# **6.** Optimalizace výkonu T-SQL dotazů

LAB OF SOFTWARE ARCHITECTURES
AND INFORMATION SYSTEMS

FACULTY OF INFORMATICS
MASARYK UNIVERSITY

# Obsah

1. Proces optimalizace dotazů
2. Metody monitorování T-SQL dotazů a serveru
3. Analýza, návrh a strategie použití indexů
4. Exekuční plány

lasaris

# Obsah

1. **Proces optimalizace dotazů**
2. Metody monitorování T-SQL dotazů a serveru
3. Analýza, návrh a strategie použití indexů
4. Exekuční plány

lasaris

# Motivace

- Výkon HW roste, což často vede k mylné představě, že dotazy lze zpracovat „hrubou výpočetní silou“ bez nutnosti optimalizovat

- Přínosy optimalizace dotazů
  - Zvýšení výkonu aplikace (zvýšení propustnosti, zkrácení doby čekání na dokončení operace)
  - Snížení nákladů na provoz – efektivnější využití HW

- I zdánlivě nepatrná změna ve zpracování dotazu může způsobit snížení délky trvání dotazů s minimálními náklady

lasaris

# Proces optimalizace

- Proces optimalizace je **iterativní**:

1. Identifikace úzkého hrdla
2. Odstranění problému
3. Skok na bod 1 – Identifikace nového úzkého hrdla

- Odstranění jednoho úzkého hrdla povede k tomu, že jiná část systému se stane novým úzkým hrdlem

lasaris

# Cíl optimalizace

- Ne všechny operace mohou být efektivně optimalizovány
- Vždy je třeba uvážit poměr mezi **náklady na optimalizaci** a **jejími přínosy**

- **Je třeba si stanovit předem reálné cíle optimalizace a tomu podřídit celý proces optimalizace**
  - Konkrétně popsat, jaké jsou požadavky na výsledky optimalizace
  - Např.: Načtení detailu objednávky musí trvat méně než 1s

lasaris

# Paretovo pravidlo

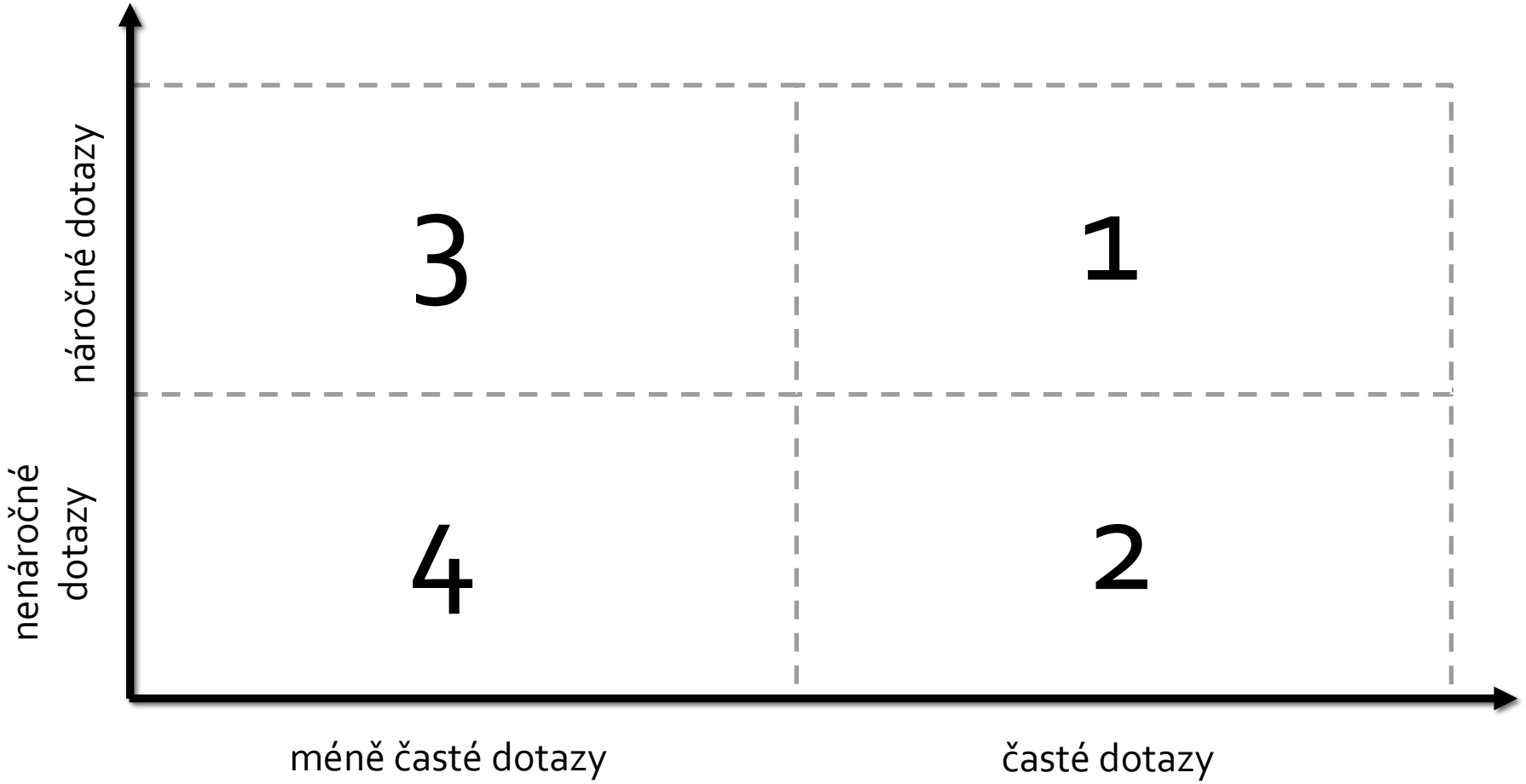
- Všeobecně známé **pravidlo 80/20**
- **80% důsledků pramení z 20% příčin**

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

- Řešením **20% nejnáročnějších zdrojů** zvýšíme výkon aplikace **až o 80%**
- Pro zvýšení výkonu o **zbylých 20%** musíme řešit **zbylých 80% zdrojů**

lasaris

# Cíl optimalizace



# Performance Baseline

- Sada výkonnostních metrik, které popisují systém v běžném stavu
  - Vzory uživatelské zátěže
  - Vzory vytížení SQL Serveru
- Pro vytvoření performance baseline nestačí jedno měření, je vhodné ji založit na dlouhodobém monitoringu systému
- Pomáhá při plánování kapacity
- Slouží jako podklad pro konsolidaci systémů
  - Virtualizace SQL Serverů → méně fyzického HW
  - Snížení nákladů na údržbu, správu i licence

lasaris

# SQL Server Performance Killers

- Indexy a statistiky
  - Chybějící indexy nebo nevhodně zvolené indexy
  - Nepřesné statistiky
  - Fragmentované indexy
- Nevhodně strukturované dotazy a databáze
  - Nevhodný návrh dotazu
  - Non-set based operace (kurzory, iterace, cykly)
  - Nadměrné uzamykání a deadlock
  - Špatný návrh databáze (denormalizace schématu)

lasaris

# SQL Server Performance Killers

- Konfigurace serveru a databází
  - Použití výchozí konfigurace SQL Serveru
  - Neoptimální konfigurace operačního systému Windows
  - Uložení a konfigurace systémových databází
  - Nevhodná konfigurace uživatelských databází
- Exekuční plány
  - Neoptimální exekuční plány
  - Častá rekompilace exekučních plánů
  - Nízká míra znovupoužití exekučních plánů

lasaris

# Chybějící nebo nevhodně zvolené indexy

- Vhodné indexy snižují množství dat, které musí SQL Server načíst a zpracovat pro získání výsledků dotazu
  - Při absenci vhodného indexu musí být zpracováno nadměrné množství dat což vede k problémům s fyzickým IO, pamětí, vytížením CPU nebo zámky

