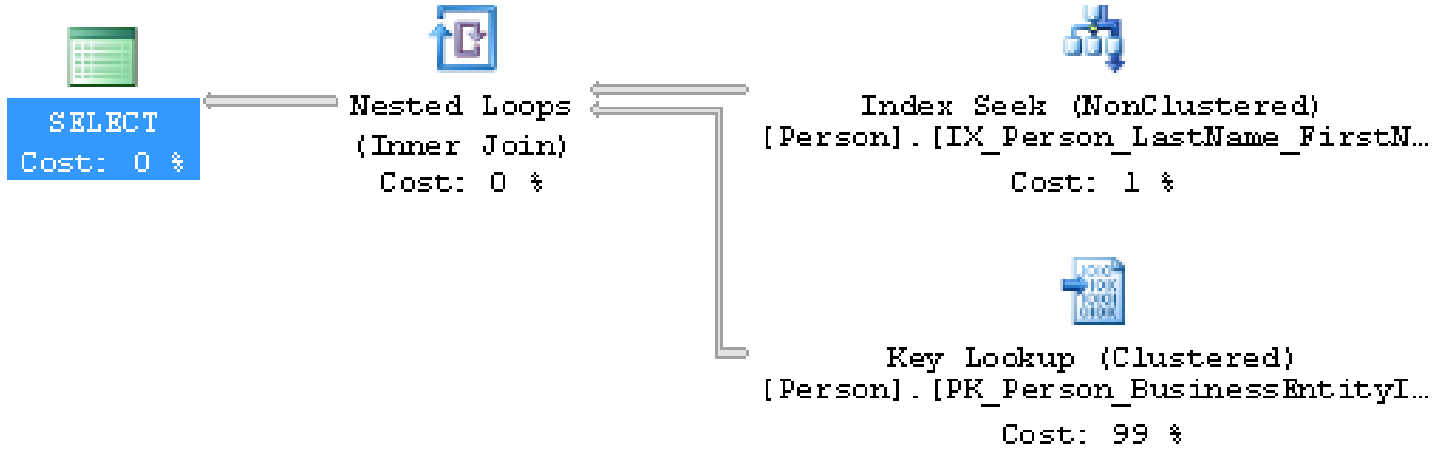
- Jedná se opět o B+ strom, jehož účelem je zefektivnit přístup k datům v tabulce

- 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)**

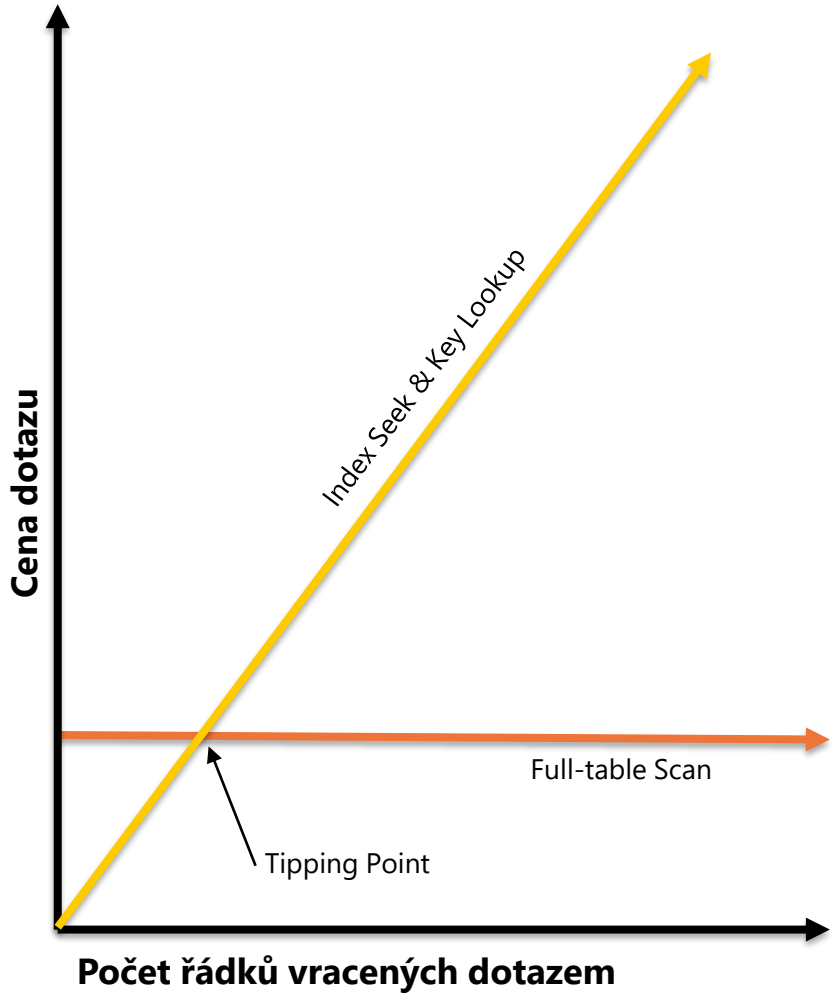
Key Lookup

- 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/

Covering Index

- 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**

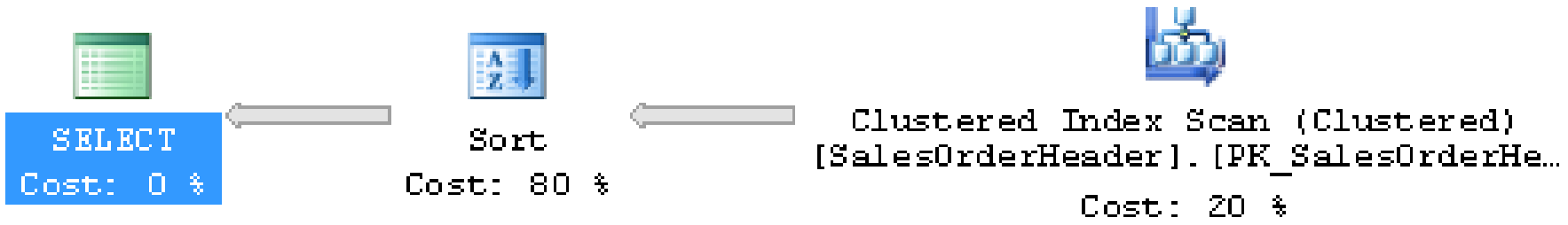
- Protože existují omezení na velikost klíče indexu, je možné do listů indexu přidat další sloupce – **Included Columns**

- 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 špatnou 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

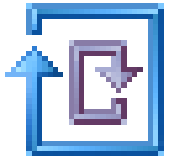
- Používáte-li v tabulce **computed column** a využitý výraz je deterministický – potom můžete pro zvýšení výkonu tento sloupec zaindexovat

- 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

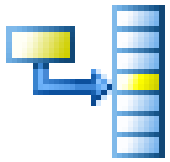


Nested Loops

- 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

Osnova

1. Aplikace jede pomalu, co s tím?
2. Jak nám pomohou indexy
3. **Jak systematicky přistupovat k optimalizaci**
4. Další zabijáci výkonu dotazů

Cíl optimalizace

- Vždy rozvažujte mezi přínosy optimalizace dotazů v porovnáním s jejími náklady
  - Přínosy musí být vyšší, než náklady na její realizaci
- Je nutné si stanovit cíl optimalizace
  - Popište a vyčíslete, čeho přesně chcete dosáhnout
  - Průběžně ověřujte, že se neodchylujete od vašeho cíle
  - Výsledky optimalizace jsou měřitelné, potřebujeme pro porovnání i stav před zahájením optimalizace – *performance baseline*
- Jsou 2 hlavní cíle optimalizace dotazů:
  - Optimalizace chování aplikace
  - Optimalizace celkové zátěže SQL Serveru

