Optimalizace SQL – poznámky

[Helios Objekty 2](#_Toc87442159)

[Autoclose 3](#_Toc87442160)

[Prefixy 4](#_Toc87442161)

[Indexy 5](#_Toc87442162)

[Optimalizace dotazů/plánů 7](#_Toc87442163)

[Paralelismus 9](#_Toc87442164)

[Omezení RAM 11](#_Toc87442165)

[WaitStats 12](#_Toc87442166)

[LOG Blokací 14](#_Toc87442167)

[Cardinality Estimater 15](#_Toc87442168)

[Čistění cache 16](#_Toc87442169)

[Co se děje pokud přeteče RAM 17](#_Toc87442170)

[T-SQL - Systémové příkazy 18](#_Toc87442171)

[SQL Objekty 20](#_Toc87442172)

# Helios Objekty

**nultá DB** – systémová databáze (s číslem 0 v TabDBHelios)

select \* from TabSetup

– heslo potřebné pro některé operace v Heliosu (SuperHeslo)

select \* from TabTop

– nastavení špinavého čtení (NOLOCK) a omezení řádků (TOP)

select \* from [dbo].[TabUzivAtr]

– ukáže informace o uživatelských atributech (sloupcích), které mohu editovat

exec hp\_BenchmarkTest

– procedura, která změří výkon počítače na databázi

select \* from Tempdb..#TabExtAkce(ID)

– do tabulky se předávají z externích akcí (většinou procedura, pluginy)

## Vývojáři Tabulek

TabDosleObjR02, ... Lukas Barton

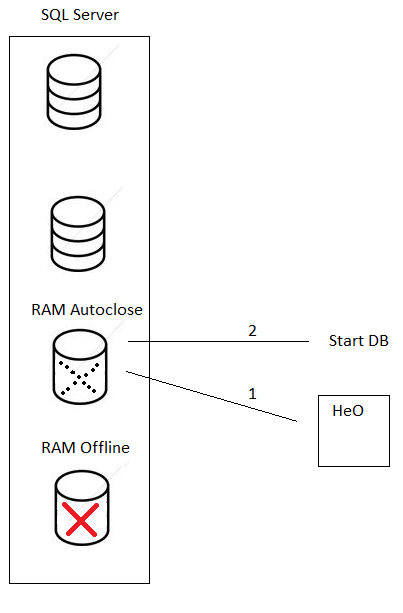
TabDenik ... Ludek Dostal

TabDokladyZbozi ... Milan Bendl

TabMZxxxx ... Michal Kubasek

# Autoclose

* vypne databázi, ale oproti offline módu se do ní lze dotázat
* při dotazu ji opět nastartuje, což tá o něco déle než standartně



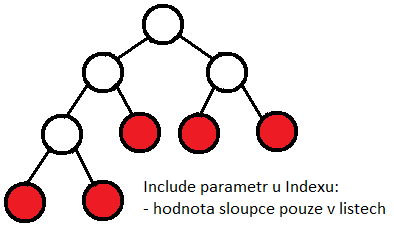
Pozn.: u účetních firem je problém s autoclosem, protože pokud nastavím autoclose, automaticky všichni uživatelé vidí všechny databáze a většinou firmy nechtějí, aby účetní moli vidět i ostatní firmy v agendě (firmy, které spravují jiné účetní)

# Prefixy

|  |  |  |
| --- | --- | --- |
| Zkratka | Význam | Detail |
| hvw\_ | Standartní helios view |  |
| hp\_ | Standartní procedura | Je pár výjimek, například PrepoctiPohybStav, která je rovněž ve standardu. |
| hpx\_ | Externí procedury |  |
| ep\_ | Slepé procedury  (Slepice) | Jsou to procedury, které Helios zavolá v konkrétních akcích pokud existují. A pokud neexistují, tak je nevolá. |
| ht\_ | Standartní trigger  (ale může se jmenovat jakkoliv) |  |
| et\_ | Externí trigger | Jakmile Helios narazí na trigger, který není jeho a nejmenuje se et\_ , tak ho bez milosti zlikviduje. |
| hf\_ | Standartní funkce |  |
| hfx\_ | Nestandartní funkce  (Zkratkou hfx si není Pepa zcela jist) |  |
| PK\_ | Primární klíč | V drtivé většině případů clusterovaný index nad ID. |
| UQ\_ | Unikátní klíč |  |
| IX\_ | Klasický index Heliosu |  |
| IXe\_ | Indexy, které vytvářím já |  |
| dta\_ | Tuning Wizard Index | Dost často dělají neplechu a nejsou optimální. |