- **Samotná existence indexu nezaručuje, že bude využíván**
  - Indexy musí být navrhovány přesně pro potřeby dotazů
  - Nelze vytvářet indexy bez znalosti rozsahu dat a typů dotazů

lasaris

# Nepřesné statistiky

- SQL Server sestavuje a optimalizuje exekuční plány na základě jejich ceny
  - Cena popisuje náročnost daného exekučního plánu při jeho exekuci z pohledu zdrojů serveru
- K získání stejného výsledku většinou vede více různých cest (exekučních plánů) a liší se právě svojí cenou
  - SQL Server vybere ten s nejnižší cenou
- Statistiky popisují rozložení dat v tabulkách, což ovlivňuje:
  - Použití indexu při zpracování dotazů
  - Zvolené fyzické operace v exekučním plánu

lasaris

# 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

lasaris

# 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

# 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

lasaris

# Špatný návrh databáze

- Tabulky v relačních databázích navrhujeme s využitím principů normalizace

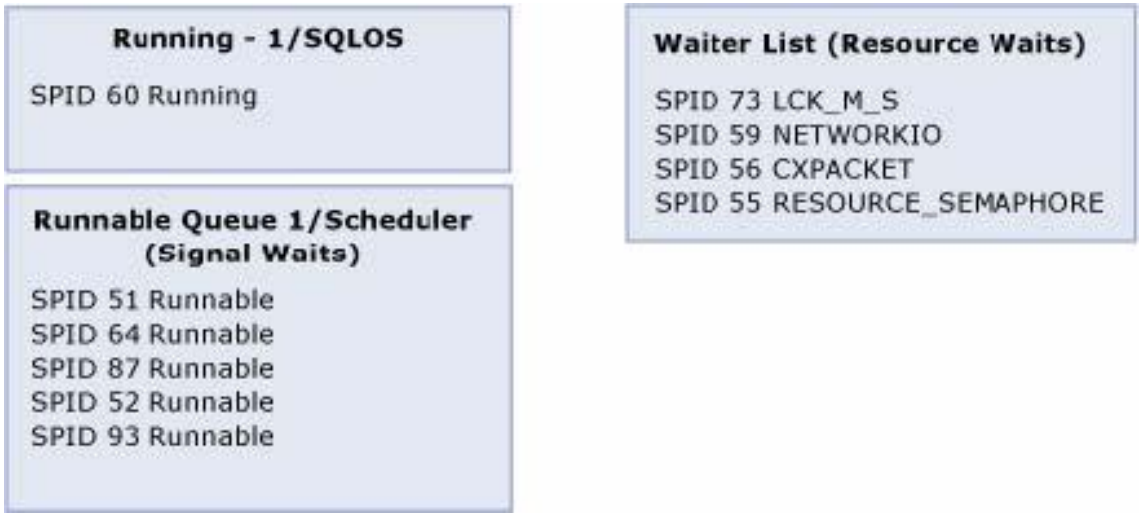
- Výhody normalizované databáze:
  - Odstranění duplicity dat a anomálií při změnách

- Další projevy špatného návrhu databáze:
  - Potřeba nadměrně spojovat tabulky
  - Nevhodné použití indexů
  - Příliš rozsáhlé zámky

lasaris

# Exekuční model

- SQL Server využívá na rozdíl od OS kooperativní plánování
  - Proces se sám vzdá CPU (plánovače) když jej nepotřebuje




  - SQL Server eviduje přes 490 zdrojů, na které procesy čekají
  - Loguje se doba čekání podle zdroje (důležitá metrika)

lasaris

# Obsah

1. Proces optimalizace dotazů
2. **Metody monitorování T-SQL dotazů a serveru**
3. Analýza, návrh a strategie použití indexů
4. Exekuční plány

lasaris

# Metody monitorování 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

lasaris

# SQL Server Profiler

- Grafický nástroj pro **trasování**
  - Zachytávání událostí, které se stanou v SQL Serveru
- Kdy se používá
  - Identifikace náročných dotazů
  - Zachycení zátěže pro SQL Server Tuning Advisor
  - Zachycení zátěže pro „replay“
  - Sběr informací o dalších událostech SQL Serveru (deadlock)
  - Sledování bezpečnostních událostí