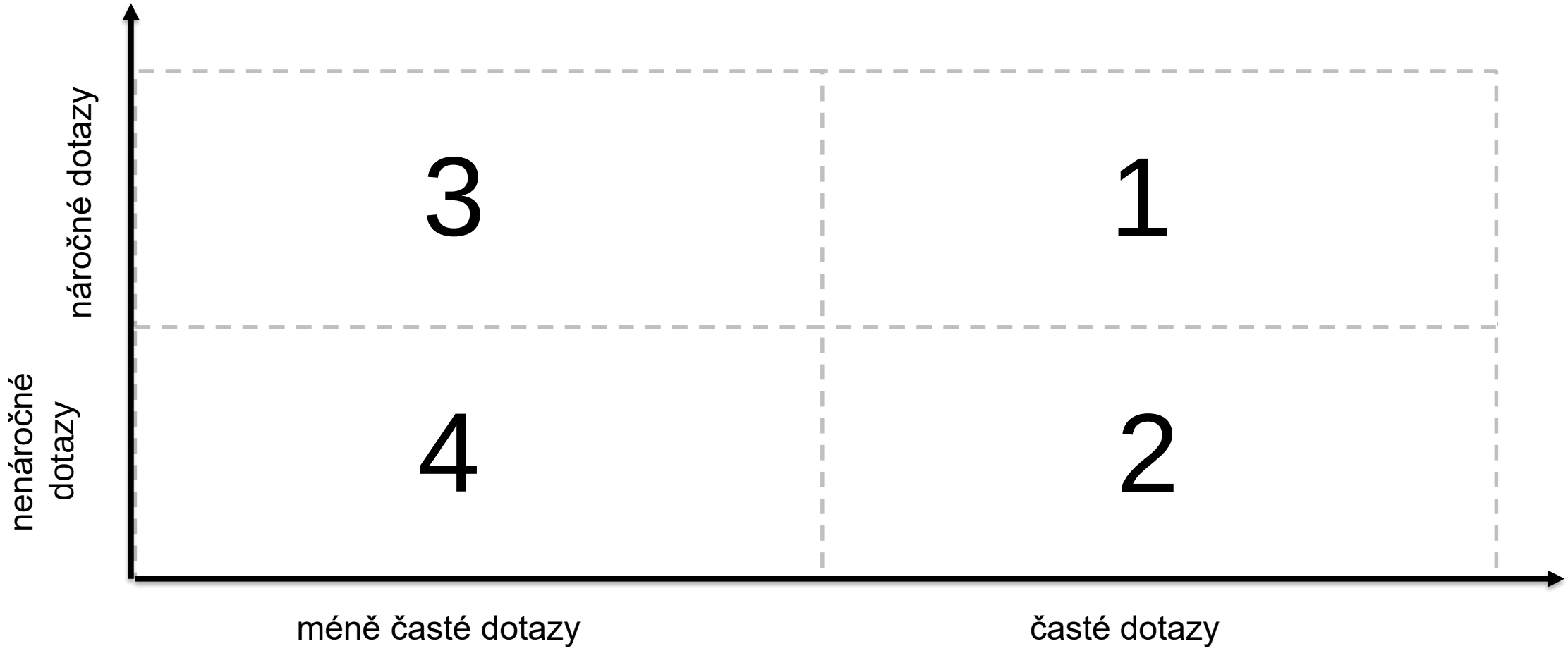
Paretovo pravidlo

- Všeobecně známé **pravidlo 80/20**
- **80% důsledků pramení z 20% příčin**

- **Paretovo pravidlo je aplikovatelné i na proces optimalizace dotazů**

- 20% dotazů konzumuje 80% zdrojů SQL Serveru
  - Je pro nás velmi zajímavé identifikovat nejnáročnější dotazy z pohledu zkonzumovaných zdrojů (Top Resouce Consuming Queries)

Cíl optimalizace



Query Store

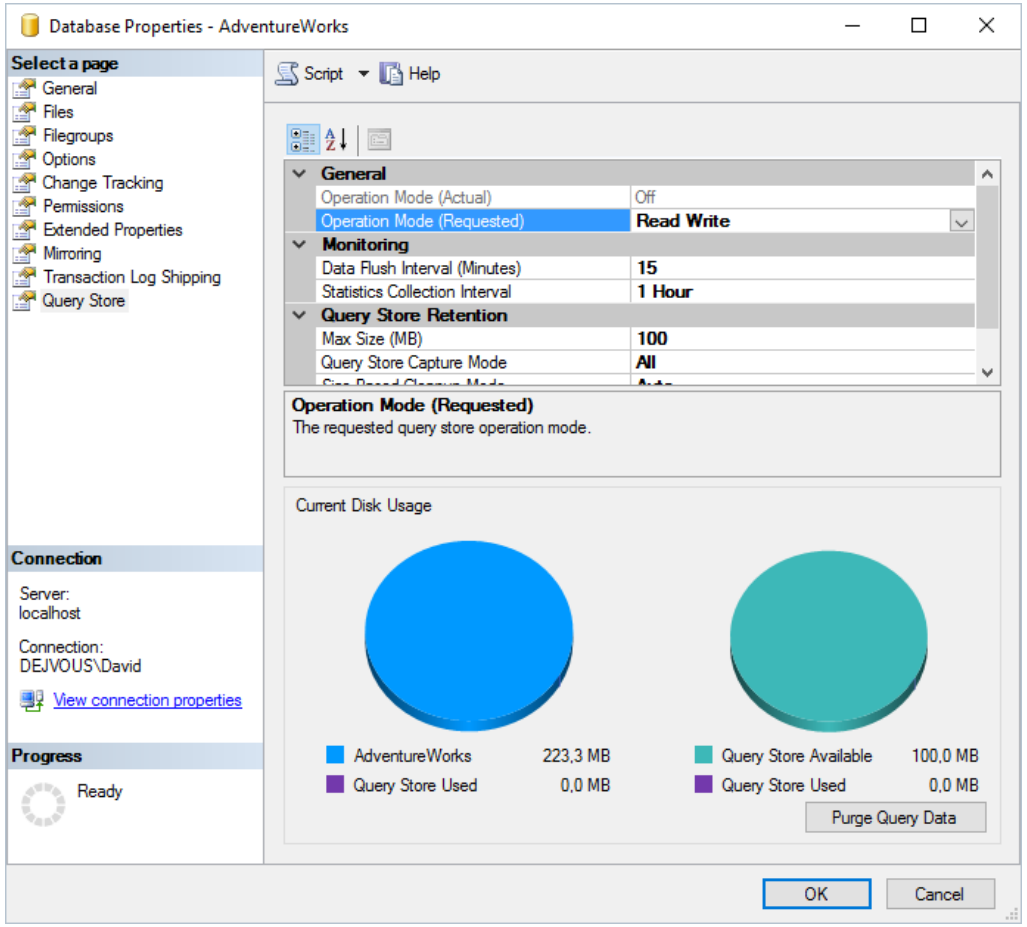
- Nový způsob monitorování náročných dotazů dostupný od SQL Serveru 2016

- Sbírá data o:
  - Exekuci jednotlivých dotazů
  - Používaných variantách exekučních plánů

- Použití:
  - Identifikace náročných často spouštěných dotazů
  - Identifikace dotazů, kde se zhoršila exekuce
  - Porovnání dopadů optimalizace

