VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

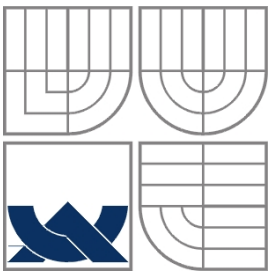
ANALÝZA A OPTIMALIZACE DATABÁZOVÝCH SYSTÉMŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

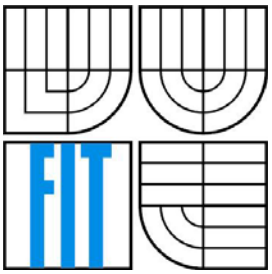
AUTOR PRÁCE
AUTHOR

Bc. JAN TŘETINA

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA A OPTIMALIZACE DATABÁZOVÝCH SYSTÉMŮ

DATABASE SYSTEMS ANALYSIS AND OPTIMIZATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JAN TŘETINA
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. JAROSLAV ZENDULKA, CSc.
SUPERVISOR

BRNO 2008

Abstrakt

S růstem požadavků na rychlost a dostupnost informačních technologií získává proces optimalizace čím dál větší váhu. Ať už se jedná o optimalizaci stránek pro vyhledávače (SEO), optimalizaci běhu operačního systému či optimalizaci aplikací (programového kódu), vždy je cílem získat rychlejší, co nejmenší (případně maximální) a lépe spravovatelné řešení.
V této práci pojednávám o optimalizaci databázových systémů, jež zahrnuje ladění od nejnižší úrovně databáze – fyzickou organizaci dat a indexů, přes ladění systému řízení báze dat (SŘBD), až po ladění nejvíce vytěžujících dotazů. Konkrétně jsem se zaměřil na optimalizaci databázových systémů Microsoft SQL Server 2005 v enterprise prostředí.

Klíčová slova

databáze, optimalizace, systém řízení báze dat, datová struktura, přístupová metoda, index, Microsoft SQL Server, Oracle Database, Sybase Adaptive Server Enterprise, monitorování

Abstract

With the increase of demands for the speed and availability of the information technologies, the process of optimization gains more and more importance. Concerning search engine optimization, optimization of operating systems or application optimization (source code), the goal is to gain faster, smaller and more maintainable solution.
In my thesis I deal with optimization of database systems, which includes low level of database tuning – physical organization of data and indices, database management system tuning and query optimization. I focused on optimization of Microsoft SQL Servers 2005 in enterprise environment.

Keywords

database, optimization, Database Management System, data structure, access path, index, Microsoft SQL Server, Oracle Database, Sybase Adaptive Server Enterprise, monitoring

Citace

Třetina Jan: Analýza a optimalizace databázových systémů. Brno, 2008, diplomová práce, FIT VUT v Brně.

Analýza a optimalizace databázových systémů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Ing. Jaroslava Zendulky, CSc.
Další cenné informace a literaturu mi poskytl Ing. Věnek Otevřel.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Třetina
19. května 2008

Poděkování

Na tomto místě bych chtěl poděkovat panu Jaroslavu Zendulkovi za cenné připomínky a důslednou kontrolu textu a dále Věnkovi Otevřelovi za možnost realizace této diplomové práce a za poskytnutí vhodné literatury. V neposlední řadě je mou milou povinností také poděkovat rodičům za nevšední podporu při studiu.

Obsah

1 Úvod

Bez databáze se v dnešním světě neobejdeme. Slouží nám především k úschově a správě dat. Setkáme se s ní v mnoha odvětvích lidské činnosti, od jednoduché kartotéky pacientů až po databáze ve finančnictví obsahující obrovské množství neustále aktualizovaných dat. Databáze vyžadují stále větší nároky především na její zabezpečení a vícenásobný přístup, jenž ovšem způsobuje časté prodlevy, které je snahou minimalizovat. Proto je nutné se zaměřit na optimalizaci databází a systémů, na nichž jsou data uložena a vyhledávána.

V této diplomové práci pojednávám o tzv. relačních databázových systémech, které mají společný jmenovatel v podobě výskytu základního prvku tabulky neboli relace. Databázi pak tvoří soubor různých tabulek, které jsou mezi sebou provázány, standardem je jazyk SQL. Další způsob úschovy dat a manipulace s nimi tvoří objektově orientované databáze, založené na principech objektového programování. Základním stavebním kamenem zde nejsou tabulky ale objekty, které mají atributy (vlastnosti) a metody. Mezi nejznámější objektově orientované databázové systémy patří ODBII, Jasmine, Oracle a DB2 (poslední dva zmiňované patří do kategorie objektově relační databáze – snaha sjednotit rysy relačních i objektových databází), standardem je ODMG 3.0 (z r. 2001). Oproti tomu deduktivní databáze jsou založené na principu logického programování a i jejich použití je úplně jiné. Najdou uplatnění především v aplikacích dolování dat a umělé inteligence. Základním prvkem je relace (napevno definované vztahy mezi entitami) a deduktivní pravidla. Příkladem deduktivní databáze je Datalog, který vychází z jazyka Prolog. Pokud je požadavkem zpracovávat aktuální a zároveň stará data, pak najde uplatnění temporární databáze, která je hojně využívána v bankovních a burzovních systémech. Předchozí typy databází jsou časově nezávislé (netemporární) – všechna data jsou platná v době dotazu. Zavedením časových razítek bude mít každý údaj časové označení a lze tak vytvářet dotazy na různá časová období. Tyto databáze představují rozšíření předchozích typů, standardem je jazyk TSQL.

V první části práce jsem se zaměřil na obecné databázové systémy, zejména na optimalizaci výkonu systému řízení báze dat (SŘBD). SŘBD představuje rozhraní mezi aplikačními programy a uloženými daty, zahrnuje tedy několik programů, které řídí organizaci, zabezpečení, způsob uložení, správu a získávání dat z databáze. Nejprve jsem pojednal o způsobu fyzického uložení dat a indexaci, poté jsem se zaměřil na základy dotazování a jejich optimalizaci a následně jsem probral optimalizaci z pohledu celé databáze. Dále jsem již popisoval konkrétní databázové systémy a to Microsoft SQL Server 2005, Oracle verze 9i a vyšší a Sybase 15.x. Uvedl jsem odpovídající nástroje a techniky pro analýzu, správu a optimalizaci výkonu.

Ve druhé části práce již popisuji mé praktické poznatky z optimalizace Microsoft SQL Serverů v podnikovém prostředí. Tyto produkční databázové servery mají vysoké požadavky od zákazníka, a proto byla prvním důležitým krokem domluva tohoto náročného procesu z důvodu co nejméně

narušeného běhu. Po ustanovení pevně rozhodnutých kroků jsem mohl vytvořit tzv. monitorovací strategii. Nejprve jsem ověřil konfiguraci serverů a nejvyšší úroveň databázového prostředí, poté jsem sledoval činnost na daných serverech pomocí čítačů a zaznamenání zátěže v podobě provedených dotazů a dávek. Následně jsem určil ty nejvíce vytěžující, jež pak byly podrobeny optimalizaci, kdy jsem na testovacím serveru zkoušel různé varianty ladění za účelem snížit dobu provádění, případně snížit počet využitých systémových prostředků (CPU, paměť, V/V subsystém).

V závěrečné části této práce shrnuji dosažené výsledky optimalizace, zdůvodňuji případné zlepšení či zhoršení odezvy a celkové rychlosti systémů a pojednávám o možnostech rozšířené správy a automatizace několika serverů dohromady.

2 Úvod do databázových systémů

Databáze je uspořádaná množina informací (dat) uložená na paměťovém médiu nebo (především v minulosti) na papíru v podobě kartoték. Správa tohoto historického způsobu uložení byla ovšem obtížná a vše bylo řízeno člověkem. Proto bylo žádoucí převedení zpracování dát na stroje, jako první paměťové médium byl použit děrný štítek. Nás budou zajímat databáze uložené pomocí bitů a bytů v počítači, databáze pak představuje kolekci dat na disku, která je fyzicky uložena v jednom nebo více souborech na databázovém serveru. Může se jednat o centralizované umístění na jednom stroji, popř. o distribuované rozložení na několik serverů, zpravidla rozmístěných geograficky.

Databáze je složena z různých fyzických a logických struktur, přičemž nejznámější logickou strukturou je tabulka. V případě fyzické struktury se jedná o soubory, které se dělí dle dat v nich uložených na databázové soubory (databázová data, metadata) a neméně důležité nedatabázové soubory (inicializační parametry, protokolovací informace atd.).

Databázi může použít člověk nebo počítačový program k získávání dat formou dotazů. Prostředníkem, umožňující tuto komunikaci, je systém řízení báze dat (dále jen SŘBD). Jedná se o počítačový program, jenž má na starost kompletní správu databáze. Typickými představiteli jsou Oracle, Microsoft SQL Server, IBM DB2, Sybase, MySQL a dal. SŘBD je tedy kolekce programů, zahrnující:

- modelovací jazyk – definuje schéma databáze dle daného datové modelu
- dotazovací jazyk – umožňuje interakci s databází, dolovaní a případnou změnu dat, nastavení zabezpečení a jiné.
- datové struktury – způsob uložení záznamů, souborů a objektů
- transakční mechanismus – ACID podmínka, datová integrita

Dle vztahu databáze a SŘBD rozlišujeme architekturu PC file – server (SŘBD běží na každém PC a sdílená data spravuje souborový server, každá tabulka je uložena v samostatném souboru) a klient – server (sdílený SŘBD představuje databázový server, který spravuje a poskytuje databázové služby, celá databáze je uložena v jednom, popř. několika souborech operačního systému).

Databáze většinou představuje hlavní přínos pro firmy, je na nich založen celý informační systém podniků, a proto je důležité ji mít dobře spravovanou a optimalizovanou.

2.1 Požadavky databázových systémů

V dnešní době se enormně zvyšuje množství dat uložených v databázích a tím i množství přístupů k nim. To má za následek zvýšení nároků a požadavků na celé databázové systémy, především jejich architekturu, návrh a dostatečně rychlé transakční zpracování. Je třeba vyvíjet a nasazovat adekvátní systémy, jež budou především splňovat tyto vlastnosti:

- Vysoká dostupnost – jedná se o online systémy a je proto žádoucí, aby systém reagoval v reálném čase, tzn., aby byl dostupný kdykoliv. Tento požadavek lze realizovat replikacemi a použitím technologie cluster – přenést zátěž na několik homogenních systémů a tím minimalizovat výpadek jednoho z nich.
- Vysoká spolehlivost – systém musí přesně odpovídat výsledkům všech transakcí, jež se vykonaly. Transakce musí být správně a bezchybně naprogramovány, musí zvládat souběžný běh a interní komunikaci modulů při jejich vykonávání. Nesmí být zapomenut žádný potvrzený výsledek, naopak musí být zajištěna návratnost, pokud jedna ze souběžných transakcí předčasně skončí.
- Vysoká výkonnost – velké podnikové systémy (enterprise) mají mnoho uživatelů, kteří přistupují k databázi, je proto nutné, aby systém byl schopen obsloužit mnoho transakcí za sekundu. Tento požadavek lze realizovat spolehlivým souběžným zpracováním transakcí, sekvenční běh je nevyhovující.
- Nízká odezva – systém musí reagovat na požadavky rychle, uživatel nemá čas na čekání, a tudíž vysoká odezva musí být eliminována. V některých aplikacích se v případě neadekvátní odezvy transakce nevykoná správně. Jedná se pak o tzv. přísné reálné (hard real-time) omezení.
- Vysoká životnost – transakční systémy jsou komplexní složité systémy s obtížným eventuálním nahrazením. Musí být proto vyvíjeny tak, aby jednotlivé hardwarové či softwarové aktualizace byly lehce nasaditelné (ať už z důvodu vyšší výkonnosti nebo funkcionality), ale bez vnitřních zásahů do systému.
- Bezpečnost – systémy zahrnují transakce, jež obsahují privátní informace, jako jsou např. osobní údaje, klíčové položky firmy, finanční záznamy a jiné. Je proto důležité zajistit bezpečnost těchto systémů, protože přístup do nich může mít mnoho uživatelů z mnoha různých míst. Uživatelé musí byt autentifikováni, následně autorizováni, tzn., že mohou provádět pouze transakce, ke kterým mají práva. Informace v databázi musí být chráněna, popř. šifrována, jejich přenos mezi systémy musí být zajištěn proti odposlouchávání.

3 Teorie optimalizace databázových systémů

3.1 Úvodem

Ladění výkonu je částí životního cyklu každé databázové aplikace a přichází na řadu především v případě, kdy je výkon aplikace omezen operacemi prováděné aplikací. Optimalizace databáze zahrnuje aktivity vedoucí ke zvýšení výkonu celého databázového systému a k maximálnímu využití systémových prostředků. Součástí je především optimalizace dotazů, s ní úzce související konfigurace databázových souborů, dále optimalizace SŘBD a operačního systému a hardwaru, na kterém SŘBD běží.

Prvním krokem ladění výkonu systému bývá tzv. V/V optimalizace neboli správné využití a nastavení vstupně – výstupních prostředků. Ty většinou představují nejnáročnější operace při práci s databází a jedná se o první problém, který optimalizátor řeší. V/V optimalizace zahrnuje především fyzickou organizaci dat a způsob indexování. Cílem návrhu databáze je tedy, aby fyzická omezení (propustnost V/V operací, velikosti paměti, výkon dotazů) neměly vliv na provádění funkcí aplikace.

Další možností optimalizace je ladění SŘBD, které zahrnuje konfiguraci paměti, procesoru a dalších zdrojů daného počítače. Následuje samotná údržba databáze, která zahrnuje zálohování, aktualizaci statistik a defragmentaci dat uvnitř databázových souborů.

Konkrétní metody budou popsány v následujících kapitolách (čerpáno z [1] a [2]), nejprve ovšem uvedu způsob ukládání dat v databázových systémech a typy indexování. Správné porozumění fyzické organizaci dat bývá základem pro řešení její následné optimalizace.

3.2 Fyzická organizace dat a indexování

Deklarativnost SQL jazyka je jedna z hlavních výhod při řešení optimalizace. SQL dotaz popisuje informaci uloženou v databázi, ale již nespecifikuje způsob, jak bude daný výraz proveden. Tuto skutečnost má na starost SŘBD. Využívá k tomu především:

- struktury úložiště – popis, jak je daný řádek tabulky uložen v souboru
- indexy – pomocné datové struktury uložené v oddělených souborech umožňující rychlý přístup k záznamům
- přístupové metody – technika pro přístup k množině řádků dané tabulky využívající algoritmus založený na způsobu uložení dat a na výběru mezi dostupnými indexy.

Poslední technika má velký vliv na výkon. V závislosti na použitém přístupu může vykonání dotazu trvat od pár sekund až po několik hodin, především v případě, kdy dotazována tabulka obsahuje tisíce záznamů. Existuje mnoho přístupových technik a každá se hodí na různé SQL dotazy.

3.2.1 Disková organizace

Databáze používají jako paměťové médium disková úložiště namísto operační (hlavní) paměti zejména z důvodu velikosti (databáze dosahují GB až TB), ceny (cena za MB operační paměti je několikrát vyšší než disková) a stálosti (oproti informacím uložených v paměti jsou data na discích stálá i v případě ztráty napájení apod.). Disky mají ovšem i nevýhody, mezi hlavní patří nízká rychlost oproti CPU. Je proto důležité se zaměřit na optimalizaci toku dat mezi hlavní pamětí a diskem, používat rychlou vyrovnávací paměť – cache, dále snaha snížit latenci ukládáním jednotlivých tabulek pospolu z důvodu pravděpodobného přístupu k jednotlivým záznamům za sebou, používat větší bloky databáze (indexy nejsou rozdělovány a umožní držet více dat v paměti po delší dobu) a efektivně ukládat data na úrovni bloků (dbát na velkou hustotu zaplnění bloku daty). Obecně je tedy hlavním cílem minimalizovat přístupy na disk.

Správce databáze musí být schopen spravovat fyzické datové soubory v databázi tak, aby zajistil vysokou úroveň výkonnosti, dostupnost a možnosti obnovy databáze. Znamená to rozdělit datové soubory na různé fyzické disky a dosáhnout tak zvýšení výkonu V/V operací. Platí zde pravidlo, že více menších disků je lepší než jeden o vysoké kapacitě. Disky se sdružují do tzv. RAID – vícenásobné diskové pole nezávislých disků, které zabezpečují pomocí určitých speciálních funkcí koordinovanou práci dvou nebo více fyzických diskových jednotek. Zvyšuje se tak výkon a odolnost vůči chybám nebo ztrátě dat. Existuje celkem šest typů polí, v databázových systémech se doporučuje používat kombinace pole 0 (prokládání dat) a 1 (zrcadlení), obecně se nedoporučuje umísťovat soubory na systémy typu RAID 5 (distribuované ukládání paritních informací).

3.2.2 Datové struktury

3.2.2.1 Hromada (heap)

Jedná se o nejjednodušší datovou strukturu, neuspořádaný datový soubor. Záznamy jsou umístěny na konec souboru bez určení, kam budou přesně uloženy, jsou tedy standardně nesetřízené. Logické adresy záznamů v datovém souboru obsahují id řádku (RowId), které se skládá z čísla stránky v souboru a tzv. čísla slotu, identifikující řádek v dané stránce. Tato datová struktura s sebou přináší nevýhodu v podobě uvolněných míst ve stránkách souboru při operacích delete a insert, vzniká tak fragmentace. Proto se provádí tzv. kompakce souborů – odstranění volných děr v souboru.

Hromada je efektivní struktura, pokud dotazy probíhají nad celou tabulkou, tzn., že jsou zahrhnuty všechny záznamy a nezáleží na pořadí jejich výběru. Z toho ovšem vyplývá i další nevýhoda – když je požadována pouze určitá množina řádků, musí se prohledávat celá tabulka.

3.2.2.2 Sekvenční (setříděné) soubory

Zmiňovanou nevýhodu hromady lze řešit řazením vkládaných údajů v závislosti na specifikovaném atributu, popř. množině atributů. Záznamy jsou uspořádány podle hodnoty tzv. vyhledávacího klíče (search key). V případě takového uspořádání, kdy jsou logicky svázané záznamy umístěny fyzicky blízko u sebe, hovoříme o shlukování (clustering). Klíč pro shlukování se pak označuje jako shlukovací klíč (cluster key). Tato technika ukládání umožňuje použít binární vyhledávání. Klasické sekvenční vyhledávání při dotazu používající klauzuli WHERE (nalezení záznamu s přesnou hodnotou atributu) prochází soubor od začátku. Až když je procházející hodnota vyšší než hledaná, lze dotaz ukončit s výsledkem nenalezeno. Tento postup, jenž je uplatněn především v heap souborech, může být velice zdlouhavý, především pokud se hledaná hodnota nachází na konci souboru. Průměrný počet stránek k nalezení hodnoty je v tomto případě S/2 (kde S je počet stránek). Oproti tomu binární vyhledávání začne procházet soubor od prostřední stránky, hodnota vyhledávacího klíče je porovnána s Id této stránky, pokud je menší, pak se proces rekurzivně opakuje v první polovině datového souboru. Nejhorší možný počet stránek potřebných k nalezení řádku činí $\log_2 S$. Tento údaj ovšem nevystihuje časovou složitost tohoto přístupu. Při binárním prohledávání se totiž mohou navštěvovat stránky umístěné na různých diskových cylindrech a dochází tak k latenci. Proto se využívá setřídění souborů pomocí indexu, které se pak prohledávají binárně. Indexy jsou totiž uloženy v hlavní paměti, a proto nedochází k diskové latenci.

Problém u setříděných souborů nastává při vložení nového záznamu. Všechny následující řádky se v datovém souboru musí posunout o jeden „slot“, což v průměru dělá polovinu stránek. Částečné řešení spočívá v ponechání prázdných slotů v každé stránce. Hodnota zaplnění pak udává v procentech počet obsazených slotů pro danou stránku. Z toho všeho vyplývá, že údržba těchto souborů může být hodně nákladná, zvlášť když se do souboru hodně zapisuje. Je totiž požadováno, aby všechny záznamy byly uloženy v souvislém diskovém prostoru z důvodu sekvenčního prohledávání.

3.2.3 Indexy a jejich druhy

Index je velmi důležitá logická struktura, využívající se k vyhledání odpovídajícího řádku (záznamu) bez nutnosti procházet celou tabulku a tím dochází k redukci času potřebného na vykonání daného dotazu. Hledaný atribut je sloupec (popř. několik sloupců) indexované tabulky a nazývá se vyhledávací klíč (pomocí této hodnoty budeme tedy přistupovat k požadovaným záznamům). Na rozdíl od kandidátního klíče nemusí mít unikátní hodnotu, tzn., že mohou existovat duplicitní vyhledávací klíče. Indexované položky mohou být uloženy např. ve struktuře B+ strom, kdy průchod indexem pro vyhledání klíčové hodnoty řádku vyžaduje velmi malý počet V/V operací.

Indexy mohou byt součástí datového souboru obsahujícího tabulku, pak se jedná o indexsekvenční soubor, nebo mohou být uloženy zvlášť do odděleného souboru, nazývaného indexový soubor.

Hustý index (dense) je index s položkami obsahujícími všechny hodnoty vyhledávacího klíče vyskytující se v datovém souboru, jeho záznamy jsou tedy v poměru 1:1 se záznamy v datovém souboru. Řídký index (sparse) je realizován nad seřazeným souborem a neobsahuje všechny hodnoty vyhledávacího klíče (chybějící se dohledají díky uspořádání). Víceatributové indexy umožňují jemnější vyhledávání.

SQL standard neposkytuje prostředky pro vytváření a správu indexů, tyto mechanismy jsou součástí většiny databázových systému, kdy se např. při vytvoření tabulky automaticky vytvoří index pro primární klíč.

3.2.3.1 Víceúrovňové indexy

Binární prohledávání záznamů indexu je mnohem rychlejší než binární hledání v seřazeném datovém souboru, protože datové záznamy jsou mnohem větší než samotné indexy. Počet V/V operací k nalezení položky indexu můžeme ještě snížit indexováním samotného seznamu položek indexu. Zhotovíme řídký index a získáme tak víceúrovňový index. Listy pak mohou odkazovat na datové záznamy v odděleném souboru nebo je přímo obsahují, pak se jedná o shlukující (clustered, primární) index, neshlukovaný (unclustered) je často označován jako sekundární index, v tomto případě se jedná o neseřazené záznamy v datovém souboru.

ISAM je index-sekvenční přístupová metoda založená na víceúrovňovém indexu, v podstatě se jedná o primární a tedy o shlukující index nad setříděnými záznamy, které jsou na úrovni listů, tzn., že ISAM je struktura úložiště pro datový soubor.

B+ strom je nejvíce používaná indexová struktura, kdy listy stromu mohou obsahovat přímo datové záznamy (jako ISAM) anebo odkazy na tyto položky, jedná se pak o sekundární index, datový soubor nemusí být setříděn. B+ strom je na rozdíl od struktury ISAM vyvážený (délka od kořene po listy je stejná v každé větvi), má odlišné sousední vazby na úrovni listů a umožňuje dynamickou samostatnou změnu stromu.

3.2.3.2 Hašovaný index

Technika hašování je využívaná jako vyhledávací algoritmus v mnoha počítačových aplikacích. V databázových systémech je použita pro indexování, kdy hašovaný index dělí indexové záznamy podle dat v tabulce do podmnožin, zvaných sektory (bucket) v závislosti na hashovací funkci h. Adresa jednotlivých sektorů, do kterých je vložen nový indexový záznam, je vypočtena pomocí funkce h aplikované na vyhledávací klíč: k->h(v) je adresa sektoru. Každý sektor je uložen ve stránce identifikované h(v). Indexový záznam může obsahovat jednotlivé datové položky (datový soubor se pak skládá ze sekvence sektorů), popř. odkazy na ně (oddělený indexový soubor sektorů).

Rozlišujeme statické hašování – velikost hašované tabulky je konstantní, používá pevně danou hašovací funkci, a dynamické hašování – automatická změna počtu sektorů při přidávání či odebrání řádků, používá rozšiřitelné a lineární hašování.

3.2.3.3 Speciální indexy

Doposud popsané indexové struktury jsou nejvíce používané a lze je využít v mnoha případech. Existují ovšem situace, kdy je nelze použít ať už z důvodu špatného využití diskového prostoru nebo nadměrného vytížení procesoru. Pak najdou uplatnění speciální typy indexů.

Rastrový (bitmapový) index je realizován pomocí jednoho nebo více bitových vektorů, jehož délka je stejná jako počet řádků v indexované tabulce. Je vhodný pro atributy, jež mají malou kardinalitu (nízký počet různých hodnot), tudíž je lze použít např. nad sloupcem pohlaví, který může nabývat pouze dvou hodnot (muž, žena). Velikost místa k uložení takovéhoto indexu je velmi malá a mohou tak být uloženy přímo v hlavní paměti, lze pak použít rychlé sekvenční vyhledávání. Velká síla těchto indexů spočívá ve vyhodnocování podmínek propojených pomocí AND, OR a NOT. Speciálním typem bitmapových indexů jsou spojovací (bitmap join) indexy, které slouží ke zrychlení konkrétních spojení. Součástí jejich definice je, které tabulky a přes jakou podmínku budou spojovány. Často jsou využívány pro spojení faktové a dimensionální tabulky v datovém skladu.

Dalším typem indexu je index s reverzním klíčem, jenž najde uplatnění především v prostředí OLAP (Online Analytic Processing). Jednotlivé bajty hodnoty klíče indexu jsou zde uloženy v opačném pořadí. Indexy s využitím funkcí jsou podobné standardním indexům s tím rozdílem, že transformace sloupců je uložena v indexu a ne ve sloupcích samotných. Lze je využít např. v případech, kdy jsou jména uloženy v databázi s různou velikostí písmen.

3.2.3.4 Jaký index vybrat

Každý index umožňuje zvýšit výkon jednotlivých skupin dotazů. Je tedy důležité vybrat vhodný index pro danou skupinu, k tomu je ovšem nutné znát typy dotazů pro určitou aplikaci a s jakou četností se vykonávají. Vytvoření indexu pro zřídka vykonávané dotazy je nežádoucí a může naopak snížit výkon.

Obecně platí tyto zásady použití indexu:

- tabulky se spoustou řádků
- sloupce často používané v dotazech
- sloupce používané při spojování tabulek (podmínka join)
- sloupce v dotazech GROUP BY a ORDER BY
- index na atribut, který je kandidátním klíčem, umožňuje lépe kontrolovat jedinečné omezení
- shlukující index na atribut použitý při vyhledání určitého rozsahu hodnot umožňuje rychleji získat výsledné prvky

3.2.4 Přístupové metody

Přístupové metoda (access path) je technika pro přístup k množině řádků dané tabulky. Dle způsobu uložení dat a výběru mezi dostupnými indexy se rozlišuje několik variant přístupu.

První metodou je table scan, která přistupuje k řádkům pomocí přímého prohledávání zdrojové tabulky, tzn., že pro každý řádek tabulky se otestuje, jestli splňuje podmínku v klauzuli WHERE. Tato metoda se užije především tehdy, kdy nelze použít alternativní metody (nelze použít existující index), v případě velkého množství dat v tabulce (podmínce bude vyhovovat 90% dat) anebo z důvodu starých statistiky (tabulka nebyla analyzována).