lasaris

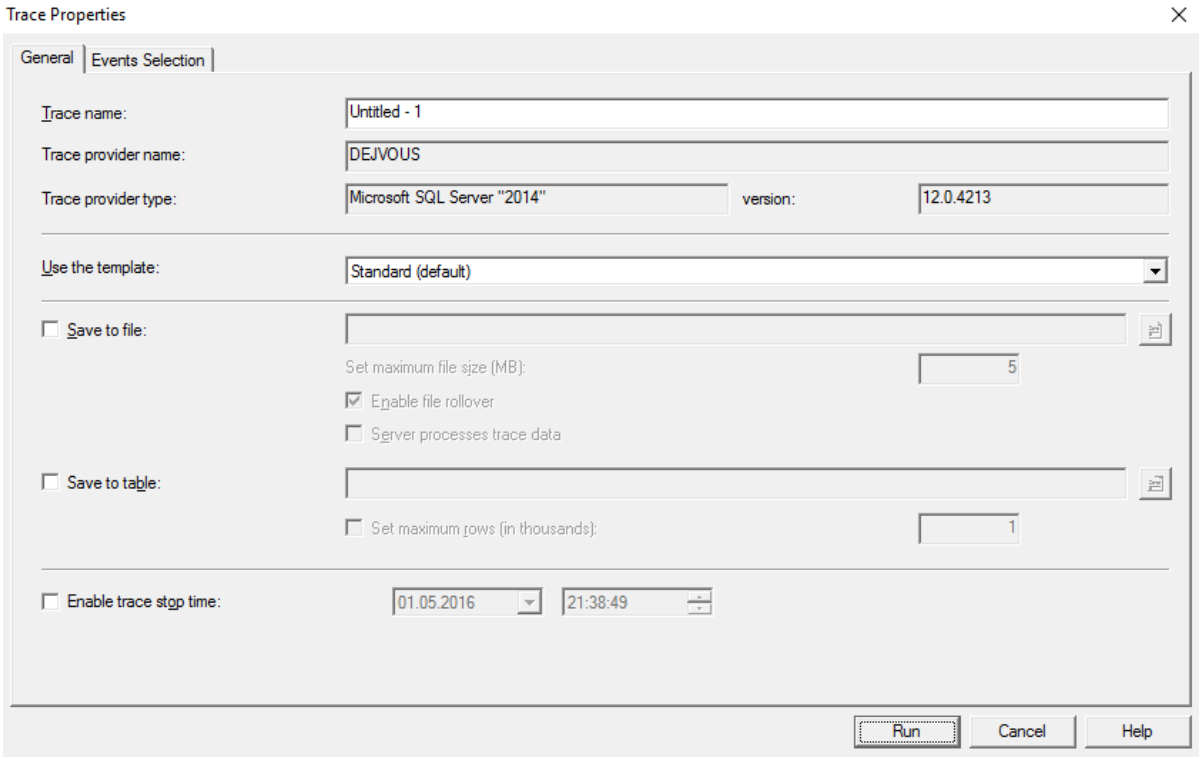
# SQL Server Profiler

- Využívá **SQL Trace Rowset provider**
  - Při vysoké zátěži může dojít ke ztrátě událostí
- Zatěžuje CPU a Temp adresář
- Nástroj určený pro interaktivní sledování
  - Kvůli zvýšené zátěži spíše krátkodobé sledování zátěže
- Pro připojení k SQL Serveru je třeba mít právo **ALTER TRACE**
- Jednoduchá konfigurace
  - Předdefinované šablony pro různé účely
- Možnost zátěž znovu přehrát

lasaris

# SQL Server Profiler

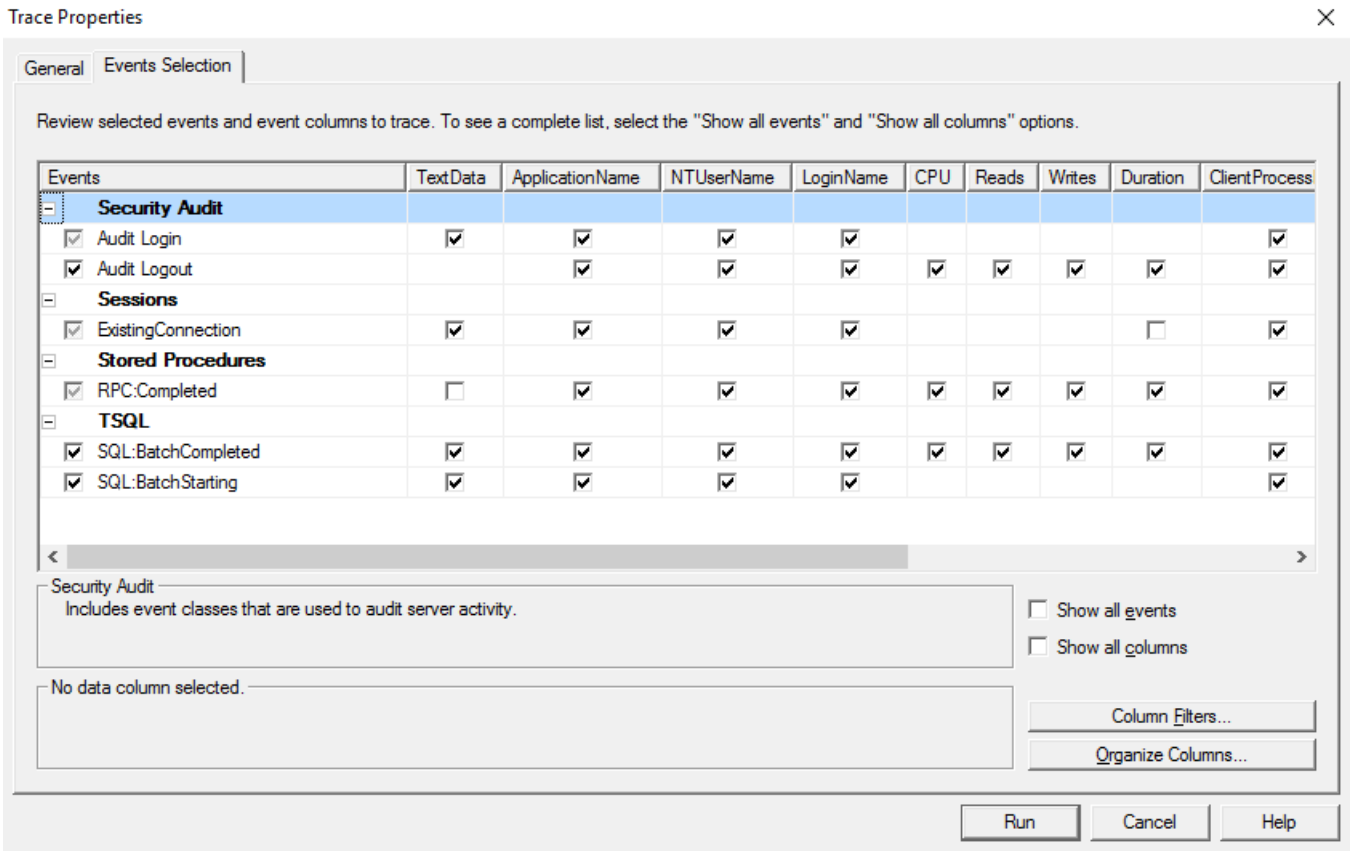
- Po přihlášení k instanci SQL Serveru zvolíme:
  - Šablonu kterou chceme použít
  - Kam ukládat zachycené události (nepoužívat Save to table)




lasaris

# SQL Server Profiler

- Dále je nutné určit, které události v SQL Serveru chceme sledovat




lasaris

# Užitečné události ke sledování

- T-SQL
  - SQL:BatchCompleted
  - SQL:StmtCompleted
- Stored Procedures
  - RPC:Completed
  - SP:Completed
  - SP:StmtCompleted
- Errors and Warnings
  - Blocked process report
- Locks
  - Deadlock graph
- Performance
  - Showplan XML

lasaris

# Doporučení pro SQL Server Profiler

- Omezte množství a rozsah zachytávaných událostí
  - Vybírejte jen potřebné typy událostí
  - Nesledujte události typu Starting
  - Omezte sloupce, které se budou zachytávat
  - Aplikujte filtr a dále omezte množství zachytávaných událostí
  - Nepoužívejte řazení a seskupování v době trasování, ale až později při analýze
- Trasujte vzdáleně
  - Omezíte dopad na naměřené výsledky a snížíte zátěž SQL Serveru

lasaris

# Načtení a analýza zachycení událostí

- Události z trasovacího souboru je možné načíst do SQL Serveru a následně je analyzovat s pomocí SQL skriptů

```
SELECT * INTO dbo.TraceTable
FROM fn_trace_gettable('D:\SQL\Trace.trn', 1)
```

- Následná analýza je díky použití jazyka SQL již jednoduchá

lasaris

# Obsah

1. Proces optimalizace dotazů
2. Metody monitorování T-SQL dotazů a serveru
3. **Analýza, návrh a strategie použití indexů**
4. Exekuční plány

lasaris

# Jak SQL Server ukládá data

- SQL Serveru ukládá data do 8 KB datových stránek
- Typy datových stránek:
  - Global allocation map and shared global allocation map
  - Text or image
  - Data
  - Page free space
  - Differential changed map
  - Bulk changed map
  - Index allocation map
  - Index
- 8 po sobě jdoucích datových stránek tvoří **extent**
  - Mixed extents
  - Uniform extents

lasaris

# Jak jsou uložena data v tabulce

## Heap

- Nemá specifické řazení po sobě jdoucích stránek
- Nemá specifické řazení řádků ve stránkách
- Nové nebo změněné řádky mohou být umístěny kamkoliv do stránek haldy
- IAM udržuje pouze seznam stránek, které tvoří tabulku