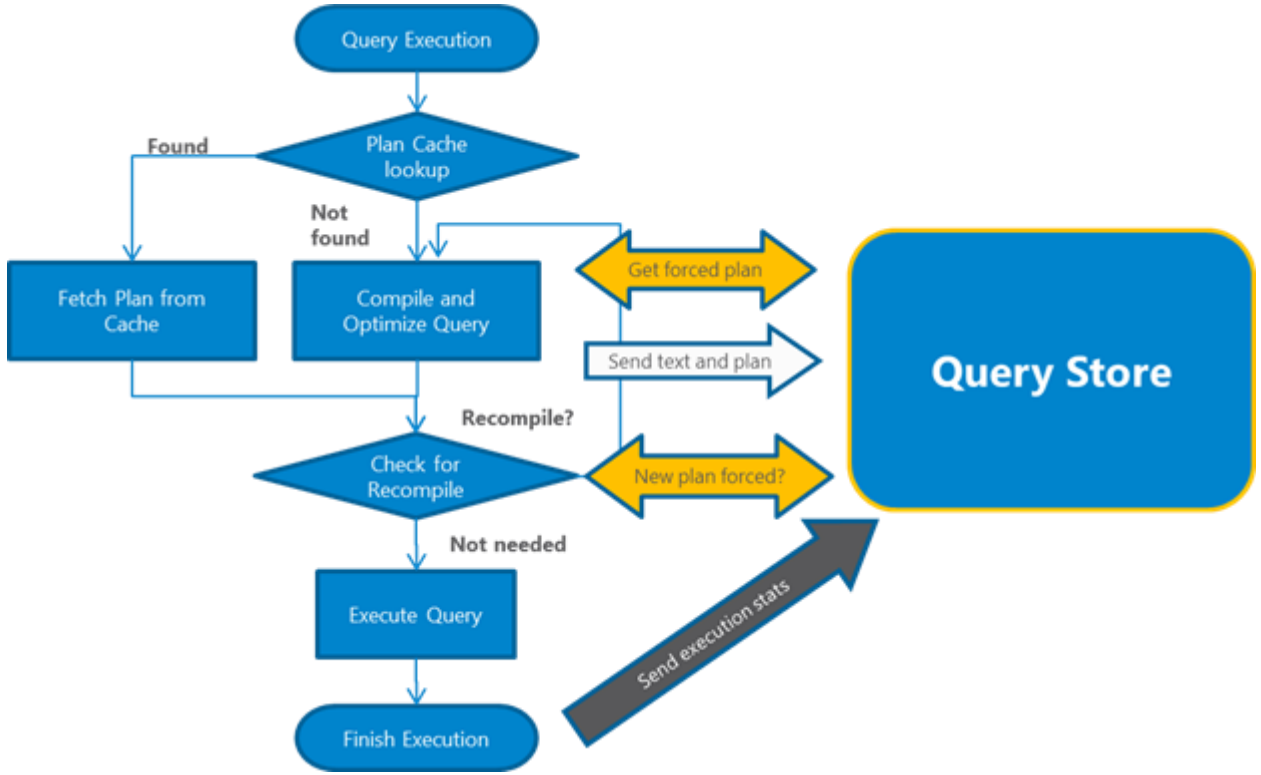
Aktivace Query Store

- Aktivace a konfigurace Query Store se provádí ve vlastnostech databáze



Sběr dat pro Query Store

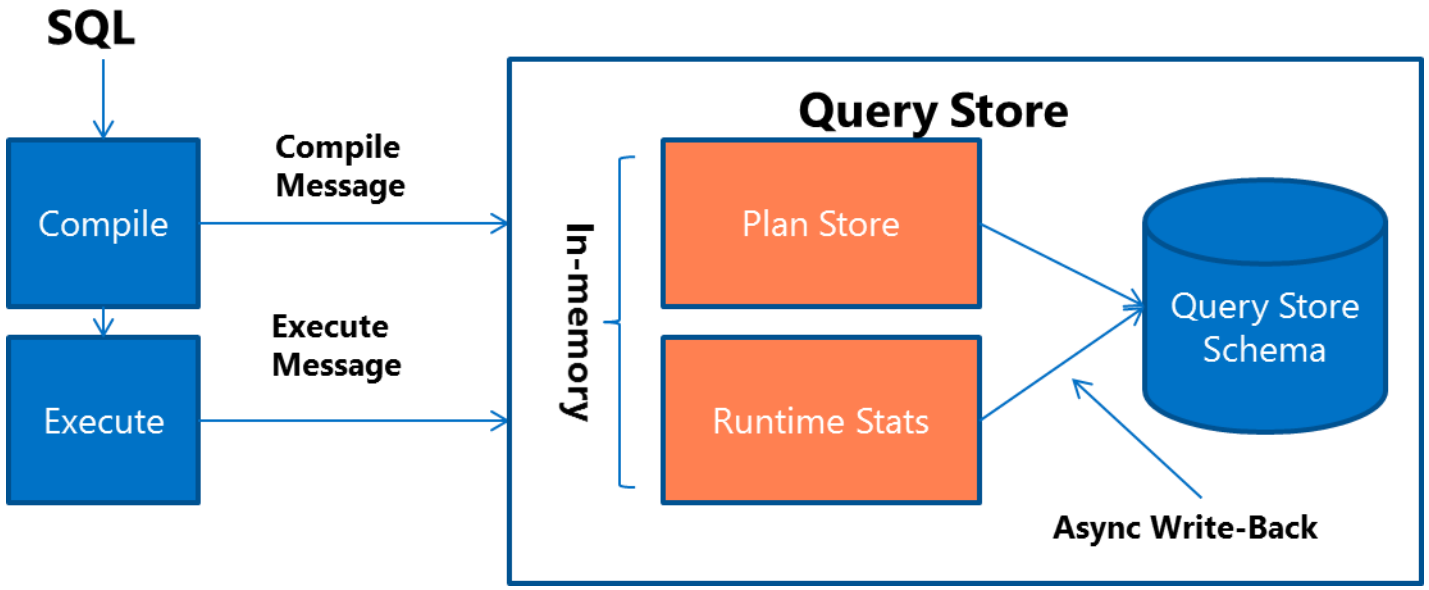
- Query Store po povolení automaticky sbírá data o kompilaci exekučního plánu včetně statistik o jeho exekuci



Zdroj: https://msdn.microsoft.com/en-us/library/mt631173.aspx

Výkonnostní dopad Query Store

- Query Store by mělo mít minimální dopad na výkon SQL Serveru
  - Udává se 1-2%
- Query Store je navrženo tak, aby data cachovalo v operační paměti a dávkově zapisovalo na disk

