TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Optimalizace v relačních databázích

Optimization in relational databases

Diplomová práce

Autor: **Bc. Radek Hátle**
Vedoucí práce: Ing. Roman Špánek, Ph.D
Konzultant: Ing. Pavel Tyl

V Liberci 29. 5. 2009

(Zadání)

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum: 29. 5. 2009

Podpis:

Poděkování

Na tomto místě bych chtěl poděkovat a vyjádřit uznání všem, kteří mi pomáhali při vzniku této práce. Především pak vedoucímu práce Ing. Romanu Špánkovi, Ph.D. za zájem, připomínky a čas, který mi věnoval. Také bych chtěl poděkovat své přítelkyni a svým rodičům za poskytnuté zázemí a trpělivost.

Anotace

Cílem diplomové práce je podrobně se seznámit s možnostmi optimalizace relačních databází vytvořených v prostředí MS SQL Server 2005. Následně analyzovat konkrétní databázi ve společnosti Continental Automotive Czech Republic s.r.o., vytvořit a implementovat sadu doporučení.

V první části diplomové práce se seznámíme se strukturami tabulek s principy vyhledávání a vkládání dat do indexů a také zde budou ukázány příklady, které ilustrují výhody a nevýhody indexů.

Další část práce je věnována nástrojům, které nám mohou pomoci při optimalizaci a ladění výkonu. Jsou to dynamické pohledy, plán provádění, pokyny a zásady a také nástroj Profiler a nástroj Databáze Engine Tuning Advisor.

Poslední část práce se zabývá rozborem a optimalizací konkrétní databáze.

Klíčová slova: MS SQL, optimalizace, indexy, plán provádění, pokyny, zásady pro plán

Abstract

The main aim of the diploma thesis is to acquaint in details with ways of optimization in relational databases, particularly with MS SQL Server 2005. Subsequently we analyze a concrete database in the company Continental Automotive Czech Republic s.r.o. After detailed study set of recommendations was created and implemented in order to improve database throughput.

In the first part of the diploma thesis we described table structures and available index structures. Examples illustrating benefits and demerits of indexes are also presented.

The following part of the thesis deals with tools that can help optimizing of database performance. Besides other we addressed dynamic views, execution plans, hints and plan guides and also tools as Profiler and Engine Tuning Advisor.

The last part of the diploma thesis describes the analyze and optimization a concrete database.

Keywords: MS SQL, optimization, indexes, execution plan, hints, plan guides

Obsah

Seznam použitých zkratek

V/V Vstupně výstupní operace
IAM Index allocation Map
LOB Lagre Object Data
RID Row ID (Identifikátor řádku)
SQL Structured Query Language
XML Extensible Markup Language
SSIS SQL Server Integration Services

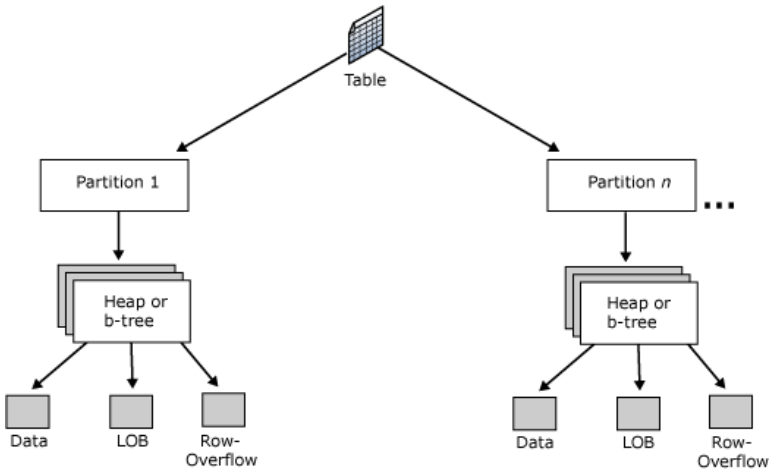
Úvod

Optimalizace výkonu databází patří k pravidelné práci administrátora databázového serveru. Je to takový proces modifikace systému, který vede ke snížení nároků celého výpočetního systému. Optimalizace SQL serveru by se dala rozdělit do dvou kategorií. Vyladění instance SQL serveru, čímž se spíše myslí doplnění hardwaru, protože v SQL serveru není příliš mnoho parametrů, které by se daly nastavit. Lze tedy například přidat výkonnější diskové jednotky, které zkrátí zpoždění při V/V operacích nebo přidat více operační paměti. Hardwarová optimalizace může být cenově dosti nákladná. Může se také stát, že špatně navržená nebo špatně indexovaná databáze, nebude dobře fungovat na sebelepším stroji. Z toho vyplývá další kategorie optimalizace a to vylaďování aplikace. Rozumí se tím vytvoření efektivnějších SQL dotazů, přidání nebo odebrání indexů, oddílů a pokynů. Dále údržba stávající aplikace, jako je například odstranění již nepotřebných dat, pravidelné zálohování, defragmentace atd.

Při optimalizaci se také může stát, že běžné metody jako je správné použití indexů, udržování stále aktuálních statistik, nebo správně napsané dotazy nezaberou a musí se sáhnout po metodě hrubé síly. Tím se rozumí použití pokynů a zásad pro plán, pomocí kterých lze řídit optimalizátor databázového serveru.

1. Organizace tabulek a indexů

Tabulky a indexy se ukládají pomocí stránek o velikosti 8KB. Tabulka se může skládat z více částí, nazývající se partitions. Každá část tabulky obsahuje řádky dat, které mají strukturu clusterovaného indexu nebo haldy (heap). Stránky jsou řízeny jednou nebo více alokačními jednotkami závisejícími na typu dat. Strukturu tabulek a oddílů reprezentuje Obr. 1-1.



Obr. 1-1: Struktura tabulek a oddílů

Zdroj: http://msdn.microsoft.com/en-us/library/ms189051.aspx

1.1 Oddíly

Dělení tabulek je uživatelsky definované a tudíž není povinné. Standardně, tabulka nebo index mají pouze jeden oddíl. Pokud jsou oddíly použity, tabulka a index se rozdělí horizontálně na několik menších částí. Oddíly byly zavedeny až ve verzi MS SQL Server 2005 Enterprise Edition a Developer Edition ve verzi MS SQL Server Standard Edition nelze oddíly použít. Příklady v této práci byly vytvořeny ve verzi Standart, a proto se zde oddíly nebudeme zabývat.

1.2 Alokační jednotky

Alokační jednotka je kolekce stránek. Stránky mohou být uspořádané nebo neuspořádané. Každá tabulka může mít jednu alokační jednotku následujícího typu:

1.2.1 IN_ROW_DATA

Jsou datové nebo indexové řádky, které obsahují všechna data kromě LOB.

1.2.2 LOB_DATA

Large object data jsou data typu text, ntext, image, xml, varchar(max), nvarchar(max), varbinary(max), nebo to také mohou byt uživatelsky definované typy.

1.2.3 ROW_OVERFLOW_DATA

Jsou data proměnné délky typu varchar, nvarchar, nebo data typu sql_variant přesahující velikost 8060 bytů.

Následující příklad získaný z [1] ilustruje, jak lze pomocí systémových pohledů zjistit informace o vybraném objektu. Jeho výsledek zobrazuje Tab. 1-1. Systémové pohledy umožňují zveřejnit metadata související s databází uspořádaným způsobem. Systémové základní tabulky na nejnižší úrovni, ukládají přímo metadata příslušné databáze. Tyto základní tabulky se používají v rámci databázového stroje a nejsou určeny přímo pro běžné uživatele. Systémové pohledy proto slouží k tomu, aby bylo možné k těmto metadatům přistupovat bez možnosti přístupu k tabulkám.

```
USE AdventureWorks;
GO
SELECT o.name AS table_name,p.index_id, i.name AS index_name ,
au.type_desc AS allocation_type, au.data_pages
FROM sys.allocation_units AS au
    JOIN sys.partitions AS p ON au.container_id = p.partition_id
    JOIN sys.objects AS o ON p.object_id = o.object_id
    JOIN sys.indexes AS i ON p.index_id = i.index_id AND i.object_id =
p.object_id
WHERE o.name = N'DatabaseLog' OR o.name = N'Currency'
ORDER BY o.name, p.index_id;
```

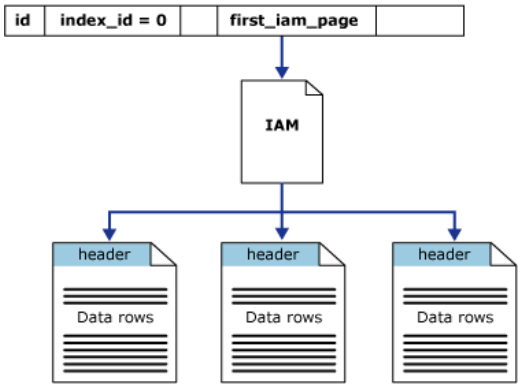
Tab. 1-1: Výsledek systémového pohledu.

table_name	type	index_id	index_name	allocation_type	data_pages
Currency	CLUSTERED	1	PK_Currency_CurrencyCode	IN_ROW_DATA	1
Currency	NONCLUSTERED	2	AK_Currency_Name	IN_ROW_DATA	1
DatabaseLog	HEAP	0	NULL	LOB_DATA	0
DatabaseLog	HEAP	0	NULL	IN_ROW_DATA	143
DatabaseLog	HEAP	0	NULL	ROW_OVERFLOW_DATA	0
DatabaseLog	NONCLUSTERED	2	PK_DatabaseLog_DatabaseLogID	IN_ROW_DATA	1

Z tohoto příkladu si můžeme všimnout, že jsou zde 3 typy objektů. Heap neboli halda, clustered je tabulka, která je setříděná a má clusterovaný index. Posledním je objekt typu nonclustered, což je neclusterovaný index. Každý typ má svůj index_id.

1.3 Heap

Heap je tabulka bez clusterovaného indexu, která má jeden řádek v systémové tabulce sys.partition, kde index_id = 0. Data v tabulce nejsou nijak seřazena a také tu není žádná posloupnost řazení stránek, proto při každém vyhledávání se musí projít celá tabulka. Každá halda má jednu alokační jednotku typu IN_ROW_DATA, LOB_DATA a jednu ROW_OVERFLOW_DATA. Obr. 1-2 ilustruje princip vyhledávání v tabulce typu heap. Sloupec firstIAM v systémovém pohledu sysindexes nebo v pohledu sys.system_internals_allocation_units odkazuje, kde se v databázi nachází první IAM. Index allocation map (IAM) je stránka, v níž jsou informace, které stránky přísluší dané tabulce. Index allocation map je podobný mapě města. Určuje polohu každé stránky v databázi. Pokud bychom neměli k dispozici IAM, bylo by třeba projít všechny stránky. Je to podobné, jako bychom chtěli nalézt ulici ve městě bez použití mapy.



Obr. 1-2: Struktura Heap

Zdroj: http://msdn.microsoft.com/en-us/library/ms188270.aspx

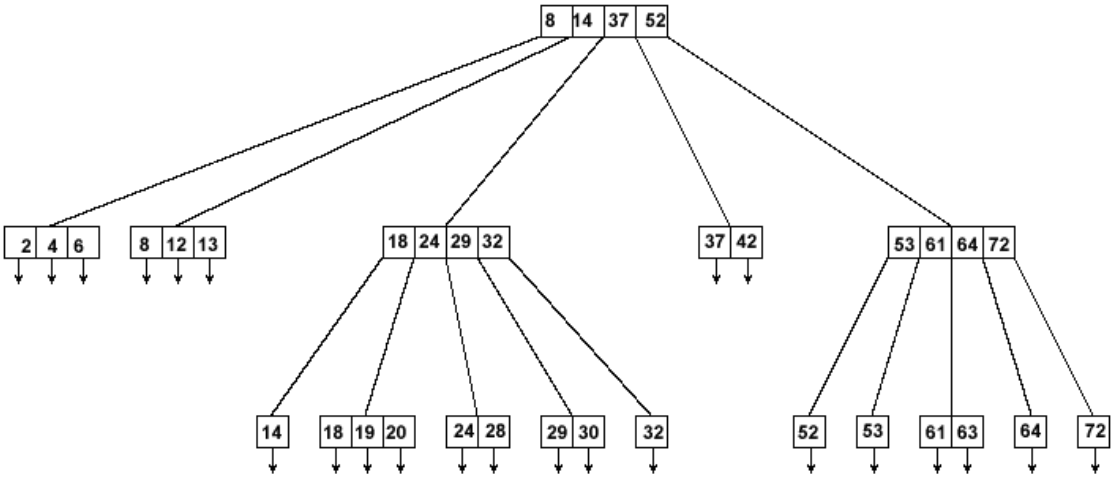
1.4 Indexy

Smysl indexů spočívá výhradně v jejich významu pro V/V operace. Pomocí indexů můžeme zmenšit počet prováděných V/V operací. Při sekvenčním procházení tabulky se generují tisíce nebo dokonce miliony V/V operací. Tyto operace jsou náročné na prostředky počítače. Pomocí indexu lze data vyhledat rychleji, protože k tomu postačuje méně čtení. Pokud probíhá méně V/V operací, zvyšuje se výkon a klesá spotřeba výpočetních prostředků.

Indexy jsou volitelné struktury, které pomáhají přistupovat k datům tabulek rychleji a efektivněji. Indexy nejsou povinné a pokud nejsou k dispozici, neovlivní to výsledek dotazů, pouze jejich výkon. Tento výkonnostní rozdíl může být však dramatický a může mít vliv na celkový výkon systému. Výkon dotazů se zvyšuje tím, že se snižuje počet operací, které jsou potřebné k nalezení daného výsledku. Pokud index chybí, nebo je nevhodně nadefinovaný, je nutné prohledat celou tabulku. Toto prohledávání se nazývá sekvenční a procházejí se všechna data v tabulce. Systém MS SQL Server má několik typů indexů, ale princip indexů je pro všechny stejný. [2]

1.5 Struktura indexu

Struktura indexů připomíná strom. Tento strom se nazývá B-strom. Začíná první stránkou indexu, která se označuje jako kořenový uzel. Kořenový uzel obsahuje rozsahy klíčových hodnot a odkazy na další stránky v indexu. Tyto mezilehlé stránky se nazývají větve. Větve také obsahují hodnoty klíče a odkazy na nižší úrovně větví a nakonec listové uzly. Listový uzel je stránka na nejnižší úrovni indexu a obsahuje hodnoty klíče. Na Obr. 1-3 je struktura B-stromu.



Obr. 1-3: Struktura B stromu

Zdroj: http://svn.apache.org/repos/asf/xml/xindice/trunk/src/documentation/resources/images/.

Počet stránek zabraných indexem závisí na jeho šířce. Šířka indexu je určena tím, kolik sloupců je v klíčích indexu a jak velké jsou tyto sloupce. Počet řádků na stránce indexu závisí na šířce indexu. Větve lze popsat z hlediska úrovní. Úroveň větví

je sada větví, které jsou stejně vzdálené od kořene uzlu. Počet V/V operací požadovaných pro načtení dat závisí na počtu úrovní větví, které je nutné projít k dosažení listového uzlu. Tento počet přímo ovlivňuje výkon načítání požadovaných dat. Počet úrovní větví závisí na šířce indexu a počtu řádků v tabulce.

Při použití indexu se načte kořenový uzel a v závislosti na aktuálně použité hodnotě klíče dojde k rozhodnutí, která větev bude načtena. Větve potom umožňují rychle zpřesňovat výběr dat a sledovat správnou cestu k nim. Nakonec je dosažen listový uzel. Listový uzel poskytuje hodnotu RID, což je číslo stránky a řádku, v případě, jedná-li se o tabulku bez clusterovaného indexu, nebo hodnotu klíče, v případě, že tabulka má clustrovaný index. [2]

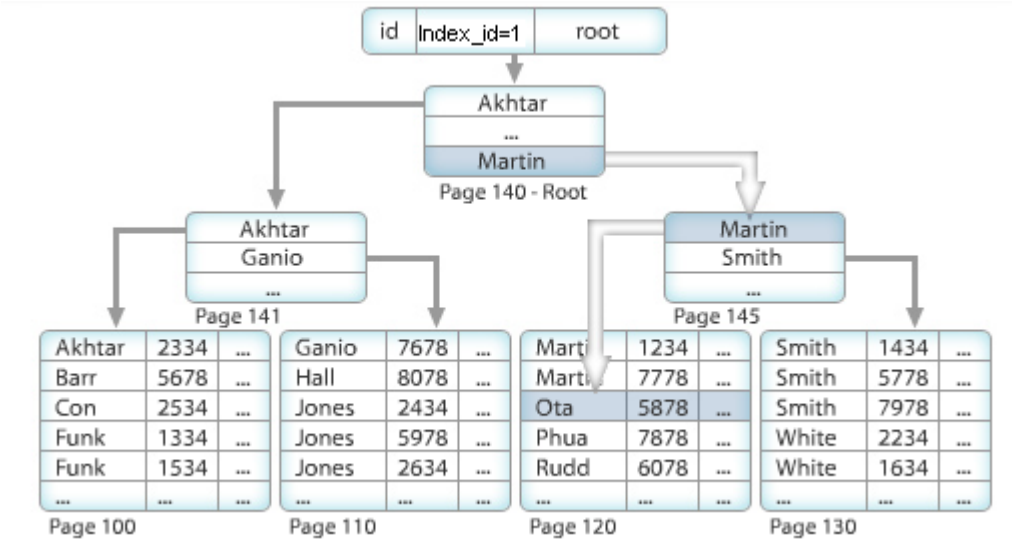
1.6 Clusterovaný Index

Clusterovaný index má jeden řádek v systémovém pohledu sys.partitions, kde je index_id=1 pro každý oddíl používající index. Standardně má clusterovaný index oddíl pouze jeden, protože, jak již bylo uvedeno výše, verze serveru MS SQL Standard Edition nepodporuje oddíly. Pokud by měl clusterovaný index více oddílů, každý by měl také strukturu B stromu, která by obsahovala příslušná data. Každá struktura clusterovaného indexu má jednu nebo více alokačních jednotek, které přísluší danému oddílu. Minimálně každý clusterovaný index má jednu jednotku IN_ROW_DATA. Jestliže clusterovaný index bude obsahovat velká data, bude mít také jednotku LOB_DATA, a samozřejmě pokud bude obsahovat data proměnné délky přesahující velikost 8060b, bude mít také alokační jednotku typu ROW_OVERFLOW_DATA. Stránky příslušející dané tabulce, na které je vytvořený clusterovaný index, jsou řazeny dle klíčové hodnoty indexu ať už sestupně nebo vzestupně, dle definice clusterovaného indexu.