## Clustered Index

- Datové stránky tabulky jsou logicky seřazeny
- Řádky jsou seřazeny dle klíče indexu
- Možnost vytvoření pouze jediného clusterovaného indexu
- B+ strom který v listech obsahuje celé řádky

lasaris

# Kdy použít Heap?

- V 99,9% případů se pro uložení dat Heap **nehodí**
- Největší výhodou je rychlé uložení dat

- Časté použití Heap je staging tabulka při přenosech data v rámci Data Warehouse
  - Data jsou rychle uložena nehledě na pořadí
  - Tabulka se pak zase celá čte – nevyhledává se v ní

lasaris

# Clustered Index

- Specifická datová struktura
- Umožňuje nalézt data za použití co nejmenšího úsilí
- Fyzicky se jedná o datovou strukturu **B+ strom**

- **V listech jsou uložené celé řádky tabulky**
- **Je to jediná kopie celých řádků tabulky**

lasaris

# Pravidla použití Clustered Indexů

- Každá tabulka by měla mít clustered index
  - Platí v 99,9% případů
- Špatně zvolený clusterovaný index je horší než žádný
- Clustered index ovlivňuje všechny non-clustered indexy
  - Obsahují clusterovaný klíč jako odkaz na celý záznam
- Odstranění nebo vytvoření Clustered Indexu má fatální dopad na všechny ostatní indexy a způsobí velké množství IO operací
- Nejdříve vytvořte Clustered Index a pak ostatní
  - V opačném případě dojde k řady zbytečných rebuildů

lasaris

# Důležité pojmy

- **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

lasaris

# Důležité pojmy

- **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
- **Index Key Limit**
  - 16 sloupců, 900 B, LOB datové typy

lasaris

# Volba správného klíče pro Clustered Index

- Dobrý clusterovaný klíč indexu by měl mít některé z těchto vlastností
  - Malou datovou velikost
  - Neměnný
  - Vzrůstající (ne nutně monotónně)
  - Unikátní
- Clusterovaný index by měl být vytvořen nad sloupci, se kterými se často provádějí operace jako
  - Vyhledávání rozsahem
  - Řazení

lasaris

# Datový typ clusterovaného klíče

- **Celé číslo**
  - Velmi efektivní, kompaktní (32 nebo 64 bitů)
  - Nezapomenout na jedinečnost a rostoucí posloupnost
- **Text**
  - Málo efektivní, veliký klíč, náročné porovnávání
- **Datum a čas**
  - Pozor na volbu datového typu a jeho velikost
  - Může být problém s jedinečností
- **GUID (uniqueidentifier)**
  - 128 bitové číslo
  - Problém s náhodnými hodnotami

lasaris

# GUID jako clusterovaný klíč

- Porovnání **int** a **uniqueidentifier** na tabulce s 1 milionem řádků:
  - 3.8 MB vs. 15.26 MB
  - Při použití v clustered indexu s 6 non-clustered indexy 22.89 MB vs. 91.55 MB
- Problém s generováním hodnot
  - NEWID()
  - NEWSEQUENTIALID()

| | Writes | Pages | Avg Used | Fragmentation | Row count |
|---|---|---|---|---|---|
| IDENTITY | 1683 | 1667 | 98.9% | 0.7% | 50000 |
| NEWID | 5386 | 2486 | 69.3% | 99.2% | 50000 |
| NEWSEQUENTIALID | 1746 | 1725 | 99.9% | 1.0% | 50000 |

# Non-clustered Index

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

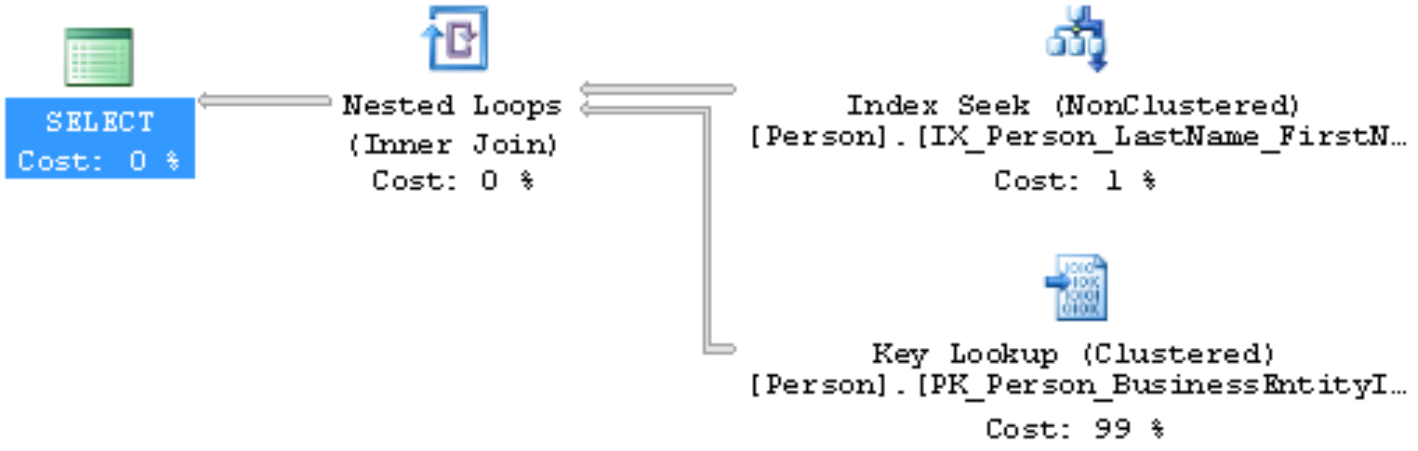
- Indexovací strategie:
  - Provádíme hledání podle vybraného sloupce
  - Načítáme pouze vybrané sloupce
  - Načítáme data seřazená podle daného sloupce

- Pokud požadujeme načtení dat, která nejsou v indexu, musí se načíst i zbytky řádů z clustered indexu nebo heap pomocí operace **lookup**

lasaris

# 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 zjišťuje náročnost hledání v indexu a lookup operace
- V případě vysoké náročnosti index nepoužije

- Pomůcka pro zjištění použitelnosti indexu:
  - (počet datových stánek tabulky) * 0.25 = low tipping point
  - (počet datových stánek tabulky) * 0.33 = high tipping point

- Je-li počet řádků vracených dotazem menší než low tipping point, index se použije
- Je-li počet řádků vracených dotazem větší než high tipping point, index se nepoužije

lasaris

# Tipping Point – příklad 1

- Tabulka
  - Počet stránek : 500.000
  - Počet řádků : 1.000.000

- 500.000 * 0.25 = 125.000 – low tipping point
- 500.000 * 0.33 = 166.000 – high tipping point