Další možností je RowId Scan, který přistupuje k řádku přes jeho RowId, jedná se o nejrychlejší možný způsob. Parametr RowId určuje datový soubor a blok ve kterém je řádek umístěn a také pozici řádku uvnitř tohoto bloku. RowId scan obdrží adresy požadovaných řádků (tyto jsou většinou obstarány předchozím index scanem) a poté se na základě těchto RowId naleznou samotné řádky.

Metoda Index Scan vybírá data z indexu. Jeho výstupem jsou většinou RowId požadovaných řádků, které jsou dále zpracovávány RowId scanem. RowId scan ale nemusí následovat v případě, že dotaz přistupuje pouze ke sloupcům, které můžou být získány přímo z indexu. Rozlišujeme typy index scanu:

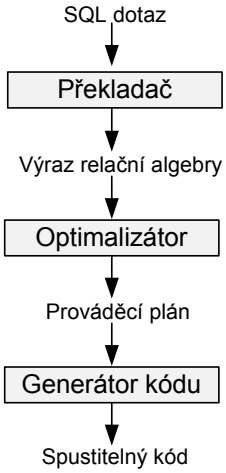
- Index Unique Scan – vrací vždy nejvýše jedno RowId a je tedy prováděn tehdy, pokud podmínka klauzule WHERE zaručuje přístup k jedné řádce (např. pokud klademe podmínku na primární klíč pomocí rovnosti).
- Index Range Scan – provádí se tehdy, pokud klademe v klauzuli WHERE ohraničující podmínky na indexované sloupce (nerovnosti, operátor LIKE).
- Fast Full Index Scan – realizuje se, pokud index obsahuje všechny sloupce užívané v dotazu. Přistupuje se pouze k datům v indexu bez přístupu do vlastní tabulky.

3.3 Základy a optimalizace dotazování

Minimálně 50% problémů s výkonem databázových aplikací souvisí s návrhem, vývojáři totiž většinou neznají všechny způsoby využívání dat a všechny obchodní procesy, které budou v aplikaci implementovány. Někdy je náprava relativně rychlá formou změny inicializačního parametru, přidáním indexu, popř. přeplánováním déle trvající operace, jindy je ovšem nutné upravit architekturu aplikace. Aplikační návrhář by tak měl rozumět principům a základním metodám procesu dotazování pokud chce zdokonalit databázový systém.

3.3.1 Architektura zpracování dotazu

Po spuštění SQL dotazu se nejprve provádí analýza překladačem (parser) SŘBD. Ten ověří syntaxi a typovou korekci a pomocí systémového katalogu rozhodne, zda jsou správně uvedeny atributy a jejich případné odkazy. Výsledkem tohoto kroku je tzv. syntaktický strom dotazu, který je vstupem pro optimalizátor, jenž SQL dotaz zpracovaný překladačem optimalizuje. Během optimalizace se ohodnotí náročnost jednotlivých variant zpracování SQL dotazu a výsledkem je optimální postup zpracování dotazu - tzv. plán provádění dotazu. Jedná se o uspořádanou množinu kroků potřebných k provedení dotazu, včetně přístupových metod k tabulkám a indexům. Vybraný plán je kompilován a výsledný kód plánu daného SQL dotazu je spuštěn. Získaný výsledek dotazu pak zasílá databázový server klientovi, který SQL dotaz spustil.



Obr. 3.1 Schéma zpracování dotazu

Po analýze se tedy daný dotaz převede na výraz relační algebry. Např. dotaz:

```
SELECT k.nazev FROM oddeleni o, kniha k WHERE o.nazev_odd = 'zabava' AND
k.odd = o.id AND k.autor = Capek
```

bude převeden na výraz:

$\pi_{Nazev}(\sigma_{nazev\_odd}$ = ´zabava´ AND odd = id AND autor = ´Capek´(ODDELENI x KNIHA))

Pokud by se daný výraz počítal přímo, tak jak jsou relační operátory specifikovány, pak by se v případě velké databáze jednalo o časově velice náročnou záležitost. Problém totiž nastane u kartézského součinu ODDELENI x KNIHA. Ke zlepšení výkonu SŘBD převede daný výraz na následující:

$\pi_{Nazev}(\sigma_{nazev\_odd}$ = ´zabava´(ODDELENI ⋈ odd = id AND autor = ´Capek´(KNIHA)))

Tato transformace je založena na heuristické analýze, která tvrdí, že operace join je výhodnější než kartézský součin a že spojení malých relací je lepší než velkých. Tyto transformace a odhady, jež jsou součástí prováděcího dotazovacího plánu, jsou vykonány modulem SŘBD optimalizátor dotazů.

3.3.2 Výpočet základních operátorů a funkcí SQL

Výpočet projekce, sjednocení a rozdílu není složitý, v případě projekce lze prohledat relaci a vymazat nechtěné sloupce. Situace se ovšem zesložití v případě použití dotazu s parametrem DISTINCT, kdy musíme odstranit duplicitní položky, které mohou být ve výsledku. Existují dva způsoby nalezení dvou shodných záznamů – již popsané hašování a externí řazení.

Řazení je jeden z nejefektivnějších způsobů jak odstranit duplicitní záznamy a je také základem mnoha algoritmů join. Když jsou datové soubory velké a jsou uloženy v externích úložištích (disk), pak se používá tzv. externí řazení. Typický algoritmus zahrnuje dva kroky: částečné seřazení a spojení. Hlavní princip spočívá v přenosu souborů do hlavní paměti, jejich následnému seřazení pomocí známých algoritmů (např. Quicksort) a poté k zapsání výsledku zpět na disk. Dochází tak k vytváření seřazených souborových segmentů, které musí být následně spojeny do jednoho samostatného souboru. Tento algoritmus je hojně používán, protože funguje téměř ve všech případech a nepotřebuje speciální datové struktury. Pokud jsou ovšem k dispozici, je možno použít jinou metodiku s nižší výslednou cenou, např. řazení B+ stromů, který je vhodný použít v případě, kdy se jedná o shlukující soubor.

Výpočet selekce je ovšem mnohem složitější. Způsob jeho výpočtu závisí na druhu podmínky použité při selekci a na fyzické organizaci dotazované relace. Jednoduchá podmínka je daná formou *atribut* operátor *hodnota*, kde operátor je jeden z množiny typů porovnání =, >, < anebo obsahuje klíčové slovo like. Klasický způsob výpočtu této selekce je procházet celou relaci a kontrolovat danou podmínku pro každý pár. Někdy ovšem není nutné procházet celou relaci. Rozlišujeme případy, kdy atribut nemá vlastní index anebo existuje B+ stromový, resp. hašovaný index na daný atribut. V případě podmínky na rovnost (atribut = hodnota) je nejúčinnější hašovaný index, kdy je nutné procházet celou relaci, pokud je ovšem seřazena dle atributu, pak lze použít binární vyhledávání k nalezení stránek relace, jež vyhovuje dané podmínce. V ostatních případech (nerovnost, rozsahový dotaz) je lepší B+ index, který využijeme k nalezení prvního záznamu relace, jenž odpovídá podmínce. Složitější podmínky zahrnují konjunkci a disjunkci.

Výpočet agregačních funkcí (AVG, COUNT) zahrnuje kompletní prozkoumání výstupu dotazu. Lze využít metody řazení, hašování a indexování.

Výpočet operace join lze realizovat pomocí jednoduché vnořené smyčky (nested loops), kdy je jedna z tabulek zvolena jako "vnější" (driving table) a jedna jako "vnitřní". Spojení pak probíhá tak, že pro každý řádek vnější tabulky jsou nalezeny odpovídající řádky tabulky vnitřní. Další metodou je Sort merge join, kdy jsou obě tabulky setříděny podle sloupců, které slouží jako klíče ke spojení a následně jsou výsledky spojeny. Posledním způsobem je Hash join, který je vhodný pro spojování velkých množin dat. Principem je zhotovení hašovací tabulky z menší ze spojovaných množin. V případě, že data jsou setříděna, je lepší ovšem použít Sort merge join.

3.3.3 Optimalizátor

V předchozí kapitole byla vysvětlena architektura zpracování dotazu s implementací jednotlivých základních operací jazyka SQL. Nyní se zaměřím na principy optimalizace dotazů, které lze uplatnit při tvorbě a efektivnější implementaci SQL dotazů. Jak již bylo uvedeno, dotaz musí být převeden na výraz relační algebry, který již může být vyhodnocen přímo algoritmy, jež byly uvedeny dříve. Problém ovšem nastává s rychlostí provedení těchto výrazů. Zde přichází na řadu optimalizátor dotazů, jenž určuje způsob vyhodnocení každého SQL dotazu a jehož úkolem je snížit prodlevu vyhodnocení a vytvořit tzv. prováděcí plán dotazu. Skládá se z modulů pro transformaci SQL, generátor a výběr prováděcího plánu, výběr zpracování dle statistik a ceny operace a modulu pro optimalizaci za běhu.

Prvním krokem procesu optimalizace je tedy úprava příkazu a jeho transformace, následně se zvolí typ optimalizace, dle níž rozlišujeme dva typy optimalizátorů: založené na pravidlech (rule-based, zastaralé) a na odhadu ceny vykonání výrazu (cost-based), který je založen na statistikách produkovaných SŘBD. Tento odhad je pak použit pro vytvoření plánu, který je vstupem pro překladač plánu výrazu – komponenta, jež zodpovídá za vyhodnocení dotazu dle vstupního plánu.

Dalším krokem je volba přístupové metody, která byla popsána v kap., následně se určí pořadí a metoda spojení tabulek.

3.3.3.1 Transformace SQL

Transformace SQL má za úkol vytvořit k původnímu SQL dotazu dotaz sémanticky ekvivalentní, který půjde efektivněji zpracovat. Lze použít transformace heuristické a založené na ceně. Heuristika je založená na jednoduchých a známých skutečnostech, např. že spojování menších tabulek je lepší než těch větších, výpočet spojení na rovnost (equi join) je jednodušší než výpočet kartézského součinu nebo, že výpočet několika operací během prohledání jedné relace je výhodnější než řešení pomocí několika prohledávání. Mezi tyto transformace patří především:

- úprava pohledů – transformace vytvořeného pohledu a požadovaného dotazu
- tzv. „zploštění“ dotazů – převod mnoha typů dotazů pomocí join, popř. semi-join
- generování tranzitivních predikátů – přidání dodatečného predikátu pro daný doraz
- eliminace společných podvýrazů – zjednodušení dotazu pomocí AND a OR spojek
- predikáty pushdown a pullup – přesun podmínek pohledů a poddotazů z původního dotazu ven nebo dovnitř mezi jednotlivými pohledy a poddotazy

Transformace založené na ceně porovnávají upravený dotaz s původním a podle předpokládané složitosti (ceny) vyhodnocení, je vybrán nejlepší. Zde patří:

- použití materializovaných pohledů – bývají menší a mohou obsahovat vyhodnocené agregační funkce. Transformace spočívá v nahrazování čtení z několika tabulek čtením z materializovaných pohledů, které může výrazně urychlit vyhodnocení dotazu.

- OR „rozšíření“ – přepis OR v části WHERE pomocí konstrukce SELECT-UNION-SELECT
- transformace „hvězdy“ – použití u OLAP transakcí, funguje na principu vkládání poddotazů do položeného dotazu. Vložené dotazy a bitmapové indexy umožňují efektivnější přístup k datům v tabulce faktů. Patentováno firmou Oracle.
- predikát pushdown pro vnější spojení pohledů – přesunutí podmínky do jedné ze spojovaných tabulek (při použití vnějšího spojení totiž nelze přehodit pořadí spojovaných tabulek). Vnější spojení pak lze urychlit za použití indexů.

Transformace založené na selekci a projekci využívají kaskádování selekce, resp. projekce, komutativnost podmínek, nahrazení posloupnosti selekcí/projekcí jednou selekcí/projekcí. V případě spojení tabulek (join) se používá asociativnost a komutativnost jednotlivých operátorů.

3.3.3.2 Odhad parametrů provedení dotazu

Prováděcí plán je relační výraz s konkrétní odhadovací metodikou přiřazenou ke každé operaci. Pro lepší představu je vhodné si znázornit dotaz ve formě stromu, kde každý vnitřní uzel představuje relační operátor a každý list pak daný relační název. Unární relační operátor má pouze jednoho syna, binární dva.

Vzhledem k tomu, že výsledek jednoho výrazu může být vstupem pro další a také z důvodu přímé závislosti ceny na velikosti vstupu, je důležité přesně odhadnout velikost výstupu daného výrazu. K tomu musíme znát vztah pro výpočet tzv. váhy atributu A (weight), který udává průměrný počet uspořádaných n-tic, jež vyhovují různým hodnotám atributu A. Systémový katalog obsahuje množinu statistik pro každou relaci R:

- Blocks(R) – počet bloků obsazených záznamem relace (tabulky) R.
- Tuple(R) – počet uspořádaných n-tic v záznamu R.
- Values (R.A) – počet různých hodnot atributu A prvku R.
- Max (Min) Val (R.A) – max. (min.) hodnota atributu A prvku R.

Dále potřebujeme definovat tzv. faktor redukce. Mějme výraz: *SELECT parametry FROM R1 V1, ... Rn Vn WHERE podmínka*. Faktor redukce je poměr

$$\frac{Blocks(\text{výsledná\_množina})}{Blocks(R_1)\times\ldots\times Blocks(R_n)}.$$

Pro tento výpočet potřebujeme znát velikost výsledné množiny. Tento faktor ovšem může být odhadnut pomocí struktury výrazu, aniž bychom znali velikost výsledku daného výrazu. Redukce jednotlivých částí výrazu je nezávislá na sobě a určí se jako *redukce (výraz) = redukce (parametry) x redukce (podmínka)*, kde redukce (parametry) je velikost redukce během projekce řádků v závislosti na atributech uvedených v klauzuli SELECT a redukce (podmínka) je velikost redukce během eliminace řádků, jež nevyhovují podmínce. Váha atributu A relace R se pak vypočte jako weigth(R_i.A) = Tuples (R_i) x redukce (R_i.A = value).

3.3.3.3 Výběr plánu

Počet vygenerovaných prováděcích plánů může být velice vysoký, je proto třeba snížit tento počet na relativně malou množinu, ze které je už jednodušší odhadnout cenu každého plánu a vybrat ten nejlepší. Používá se adaptivní vyhledávací strategie, která má zajistit to, že doba strávená hledáním optimálního řešení nepřesáhne zlomek doby potřebné na vyhodnocení, tzn., že pokud bude dotaz trvat 1 vteřinu, nemá smysl jej 10 vteřin optimalizovat, ovšem v případě dotazu běžícího několik hodin, je vhodné věnovat několik vteřin nebo i minut na nalezení lepšího plánu. Na začátku tvorby exekučního plánu je před samotným prohledáváním prostoru plánů použito několik heuristik, které mají šanci najít optimální plán okamžitě. Tento postup zahrnuje tři kroky: výběr logického plánu, redukce prohledávacího prostoru a volba heuristického vyhledávacího algoritmu.

3.4 Ladění databáze

Ladění databáze je především proces modifikace aplikace a nastavování parametrů SŘBD za účelem zvýšení výkonu celého systému. Chceme např. snížit dobu odezvy (= čas k vykonání určitého úkolu – SQL dotazu), zvýšit výkonnost (množství práce za jednotku času). Ladění ovšem nemění sémantiku systému, tzn., že vyladěný i původní systém vrací stejnou informaci uživatelům a nachází se ve stejném stavu při stejné posloupnosti požadavků.

Prvním krokem ladění je zjištění slabých míst sytému. Pokud se zaměříme na část systému, která je vytížená pouze z 2%, pak její vyladění zvýší výkon právě jen o 2%. Nejvyšší úroveň ladění v databázových systémech, a tedy zasluhující nejvyšší pozornost, je samotný SQL kód a schéma databáze. Jedná se o aplikačně orientovanou část, která zahrnuje především způsoby, jak budou vykonány jednotlivé SQL dotazy a které indexy budou vytvořeny. SŘBD pak představuje další pomyslnou úroveň, která zahrnuje způsob fyzického úložiště dat a správu vyrovnávací paměti. Nejnižší úroveň pak představuje hardware. Systém musí disponovat dostatkem paměti, ať už diskové či hlavní (RAM), rychlostí, počtem CPU a adekvátní síťovou infrastrukturou.

3.4.1 Disková vyrovnávací paměť

Rozdílná rychlost mezi CPU a přesunem stránek mezi ním a diskem je vysoká. Cena plánu výrazu je dána odhadem počtu přesunutých stránek, optimalizátor výrazu má pak za úkol najít plán, který minimalizuje tento počet. I když je zvolen dobrý plán, stále hraje velkou roli přenosová rychlost. Tu má za úkol zvýšit cache, neboli vyrovnávací paměť. Jedná se o rychlou hlavní vyrovnávací paměť SŘBD, ve které jsou uloženy aktuální požadované databázové stránky. Když transakce přistupuje k databázové položce, SŘBD přenese požadovanou stránku z disku do cache a zkopíruje hodnoty položky z cache do aplikačního bufferu. Stránka pak zůstává v cache, neboť je velká pravděpodobnost, že bude znovu použita (v případě aktualizace nebo čtení následující položky, jež je

umístěna ve stejné stránce). Zde se pod pojmem databázová položka představuje tabulka, popř. index, může jí být ovšem také prováděcí plán nebo uložená procedura. V tomto případě pak některé SŘBD spravují oddělenou vyrovnávací paměť nazvanou procedure cache.

Cache se může pochopitelně zaplnit a nová stránka pak musí přepsat stávající. Pokud nebyla od uložení do cache zaktualizována, pak může být jednoduše přepsána novou, mluvíme o čisté stránce (clean). Pokud ovšem došlo ke změnám v databázi, pak musí být stránka z cache nahrána zpět do databáze, jedná se o tzv. špinavou (dirty) stránku. Existuje několik algoritmů pro nahrazení stránky v cache, např. LRU (least recently used), jenž nahradí stránku, která byla nejméně použita, čímž dochází k úschově aktivních stránek v cache. Další algoritmy jsou MRU, FIFO a dal.

Pro zajištění velkého výkonu je cílem mít velký počet transakčních požadavků na stránku realizovánu přímo z cache. Většinou se jedná o 90%, tzn., že pouhých 10% přístupů není řešeno pomocí vyrovnávací paměti. Cache pak musí mít velikost řádově v MB, v případě velkých aplikací i GB.

SŘBD nabízí několik možností ladění:

- spravování hlavní cache jako několik oddělených. Administrátor pak může jednotlivé prvky databáze pevně navolit do vlastní cache a tím zajistit, že všechny stránky budou uloženy a nepřepsány.
- rozdělení cache na několik bufferů rozdílné velikosti. Výhoda spočívá v možnosti alokování různě velkých bloku paměti do odpovídající cache.
- možnost měnit algoritmus nahrazení stránky v cache. Tuto mechaniku lze použít v případě několika násobných cache a lépe tak volit prvky, jež jsou v dané cache nahrány.

Výkon systému a lepší využití procedurální cache může být také zvýšen nepoužíváním explicitních konstant v SQL příkazu. Místo pevně dané hodnoty v klauzuli WHERE (WHERE a.id = 10) zvolíme proměnnou (WHERE a.id = :klic). V prvním případě se při každém spuštění příkazu s jinou hodnou a.id mění prováděcí plán, ovšem ve druhém případě zůstává stejný.

3.4.2 Ladění schématu

Schéma databáze je základem celého systému, a pokud je dobře navrhnuto, pak mohou být SQL dotazy vykonávány efektivně. Ladění na aplikační úrovni zahrnuje především odhad velikosti tabulek, četnosti dotazů a aktualizací adresovaných databázi. Základem je také správná správa indexů, denormalizace a rozdělování (partitioning), které se využívá především v případě velkých tabulek.

V případě indexů je nutno zvážit, na který prvek databáze budou vytvořeny, protože každý index je asociován s typem úložiště a tím může docházet k velkým prodlevám. Je tedy důležité promyslet, zda je vhodné vytvořit index na tabulku, jejíž řádky jsou často přidávány nebo mazány, popř. vytvoření indexu s vyhledávacím klíčem, jenž zahrnuje často aktualizované sloupce. Nastává

otázka, zda budou tato eventuální zvýšení výkonu při zpracování dotazů dostatečně vyvážena vzhledem k nárůstu ceny při vykonání dotazů, jež modifikují tabulku.

Denormalizace je vhodná při častém používání tzv. read-only transakcí (dotazy na čtení z databáze). Dochází tak díky následné redundanci ke zvýšení výkonu, neboť např. místo dotazů nad vícero tabulkami (join) se po přidání redundantní informace ve formě nových sloupců provádí dotaz pouze nad jednou tabulkou.

Cena přístupů k obzvlášť velkým tabulkám může být snížena dělením tabulky na části. Lze tak oddělit data, která jsou často vybírána (adresována) od dat, na která je spíše jen odkazováno. „Zabalením" často dotazovaných dat do několika stránek lze redukovat počet V/V operací. Existují dva způsoby dělení – horizontální a vertikální. V prvním případě mají všechny nově vytvořené části stejnou množinu sloupců a každý obsahuje podmnožinu řádků. Např. v tabulce studenti tak lze odlišit aktivní a pasivní studenty a absolventy. Vertikální rozdělení je na základě podmnožiny daných sloupců. Tento způsob může být efektivní v případě velkého množství sloupců (velmi dlouhých řádků tabulky), nebo pokud jsou některé sloupce málo dotazovány. Bez rozdělení by byl výkon snížen potřebou přenést neaktivní data, kdy by byla dotazována pouze data aktivní. Např. databázový systém Oracle odděluje sloupce, které jsou málo dotazovány bez potřeby explicitního dělení. I když máme tabulku v BCNF formě, mohou být zřídka dotazované sloupce odděleny. Bez těchto sekundárních informací se stává hlavní část tabulky menší a dotazy odkazující na ni jsou pak mnohem rychlejší.

Rozdělením velkých tabulek tak lze dosáhnout zvýšení výkonu v multiprocesorových systémech, jednodušší správy a rychlosti přenosu dat mezi danými tabulkami (v případě vykonání dotazu *INSERT INTO SELECT FROM*). Data totiž nejsou při rozdělení přesunuta fyzicky, pouze jsou změněna metadata popisující jejich úložiště.

3.4.3 Ladění DML

Modifikace schématu databáze v případě konkrétní tabulky může mít globální vliv na výkon všech SQL příkazů, které se na danou tabulku dotazují. Oproti tomu změna výrazu nebo dotazu SŘBD má pouze lokální vliv – ovlivní výkon při vykonání pouze tohoto výrazu. Je vhodné se vyvarovat třídění a procházení celých tabulek, omezit komunikaci klient - server, popř. zvážit restrukturalizaci dotazů.

Třídění velkých dat je náročná a drahá záležitost, a proto bychom se jej měli pokud možno vyvarovat. Je tedy důležité zvážit použití klauzulí ORDER BY, GROUP BY a DISTINCT (s ním související UNION a EXCEPT), které operaci třídění vyžadují. Často je možné se jí vyhnout díky tomu, že data jsou již skutečně setříděna, a to díky přístupu podle indexu, který zajistí automaticky setřídění. Další možností je existence UNIQUE omezení, jenž odstraní nutnost třídění kvůli DISTINCT. Nenáročné utřídění malých dat na počátku vyhodnocení může také zajistit požadované utřídění celých dat. Pokud je řazení nevyhnutelné, je třeba zvážit použití shlukujícího indexu. Optimalizátor řeší problém třídění tvorbou dvou plánů vyhodnocení - obvyklý plán na základě

lokálních optimalizací a plán, který je zaměřen na využití přístupů pro eliminaci třídění a nakonec je pro oba plány spočítána celková cena. Často se plán eliminující třídění ukáže jako lepší.

Je vhodné se také vyvarovat zbytečnému procházení celé tabulky, které je nutné např. při použití výrazu „<>“ v podmínce WHERE. Komunikace klient – server je taky velice drahá záležitost, a proto je jí třeba eliminovat. Hlavním problémem je tzv. cursor, který vyvolává spojení pro každý zachycený řádek. Pokud tedy aktualizujeme záznamy v tabulce, pak je nejvhodnější použití dotazu UPDATE namísto vybrání řádků, jejich následné modifikace a konečného zápisu zpět do databáze (prohledávací varianta dotazu UPDATE). Pokud požadujeme získání agregačních informací, je vhodné zvážit počítání těchto dotazů pomocí uložených procedur a pouze celkový výsledek poslat klientovi.

3.4.4 Nástroje SŘBD

Většina SŘBD poskytuje mnoho různých nástrojů, umožňující optimalizaci a ladění databáze. Tyto pomůcky většinou požadují vytvoření tzv. modelu databáze, na kterém se budou zkoušet různé varianty plánů.

Typický nástroj SŘBD je tzv. „showplan“, který umožní uživateli zobrazit prováděcí plán dotazu a vypsat tak posloupnosti jednotlivých kroků zpracování SQL dotazu, jak je sestavil optimalizátor. Jedná se tedy především o zobrazení použitých zdrojů (tabulek, indexů) při zpracování každého kroku, použité strategie řešení a přístupové metody (pořadí spojovaných tabulek, využití indexních klíčů, strategie agregace, třídění apod.), strategie práce s datovou cache během zpracování dotazu. Příkaz pro jeho vytvoření (popř. uložená procedura) není součástí standardu jazyka SQL, každý výrobce tudíž používá odlišnou syntaxi.

Většina výrobců poskytuje grafické rozhraní pro zobrazení prováděcího plánu. Například IBM má Visual Explain pro DB/2, Oracle poskytuje Oracle Diagnostic Pack a MS SQL Server obsahuje SQL Profiler. Tyto nástroje neukazují pouze prováděcí plány, ale umí navrhnout i indexy pro zrychlení různých dotazů. Provedením prováděcího plánu se objeví tzv. hinty, udávající míru vhodnosti použití vytvořených indexů. Pokud je index používán jen v jednom dotazu, je možné jej zrušit bez velké ztráty výkonu a pomocí právě této nápovědy je možné tento experiment snadno provést, bez smazání a nového vytvoření indexu.

Některé SŘBD umožňují programátorům vkládat tyto návrhy přímo do SQL příkazů a ty pak optimalizátor může použít při vytváření prováděcího plánu. Jedná se např. o správné pořadí tabulek při spojení join v klauzuli FROM, specifikace join techniky (hašování, sort–merge) nebo druh indexu při paralelním zpracování dotazů.

3.4.5 Řízení fyzických zdrojů

Fyzické zdroje přístupné pro SŘBD jako jsou CPU, V/V a síťová zařízení jsou důležitým faktorem výkonu aplikace. Většinou ovšem databázoví administrátoři, popř. programátoři nemají na ně přímý vliv, a tak některé SŘBD poskytují podporující mechanismy pro jejich řízení.