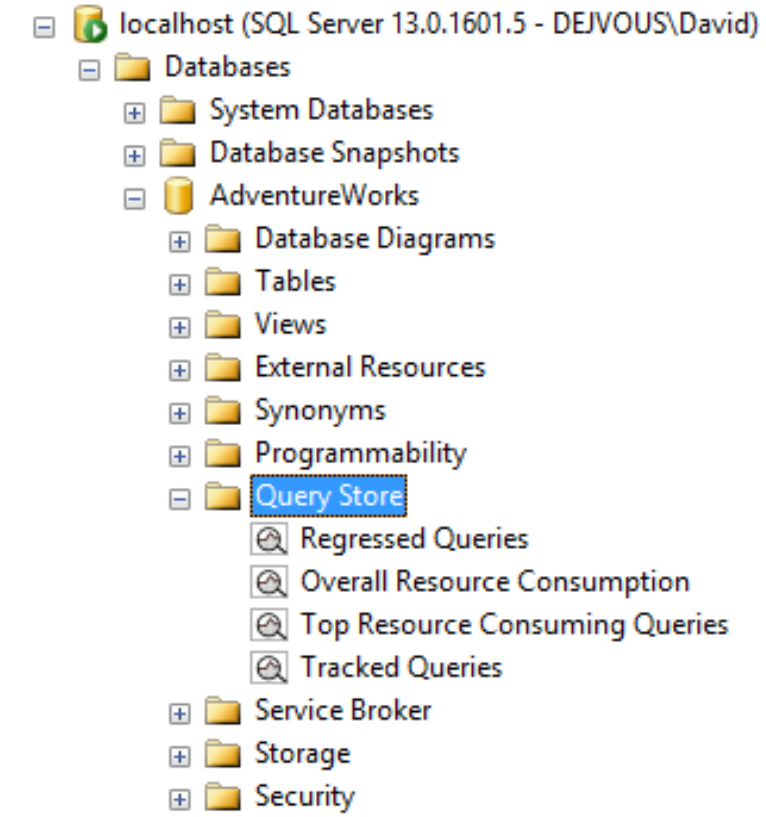


Zdroj: https://msdn.microsoft.com/en-us/library/mt631173.aspx

Vizualizace dat

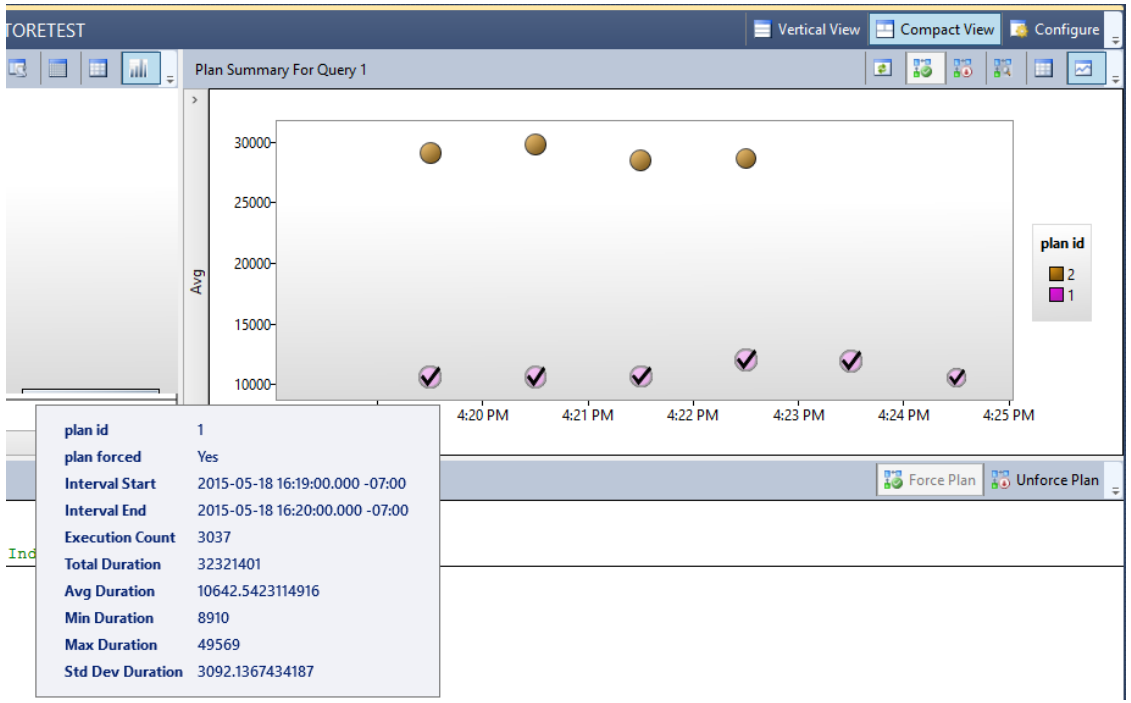
- Query Store přidává do SQL Server Management Studia 2016 novou sekci

- Hlavní pohledy na data:
  - Regressed Queries
  - Overall Resource Consumption
  - Top Resource Consuming Queries
  - Tracked Queries



Použití: Identifikace alternativních plánů

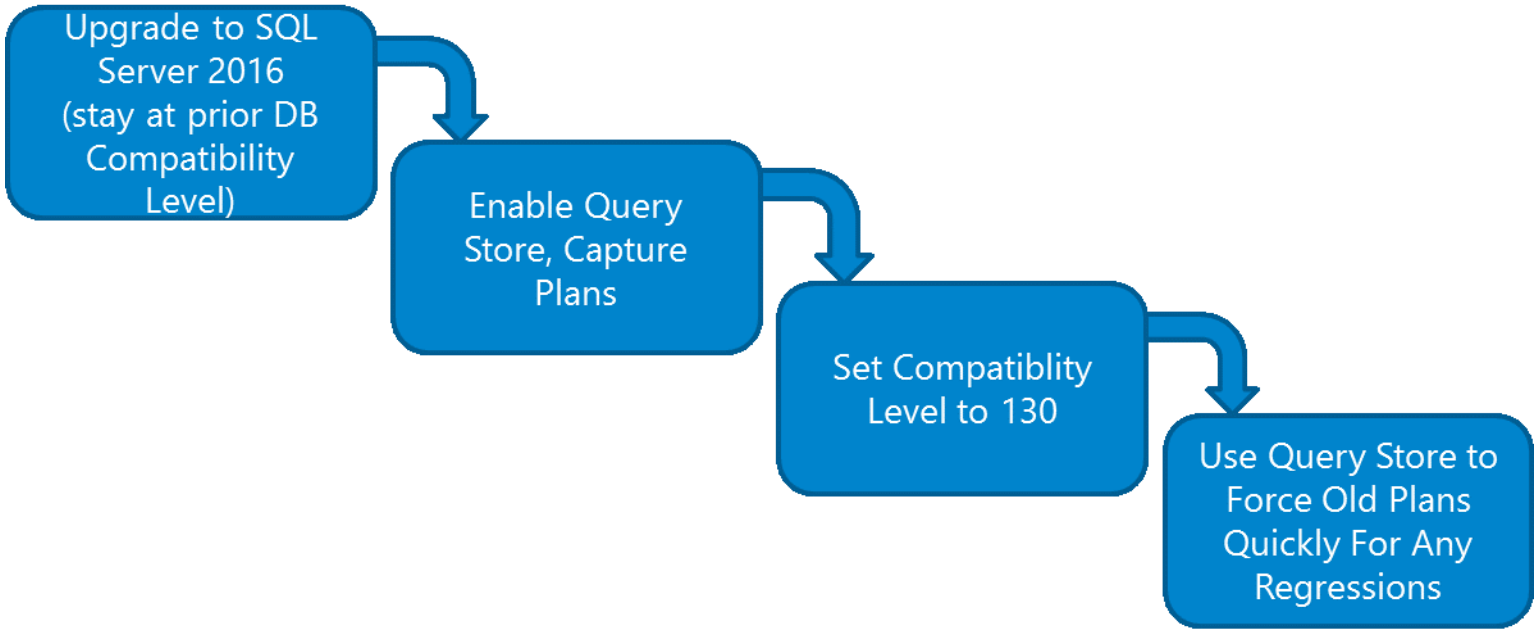
- V současnosti je problém identifikovat různé varianty exekučních plánů stejného dotazu a porovnat jejich efektivitu (plan choice change regression)



Zdroj: https://msdn.microsoft.com/en-us/library/mt614796.aspx

Použití: Spolehlivá migrace na novou verzi

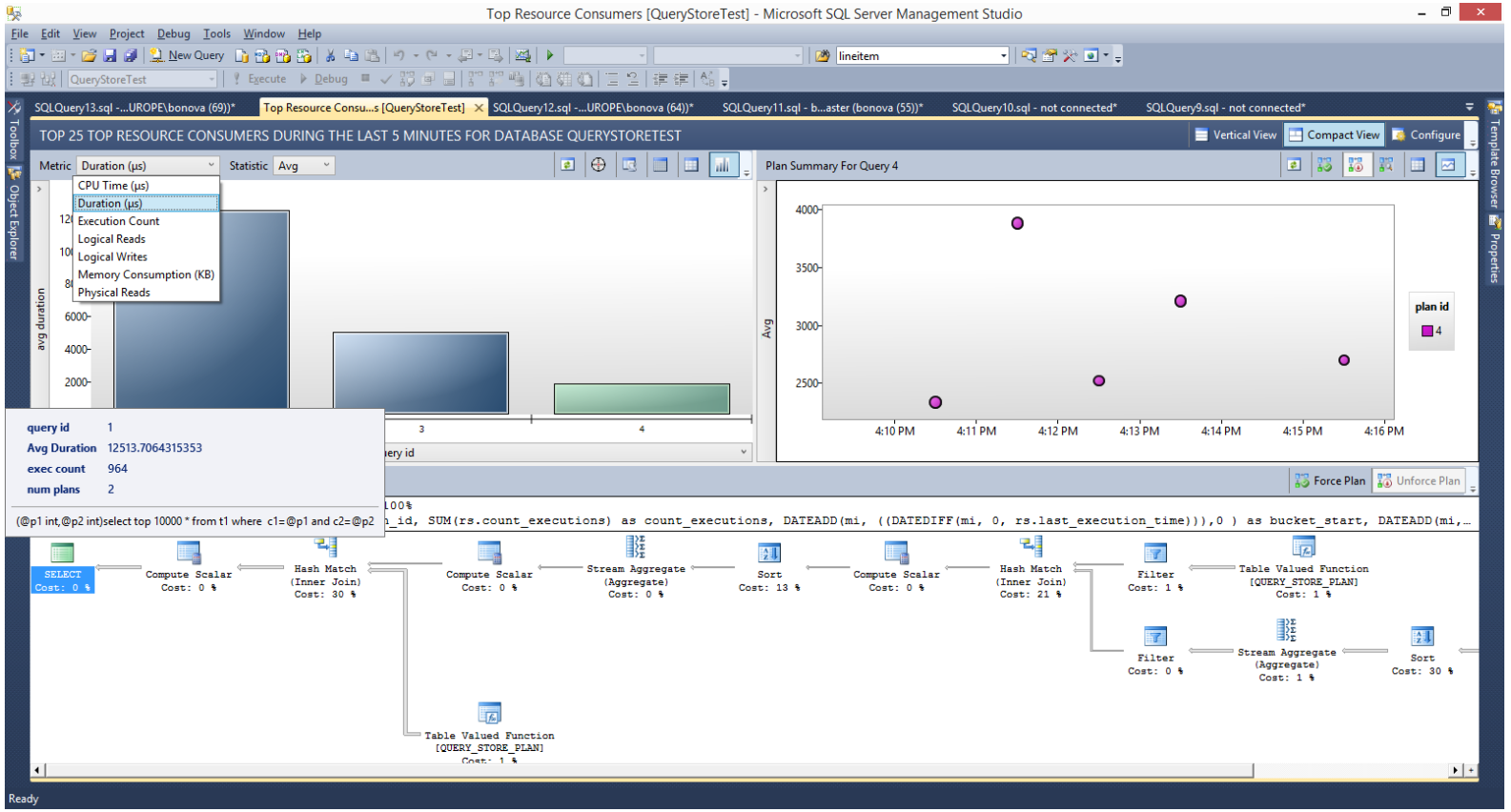
- Při upgradu na novou verzi SQL Serveru může docházet ke generování odlišných exekučních plánů
- Query Store tyto situace umí detekovat a opravit