- Index se použije pro nalezení záznamů a lookup
  - Dotaz musí vrátit méně než 12.5 %– 16.6% řádků

lasaris

# Tipping Point – příklad 2

- Tabulka
  - Počet stránek : 10.000
  - Počet řádků : 1.000.000

- 10.000 * 0.25 = 2.500 – low tipping point
- 10.000 * 0.33 = 3.333 – high tipping point

- Index se použije pro nalezení záznamů a lookup
  - Dotaz musí vrátit méně než 0.25 % – 0.33 % řádků

lasaris

# Jeden sloupec vs. složený index

- Indexy nejsou vždy vytvořeny nad jediným sloupcem
- Více sloupcový index se nazývá „složený index“ (composite index)
- Výhody složených indexů
  - Možná využitelnost více dotazy
  - Možnost definovat řazení podle každého ze sloupců
  - Dotaz může být založen na více podmínkách
  - Dva sloupce mohou mít vyšší selektivitu jež jediný
  - **Vyvarovat se složenému clusterovanému indexu**

- **Pozor na vnitřní strukturu složeného indexu**

lasaris

# 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**

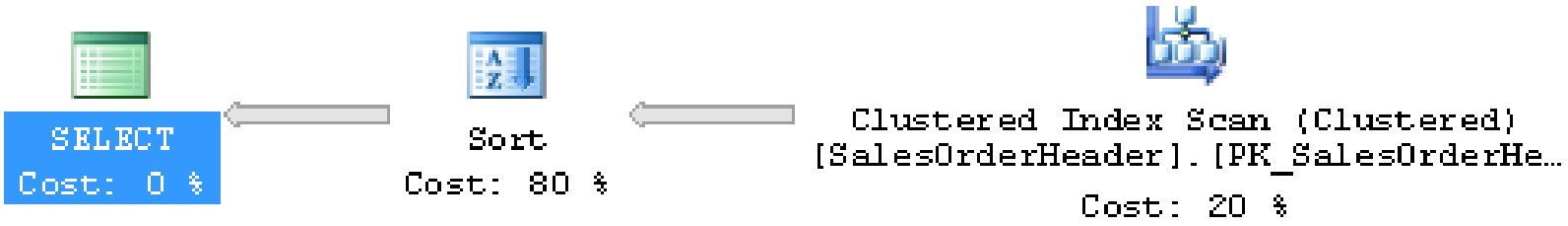
- Snažíme se vytvořit index, který poskytne co nejvíce dat

lasaris

# 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 operaci Sort při zpracování dotazu




lasaris

# Filtrovaný index

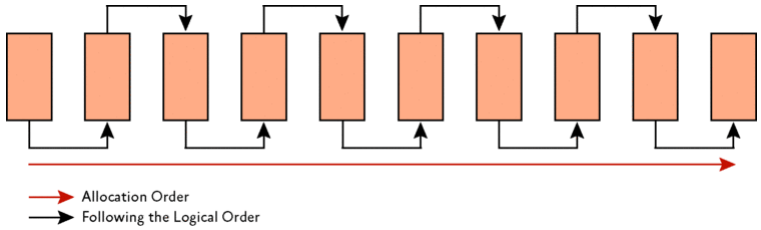
- Do non-clustered indexu je možné přidat predikát, který definuje podmnožinu řádků, které budou indexovány

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

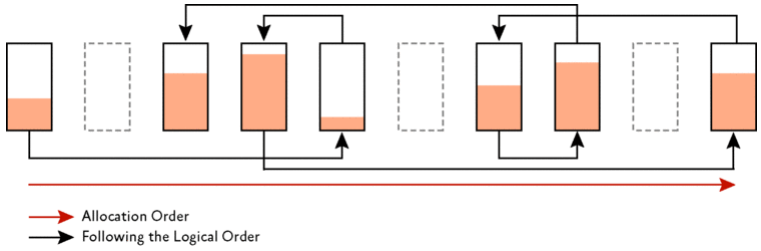
lasaris

# Fragmentace indexů

- Modifikací dat v databázi dochází k fragmentaci indexů
  - Interní fragmentace – datové stránky jsou poloprázdné
  - Externí fragmentace – datové stránky nejsou uloženy souvisle




- Vysoká fragmentace indexů snižuje jejich efektivitu
  - Čte se víc datových stránek, než je nezbytné




lasaris

# Údržba indexů

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

- **Reorganize**
  - Přeskupení dat v datových stránkách
  - 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)

lasaris

# Obsah

1. Proces optimalizace dotazů
2. Metody monitorování T-SQL dotazů a serveru
3. Analýza, návrh a strategie použití indexů
4. **Exekuční plány**

lasaris

# Exekuční plán

- Popis, jak fyzicky vykonat databázový dotaz
- Dotaz v jazyce T-SQL je deklarativní zápis, jaká data chceme získat, ale nepopisuje fyzickou exekuci
- Exekuční plán obsahuje informace o:
  - Typu provedených fyzických operací
  - Jejich pořadí ve kterém se provádí
  - Odhadovaný počet řádků, které budou výstupem jednotlivých operací
  - Odhady ceny dle náročnosti operace

lasaris

# Optimalizace exekučního plánu

- Techniky používané SQL Serverem při optimalizaci dotazu:
  - Optimalizace syntaxe dotazu
  - Nalezení triviálního plánu
  - Použití indexů a strategie spojení na základě statistik
  - Fáze vícenásobné optimalizace dotazu pro dosažení nižší ceny
  - Uložení plánu do cache pro opětovné použití

# Sestavení exekučního plánu

- Sestavení exekučního plánu má následující fáze:
  1. Text T-SQL dotazu je předán **Parseru**
  2. Parser vygeneruje **Parser Tree**
  3. **Algebrizer** sestaví **Query Procesor Tree**
     - DDL příkazy nejsou optimalizovány
  4. **Optimizer** vygeneruje **Execution Plan**
  5. Je zahájena exekuce dotazu