Výkon disků vedoucí ke zlepšení diskových operací (přístupů k datům) může být zvýšen zapojením do diskových polí, tzv. RAID, viz kapitola. SŘBD umožňuje uživateli specifikovat, na jaký disk se daná položka umístí. Rozdělení tabulek na části podle četnosti přístupů k daným sloupcům pojednané a jejich následné umístění na různé disky zvýší výkon, neboť diskové požadavky na celou tabulku mohou být vykonány souběžně.

V případě řízení CPU se doporučuje paralelismus (zapojení více procesorů při provádění jediného příkazu). V obecných případech je pouze jeden proces či vlákno přiřazené k prováděcímu plánu daného SQL výrazu. Procesy jsou vykonávány sekvenčně – v jeden časový okamžik běží pouze jeden proces, který požaduje službu procesoru k vykonání kódu nebo požaduje V/V přenos, musí se tak čekat na dokončení každé jednotlivé operace. V případě OLAP transakcí je využití zdrojů nízké, pokud se ovšem jedná o OLTP transakce, kdy v jednom okamžiku požaduje přístup několik souběžně pracujících uživatelů, je využití zdrojů velmi vysoké a čas odezvy je pak nevyhovující. Tato skutečnost může být zlepšena použitím paralelního zpracování dotazů, kdy je několik souběžných procesů přiřazeno k vykonání různých částí prováděcího plánu. Toho lze docílit přidáním více jednotek CPU, každý procesor pak pracuje samostatně. Mechanismus SŘBD hintů může aplikační programátor použít k vyžádání paralelního zpracování. Vhodné je také použití méně rychlých procesorů (preferovat menší množství rychlejších před použitím většího množství pomalejších procesorů, operační systém bude mít na starost méně prováděcích front).

3.4.6 Správa optimalizátoru

Jak již bylo uvedeno, existují dva typy optimalizátorů – založený na ceně a na pravidlech. První typ (první implementace v Oracle 7 – 1992) používá k odhadu velikosti výstupu statistiky, které nezahrnují pouze tabulky ale také indexy, jež mohou být použity k přístupu k tabulce (šířka, počet listů stromu k uložení stránek, počet hodnot vyhledávacích klíčů na úrovni listů a dal.). Optimalizátor vyžaduje pravidelné provádění analýz tabulek a indexů v databázi, které vedou k aktualizaci statistik. Frekvence analýz by měla odrážet počet změn v objektech, u aplikací OLTP je vhodné ji vykonávat dle časového rozvrhu (např. jednou týdně).

Přesnost určení ceny dotazu zásadně ovlivňuje efektivnost optimalizátoru. Vychází ze znalosti principu všech používaných přístupových metod, statistik o objektech v databázi, statistik o výkonu použitého HW a vlivu různých optimalizací (cache, optimalizace V/V operací, paralelizmus). Rozlišujeme statistiky objektů v databázi: počet bloků a řádků v tabulce, počet úrovní v B+ stromu u indexu a statistiky pro každý sloupec sloužící pro odhad výsledku podmínky WHERE, dále

systémové statistiky: výkon HW, CPU a V/V operací, přičemž kombinace těchto hledisek není na základě pevného vzorce, ale pozorovaného chování systému za běžné zátěže. Posledním typem jsou uživatelské statistiky, které zahrnují uživatelem definované funkce a doménové indexy, optimalizátor díky nim dokáže pracovat s uživatelskými funkcemi a indexy stejně jako s vlastními.

Statistiky lze získat ve formě histogramu popisující rozdělení hodnot ve sloupcích. Bez něho může optimalizátor zjistit pouze počet řádku a maximální, minimální a průměrnou hodnotu záznamů pro daný sloupec, SQL výraz dotazující se na konkrétní hodnotu pak musí projít celou tabulku. S histogramem, jenž obsahuje počet řádků pro každou hodnotu daného sloupce, optimalizátor lehce zjistí, že pro určitý výraz odpovídá přesný počet řádků a lze tak použít přístupovou metodu s indexem namísto procházení cele tabulky. Správa histogramu je ovšem časově náročný proces, SŘBD proto nabízí možnosti volby, pro které sloupce se bude histogram vytvářet a udržovat.

Po určité době provozu se může projevit snížení výkonu i v případě stále stejné zátěže. Příčinou jsou změny v databázi, a ačkoli se velikosti tabulek jeví jako stejné, přidáváním a mazáním řádků se organizace tabulek a indexů zhoršuje. Např. u B+ stromů může v případě rozsahových dotazů dojít k nárůstu počtu listů a tím pádem roste cena procházení stromu, v případě struktury hromada zůstává prázdné místo uvolněné při smazání řádků, nové záznamy se ukládají na konec souboru, tato skutečnost také vede ke zvýšení doby procházení tabulky. Správa statistik je podobně jako správa histogramu časově náročná, proto se neaktualizuje při každé změně tabulky. SŘBD ovšem umožňuje vykonat příkazy pro znovu ohodnocení statistik, využívané především v případě, kdy dojde k velké změně stavu tabulky od posledního ohodnocení. Určení sloupců, pro které se vyplatí udělat histogram, se provede sledováním rozložení hodnot ve sloupci, popř. četností výskytu sloupce v klauzuli WHERE. Použití neaktuálních statistik vede ke špatným prováděcím plánům.

4 Nástroje a techniky analýzy a optimalizace výkonu konkrétních SŘBD

4.1 Microsoft SQL Server 2005

Microsoft SQL Server 2005 je komplexní platforma poskytující centralizovaná datová řešení. Zahrnuje mnoho navzájem závislých komponent, porozuměním těchto závislostí lze vytvářet a spravovat výkonná softwarová řešení. Je vyvíjen firmou Microsoft a aktuální verzí je SQL Server 2005 (verze 9.0), ovšem již v této době vychází nová verze 2008 (10.0). V této části práce jsem čerpal z literatury [3] a [4].

Základem SQL Serveru je nástroj pro relační databáze, který poskytuje zabezpečené rozhraní pro ukládání, čtení a modifikaci dat v relačním nebo XML formátu a zahrnuje podporu replikací a fulltextové vyhledávání. Jádro SQL Serveru dále obsahuje služby analytické (OLAP), integrační (SSIS - nástroj pro import a export dat a jejich transformaci), oznamovací (posílání upozornění v případě specifických chyb), reportovací (extrakce dat ze zdroje a vytváření reportů) a tzv. Service Broker (spolehlivý způsob komunikace mezi softwarovými službami založený na transakčních zprávách). Součástí SQL Serveru je také rozhraní .NET Common Language Runtime (CLR), umožňující vytvářet databázová řešení (uložené procedury, trigery, uživatelem definované typy a funkce) pomocí kódu napsaného v .NET jazyce, jako je např. Microsoft Visual C# .NET nebo Microsoft Visual Basic .NET.

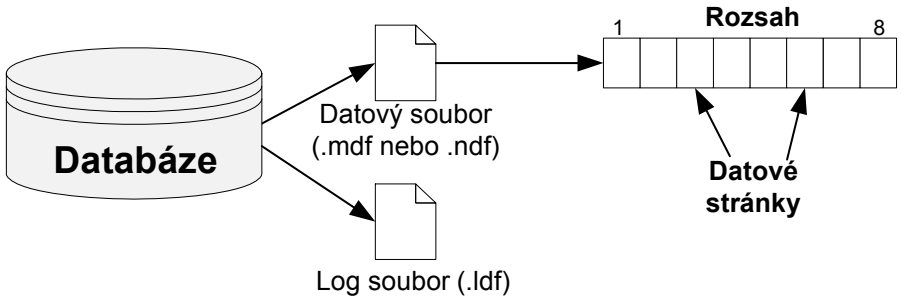
Dále SQL Server obsahuje podporu tzv. DDL trigerů, které spustí uložené procedury, když je proveden DDL příkaz (ALTER TABLE). Oproti předchozím verzím se zlepšila architektura zabezpečení, která poskytuje rozšířenou ochranu databáze pomocí tzv. principals („kdo“ má práva – skupiny osob, uživatelé) a securables (na „co“ má práva – zabezpečitelné objekty). Nově je podporován XML formát, databáze tak může použít metody k získání a úpravě dat XML pomocí XQuery dotazů. Udržovatelnost databáze je vylepšena podporou online obnovy a online operací s indexy. Nový nástroj Database Tuning Advisor umožňuje jednodušší monitorování provozu a optimalizaci databáze, server také podporuje tzv. partitioning – tabulky a indexy mohou byt horizontálně rozděleny na několik souborů (filegroups) a lze tak zvýšit výkon, udržovatelnost a jednodušeji spravovat obnovovací proces.

4.1.1 Struktury SQL Serveru

SQL Server 2005 zavádí novou logickou strukturu databáze. Jedna instance databázového serveru může obsahovat přes 30 000 databází a každá databáze může mít více než 2 miliardy objektů. Všechna data jsou obsažena v objektu databáze, ten se může skládat z několika objektů Schéma, které obsahuje tabulky, indexy a další logické objekty databáze. Nový objekt Schéma je tedy kontejnerem objektů a používá se k definici jmenných prostorů pro objekty databáze.

Nejznámější logické struktury, tabulky, jsou v databázi SQL Serveru definovány jako objekty a obsahují dvě jednotky uložení dat: datové stránky (základní jednotka uložení o velikosti 8 KB) a rozsahy (množina osmi po sobě jdoucích datových stránek, tzn. blok o velikosti 64 KB určený pro alokaci volného místa). Data v tabulkách je možno uspořádat do oddílů, které obsahují datové řádky buď v hromadě nebo struktuře shlukujícího indexu. Pro zrychlení přístupu se oddíly ukládají do více skupin souborů - filegroups. Implicitně mají tabulky pouze jeden oddíl, pokud jich má tabulka více, jsou data oddělena horizontálně.

Další důležitá struktura, index, používá stránky stejně jako tabulka, její velikost je tedy také 8 KB. SQL Server spravuje indexy ve struktuře B+ stromu a podporuje pouze dva typy indexů: shlukující (clustered) a neshlukující. Dále nabízí i zvláštní typ indexu pro XML data.



Obr. 4.1 Schéma databáze SQL Serveru

Každá databáze se skládá ze dvou souborů – datového a protokolového, viz obr. 4.1. Používají se tři typy souborů:

- Primární datový soubor (.mdf) – uchovávají data a udržují záznamy o dalších souborech. Každá databáze má jeden mdf soubor.
- Sekundární datový soubor (.ndf) – zde jsou uložena další data databáze. Databáze může obsahovat několik ndf souborů. Používají se pro rozšíření databázového prostoru a pro partitioning.
- Transakční protokol (.ldf) – obsahuje historii změn v databázi a informace potřebné pro její obnovu. Každá databáze má alespoň jeden ldf soubor.

4.1.2 Základní nástroje

4.1.2.1 SQL Server Management Studio

SQL Server 2005 poskytuje grafické rozhraní umožňující databázovému administrátorovi vykonávat každodenní úkoly rychle a pohodlně. Jedná se o SQL Server Management Studio – integrovaný nástroj pro přístup, konfiguraci, správu a administraci všech komponent SQL Serveru 2005. Umožňuje kompletní správu relační databáze, analytických, reportovacích a integračních služeb a dále poskytuje grafické rozhraní pro vytváření Transact-SQL, XMLA, MDX a XQuery dotazů. SQL Server Management Studio využívá Microsoft Visual Studio® Framework včetně funkcionality Visual Studia při vytváření dotazů či skriptů. Dále obsahuje tzv. Object Explorer – panel pro jednoduchou navigaci a správu serverů včetně příslušných databází a Solution Explorer – panel pro správu uložených projektů a uživatelských řešení. Jedna ze zajímavých nových vlastností je, že uživatel nemusí být připojen k databázi při psaní skriptů či dotazů, lze si tak přednastavit v offline režimu svá řešení a posléze je spustit na příslušných databázích.

4.1.2.2 Nástroje příkazové řádky

Pro automatizaci instalace, správu nebo údržbu pomocí skriptů jsou vhodné nástroje příkazové řádky. Základním nástrojem je SQLCMD.EXE, který umožňuje psát příkazy T-SQL, spouštět SQL dotazy, vykonávat uložené procedury a provádět další úlohy.

Mezi další nástroje patří především BCP, který představuje program pro kopírování velkého množství dat. Pro optimalizaci je vhodný nástroj DTA, který se používá k analýze pracovního vytížení a doporučuje změny v optimalizaci.

4.1.2.3 SQL Management Objects

SQL Server 2005 poskytuje API k SQL Management Objects (SMO) pro podporu běžných nebo často se opakujících administrativních úloh. Je výhodné tyto akce zautomatizovat z důvodu redukce možných chyb popř. nekonzistence dat, zvlášť když je ve správě více SQL Serverů a instancí. SMO poskytuje rozhraní pro vytváření administrativních programů a skriptů, lze jej tak využít k vytváření aplikací, jež mohou sloužit např. k získávání a modifikaci konfiguračních nastavení, vytváření nových databází, správu úloh agenta a plánování záloh. SMO je implementováno pomocí .NET knihovny (Microsoft.SqlServer.Smo.dll) a definuje hierarchii objektů a vlastních modelů. Jedná se tedy o množinu programovatelných objektů určených k psaní programů pro správu SQL objektů a úloh.

SQL-DMO (Dynamic Managment Objects) bylo vytvořeno jako základní programovací rozhraní pro SQL Server. SMO jej rozšiřuje a poskytuje zjednodušený pohled k rozhraní Windows Management Instrumentation (WMI). Lze tak využít WMI společně se SMO pro monitorování a konfiguraci SQL Serverů a instancí.

K vytváření SMO aplikací lze využít Visual Studio 2005, lze tak navrhovat klasické Windows Forms, ASP.NET Web nebo konzolové aplikace v závislosti na požadavcích.

4.1.3 Správa a optimalizace fyzických zdrojů

Jednou z možností optimalizace fyzických zdrojů je nastavení využití paměti. SQL Server upravuje své paměťové nároky dynamicky a implicitně jsou použita tato nastavení: minimální alokovaná paměť 0 MB, maximální nastavena tak, aby server dokázal využívat virtuální paměť na disku i RAM, min. množství paměti pro spuštění dotazu je 1024 KB a Address Windowing Extension (AWE) je zakázán. Obvykle se min. a max. hodnoty nenastavují. Na vyhrazeném systému, kde běží pouze SQL Server, však můžeme dosáhnout lepšího výkonu nastavením min. hodnoty na 8 MB + 24 KB x průměrný počet současně připojených uživatelů k serveru ([3]). Dynamické řízení paměti je použito i pro operace s indexy, pokud ovšem toto automatické přidělování nefunguje (zejména v případech rozdělených tabulek), pak lze nastavit určité množství paměti buď ve vlastnostech daného serveru v položce Index Creation Memory a nebo pomocí uložené procedury *sp_configure*.

SQL Server používá k optimalizaci využití procesorů afinitu procesoru a V/V operací. Tím se přiřadí procesory k určitým prováděcím vláknům, eliminuje se tak načítání procesoru a sníží se přesouvání vláken mezi procesory. V/V afinita určuje, které procesory jsou vhodné k provedení V/V operací SQL serveru. Afinity se nastavují a optimalizují automaticky při instalaci serveru, lze je však nastavit ručně ve vlastnostech daného serveru.

Paralelní zpracování dotazů provádí SQL Server v případech, kdy je počet procesorů vyšší než počet aktivních spojení a pokud jsou odhadnuté náklady na sériové zpracovaní vyšší než úroveň plánu dotazu. Implicitně je nastavení maximálního stupně paralelního zpracování na hodnotu 0, což znamená, že se řízení procesorů řídí automaticky. Pokud je nastavena hodnota 1, pak se paralelní zpracování nemá použít.

Existuje několik důležitých nastavitelných vlastností každé databáze, týkající se fyzických zdrojů a které je možné spravovat automaticky. Jedná se zejména o Auto Close (po odpojení posledního uživatele a ukončení procesu se databáze uzavře a jsou k dispozici uvolněné zdroje), Auto Create a Auto Update Statistics (SQL Server automaticky vytváří a udržuje statistiky k určení nejvýhodnějšího způsobu vyhodnocení dotazů), Auto Shrink (redukce velikosti datových a protokolových souborů) a Auto Growth (růst velikosti datového souboru v % nebo po MB).

4.1.4 Nástroje pro optimalizaci

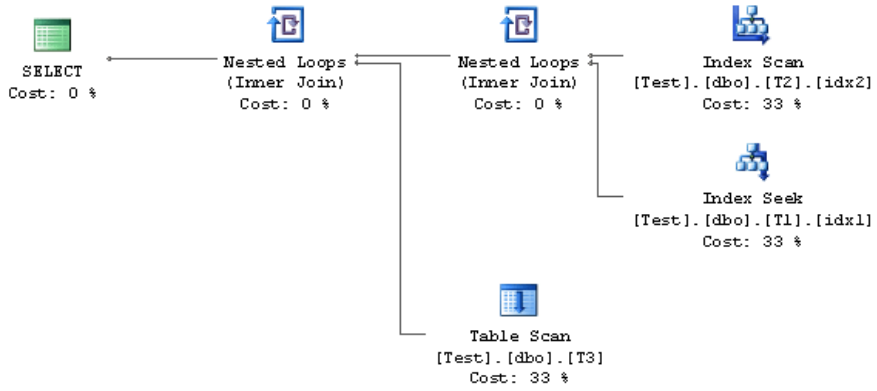
Administrátoři často potřebují monitorovat databázový server z důvodu poklesu výkonu nebo v případě identifikace problému. Pro tyto účely, ale i např. pro sledování činnosti uživatelů a řešení optimalizace lze využít nástroj SQL Profiler. Události, které se mohou sledovat v Profileru, se podobají událostem čítačů, jež se monitorují v nástroji Performance Monitor. Jsou seřazeny do skupin

nazvaných třídy událostí a je tak možné sledovat jednu nebo více tříd. Profiler lze využít mimo jiné k nalezení pomalých dotazů a ke zjištění, co zpomalení dotazů způsobuje, zobrazení prováděcího plánu, jeho exportu do XML formátu, k zobrazení a analýze uvíznutí (dreadlock), proč k němu došlo a které procesy jej vyvolaly.

Dalším vhodným nástrojem je Database Engine Tuning Advisor, jenž lze použít k usnadnění indexování a optimalizace. Před jeho spuštěním by se měl vytvořit reprezentativní vzorek činnosti v databázi, soubor s tímto vzorkem se posléze použije, vybere se databáze, popř. tabulky, které chceme analyzovat a typy doporučení, jež chceme využít (typy indexů, strategii rozdělení, maximální prostor pro doporučené struktury a dal.). Po dokončení analýzy se zobrazí průvodce doporučení, který se skládá z doporučení rozdělení a indexů a který zobrazí i odhadnutelné zlepšení v případě použití výsledků. Doporučení se mohou uložit jako soubor s SQL skriptem nebo je lze rovnou aplikovat.

Pro zobrazení prováděcího plánu lze využít příkaz SET SHOWPLAN, popř. SET STATISTICS. Rozdíl mezi nimi spočívá v tom, že SHOWPLAN odhaduje prováděcí plán optimalizátoru před spuštěním daného dotazu, zatímco příkaz STATISTICS poskytuje kompletní prováděcí plán po ukončení dotazu.

Prováděcí plán lze zobrazit také v SQL Management studiu, kde jsou přehledně zobrazeny jednotlivé kroky optimalizátoru s podrobnými informacemi o ceně a použitém algoritmu, viz obr. 4.2



Obr. 4.2 Prováděcí plán v Management studiu

4.2 Oracle Database 9i a vyšší

Oracle Database je multiplatformní databázový systém s pokročilými možnostmi zpracování dat, vysokým výkonem a snadnou škálovatelností. Jedná se tedy o SŘBD podporující relační i objektový model a nabízející odpovídající strategie pro monitorování, zabezpečení a vylaďování samostatných i síťových databází. Je vyvíjen firmou Oracle Corporation a aktuální verzí je Oracle Database 11g. V této práci se zaměřím především na předchozí verzi 10g, která dle citace [5]

„je výrazným evolučním, pokud ne rovnou revolučním, krokem od předchozí verze Oracle9i. Databáze Oracle10g není jen bohatší na funkce, ale snadněji se spravuje díky většímu množství podpůrných nástrojů.“.

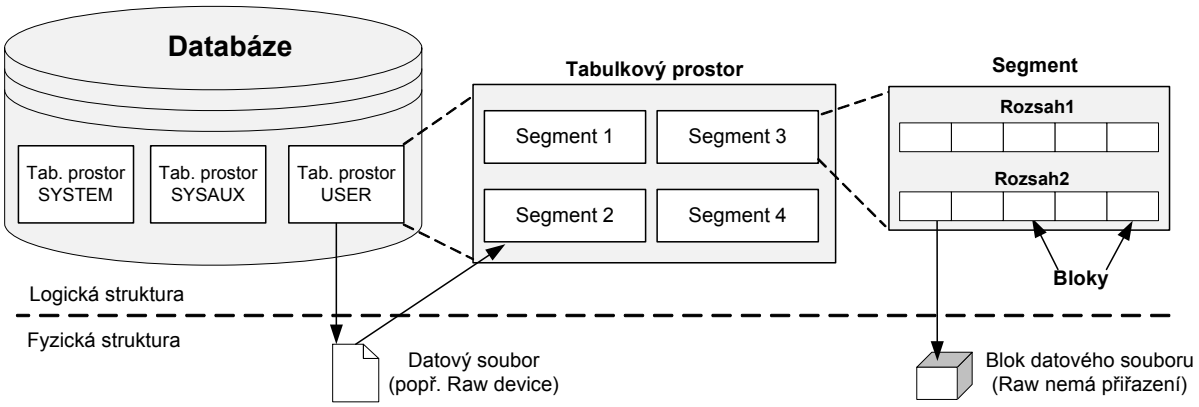
Tento systém podporuje nejen standardní relační dotazovací jazyk SQL podle normy SQL92, ale také proprietární firemní rozšíření Oracle (např. pro hierarchické dotazy), imperativní programovací jazyk PL/SQL rozšiřující možnosti vlastního SQL (v tomto jazyce je možné tvořit uložené procedury, uživatelské funkce, programové balíky a triggery), dále podporuje objektové databáze (SQL:1999) a databáze uložené v hierarchickém modelu dat (XML databáze, jazyk XSQL).

Oproti MS SQL Serveru zde představuje instance jinou roli. Vlastní databáze Oracle je uložena na pevném disku serveru, instance databáze Oracle existuje v operační paměti. Skládá se z rozsáhlého bloku paměti, která má vyhrazené místo v systémové globální oblasti System Global Area (SGA) a z procesů, které běží na pozadí a které komunikují s SGA a databázovými soubory na disku.

4.2.1 Základní struktury databáze Oracle

4.2.1.1 Logické struktury

Datové soubory databáze Oracle jsou sdruženy do jednoho nebo více tabulkových prostorů, které obsahují segmenty, jež znázorňují logické databázové struktury (tabulky, indexy). Tyto segmenty se dále dělí do rozsahů (extent) a bloků. Popsané logické rozdělení úložného prostoru umožňuje účinnější kontrolu nad využitím diskového prostoru.



Obr. 4.3 Struktura databáze Oracle

Při instalaci Oracle 10g jsou vytvořeny dva tabulkové prostory SYSTEM a SYSAUX, dále Oracle umožňuje vytvořit speciální typ prostoru s názvem „bigfile tablespace“, jehož maximální velikost může být až 8EB (10^6 TB) obsahující pouze jeden datový soubor, popř. tzv. dočasné prostory, které lze využít ke při operacích řazení nebo jako pracovní oblast při vytváření indexů (docílí se tak snížení V/V operací mezi dočasnými a trvalými segmenty, uloženými v různých tabulkových prostorech).

Nejmenší úložnou jednotkou databáze Oracle je blok, jehož velikost je udávána v bajtech. Několik bloků tvoří rozsah, jenž označuje nově alokovaný prostor při zvětšování databáze. Skupina rozsahů pak značí segment, a ty pak dohromady tvoří databázový objekt (tabulku, index). Oracle rozlišuje 4 typy segmentů: datový a indexový (úložiště tabulky, resp. indexu), dočasný (slouží pro provedení SQL příkazu, jenž vyžaduje diskový prostor, např. operace řazení) a návratový (rollback, využíván k ukládání původních hodnot při manipulacích s daty a pro udržování konzistentních pohledů na data tabulky).

Mezi základní logické databázové struktury patří tabulka, která je hlavní úložnou jednotkou databáze Oracle. Relační tabulka je zde organizována jako halda (heap), jednotlivé řádky nejsou tedy uloženy v žádném pořadí. Dočasné tabulky slouží k dočasnému uložení dat a to buď po dobu trvání transakce nebo relace. Dále Oracle poskytuje indexově orientované tabulky, které lze použít v případě tabulky s malým počtem sloupců a s přístupem přes hodnotu pouze jednoho sloupce, kdy je zbytečné vytvářet zvlášť indexy (a udržovat tak dvě úložné struktury – dat a indexů). Dalším podporujícím typem tabulky databáze Oracle je objektový, jenž se skládá z řádků, které jsou zároveň objekty. Externí tabulky umožní uživateli přistupovat ke zdroji dat (např. textový soubor) jako k tabulce z databáze, seskupené tabulky (clustered) zase umožňují zvýšit výkon dotazů v případě přístupu k více než dvěma tabulkám současně. Snižují se tak nároky na místo potřebné k uložení společných sloupců dotazovaných tabulek. Posledním typem tabulky jsou tzv. rozdělené tabulky (partitioned), které usnadňují správu rozsáhlých tabulek (větší než 2GB) rozdělením na menší oddíly.

Databáze Oracle samozřejmě podporuje i další důležitou logickou strukturu – indexy. Základním a nejobvyklejším používaným typem je jedinečný index. Využívá se pro implementaci primárního klíče nad tabulkou, zajistí nám tedy, že v indexovaném sloupci nebudou žádné duplicitní hodnoty. Pokud chceme zvýšit rychlost přístupu nad duplicitními hodnotami, lze využít tzv. duplicitní indexy (lze jej tedy vytvořit např. nad sloupcem Příjmení). Dalším typem indexu je index s reverzním klíčem, jenž najde uplatnění především v prostředí OLAP. Indexy s využitím funkcí jsou podobné standardním indexům s tím rozdílem, že transformace sloupců je uložena v indexu a ne ve sloupcích samotných. Lze je využít v případech, kdy jsou jména uložena v databázi s různou velikostí písmen. Posledním podporujícím typem databáze Oracle jsou rastrové indexy, které mimo velké úspory místa mohou urychlit dobu odezvy, neboť Oracle může ještě před vlastním přístupem do tabulky velmi rychle odstranit potenciální řádky z dotazu, který obsahuje vícenásobné klauzule WHERE.

Další logickou strukturou jsou pohledy, které umožní uživateli přístup k prezentaci dat z jedné nebo více tabulek. V Oracle existují klasické, materializované a objektové pohledy, nebudu uvádět podrobnější popis, neboť nehrají v otázce optimalizace důležitou roli.

4.2.1.2 Fyzické struktury

Databáze Oracle využívá mnoho fyzických úložných struktur pro uložení a správu dat. Datové soubory, soubory protokolu a jeho archivované soubory obsahují aktuální uživatelská data, řídicí

soubory udržují stav databázových objektů a ostatní struktury, jako jsou textové soubory s poplachy (alerts log) a trasovací soubory, obsahují protokol s informacemi o událostech a chybových stavech databáze.