Zdroj: https://msdn.microsoft.com/en-us/library/mt614796.aspx

Použití: Identifikace nejnáročnějších dotazů

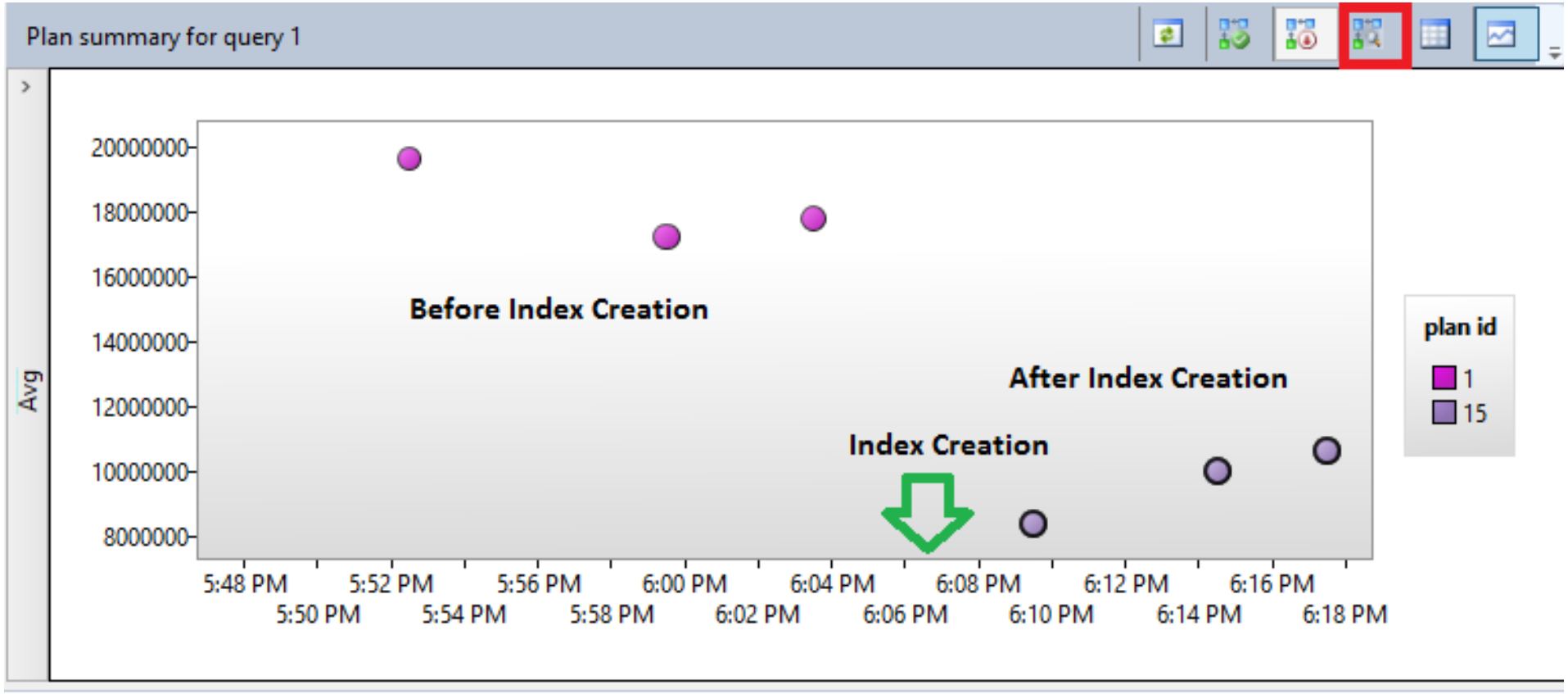
- Díky sběru informací o exekuci dotazů umí Query Store identifikovat nejnáročnější dotazy podle různých kritérií



Zdroj: https://msdn.microsoft.com/en-us/library/mt614796.aspx

Použití: A/B testování

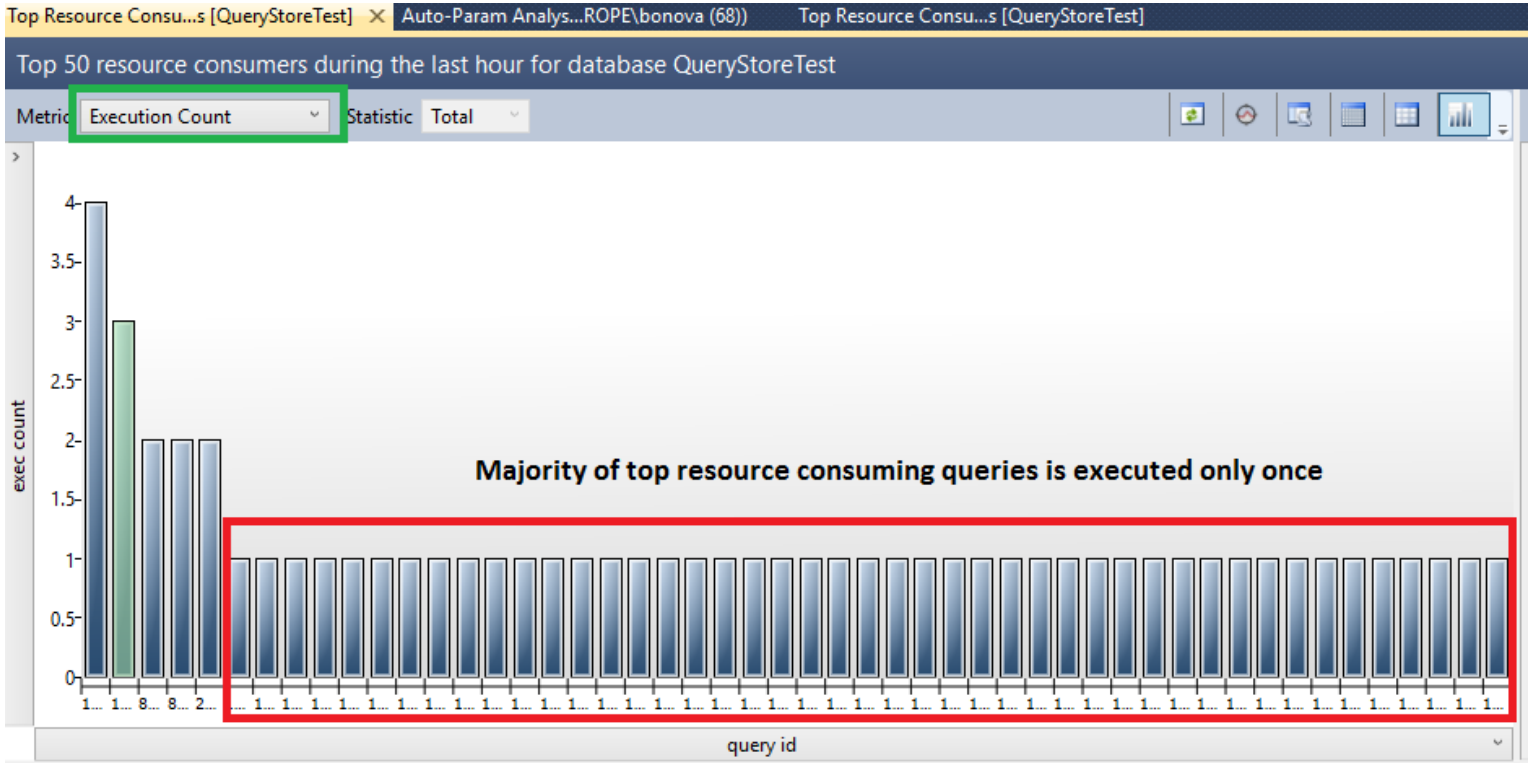
- Po provedení optimalizace je možné měřit a analyzovat její dopad