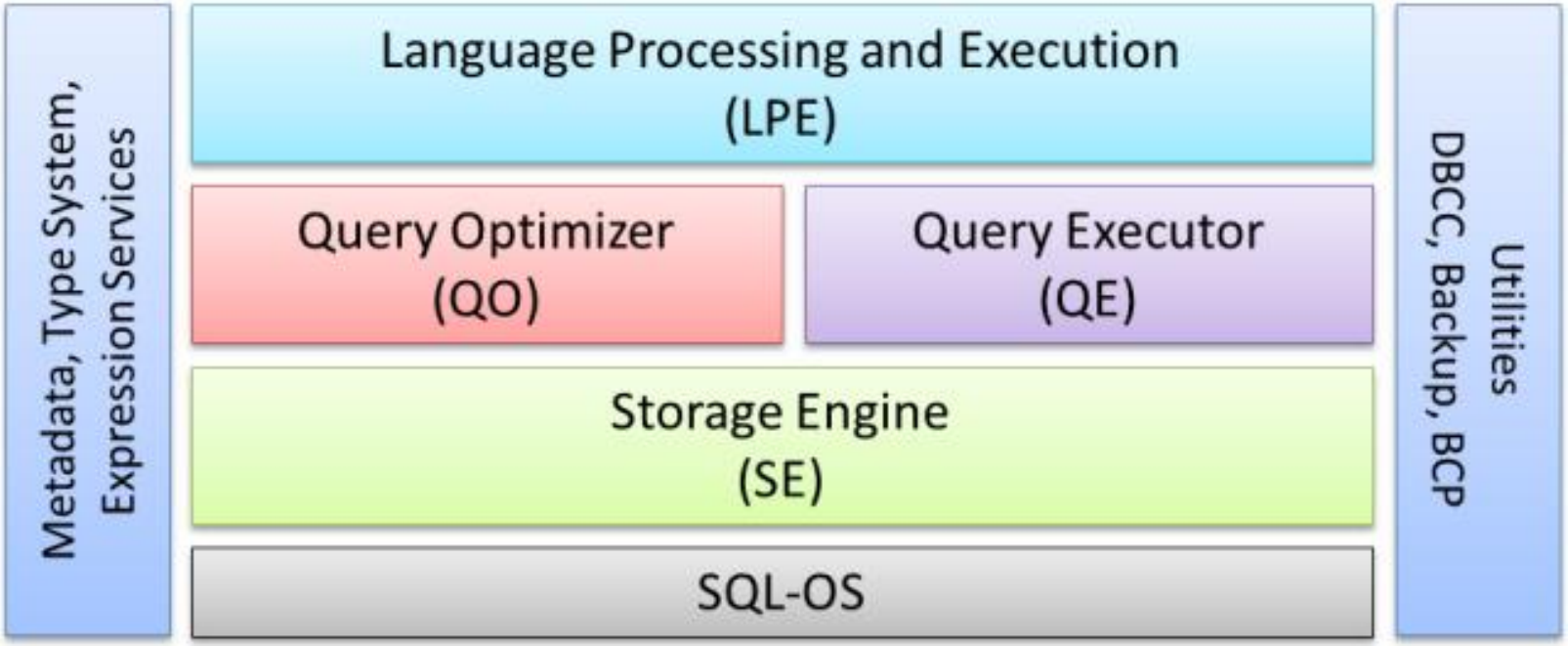
lasaris

# Sestavení exekučního plánu

- Sestavení exekučního plánu má následující fáze:
  1. Text T-SQL dotazu je předán **Parseru**
  2. Parser vygeneruje **Parser Tree**
  3. **Algebrizer** sestaví **Query Procesor Tree**
     - DDL příkazy nejsou optimalizovány
  4. **Optimizer** vygeneruje **Execution Plan**
  5. Je zahájena exekuce dotazu

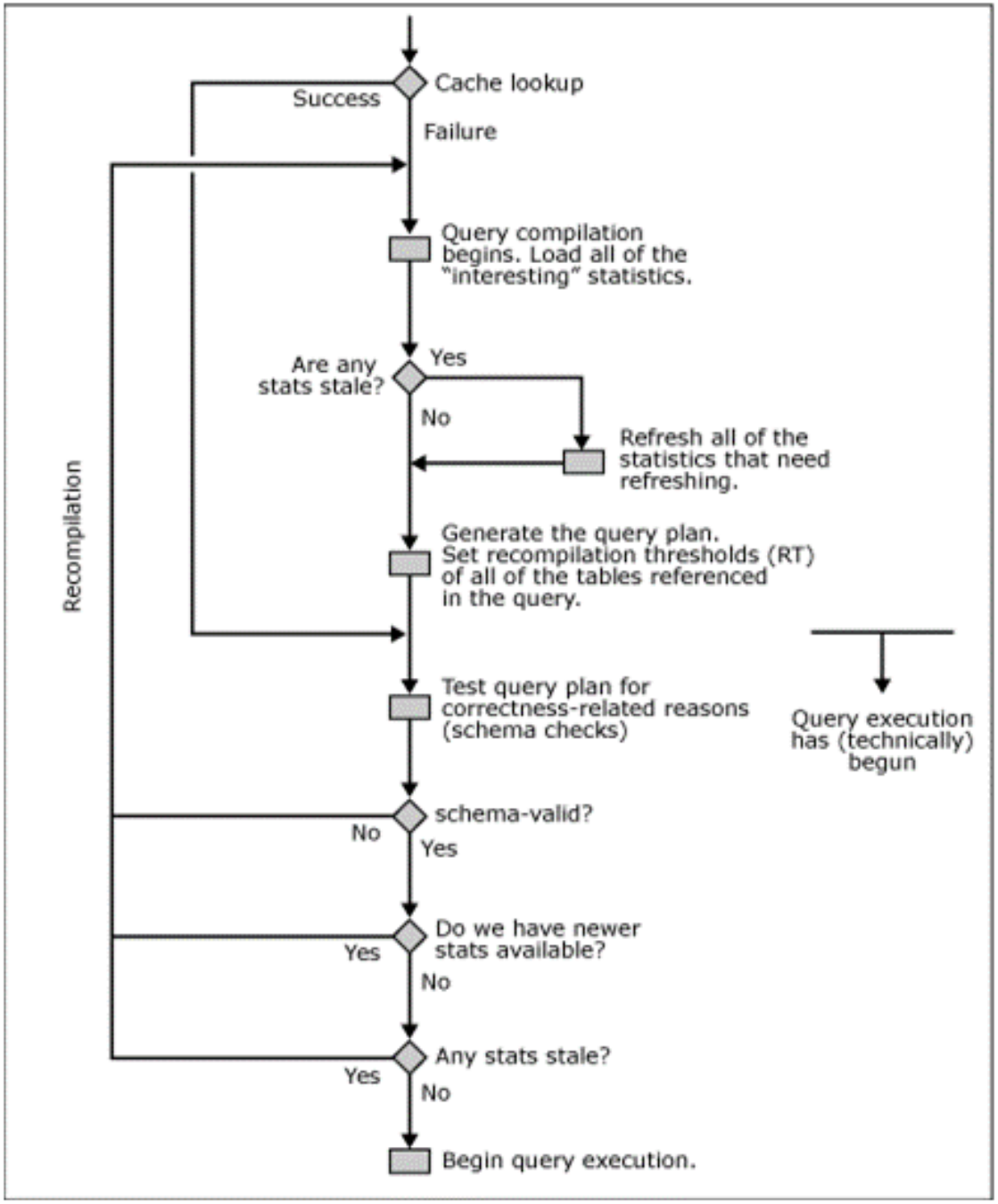
lasaris

# Zpracování dotazů v SQL Serveru





lasaris


Cache lookup
Success
Failure
Query compilation begins. Load all of the "interesting" statistics.
Are any stats stale?
Yes
No
Refresh all of the statistics that need refreshing.
Generate the query plan. Set recompilation thresholds (RT) of all of the tables referenced in the query.
Test query plan for correctness-related reasons (schema checks)
Query execution has (technically) begun
schema-valid?
No
Yes
Do we have newer stats available?
Yes
No
Any stats stale?
Yes
No
Begin query execution.
Recompilation

# Parser

- Kontroluje syntaxi zaslaného dotazu
- Pokud je nalezena chyba v syntaxi, je ukončeno zpracování celé dávky

```
SELEKT * FROM Person.Person
```