Datový soubor je fyzické místo na disku, kde jsou uložena všechna data databáze. Jeden datový soubor odpovídá jednomu fyzickému souboru OS na disku a je vytvořen pro každou databázi Oracle. Každý datový soubor patří jednomu tabulkovému prostoru, ten se však může skládat z mnoha datových souborů.

Soubor protokolu logu (redo log) slouží pro záznam provedených transakcí v databázi, tzn., že je aktualizován při každém přidání, změně nebo odstranění dat v tabulce, indexu, popř. jiném objektu databáze. Každá databáze Oracle musí mít alespoň dva soubory protokolů (obvykle se používají tři), neboť tyto soubory jsou využívány kruhovým způsobem: jakmile je jeden soubor zaplněn, je označen jako ACTIVE – potřebný pro případnou obnovu instance, nebo jako INACTIVE – není třeba obnovy; záznamy se nyní přidávají do dalšího souboru a ten je označen jako CURRENT.

Archivované soubory protokolu vznikají v režimu „archivelog“, kdy se zaplněné soubory protokolu kopírují na jedno nebo více umístění a lze je použít k obnově databázových dat. V případě režimu „noarchivelog“ nejsou v případě selhání disku jednotlivé záznamy o transakcích dostupné, integrita databáze je ovšem ochráněna, neboť všechny transakce potvrzené operací commit jsou dostupné v online souborech protokolu.

Řídicí soubor obsahuje metadata databáze (data o fyzické struktuře) a další důležité informace (název a doba vytvoření databáze, umístění všech datových souborů). Alespoň jeden řídicí soubor musí obsahovat každá databáze, pro její chod je velmi důležitý a proto je vhodné jej udržovat ve více kopiích.

Inicializační soubor parametrů obsahuje umístění trasovacích a řídících souborů, archivovaných souborů protokolu, jsou zde také uložena omezení velikostí různých struktur v paměťové oblasti SGA a také informace o tom, kolik se může souběžně připojit uživatelů. Tento soubor se otevírá při spuštění instance databáze Oracle. Jedná se buď o editovatelný textový soubor (init<SID>.ora) nebo binární soubor parametrů serveru (SPFILE).

Soubory s protokoly poplachů a trasování se využívají v okamžiku vzniku chyby při běhu databáze. Pokud se jedná o chybové zprávy, pak se provede záznam do protokolu poplachů (alert log), v případě procesů běžících na pozadí do trasovacího protokolu (trace log).

Posledním typem souboru používaného v databázi Oracle je soubor hesel, který se používá pro autentifikaci správců systému při provádění administrativních úloh (vytváření, spouštění či zastavování databáze apod.). V tomto souboru jsou uloženy informace o oprávněních SYSDBA a SYSOPER, autentifikace ostatních uživatelů se provádí přímo v databázi.

4.2.1.3 Paměťová struktura

Pro uložení různých komponent instance využívá Oracle fyzickou paměť serveru, která obsahuje spustitelný kód, informace o relacích, jednotlivé procesy databáze a informace sdílené mezi procesy. Paměťové struktury obsahují mimo jiné uživatelské příkazy jazyka SQL a vyrovnávací paměť, která obsahuje datové bloky databázových segmentů (načtených z disku z důvodu provádění příkazu SELECT) a informace o dokončených databázových transakcích.

Datová oblast vyhrazená pro instanci se nazývá globální systémová oblast SGA (System Global Area). Jedná se o skupinu paměťových struktur instance Oracle sdílenou uživateli databázové instance. Při spuštění instance je oblasti SGA vyhrazena paměť v závislosti na nastavení v inicializačním souboru parametrů. Tato paměť je alokována po jednotkách zvaných granule, jež mohou mít velikost 4 nebo 16MB. SGA obsahuje několik typů vyrovnávací paměti, např. vyrovnávací paměť knihoven, datového slovníku nebo protokolu.

Pro každý server a proces běžící na pozadí v paměti existuje globální programová oblast PGA (Program Global Area). Jedná se o oddíl paměti alokovaný pro privátní použití jedním procesem. Spustitelné kódy jsou spuštěny jako součást instance Oracle a jsou uloženy v oblasti pro softwarový kód. Tyto oblasti jsou statické, jsou umístěny v privilegované paměťové oblasti odděleně od ostatních uživatelských programu a mění se pouze s instalací nové softwarové verze.

4.2.2 Správa objektů a fyzických zdrojů

Oracle poskytuje mnoho nástrojů pro rozvržení a kontrolu velikosti datových souborů databáze. Je vhodné nastavit tabulkový prostor tak, aby se v případě potřeby rozšiřoval automaticky. K tomu lze použít příkaz ALTER DATABASE (popř. TABLESPACE) nebo nástroj EM Database Control, který umožňuje jak změnu velikosti a aktivaci automatického zvětšování velikosti datového souboru, tak přidání dalšího souboru k tabulkovému prostoru.

Důležitou roli ve výkonnosti hraje i velikost objektů, která se určuje především z důvodu předběžného určení požadovaného místa a minimalizace nevyužitého místa. Z důvodu špatného nastavení velikosti rozsahů může dojít k výraznému snížení výkonu (především při prohledávání celé tabulky), a proto je vhodné vytvářet rozsahy podstatně větší než velikost V/V vyrovnávací paměti, bude tak potřeba velmi málo dalších operací čtení (v případě velikosti vyrovnávací paměti 128 KB je možné zvolit velikost rozsahů 128, 256, 512 KB atd.).

Časté operace nad tabulkou (vkládaní, aktualizace, mazání) mohou po nějaké době způsobit fragmentaci a je tedy nutné provést defragmentaci segmentu (segment shrink), která zpřístupní volné místo v segmentu ostatním segmentům v tabulkovém prostoru. Pro určení segmentů, které vyžadují defragmentaci, lze využít nástroj Segment Advisor, který provede analýzu trendu růstu nad vybranými segmenty. Po jeho spuštění můžeme výsledky zobrazit pomocí pohledu datového slovníku

DBA_ADVISOR_FINDINGS, popř. pomocí DBA_ADVISOR_RECOMMENDATIONS, jenž obsahuje informace o doporučené operaci a možné úspoře místa.

Architektura Automatic Storage Management (ASM) zvyšuje výkon automatickým rozmístěním databázových objektů mezi různá zařízení a také zvyšuje dostupnost databáze tím, že umožňuje přidávat nová disková zařízení bez nutnosti zastavit databázi. ASF rozděluje datové soubory a ostatní databázové struktury na rozsahy (extents), které se dále rozmísťují mezi disky, aby se zvýšil výkon i spolehlivost.

Architektura Optimal Flexible Architecture (OFA) poskytuje komplexního průvodce pro usnadnění správy softwaru a databázových souborů. Umožňuje zvýšit výkon databáze umístěním databázových souborů, při kterém se minimalizuje vznik úzkých míst při provádění V/V operací. Tento nástroj lze použít již při instalaci systému a umožní tak uživatelům lépe pochopit způsob uložení databáze na disku. Architektura je použitelná hlavně na klasických unixových systémech, v případě Windows jsou možnosti umístění programů a dat dané vlastnostmi OS, na Linuxu se dostává do kolize s FHS (Filesystem Hierarchy Standard).

4.2.3 Nástroje pro správu optimalizátoru

Optimalizátor Oracle vyžaduje pravidelné provádění analýz tabulek a indexů v databázi. Frekvence analýz by měla odrážet počet změn v objektech, u aplikací OLTP je vhodné ji vykonávat dle časového rozvrhu (např. jednou týdně). Statistické údaje se shromažďují prováděním procedur z balíčku DBMS_STATS, analýzu je možné provést pro celé schéma (procedurou GATHER_SCHEMA_STATS) nebo pro konkrétní tabulku, kdy je i automaticky provedena analýza příslušných indexů. Statistické údaje lze zjistit v pohledech DBA_TABLES, resp. DBA_INDEXES. Výstupem optimalizátoru je pak prováděcí plán, jenž je vhodným prvkem při ladění výkonnosti a plánovaní využití indexů. Vygeneruje se pomocí příkazu „explain plan“ nebo „set autotrace“. I Oracle samozřejmě poskytuje grafické zobrazení plánu, viz obr. 4.4.

Statistics | Activity | **Plan** | Tuning Information

Data Source **Cursor Cache** Capture Time **Oct 17, 2006 10:10:20 AM** Parsing Schema **SH** Optimizer Mode **ALL_ROWS**

Expand All | Collapse All

Operation	Object	Object Type	Order	Rows	Size (KB)	Cost	Time (sec)	CPU Cost	I/O Cost
SELECT STATEMENT			6			302280412			
SORT AGGREGATE			5	1	0.024				
NESTED LOOPS			4	5557	135.669	302280412	3627365	20942879354762	298681793
PARTITION RANGE ALL			2	918843	11,664.999	421	6	187177997	389
TABLE ACCESS FULL	SALES	TABLE	1	918843	11,664.999	421	6	187177997	389
TABLE ACCESS FULL	CUSTOMERS	TABLE	3	1	0.012	329	4	22792460	325

Show Explain Rewrite

Obr. 4.4 Zobrazení prováděcího plánu ([6])

Pro zobrazení využití paměti lze využít nástroj Statspack, který dokáže také vytvořit souhrn změn ve statistických údajích databáze za určité časové období. Pro sběr těchto údajů a generování přehledů a názorných statistik lze pak využít úložiště Automatic Workload Repository. AWR je

součástí každé databáze Oracle, obsahuje informace o všech klíčových statistikách a zatížení databáze, které jsou aktualizovány každých 30 minut. Místo ručního vytváření sestav z AWR je možné využít nástroj Automatic Database Diagnostic Monitor, který využívá data z AWR a je dostupný přes část Performance Analysis nástroje Oracle Enterprise Manager.

Dotazy provádějící logické i fyzické operace čtení lze získat pomocí pohledů V$SQL. Tento pohled obsahuje součet obou operací prováděných každým dotazem, uloženým v SGA spolu s informacemi o počtu provedení každého dotazu.

Využití indexů lze sledovat pomocí dynamického výkonového pohledu „V$OBJECT_USAGE. Nalezením a následným odstraněním nepoužívaného indexu se zkrátí doba potřebná pro zpracování operací vkládání, aktualizace a odstranění. Nejprve se musí nastavit sledování daného indexu a následně si vypíšeme informace z pohledu.

Vhodným nástrojem je také Tuning Pack. Jedná se o soubor aplikací, které usnadňují optimalizaci databáze, nalezení a vyladění problematických SQL dotazů i konfiguraci paměti. Balík zahrnuje aplikace SQL Analyze (nalezení a oprava problematických SQL dotazů, průvodce pro snadné vyladění dotazů všemi přístupnými metodami optimalizace), Oracle Expert (celkově automatizuje proces ladění databáze) a Index Tuning Wizard (hledá nesprávně zvolené indexy a lokalizuje místa, kterým by založení indexů prospělo).

4.3 Sybase 15.0

Třetí databázové řešení, které popíšu a uvedu jeho optimalizační nástroje, je produkt od firmy Sybase Adaptive Server Enterprise (ASE). Zde jsem čerpal především z pramenů [7], [8] a [10].

ASE představuje relační SŘBD, jehož aktuální verze je 15.0. Až do verze 4.9 (1993) byl vyvíjen společně s firmou Microsoft pod názvem SQL Server, v r. 1996 došlo k ukončení spolupráce a z tohoto důvodu se od verze 11.5 používá aktuální název ASE.

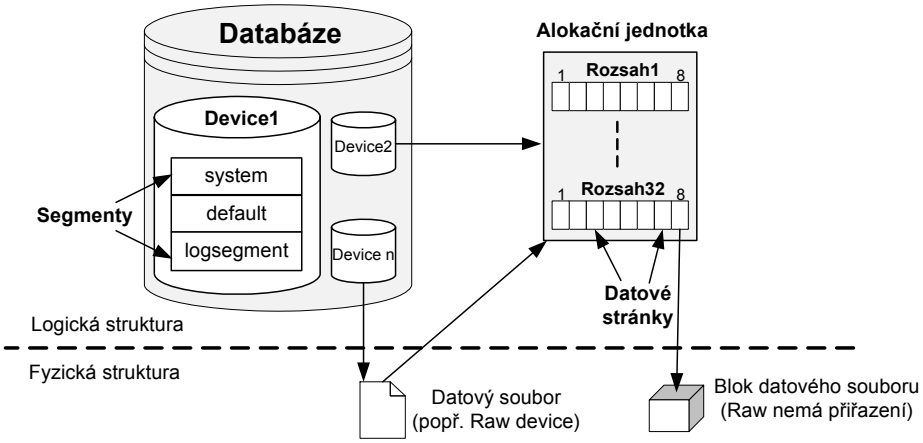
ASE s podporou otevřených standardů poskytuje platformu pro flexibilní a rychlou integraci datových zdrojů v prostředí heterogenních systémů. Vylepšený optimalizátor dotazů zajišťuje lepší úroveň výkonnosti a škálovatelnosti oproti předchozím verzím. ASE se nyní zaměřuje především na oblast e-Businessu, zahrnující podporu datových typů charakteristických pro OLAP prostředí, dynamické změny reflektující běh Internetových aplikací a zvýšenou bezpečnost obchodních dat v distribuovaném prostředí Internetu. Inovace ASE jsou tedy cíleny zejména na pokročilou správa dat pro e-Business, dynamickou optimalizaci výkonu a bezpečnost. ASE podporuje téměř všechny známé platformy (IBM AIX, Linux, Microsoft Windows, Sun Solaris a dal.).

Dle serveru gartner.com má Sybase pouze 3%ní poddíl na trhu mezi relačními databázovými systémy za vedoucím Oracle se 47%, následovaný IBM (DB2) s 21% a Microsoft se 17,5% (statistiky za rok 2006).

4.3.1 Základní struktury

Logické a zejména fyzické struktury jsou podobné strukturám v Microsoft SQL Serveru. Základní jednotkou uložení dat je datová stránka, jež představuje minimální množství dat přenositelných mezi diskem a vyrovnávací pamětí a její velikost může být 2, 4, 8 nebo 16 KB. Další jednotkou je rozsah, jenž se skládá z 8mi datových stránek, 32 rozsahů pak tvoří alokační jednotku. Zvětšení místa pro databázový objekt je realizováno alokováním rozsahu z alokační jednotky.

Pro namapovaní logické jednotky k souborům OS používá ASE tzv. zařízení (device), podobně jako tabulkové prostory v databázi Oracle. ASE device může být ovšem asociováno pouze s jedním datovým souborem (tabulkové prostory Oracle mohou mít více souborů OS), popř. RAW strukturou. Při vytvoření databáze na daný device se současně automaticky vytvoří 3 segmenty, které určují, kam bude objekt umístěn (lze tak vytvořit tabulku či index na určitý segment). Záznam v tabulce pak představuje jednu datovou stránku, oproti Oracle, kde jednotlivé řádky tabulky mohou být rozmístěny ve více blocích. Při instalaci ASE se vytvoří 5 systémových databází, které nelze smazat: master, model, sybsystemprocs, sybsystemdb a tempdb.



Obr. 4.5 Schéma databáze ASE

Paměťové struktury ASE jsou podobné jako v prostředí databáze Oracle, tzn., že v paměti jsou uloženy prováděcí plány, cache a stavové informace klientů a úloh daného serveru. Na disku jsou pak uloženy záznamy, metadata, informace o alokacích a přístupových metodách.

ASE nyní obsahuje podporu velkých disků - device velikosti až 4TB, počet deviců až 2,147,483,647 v rámci serveru.

4.3.2 Monitorování serveru

SQL Monitor je nástroj firmy Sybase, který spolupracuje s ASE a poskytuje grafický přehled o výkonu serveru. Zobrazená data jsou vhodná k nalezení výkonnostního problému. SQL Monitor obsahuje dvě části: serverovou, která běží na stejném stroji jako SQL server a má přístup ke sdílené

paměti; a klientská část, která může běžet kdekoliv a která poskytuje data ze serveru. Standardní nastavení podporuje pět připojení mezi serverem a klientem, jejich počet může být navýšen na maximálně 20.

SQL Monitor umožňuje zobrazit několik podoken:

- Cache – zobrazuje grafy výkonu datové a procedurální vyrovnávací paměti, z nichž jde vyčíst zejména, které požadavky jsou obslouženy pomocí datových stránek přímo z cache.
- V/V zařízení – poskytuje grafy a přehled o přístupech na disk, které mohou pomoci s rozdělením výkonu serverových zařízení
- Přehled výkonnosti – zahrnuje využití CPU, počet transakcí za sekundu, vytížení sítového provozu a dal.
- Výkonnostní trendy – odhaduje vytíženost prvků z předchozího bodu
- Aktivita procesů – zobrazuje statistiky zvoleného procesu, zejména jeho využití CPU a přístupů na disk
- Seznam procesů – zobrazuje procesy a status aktuálního serveru, který proces přistupuje k jakému objektu apod.
- Transakční aktivita – poskytuje sloupcový diagram zobrazující přehled všech transakcí seřazený dle typu transakce.

Další nástroje pro monitorování výkonu jsou přímo zabudovány v SQL serveru. Prvním z nich je uložená procedura sp_monitor, která zobrazuje mnoho výkonnostních parametrů daného serveru, např. vytíženost CPU, V/V zařízení, počet připojení. Každý parametr obsahuje dvě čísla: první zobrazuje interval od posledního spuštění SQL serveru a druhé číslo udává interval od posledního spuštění sp_monitoru. Lze tak naplánovat pravidelné spouštění této procedury, výstup zapisovat do souboru a mít tak k dispozici přehledné statistiky v daných intervalech.

ASE zavedlo od verze 12.5 tzv. MDA tabulky – Monitoring and Diagnostic Access. Nejedná se přímo o databázové tabulky, ale o tzv. proxy tabulky, jež jsou umístěny na daném serveru (konkrétně jsou uloženy v databázi master) a využívají vzdálené volání procedur (RPC). MDA tabulky jsou dostupné přes T-SQL dotaz a jsou vytvořeny pomocí RPC, jenž může přímo přistupovat k paměťovým strukturám. MDA tabulky umožňují získat informace o aktuální činnosti na úrovni dotazů, tabulek, procedur a běžících procesů. V aktuální verzi Sybase serveru se vyskytuje 36 MDA tabulek, např. monState (výpis obecných informací a stavu serveru), monDeadLock (informace o uvíznutí), monCachedObject (statistiky o databázových objektech, které mají své stránky umístěny v datové cache) a dal. MDA tabulky tak poskytují vhodný nástroj pro získání informací o serveru, jež obvykle nejsou přes standardní SQL dotazy dostupné.

4.3.3 Správa a optimalizace fyzických zdrojů

Kvalitní systém musí být schopen se přizpůsobit měnícímu se charakteru zátěže. Adaptive Server Enterprise představuje databázový systém pro hybridní transakčně/analytické prostředí díky možnosti dynamického ladění výkonnostních charakteristik. ASE umožňuje dynamicky nastavovat cache pro datové a indexové stránky, redukovat I/O operace, měnit uživatelské priority, řídit zdroje na úrovni CPU, limitovat zdroje pro konkrétní dotazy nebo dávky. Administrátor může snadno rekonfigurovat systém za běhu bez restartování a hladce přejít z režimu transakčního zpracování dat na analytický režim typický pro systémy pro podporu rozhodování. Dynamická rekonfigurace systému může být navíc plně automatická s využitím skriptů, které mohou upravovat parametry podle měnící se zátěže.

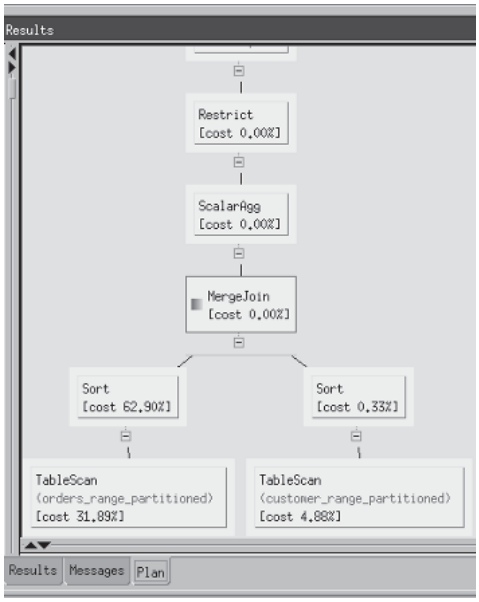
Na rozdíl od předchozích verzí je nyní možnost nastavení automatického růstu databáze, resp. jejích datových souborů. Povolení a nastavení limitu lze provést pomocí systémové procedury sp_dbextend. ASE také podporuje partitioning pomocí vestavěných funkcí na mnoha úrovních s mnoha možnostmi nastavení, díky němuž lze jednodušeji spravovat velké tabulky a indexy.

4.3.4 Nástroje pro optimalizaci

ASE na rozdíl od předchozích verzí podporuje automatické aktualizace statistik. Tato nová vlastnost zahrnuje funkci datachange, která zjišťuje, na jaké úrovni (tabulka, index, popř. sloupec) je třeba provést aktualizaci. Jejím výsledkem je procentuální ohodnocení změny daného objektu (0% znamená žádnou změnu od poslední aktualizace statistik). Statistiky je možné také měnit manuálně příkazem update statistics <název tabulky>. Nástrojem optdiag lze pak zobrazit a monitorovat statistiky použité optimalizátorem a zjistit tak tabulky, které vyžadují aktualizaci, popř. reorganizaci.

ASE dále vylepšilo optimalizátor dotazů - zlepšení výkonnosti při zpracování dotazů a nové Ineractive ISQL prostředí včetně grafického query plan vieweru, jenž umožňuje grafické zobrazení prováděcího plánu. Je začleněn do ISQL a lze jej tak spustit z nástroje Sybase Central (grafický nástroj pro správu databáze) nebo z příkazové řádky. Po napsání SQL dotazu se následně ukáže prováděcí plán, viz obr. 4.6.

Pro zobrazení prováděcího plánu lze také použít příkaz „set showplan“, který se provede před samotným spuštěním SQL skriptu nebo dotazu Pokud chceme, aby daný skript či dotaz nebyl vykonán, a tedy jen zobrazit příslušný prováděcí plán, pak před spuštěním příkazu set showplan zapíšeme „set noexec“. Jakmile se ovšem dotaz spustí, plán již nejde měnit a po ukončení dotazu je nenávratně pryč. Proto nově přibyl parametr „show_abstract_plan“, který vygeneruje tzv. abstraktní plán pro daný dotaz. Tento plán umožňuje administrátorovi zobrazit a změnit přístupovou metodu, typ a pořadí join spojení, úroveň paralelismu, strategii vyrovnávací paměti a využití indexu a to vše přímo na úrovni daného dotazu. Abstraktní plán tak může být uložen, změněn a poté znovu použit na originální dotaz. ASE poskytuje 15 uložených procedur pro práci s abstrakntím plánem.



Obr. 4.6 Graficky znázorněný plán dotazu (převzato z [10])

Vylepšení optimalizátoru bylo realizováno mimo jiné přidáním konfiguračního parametru „optimization goal“, díky němuž může uživatel zvolit optimalizační strategii, jež nejvíc vyhovuje danému prostředí a dotazům. Tento parametr zahrnuje skupinu přednastavených kritérií, která vylepšují chování optimalizátoru a která mohou být nastavena na úrovni serveru, aktuálního spojení nebo dotazu. Tato kritéria tedy představují specifické relační algoritmy pro prováděcí plán, např. merge_join, parrallel_query, hash_join. Výpis aktuálních parametrů lze provést příkazem: `select @@optgoal.`

5 Analýza enterprise prostředí

Prvním krokem analýzy enterprise prostředí bylo zjištění hardwarové a softwarové konfigurace serverů. Zaměřil jsem se na čtyři nejvytíženější Microsoft SQL Servery, u kterých je požadováno nejpřísnější SLA (Service Level Agreement, tzv. smlouva se zákazníkem), pojmenované jako SQL1 – 4 (číslování seřazené dle důležitosti). Analýza všech serverů probíhala na stejné bázi, v případě odlišností jsem uvedl rozdíly.

5.1 Konfigurace serveru

Všechny servery obsahují téměř shodnou konfiguraci, viz následující tabulka serveru SQL1:

Operační systém	Server 2003 Enterprise Edition SP2 32bit
Model	IBM eServer BladeCenter LS20
CPU	4x Dual Core AMD Opteron 280 (2400 MHz, L2 cache 2 × 1024 KB, HyperTransport 1000 MHz, Socket 940)
Paměť	4GB RAM se zapnutou podporou PAE
Disk	3x SCSI IBM 2145 SDD (20 GB pro systém, 140 GB pro aplikace a 4GB pro cluster quorum)

Tab. 5.1 Základní konfigurace serveru

Dle rychlostí, jakým disponují jednotlivé systémové prostředky daného serveru, jsem určil, jak rychlé jsou přístupy k tabulce o velikosti např. 10GB.

Systémový prostředek	Rychlost	Propustnost	Přístup k tabulce 10GB
CPU	2,4 GHz	8 GB/sec	1,2 sec
Paměť	500 MHz	2 GB/sec	5 sec
Disk	5 – 200 MB/sec	5 – 200 MB/sec	33 minut

Tab. 5.2 Rychlosti systémových prostředků

Z těchto hodnot vyplývá, že je vhodné se zaměřit na co nejrychlejší přesun dat z disku do pamětí RAM a následně do vyrovnávacích pamětí (cache) CPU. V SQL serveru není moc možností pro nastavení CPU, to je kontrolováno vývojáři dané aplikace (paralelní zpracování aj.). SQL server

je ovšem koncipován tak, aby se snažil načíst co nejvíce dat do paměti a eliminovat tak přístupy na disk.

Podnikové SQL servery jsou osazeny 32bitovým operačním systémem a z toho vyplývá i omezení paměťového prostoru na 4GB (2^{32} = 4294967296B = 4GB), přičemž 2GB jsou určené pro uživatelský adresový prostor a 2GB pro jádro OS. Intel ovšem vyvinul způsob, jak toto omezení obejít rozšířením adresové sběrnice na 36 bitů. Toto řešení se nazývá PAE (Physical Adress Extension) a umožňuje na 32bitových systémech použít až 64GB RAM. PAE se nastaví příznakem v souboru boot.ini a povolí tak systému použít více než 4GB RAM.