# Indexy

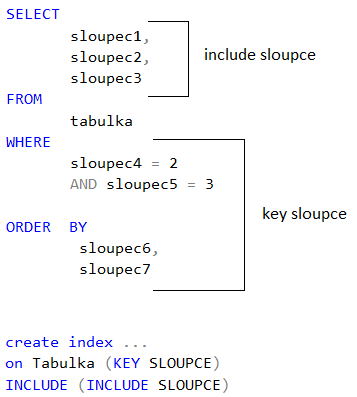
## Included sloupce



INCLUDED sloupce se ukládají pouze do koncových úrovní, čímž šetří místo, protože si SQL Server data netahá celou dobu sebou, zatímco klíčové sloupce jsou ve všech úrovních struktury B-stromu.

příklad: chci najít id 11 a k ní hodnotu ve sloupci „popis“, který je included. SQL Server začne prohledávat index stylem: na jedné hromadě mám 5,1,6,7,8 a na druhé mám 11,10,3,2. Vleze do úrovneň níž a zase hledá a zjistí: na jedné hromadě je 11,10 na druhé 3,2, vleze do další úrovně, kde už je jen samostatně 11. Zde je také hodnota sloupce „popis“ (např.: „Coca-Cola light“) Tedy 11 je v indexu uloženo 3x, zatímco sloupec popis je uložen jen jednou na konci.

Rozlišení, které sloupce dát jako include:



## Fragmentace Indexů

Data jsou uložena v tzv.: stránkách, na stránkách v tabulce vznikají prázdná místa po update a delete operacích a tím jak musí index prohledávat více stránek, dochází k jeho zpomalení.

Velikost jedné stránky je 8 KB (1 MB je tedy 128 stránek ... 128 x 8 = 1024).

Každá stránká začíná 96 Byte hlavičkou, která obsahuje systémové informace o stránce.

Indexy o velikosti pod 4 MB nejsou důležité.

Řešením je příkaz REBUILD A REORAGANIZE index.

U REBUILDU (lze provést online pouze v Enterprise) dojde k odebrání a znovu vytvoření indexu, u REORGANIZE (vždy se provádí online) se seřadí stránky odebráním volného nebo nepoužívaného prostoru ve stránkách. Při online volbě mohou uživatelé s tabulkou dále pracovat.

## Columnstore Index

Hodí se pro OLAP databáze, protože provádí kompresi dat, což je celkem drahá operace. Pokud v tabulce běží hodně transakcí, insertů a updatů, columnstore index by data naopak zpomalil. Pokud ale naopak z tabulky pouze vybírám a dělám nad nimi agregace, columnstore index se naopak hodí a výběr dat výrazně zrychlí.

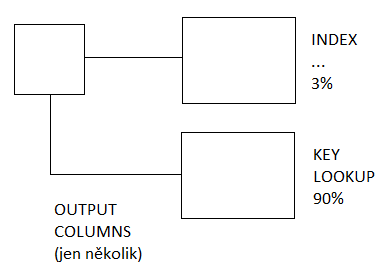
# Optimalizace dotazů/plánů

Postup při optimalizaci:

1. Ctrl + L … náklady (exekuční plán)
2. Missing INDEX – Vytvořit INDEX
3. Zrušit ORDER BY
4. Omezit počet vracených řádků
5. Změřit dobu běhu (set statistics time, io on)

|  |  |
| --- | --- |
| **Problém** | **Optimalizace** |
| if (a = b) else if (a = c) else (a = d) | a in (b, c, d) |
| select top(1) case when a = xx then 1 else 0 end  select top(1) a where a is not null  where a in (select a where b = c) | select exists () |
|  | Může značit problém s paralismem  (nastavit maxdop na 1 nebo vyzkoušet, kolik jader je optimálních) |
|  | Přečíst si varování a zkusit ho vyřešit (např.: pokud je problém z konverzí, dát hodnotu z 17500 na hodnotu ‘17500‘) |
| V select jsou 2 sloupce u jedné tabulky a u joinu je jeden sloupec na který mám index | Zkusit vytvořit ještě jeden index na oba sloupce a porovnat časy a náklady dotazu |
| Existuje Index Seek a Key Column na stjné tabulce a jdou do jednoho výstupu | Udělat index nad sloupci z obou outputů a tím sloučit tyto dva operátory do jednoho |
|  | Zjistit, které sloupce vytahuji a udělat na sloupce non-clustere index a tím změnit plán na Index seek (scan prohledává podle clustered indexu) |
|  | Nejrychlejší, jediná možnost optimalizace pokud uvidíme key\_lookup na stejnou tabulku, tak vytvořit index se sloupci z tohoto key\_lookupu |
|  | Vytvořit index nad sloupci, které hledám.  Nejpomalejší z hlediska výkonu, prohledává se celá tabulka, není zde žádný index. |