- Výsledkem je Parser Tree, který obsahuje převedený text dotazu na tokeny, předané k dalšímu zpracování

lasaris

# Algebrizer

- Vyhodnotí všechny názvy objektů
  - Zjištění skutečných názvů objektů při použití synonym
  - Použití tabulek, sloupců a jejich datových typů
  - Identifikuje použití agregace a seskupení
- Může provést jednoduchou optimalizaci dotazu
  - Např. nahrazení operátoru BETWEEN operátory >= a <=
  - Identifikace implicitní konverze
- Výstupem je **Query Processor Tree**

lasaris

# Optimizer

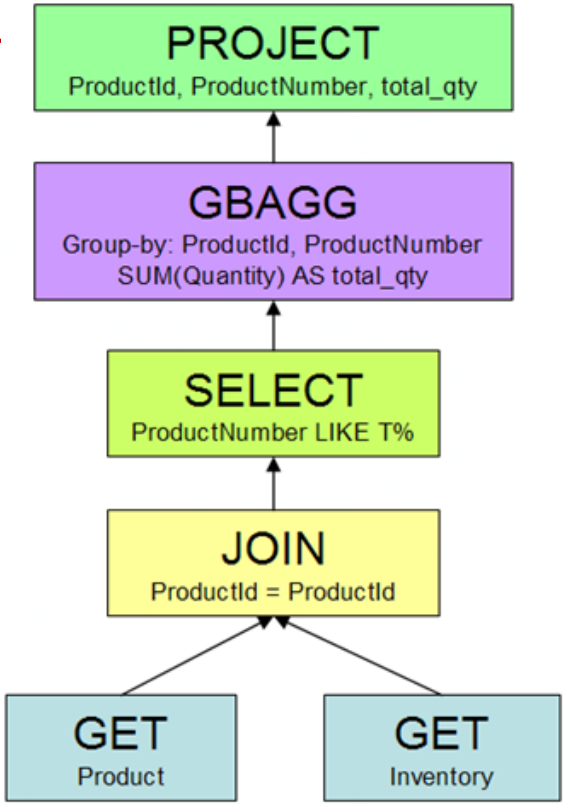
- Parser a algebrizer vytvoří strom logických operací, které je nutné provést
- Pro vykonání dotazu je nutné převést logické operace na fyzické a provést optimalizaci
- Query Optimizer vytvoří exekuční plán, který je použit pro vykonání dotazu

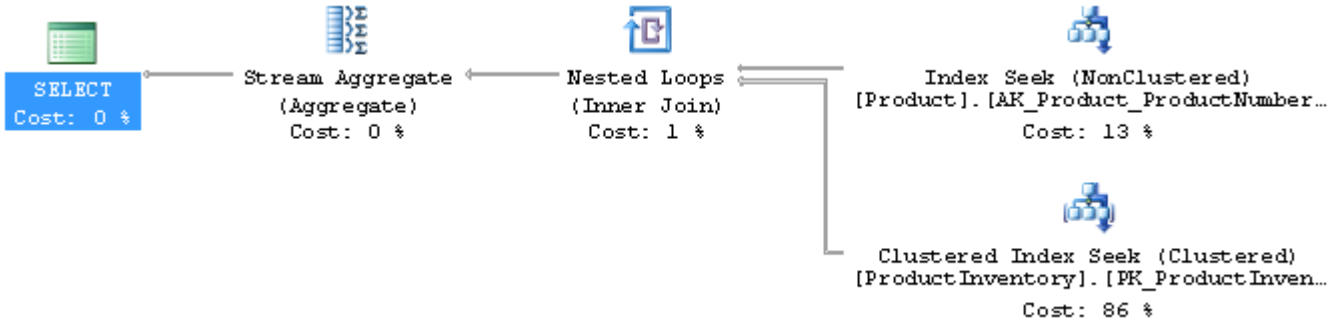
- Cílem není nalézt úplně nejlepší plán, ale pouze dostatečně dobrý plán

lasaris

# Fáze zpracování dotazu

```
SELECT P.ProductNumber,P.ProductID,
       SUM(I.Quantity) AS TotalQuantity
FROM Production.Product P
  INNER JOIN Production.ProductInventory AS I
    ON I.ProductID = P.ProductID
WHERE P.ProductNumber LIKE N'T%'
GROUP BY P.ProductID, P.ProductNumber;
```



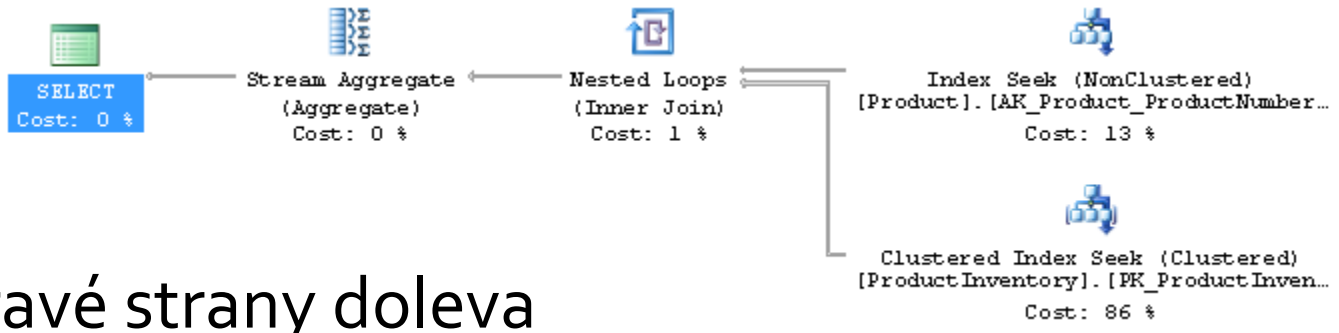




lasaris

# Čtení exekučního plánu

- Exekuční plán lze číst 2 způsoby:
  - Jednoduchý ale nesprávný (příliš zjednodušený)
  - Správný způsob (využití iterátorů)

- Jednoduše:




- Z pravé strany doleva
  - Shora dolů, pokud jsou operace nad sebou
- Šipky označují směr toku dat

lasaris

# Čtení exekučního plánu

**Clustered Index Seek (Clustered)**

Scanning a particular range of rows from a clustered index.

| | |
|---|---|
| **Physical Operation** | Clustered Index Seek |
| **Logical Operation** | Clustered Index Seek |
| **Actual Execution Mode** | Row |
| **Estimated Execution Mode** | Row |
| **Storage** | RowStore |
| **Actual Number of Rows** | 27 |
| **Actual Number of Batches** | 0 |
| **Estimated Operator Cost** | 0,0214026 (86%) |
| **Estimated I/O Cost** | 0,003125 |
| **Estimated CPU Cost** | 0,0001597 |
| **Estimated Subtree Cost** | 0,0214026 |
| **Number of Executions** | 15 |
| **Estimated Number of Executions** | 14,4 |
| **Estimated Number of Rows** | 2,47454 |
| **Estimated Row Size** | 9 B |
| **Actual Rebinds** | 0 |
| **Actual Rewinds** | 0 |
| **Ordered** | True |
| **Node ID** | 3 |