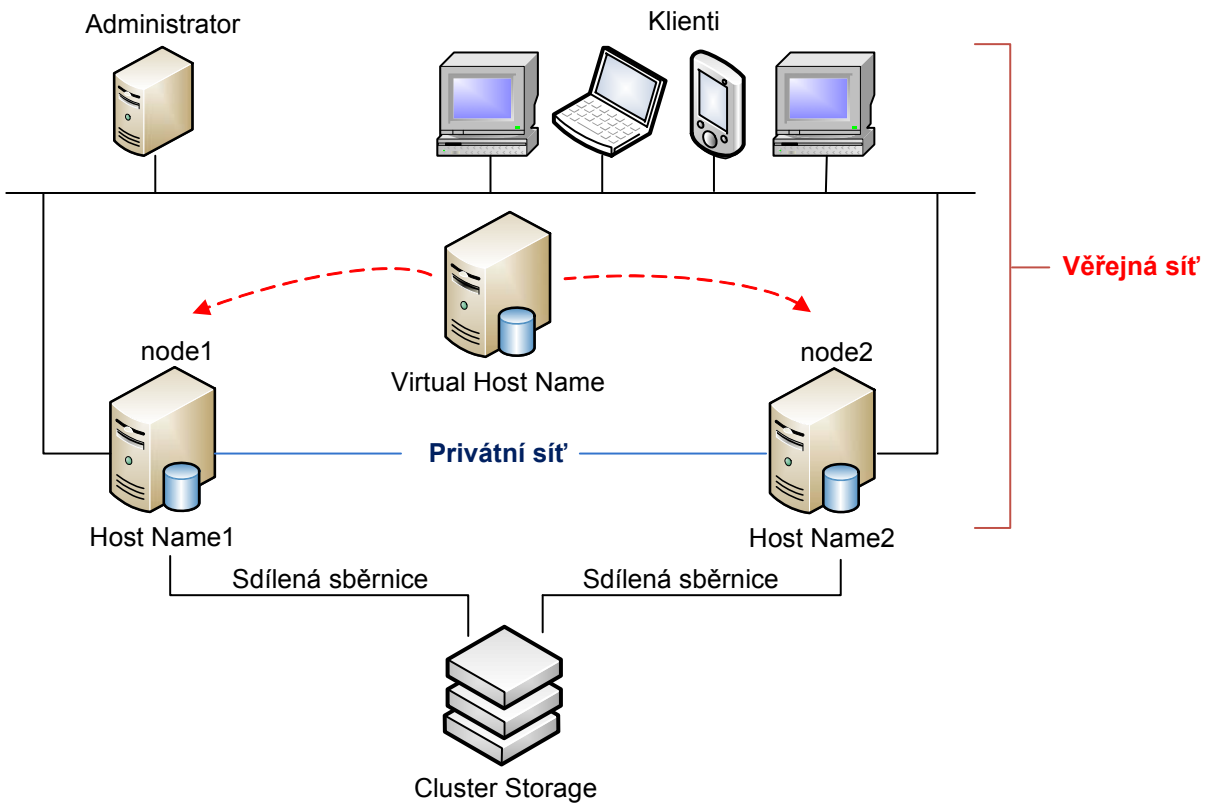
V našem případě (32bitový OS se 4GB RAM) je PAE povoleno a to je tedy ideální nastavení. Doporučené a nejvhodnější je ovšem použití 64bitové architektury, tzn. 64bitový OS a 64bitový SQL Server, daný server pak není omezen rozsahem paměti a umožňuje rychlejší přístup k datům a jejich rychlejší zpracování.

Vzhledem k tomu, že jsem se zaměřil na nejvíce vytížené systémy, které požadují nejpřísnější SLA, tyto systémy využívají cluster řešení a tím i odpovídající diskový subsystém. Ten je realizován připojením přes optiku do tzv. SAN (Storage Area Network, datová síť sloužící pro připojení diskových polí, popř. páskových knihoven k serverům), kde jsou SAS disky nebo v levnějších midrange polích eSATA disky. Seriál Attached SCSI (SAS) představuje sběrnici pro transfer dat do koncových zařízení s větší rychlostí než klasické SCSI (Fibre Channel) či SATA a podporou většího počtu zařízení. Konkrétně jsou servery osazeny třemi SCSI disky pomocí hardwarového RAID 1 (20 GB pro systém, 140 GB pro aplikace a 4GB pro quorum – cluster). Jedná se o řadu IBM 2145 SDD – Subsystem Device Driver, který poskytuje multipath v IBM diskovém prostředí a optimalizuje přístupy na disk. Řada 2145 představuje SAN Volume Controller (SVC), který umožňuje virtualizaci diskového úložiště (abstrakce logického datového prostoru od fyzického) a tím jednodušší správu clusteru. Vzhledem k tomu, že SQL server je daleko více závislý na výkonu diskového prostředí než ostatní aplikace, neboť spravuje velké množství dat v uživatelských databázích, je důležité mít správně nastaven diskový subsystém.

Co se týče síťové komunikace, v dnešní době poskytuje síťová infrastruktura dostatečnou kapacitu umožňující SQL serveru maximálně využít všechny systémové prostředky, aniž by došlo k přetížení síťové komunikace.

5.1.1 Cluster řešení

Pro zajištění vysoké dostupnosti SQL server využívá cluster řešení typu failover, tzn., že v jednu dobu je aktivní 1 uzel (node) a v případě jeho nedostupnosti je automaticky proveden přesun všech zdrojů a prostředků na druhý uzel, viz schéma:



Obr. 5.1 Schéma cluster řešení

Jedná se o tzv. single cluster topologii, kdy je nainstalován pouze jeden virtuální server a dva uzly (aktivní/pasivní). Datové a log soubory databáze jsou umístěny ve sdíleném úložišti (cluster storage), spustitelné a binární soubory pro daný virtuální server jsou uloženy v privátních úložištích každého uzlu. Virtuální server je řízen vždy pouze jedním primárním uzlem, sekundární uzel je ve stavu čekání. V případě selhání primárního uzlu je SQL server aktivován na sekundárním a převezme kontrolu nad sdílenými daty v cluster úložišti, dochází k failoveru. Doporučuje se, aby oba uzly měly totožnou hardwarovou i softwarovou konfiguraci z důvodu identického běhu SQL serveru v případě failoveru. Cluster technologie vyžaduje tzv. quorum drive – logická jednotka ve sdíleném cluster úložišti, jež obsahuje informace o stavu daného clusteru a logovací soubor informující o službě MS Distributed Transaction Coordinator (správce transakcí umožňující klientským aplikacím pracovat v jedné transakci s více datovými zdroji).

Během analýzy jsem ověřil správné nastavení závislosti jednotlivých zdrojů a prostředků v cluster manageru, např. SQL server je závislý na síťovém názvu a disku (tyto části musí být uvedeny do stavu ONLINE dříve než samotný server):

SQL Agent → SQL Server → SQL Network Name → SQL IP Address
SQL Server → SQL Fulltext → Disk E:

V neprodukční dohodnuté době jsem také otestoval failover, a tedy vynucený přesun zdrojů z jednoho uzlu na druhý. Test proběhl úspěšně, server a databáze byly dostupné, vyskytl se ovšem problém s TSM agentem TDP_SQL_Scheduler, který má na starost plánované zálohy, tato služba na druhém uzlu nenastartovala. Problém se vyřešil správným nastavením virtuální cesty v registrech.

5.2 Analýza databázového prostředí

Po analýze hardwarové a softwarové konfigurace serveru a analýze clusteru jsem se zaměřil na základní nastavení databázového serveru, viz následující tabulka serveru SQL1:

Verze	9.2.3042.00 (Service Pack 2), Enterprise Edice
Instance	Instance1
Instalační cesta	C:\Program Files\Microsoft SQL Server\MSSQL.2\MSSQL
Umístění datových souborů	E:\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\master.mdf E:*.mdf
Autentifikace	Mixed mode (SQL + WIN)
Zálohovací strategie	Full recovery model, záloha pomocí TSM: 1x denně Full backup, co hodinu backup logu
Alokace paměti	Dynamické řízení, AWE zakázáno Min: 0 MB, Max: virtuální + fyzická, pro dotaz: 1024 kB

Tab. 5.3 Konfigurace databázového serveru

Spustitelné a datové soubory jsou správně dle cluster topologie umístěny na různých discích, což umožňuje i lepší propustnost a zvýšení výkonu.

Pro zajištění vysoké dostupnosti v případě havárie (disaster recovery) jsou na všech systémech implementovány zálohy pomocí IBM technologie Tivoli Storage Manager (TSM). Toto řešení umožňuje centralizovanou a automatizovanou správu dat a záloh pomocí agentů zvaných Tivoli Data Protection (TDP), jež jsou připojeny na TSM server. Všechny databázové systémy mají nastaven úplný model obnovy dat (full recovery model), kdy jsou všechny operace protokolovány, včetně hromadných operací a načítání dat. Zálohovací strategie pak tvoří každodenní úplnou zálohu a hodinové zálohy transakčního logu.

U všech analyzovaných serverů byla nastavena paměťová konfigurace na automatické řízení SQL Serverem, dochází tak k automatické a dynamické alokaci paměti dle zatížení a dostupnosti

zdrojů. Celkové využití pamětí kolísá mezi nastavenou minimální a maximální hodnotou. Podpora paměti AWE (Address Windowing Extensions) je zakázána, více v kapitole 7.1.2 Optimalizace využití paměti.

5.2.1 Analýza zabezpečení

Zabezpečení SQL Serveru je úzce provázáno se zabezpečením domén Windows. Umožňuje ověřování založené na uživatelských účtech, členství ve skupině i pomocí standardních uživatelských účtů SQL Serveru.

Všechny analyzované servery jsou nastaveny na ověřování přístupu pomocí Windows a SQL Serveru, jedná se o tzv. mixed (smíšený) mód autentifikace, kdy uživatelé domény Windows používají k přístupu na SQL Server jediný účet, ostatní uživatelé, jež nemají doménový účet (např. externisti), se přihlašují pomocí účtu vytvořeného přímo na SQL Serveru. Tyto zabezpečovací režimy jsou nastaveny na úrovni serveru, tzn., že platí pro všechny databáze.

Ověřil jsem i nastavení a pořadí protokolů pro připojení k daným SQL serverům:

1. Shared Memory: protokol sdílené paměti pouze pro lokální spojení.
2. TCP/IP: upřednostňovaný protokol pro lokální i vzdálená spojení, SQL server naslouchá na portu TCP 1433.
3. Named Pipes: pro připojení aplikací napsaných pro Windows NT a 98, lze jej zakázat.

Poté jsem zkontroloval zabezpečení na základních úrovních SQL Serveru z pohledu kontextů:

- Úroveň Windows: přihlašovací jméno domény Windows, skupina Windows
- Úroveň SQL Server: role serveru, přihlašovací jméno SQL Serveru
- Úroveň databáze: uživatel databáze, role databáze

5.2.1.1 Ověření auditu

Audit slouží ke sledování přístupů uživatelů k SQL Serveru. Lze jej použít v obou ověřovacích režimech u důvěryhodných i nedůvěryhodných spojení.

Na všech analyzovaných serverech byl audit povolen, tzn, že přihlášení uživatelů byla zaznamenávána. Nastavena byla možnost *Failed Logins Only* a tedy ukládání pouze chybných pokusů o přihlášení (implicitní nastavení).

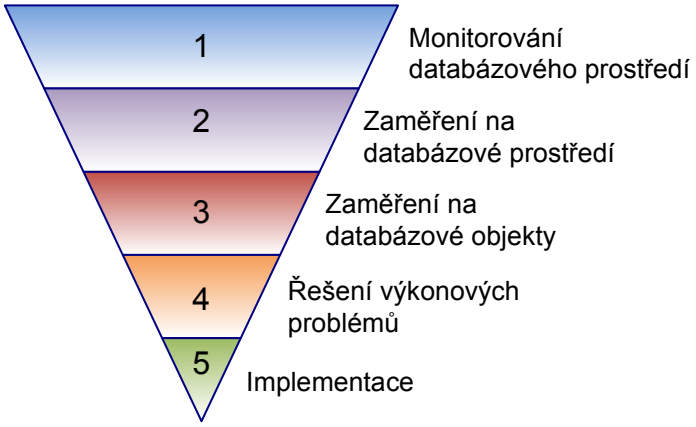
6 Monitorování SQL Serverů

Databázové enterprise systémy ve firmě se primárně využívají pro OLTP prostředí a od toho se také odvíjela analýza a následná optimalizace. Na rozdíl od OLAP systémů, kde je požadován optimalizovaný V/V subsystém pro analýzu velkých objemů dat, OLTP systémy vyžadují V/V subsystém vyladěný pro časté čtení a zápis menších datových jednotek.

Každý server je nejvíce využíván v různou dobu, konkrétně server SQL1 má největší vytížení na přelomu měsíce, kdy se provádí sběr dat pomocí aplikace App1. Vzhledem k těmto skutečnostem jsem přizpůsobil monitoring a sledování systémů.

Prvním krokem monitoringu enterprise prostředí bylo zjištění slabých míst v systému. Dále jsem jej využil k auditu uživatelů (zjištění počtu připojení a aktivit), ustanovení prahu výkonu (baseline) a především k diagnostice eventuálních výkonnostních problému, které lze pak použít jako základ pro optimalizaci.

Nejprve jsem se tedy zaměřil na databázové prostředí a postupně jsem dle výsledných analýz zaostřoval pozornost na konkrétní databázové objekty a s nimi související výkonnostní problémy, viz následující schéma:



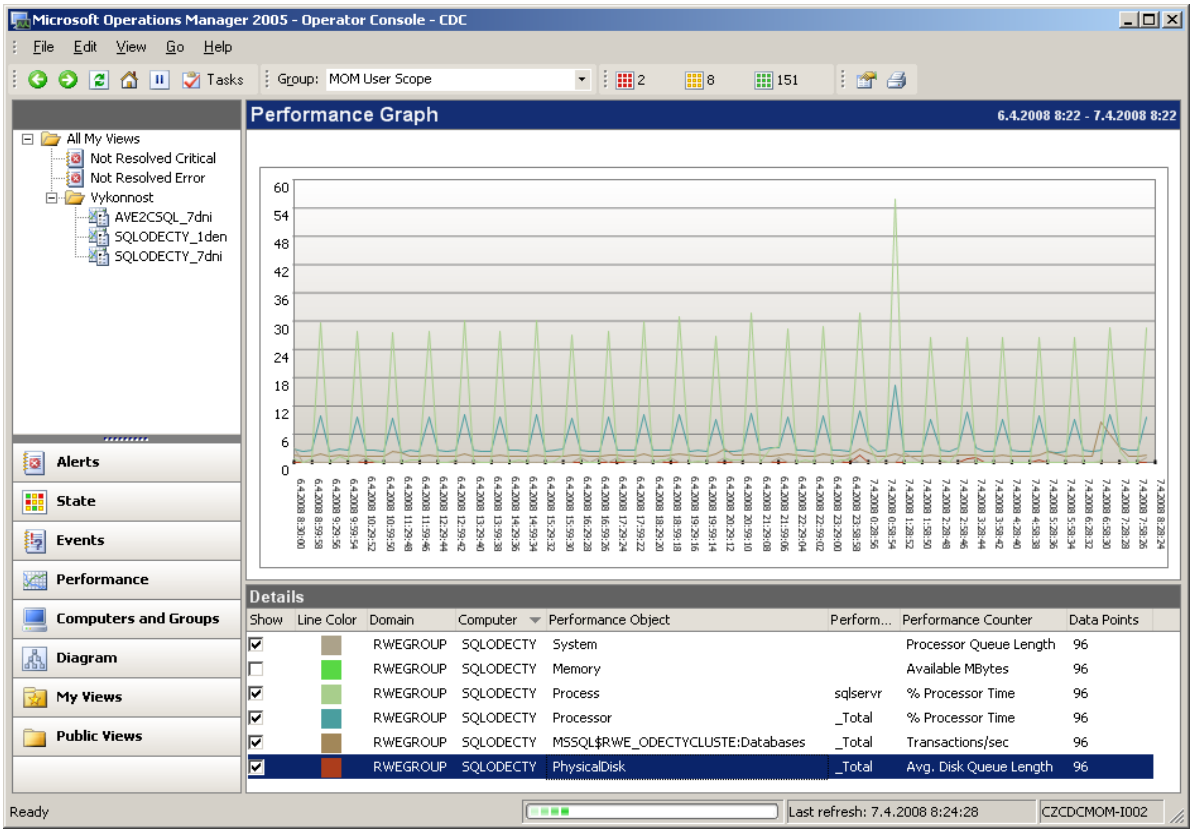
Obr. 6.1 Princip monitoringu

1. Vytvoření sledovací strategie umožňující zaznamenat dostatek informací pro identifikaci výkonnostních problémů v databázovém enterprise prostředí. Využil jsem nástroj OS System (Performance) Monitor, jenž obsahuje čítače zobrazující podrobnější přehled o výkonnosti serveru
2. Analýza zaznamenaných dat k zaostření se na hlavní důvod problému (CPU, paměť, V/V). Zde jsem využil především tyto nástroje:
    - SQL Profiler: umožňuje zaznamenat zátěž serveru (včetně dotazů) a analyzovat jej.

- Activity Monitor: součást SQL serveru, která zobrazuje informace o běžících, popř. zablokovaných procesech, zámcích a aktivitách uživatelů.

3. Identifikace analyzovaných dat pro zjištění specifických problémů, jako např. pomalý diskový subsystém, nadměrný počet databázových uživatelů nebo problém se specifickou uloženou procedurou. Toto zaměření na konkrétní databázové objekty vyvolalo změnu v monitorovací strategii (přidání specifických čítačů a událostí SQL Profileru). Využil jsem také dynamické pohledy (DMV).
4. Řešení výkonových problémů analyzovaných v předchozích třech krocích. Zahrnuje ladění dotazů, správu indexů, ověření síťové komunikace, identifikace objektů způsobující případné uváznutí (deadlock).
5. Implementace změn na serveru, popř. klientech. V posledních dvou krocích jsem využil nástroj Database Tuning Advisor (DTA).

Během sledovaného období byly servery monitorovány také nástrojem Microsoft Operations Manager (MOM), jenž pomocí agentů, umístěných na serverech, sbírá kromě výkonnostních údajů také důležité údaje o dostupnosti služeb, místu na disku a obecně stavu celého serveru. Díky MOMu jsem tudíž mohl případně zjistit, zda mé monitorovací nástroje nevytěžují nadměrně server a podniknout pak případné kroky ke změně četnosti sbírání dat apod.



Obr. 6.2 Ukázka výkonnostního přehledu nástroje MOM

K diagnostice problémů jsem vzal v potaz následující aspekty:

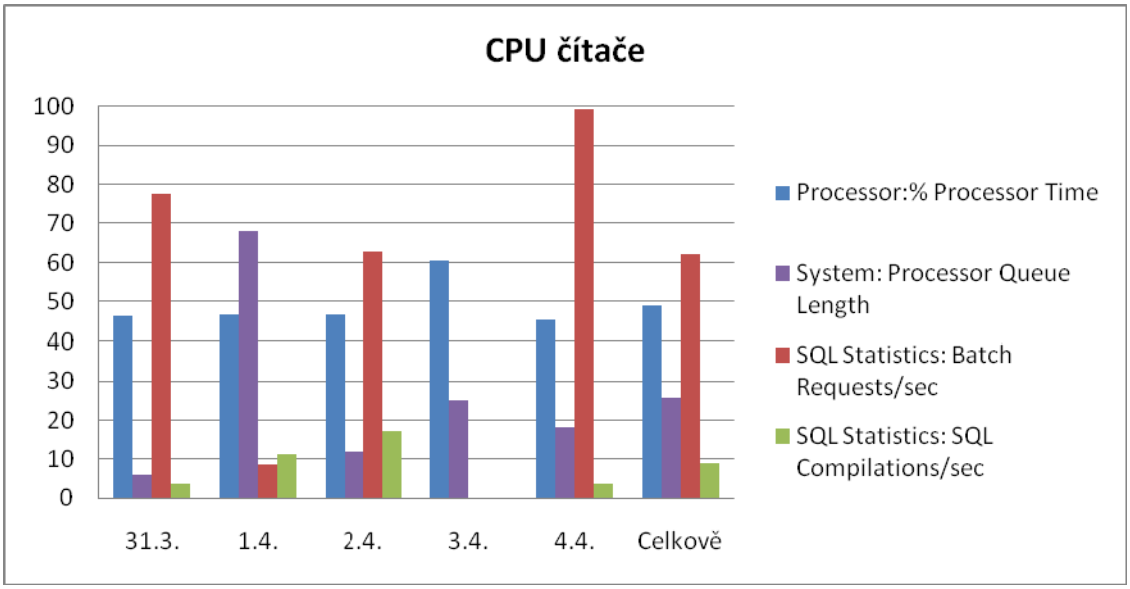
- Fyzické zdroje: CPU, paměť a V/V prostředky
- Tempdb: SQL server využívá pro všechny instance pouze jedinou dočasnou (pomocnou) databázi, a proto se zde můžou objevit výkonnostní problémy.
- Pomalé dotazy nad databází: změny ve statistikách využívaných optimalizátorem mohou vést ke špatným prováděcím plánům, chybějící indexy mohou vést ke zbytečným procházením celých tabulek, blokování přístupu zase může vést ke zpomalení aplikace.

Nejprve jsem se tedy zaměřil na diagnostiku a monitorování SQL serverů z hlediska optimálního využití fyzických zdrojů. Před zjištěním, zda zde existuje úzké místo, jsem musel zjistit, jak jsou využívány fyzické zdroje za normálních podmínek, tzn. určit práh.

6.1 Sledování systémových prostředků

Monitoring systémových prostředků zahrnoval čítače dvou typů: základní systémové parametry a parametry týkající se přímo SQL serveru. Četnost sbírání dat byla nastavena s ohledem na vytížení serveru na jednu minutu. Na konci každého dne pak byly do tabulky zaznamenány tři hodnoty za dané sledované období: průměrná, minimální a maximální. Dle specifik různých dokumentací a dle předběžného monitorování během dřívější doby (pomocí MOM) jsem označil ty, které mohly být příznakem slabého místa systému a které mi pak posloužily v dalších krocích analýzy.

Uvedu zde grafy vybraných čítačů nejvytíženějšího serveru SQL1 a popíšu jejich význam.

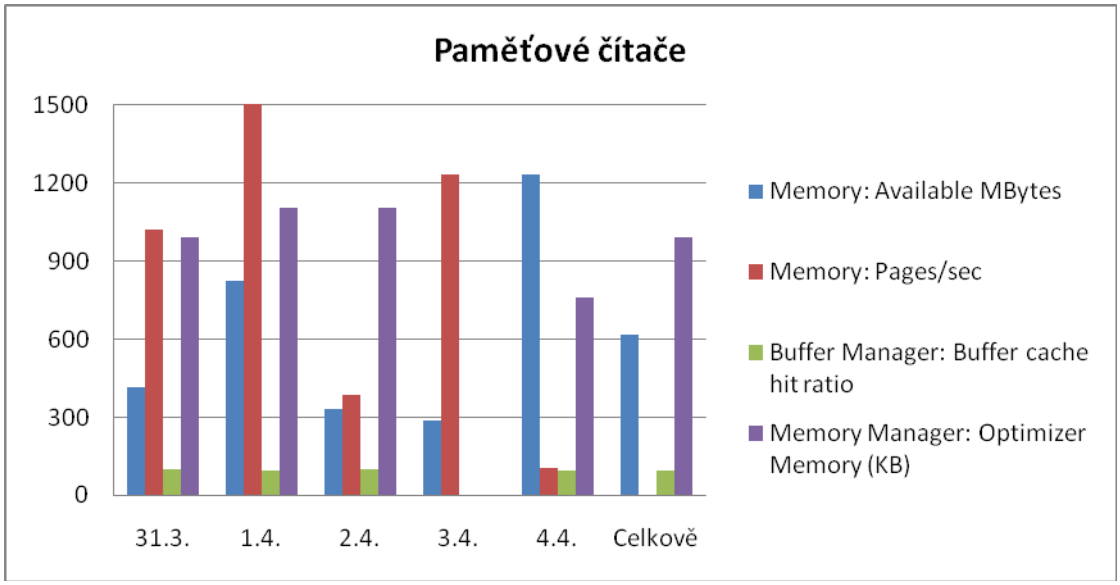


Obr. 6.3 Graf čítačů CPU

Úzká místa způsobená CPU se většinou objeví náhle a nečekaně, bez předchozí větší zátěže a jsou zpravidla zapříčiněna neoptimálním prováděcím plánem, špatnou konfigurací popř. nesprávným návrhem databáze. K jejich nalezení jsem sledoval tyto maximální hodnoty (celková znamená průměrnou hodnotu všech maximálních za dané období):

- **Čas procesoru**: procento doby, během které procesor zpracovává podproces, který není nečinný. Tento ukazatel je primárním indikátorem aktivity procesoru a zobrazuje průměrný čas zaneprázdnění procesoru (v procentech) ve vzorkovací periodě. Práh: max. 80%
- **Délka fronty procesoru**: obsahuje počet podprocesů umístěných ve frontě procesoru, v počítačích s více procesory je jediná fronta pro využití času procesoru, je tedy nutné podělit tuto hodnotu počtem procesorů obsluhujícím danou pracovní zátěž. Práh: max. 10-12
- **Počet SQL požadavků / s**: zobrazuje počet SQL dávek či příkazů, jež SQL server přijímá za sekundu. Obecně více než 1000 příkazů za sekundu značí velmi vytížený SQL server.
- **Počet kompilací / s:** indikuje počet kompilací vykonaných SQL serverem za sekundu. Práh: více než 100.

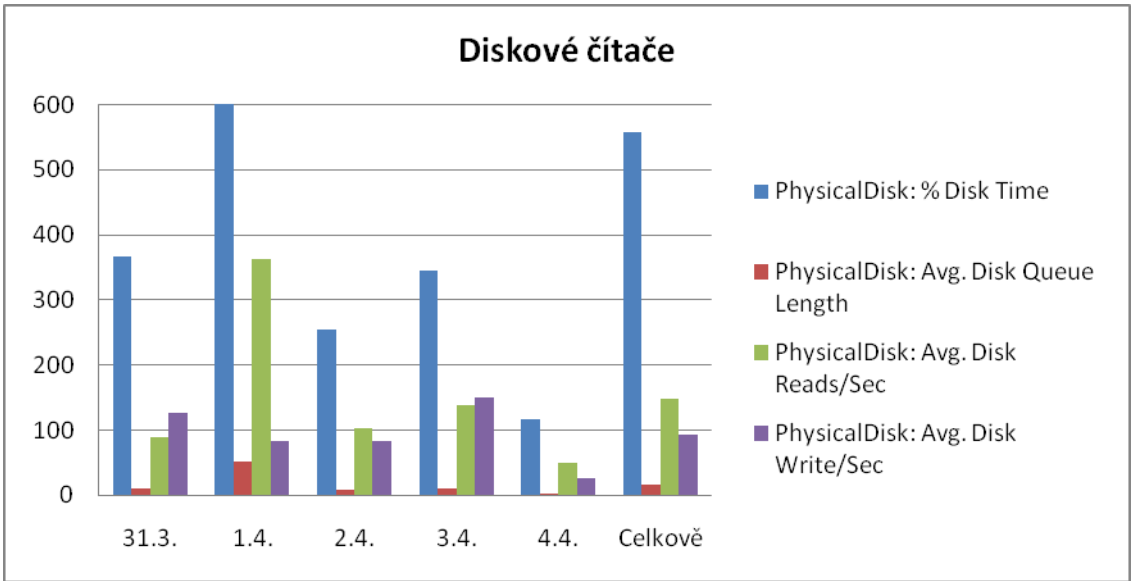
V grafu pamětových čítačů jsem uvedl ty, které za dané sledovací období měnily svou hodnotu, popř. se projevila hodnota kritická:



Obr. 6.4 Graf pamětových čítačů

- **MB k dispozici:** zobrazuje velikost fyzické paměti (v MB), která je k dispozici procesům spuštěným v počítači. V grafu je uvedena minimální hodnota, neboť tato může být pro nás kritická.
- **Stránky / s:** obsahuje počet stránek čtených z disku nebo zapisovaných na disk, které mají vyřešit hardwarové chyby stránek. Tento čítač je navržen jako hlavní ukazatel chyb způsobujících opožďování celého systému.
- **Buffer cache hit:** procento stránek nalezených ve vyrovnávací paměti SQL serveru bez nutnosti jejich čtení z disku. Práh: min. hodnota větší než 98%.
- **Pamět optimalizátoru:** množství paměti dostupné pro optimalizaci dotazů.