1.6.1 Princip vyhledávání za použití clusterovaného indexu

Tabulka s clusterovaným indexem má v systémovém pohledu sys.system_internals_allocation_units sloupec s názvem root_page, ve kterém se nachází adresa odkazující na kořenový uzel indexu. Obr. 1-4 ilustruje vyhledávání v tabulce, na které je vytvořený clusterovaný index a znázorňuje provádění následujícího dotazu:

```
Select lastname, firstname from member where lastname = 'Ota'
```



Obr. 1-4: Vyhledávání v clusterovaném indexu

Zdroj: http://www.microsoft.com/learning/en/us/syllabi/2073b.aspx

1.7 Neclusterovaný Index

Neclusterovaný index používá stejný B-strom jako clusterovaný index s rozdílem, že data v tabulce nejsou fyzicky řazena podle klíče indexu. Neclusterovaný index může být vytvořen na libovolné tabulce, nebo pohledu s clusterovaným indexem nebo bez něho. Neclusterovaných indexů na jedné tabulce může být několik. Každý řádek stránky neclusterovaného indexu obsahuje klíčovou hodnotu indexu a lokátor. Tento lokátor ukazuje na datové řádky v clusterovaném indexu nebo v haldě.

Lokátor neclusterovaného indexu je buď ukazatel řádku, nebo klíč clusterovaného indexu. Pokud je tabulkou halda, znamená to, že nemá clusterovaný index a lokátor je ukazatel na řádek, který se skládá z id stránky a čísla řádku na stránce. Celý ukazatel se pak nazýva RID. Jestliže tabulka nebo pohled mají clusterovaný index, pak je lokátorem klíčová hodnota clusterovaného indexu. V případě, že klíč clusterovaného indexu netvoří unikátní hodnotu, SQL server vytvoří pomocný unikátní klíč, který se nazývá uniqueifier. Tento pomocný klíč má hodnotu 4 bajtů a není nijak uživatelsky přístupný. Pomocný klíč je vytvořen, pouze pokud je třeba vytvořit unikátní hodnotu klíče v clusterovaném indexu, který je použit jako lokátor neclusterovaného indexu.

Neclusterovaný index stejně jako clusterovaný má jeden řádek v systémovém pohledu sys.partitions, ale hodnota index_id > 1 pro každý oddíl tabulky. Tato hodnota může být až 249 podle počtu vytvořených indexů na tabulce. Minimálně neclusterovaný

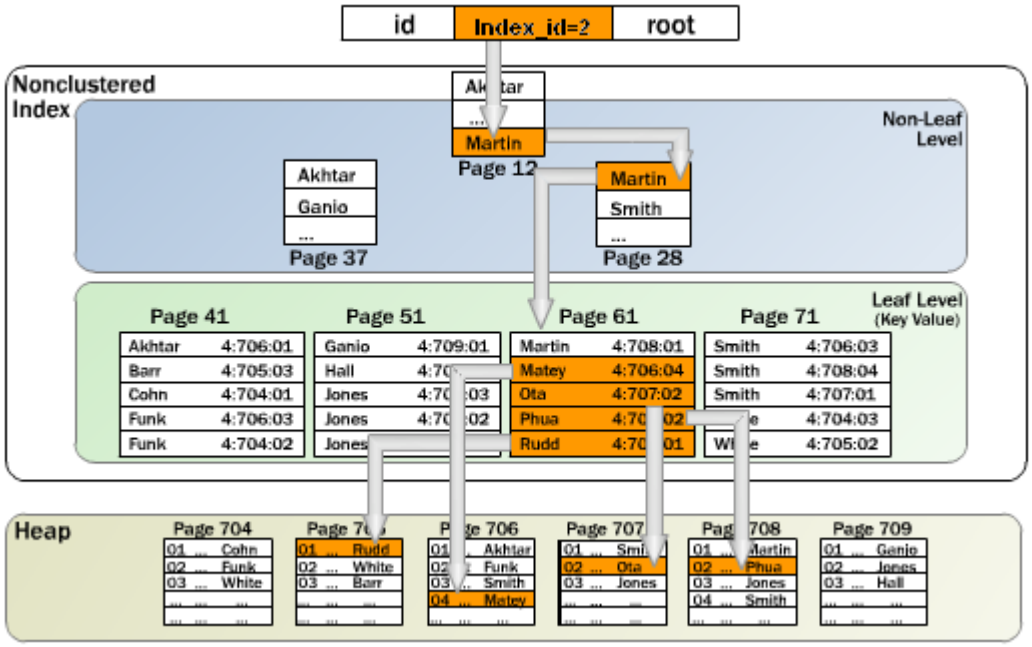
index bude mít alokační jednotku typu IN_ROW_DATA, dále podle obsahujících dat skládajících se z klíče bude mít také LOB_DATA a ROW_OVERFLOW_DATA.

1.7.1 Princip vyhledávání dat na datové struktuře heap za použití neclusterovaného indexu

Obr. 1-5 ukazuje jak budou vyhledávána data při zpracování následujícího dotazu:

```
select lastname, firstname from member where lastname between 'Master' and 'Rudd'.
```

Také je z něho patrné, jak vypadá RID a na jaké stránky odkazuje pro vrácení požadovaných dat.



Obr. 1-5: Vyhledání dat v datové struktuře heap za pomocí neclusterovaného indexu

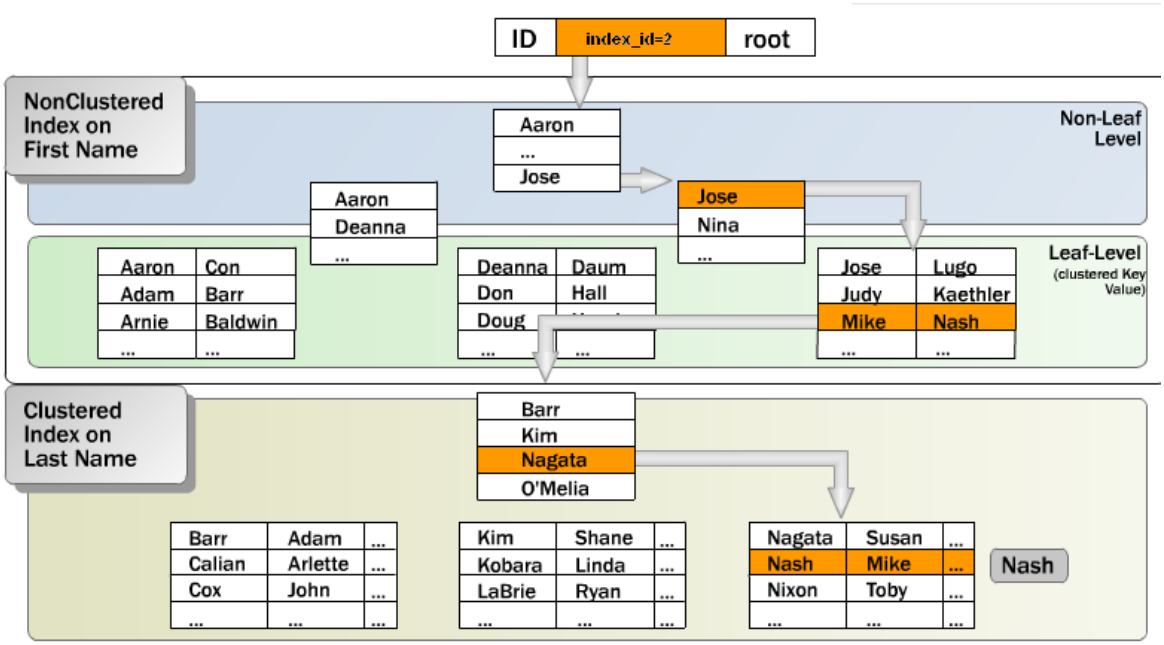
Zdroj: http://www.microsoft.com/learning/en/us/syllabi/2073b.aspx

1.7.2 Princip vyhledávání dat na datové struktuře clustered za použití neclusterovaného indexu.

Pro demonstraci vyhledávání je použit následující dotaz:

```
select lastname,firstname,phone from member where firstname = 'Mike',
```

který ilustruje Obr. 1-6, kde je zobrazen průběh vyhledávání na tabulce `member`. Tato tabulka obsahuje clusterovaný index na hodnotě `lastname` a neclusterovany index na hodnotě `firstname`. SQL server nejprve prochází nelistovou část indexu a porovnává hledanou hodnotu „`Mike`“ s klíči neclusterovaného indexu. V poslední listové úrovni konečně nalezne požadovanou hodnotu „`Mike`“, které přísluší klíč clusterovaného indexu „`Nash`“. Dále SQL server začne procházet clusterovaný index dokud nenalezne hledanou hodnotu v clusterovaném klíči.



Obr. 1-6: Vyhledávání na datové struktuře clustered za použití clusterovaného indexu.
Zdroj: http://www.microsoft.com/learning/en/us/syllabi/2073b.aspx

1.8 Pokrývající index

Pokrývající index je takový index, který zahrnuje dostatek informací k tomu, aby nebylo nutné provádět vyhledávání záložky.

Pokud je například vytvořen index podle sloupců `AddressLine1, AddressLine2, City, StateProvinceID, PostalCode`, lze zadat následující dotaz získaný z [4]:

```
SELECT AddressLine1, AddressLine2, City, StateProvinceID, PostalCode
FROM Person.Address WHERE PostalCode BETWEEN '98000' and '99999';
```

Vzhledem k tomu, že v indexu existují hodnoty `AddressLine1, AddressLine2, City, StateProvinceID, PostalCode` je hodnota vrácena, aniž by bylo nutné prohledávat záložku. Jak již bylo řečeno v úvodu, indexy by se měly udržovat co nejužší, proto v systému SQL Server 2005 je zaveden index se zahrnutými sloupci.

1.8.1 Index se zahrnutými sloupci

Index se zahrnutými sloupci je takový, který obsahuje hodnoty dalších sloupců, jež se nepoužívají v hodnotách klíče indexu. Díky tomu lze vytvářet úzké indexy, které přitom poskytují funkci pokrývání. Jelikož velikost a počet sloupců klíče určuje počet

úrovní indexu, tím pádem i rychlost prohledávání B stromu, je výhodné je udržovat co nejmenší.

1.8.2 Výhody a nevýhody zahrnutých sloupců

U indexu se zahrnutými sloupci nejsou tyto sloupce součástí klíče indexu, ale jsou uloženy v listovém uzlu indexu podobně jako u clusterovaného indexu. Index se zahrnutými sloupci nabízí oproti clusterovanému indexu několik výhod, které byly uvedeny v [2]:

- Na rozdíl od clusterovaného indexu lze pro tabulku nebo clusterovaný index definovat více než jeden index se zahrnutými sloupci.
- Index se zahrnutými sloupci musí obsahovat pouze sloupce, které jsou nutné pro funkci pokrývání.
- Neklíčové sloupce mohou obsahovat datové typy, které nejsou kompatibilní s klíčovými sloupci, například image nebo text z důvodů velikosti.
- Lze se vyhnout vyhledáváním záložky.

Následující příklad uvádí důkaz, kde výsledkem je to, že „úzký index“ se zahrnutými sloupci je lepší než „široký index“, který také pokrývá dotaz, ale svými klíčovými hodnotami. Výsledek měření je uvedený v Tab. 1-2.

Tab. 1-2: Výsledek měření „širokého“ indexů a indexů se zahrnutými sloupci.

	cena plánu	fyzické čtení [počet stránek]	logické čtení [počet stránek]	dosažený čas [ms]	spotřeba CPU [ms]
„Široký“ index	0,180413	3	216	202	0
index se zahrnutými sloupci	0,0269551	1	30	128	0
úspora prostředků [%]	**85,1%**	**66,7%**	**86,1%**	**36,6%**	**0**

Cena plánu byla vyčtena z optimalizátoru, dokumentace neuvádí rozměry jednotek, je to pouze hodnota, kterou optimalizátor oceňuje náročnost prováděného dotazu.

Tab. 1-3: Definice indexů

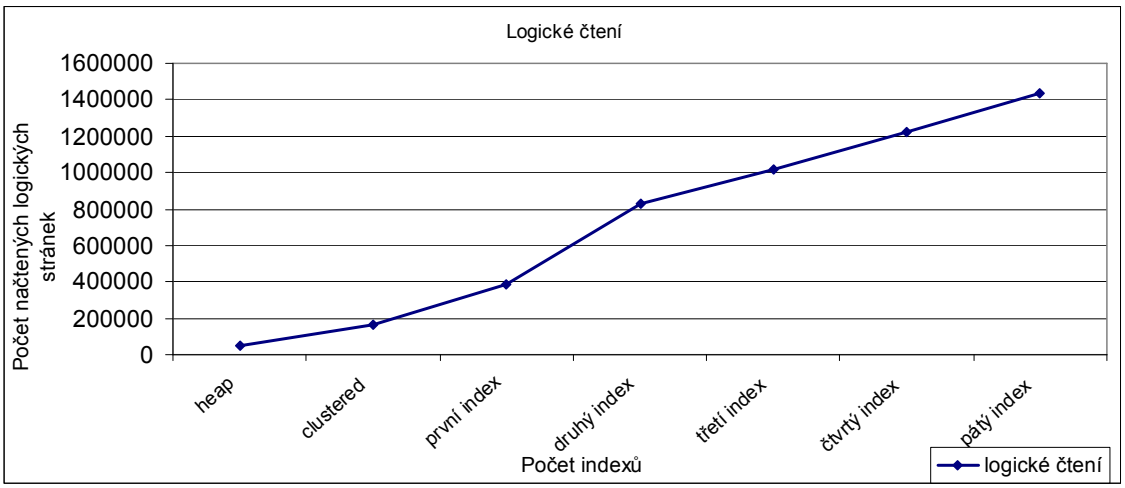
definice "širokého" indexu	CREATE NONCLUSTERED INDEX [jedna] ON [Person].[Address] ([AddressLine1] ASC,[AddressLine2] ASC,[City] ASC, [StateProvinceID] ASC,[PostalCode] ASC)
definice indexu se zahrnutými sloupci	CREATE NONCLUSTERED INDEX [dva] ON [Person].[Address] ([PostalCode] ASC)INCLUDE ([AddressLine1],[AddressLine2],[City],[StateProvinceID])

1.9 Omezení a nevýhody indexů

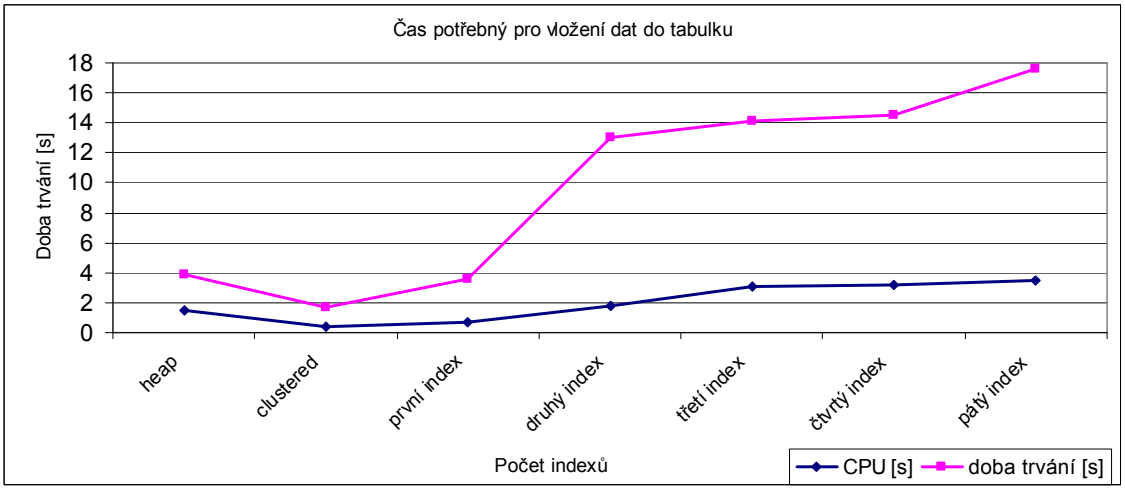
Indexy mají také své nevýhody, a to v podobě vkládání mazání a aktualizace záznamu. Tyto nevýhody se projevují vyššími náklady na prostředky počítače. Následující příklad porovnává vkládání dat do tabulky, která nejdříve byla typu heap a postupně byl počet indexů zvyšován. Na konci měření měla tabulka 1 clusterovaný a 5 neclusterovaných indexů. Výsledek naměřených hodnot znázorňuje Tab. 1-4, která zobrazuje náročnost prováděného plánu, logické a fyzické čtení, uplynulý čas a spotřebovaný čas CPU. Graf 1-1 zpřehledňuje nárůst logického čtení při postupném vkládání indexů do tabulky. Graf 1-2 zobrazuje celkový čas a čas CPU, který byl zapotřebí pro vložení 50 000 řádků do tabulky Person.Address, ve které se počet indexů postupně zvyšoval. Daná tabulka obsahovala 1,5 milionu záznamů. Z Grafu 1-2 lze dále vyčíst, že vkládání dat do tabulky, která je typu Heap, je časově náročnější než vkládání dat do tabulky, která je typu clustered.