Nejlepší na optimalizaci:



* vezme se sloupec v key lookup a vytvoří se s ním index společně ze sloupce z původního indexu

## Set Statisticks IO

**Logical Reads**

* data načtena z cache (RAM)

**Physical Reads**

* data načtena z disku

Jednotka měření STATISTICS IO = 1 read = a single 8kb page = 8192 bytes.

## Exekuční plán – ikony:

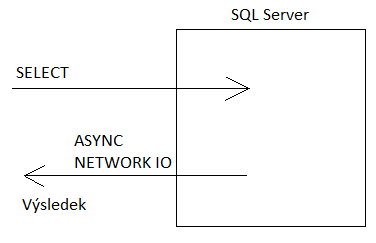
|  |  |
| --- | --- |
|  | Uloží si kopii všech dat, která přečte do indexované tabulky (v tempdb) a v případě potřeby vrátí rychle jejich podmnožinu, protože nemusí volat znovu operátory, které data získávají.  Podobný Table Spool s rozdílem, že index spool nemůže číst data uložená jiným spolem. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Paralelismus

MAXDOP – ideál 1/2 počtu jader v jednom node (max 8)

COST TRESHOLD FOR PARALELISM (Default 5, Nastavit 20, 50, 100, 200, 500)

ASYNC\_NETWORK\_IO – čekání na výsledek (např.: server nemůže poslat celý set dat, protože nemá dostatek paměti například kvůli **příliš velkému result setu** 200 000, je zahlcen, tak pošle rychle menší set, aby uvolnil paměť a načetl do ní další set, který pošle, nebo např.: server je zatížen, takže mu trvá než processor vrátí výsledek a tím vzniká čekání, nebo u klienta se počítají hodnoty z vráceného výsledku v aplikaci, takže server čeká než se spočítá hodnota u klienta a pak pošle další výsledek)



Při použití více vláken se může stát, že všechna vlákna až na jedno dokončí svou práci, ale musí čekat na to jedno, které ještě pracuje (projevuje se vysokými hodnotami CXCONSUMER, CXPACKET). Potom je dobré rekonfigurovat paralelismus:

CXCONSUMER

CXPACKET

--jsou dost vysoko. Navic se objevuji pomerne casto v logu blokaci. to muze znamenat problem s paralelismem.

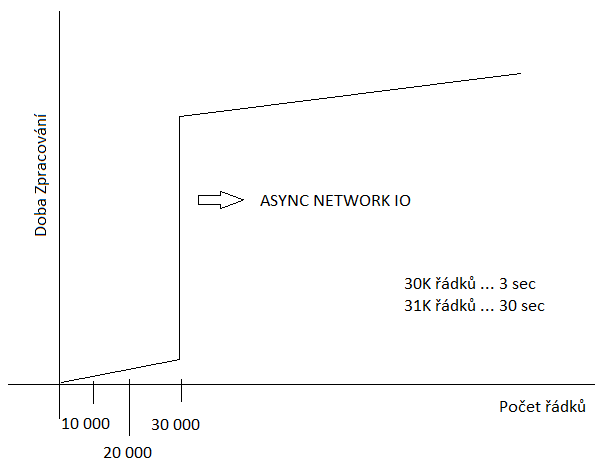
exec sp\_configure 'max degree of parallelism'