Dále jsem ovšem sledoval např. Memory Manager: Memory Grants Pending (počet procesů čekající na přidelění paměťového prostoru), která byla po celou dobu nulová, Memory: Free System Page Table Entries (počet položek stránkovací tabulky, které momentálně nejsou používány systémem), jejíž hodnota byla téměř neměnná a dosahovala hodnot daleko od kritických, Buffer Manager: Page life expectancy (jak dlouho jsou datové stránky uloženy v cache), hodnotota tohoto čítače by neměla klesnout pod 300 sekund, naše průměrná hodnota se pohybovala nad 10 000 a Buffer Manager: Lazy writes/sec (přesun tzv. špinavých stránek z bufferu na disk za účelem uvolnit místo v paměti), jehož hodnota nepřesáhla 10 a tedy v normě.



Obr. 6.5 Graf čítačů diskového subystému

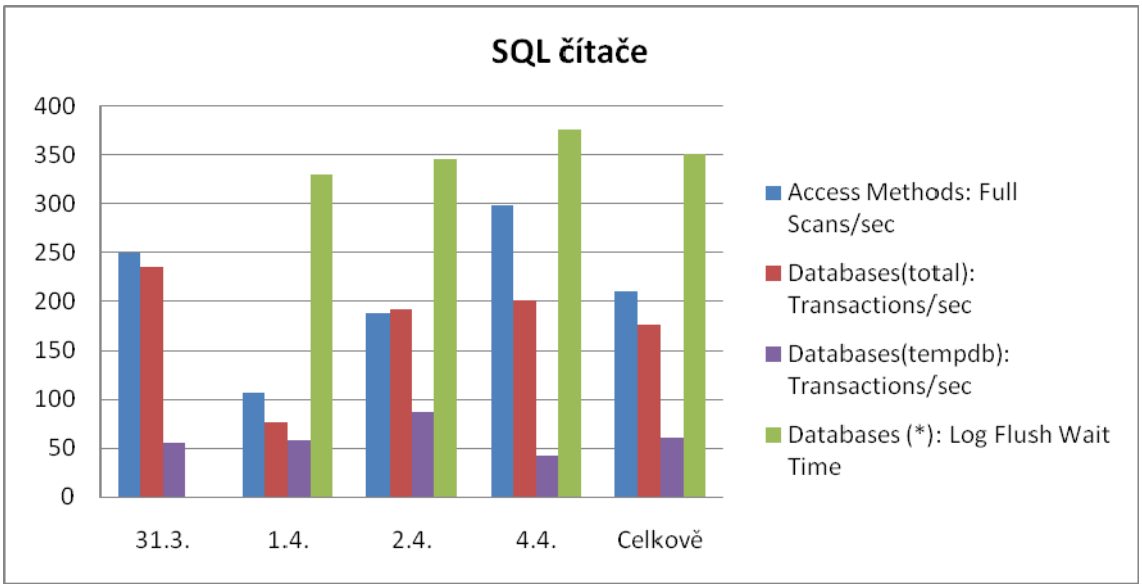
Pro nalezení slabého místa diskového subsystému jsem zvolil mimo jiné tyto čítače:

- **% času disku:** procento uběhlé doby, po kterou vybraná disková jednotka vyřizovala žádosti o čtení a zápisy. Treshold je méně než 50% na daný disk, v grafu jsou uvedeny maximální hodnoty, průměrná hodnota za dané období činí 3,44.

- **Střední délka fronty disku:** průměrný počet požadavků na čtení nebo zápis, které byly zařazeny do fronty vybraného disku.Treshold činí méně než 2 na disk.
- **Čtení z disku/s:** míra čtecích operací na disku.
- **Zápisy na disk/s:** míra zapisovacích operací na disku. Průměrná hodnota posledních dvou čítačů by měla činit 20ms, maximálni hodnoty, jež jsou zaznemány v grafu, činí daleko více a mohou znamenat V/V problém. Průměrné hodnoty jsou ovšem v normálu – 2,1 a 9,9 (což značí velice rychlý přístup na disk).

Dále jsem se zaměřil na Paging File: % Usage, jehož hodnota nepřesáhla 10%.

Poslední graf zobrazuje několik čítačů souvisejících se statistikami SQL serveru:



Obr. 6.6 Graf SQL čítačů

- **Full scans / s:** počet přístupů na celou tabulku za sekundu. Požadavkem je co nejnižší hodnota, resp. zajistit co nejvíce přístupu pomocí indexů a vyvarovat se tak dotazů bez podmínek.
- **Transakcí / s:** počet provedených transakcí za sekundu. Jsou zaznamenány dvě hodnoty: transakce provedené nad všemi databázemi a transakce pouze v rámci dočasné databáze – tempdb.
- **Čekací doba na zápis do logu:** celková doba v ms pro zápis do logu, tzv. log flush. Při spuštění transakce jsou data nejprve zapsána do tzv. log cache, což je místo v paměti použité pro záznam dat, jež budou zapsána do log souboru. Pokud je transakce potvrzena, pak dojde právě k log flush – data z cache jsou zapsána do fyzického log souboru. Tento čítač tedy úzce souvisí s počtem transakcí, neboť po ukončení každé transakce dojde k zápisu do logu a tato čekací doba by měla být co nejmenší.

Mezi další SQL čítače, které jsem použil pro monitoring, jsou General Statistisc: User Connections (počet připojení, musí se brát v potaz, že několik uživatelů může sdílet jedno připojení), které v průměru dosahovalo hodnoty 20, Locks: Number of Deadlocks/sec (počet uváznutí) bylo po celé sledované období nulové, což značí korektně navrhnutou aplikaci využívající danou databázi.

6.2 Analýza pomocí SQL Profileru

Během sledovaného období jsem zaznamenával zátěž pomocí nástroje SQL Profiler, jenž umožňuje zachytit námi zvolené události vyvolané aktivitou SQL serveru a vytvořit tak tzv. trace – stopu. Pro nastavení a spuštění stopy lze použít grafickou nadstavbu SQL Profileru, já jsem ovšem použil systémové uložené procedury, které mají menší zátěž na systémové prostředky a které lze pak použít jako zdroj naplánované úlohy a mít tak kontrolu nad spouštěním jednotlivých stop.

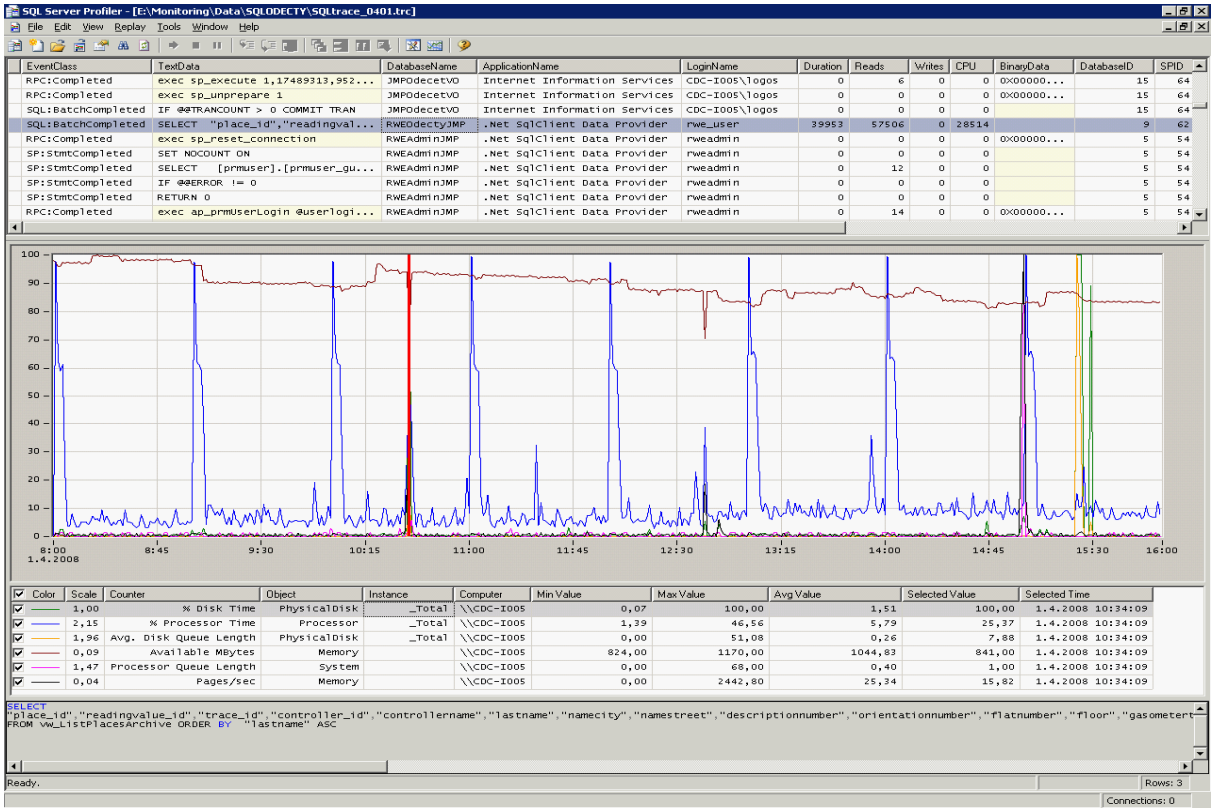
Zaznamenal jsem následující události:

- SQL: BatchCompleted – objeví se v případě vykonání TSQL příkazu.
- RPC:Completed – zaznamenán v případě dokončení vzdáleného volání procedury (Remote procedure call).
- SP:StmtCompleted – představuje vykonání TSQL příkazu vně procedury.

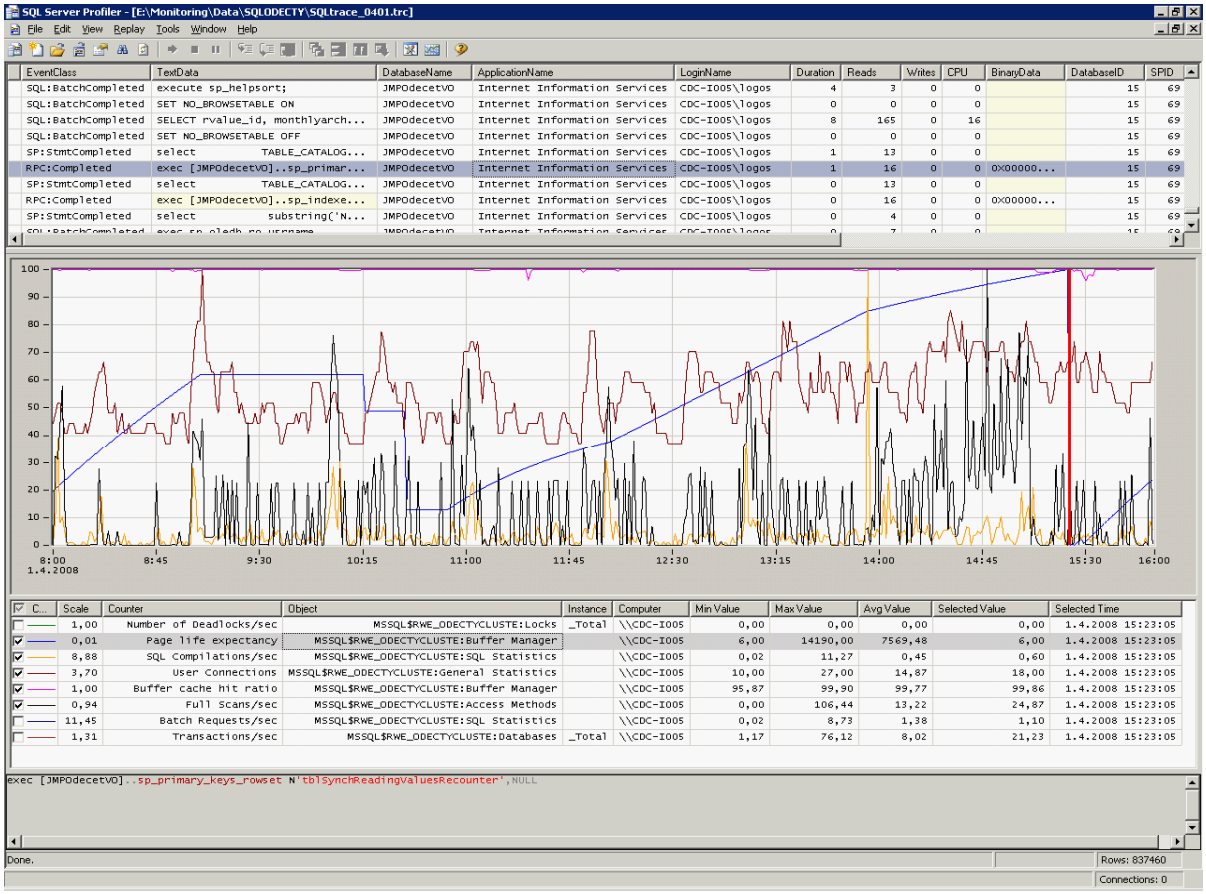
U všech těchto událostí jsem nastavil příslušné parametry: název a ID databáze, aplikace, login (pod jakým uživatelem byla daná událost vykonána), trvání, začátek a konec vykonání události, počet logických čtení vykonaných SQL serverem během události, počet fyzických zápisů a množství CPU času v ms.

Grafické rozhraní SQL Profileru umožňuje importovat výkonnostní data zaznamenané čítači systémového monitoru a přehledně tak zobrazit výstupy těchto dvou nástrojů dohromady, viz obrázky 6.7 a 6.8.

V horní části lze vidět události SQL Serveru, v prostřední části pak vybrané systémové čítače, kdy červená svislá linie ukazuje aktuálně zvolenou hodnotu (v prvním případě nejvíce vytížený disk – čítač Avg. Disk Qeue Length) a nakonec ve spodní části je zobrazena potenciální příčina, tedy dotaz, který byl v tuto dobu vykonán.



Obr. 6.7 SQL Profiler s načtenými systémovými čítači.



Obr. 6.8 SQL Profiler s načtenými čítači SQL Serveru.

Na obrázku 6.8 je pak zobrazen SQL Profiler s načtenými SQL čítači a vyznačena je kritická hodnota Page Life Expectancy 6/s. V tomto případě se ovšem nejednalo o příčinu SQL serveru, neboť jak je vidět z grafu, server již v tu dobu nebyl tak vytížen.

Tímto způsobem jsem ověřil všechny kritické hodnoty zaznamenané Performance Monitorem a zjišťoval jejich potenciální příčiny. Nevýhodou SQL Profileru je ovšem náročnost na filtrování již zaznamenaných stop, kdy každý dotaz způsobí kompletní načtení všech událostí a v našem případě, kdy se jednalo o řádově statisíce záznamů, trvalo načítání velice dlouhou a pro účely testování nepřípustnou dobu. Proto jsem dané stopy načetl do pomocných tabulek a dotazy pak prováděl přímo nad nimi, vše samozřejmě na testovacím serveru. Výsledkem byla tabulka zobrazující nejvíce náročné procedury či dotazy z hlediska prováděcího času, vytíženosti CPU a V/V (především zápisů) pro jednotlivé servery, viz část tabulky pro server SQL1:

Detekce dle	Událost	Hodnota [ms]	Aplikace	Účet
času	ap_regenerateStatistics	107510807	TSQL JobStep	cluster
	ap_prmPermissionUserGetAccess	259988	.Net SqlClient Data Provider	xadmin
	ap_prmUserSetConfigData	200000	.Net SqlClient Data Provider	xadmin
	ap_synchronizationEnd	4014088	IIS	SQL1\user
	TSQL v SP	101563492	TSQL JobStep	cluster
	TSQL batch	61360554	.Net SqlClient Data Provider	x_user
CPU	ap_regenerateStatistics	107203	TSQL JobStep	cluster
V/V	ap_synchloginsert	757	.Net SqlClient Data Provider	x_user
	TSQL v SP	400	.Net SqlClient Data Provider	x_user

Tab. 6.1 Příklad výstupu analýzy pomocí SQL Profileru

U všech událostí jsem zapsal typ aplikace, která danou proceduru či dávku vykonala a pod jakým účtem. Tímto způsobem vznikly tabulky pro všechny servery a ty poté byly základem pro proces optimalizace.

6.3 Sledování databázových parametrů

Důležitým bodem monitoringu je i sledování a ověření základních databázových parametrů, zejména vytíženost jednotlivých databází, využití paměti a zjištění parametrů optimalizátoru. Tyto údaje jsem zjišťoval pomocí tzv. Dynamic Management Views (DMV – dynamické pohledy), jež umožňují zjistit podrobnější informace o stavu serveru a jednotlivých databází, využití prostředků, aj. a tím diagnostikovat případné výkonnostní problémy. Tyto pohledy jsou však při každém restartu serveru vynulovány a tento fakt se musí brát v potaz (v případě delšího nepřerušovaného běhu serveru máme solidní vzorek dat pro představu zátěže, v opačném případě není doporučeno se na tyto data spoléhat). Proto jsem při jejich použití nejprve ověřil, jak dlouho daný server běží. K tomu jsem využil dotaz na datum vytvoření databáze *tempdb,* neboť ta je při každém restartu znovu vytvořena.

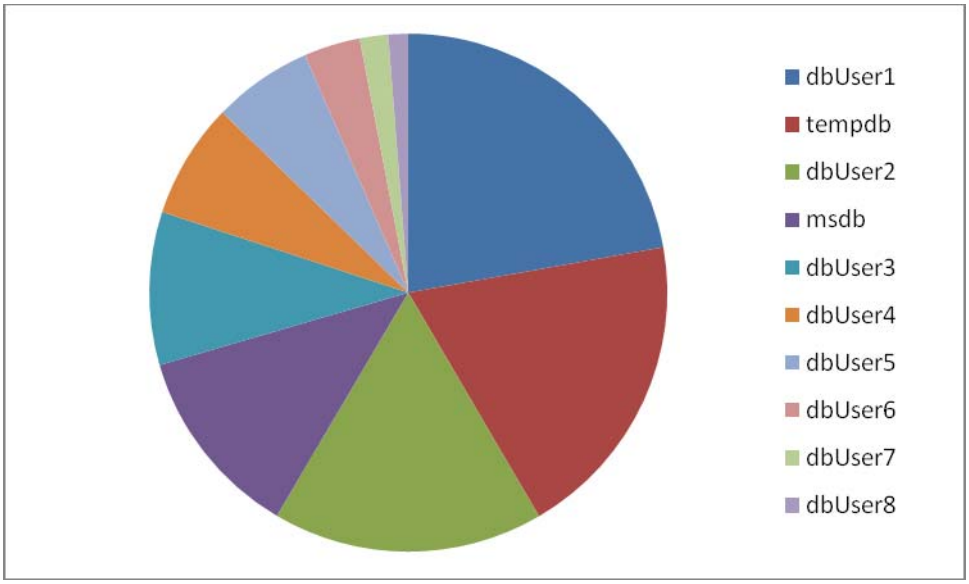
Pomocí pohledu *sys.dm_io_virtual_file_stats* (Příloha 1 – Analýza databáze) jsem vytvořil dotaz vracející informace o nejvytíženějších databázích daného serveru z hlediska V/V přístupu k datovým souborům databáze, viz následující tabulka (jména databází byla pozměněna):

P.	Databáze	Typ souboru	Čtení		Zápis		Celkem V/V		
			[MB]	Počet	[MB]	Počet	[MB]	Čekání [s]	Čekání [%]
1	dbUser1	DATA	4236,16	24052	1033,41	10485	5269,56	2536,2	21,46
2	tempdb	DATA	2551,81	41566	2620,54	41671	5172,35	2215,95	18,75
3	dbUser2	DATA	1496,43	7394	565,24	6063	2061,67	1931,22	16,34
4	msdb	DATA	1283,91	8634	25,68	2768	1309,59	1372,54	11,62
5	dbUser3	DATA	1621,2	7586	502,56	5450	2123,77	1091,72	9,24
6	dbUser4	DATA	1146,27	4697	484,93	4861	1631,2	825,07	6,98
7	dbUser5	DATA	1805,52	15037	75,41	4073	1880,94	710,07	6,01
8	dbUser6	DATA	540,09	3035	195,64	1569	735,73	401,29	3,4
9	dbUser7	LOG	1073,48	2174	1077,42	24347	2150,9	202,09	1,71
10	dbUser8	LOG	558,29	1525	562,9	14364	1121,19	141,6	1,2

Tab. 6.2 Vytíženost databází

V tabulce jsou seřazeny jednotlivé databáze dle vytížení, je uveden příslušný databázový soubor (datový nebo transakční protokol – log soubor), počet V/V operací a jejich velikost a nakonec čekání na dokončení V/V operace.

Následující graf znázorňuje nejvíce vytížené databáze z pohledu čekání na dokončení V/V operací:



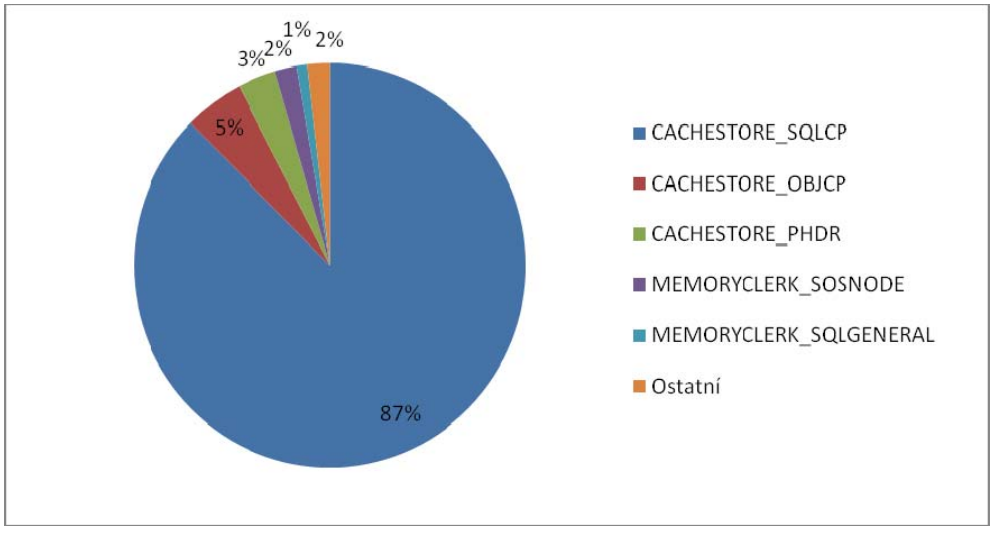
Obr. 6.9 Graf vytíženosti databází

Tímto způsobem jsem ověřil všechny analyzované servery a zjistil, na které databáze se mám nejvíce zaměřit.

Následující tabulka ukazuje využití paměti jednotlivými komponentami serveru (v kB):

Název objektu	Typ objektu	Využití paměti
SQL Plans	CACHESTORE_SQLCP	644 496
Object Plans	CACHESTORE_OBJCP	70 512
Bound Trees	CACHESTORE_PHDR	34 472
SOS_Node	MEMORYCLERK_SOSNODE	15 672
Default	MEMORYCLERK_SQLGENERAL	9 072
Ostatní	-	28 580

Tab. 6.3 Využití paměti SQL Serveru



Obr. 6.10 Graf využití paměti SQL Serveru

Nejvíce paměti zabírají SQL příkazy a dávky, jež nejsou uloženy v procedurách, funkcích či triggerech, dále pak zkompilované plány pro procedury a třetí objekt jsou algebraické stromy pro pohledy a omezení.

Pomocí DMV *sys.dm_exec_query_optimizer_info* jsem zjistil statistiky optimalizátoru SQL serveru. Pro získání ucelenějšího pohledu na práci optimalizátoru je nutno daný dotaz provádět pravidelně, viz následující tabulka:

Parametr		**Výstup**					
		31.3.	**1.4.**	**Rozdíl**	**2.4.**	**4.4.**	**Rozdíl**
Celkový počet optimalizací		128341	146957	18616	12517	28393	15876
z toho pro dotaz zahrnující	INSERT	82198	88774	6576	6110	16438	10328
	DELETE	619	1153	534	114	189	75
	UPDATE	10219	12163	1944	908	2215	1307
	poddotaz	5146	6770	1624	2134	2660	526
Počet tabulek odkaz. dotazem		1,259	1,339	0,080	2,369	1,723	-0,646
Počet pohledů odkaz. dotazem		18717	24875	6158	3617	5682	2065
Počet optimalizací pro vzdálený datový zdroj		21	73	52	9	12	3
Počet optimalizací pro dynamický kurzor		6	32	26	4	4	0
MAXDOP hodnota pro optimalizovaný plán		0,0017	0,0026	0,0021	0,0046	0,0020	0,0033
AVG čas na opt. 1 dotazu [ms]		18,307	19,374	18,841	57,273	50,418	53,845
AVG cena optimalizace		0,166	0,223	0,194	0,482	0,265	0,373

Tab. 6.4 Základní parametry optimalizátoru

Z této tabulky lze vyčíst, že optimalizátor serveru SQL1 provede denně zhruba 15000 optimalizací, z toho nejvíce pro operaci INSERT a jedna optimalizace průměrně zabere 25ms.

6.4 Analýza dotazů a indexů pomocí DMV

Po analýze dotazů pomocí SQL Profileru jsem provedl analýzu také pomocí DMV, ale jak již bylo zmíněno, vzhledem k vynulování jejich hodnot po restartu je nelze brát jako nejpřesnější údaje. Výsledné údaje mi tedy posloužily zejména jako ověření výsledku získaných pomocí SQL Profileru, pro zjištění využitelnosti a fragmentace indexů jsou ovšem tyto DMV jediným možným způsobem detekce.

6.4.1 Detekce nejnáročnějších procedur

Nejprve jsem analyzoval dotazy a uložené procedury z pohledu nejvíce vytěžujících CPU a V/V prostředků. Využil jsem k tomu DMV *sys.dm_exec_query_stats, sys.dm_exec_sql_text a sys.dm_exec_query_plan* (Příloha 1 – Detekce nejpomalejších dotazů), které spolu v kombinaci umožní zobrazit tzv. SQL, resp. plan handle a díky tomu lze pak následně určit příslušnou proceduru dané databáze. SQL_handle představuje část kódu odkazující se na dávku či uloženou proceduru, plan_handle pak tzv. token, který odkazuje na kompilovaný plán, jehož je dotaz součástí.