Zdroj: https://msdn.microsoft.com/en-us/library/mt614796.aspx

Použití: Optimalizace ad-hoc dotazů

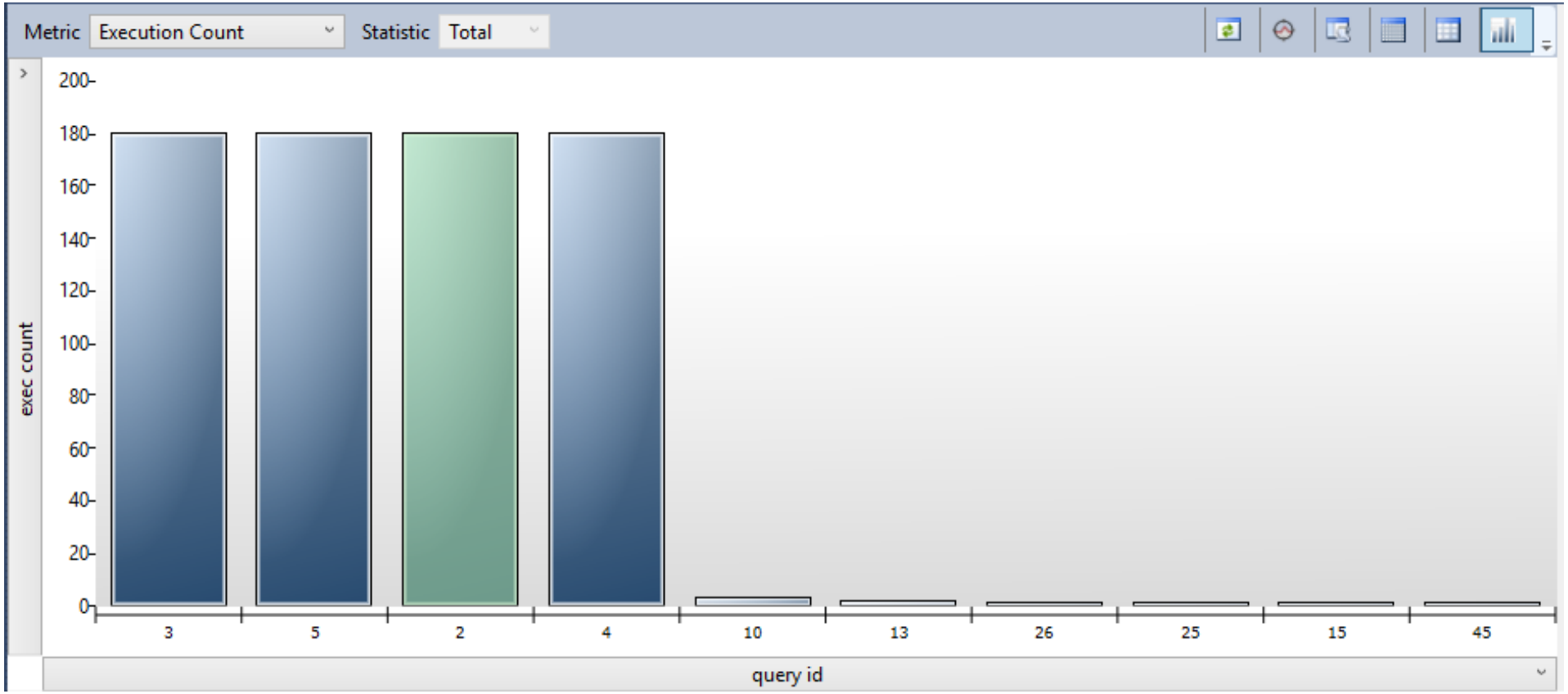
- Identifikace, že je posíláno veliké množství dotazů lišících se hodnotou parametru, ale SQL Server generuje pro každý dotaz individuální exekuční plán



Zdroj: https://msdn.microsoft.com/en-us/library/mt614796.aspx

Použití: Optimalizace ad-hoc dotazů

- Po rekonfiguraci SQL Serveru jsou dotazy správně parametrizovány



Zdroj: https://msdn.microsoft.com/en-us/library/mt614796.aspx

Osnova

1. Aplikace jede pomalu, co s tím?
2. Jak nám pomohou indexy
3. Jak systematicky přistupovat k optimalizaci
4. **Další zabijáci výkonu dotazů**

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

Nevhodný návrh dotazu

- V některých případech složitost dotazu překročí optimalizační schopnosti SQL Serveru a ten sestaví neefektivní exekuční plán
- Pokud nepomohou jiné optimalizační techniky, může být řešením dotaz rozdělit na více menších dotazů
- Pozor na režii spojenou s uložením dat do dočasných tabulek
  - **Pozor na proměnné typu tabulka** – odhadovaný počet řádků je vždy 1

Non-set based operace

- Dotazy v T-SQL popisují jaká data chceme získat, ale nepopisují jak je získat
- Efektivní způsob načtení dat je pak sestaven v rámci optimalizace dotazu
- SQL Server je optimalizovaný na množinové operace
  - Pokud budeme pracovat s daty řádek po řádku, bude to mít významný dopad na výkon dotazu
  - Ne vždy se jde vyhnout kurzorům nebo cyklům, ale měli bychom se o to snažit
  - Pozor na použití skalárních funkcí – jejich jednotlivá volání se vyhodnocují zcela nezávisle

Nadměrné uzamykání a deadlock

- SQL Server pro řízení souběžného přístupu používá mechanizmus zámků
- Kvůli použití zámků dochází k blokování
  - To není problém, pokud netrvá dlouho
- Pokud zpracováváme rozsáhlé transakce a zvolíme nevhodnou úroveň izolace, vzniká velké množství zámků, které blokují další operace až do konce transakce
- Pokud k datům přistupujeme v nevhodném pořadí, může dojít k vzájemnému uváznutí - deadlock

Špatný návrh databáze

- Tabulky v relačních databázích navrhujeme s využitím principů normalizace

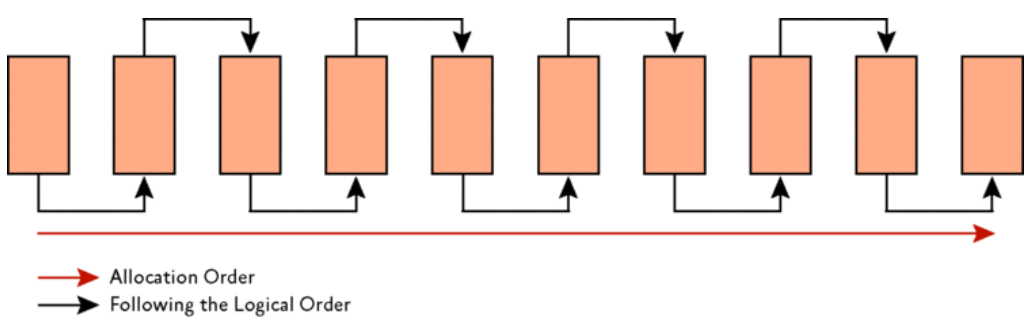
- Výhody normalizované databáze:
  - Odstranění duplicity dat a anomálií při změnách