Tab. 1-4:Naměřené hodnoty prováděného dotazů při postupné přidávání indexů.

index/typ tabulky	cena planu	logické čtení [počet stránek]	fyzické čtení [počet stránek]	spotřeba CPU [s]	dosažený čas [s]
heap	1,02184	50707	548,4	1,4844	3,861
clustered	4,61562	165097,4	1,2	0,3502	1,6454
první index	11,5931	382138,2	2,8	0,7186	3,6228
druhý index	16,6915	828513	86	1,7502	13,0046
třetí index	20,1697	1018297,6	632,6	3,0908	14,1468
čtvrtý index	20,6532	1223748,8	728	3,2062	14,553
pátý index	21,1367	1438723,8	644,2	3,4782	17,6348



Graf 1-1: Při zvyšujícím se počtu indexů v tabulce, roste počet načtených logických stránek.



Graf 1-2: Zvyšující se počet indexů, má za následek delší dobu trvání pro vložení dat do tabulky.

V příkladě, který znázorňoval nevýhodu indexu v podobě vkládání, byla použita tabulka Person.Contact ze vzorové databáze AdventureWorks. Tato tabulka obsahovala 1,5 milionu řádků o osmi sloupcích. Počet vkládaných řádků byl 50 000. Jednotlivé indexy, které byly postupně vytvořeny, jsou uvedeny v Tab. 1-5. Křížek zobrazuje vytvoření indexů na daném sloupci.

Tab. 1-5: Rozložení indexů v tabulce

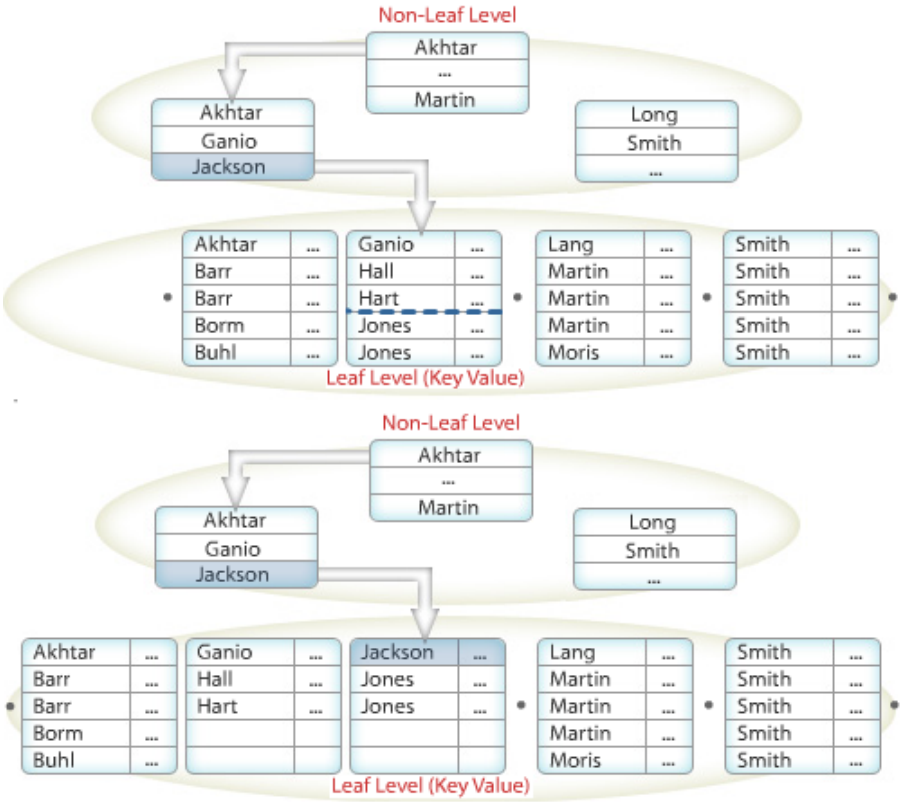
	clusterovaný index	první index	druhý index	třetí index	čtvrtý index	pátý index
AddressID	X	X			X	X
AddressLine1		X			X	
AddressLine2		X		X		X
City		X		X		
StateProvinceID			X	X		X
PostalCode			X	X		
rowguid			X		X	X
ModifiedDate			X		X	

1.10 Princip vkládání nových dat do indexů

Z předchozích grafů je patrné, že při vkládání dat do tabulky, nad kterou je vytvořeno několik indexů, se proces vkládání zpomaluje. To zapříčiňuje tzv. page split neboli rozdělení stránky. Pokud je stránka, do které se mají vložit nová data plná, databázový stroj vezme zhruba polovinu stránky a její obsah přesune na stránku novou. Tím dojde k jejímu rozdělení a uvolní se prostor pro vložení nových dat. Obr. 1-7 ilustruje jak budou indexové stránky rozděleny při použití následujícího dotazu:

```
Insert into member (lastname) VALUES ('Jackson');.
```

Horní část obrázku uvádí obsah stránek před rozdělením, dolní část obsah stránek po rozdělní. Rozdělování stránek také zapříčiňuje fragmentaci indexů, která dále způsobuje nárůst V/V operací.



Obr. 1-7: Rozdělení stránek při vkládání nových dat.

Zdroj: http://www.microsoft.com/learning/en/us/syllabi/2073b.aspx

1.11 Redukce náročnosti vkládání dat do indexované tabulky

Rozdělování stránek lze částečně odstranit tím, že při vytváření indexů, se indexové stránky nezaplní úplně a zbylý prostor tak zůstane volný pro vložení nových dat. Zaplnění stránky se nazývá fill factor, který se při definici indexu uvádí v rozmezí

od 0 do 100. Defaultně je nastaven na 0, což znamená optimalizované zaplnění indexu. Tím je myšleno, že listové uzly jsou úplně zaplněny a v horní úrovni indexového stromu nechává prostor pro nová data. Jakákoliv jiná hodnota bude znamenat skutečné procento zaplnění. Fill factor tedy může redukovat prostředky potřebné pro vkládání nových řádků do tabulky. V případě, že stránky zůstanou z určité kapacity nezaplněné, bude potřeba více stránek pro zaindexovaní celé tabulky. Například při použití fill faktoru 50 bude zapotřebí použít dvojnásobku stánek, než v případě zcela zaplněné indexové stránky. Neúplné zaplnění stránky negativně ovlivňuje vyhledávání dat, protože se musí k dosažení výsledku projít více stránek. Je tedy nutné stanovit kompromis mezi vkládáním a vyhledáváním dat. Jednotlivé typy indexů se vyznačují několika omezeními. Tato omezení jsou uvedena v Tab. 1-6, která byla získaná z [2].

Tab. 1-6: Omezení clusterovaných a neclusterovaných indexů

Omezení indexu	hodnota
Počet clusterovaných indexů v tabulce	1
Počet neclusterovaných indexů v tabulce	249
Počet klíčových sloupců v indexu.(Poznámka:Tato hodnota nezahrnuje dodatečné neklíčové sloupce v indexu se zahrnutými sloupci)	16
Maximální velikost záznamů klíče indexu. (poznámka:Tato hodnota nezahrnuje dodatečné neklíčové sloupce v indexu se zahrnutými sloupci)	900 bajtů

1.12 Fragmentace indexu

Jak jsou časem modifikována data v tabulkách a aktualizovány indexy, stávají se čím dál tím více fragmentovány. Fragmentace indexu znamená, že logické řazení dat nekoresponduje s fyzickým uložením dat v databázi. Čím více je index fragmentován, tím více klesá výkon použitého indexu. Pro odstranění fragmentace lze použít buď reorganizaci nebo rebuild indexů.

Reorganizace indexů způsobuje přeskupování dat a kompresy indexů. Jakékoliv prázdné stránky jsou odstraněny. Komprese indexů je založena na faktoru zaplnění, který je uvedený v systémovém pohledu sys.indexes. Reorganizace indexů je prováděna tzv. online, což znamená, že ostatní procesy mohou přistupovat k tabulce.

Rebuild indexů pracuje v podstatě tak, že zruší index a vytvoří nový. Tímto způsobem je fragmentace odstraněna, protože již v nově vytvořeném indexu bude logické uspořádání dat korespondovat s fyzickým. Vzhledem k tomu, že při rebuildu dochází k úplnému odstranění indexů, ostatní procesy tyto indexy již nemohou použít,

což může způsobit značnou degradaci výkonu do doby, než budou znovu indexy vytvořeny. Navíc procesy nemohou uzamknout tabulku v době, kdy se provádí rebuild.

1.13 Online rebuild indexů

Verze MS SQL 2005 přináší novou metodu jak odstranit nevýhody popsané výše. Tato možnost se nazývá Online rebuild. To znamená, že ostatní procesy mohou i nadále přistupovat k indexům i za stavu, kdy se provádějí údržby v podobě rebuildu. Zajišťují to následující kroky. Původní index zůstává nadále funkční pro čtení a modifikaci dat. Nový index vznikne tak, že se vytvoří kopie původního a všechny transakce se zapisují do obou indexů. Nový index se v tu chvíli používá pouze pro zapsání změn a původní pro čtení. Ve chvíli, kdy je online rebuild indexu dokončen, starý index se natrvalo odstraní a všechny požadavky jsou směřovány na nový, právě vytvořený index. Jelikož oba indexy existují v určitou chvíli najednou, musí administrátor počítat s dvojnásobnou spotřebou diskového prostoru.

1.14 Plán údržby

MS SQL Server obsahuje průvodce, pomocí kterého lze snadno vytvořit plán údržby pro opravu fragmentace, kontrolu konzistence, provádění „shrinku“ (defragmentace) databázových souborů, zálohování databáze nebo také aktualizaci statistik atp. Takto vytvořené úlohy, lze pomocí SQL Server Agent naplánovat a ten je bude spouštět v určitou dobu na námi zvolené databázi.

Doporučení, které uvádí Microsoft, je takové, že pokud fragmentace indexu je menší než 30 %, je vhodné použít k odstranění reorganizaci indexu. Pokud je fragmentace větší jak 30 %, je vhodnější použít rebuild indexu. Vzhledem k tomu, že průvodce pro vytvoření plánu údržby nenabízí kritérium pro stanovení hranice kdy použít reindex nebo rebuild indexů, byla vytvořena vlastní procedura, kde základem jsou dynamické pohledy. Tato procedura bude podrobně rozebrána v kapitole dynamických pohledů.

2. Pohledy

Pohled si lze představit jako virtuální tabulku, která je vytvořena pomocí jazyka T-SQL příkazem SELECT. Pohledy přinášejí mnoho výhod. Jednou z nich je zpřehlednění práce s daty a tvoření složitých dotazů. Pohledy lze vytvářet opět z pohledů, které mohou být různě spojovány a kombinovány s dalšími tabulkami a pohledy. Další neopomenutelnou vlastností pohledů je ochrana dat. Představme si tabulku, která obsahuje informace o zaměstnancích jako je jméno, příjmení, rodné číslo, bydliště, plat atd. Vytvoříme-li pohled nad tabulkou obsahující zmiňovaná data, kde nezahrneme citlivé sloupce jako je plat a rodné číslo, nemusíme se obávat jejich zneužití. Samozřejmě my jako správci bychom mohli nastavit jednotlivá práva ke každému sloupci v tabulce. Tato cesta by však byla zdlouhavější a použití pohledů v tomto případě je mnohem elegantnější řešení.

2.1 Indexované pohledy

Při každém otevření pohledu dojde ve skutečnosti ke spuštění SQL dotazu, který může být dosti náročný na prostředky serveru a to může mít za následky, že klient bude netrpělivě dlouho čekat na svoji odpověď, kterou poslal na daný server. V lepším případě se klient dočká své odpovědi za několik vteřin, v tom horším může čekat i minuty nebo se nemusí dočkat vůbec. Naštěstí existuje řešení, které vytvoří z virtuální tabulky tabulku fyzickou. Takováto tabulka se nazývá indexovaný pohled a od běžného pohledu se liší tím, že obsahuje jedinečný clusterovaný index, který fyzicky uloží data daného pohledu. To znamená, že index ve skutečnosti obsahuje data pohledu a pohled se nevyhodnocuje při každém přístupu. Indexovaný pohled se někdy označuje jako materializovaný pohled. Sada výsledků indexu je ve skutečnosti uložena v databázi jako tabulka s clusterovaným indexem. Tento fakt může představovat značnou výhodu, jak již bylo řečeno, protože tyto pohledy mohou obsahovat složitá spojení a agregační funkce. Tím se omezuje nutnost počítat nejen agregované hodnoty při každém přístupu.

2.1.1 Ukázka využití indexovaného pohledu

Pro demonstraci síly clusterovaného pohledu byly použity 3 tabulky s názvem table1, table2, table3. Dvě z nich obsahovaly 3 atributy a jedna 2. Tyto tabulky byly spojeny a byl vytvořen pohled s clusterovaným indexem. Syntaxe pohledu v SQL je následující:

```
CREATE VIEW [dbo].[IndexovanyPohled] WITH SCHEMABINDING AS
SELECT     table1.id, table1.DATA1, table2.DATA2, table3.DATA3
FROM         dbo.table1 INNER JOIN
             dbo.table2 ON table2.id = table1.Join2 INNER JOIN
             dbo.table3 ON table3.id = table2.Join3
GO
CREATE UNIQUE CLUSTERED INDEX [PK_INDEX_ID] ON
[dbo].[IndexovanyPohled]([ID] ASC)
```

Z prvního pohledu se jedná celkem o běžný pohled až na klíčové slovo `WITH SCHEMABINDING`. To zaručuje, že zdrojové tabulky nemohou být modifikované a ani nemohou být smazány. Samozřejmě lze do těchto tabulek nadále vkládat data. Klíčové slovo `SCHEMABINDING` je pro vytvoření materializovaného pohledu nutné. Vzhledem k tomu, že MS SQL Server Standard Edition s materializovanými pohledy nepočítá, ale umí je zpracovávat, je nutné použít nad daným pohledem hint `with (noexpand)`. Hinty jsou nápovědy, kterými lze řídit optimalizátor dotazů. Jejich podrobné využití bude uvedeno v některé z následujících kapitol. SQL dotaz zobrazený níže, demonstruje použití hintu, který je nezbytný a napovídáme tím optimalizátoru, že na pohledu je vytvořen clusterovaný index a že optimalizátor nemá rozvíjet plán dotazu do úrovní použitých tabulek.

```
SELECT [id],[DATA1],[DATA2],[DATA3]FROM [dbo].[IndexovanyPohled] with
(noexpand)
```

Tab. 2-1: Ušetřené prostředky materializovaného pohledu.

	cena plánu	fyzické čtení [počet stránek]	logické čtení [počet stránek]	dosažený čas [ms]	spotřeba CPU [ms]
běžný pohled	27,67	12	6728	13928	4233
materializovaný pohled	2,64	3	2380	11783	453
úspora prostředků [%]	**90,5%**	**75,0%**	**64,6%**	**15,4%**	**89,3%**

V Tab. 2-1 jsou uvedeny naměřené hodnoty, které naznačují sílu clusterovaných pohledů. Dle našeho očekávaní se výkon rapidně zlepšil. Clusterovaný pohled se nejvíce projevil při generování plánu kde úspora byla 90,5 procent a dále při spotřebě CPU, kde byla 89,3 procent.