**Object**
[AdventureWorks2012].[Production].[ProductInventory].[PK_ProductInventory_ProductID_LocationID] [I]

**Output List**
[AdventureWorks2012].[Production].[ProductInventory].Quantity

**Seek Predicates**
Seek Keys[1]: Prefix: [AdventureWorks2012].[Production].[ProductInventory].ProductID = Scalar Operator ([AdventureWorks2012].[Production].[Product].[ProductID] as [P].[ProductID])

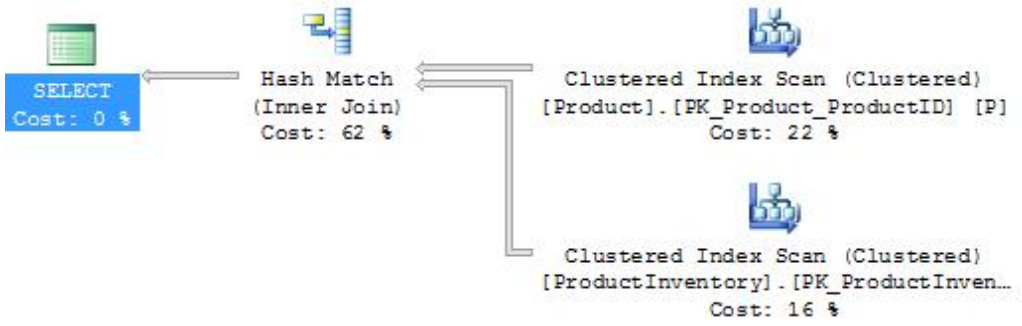
- Každá operace obsahuje detailní popis
  - Typ operace
  - Odhadované a skutečně zpracované množství dat
  - Cena z pohledu CPU a IO
  - Cena všech předcházejících operací

lasaris

# Správné čtení exekučního plánu

- Každý iterátor má 3 metody:
  - Open
  - GetNext / GetRow
  - Close

- Root iterátor volá metody svých přímých potomků, kteří dále volají metody svých potomků…




lasaris

# Příklad správného čtení exekučního plánu

Open metoda root iterátoru Hash Join :

1. Hash Join volá metodu **Open** build potomka (horní scan)
2. Open metoda horního Scan iterátoru se připojí do Storage Engine (SE) a je připravena na získání dat z tabulky. Vykonávání se vrací do Hash Join.
3. Hash Join opakovaně volá **GetNext** metodu scan iterátoru.
4. Každé volání **GetNext** vrací jeden řádek do Hash Join. Tyto řádky jsou použity pro tvorbu hash tabulky.
5. Pokud jsou vráceny všechny řádky, Hash Join volá **Close** metodu build iterátoru scan (čištění a uzavření spojení do SE)
6. Hash Join volá **Open** probe iterátoru (dolní scan)
7. Probe iterátor otevírá připojení do SE a vrací kontolu Hash Join
8. Hash Join ukončuje svoji Open metodu a vrací kontrolu do Execution Engine

lasaris

# Příklad správného čtení exekučního plánu

Execution Engine volá metodu **GetNext** root iterátoru Hash Join:

1. Hash Join volá metodu **GetNext** probe potomka (dolní scan)
2. Scan vrací jeden řádek z tabulky iterátoru Hash Join
3. Hash Join provádí spojení s použitím hash tabulky
4. Pokud řádek není spojen, pokračuje bodem 1
5. Pokud řádek je spojen, vrací řádek do Execution Engine

- Celý proces se opakuje, dokud probe potomek (dolcí scan) vrací záznam z tabulky.
- Pokud probe potomek nemá další záznam, Execution Engine ví, že dotaz byl dokončen a zavolá Close metodu Hash Join.

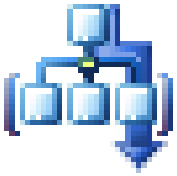
lasaris

| Iterator | Open metoda | GetNext Metoda | Close Metoda |
|---|---|---|---|
| **Scan/Seek** | Otevře připojení do Storage Engine | Získá nový řádek z tabulky/indexu | Ukončí připojení do Storage Engine |
| **Filter** | Použije metodu Open potomka | Volá metodu GetNext potomka dokud řádek nesplňuje podmínku. Pak vrátí řádek. | Uzavře potomka. |
| **Sort** | Vytvoří worktable. Otevře potomka a volá jeho GetNext pro získání všech dat. Uzavře potomka a seřadí všechna data. | Vrací řádek dle řazení. | Odstraní worktable |
| **Hash Join** | Otevře build vstup a získá všechna data GetNext. Uzavře build, vytvoří hash table a otevře probe vstup. | Volá GetNext probe vstupu dokud nenalezne shodu. Vrací shodný řádek. | Uzavře probe vstup. Odstraní hash tabulku. |
| **Merge Join** | Otevře oba potomky. | Volá GetNext potomka s nejnižší klíčovou hodnotou dokud není nalezena shoda. Vrací shodný řádek. | Uzavře potomky. |

# Operátory pro načtení dat

**Clustered / Non-clustered index scan**

- Načtení všech řádků daného indexu



**Clustered / Non-clustered Index Seek**

- Hledání v indexu s využitím klíče

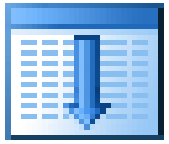


**Table Scan**

- Kompletní průchod všech záznamů v Heap

lasaris

# Operátory pro načtení dat

## Key Lookup

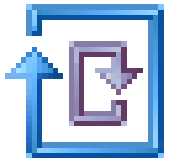
- Načtení řádků z Clustered Indexu v kombinaci s Nested Loop Join

## RID Lookup

- Načtení řádků z Heap v kombinaci s Nested Loop Join

lasaris

# Operátory pro propojení dat

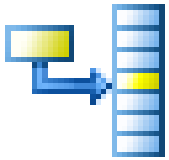


## Nested Loops

- Používán pro: Inner, Left Outer, Left Semi a Left Anti Semi Join
- S každým příchozím řádkem v outer tabulky hledá v inner tabulce

## Merge Join

- Spojení záznamů pokud jsou seřazeny podle stejného sloupce



## Hash Match

- Přečte záznamy na build vstupu a podle daného sloupce spočítá hashovací tabulku
- Čte záznamy na probe vstupu a dohledává k nim záznamy z build vstupu

lasaris

# Další operátory

## Filter

- Na výstup pošle pouze záznamy ze vstupu, které vyhovují predikátu

## Sort

- Vytvoří dočasnou tabulku, ve které seřadí záznamy

## Top

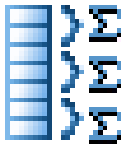
- Ze vstupu vrátí pouze specifický počet prvních záznamů

## Compute Scalar

- Vyhodnocuje výraz a spočítá skalární hodnotu

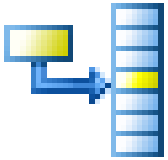
lasaris

# 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

lasaris

# Paralelní zpracování

## Distribute Streams

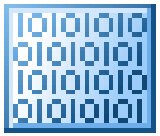
- Záznamy rozdělí na více paralelně zpracovávaných výstupů

## Gather Streams

- Sloučí se vstupy z několika 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

lasaris