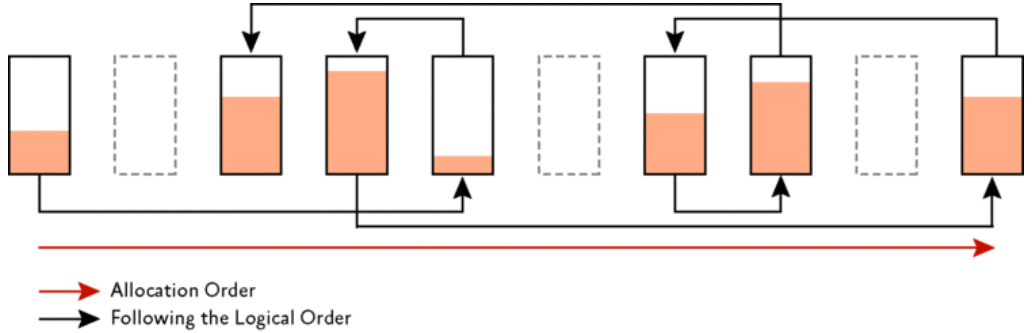
- Další projevy špatného návrhu databáze:
  - Potřeba nadměrně spojovat tabulky (pokusy o dědičnost [1])
  - Nevhodné použití indexů (chybějící indexy, příliš mnoho indexů)
  - Příliš rozsáhlé zámky

Zdroj: [1] https://www.root.cz/clanky/na-co-si-dat-pozor-pri-navrhu-databaze/

Fragmentace indexů

- Modifikace řádků v tabulce způsobuje snížení efektivity použitých indexů z důvodu fragmentace indexů:
  - **Interní fragmentace** – datové stránky nejsou zcela zaplněny
  - **Externí fragmentace** – datové stránky nejsou uložené ve správném logickém pořadí

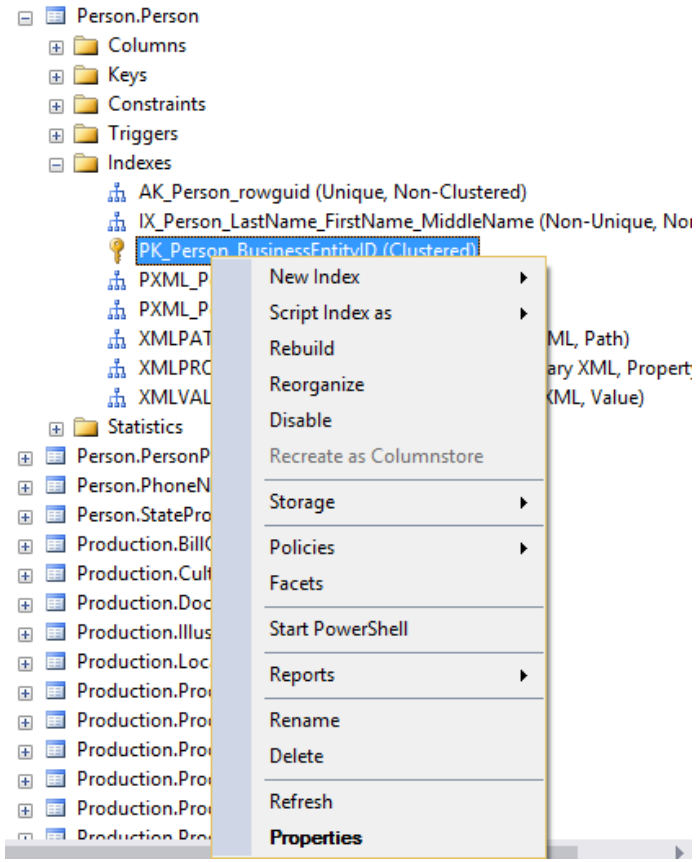


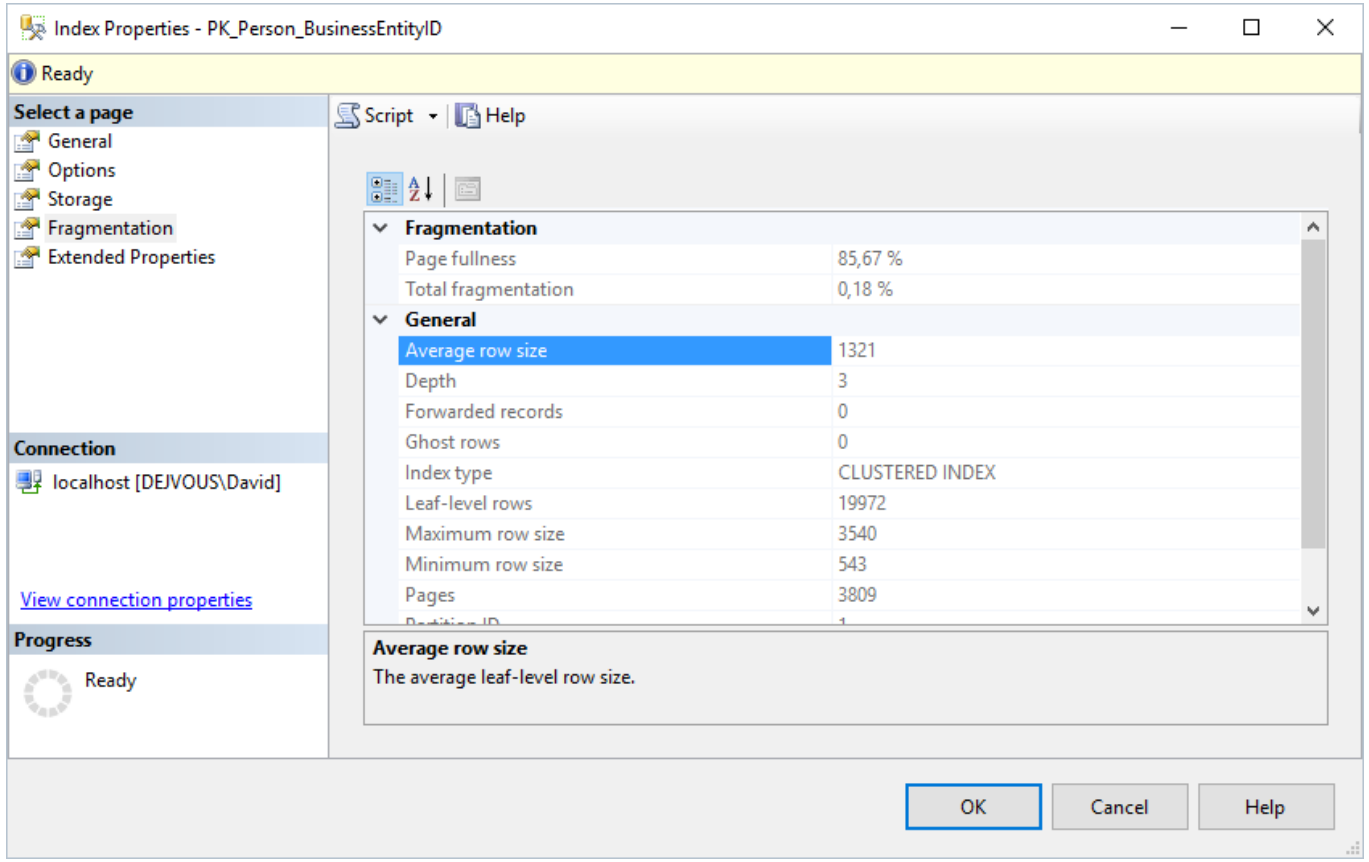


Zdroj: https://technet.microsoft.com/en-us/library/2008.08.database.aspx

Informace o indexech

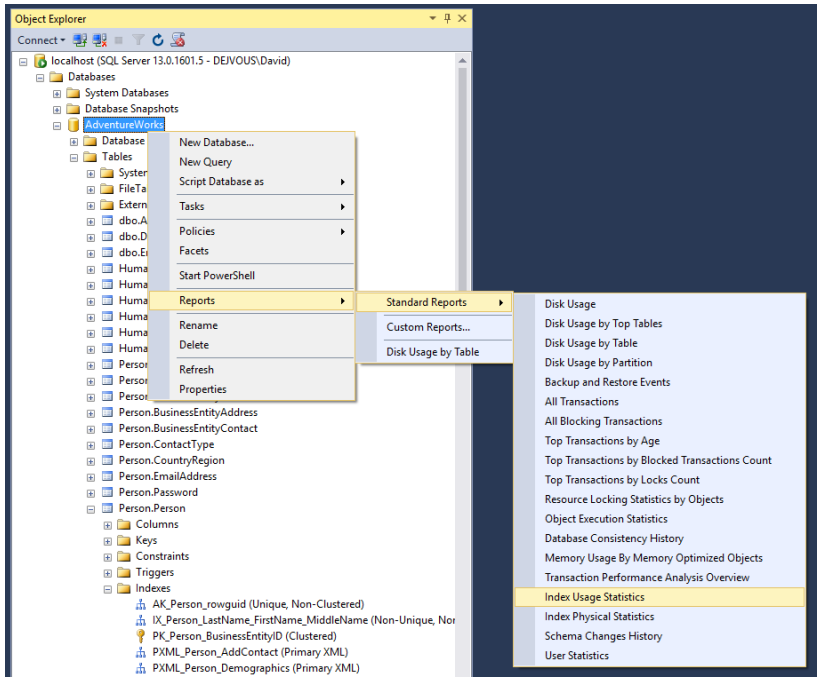
- SQL Server Management Studio
  - Vlastnosti indexu