Vytvořil jsem komplexní dotaz, jenž mi umožnil detekci dle mnou zvolených parametrů v klauzuli *ORDER BY:*

```
SELECT
  CASE WHEN dbid = 32767 THEN 'Resource' ELSE db_name(dbid) END AS 'Databaze',
  object_name(objectid,dbid) AS 'Fce/Proc',
  SUM(usecounts) AS 'Pocet pouziti',
  SUBSTRING(CONVERT(char(23),DATEADD(ms,SUM(total_elapsed_time)/1000,0),121),12,23)
    AS 'Celk. provadeci cas',
  SUBSTRING(CONVERT(char(23),DATEADD(ms,SUM(total_elapsed_time/usecounts)/1000,0),121),12,23)
    AS 'Prum. provadeci cas',
  SUM(total_logical_reads) / SUM(usecounts) * 1.0 AS 'Prum. pocet cteni',
  SUM(total_logical_writes) / SUM(usecounts) * 1.0 AS 'Prum. pocet zapisu',
  SUM(total_worker_time) / SUM(usecounts) * 1.0 AS 'CPU vytizeni',
  SUM(plan_generation_num) AS 'Pocet rekompilaci',
  dbid, objectid
FROM
  sys.dm_exec_cached_plans cp CROSS APPLY
  sys.dm_exec_sql_text(cp.plan_handle) JOIN
    (SELECT
       SUM(total_elapsed_time) AS total_elapsed_time,
       SUM(total_logical_reads) AS total_logical_reads,
       SUM(total_logical_writes) AS total_logical_writes,
       SUM(total_worker_time) AS total_worker_time,
       SUM(plan_generation_num) AS plan_generation_num,
       plan_handle
     FROM sys.dm_exec_query_stats
     GROUP BY plan_handle)
  qs ON cp.plan_handle = qs.plan_handle
WHERE objtype = 'Proc'
GROUP BY dbid, objectid
ORDER BY SUM(total_elapsed_time) / SUM(usecounts) * 1.0 DESC;
```

Na začátku tohoto dotazu kontroluju dbid a pokud je rovno 32767, pak se jedná o systémovou databázi, která je normálně skryta (nelze ji např. nalézt v Management studiu, její datové soubory se nazývají *mssqlsystemreource*) a nazývá se „Resource“ database. Tato databáze obsahuje všechny kompilované systémové uložené procedury a funkce. Pro zajištění spojení výstupu pohledu *sys.dm_exec_sql_text* s informací o uloženém plánu v pohledu *sys.dm_exec_cached_plans* používám operátor *CROSS APLLY*. Tímto dotazem jsem tedy zkontroloval nejnáročnější dotazy z pohledu času vykonání (viz obr. 6.11), spotřebovaných V/V prostředků, času CPU (*total_worker_time*) a počtu rekompilací (*plan_generation_num*).

Results | Messages

	Databaze	Fce/Proc	Pocet pouziti	Celk. provadeci cas	Prum. provadeci cas	Prum. pocet cteni	Prum. pocet zapisu	CPU vytizeni	Pocet rekompilaci
1	RWEOdectySMP	ap_regenerateStatistics	211	00:17:19.240	00:00:04.927	600000.0	1.0	4797186.0	7
2	RWEOdectySCP	ap_regenerateStatistics	211	00:09:43.413	00:00:02.763	271727.0	0.0	2722601.0	434
3	RWEOdectySTP	ap_regenerateStatistics	211	00:07:24.667	00:00:02.107	186773.0	0.0	2104085.0	427
4	RWEOdectyVCP	ap_regenerateStatistics	211	00:06:29.507	00:00:01.847	163504.0	0.0	1844714.0	434
5	RWEOdectySCP	fn_gettraceanomalystatus	211	00:05:17.853	00:00:01.507	66143.0	0.0	1489574.0	12

Obr. 6.11 Příklad výstupu detekce nejpomalejších procedur

Při vykonání tohoto dotazu jsem si všiml, že několik výsledných řádků obsahuje stejnou proceduru či dávku. Důvodem je ukládání několika různých plánů v procedurální cache pro tutéž uloženou proceduru. Výsledné záznamy se shodovaly s výsledky pomocí nástroje SQL Profiler.

Dále jsem provedl detekci plánů, jež by mohly běžet paralelně a to pomocí dotazu na pohled *sys.dm_exec_cached_plans* hledáním, zda relační operátor má v uložených plánech atribut *Parallel* nenulovou hodnotu:

```
WHERE
  cp.cacheobjtype = 'Compiled Plan' AND qp.query_plan.value
  ('declare namespace p="http://schemas.microsoft.com/sqlserver/2004/07/showplan";
    max(//p:RelOp/@Parallel)', 'float') > 0
```

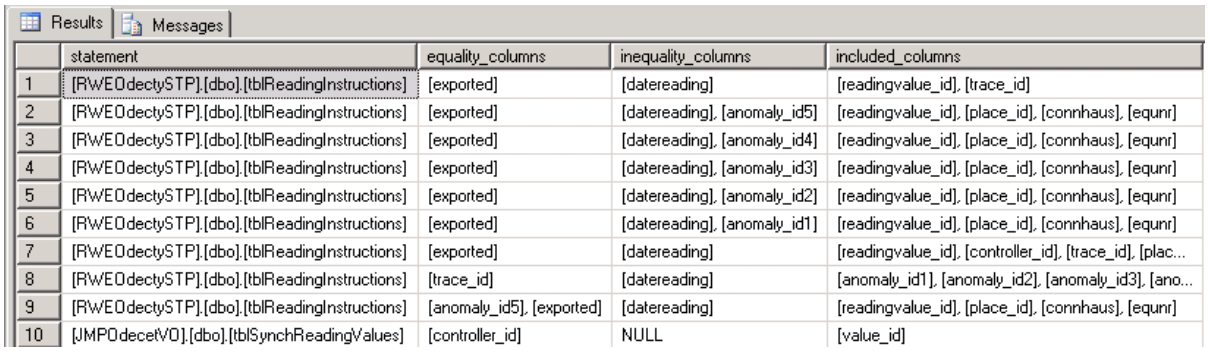
Výsledkem byly dávky či procedury, které namísto paralelního běhu byly provedeny sekvenčně. Důvodem může být např. to, že optimalizátor usoudil, že sekvenční provedení je cenově výhodnější, nebo v případě, že server v danou chvíli nedisponoval prostředky pro paralelní běh. Tuto skutečnost lze ovlivnit tzv. query hints, ale pouze nastavením parametru MAXDOP, jenž určuje, kolik procesorů bude využito pro paralelní zpracování.

6.4.2 Četnost použití a fragmentace indexů

Detekce statistik indexů je užitečná pro optimalizaci výkonnosti dotazů, tyto informace mohou napomoct ke zjištění užitečnosti a ceny indexů.

Nejprve jsem určil míru využití jednotlivých indexů ve všech databázích pomocí DMV *sys.dm_db_index_usage_stats*, jenž zobrazuje počet prohledávání (operace *seek* a *scan*) uskutečněných uživateli i systémem a čas posledního provedení této operace (poslední přístup k indexu). Lze tak získat přehled o užitečnosti jednotlivých indexů, musíme však brát v potaz, že jako každý jiný DMV i tento je po restartu serveru vynulován. Takto jsem získal přehled o indexech, jež měly četnost použití nulovou (user a system seek i scan) a pokud se jednalo o neshlukující indexy (nonclustered), pak jsem dal návrh na jejich odstranění.

Pomocí pohledu *sys.dm_db_missing_index_details* jsem získal přehled o všech chybějících indexech, viz následující ukázka výpisu výsledku:



Results | Messages

	statement	equality_columns	inequality_columns	included_columns
1	[RWEOdectySTP].[dbo].[tblReadingInstructions]	[exported]	[datereading]	[readingvalue_id], [trace_id]
2	[RWEOdectySTP].[dbo].[tblReadingInstructions]	[exported]	[datereading], [anomaly_id5]	[readingvalue_id], [place_id], [connhaus], [equnr]
3	[RWEOdectySTP].[dbo].[tblReadingInstructions]	[exported]	[datereading], [anomaly_id4]	[readingvalue_id], [place_id], [connhaus], [equnr]
4	[RWEOdectySTP].[dbo].[tblReadingInstructions]	[exported]	[datereading], [anomaly_id3]	[readingvalue_id], [place_id], [connhaus], [equnr]
5	[RWEOdectySTP].[dbo].[tblReadingInstructions]	[exported]	[datereading], [anomaly_id2]	[readingvalue_id], [place_id], [connhaus], [equnr]
6	[RWEOdectySTP].[dbo].[tblReadingInstructions]	[exported]	[datereading], [anomaly_id1]	[readingvalue_id], [place_id], [connhaus], [equnr]
7	[RWEOdectySTP].[dbo].[tblReadingInstructions]	[exported]	[datereading]	[readingvalue_id], [controller_id], [trace_id], [plac...
8	[RWEOdectySTP].[dbo].[tblReadingInstructions]	[trace_id]	[datereading]	[anomaly_id1], [anomaly_id2], [anomaly_id3], [ano...
9	[RWEOdectySTP].[dbo].[tblReadingInstructions]	[anomaly_id5], [exported]	[datereading]	[readingvalue_id], [place_id], [connhaus], [equnr]
10	[JMPOdecetVO].[dbo].[tblSynchReadingValues]	[controller_id]	NULL	[value_id]

Obr. 6.12 Výpis chybějících indexů

Výsledkem tohoto DMV je název tabulky, na kterou se vztahují chybějící indexy a následně tři typy sloupců, jež by mohly být součástí nově vytvořeného indexu:

- Equality: dotaz na shodu a tedy *tabulka.sloupec = konstanta*
- Inequality: neshoda, např. *tabulka.sloupec > konstanta*
- Included: tzv. pokrývající (covered) sloupce

První dva zmíněné typy sloupců se mohou použít jako základ indexu, poslední typ se použije v klauzuli *INCLUDED*. Výstup tohoto DMV jsem poté porovnal s výsledky nástroje Database Tuning Advisor a s výsledky ladění konkrétních dotazů.

Následně jsem dotazem nad DMV *sys.dm_db_index_physical_stats* určil míru defragmentace jednotlivých indexů.

```
SELECT
  object_name(i.object_id) AS Tabulka,
  i.name AS 'Index',
  sip.index_level as 'Uroven',
  sip.avg_page_space_used_in_percent AS 'Mira zaplneni',
  sip.avg_fragmentation_in_percent AS 'Externi (logicka) fragmentace',
  sip.fragment_count AS 'Pocet fragmentu',
  sip.avg_fragment_size_in_pages AS 'Pocet stranek na fragment',
  sip.ghost_record_count AS 'Duchove'
FROM
  sys.dm_db_index_physical_stats(db_id(), NULL, NULL, NULL, 'DETAILED') sip INNER JOIN
  sys.indexes i
    ON i.object_id = sip.object_id
    AND i.index_id = sip.index_id
--WHERE sip.avg_fragmentation_in_percent > 50
ORDER BY sip.avg_fragmentation_in_percent DESC
```

Nejprve jsem se zaměřil na sloupec *avg_page_space_used_in_percent*, který udává jasnou informaci o interní fragmentaci dané úrovně indexu. Interní fragmentace se projeví tehdy, kdy datové stránky alokované pro jednotlivé úrovně indexu nejsou plně využity. Tato skutečnost se může projevit snížením rychlosti při provádění dotazu nad seřazenými hodnotami, kdy SQL server musí číst všechny datové stránky indexu, ačkoli je jich potřeba jen několik.

Pokud je hodnota tohoto sloupce na úrovni listů (index level = 0) více než 90%, pak je index skoro zaplněn a tím pádem zde není téměř žádná fragmentace. Hodnota pod 75% značí interní fragmentaci a řešením může být reorganizace (doporučuje se provádět v případě míry fragmentace mezi 60 až 75%) nebo znovu sestavení indexu (rebuild, v případě míry pod 60%).

Pro určení externí fragmentace je užitečný sloupec *avg_fragmentation_in_percent*. Pokud se jedná o fragmentaci na úrovni listů, pak se jedná o logickou fragmentaci (nesoulad v logickém pořadí indexových stránek vůči fyzickému), v případě že je fragmentována hromada tabulky, pak se tato fragmentace označuje jako extent. Obecně hodnota nad 10% na úrovni listů se považuje za logickou fragmentaci.

Interní fragmentace nemusí být ovšem vždy být na škodu, naopak může pomoct k eliminaci externí (logické) fragmentaci, např. když je index interně fragmentován, pak datové stránky mají místo pro nové záznamy a výsledkem je, že není třeba rozdělit žádné stránky pro nové záznamy.

Sloupce *fragment_count* a *avg_fragment_size_in_pages* zobrazují počet fragmentů na úrovní listů, resp. průměrný počet stránek na fragment. Fragment značí množinu posloupných stránek v jednom souboru, maximální počet fragmentů daného indexu je roven počtu stránek na úrovni listů. Pokud hodnota *avg_fragment_size_in_pages* je mezi 8 a 32, pak je odhadovaný výkon operací SELECT považován za uspokojivý.

Hodnoty sloupce *ghost_record_count* mohou být užitečné k nalezení záznamů, jež byly odstraněny, ale fyzicky jsou stále dostupné. Tyto záznamy mohou být zdrojem určitých výkonnostních problémů, protože zabírají místo pro nové záznamy. SQL server je ovšem automaticky fyzicky odstraní po určité době, kdy má dostatek prostředků pro tuto operaci.

Results | Messages

	Tabulka	Index	Uroven	Mira zaplneni	Externi (logicka) fragmentace	Pocet fragmentu	Pocet stranek na fragment	Duchove
1	tblSynchronizeTime	PK_tblSynchronizeTime2	0	90,560217445021	99,0521327014218	210	1,0047619047619	1
2	tblTraceArchive	PK_tblTraceArchive	0	90,0470101309612	98,936170212766	94	1	0
3	tblTraceStatisticsArchive	PK_tblTraceStatisticsArchive	0	98,8894860390413	97,1428571428571	35	1	0
4	tblTrace	PK_tblTrace	0	98,8126142821843	96,4285714285714	28	1	0
5	tblSynchronizeGuid	NULL	0	77,1625277983692	95,4545454545455	22	4	0
6	tblGasometerType	PK_tblGasometerType	0	95,8611317024957	83,3333333333333	6	1	0
7	tblTraceStatistics	PK_tblTraceStatistics	0	93,9965900667161	58,3333333333333	8	1,5	0
8	tblReadingInstructions	PK_tblReadingInstructions	1	91,1732517914505	12,5	8	2	0
9	tblReadingInstructionsArchive	PK_tblReadingInstructionsArchive	1	94,5332097850259	7,14285714285714	8	7	0

Obr. 6.13 Příklad výstupu dotazu ohledně fragmentace indexů

Tímto způsobem jsem zaznamenal všechny indexy v dané databázi a dle míry interní a externí fragmentace jsem vytvořil doporučení pro znovu sestavení indexů, popř. reorganizaci.

7 Optimalizace

Optimalizace probíhala na testovacím stroji, na kterém byly pomocí technologie TSM nahrány všechny databáze. Server měl téměř identickou sestavu jako servery analyzované, a tudíž bylo možno výsledné reporty a naměřené hodnoty použít k posouzení úspěšnosti optimalizace.

7.1 Optimální nastavení serveru

Prvním krokem optimalizace celého databázového systému bylo zaměření se na systémovou úroveň výkonnosti serveru, která zahrnuje počet a typ procesorů, velikost paměti, typ souborového systému, apod. Většinu výkonnostních problémů ovšem nelze vyřešit těmito nastaveními, proto je důležité se také zaměřit a analyzovat aplikaci a dotazy nad danou databází.

V tomto bodě nebylo nalezeno kritické slabé místo, eventuální odchylky zaznamenané čítači byly dle podrobnějšího prostudování způsobeny spíše než nedostatkem systémových prostředků aspekty na databázové úrovni. Proto jsem se nejvíce zaměřil na optimalizaci dotazů a s ní související správu indexů. Vyzkoušel jsem ovšem všechny možnosti nastavení, jež SQL Server poskytuje pro dosažení optimálního běhu serveru.

7.1.1 Konfigurace procesorů

SQL Server 2005 podporuje symetrický multiprocesing a umí zpracovávat složité paralelní dotazy. Ty mají smysl v případě, že se systémem pracuje relativně malý počet uživatelů a zpracovávají se velké dotazy.

Všechny analyzované servery obsahují více procesorů (zpravidla čtyři) a lze tak SQL Server nastavit kdy a za jakých okolností je bude využívat. Jak již bylo zmíněno v teoretické části, SQL Server využívá k optimalizaci využití procesorů jejich afinitu (afinita procesoru pro zpracování vláken a afinita pro V/V operace). Vyzkoušel jsem nastavení priorit, kdy dva procesory byly určeny pro zpracování vláken a zbylé dva pro V/V operace bez přeskakování. Využil jsem k tomu uloženou proceduru *sp_configure*:

```
exec sp_configure 'affinity mask', 10
exec sp_configure 'affinity I/O mask', 5
```

kde celočíselná hodnota představuje bitovou masku procesoru. V tomto případě jsem pro zpracování vláken určil procesory CPU1 a CPU3 (bitová maska 1010 => 10) a pro zpracování V/V operací CPU 0 a CPU2. Při testování s větší zátěží ovšem nedošlo k lepším výsledkům, nastavení afinit je lépe využitelné v případě většího počtu procesorů a tudíž jsem ponechal řízení nastavení afinit automaticky SQL Serverem.

Dále jsem se zaměřil na nastavení paralelního zpracování. SQL Server provádí dotazy paralelně v případě, že počet procesorů je vyšší než počet aktivních spojení a pokud je odhadnutý čas potřebný k provedení dotazu sériově vyšší než úroveň plánu dotazu. Navíc musíme mít na paměti, že ne všechny dotazy lze zpracovávat paralelně, např. příkazy UPDATE, INSERT či DELETE se implicitně provádí sekvenčně. Pokud ovšem obsahují klauzuli WHERE (v případě příkazu INSERT se jedná o klauzuli SELECT), pak se také mohou provádět paralelně.

Všechny analyzované servery měly nastavení maximálního stupně paralelního zpracování (*Max Degree of Parallelism*) na hodnotu 0, což znamená automatické řízení všech dostupných procesorů při paralelním zpracování. V případě nastavení na hodnotu 1 se použil pouze jeden procesor (vypnut paralelismus). Další možností nastavení je tzv. Práh odhadovaných nákladů (*Cost Threshold for Parallelism*), jenž se uplatní při porovnání počtu sekund potřebného ke spuštění sériového plánu s náklady pro paralelní zpracování. Pokud je odhad sériového plánu vyšší než nastavený práh, pak se daný dotaz provede paralelně. Lze jej nastavit opět pomocí procedury *sp_configure,* implicitní hodnota je 5:

```
exec sp_configure 'cost threshold for parallelism', <0-32767>
```

Poslední možností pro optimalizaci využití procesorů byla konfigurace vláken a priorit serveru. Vlákna jsou důležitou součástí operačního systému podporujícího multitasking. Představují cesty paralelního provádění kódu a umožňují aplikacím efektivněji využít procesor. SQL Server je nastaven na srovnávání vláken s připojením uživatelů, v případě dostačujícího počtu vláken (počet vláken je větší než počet připojení) se každé uživatelské připojení obsluhuje zvlášť. V opačném případě se vlákna musí sdílet a může tak docházet ke snížení výkonu a dobu odpovědi. Lze nastavit maximální počet vláken, jež bude SQL Server obsluhovat, v případě 32bitových systémů doporučuje Microsoft nastavení na hodnotu 1024 (32^2):

```
exec sp_configure 'max worker threads', 1024
```

Zvýšení výkonu lze také dosáhnout pomocí zvětšení priority vláken, kdy dané vlákno dostane vyšší preference při získávání procesorového času a ostatní vlákna ho nepřeruší. Na vyhrazených systémech, kde běží pouze SQL Server, lze dosáhnout lepších výsledků, v případě běhu ostatních aplikací se ovšem může jejích výkon snížit. Vlákna SQL Serveru mají standardně prioritu 7, v případě povolení zvýšení priority (nastavení na 1) je jejich hodnota 13.

```
exec sp_configure 'priority boost', 1
```

Z důvodu přepínání vláken mezi uživatelským kontextem a jádrem dochází ke spotřebě času a zdroje procesoru. Proto SQL Server umožňuje používání tzv. nitek, kdy aplikace pracuje s vlákny přímo a nedochází tak ke změně režimu. Server alokuje pro každý procesor jedno vlákno a pro každé paralelní uživatelské spojení jednu nitku. Nitky fungují lépe, pokud má server více procesorů a nízký

poměr uživatelů k procesorům (dle [3] je ideální poměr kolem hodnoty 10). Povolení používání nitek lze nastavit následovně:

```
exec sp_configure 'lightweight pooling', 1
```

Poslední tři nastavení, týkající se vláken a nitek, jsem doporučil na serveru SQL3, který zvláště pro poslední zmiňovanou konfiguraci má ideální předpoklady (4 CPU a průměrně 20 uživatelů). Na testovacím stroji jsem ovšem nemohl eventuální vylepšení vyhodnotit z důvodu těžko realizující simulace připojených uživatelů, a proto jsem byl odkázán na reporty z tohoto produkčního systému.

7.1.2 Optimalizace využití paměti

Jak bylo uvedeno v analýze, servery jsou nastaveny na dynamické řízení paměti. Změny v nastavení v této sekci se moc nedoporučují, neboť SQL Server by pak mohl alokovat příliš mnoho paměti (vytvořil by se nedostatek potřebných zdrojů pro jiné aplikace a zvýšilo by se tak stránkování paměti) nebo naopak příliš málo paměti (SQL Server by neobsluhoval úkoly včas). Lze nastavit minimální a maximální alokovanou paměť, povolení paměti AWE a alokaci paměti pro dotazy.

Na systému SQL3, kde běží pouze SQL Server jsem doporučil nastavení minimálního množství paměti dle vzorce uvedeného v kapitole 4.1.3 Správa a optimalizace fyzických zdrojů:

```
8MB + (24kB × Počet uživatelů) = 8MB + 24kB × 20 = 8,5MB
```

a následné nastavení pomocí procedury *sp_configure:*

```
exec sp_configure 'min server memory', 9
```

V tomto případě při menší zátěži již nedojde k uvolnění této paměti. Maximální hodnotu alokované paměti je vhodné nastavit v případě, že na daném serveru běží více instancí a zajistit tak, aby instance o paměť nesoutěžily. V analyzovaném enterprise prostředí ovšem využívají všechny databázové systémy pouze jednu instanci. Lze také vypnout dynamickou konfiguraci nastavením minimální a maximální hodnoty na stejnou velikost. SQL Server pak využívá pevnou množinu paměti, OS neodsouvá paměťové stránky na disk a tím pádem je omezeno stránkování a čtení do vyrovnávací paměti.

Podpora paměti typu AWE je na všech analyzovaných server zakázána. Jedná se o API, jež umožňuje 32bitovým aplikacím využívat více fyzické paměti, než má virtuální adresový prostor, konkrétně 4GB. Je-li podpora AWE povolena, SQL Server při svém spuštění dynamicky alokuje paměť AWE a podle potřeby uvolňuje v rámci omezení nastaveného pro minimální a maximální dostupné množství paměti. Cílem je vyvážit využití paměti SQL serverem s požadavky celého systému. Microsoft doporučuje povolení AWE na SQL Serverech běžících na Windows Server 2003. Její povolení lze provést následovně:

```
exec sp_configure 'awe enabled', 1
```

Po povolení AWE se musí ověřit, zda účet, pod kterým SQL Server běží, má nastavena práva k uzamčení stránek v paměti (Lock Pages in Memory). To lze ověřit v nástroji Group Policy.

Jelikož jsem neshledal žádna úzká paměťová místa v analyzovaných systémech, toto paměťové nastavení jsem pouze uvedl do výsledného reportu jako doporučení a případná realizace se odvíjela od domluvy s aplikačními správci a programátory.

Poslední možností optimalizace paměti SQL Serveru byla alokace paměti pro dotazy a indexování. Implicitně se alokuje pro vykonání dotazu 1024kB, přičemž tato hodnota je zaručena pro každého uživatele a může být nastavena v rozmezí 512kB – 2GB. Zvýšením alokace lze dosáhnout zlepšení výkonu dotazů provádějících náročné operace vyžadující mnoho času procesoru (např. třídění, hašování, spousta paralelních dotazů), v případě příliš vysoko nastavené hodnoty se ovšem může snížit výkon celého systému. Dle [3] je vhodné použít jako startovací bod optimalizace následující rovnici:

$$\frac{\text{Volna pamet}}{\text{Prumerna velikost dotazu} \times \text{Prumerny pocet paralel. dotazu}}$$

V analyzovaném prostředí bylo nejvíce paralelních dotazů provedeno na serveru SQL1 (zhruba 10), volná paměť se pohybovala kolem 1000MB a průměrná velikost dotazu činila 2MB. Výsledná nastavitelná hodnota je tedy rovna 50MB, což představuje maximum, které by se mohlo v daném prostředí přiřadit.

```
exec sp_configure 'min memory per query', 51200
```

Problém s rychlostí vykonávání dotazů ovšem nebyl tak markantní a spíše jsem se zaměřil na ladění konkrétních dotazů pomocí analýzy struktury, výše uvedené nastavení jsem opět použil jako doporučení.

7.1.3 Konfigurace V/V subsystému

Ověřil jsem také možnosti nastavení V/V subsystému. Jak bylo uvedeno v analýze, servery využívají hardwarový RAID 1, jenž je dražší než softwarové řešení, ale poskytuje mnohem vyšší výkon.

Je doporučeno oddělit datové (mdf) a transakční (ldf) soubory databáze, podobně je vhodné umístit datové soubory databáze tempdb na jiný disk než jsou data ostatních databází z důvodu paralelního přístupu. Tento požadavek ovšem nešlo realizovat z důvodu omezených možností diskového pole připojeného do SAN. Dle analýzy diskových čítačů jsem ovšem neshledal žádné často se opakující kritické události.

7.2 Použité techniky ladění dotazů

Po identifikaci nejpomalejších dotazů pomocí DMV a SQL Profileru následovala jejich optimalizace. Laděním nejhůře vykonávaných dotazů může dojít k velkému navýšení výkonu celé aplikace při víceméně malých nákladech. Tyto dotazy s nežádoucí velkou prováděcí dobou mohou být způsobeny:

- Pomalou síťovou komunikací
- Nedostatkem pamětí
- Nedostatkem nebo neaktuálními statistikami
- Nedostatkem nebo špatně navrhnutými indexy

Statistiky distribuce hodnot ve sloupcích jsou automaticky vytvářeny pro indexované sloupce. Lze je ovšem vytvořit také pro neindexové sloupce příkazem *CREATE STATISTICS* (v případě že je nastavena volba *auto update statistics*, jsou statistiky vytvořeny automaticky). Tyto statistiky využívá optimalizátor dotazů pro volbu optimální strategie k vykonání dotazu. Správa statistik neindexových sloupců zahrnutých v join operátorech může zvýšit výkon dotazu.