--0

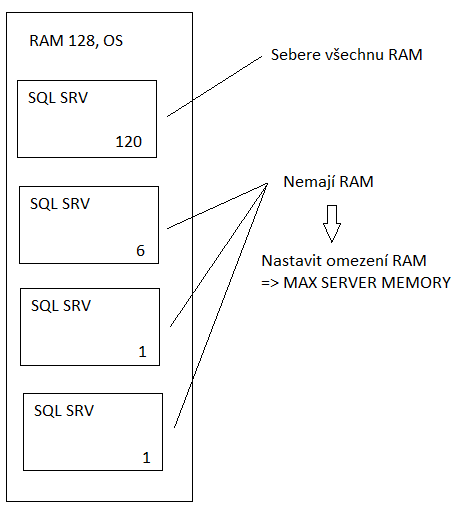
-- paralelismus neni vubec omezeny. To neni dobre. Idealni hodnota je 1/2 poctu jader v jednom node. To je 2

exec sp\_configure 'max degree of parallelism', 2

reconfigure



# Omezení RAM



# WaitStats

**BACKUPIO**

**BACKUPBUFFER**

* Nastanou, pokud zálohujeme na nějaké pomalé zařízení (např.: pomalý disk), nebo nějakým pomalým zálohovacím systémem

**CX packet**

* značí problém s paralelismem (1 vlákno čeká až druhé dokončí práci)

řešení: Nastavit vyšší cost treshold a změnit max degree of parallelism

EXEC sys.sp\_configure N'cost threshold for parallelism', N'25'  
GO  
EXEC sys.sp\_configure N'max degree of parallelism', N'2'  
GO  
RECONFIGURE WITH OVERRIDE  
GO

**FT\_RESTART\_CRAWL**

**FT\_METADATA\_MUTEX**

**FT\_IFTSHC\_MUTEX**

**FT\_IFTSISM\_MUTEX**

**FT\_IFTS\_RWLOCK**

**FT\_COMPROWSET\_RWLOCK**

**FT\_MASTER\_MERGE**

**FT\_IFTS\_SCHEDULER\_IDLE\_WAIT**

* Tyto wait typy lze ignorovat, je to zpracování úloh na pozadí

**IO\_COMPLETION**

* mohou chybět indexy nad náročnými tabulkami, případně problém s pamětí/diskem

**LCK\_M\_XXX**

* Když si chce úloha zamknout zdroj(tabulku) pro sebe, která už je zamčený jiným zdrojem, nastává pak tento wait type

řešení: Nedělat velké transakce, zachovávat malé. Zkontrolovat Isolation level, jestli není nastavený na SERIALIZABLE (pokud to není záměr). Zkusit READ UNCOMMITTED. Identifikovat blokujici dotazy a zkusit je optimalizovat. Použít partitions, to umožní transakcím přistupovat k různým partitions současně.

**PAGEIOLATCH\_XX**

* nastane, když nějaká úloha čeká na data než se přesunou z disku do buffer cache, znamená to slabý hardware

**SOS\_SCHEDULER\_YIELD**

* hodně čekajících procesů, CPU je zatíženo a nestíhá obsluhovat všechna vlákna

řešení: Zkontrolovat počet běžících úloh, optimalizovat dotazy tak, aby byly rychlejší, fragmentace indexů

Dokumentace online:

<https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-os-wait-stats-transact-sql?view=sql-server-2017>

SQLShack dokumentace:

<https://www.sqlshack.com/sql-server-wait-types/>

Waitopedia spotlightcloud:

<https://app.spotlightcloud.io/public/waitopedia>

# LOG Blokací

Příkaz 1 . . . Začátek **STMT**

Příkaz 2

.

.

.

Příkaz 100 . . . => Aktuální příkaz – STATEMENT **STMT**

.

.

.