3. Statistiky

Optimalizátor dotazů využívá statistiky k vytvoření co nejlepšího plánu provádění. Statistiky jsou objekty, které obsahují statickou informaci o distribučních hodnotách jednoho nebo více sloupců tabulek či indexů. Optimalizátor využívá statistiky k odhadu mohutnosti, nebo počtu řádků výsledného dotazu. Odhad mohutnosti dovoluje optimalizátoru vytvořit vysoce kvalitní plán provádění. Například optimalizátor využívá odhad mohutnosti ke zvolení operace index seek namísto operace index scan, čímž se může zlepšit výkon dotazu. Každý statistický objekt obsahuje v prvním sloupci histogram zobrazující distribuční hodnotu sloupce, na kterém byly statistiky vytvořeny. Statistiky na vícenásobných sloupcích obsahují statistickou informaci o korelaci mezi sloupci. Tyto korelační hodnoty jsou odvozeny z počtu odlišných řádků ze sloupcových hodnot. Statistiky lze prohlížet pomocí příkazu DBCC SHOW STATISTICS. Funkce má dva parametry - první je název tabulky, druhý název statistiky. Návratové hodnoty funkce lze nalézt na stránce [5].

3.1 Aktualizace statistik

Je tedy jasné, že statistiky musejí být neustále aktualizovány. O to se stará optimalizátor, který je nesmírně citlivý na přesnost dostupných statistických informací. K tomu, aby mohl optimalizátor aktualizovat statistiky, je nutné mít hodnotu AUTO_UPDATE_STATISTICS nastavenou na ON. Statistiky lze také aktualizovat pomocí funkce `sp_updatestats`.

3.2 Vytváření statistik

Statistiky se vytvářejí automaticky společně s vytvářením indexů. Pokud je funkce AUTO_CREATE_STATISTICS nastavena na ON, MS SQL server bude vytvářet statistiky sám, dle potřeby. Pro vytvoření statistik je také možné použít utilitu Databáze Engine Tuning Advisor, o kterém bude řeč v následujících kapitolách. Statistiky lze také vytvořit ručně pomocí klausule CREATE STATISTICS.

4. Nástroje pro vyladění výkonu

4.1 Dynamické pohledy

Dynamické pohledy DMV jsou systémové pohledy, které zobrazují různé vnitřní čítače databázového stroje a prezentují je způsobem, pomocí něhož můžeme snadno monitorovat výkon. Pohledy DMV obsahují okamžité hodnoty, které se neustále mění, což znamená, že při dalším spuštění dotazu dostaneme jiné výsledky. Všechny dynamické pohledy jsou definované ve schématu SYS a jejich názvy mají tvar dm_ *. Při každém volání pohledu je nutné název schématu vždy uvést. Dynamické pohledy se dělí do dvanácti skupin. Všechny tyto skupiny lze nalézt na stránce [6].

4.1.1 Skupiny DMV pohledů a funkcí související s prováděním

Do této kategorie patří 17 dynamických pohledů, které obsahují informace o právě provedených dotazech. Tyto informace jsou velice užitečné při analýze a vylaďování výkonu. V těchto pohledech lze nalézt, kolikrát se který dotaz spustil, jak dlouho běžel, kolik prostředků spotřeboval atd.

Sys.dm_exec_query_stats

Pomocí tohoto pohledu můžeme zjistit dobu trvání a počet použití jednotlivého plánu provádění, který je momentálně uložen ve vyrovnávací paměti. Dále je možné zjistit časový okamžik kompilace plánu, fyzické a logické čtení a další užitečné informace. Při vylaďování databáze je tento pohled velice užitečný.

Funkce sys.dm_exec_query_plan

Tato funkce přebírá v parametru identifikátor plánu a vrací odpovídají plán v XML.

Pohled sys.dm_exec_sql_text

Tento pohled má vstupní parametr opět identifikátor plánu a vrací odpovídající text příkazu SQL. Pomocí zmíněných funkcí a pohledů lze například vytvořit následující dotaz, který zobrazuje deset nejdéle trvajících příkazů.

```
SELECT top 10 OBJECT_NAME(qp.objectid)AS nezev_obj,* FROM
sys.dm_exec_query_stats
CROSS APPLY
sys.dm_exec_query_plan(sys.dm_exec_query_stats.plan_handle) AS qp
CROSS APPLY  sys.dm_exec_sql_text(sys.dm_exec_query_stats.plan_handle)
ORDER BY total_worker_time/execution_count DESC
```

4.1.2 Dynamické pohledy související s indexy

V této kategorii lze nalézt 3 dynamické pohledy související s indexy a po doinstalování service pack 1 se nabídka rozšíří o další 4. Tyto dynamické pohledy jsou velice užitečné a pomocí nich lze zobrazit provozní přístupové a fyzické statistiky. Dokonce v těchto pohledech lze nalézt doporučení, na kterém sloupci by bylo vhodné vytvořit index a jakého typu. Dále obsahují informace o tom, kolikrát a kdy naposledy se index využil.

Funkce sys.dm_db_index_physical_stats

Tato dynamická funkce má 5 vstupních parametrů. Jsou to: Id databáze, ID objektu, Id indexu, číslo oddílu a režim. Vrací informace o fragmentaci, velikosti dat a indexů v tabulce nebo pohledu.

Následují příklad uvádí, jak lze pomocí funkce sys.dm_db_index_physical_stats a ukazatelů vytvořit proceduru pro rebuil a reindexaci fragmentovaných indexů:

```
use AdventureWorks
go
declare @sql varchar(20),@sql1 varchar(20),@sql2 varchar(20),@sql3
varchar(20),
@sql4 varchar(20),@obj_name varchar(255),@indexname varchar(255),@akce
varchar(255),
@frag INT,@pagecount int,@typ varchar(50)

set @sql='' set @sql1=''set @sql2=''set @sql3=''set @sql4=''set
@obj_name=''
set @indexname='' set @akce='' set @frag=0 set @typ=''

declare kurzor CURSOR FORWARD_ONLY READ_ONLY FOR
SELECT
schema_name(sys.objects.schema_id)+'.'+object_name(sys.objects.object_
id) as obj_name,sys.indexes.NAME,CASE WHEN
sys.dm_db_index_physical_stats.avg_fragmentation_in_percent<30 THEN
'reindex' ELSE'rebuilt' END AS
akce,sys.dm_db_index_physical_stats.avg_fragmentation_in_percent AS
frag,sys.dm_db_index_physical_stats.page_count
,sys.indexes.type_desc
FROM sys.dm_db_index_physical_stats(4,NULL,NULL,NULL,'DETAILED')

join sys.indexes ON
sys.indexes.OBJECT_ID=sys.dm_db_index_physical_stats.OBJECT_ID AND
sys.dm_db_index_physical_stats.index_id=sys.indexes.index_id
join sys.objects on sys.indexes.object_id=sys.objects.object_id
 WHERE sys.dm_db_index_physical_stats.index_level=0
 AND sys.indexes.NAME IS NOT null and
sys.dm_db_index_physical_stats.page_count>1000 and
sys.indexes.is_disabled=0
 ORDER BY sys.indexes.type_desc
```

```
open kurzor
FETCH NEXT FROM kurzor into
@obj_name,@indexname,@akce,@frag,@pagecount,@typ
WHILE @@FETCH_STATUS = 0
      begin
            set @sql=''
            IF (@frag>=30) begin
                  set @sql1='ALTER INDEX '+@indexname
                  set @sql2=' ON '+@obj_name
                  set @sql3=' REBUILD '
                  set @sql=@sql1+@sql2+@sql3
                  exec(@sql)
            end
            IF ((@frag>0.5) and (@frag<30)) begin
                  set @sql1='ALTER INDEX '+@indexname
                  set @sql2=' ON '+@obj_name
                  set @sql4=' REORGANIZE '
                  set @sql=@sql1+@sql2+@sql4
                  exec(@sql)
            end
            FETCH NEXT FROM kurzor into
@obj_name,@indexname,@akce,@frag,@pagecount,@typ
      end
CLOSE kurzor
DEALLOCATE kurzor
```

Pohled sys.dm_db_index_usage_stats

Z tohoto pohledu je možné zjistit počet indexových operací různého typu a jeho poslední časový okamžik. Při použití indexu se inkrementuje odpovídající čítač o jedničku pro každé jednotlivé hledání, prohledávání, vyhledávání nebo aktualizaci nad indexem. Z následujícího příkladu, který byl získán z [2], lze zjistit ke kterému indexu se přistupuje nejméně.

```
SELECT OBJECT_NAME(ios.object_id) AS 'obj_name',
    ios.object_id, i.name AS 'idx_name', i.index_id,
    (user_seeks + user_scans + user_lookups + user_updates)
        AS 'total_usage_count',
    user_seeks, user_scans, user_lookups, user_updates
FROM sys.dm_db_index_usage_stats ios, sys.indexes i
WHERE database_id = db_id()
    AND objectproperty(ios.object_id, 'IsUserTable') = 1
    AND i.object_id = ios.object_id
    AND i.index_id = ios.index_id
      ORDER BY total_usage_count ASC;
```

Další příklad, uvádí jinou možnost použití systémového pohledu, ze kterého je možné zjistit neclusterovaný index, ke kterému ještě nebylo vůbec přistupováno. Nepoužité indexy jsou v databázi zbytečné a výkon databáze snižují, což bylo ukázáno v kapitole 1.9.

```
SELECT OBJECT_NAME(i.object_id) AS 'obj_name',
    i.name AS 'idx_name', i.index_id
```

```
FROM sys.indexes i, sys.objects o
WHERE OBJECTPROPERTY(o.object_id, 'IsUserTable') = 1
    AND o.object_id = i.object_id
      AND index_id>1
    AND i.index_id NOT IN (
        SELECT s.index_id
        FROM sys.dm_db_index_usage_stats s
        WHERE s.object_id = i.object_id
            AND i.index_id=s.index_id
            AND database_id = db_id())
      ORDER BY obj_name, i.index_id ASC;
```

Pohledy sys.dm_db_missing_index_*

Pod tento název patří celkem 4 dynamické pohledy, které byly zavedeny společně s SP1. Jsou to pohledy:

Sys.dm_db_missing_index_columns

Sys.dm_db_missing_index_details

Sys.dm_db_missing_index_group_stats

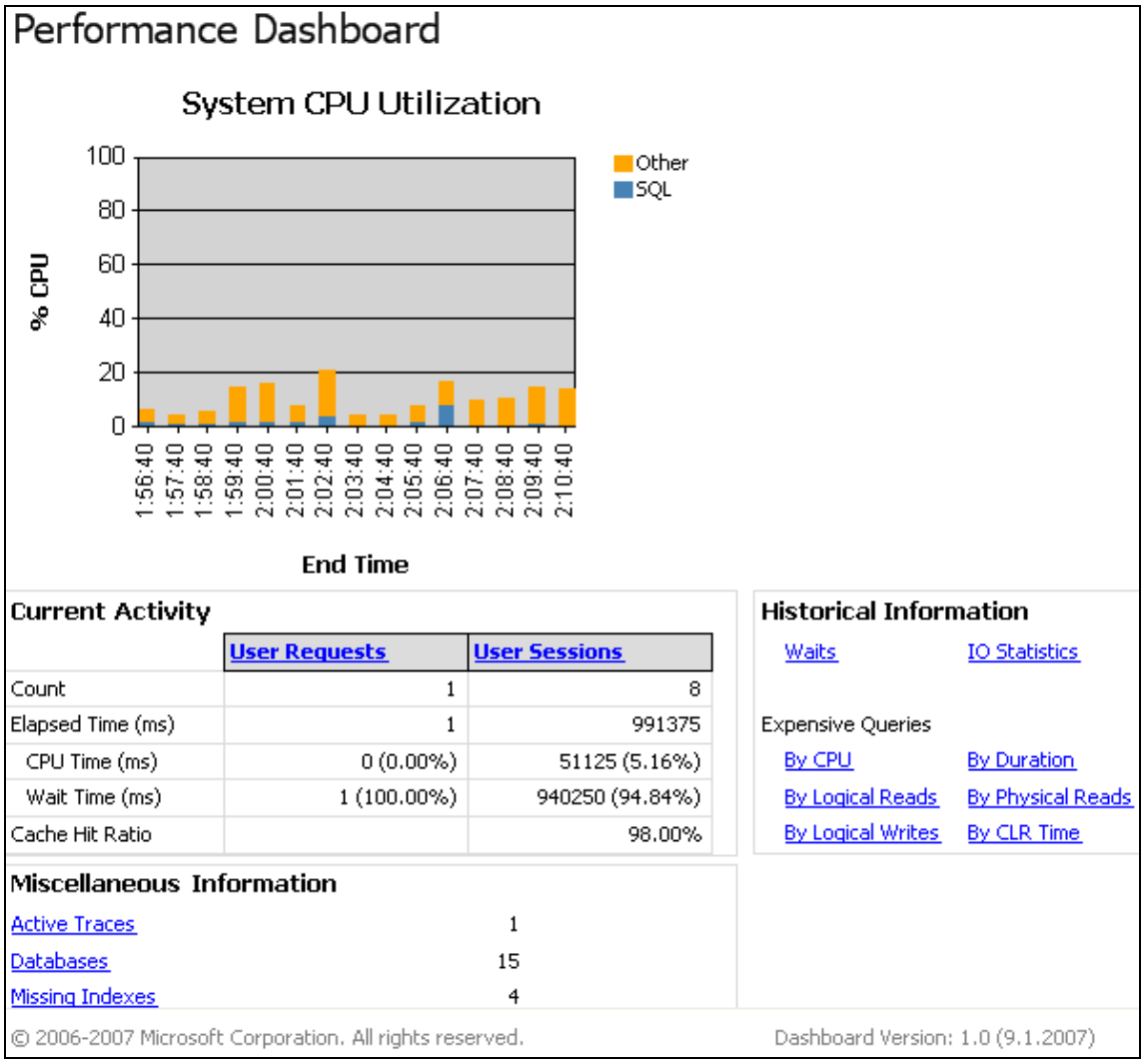
Sys.dm_db_missing_index_groups

Pomocí těchto pohledů lze vytvořit dotaz, který zobrazí chybějící indexy. Vzhledem k tomu, že dynamické pohledy se při restartování MS SQL serveru mažou, je vhodné jejich obsah pravidelně zálohovat, například vytvořením databázového skladu pomocí SQL Server Agenta. Následující příklad uvádí jednu z možností, jak pohledy obsahující chybějící indexy využít.

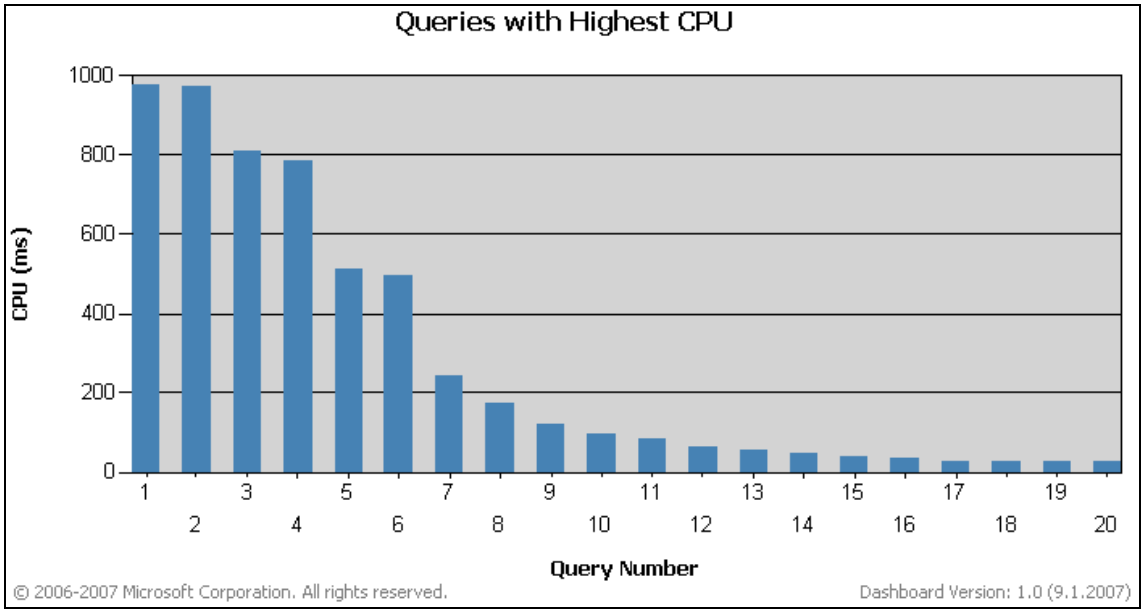
```
select db_name(d.database_id) dbname, object_name(d.object_id)
table_name,
 d.index_handle,d.equality_columns, d.inequality_columns,
 d.included_columns, d.statement as fully_qualified_object
from  sys.dm_db_missing_index_groups g
       join sys.dm_db_missing_index_group_stats gs
on gs.group_handle = g.index_group_handle
       join sys.dm_db_missing_index_details d
on g.index_handle = d.index_handle
where  d.database_id =  d.database_id
 and d.object_id =  d.object_id
```

4.2 Performance Dashboard Reports

Na stránkách [7] lze nalézt utilitu, která zobrazuje momentální aktivity serveru, informace jsou podobné jako informace získané z dynamických pohledů, pouze s rozdílem, že jsou zobrazeny graficky. To ilustruje Obr. 4-1 a Obr. 4-2.