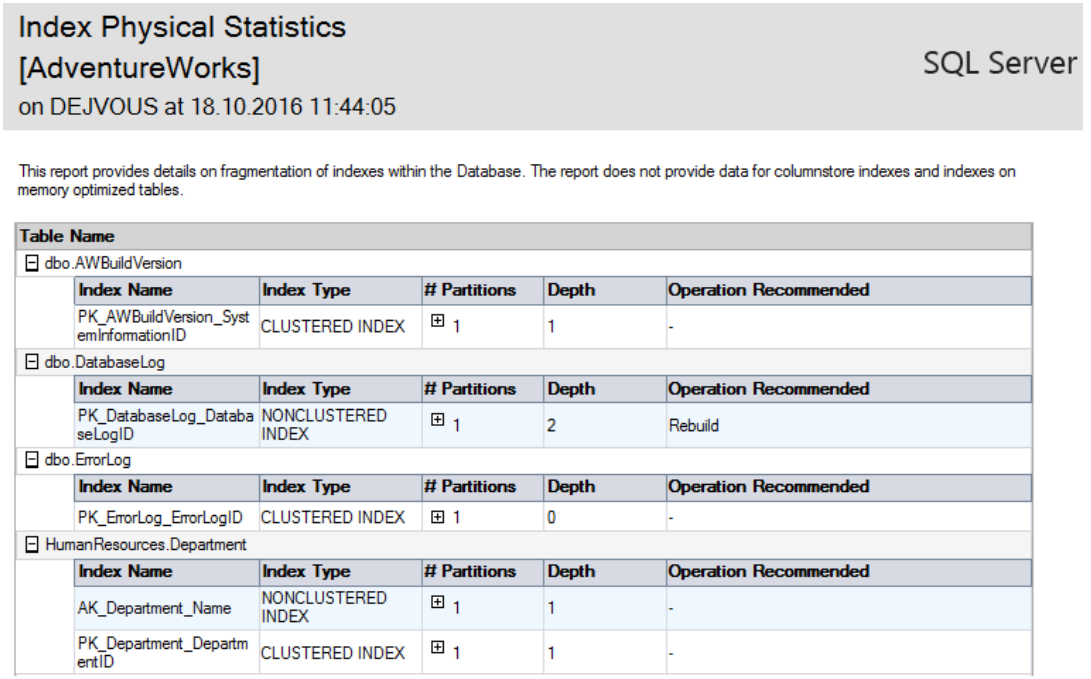




Informace o indexech

- Zabudované reporty v SQL Server Management Studiu
  - **Index Physical Statistics** – Velikost indexu, míra fragmentace, doporučená údržba
  - **Index Usage Statistics** – Statistika operací nad indexem (čtení vs. zápis)





Index Physical Statistics
[AdventureWorks]
on DEJVOUS at 18.10.2016 11:44:05

SQL Server

This report provides details on fragmentation of indexes within the Database. The report does not provide data for columnstore indexes and indexes on memory optimized tables.

Table Name	Index Name	Index Type	# Partitions	Depth	Operation Recommended
dbo.AWBuildVersion	PK_AWBuildVersion_SystemInformationID	CLUSTERED INDEX	1	1	-
dbo.DatabaseLog	PK_DatabaseLog_DatabaseLogID	NONCLUSTERED INDEX	1	2	Rebuild
dbo.ErrorLog	PK_ErrorLog_ErrorLogID	CLUSTERED INDEX	1	0	-
HumanResources.Department	AK_Department_Name	NONCLUSTERED INDEX	1	1	-
	PK_Department_DepartmentID	CLUSTERED INDEX	1	1	-

Informace o indexech

- V obou případech jsou data načítána ze systémových pohledů:
- Indexy a struktura indexů: **`sys.indexes`**
  - https://msdn.microsoft.com/en-us/library/ms173760.aspx

- Fyzické statistiky: **`sys.dm_db_index_physical_stats`**
  - https://msdn.microsoft.com/en-us/library/ms188917.aspx

- Statistiky použití: **`sys.dm_db_index_usage_stats`**
  - https://msdn.microsoft.com/en-us/library/ms188755.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, lze přerušit a pak obnovit

- **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 – **pozor, snižujeme efektivitu indexů pro čtení!**
- 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

Nevhodná konfigurace uživatelských databází

- Důležité je správné rozmístění datových souborů a transakčního logu na disky
- Nastavení RECOVERY MODEL
- Pozor na nastavení AUTO OPTIONS u uživatelských databází:
  - AUTO SHRINK
  - AUTO CLOSE
  - AUTO UPDATE STATISTICS
  - AUTO CREATE STATISTICS

Neoptimální exekuční plány

- SQL Server při sestavování exekučního plánu provádí **cost-based optimalizaci**, kdy vybírá **dostatečně dobrý exekuční plán s nejnižší cenou**
- Jednomu dotazu odpovídá celá řada více či méně efektivních způsobů, jak jej fyzicky provést
- Problém je, že cena různých operací není konstantní a odvíjí se od počtu zpracovaných záznamů
  - Před exekucí dotazu, ale SQL Server nezná např. selektivitu dotazu – proto používá **odhad založený na statistikách, které musí být udržované aktuální**
  - Chyba v predikci kardinality při použití **table valued funkcí** a **table valued proměnných**
  - Problémy s **parameter sniffingem**

Parameter sniffing

- Když SQL Server kompiluje exekuční plán uložené procedury s parametry, použije první sadu parametrů pro sestavení plánu
- Tento plán ale nemusí být efektivní pro ostatní hodnoty parametrů
- Řešení:
  - Rekompilace procedury vždy před spuštěním
    WITH RECOMPILE
  - Query Hint – instrumentace kompilátoru, která hodnota parametru je nejlepší pro sestavení plánu:
    OPTION (OPTIMIZE FOR (@CustomerID=800));
  - Rozdělení procedury na víc procedur

Nízká míra znovoupoužití exekučních plánů

- Sestavení exekučního plánu je velmi náročná výpočetní operace (vysoká cena z pohledu CPU)
- Z toho důvodu se vygenerované exekuční plány ukládají do cache exekučních plánů, aby mohly být znovu použity bez nutnosti opakovaného generování
- Cachují se jak **ad-hoc dotazy**, tak **uložené procedury**
- Problém pro SQL Server může být O/R mapper, který generuje veliké množství podobných ale nikoliv identických ad-hoc dotazů
- Kompilováno by mělo být max. 10% dávek zpracovávaných dotazů

Časté rekompilace exekučních plánů

- V okamžiku, kdy je exekuční plán uložený v cache exekučních plánů neplatný, je odebrán a musí být při dalším spuštění dotazu rekompilován

- Důvody pro rekompilaci mohou být následující:
  - Změna parametrů spojení
  - Změna schématu objektu, na který se dotaz odkazuje
  - Vynucení příkazem

- Rekompilace a kompilace by měly být nejvýše v poměru 1:10
  - Performance counter **SQLServer: SQL Statistics: SQL Compilations/Sec** a **SQLServer: SQL Statistics: SQL Re-Compilations/Sec**

Osnova

1. Aplikace jede pomalu, co s tím?
2. Jak nám pomohou indexy
3. Jak systematicky přistupovat k optimalizaci
4. Další zabijáci výkonu dotazů

Dotazy

RNDr. David Gešvindr

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

david@wug.cz

@gesvindr