Příkaz N

# Cardinality Estimater

Správně: cardinality estimator 70

Špatně: cardinality estimator 120

# Čistění cache

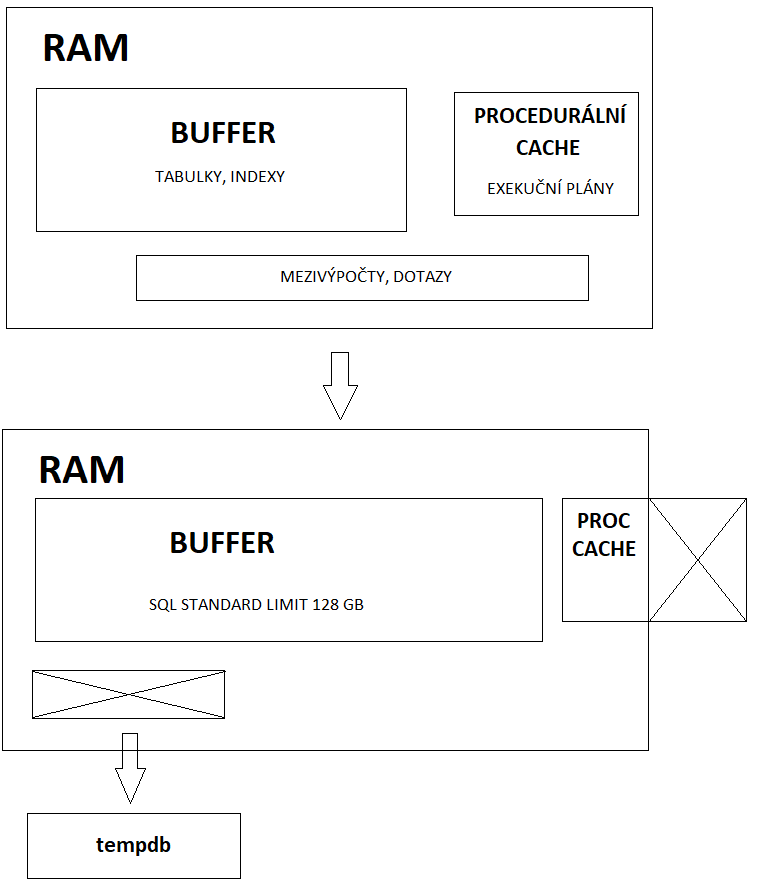
DBCC FREEPROCCACHE

DBCC DROPCLEANBUFFERS

DBCC FLUSHPROCINDB(6)

# Co se děje pokud přeteče RAM

SQL server začne umazávat procedurální cache



# T-SQL - Systémové příkazy

SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES

* zobrazí tabulky z aktuální databáze

SELECT name, object\_id, create\_date, modify\_date FROM sys.tables WHERE name like 't%'

* zobrazí tabulku a její datum vytvoření, začínající na „t“

SELECT \* FROM syslogins

* zobrazí všechny loginy do SQL Serveru

SELECT \* FROM master.sys.database\_principals

* zobrazí users\_name do SQL Serveru

SELECT

USER,

CURRENT\_USER,

SESSION\_USER,

SYSTEM\_USER,

USER\_NAME(),

USER\_ID(),

USER\_SID(),

SUSER\_NAME(),

SUSER\_SNAME(),

SUSER\_ID(),

SUSER\_SID(),

ORIGINAL\_LOGIN(),

DATABASE\_PRINCIPAL\_ID()

* vybere info o aktuálním uživateli

SELECT \* FROM fn\_my\_permissions(NULL, 'Database');

* zobrazí práva uživatele na objektu (Server, Databáze…)

SELECT \* FROM XXX OPTION (RECOMPILE)

* zkompiluje znovu dotaz a sestaví pro něj znovu exekuční plán

sp\_spaceused

* zobrazí informace o velikosti databáze a indexů na ní

Select SERVERPROPERTY('MachineName') as 'MachineName'

* zobrazí opravdové jméno počítač (jméno instance a počítače nemusí být vždy shodné), zejména pokud je instancí na počítači víc

select @@Servername

* zobrazí jméno instance, !!! Nikoliv počítače !!!

# SQL Objekty

## Indexované view (indexed view)

* SQL k němu přistupuje jako k normálnímu objektu, jeho data jsou v databázi uložena stejným způsobem jako v jiném clustered indexu
* používá **schema binding**, to znamená, že nemůžeme měnit strukturu (přidání/změna typu sloupce) na tabulkách, ze kterých je view tvořen, prvně je nutné view odebrat nebo změnit (alter)
* v jeho definici nelze použít EXISTS, NOT EXISTS, COUNT, OUTER JOIN, MIN, MAX, SUB QUERY, TABLE HINTS, TOP, UNION
* z datových typů nelze použít TEXT,NTEXT, IMAGE, XML
* v clusterovaném indexu nesmíme použít dataový typ FLOAT
* pokud používá GROUP BY měli bychom do definici přidat COUNT\_BIG(\*)
* nelze také použít nedeterministické funkce jako GETDATE(), které vrací pokaždé jinou hodnotu
* **!!! Pokud nemáme SQL Server Enterprise**, je nutné vynutit použití indexovaného view v hintem **WITH (NOEXPAND)** v klauzuli FROM v dotazu

Indexované view může být výhoda, pokud jeho data nejsou často updatována, protože snížení výkonu údržbou změn dat je výkonově pomalejší než zvýšení výkonu používáním tohoto view

Hodí se tedy pro OLAP databáze, naopak nejsou vhodná pro OLTP