Obr. 4-1: Hlavní grafické okno performance Dashboard.



Obr. 4-2: Nejdéle trvající dotazy v grafickém podání.

4.3 Pokyny a zásady

Další možností, jak optimalizovat databázi, je využití tzv. pokynů (hints), tato metoda by se dala nazvat metodou hrubé síly. Pokyny je doporučeno použít pouze, když už běžné metody, jako jsou například indexy a přesné statistiky, k vylaďování dotazů nestačí. Dokonce Microsoft tuto metodu doporučuje používat velmi obezřetně a jen zkušeným vývojářům a administrátorům.

MS SQL Server generuje plány provádění dotazů dynamicky v nákladově orientovaném optimalizátoru, který zkoumá stav několika systémových prostředků a za pomocí složitých algoritmů vygeneruje nejlepší plán prováděného dotazu. Vzhledem k tomu, že generování plánu může být u velmi složitých dotazů nákladné, plán se ukládá do vyrovnávací paměti (cache), aby jej při dalším volání dotazu bylo možné znovu použít a nemusel se opět vytvářet. Jelikož plány dotazů jsou optimalizovány na konkrétní data v tabulkách, stroj neustále sleduje jejich změny a jakmile usoudí, že by bylo vhodné vytvořit plán nový, spustí rekompilaci plánu. Tento mechanismus funguje ve většině případů správně, ale občas z vlastní zkušenosti databázový administrátor nebo vývojář může usoudit, že vygenerovaný plán není optimální a může jej změnit za pomocí pokynů a vytvořit tak lepší plán něž vygeneroval optimalizátor. Ještě je nutné podotknout, že pokyny v žádném případě nemění výsledek daného dotazu, pouze způsob, jakým budou data vrácena, tedy chování optimalizátoru.

Pokyny lze rozdělit do tří základních typů: pokyny pro dotazy, pokyny pro tabulky a pokyny pro spojení tabulek.

4.3.1 Pokyny pro spojení tabulek

V databázi existují tři fyzické způsoby spojení tabulek. Každý je odlišný a hodí se na určitý druh tabulky. Tato spojení se nazývají: Nested loop, Hash match a Merge join. Optimalizátor si obvykle sám zvolí jaký druh spojení bude nejvhodnější a spojí dané tabulky. Způsob spojení lze vynutit pomocí pokynů.

Nested loop

Tento způsob spojení probíhá tzv. vnořeným cyklem, kde se u každého řádku vnitřní tabulky kontroluje, podle spojovacích kritérii, zda-li jim odpovídají záznamy z vnější tabulky. Tento způsob spojení je nejběžnější a používá se v případech, kdy se spojuje malá tabulka s velkou.

Hash match

Hashované spojení tabulek znamená, že jedna tabulka se uspořádá jako hashovací a ve druhé se pak prochází jeden řádek po druhém. V té se kontroluje rovnost hashe.

Merge join

V tomto případě slučování tabulek probíhá tak, že se nejprve obě tabulky seřadí, a poté se porovnává jeden řádek po druhém z první tabulky, s odpovídajícím řádkem druhé tabulky v sestupném pořadí.

Následující příklad uvádí, jak se zapisují pokyny pro spojení tabulek a to ve dvou různým možnostech zápisu:

```
SELECT  pm.Name,pmi.ModifiedDate
FROM Production.ProductModel pm
INNER loop JOIN Production.ProductModelIllustration pmi
ON pm.ProductModelID = pmi.ProductModelID

SELECT  pm.Name,pmi.ModifiedDate
FROM Production.ProductModel pm,Production.ProductModelIllustration
pmi
where pm.ProductModelID = pmi.ProductModelID
option (loop JOIN)
```

4.3.2 Pokyny pro dotaz

Pokyny pro spojení tabulek se zadávají na konec dotazu v klauzuli option, nebo před klauzuli join tak, jako to ilustruje příklad uvedený výše. V jedné klauzuli lze zadat i několik pokynů. Tabulka Tab. 4-1 uvádí seznam pokynů, které lze pro dotazy použít.

Tab. 4-1: Seznam pokynů pro dotazy

<query_hint > ::=
{ { HASH | ORDER } GROUP
| { CONCAT | HASH | MERGE } UNION
| { LOOP | MERGE | HASH } JOIN
| FAST number_rows
| FORCE ORDER
| MAXDOP number_of_processors
| OPTIMIZE FOR (@variable_name = literal_constant [, ...n])
| PARAMETERIZATION { SIMPLE | FORCED }
| RECOMPILE
| ROBUST PLAN
| KEEP PLAN
| KEEPFIXED PLAN
| EXPAND VIEWS
| MAXRECURSION number
| USE PLAN N'xml_plan' }

Fast

Pokyn Fast znamená, že optimalizátor vrátí co nejrychleji prvních několik řádků. Po navrácení těchto řádků proces pokračuje a stroj vrátí celou výslednou množinu. Tento pokyn lze použít například na webové aplikaci, kde se stránky načítají po 40-ti řádcích a je zcela zásadní vrátit co nejrychleji požadovaná data zákazníkovi.

```
select AddressID,AddressLine1,City,PostalCode
from Person.Address where city='London'
option (fast 40)
```

Recompile

Tento pokyn znamená, že po skončení dotazu SQL server zahodí vygenerovaný plán prováděného dotazu. Tím pádem optimalizátor musí při dalším spuštění dotazu vygenerovat plán znovu, což může být dost nákladné. Má to ovšem smysl za podmínek, kdy celkový přínos rekompilace převýší náklady na samotnou rekompilaci.

Keepplan

SQL Server ukládá plány provádění do vyrovnávací paměti a ty následně rekompiluje za podmínek, že je zapnuta automatická aktualizace statistik a že se v dosud provedených příkazech DELETE, INSERT, UPDATE podkladová data změnila natolik, že má rekompilace smysl. Pokyn keep plan zvýší prahovou hodnotu, což bude mít za následek to, že se dotaz, ani při opakované aktualizaci tabulky, nebude rekompilovat tak často.

Keepfixedplan

Tento pokyn zajistí, že se plán nebude rekompilovat při změně údajů ve statistikách, ale pouze při změně schématu podkladových tabulek nebo při spuštění systémové procedury sp_recompile nad určitou tabulkou.

Optimize for

Při vytváření optimálního plánu dotazu, SQL Server také zkoumá hodnoty jeho parametru, výsledný plán uloží do vyrovnávací paměti a při opětovném spuštění plánu, SQL Server využije již uložený plán. Může se stát, že první hledané hodnoty v tabulce budou značně rozptýlené, a tím pádem se při prvním spuštění dotazu vytvoří ne úplně ideální plán, který se při opětovném spuštění dotazu, ale již s jiným parametrem, použije a dosáhneme tak menšího výkonu. V tuto chvíli je možné použít pokyn optimize for a spustit tak dotaz s konkrétní hodnotou. Optimalizátor tak vytvoří lepší

plán, který se uloží do vyrovnávací paměti. Následující příklad získaný z [12], uvádí použití pokynu optimize for:

```
DECLARE @Country VARCHAR(20)
SET @Country = 'US'

SELECT *
    FROM Sales.SalesOrderHeader h, Sales.Customer c,
        Sales.SalesTerritory t
    WHERE h.CustomerID = c.CustomerID
        AND c.TerritoryID = t.TerritoryID
        AND CountryRegionCode = @Country
   OPTION (OPTIMIZE FOR (@Country = 'CA'))
```

Use plan

Pomocí tohoto pokynu lze vnutit danému dotazu jiný plán provádění, který dle našeho uvážení by mohl být lepší. Plán provádění bude uložen do vyrovnávací paměti. Tento pokyn funguje podobně jako u předchozího příkladu. Názorná ukázka je uvedena v kapitole 4.4.3.

Nebudu zde detailně rozebírat všechny pokyny, protože jejich popis lze nalézt v knize SQL SERVER Books online nebo na adrese [8].

4.3.3 Pokyny pro tabulky

Pokyny pro tabulky fungují podobně, jako pokyny pro spojení nebo pro dotazy. Zadávají se pomocí klauzule WITH. Pomocí pokynů lze určit, zda-li se má použít úplné prohledávání tabulky, jeden nebo více indexů, či jestli se má zvolit určitá metoda zamykání tabulky. Tabulka Tab. 4-2 uvádí seznam pokynů, které lze pro tabulky použít.

Tab. 4-2: Seznam pokynů pro tabulky.

<table_hint> ::=	<table_hint_limited> ::=
[NOEXPAND] {	{ KEEPIDENTITY
INDEX (index_val [,...n])	| KEEPDEFAULTS
| FASTFIRSTROW	| FASTFIRSTROW
| HOLDLOCK	| HOLDLOCK
| NOLOCK	| IGNORE_CONSTRAINTS
| NOWAIT	| IGNORE_TRIGGERS
| PAGLOCK	| NOWAIT
| READCOMMITTED	| PAGLOCK
| READCOMMITTEDLOCK	| READCOMMITTED
| READPAST	| READCOMMITTEDLOCK
| READUNCOMMITTED	| READPAST
| REPEATABLEREAD	| REPEATABLEREAD
| ROWLOCK	| ROWLOCK
| SERIALIZABLE	| SERIALIZABLE
| TABLOCK	| TABLOCK
| TABLOCKX	| TABLOCKX
| UPDLOCK	| UPDLOCK
| XLOCK }	| XLOCK }

Index

Pomocí tohoto pokynu lze optimalizátoru říci, které z indexů má při zpracování dotazu použít. Do pokynu lze zadat několik indexů, které se budou využívat tak, jak jsou vypsány v pořadí. Příklad znázorňuje vynucené použití indexu s názvem PK_Contact_ContactID

```
select ContactID,Title,FirstName,LastName,EmailAddress
from Person.Contact with(index(PK_Contact_ContactID))
where LastName='Hill'
```

Noexpand

Tento pokyn již byl použit při ukázce vytváření materializovaných pohledů v kapitole 2.1 a říká, že se daný pohled nemá rozvíjet na úroveň podkladových tabulek.

Readuncommited

Dle tohoto pokynu nebude možné vydávat žádné sdílené zámky, které by mohly bránit v modifikaci dat v tabulce, a žádné výlučné zámky z jiných transakcí, které by bránily čtení dat v tabulce. To znamená, že je možné přečíst tabulku, nad kterou právě probíhá aktualizace a může se stát, že se načtou nepotvrzená data, která byla zaznamenána jen přechodně nebo vymazána jinou transakcí.

Další příklady využití pokynů pro tabulky lze nalézt na adrese [9].

4.4 Plán provádění dotazu

MS SQL Server umí zobrazovat plán prováděného dotazu, který vytvořil optimalizátor dotazů při zpracování určitého příkazu T-SQL. Pomocí tohoto plánu lze zjistit, zda-li se určitý dotaz provádí efektivně, jestli se využívají indexy tak, jak jsme zamýšleli, nebo lze také najít nejslabší místo prováděného dotazu. Tím je myšlen proces, který zabírá nejvíce prostředků z celého prováděného dotazu. Pomocí plánu provádění lze také nalézt chybějící indexy.

SQL Server umí zobrazovat dva druhy plánů. Prvním z nich je skutečný (actual) plán, ten se vytvoří až po samotném spuštění dotazu. Druhým je pak odhadovaný (estimate). Odhadovaný plán se provede dle nejlepších úvah optimalizátoru a na základě toho, bez nutnosti spouštění dotazu, se vygeneruje. Na odhadovaný plán je nutné si dát pozor, protože se nemusí shodovat s aktuálním a navíc v něm chybějí některé informace.

Plán provádění lze zobrazit ve 3 verzích. První je grafická, ta je nastavena defaultně. Snadno se čte, ale většina informací je skryta. Druhá verze je ve formátu XML a obsahuje mnoho informací, například již zmiňované chybějící indexy. Tu lze zobrazit pomocí klausule SHOWPLAN_XML nebo STATISTICS_XML. Poslední verze je textová. Opět obsahuje více informací než grafická verze, ale také se o něco hůře čte. Textovou verzi lze zobrazit pomocí klauzule SHOWPLAN_ALL nebo SHOWPLAN_TEXT.

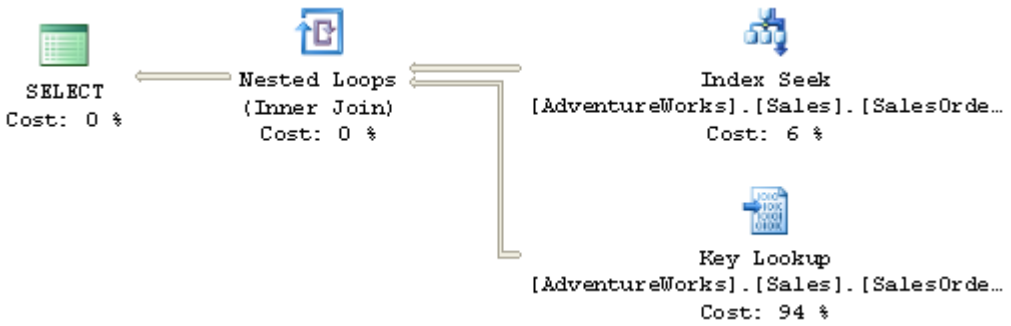
4.4.1 Grafická verze prováděného plánu

Naučit se číst prováděný plán, je jako se učit nový cizí jazyk. V grafickém plánu je několik typů ikonek, které zobrazují určité akce. Význam všech těchto ikonek lze

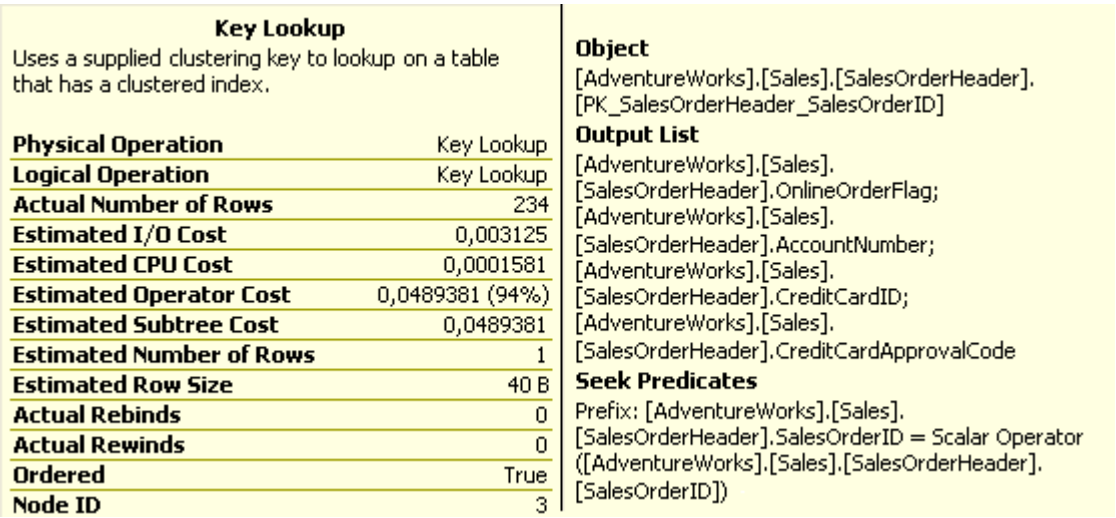
nalézt v (online book pod názvem Graphical Execution Plan icons). Grafický plán se čte zprava do leva a od shora dolů. Čtení grafického plánu bude vysvětleno na několika ukázkových dotazech.

Následující dotaz dostupný z [11], pomocí kterého vyhledáme všechny záznamy obsahující ve sloupci SalesPersonID hodnotu 288, má plán provádění, který je zobrazen na Obr. 4-3. Na tomto plánu jsou vidět čtyři druhy ikonek. Po kliknutí na vybranou ikonu se zobrazí detailní informace, které jsou vidět na Obr. 4-4. Ikona index seek znamená použití indexu, který byl zapotřebí pro vyhledání podmínky where. Key lookup značí vyhledávaní záložky. Tato ikona nám také říká, že použitý index není ideální, protože všechny informace, které jsme hledali, nebyly obsaženy v indexu. To generuje další V/V. Tento dotaz můžeme optimalizovat vytvořením pokrývajícího indexu se zahrnutými sloupci, o kterém již byla řeč v kapitole 1.8. Další operace prováděného dotazu je spojení Nested Loops. Tato operace je nutná, protože část výsledků bylo získáno z indexu a část výsledků pomocí prohledávání záložek. Po kliknutí na ikonu select dostaneme celkovou náročnost plánu, která je v tomto případě 0,052.

```
SELECT  AccountNumber
       ,CreditCardApprovalCode
       ,CreditCardID
       ,OnlineOrderFlag
FROM    Sales.SalesOrderHeader
WHERE   SalesPersonID = 288;
```

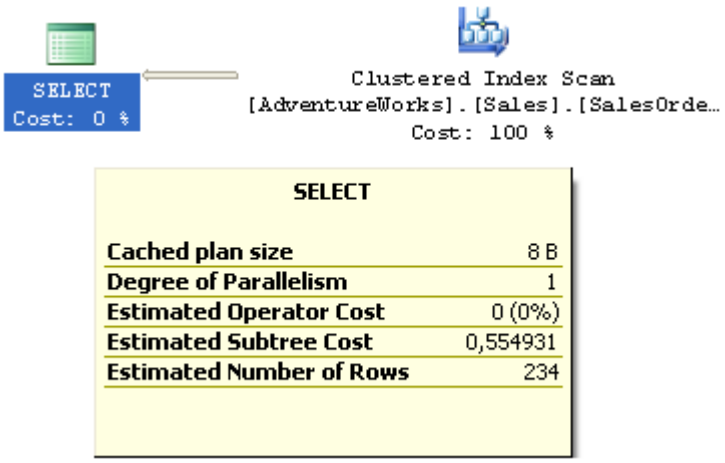


Obr. 4-3: Plán provádění předchozího dotazu.



Obr. 4-4: Detailní informace operátoru Key Lookup.

Pokud změní vyhledávací podmínku na hodnotu 278, plán provádění bude dle Obr. 4-5. Z tohoto plánu lze vyčíst, že k dosažené hodnotě bylo zapotřebí projít clusterovaný index a výsledná cena plánu je 0,55. V tomto případě cena plánu je vyšší než při vyhledávaní hodnoty 288.



Obr. 4-5: Plán provádění předchozího dotazu s hodnotou 278.

Pokud prováděný dotaz přepíšeme a použijeme pokyn optimize for, lze dosáhnout stejného plánu jako u hodnoty 278. Předchozí dotaz lze přepsat na proceduru, která byla získána z [11] a je definována níže.

```
CREATE PROCEDURE Sales.uspGetCreditInfo ( @SalesPersonID INT )
AS
SELECT  AccountNumber,
CreditCardApprovalCode,
CreditCardID,
OnlineOrderFlag
FROM SalesOrderHeader
WHERE SalesPersonID = @SalesPersonID
```

Pokud proceduru zavoláme s hodnotou 288, celková cena prováděného dotazu bude 0,052. Tento plán se uloží do vyrovnávací paměti a další spuštění procedury bude stát 0,052. Ale pokud proceduru spustíme s hodnotou 278 a plán provádění ještě nebude uložen ve vyrovnávací paměti, plán se vygeneruje pro hodnotu 278, kde cena plánu bude 0,55 a tato cena bude již pro všechny další hledané hodnoty stejná, čili zhruba 10x vyšší.

4.4.2 XML verze prováděného plánu

XML verzi prováděného plánu lze zobrazit, jak již bylo řečeno v úvodu, pomocí klauzule STATISTICS XML, to znázorňuje příklad uvedený níže. Po kliknutí na vytvořený plán XML, se otevře nové okno, ve kterém bude zobrazen celý plán. V tomto plánu lze například najít i chybějící indexy, které v grafické verzi nenajdeme. Tyto chybějící indexy jsou obsaženy v XML elementu `MissingIndexes`. Element `ColumnGroup` nám pak naznačí typ indexu, kde jeho atribut nabývá hodnot `EQUALITY`, `INEQUALITY` nebo `INCLUDE`. Část plánu provádění, po spuštění následujícího dotazu získaného z [10], je vidět na Obr. 4-6.

```
SET STATISTICS XML ON;
GO
SELECT CustomerID, SalesOrderNumber, SubTotal
FROM Sales.SalesOrderHeader
WHERE ShipMethodID > 2
AND SubTotal > 500.00
AND Freight < 15.00
AND TerritoryID = 5;
GO
SET STATISTICS XML OFF;
```

```
    <MissingIndexes>
            <MissingIndexGroup Impact="95.8296">
              <MissingIndex Database="[AdventureWorks]"
Schema="[Sales]" Table="[SalesOrderHeader]">
                <ColumnGroup Usage="EQUALITY">
                  <Column Name="[TerritoryID]" ColumnId="14" />
                </ColumnGroup>
                <ColumnGroup Usage="INEQUALITY">
                  <Column Name="[ShipMethodID]" ColumnId="17" />
                  <Column Name="[SubTotal]" ColumnId="21" />
                  <Column Name="[Freight]" ColumnId="23" />
                </ColumnGroup>
                <ColumnGroup Usage="INCLUDE">
                  <Column Name="[SalesOrderNumber]" ColumnId="8" />
                  <Column Name="[CustomerID]" ColumnId="11" />
                </ColumnGroup>
              </MissingIndex>
            </MissingIndexGroup>
    </MissingIndexes>
```

Obr. 4-6: XML verze plánu provádění zobrazující chybějící indexy.

4.4.3 Zásady pro Plán

Zásady pro plán nabízejí mechanismus, který slouží pro zanesení pokynů do původního dotazu, aniž by se tento dotaz musel přímo modifikovat. Zásada pro plán funguje tak, že každý dotaz, který je spuštěn se nejprve porovnává s plánem provádění ve vyrovnávací paměti. Pokud se nalezne shoda, dotaz se provede dle plánu ve vyrovnávací paměti. V opačném případě se začne prohledávat v množině existujících zásad v aktuální databázi a opět se hledá shoda. Po nalezení zásady pro plán, se původní shodný příkaz nahradí příkazem ze zásady pro plán. Plán se zkompiluje a uloží do vyrovnávací paměti a nakonec se provede. Následující příklad uvádí použití zásady pro plán s pokynem use plan. Zásada se vytváří pomocí procedury `sp_create_plan_guide`.

Nejprve je nutné zavolat dotaz a vygenerovat plán provádění v XML pomocí SET STATISTICS XML, toto zobrazuje příklad níže. V níže uvedeném příkladu chceme dosáhnout stejného plánu provádění pro vyhledávací podmínku 288, podobně jako tomu bylo u předchozího příkladu, kde plán provádění u vyhledávací podmínky 288 byl lepší než u podmínky 278.

```
SET STATISTICS XML ON
GO
SELECT  AccountNumber,
        CreditCardApprovalCode,
        CreditCardID,
        OnlineOrderFlag
FROM    Sales.SalesOrderHeader
WHERE   SalesPersonID = 288;
GO
SET STATISTICS XML OFF
```

Takto vygenerovaný plán pak použijeme v proceduře `sp_create_plan_guide`, která byla získána z [11]. Její definice a paramatery jsou následující:

```
EXEC sp_create_plan_guide
    @name = N'UsePlanPlanGuide',
    @stmt = N'SELECT  AccountNumber
       ,CreditCardApprovalCode
       ,CreditCardID
       ,OnlineOrderFlag
FROM    Sales.SalesOrderHeader
WHERE   SalesPersonID = @SalesPersonID --288 --277',
    @type = N'OBJECT',
    @module_or_batch = N'Sales.uspGetCreditInfo',
    @params = NULL,
          @hints = N'OPTION(USE PLAN N''<ShowPlanXML…
```

Nyní již po spuštení procedury Sales.uspGetCreditInfo s jakýmkoli parametrem bude plán vypadat stejně jako s hodnotou parametru 288. Vytvořená zásada plánu pomocí procedury sp_create_plan_guide zůstane uložená v systémové tabulce sys.plan_guides, která se nevymaže ani po spuštění procedury dbcc freeproccache, která maže vyrovnávací pamět. Hint USE PLAN lze použít i bez procedury sp_create_plan_guide, ale plán bude uložen ve vyrovnávací paměti, která může být smazána.

4.5 SQL Server Profiler

SQL Server Profiler je utilita, sloužící k zachycení síťové aktivity mezi klientskými aplikacemi a instancí MS SQL Serveru. Má velmi bohaté grafické prostředí, kde lze nastavit různé události, které si přejeme filtrovat. Toto umožňuje snadnou a přehlednou orientaci v pořízené stopě. Pomocí Profileru je možné například ladit dotazy nebo procedury a sledovat ty, které spotřebovávají nejvíce prostředků. Lze také Profiler nastavit tak, aby zachytával plány provádění. Dokonce pomocí tohoto nástroje se dá analyzovat deadlock. Pořízenou stopu (trace), můžeme uložit do tabulky nebo souboru k pozdějšímu zkoumání.

4.5.1 Předdefinované šablony stop

Profiler obsahuje osm předdefinovaných šablon stop. To nám ulehčí práci při jejich vytváření. Samozřejmě lze také vytvořit vlastní šablonu, záleží jen na nás, co chceme sledovat.

SP_Counts

Tato šablona seskupuje všechna zadaná volání uložených procedur. Výsledkem je seskupení provedených procedur, kde nalezneme jejich počet provedení.

Standard

Šablona shromažduje obecné informace o uložených procedurách, zadaných dávkách T-SQL a také informace o všech připojeních. Tato šablona je vhodná pro sledování všeobecné aktivity.

TSQL

Šablona sleduje všechny zadané příkazy jazyka T-SQL a čas, kdy byly spuštěny. Tato šablona slouží pro ladění klientských aplikací.

TSQL_Duration

Šablona sleduje veškeré dotazy a jejich dobu provádění. Je vhodná pro identifikaci pomalých dotazů.

TSQL_Gruped

Je podobná šabloně TSQL, ale výsledky seskupuje podle uživatelů nebo klientských aplikací.

TSQL_replay

Tato šablona shromažďuje podobné informace o zadaných T-SQL příkazech tak, aby je bylo možné přehrát znovu. Šablonu lze využít při srovnávacích testech.

TSQL_SPs

V této šabloně se shromaždují podrobné informace o prováděných procedurách. Můžeme tak analyzovat jednotlivé příkazy z procedur.

Tuning

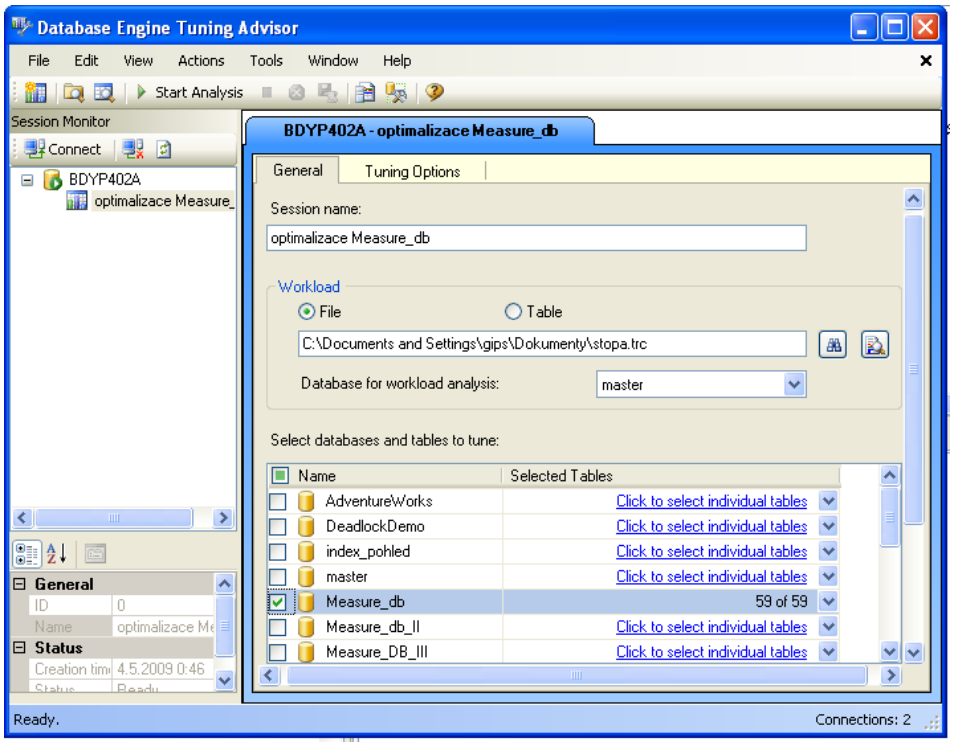
Pomocí této šablony se sledují informace o příkazech T-SQL a uložených procedurách. S touto šablonou můžeme vygenerovat soubor provozní zátěže, který pak po předložení nástroji Databáze Engine Tuning vytvoří sadu doporučení pro optimalizaci vybrané databáze.

4.6 Databáze Engine Tuning Advisor

Nástroj Databáze Engine Tuning Advisor slouží pro usnadnění procesu indexace a optimalizace. Tento nástroj funguje tak, že se mu předloží stopa pořízená pomocí Profileru s reálným chodem SQL Serveru. Advizor na základě zachycené stopy vygeneruje sadu doporučení. U programu můžeme omezit celkovou dobu analýzy provozní zátěže, obvykle se to nedoporučuje, protože čím více času analyzátor stráví při rozboru zátěže, tím kvalitnější budou výsledná doporučení. Tato doporučení můžeme převést do podoby skriptu XML nebo posloupnosti dotazů T-SQL. Můžeme je pak sami posoudit a aplikovat jednotlivě nebo najednou.

4.6.1 Možnosti nastavení vylaďované databáze

Pod záložkou tunning options, která se nachází v horní části okna, jak naznačuje Obr. 4-7, lze nalézt několik možností nastavení.

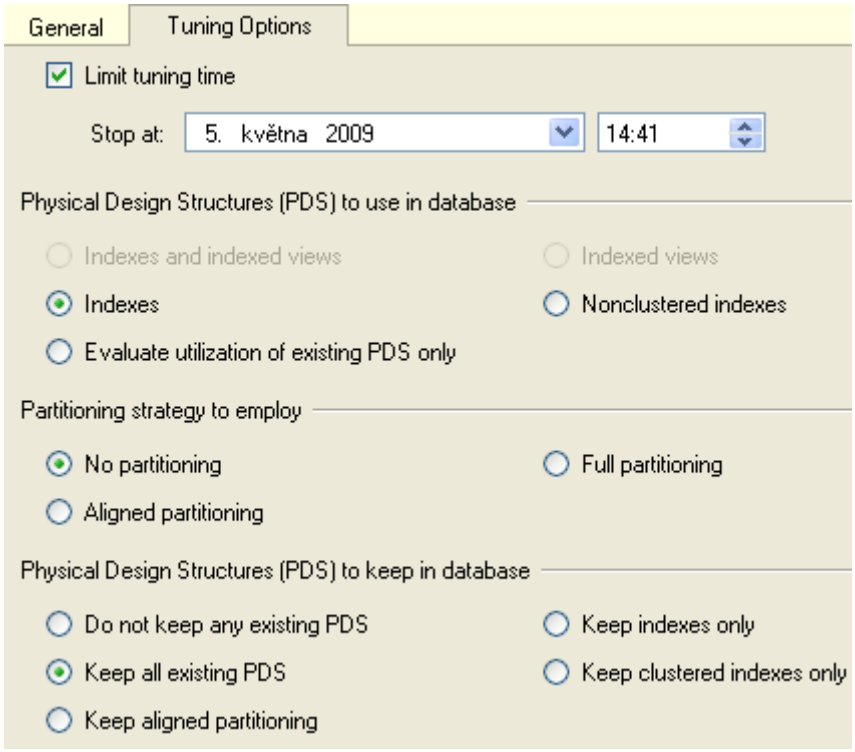


Obr. 4-14: Databáze Engine Tuning Advizor.

Ve Skupině combo boxů pod názvem Physical Design Structures (PDS) to use in databáze, se volí typ struktury, kterou chceme ladit. To je zobrazeno na Obr. 4-15. Jsou to buď indexy a indexované pohledy, nebo jen samotné indexy, samotné indexované pohledy, nebo neclusterované indexy. Lze také vybrat možnost, aby program nedoporučil žádnou možnost pro zlepšení výkonu, ale pouze jen zanalyzoval existující struktury.

Další nastavitelná skupina se týká dělení. V této oblasti se určí, zda-li se vůbec má brát v úvahu strategie rozdělování. Jak již bylo řečeno, verze standard nepodporuje oddíly, proto zde nebudu ani uvádět další možnosti nastavení.

Poslední možnost nastavení je určena k tomu, zda-li se mají zachovat současné struktury či nikoli. Můžeme tak například nastavit, jaké typy indexů se mají odstranit a které ponechat. Další volby v této sekci se týkají toho, zda-li je možné odstranit všechny databázové struktury nebo je vhodné je ponechat.



Obr. 4-8: Možnosti nastavení Databáze Engine Tuning Advizor.

5. Vyladění konkrétní databáze

Při nástupu do společnosti Continental Automotive Czech Republic s.r.o., jsem se seznámil s databází vyhodnocující statistická data z výroby, která byla provozována na MS SQL serveru. Ve chvíli, kdy jsem databázi přebral od původního autora, databáze zpracovávala zhruba polovinu výrobních dat. Můj úkol byl doplnit do databáze další informace tak, aby databáze mohla zpracovávat data z celého závodu. Dalším úkolem bylo rozvíjet databázi dle požadavků, které přicházely od mých kolegů.

Po částečném seznámení s databází jsem doplnil chybějící informace a provedl určité změny tak, aby databáze zpracovávala všechna výrobní data. Po několika dnech databáze přestala fungovat.

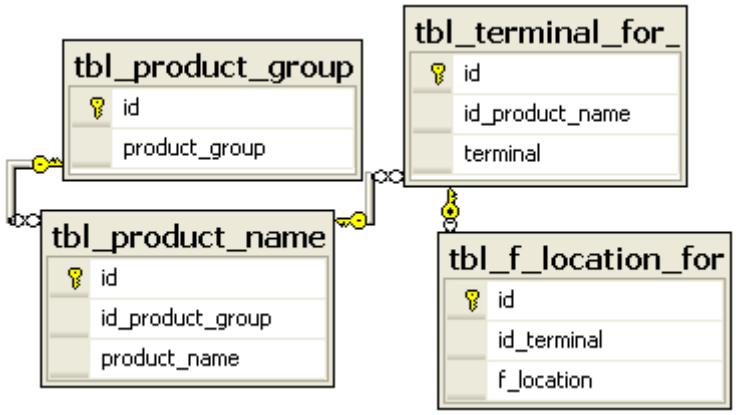
5.1 Princip databáze

Databáze byla navrhnuta tak, že pomocí balíčků, které byly vytvořeny ve Visual studiu za pomocí šablony Integration Services (SSIS), se stahovala dávka dat ze serveru ORACLE z předchozích 24hodin. Na serveru Oracle byla data shromažďována z celé výroby, kde se připisovala několik málo vteřin po té, co výrobek projel testerem na výrobní lince.

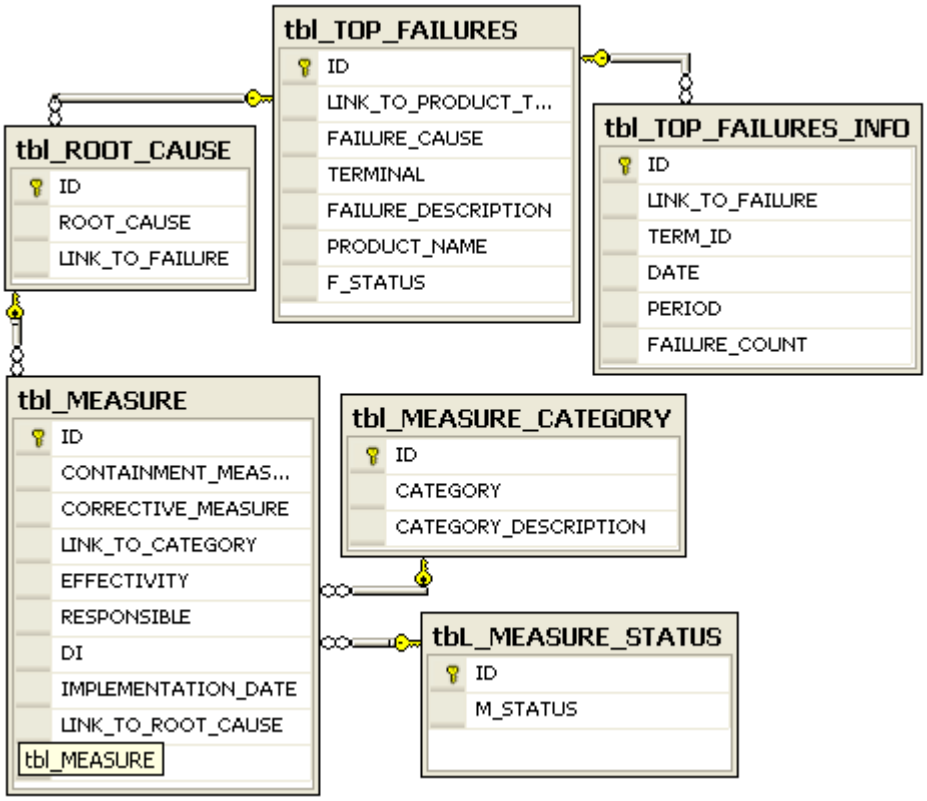
Takto stažená data se na MS SQL Serveru začala vyhodnocovat, třídit a ukládat do tabulek pomocí několika procedur, které byly v určitý čas spuštěny pomocí SQL Server Agenta. Procedury zpracovávaly řádově statisíce řádků dat denně. Řádky se různě filtrovaly a data se vypočítávala, tento proces trval několik minut. Po přidávání dalších produktů do databáze, se množství zpracovávaných dat postupně zvyšovalo. To mělo za následek i delší dobu trvání spouštěných procedur a dokonce někdy se stalo, že procedury běžely stále, bez ukončení a data se nevyhodnotila. Po konzultaci s původním autorem jsem zpracovávání dávky předělal z původních 24 hodin na zpracovávání dat po každé uplynulé hodině. To přineslo malé zlepšení, protože dat již bylo méně a server je bez problému zpracoval. Řešení se ukázalo jako nevhodné, protože se občas stalo, že data z výrobních linek dorazila na server Oracle z nějakého důvodu později, tedy data chyběla a vyhodnocení, které prováděla moje databáze nebylo správné. Proto bylo třeba chybná data odmazat a znovu ručně spustit všechny procedury. Proto jsem začal studovat možnosti optimalizace databáze.

5.2 Struktura Databáze

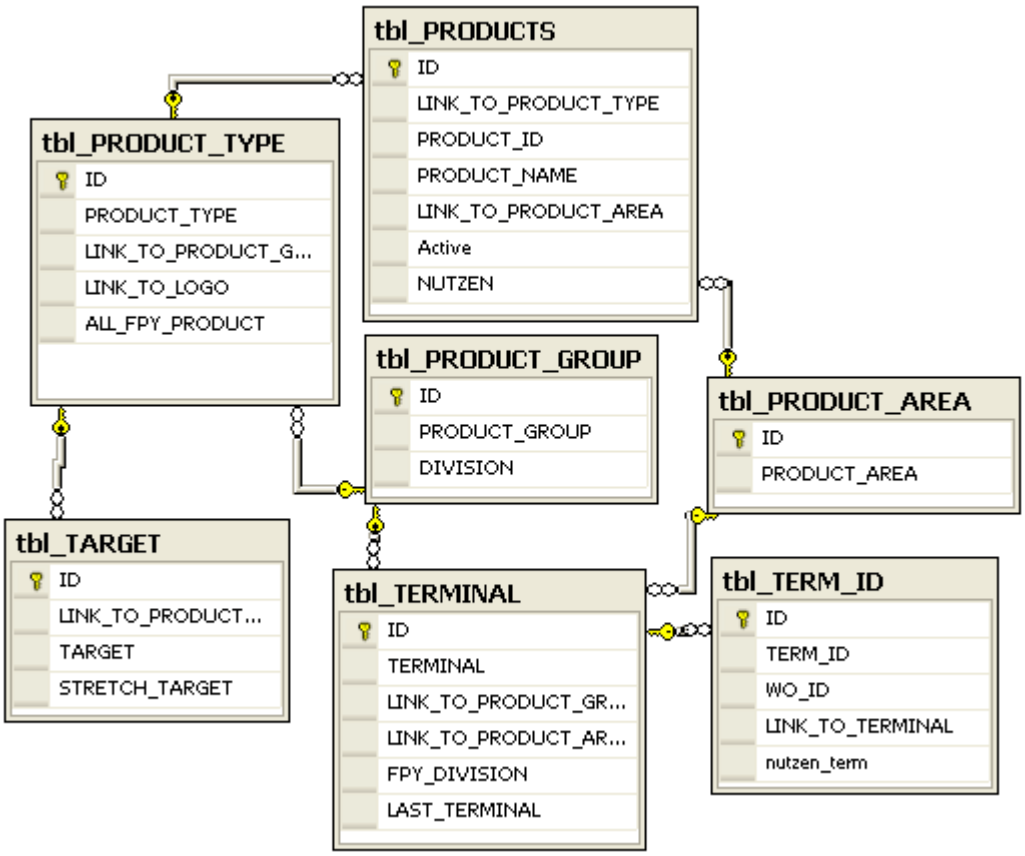
Databáze nebyla nikterak rozsáhlá, obsahovala 46 tabulek a 127 pohledů. Část tabulek je znázorněna v databázových schématech na obrázcích Obr. 5-1, Obr. 5-2 a Obr. 5-3.



Obr. 5-1: Databázové schéma 1.

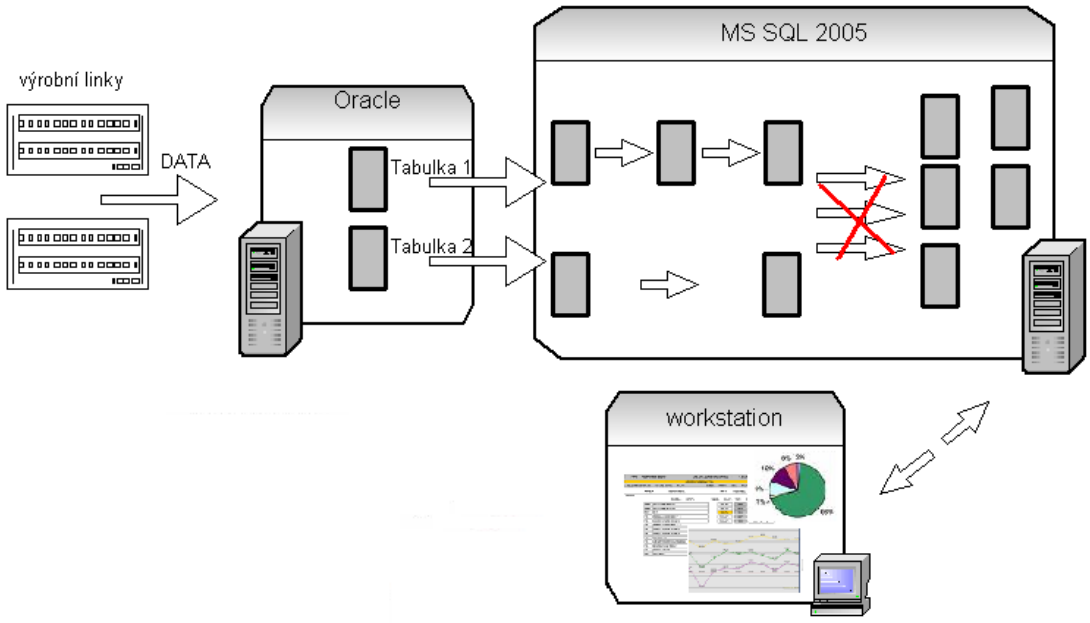


Obr. 5-2: Databázové schéma 2.



Obr. 5-3: Databázové schéma 3.

Dále databáze obsahovala další 2 velké tabulky, které nejsou zobrazeny ve schématech. V těchto tabulkách se ukládala historie výpočtů a obsahovaly řádově milióny řádků. Ostatní tabulky sloužily pro pomocné výpočty, nebo k vytváření reportů. Následující Obr. 5-4 naznačuje princip stahování a aktualizaci dat. Červený kříž znázorňuje místo, kde se procedury zastavily. Šedivé čtverečky představují jednotlivé tabulky.



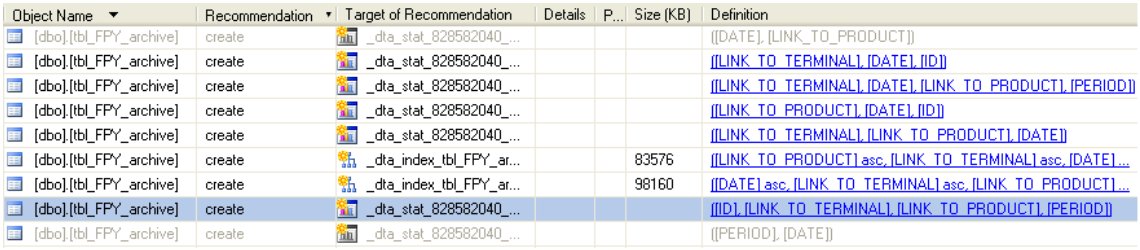
Obr. 5-4: Princip stahování a aktualizace dat.

5.3 Kroky optimalizace

5.3.1 Použití Profileru

Dříve, než jsem se seznámil s problematikou indexů, mi bylo doporučeno použít Profiler na zjištění, zda-li na serveru neběží něco, co by jej nadměrně vytěžovalo. Po odchycení aktivity serveru pomocí šablony standard, která běžela několik dní a po jejím následném zkoumání, jsem zjistil, že nejvíce prostředků, spotřebovávají procedury z mojí databáze.

Teprve po seznámení s funkcemi indexů a zjištěním, že databáze žádné indexy neobsahuje kromě clusterovaných, byl dále použit Profiler. V tomto nástroji jsem zvolil šablonu tuning. Ze zachycené stopy byly odečteny doby trvání jednotlivých procedur. Časy jsou zobrazeny v Tab. 5-1. Označení P1-P6 představuje název procedury a K1-K4 krok optimalizace. Tato stopa byla také použita v nástroji Databáze Engine Tuning Advisor, který vygeneroval sadu doporučení, která obsahovala vytvoření 17 statistik a sedm neclusterovaných indexů. Část návrhu je vidět na Obr. 5-5.



Object Name	Recommendation	Target of Recommendation	Details	P...	Size (KB)	Definition
[dbo].[tbl_FPY_archive]	create	_dta_stat_828582040_...				([DATE], [LINK_TO_PRODUCT])
[dbo].[tbl_FPY_archive]	create	_dta_stat_828582040_...				([LINK_TO_TERMINAL], [DATE], [ID])
[dbo].[tbl_FPY_archive]	create	_dta_stat_828582040_...				([LINK_TO_TERMINAL], [DATE], [LINK_TO_PRODUCT], [PERIOD])
[dbo].[tbl_FPY_archive]	create	_dta_stat_828582040_...				([LINK_TO_PRODUCT], [DATE], [ID])
[dbo].[tbl_FPY_archive]	create	_dta_stat_828582040_...				([LINK_TO_TERMINAL], [LINK_TO_PRODUCT], [DATE])
[dbo].[tbl_FPY_archive]	create	_dta_index_tbl_FPY_ar...			83576	([LINK_TO_PRODUCT] asc, [LINK_TO_TERMINAL] asc, [DATE] ...
[dbo].[tbl_FPY_archive]	create	_dta_index_tbl_FPY_ar...			98160	([DATE] asc, [LINK_TO_TERMINAL] asc, [LINK_TO_PRODUCT] ...
[dbo].[tbl_FPY_archive]	create	_dta_stat_828582040_...				([ID], [LINK_TO_TERMINAL], [LINK_TO_PRODUCT], [PERIOD])
[dbo].[tbl_FPY_archive]	create	_dta_stat_828582040_...				([PERIOD], [DATE])

Obr. 5-20: Návrh na nové struktury v databázi.

Po aplikování jsem opět spustil aktualizaci dat a také Profiler k opětovnému zachycení stopy a odečtení hodnot jednotlivých procedur. Zda–li došlo ke zlepšení výkonu, je vidět v Tab. 5-10 a v grafu Graf 5-3. Celkový čas běžících procedur se zkrátil o140 s.

Tab. 5-10: Doby trvání procedur v jednotlivých krocích optimalizace

	K1	K2	K3	K4
P1	166	164	172	36
P2	126	2	2	4
P3	54	29	29	18
P4	18	18	9	7
P5	575	584	9	7
P6	85	85	13	11
∑[s]	**1023**	**883**	**235**	**83**

5.3.2 Použití DMV

Dále byly použity dynamické pohledy, pomocí kterých byly zobrazeny další návrhy na chybějící indexy viz. Tab. 5-11.

Tab. 5-2: Návrh indexů pomocí DMV.

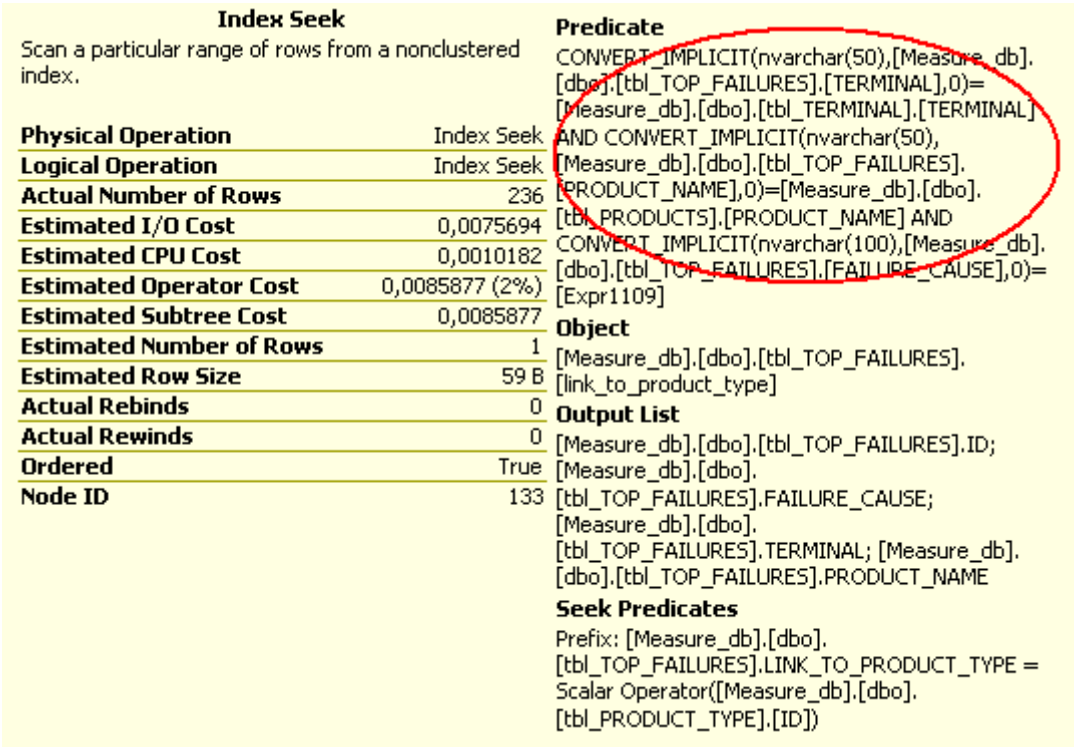
table name	equality columns	inequality columns	included_columns
tbl_failures_info_extend	NULL	[date]	NULL
tbl_failures_info_extend	NULL	[date]	[id]
Tab_FAILURES_archive	NULL	[PRODUCT_ID]	[ID], [SERIAL]
Tab_FAILURES_archive	NULL	[EVENT_DATE], [F_CAUSE]	[PRIMARY_KEY], [TERM_ID], [PRODUCT_ID], [WO_ID],
tbl_TOP_FAILURES_INFO	NULL	[DATE]	[LINK_TO_FAILURE], [TERM_ID], [PERIOD], [FAILURE_COUNT]
tbl_TOP_FAILURES_INFO	NULL	[DATE]	[ID], [LINK_TO_FAILURE], [TERM_ID], [PERIOD], [FAILURE_COUNT]
tbl_TOP_FAILURES	[LINK_TO_PRODUCT TYPE]	NULL	[ID], [FAILURE_CAUSE], [TERMINAL], [PRODUCT_NAME]
tbl_FPY_archive	NULL	[DATE]	NULL
tbl_FPY_archive	NULL	[DATE]	[ID]
Tab_EVENTS_archive	NULL	[PRODUCT_ID]	[ID], [SERIAL]
Tab_EVENTS_archive	NULL	[EVENT_DATE]	[TERM_ID], [PRODUCT_ID], [WO_ID], [PASS_COUNT]
tbl_FAILURES_archive_only_first_entrys	[PRIMARY_KEY]	[EVENT_DATE]	NULL
tbl_FAILURES_archive_only_first_entrys	[PRIMARY_KEY]	[EVENT_DATE], [F_CAUSE]	NULL
tbl_FAILURES_archive_only_first_entrys	NULL	[EVENT_DATE]	[PRIMARY_KEY], [TERM_ID], [PRODUCT_ID], [WO_ID]

Opět byly změřeny doby trvání procedur, stejným způsobem jako v předešlém případě. Dle Tab. 5-1 a grafu Graf 5-1 lze říci, že indexy, které byly vytvořeny pomocí DMV, jsou mnohem účinnější a efektivnější než ty, které navrhnul nástroj Databáze Engine Tuning Advisor, který je přímo určen pro vytvoření indexů a statistik.

Barevné řádky v tabulce označují sjednocení indexů, které jsem provedl dle svého uvážení, protože tabulky v databázi se neustále aktualizovaly, nechtěl jsem vytvářet příliš mnoho indexů, aby nepůsobily spíše jako další zátěž databáze. Pomocí dalšího dynamického pohledu, který byl ukázán v kapitole 4.6 jsem zkontroloval, zda-li se vytvořené indexy využily, či nikoli.

5.3.3 Plán Provádění

Dále jsem prozkoumal plán provádění každé procedury, pomocí kterých jsem vytvořil další indexy. Ty mi nenabídl Profiler ani DMV, pouze jsem usoudil, dle svého uvážení a postupným prohlížením plánu, že by bylo vhodné jej vytvořit. V plánech provádění jsem také odhalil konverzi některých datových typů. Obr. 5-6 naznačuje, jak taková konverze vypadá. Později byla odstraněna vhodnějším datovým typem.



Index Seek

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Number of Rows	236
Estimated I/O Cost	0,0075694
Estimated CPU Cost	0,0010182
Estimated Operator Cost	0,0085877 (2%)
Estimated Subtree Cost	0,0085877
Estimated Number of Rows	1
Estimated Row Size	59 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	133

Predicate

CONVERT_IMPLICIT(nvarchar(50),[Measure_db].[dbo].[tbl_TOP_FAILURES].[TERMINAL],0)=[Measure_db].[dbo].[tbl_TERMINAL].[TERMINAL] AND CONVERT_IMPLICIT(nvarchar(50),[Measure_db].[dbo].[tbl_TOP_FAILURES].[PRODUCT_NAME],0)=[Measure_db].[dbo].[tbl_PRODUCTS].[PRODUCT_NAME] AND CONVERT_IMPLICIT(nvarchar(100),[Measure_db].[dbo].[tbl_TOP_FAILURES].[FAILURE_CAUSE],0)=[Expr1109]

Object

[Measure_db].[dbo].[tbl_TOP_FAILURES].[link_to_product_type]

Output List

[Measure_db].[dbo].[tbl_TOP_FAILURES].ID; [Measure_db].[dbo].[tbl_TOP_FAILURES].FAILURE_CAUSE; [Measure_db].[dbo].[tbl_TOP_FAILURES].TERMINAL; [Measure_db].[dbo].[tbl_TOP_FAILURES].PRODUCT_NAME

Seek Predicates

Prefix: [Measure_db].[dbo].[tbl_TOP_FAILURES].LINK_TO_PRODUCT_TYPE = Scalar Operator([Measure_db].[dbo].[tbl_PRODUCT_TYPE].[ID])

Obr. 5-6: Odhalení nekompatibilních dat pomocí plánu provádění.

5.3.4 Použití Statistics io a statistics time

Pomocí příkazu statistics IO on a Statistisc Time on, jsem si při provádění procedur zobrazil podrobné informace, jako byl čas provádění, spotřeba CPU a vstupně výstupní informace. Zaměřil jsem se na tabulky, které vykazovaly nejvyšší počet V/V a pokusil se je optimalizovat pomocí dalších indexů.

5.3.5 Materializované pohledy

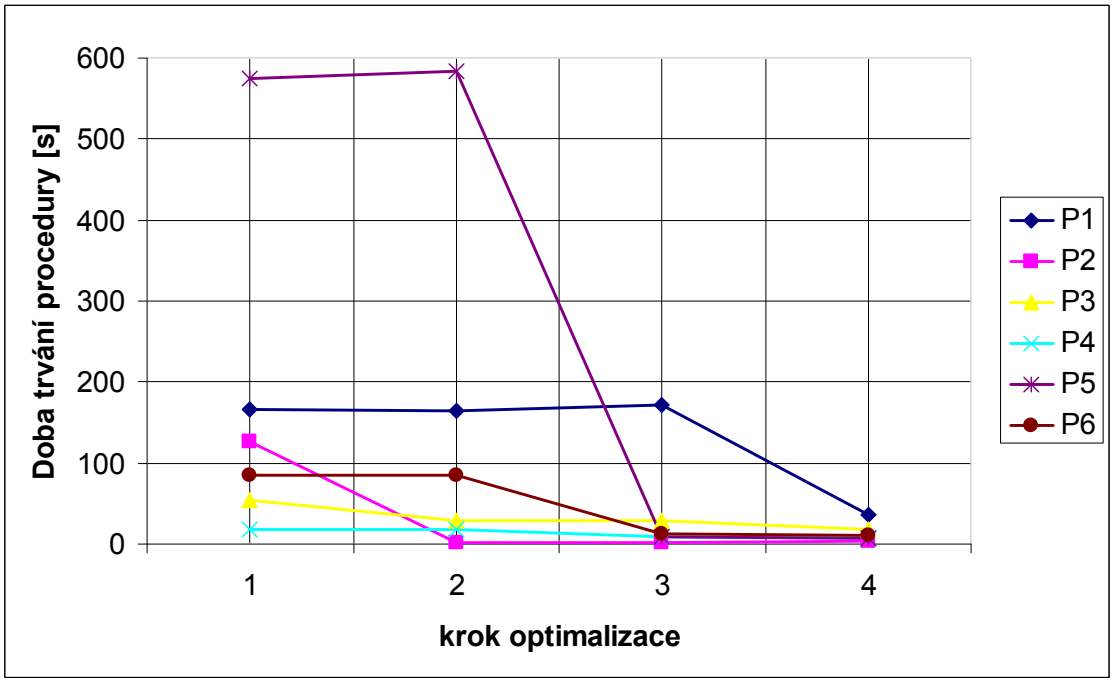
V databázi jsem vytvořil několik materializovaných pohledů. Vycházel jsem z toho, že materializované pohledy, jsou vhodné pouze u pohledů, které jsou vytvořeny z tabulek, které se často neaktualizují. Efektivita těchto pohledů byla ukázaná v kapitole 2.1, kde byl také popsán příklad i s tabulkou, která zobrazovala ušetřené prostředky.

5.3.6 Celkové zlepšení

Po odstranění chybné konverze, po přidání indexů za pomocí statistik a plánů provádění a po vytvoření materializovaných pohledů, jsem naměřil celkovou optimalizaci. Ta je vidět v tabulce Tab. 5-1 ve sloupci K4.

Celkový čas všech procedur byl vyladěn na 83 s, což je necelá minuta a půl z původních 17 minut. Výsledná čísla jsou průměrná. Graf 5.1 zobrazuje celkové výsledky.

Z tabulky Tab. 5-1 a grafu Graf 5-1 lze také vyčíst, že některé přidané indexy nepatrně zpomalily v určitém kroku procedury P1 a P5, které vkládaly data do určité tabulky. Tato tabulka byla dále zdrojovou pro další procedury, které byly výrazně urychleny pomocí nových indexů.



Graf 5-1: Doby trvání procedur

6. Metodika provádění testů

Testy probíhaly na notebooku s procesorem Dualcore, kde byl současně provozován MS SQL Server Standard Edition na operačním systému Windows XP Professional. Naměřené hodnoty byly vyčteny z jednotlivých plánů dotazů, bylo zde použito zobrazení statistik pomocí klíčových slov `set statistics io on` a `set statistics time on`. V teoretické části byl každý test spuštěn 5x, uvedené hodnoty jsou průměrné. Dále vždy byla resetována cache pomocí procedury `dbcc freeproccache` a vymazán buffer pomocí procedury `dbcc dropcleanbuffers`, tím bylo dosaženo přesnějších výsledků. Naměřené hodnoty při analýze praktické části byly získány pomocí opětovné aktualizace dávky dat, po sobě jdoucích sedmi dní, ze kterých byla opět vypočtena průměrná hodnota. Uvedené příklady se mohou na jednotlivých počítačích lišit.

Závěr

Cílem diplomové práce bylo se seznámit s možnostmi optimalizace relačních databází vytvořených v prostředí MS SQL Server 2005 a získané poznatky aplikovat na databázi ve společnosti Continental Automotive Czech Republic s.r.o.

K optimalizaci byl použit nástroj Profiler, který sleduje aktivity mezi klientem a serverem. Pomocí tohoto nástroje byly zachyceny aktivity serveru a uloženy do tabulky. Z této tabulky byly později odečítány doby jednotlivých procedur, které tvořily největší zátěž databáze, avšak plnily hlavní úlohy aplikace a to stahování a aktualizace dat z jiných databázových zdrojů. Dále byly zachycené stopy pomocí Profileru použity v nástroji Databáze engine tuning advisor, který je přímo určený k optimalizaci databází a je jako součást softwarové výbavy dodáván společně s databázovým prostředím MS SQL Server 2005. Tento nástroj vygeneroval sadu doporučení v podobě indexů a statistik a vzhledem k tomu, že verze SQL serveru 2005 byla standard, nevygeneroval již návrhy na vytvoření materializovaných pohledů a oddílů. Materializované pohledy na verzi standard lze však také vytvořit, ale za pomocí pokynů, které byly rozebrány v teoretické části této práce. V aplikaci tak bylo vytvořeno několik materializovaných pohledů, které nástroj Engine tuning advizor ze zmiňovaného důvodu nenabídl. Dále byly použity dynamické pohledy, které se ukázaly jako velmi silným nástrojem, pro sledování stavu serveru nebo databáze a lze pomocí nich diagnostikovat výkonnostní potíže. Pomocí plánu provádění, byly odstraněny některé chybové konverze datových typů, které byly vytvořeny původním autorem databáze. Pomocí všech těchto nástrojů byla databáze zoptimalizována a celkové časy běžících procedur se snížily více jak desetkrát.

Dle mého názoru, a získaných zkušeností, lze říci, že nástroj Databáze engine tuning advisor se moc neosvědčil jako vhodný nástroj pro ladění výkonu, i když je k tomu přímo určený. Charakterizoval bych jej spíše tím, že jeho návrhy na zlepšení jsou pouze orientační a nikoliv zásadní.

Seznam použité literatury a odkazy

[1] Table and Index Organization. *MSDN* [online]. 2009 [cit. 2009-02-20]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms189051.aspx>.

[2] WHALEN, Edward, et al. *Microsoft SQL SERVER 2005 : Velký průvodce administrátora*. Jiří MAtoušek; Jakub Mikulaštík. 2008. Vydání první vyd. [s.l.] : Computer Press a.s., 2008, 2008. 1080 s. ISBN 978-80-251-1949-5.

[3] *Course 2073B: Programming a Microsoft SQL Server 2000 Database, Online Learning CD-ROM* [online]. c2002 [cit. 2009-02-30]. Dostupný z WWW: <http://www.microsoft.com/learning/en/us/syllabi/2073b.aspx>.

[4] Creating Indexes with Included Columns. *Microsoft TechNet* [online]. 2008 [cit. 2009-03-25]. Dostupný z WWW: <http://technet.microsoft.com/en-us/library/ms189607(SQL.90).aspx>.

[5] DBCC SHOW_STATISTICS. *MSDN* [online]. 2009 [cit. 2009-03-30]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms174384.aspx>.

[6] Dynamic Management Views and Functions (Transact-SQL). *MSDN* [online]. 2009, vol. 2009 [cit. 2009-03-30]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms188754.aspx>.

[7] SQL Server 2005 Performance Dashboard Reports. *Microsoft download center* [online]. 2009 [cit. 2009-04-01]. Dostupný z WWW: <http://www.microsoft.com/downloads/details.aspx?FamilyId=1d3a4a0d-7e0c-4730-8204-e419218c1efc&displaylang=en>.

[8] Query Hint (Transact-SQL). *MSDN* [online]. 2009 [cit. 2009-04-03]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms181714(SQL.90).aspx>.

[9] Table Hint (Transact-SQL). *MSDN* [online]. 2009 [cit. 2009-04-10]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms187373(SQL.90).aspx>.

[10] Using Missing Index Information to Write CREATE INDEX Statements. *MSDN* [online]. 2009 [cit. 2009-04-15]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms345405(SQL.90).aspx>.

[11] FRITCHEY, Grant . *Dissecting SQL Server Execution Plans*. [s.l.] : [s.n.], 2008. 182 s. ISBN 978-1-906434-01-4.

[12] Optimize Parameter Driven Queries with the OPTIMIZE FOR Hint in SQL Server. *MSSQLTIPS* [online]. 2007 [cit. 2009-04-25]. Dostupný z WWW: <http://www.mssqltips.com/tip.asp?tip=1354>.

[13] BRUST, Andrew J., FORTE, Stephan. *Mistrovství v programování SQL Serveru 2005*. 2006. vyd. [s.l.]: Computer press, a.s., 2006. 847 s. ISBN 978-80-251-1607-4.

[14] Microsoft: Microsoft Developer Network, Dostupný z WWW: http://msdn.microsoft.com/cs-cz/default.aspx

[15] Sitansu S. Mittra, Databáze Performance Tuning and Optimization, Springer; 1 edition (December 13, 2002). ISBN-10: 0387953930, ISBN-13: 978-0387953939

[16] STANEK, William R. *Microsoft SQL Server 2005 : Kapesní rádce administrátora*. Luděk Horčička. 2006. vyd. [s.l.] : Computer press, a.s., 2006. 542 s. ISBN 80-251-1211-X.

[17] *SQLserverCentral.com : Articles* [online]. c2002-2009 [cit. 2009-05-10]. Dostupný z WWW: <www.sqlservercentral.com>.

Seznam obrázků

Seznam tabulek

Příloha A: Obsah přiloženého CD

DP/DP_radek_hatle.pdf

Tento dokument ve formátu pdf.