7.2.1 Zobrazení statistik dotazu

Optimalizace dotazů probíhala v několika iteracích, kdy při každé změně jsem musel ověřit eventuální změnu ve výkonnosti (ať už v negativním či pozitivním směru). Zaměřil jsem se především na tři nejdůležitější aspekty, díky kterým jsem i analyzoval dotazy: prováděcí čas, V/V prostředky a CPU čas.

- Prováděcí čas dotazu lze zkontrolovat v Managament studiu, přesnější výsledek lze ovšem získat pomocí příkazu *SET STATISTICS TIME ON*. Výsledkem je pak množství času pro syntaktickou analýzu, kompilaci a nakonec provedení dotazu. V případě, že ovšem chceme změřit prováděcí část nejen pouze pro určitý SQL příkaz ale pro komplexní uloženou proceduru, kdy chceme zjistit, která její část je prováděna nejdéle, pak lze použít:

```
DECLARE @Start as DATETIME
SET @Start = GETDATE()
...
Print 'Provadeci cas v ms: '+ CAST(DATEDIFF(ms,@Start,GETDATE())
as char (10))
```

- Množství CPU, které je potřeba pro vykonání dotazu či procedury, je dalším důležitým aspektem pro ladění. Jeho informaci lze získat podobně jako v případě prováděcího času příkazem *SET STATISTICS*.

SQL server ovšem poskytuje systémovou proměnnou *@@CPU_BUSY*, která udává celkové množství CPU času od posledního startu serveru. Pro zjištění průměrného množství pro daný dotaz jsem použil následující konstrukci:

```
DECLARE @CPU_Start as int
SET @CPU_Start = @@CPU_BUSY
...
Print 'CPU v ms: '+ CAST((@@CPU_BUSY - @CPU_Start)*@@TIMETICKS /
1000 as char (10))
```

Tato metoda je ovšem nepřesná, neboť proměnná *@@CPU_BUSY* zaznamenává jakýkoliv běh na serveru a tudíž jsem získal pro stejný dotaz různé hodnoty.
Přesnější výsledky jsem získal pomocí SQL profileru, tyto hodnoty jsou ovšem uvedeny pro celý kompletní dotaz či proceduru, pro část kódu mi postačovala výše zmíněná dávka.

- Měření množství V/V prostředků využitých dotazem rozlišuje logické a fyzické V/V operace. První z nich znamená operace s daty v paměti a cache, fyzické operace představují přímý přístup na disk, kde je daná databáze uložena a také proto jsou náročnější než logické. V/V prostředky znamenají obecně nejdražší operace, které nejvíce ovlivní výkon TSQL dotazu a proto je důležité co nejvíce minimalizovat jejich počet.
  Možností pro zjištění jejich počtu je použití příkazu *SET STATISTICS IO ON*. Při testování jsem ovšem musel vzít na vědomí, že při opakovaném spouštění dotazů SQL server uchovává potřebná data v bufferu vyrovnávací paměti. Proto jsem vždy při optimalizační změně použil příkaz *DBCC DROPCLEANBUFFER*, který vyprázdní buffer cache bez nutnosti restartovat server.

Být schopen zjistit míru využití systémových prostředků různými dotazy je kritický bod analýzy výkonnosti databáze a aplikace. Vědět, které prostředky daný dotaz využívá nejvíc, pomáhá k přesnému zaměření se ohledně optimalizace dotazu a vyladění kódu.

7.2.2 Prováděcí plán dotazu a query hints

Vhodnou pomůckou při testování optimalizačních kroků je zobrazení prováděcího plánu. Lze ukázat jeho grafickou, textovou nebo XML podobu. V plánu jsem se vždy zaměřil na nejnáročnější operace, jež jsou ukázány v procentech vzhledem k celkové náročnosti dané dávky či dotazu. Snažil jsem se eliminovat náročné operace, jako jsou Clustered Index Scan, Table Scan apod.

Existuje několik variant pro zobrazení prováděcího plánu rozlišené v případě, zda byl pro jeho zobrazení dotaz proveden nebo je plán odhadnut. V první variantě lze použít příkaz *SET STATISTICS PROFILE ON,* kdy je zobrazen aktuální plán v textové podobě a jež poskytuje nejvíce informací (cena operací, použitý paralelismus apod.).

	Rows	Executes	StmtText	PhysicalOp	LogicalOp	Argument	TotalSubtreeCost
1	1909	1	INSERT tblTraceStatistics SELECT * from dbo.fn_TraceStatistics() order by trace_id	NULL	NULL	NULL	29,43569
2	1909	1	|--Clustered Index Insert(OBJECT:([RWEOdectyJMP].[dbo].[tblTraceStatistics].[PK_tblTraceStatisti...	Clustered Index Insert	Insert	OBJECT:([RWEOdectyJMP].[dbo].[tblTraceStatistics].[PK_t...	29,43569
3	0	0	|--Compute Scalar(DEFINE:([Expr1028]=isnull([Expr1012],(0)), [Expr1029]=isnull([Expr1018],(0)),...	Compute Scalar	Compute Scalar	DEFINE:([Expr1028]=isnull([Expr1012],(0)), [Expr1029]=isnull...	29,41637
4	1909	1	|--Top(ROWCOUNT est 0)	Top	Top	TOP EXPRESSION:((0))	29,41618
5	1909	1	|--Parallelism(Gather Streams, ORDER BY:([AA].[trace_id] ASC))	Parallelism	Gather Streams	ORDER BY:([AA].[trace_id] ASC)	29,41599
6	1909	2	|--Merge Join(Right Outer Join, MERGE:([RWEOdectyJMP].[dbo].[tblReadingInstructi...	Merge Join	Right Outer Join	MERGE:([RWEOdectyJMP].[dbo].[tblReadingInstructions].[t...	29,37233
7	0	0	|--Compute Scalar(DEFINE:([Expr1024]=[Expr1024]))	Compute Scalar	Compute Scalar	DEFINE:([Expr1024]=[Expr1024])	7,60103
8	0	0	| |--Compute Scalar(DEFINE:([Expr1024]=CONVERT_IMPLICIT(int,[globalagg10...	Compute Scalar	Compute Scalar	DEFINE:([Expr1024]=CONVERT_IMPLICIT(int,[globalagg10...	7,600961

Obr. 7.1 Aktuální prováděcí plán

Druhou možností pro zobrazení aktuálního plánu je použitím příkazu *SET STATISTICS XML ON*, jež zobrazí plán v podobě XML s informací navíc ohledně chybějících indexů (element MissingIndex):

```
- <QueryPlan DegreeOfParallelism="2" MemoryGrant="615" CachedPlanSize="115" CompileTime="430" CompileCPU="411"
  - <MissingIndexes>
    - <MissingIndexGroup Impact="18.6234">
      - <MissingIndex Database="[RWEOdectyJMP]" Schema="[dbo]" Table="[tblReadingInstructions]">
        - <ColumnGroup Usage="INEQUALITY">
            <Column Name="[datereading]" ColumnId="29" />
          </ColumnGroup>
        - <ColumnGroup Usage="INCLUDE">
            <Column Name="[trace_id]" ColumnId="3" />
            <Column Name="[anomaly_id1]" ColumnId="34" />
            <Column Name="[anomaly_id2]" ColumnId="35" />
            <Column Name="[anomaly_id3]" ColumnId="36" />
            <Column Name="[anomaly_id4]" ColumnId="37" />
            <Column Name="[anomaly_id5]" ColumnId="38" />
          </ColumnGroup>
        </MissingIndex>
      </MissingIndexGroup>
    - <MissingIndexGroup Impact="22.2733">
      - <MissingIndex Database="[RWEOdectyJMP]" Schema="[dbo]" Table="[tblReadingInstructions]">
        - <ColumnGroup Usage="INEQUALITY">
            <Column Name="[datereading]" ColumnId="29" />
          </ColumnGroup>
        - <ColumnGroup Usage="INCLUDE">
            <Column Name="[trace_id]" ColumnId="3" />
          </ColumnGroup>
        </MissingIndex>
      </MissingIndexGroup>
    - <MissingIndexGroup Impact="23.1616">
      - <MissingIndex Database="[RWEOdectyJMP]" Schema="[dbo]" Table="[tblReadingInstructions]">
        - <ColumnGroup Usage="EQUALITY">
            <Column Name="[datereading]" ColumnId="29" />
          </ColumnGroup>
        </MissingIndex>
      </MissingIndexGroup>
    </MissingIndexes>
```

Poslední možností je volba „Include Actual Execution Plan“ v Management studiu, jež zobrazuje aktuální prováděcí plán v grafické a velice přehledné podobě.

Pro volbu zobrazit pouze plán bez provedení dotazu lze použít následující příkazy:

- *SET SHOWPLAN_TEXT ON* - odhadovaný plán v textové podobě, dotaz není proveden

- *SET SHOWPLAN_XML ON* – odhadovaný plán v XML
- *SET SHOWPLAN_ALL ON* – odhadovaný plán v textové podobě s informacemi o ceně jednotlivých operací
- Volba „Display Estimated Execution Plan“ v Management Studiu – grafická nejvíce přehledná podoba.

Po zobrazení prováděcího plánu a zaměření se na kritické operace přicházelo na řadu samotné ladění dotazů. Zde jsem zkoumal zdrojový kód procedur, přičemž jsem zkoušel uplatnit i tzv. *query hints*, jež umožňují provést dotaz se specifickým parametrem použitým v klauzuli OPTION. Zde uvedu několik příkladů, jež jsem využil:

- *{ LOOP | MERGE | HASH } JOIN* – specifikace jakou metodou bude provedena operace JOIN.
- *FAST x* – dotaz bude optimalizován pro výpis prvních *x* řádků. Po jejich výpisu pokračuje dotaz pro výpis zbytku výsledku.
- *MAXDOP x* – přepíše nastavení úrovně paralelismu číslem *x* (počet procesorů použitých pro paralelní zpracování).
- *OPTIMIZE FOR* – optimalizátor použije specifickou hodnotu (námi zvolenou) pro lokální proměnnou (např. sloupec v klauzuli WHERE), když je dotaz zkompilován a optimalizován.

7.2.3 Doporučení při psaní dotazů

Existuje několik osvědčených doporučení a praktik, které jsem při optimalizaci dotazů využíval a zkoušel aplikovat. Nyní zde uvedu několik příkladů, jež jsem uplatnil a které se ukázaly jako užitečné.

- Eliminace dotazu SELECT *

  Použití dotazu s klauzulí SELECT * může mít velký vliv na výkonnost, zvláště v případě tabulky s mnoha sloupci. Výpis požadovaných sloupců dělá následně dotaz srozumitelnější a eliminuje změny dotazu v případě změny struktury tabulky (např. přidání nových sloupců).

- Použití klauzule WHERE

  Je vhodné vždy použít klauzuli WHERE v SELECT dotazu z důvodu omezení počtu výsledných řádků. V případě jeho absence musí SQL Server provést náročnou operaci table scan a vrátit tak všechny záznamy dané tabulky. V tomto případě je pak tabulka během vykonávání dotazu uzamčena, dochází ke zvýšení síťového přenosu a snižuje se i možnost znovu použití dat z cache.

- Vyvarování se klauzule ORDER BY

  Pokud to není opravdu nutné, je vhodné eliminovat klauzuli ORDER BY, neboť při jejím použití dochází ke zpomalení dotazu. V mnoha případech je efektivnější setřízení dat na straně klienta (na aplikační úrovni).

- Nepoužívat v dávce proměnné u klauzule WHERE

  Pokud se provede dotaz, jenž je součástí dávky a jenž obsahuje klauzuli WHERE s proměnnou, pak optimalizátor dotazů provede místo index scanu nežádoucí table scan. Při analýze dotazu a volby přístupové metody totiž nelze rozpoznat hodnotu dané proměnné a tím pádem nelze získat potřebné informace pro použití indexů.

- Využití klauzule EXISTS

  Pokud subdotaz obsahuje klauzuli EXISTS, pak SQL Server ukončí získávání záznamů, když alespoň jeden vyhovuje podmínce, např.:

  ```
  IF EXISTS (SELECT * FROM Test.db1 WHERE AutoID = 1)
  PRINT 'Nelze smazat dany zaznam'
  ```

- Přepis subdotazů pomocí JOIN

  Operace JOIN jsou efektivnější než subdotazy z důvodu jejich paralelního zpracování namísto sekvenčního běhu subdotazů.

- Využití klauzule CASE

  V případě dotazů, jež v několika krocích přistupují ke stejné tabulce, popř. modifikují tabulku na základě podmínek, je vhodné použít příkaz CASE, jenž prochází tabulku jen jednou.

7.3 Konkrétní optimalizace dotazů

Dle analýzy jsem se postupně zaměřoval na nejvíce zatěžující dotazy a dávky, přičemž jsem dbal i na jejich četnost (optimalizace dotazu vykonaného s četností 1% zvýší výkon celého databázového systému právě a maximálně o dané 1%). Nyní zde uvedu několik příkladů, jak jsem postupoval při ladění konkrétních dotazů.

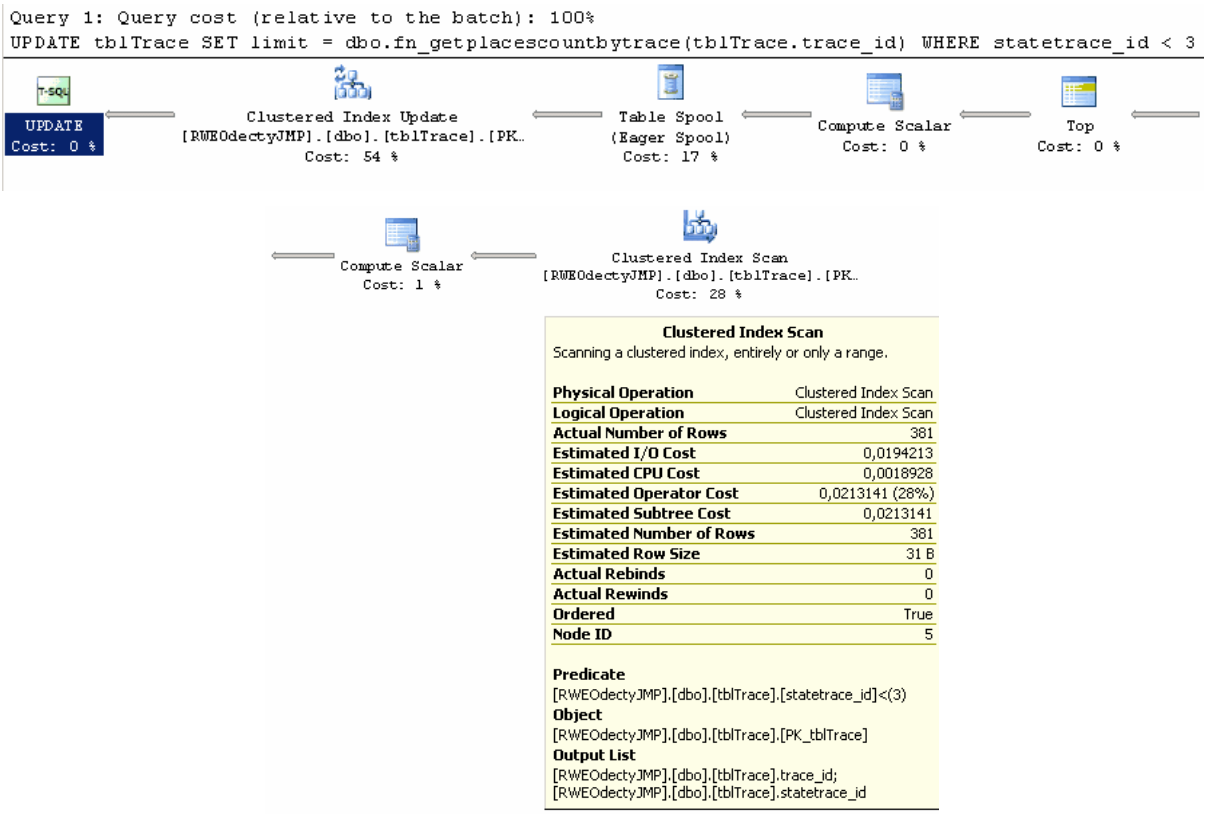
Procedura:	*ap_regenerateStatistics*
Prováděcí čas v ms:	22700
CPU v ms:	40700
Ovlivněno záznamů:	3500

Nejnáročnější část:

```
UPDATE tblTrace
SET limit = dbo.fn_getplacescountbytrace(tblTrace.trace_id)
WHERE statetrace_id < 3
```

Prováděcí čas v ms: 20500

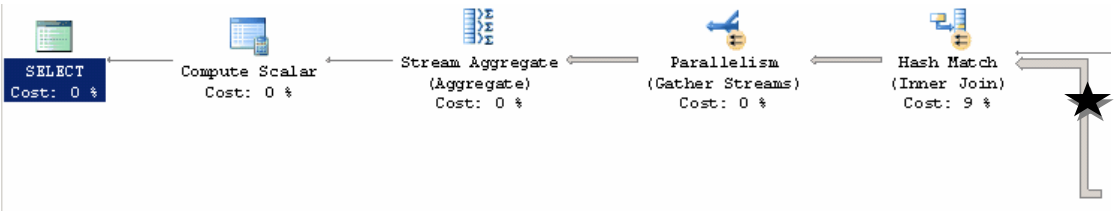
CPU v ms: 37125

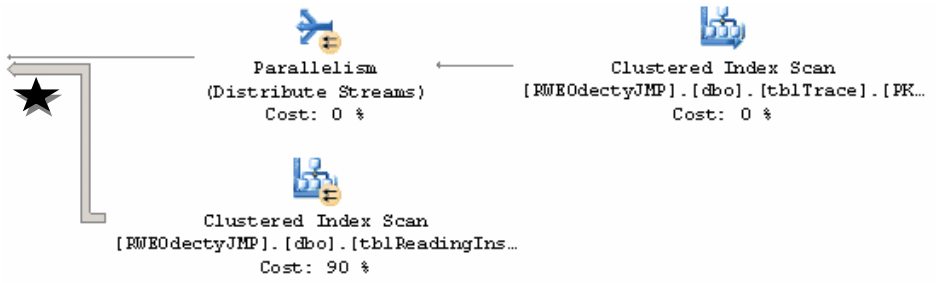


Obr. 7.2 Prováděcí plán procedury *ap_regenerateStatistics*

Na prováděcím plánu 7.2 lze vidět, že nejnáročnější částí je operace Clustered Index Update, jež je výsledkem volání funkce *fn_getplacescountbytrace:*.

```
SELECT COUNT(place_id)
FROM tblReadingInstructions INNER JOIN tblTrace
ON tblReadingInstructions.trace_id = tblTrace.trace_id
WHERE tblTrace.trace_id = <trace>
```





Obr.7.3 Prováděcí plán funkce *fn_getplacescountbytrace* (symbol ★ značí pokračování plánu)

Na tomto prováděcím plánu je již nejnáročnější částí operace Clustered Index Scan, která musí přečíst celý clustered index (v tomto případě privátní klíč tabulky *tblReadingInstructions*).

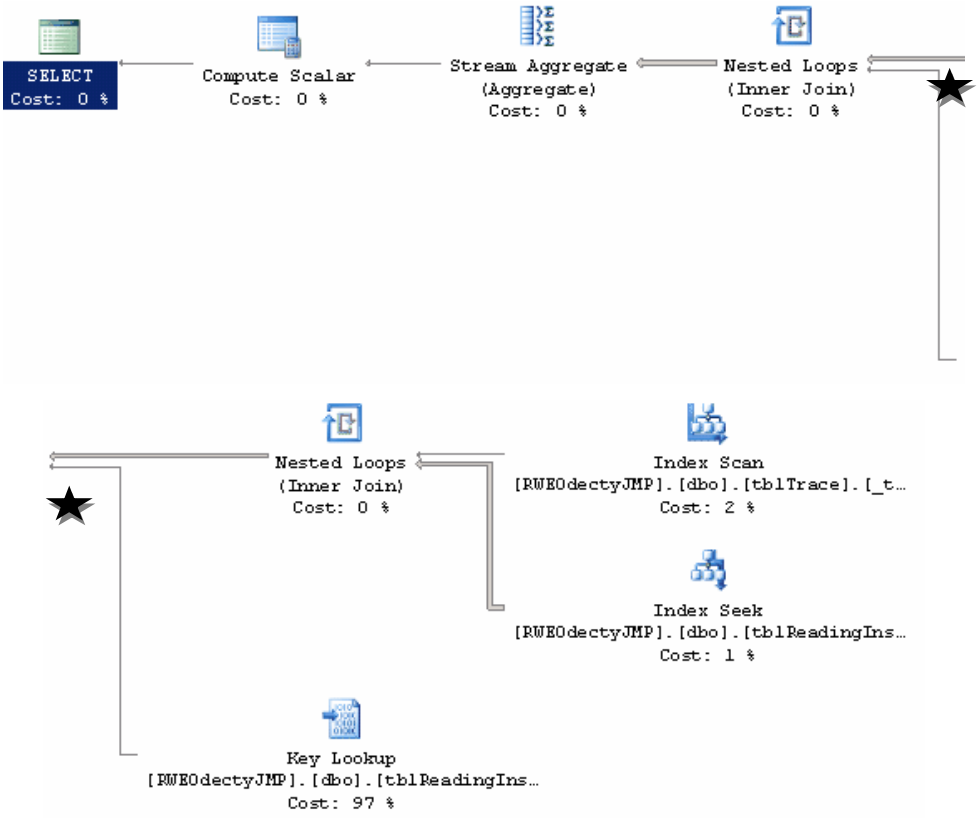
Řešení:

Vytvořit nonclustered indexy zahrnující sloupce *statetrace_id* a *traceid*

```
CREATE NONCLUSTERED INDEX IX_TblTrace
ON dbo.tblTrace (statetrace_id)
INCLUDE (trace_id)
```

Výsledek:

Prováděcí čas v ms:	3546
CPU v ms:	2687
Zlepšení:	cca +80%



Obr.7.4 Optimalizovaný prováděcí plán funkce *fn_getplacescountbytrace*

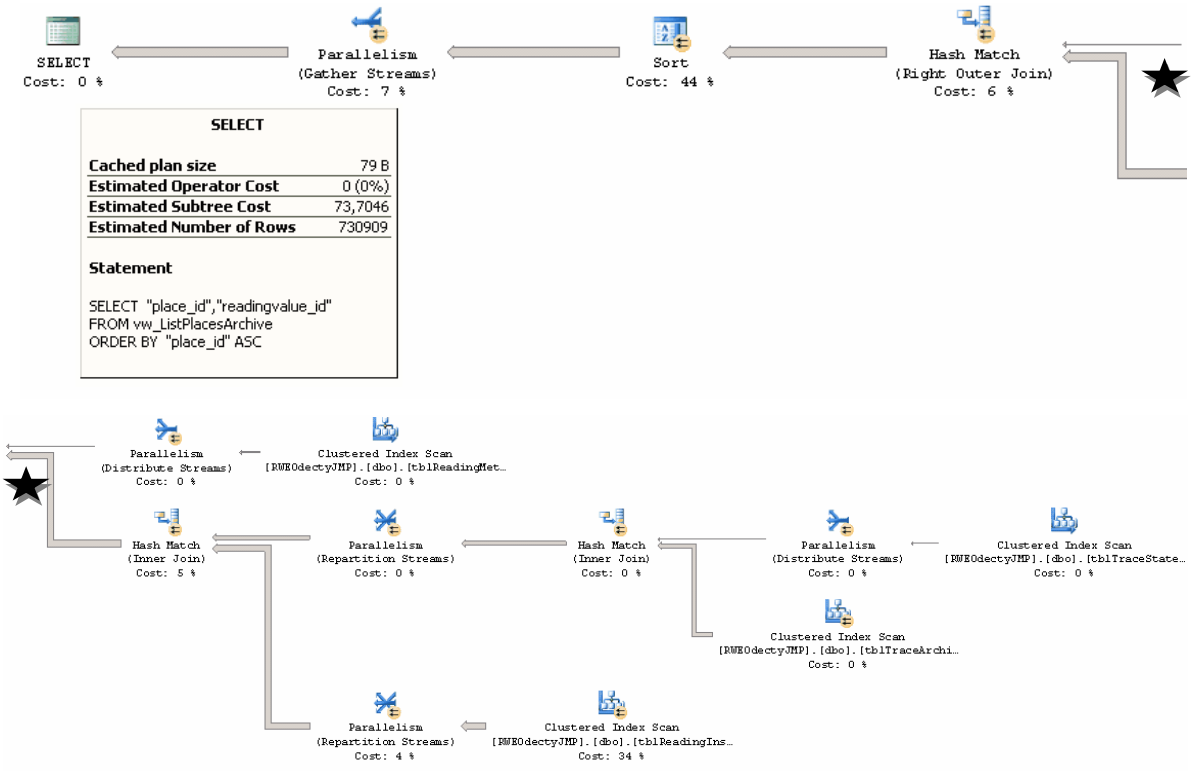
Jak je vidět na obr. 7.4, zde se již použije Index Seek a nejnáročnější operace je nyní Key lookup, jež má ovšem minimální náklady (0,48 místo 5,8). Výkonnost lze ovšem ještě zvýšit použitím tzv. covered indexu (použít klauzuli INCLUDE). Změnila se i metoda výpočtu operace JOIN z Hasch Match na Nested Loops.

Ne vždy ovšem bylo možné daný vytěžující dotaz či proceduru zoptimalizovat, viz další příklad:

TSQL dotaz:

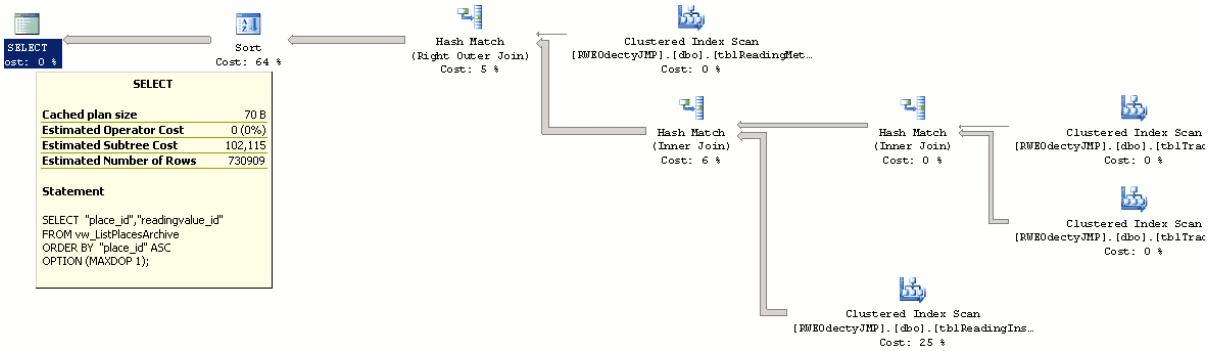
```
SELECT  "place_id","readingvalue_id",a dal.
FROM vw_ListPlacesArchive
ORDER BY  "place_id" ASC
```

Prováděcí čas v ms:	72673
CPU v ms:	50625
Ovlivněno záznamů:	750 000



Obr. 7.5 Prováděcí plán pohledu *vw_ListPlacesArchive*

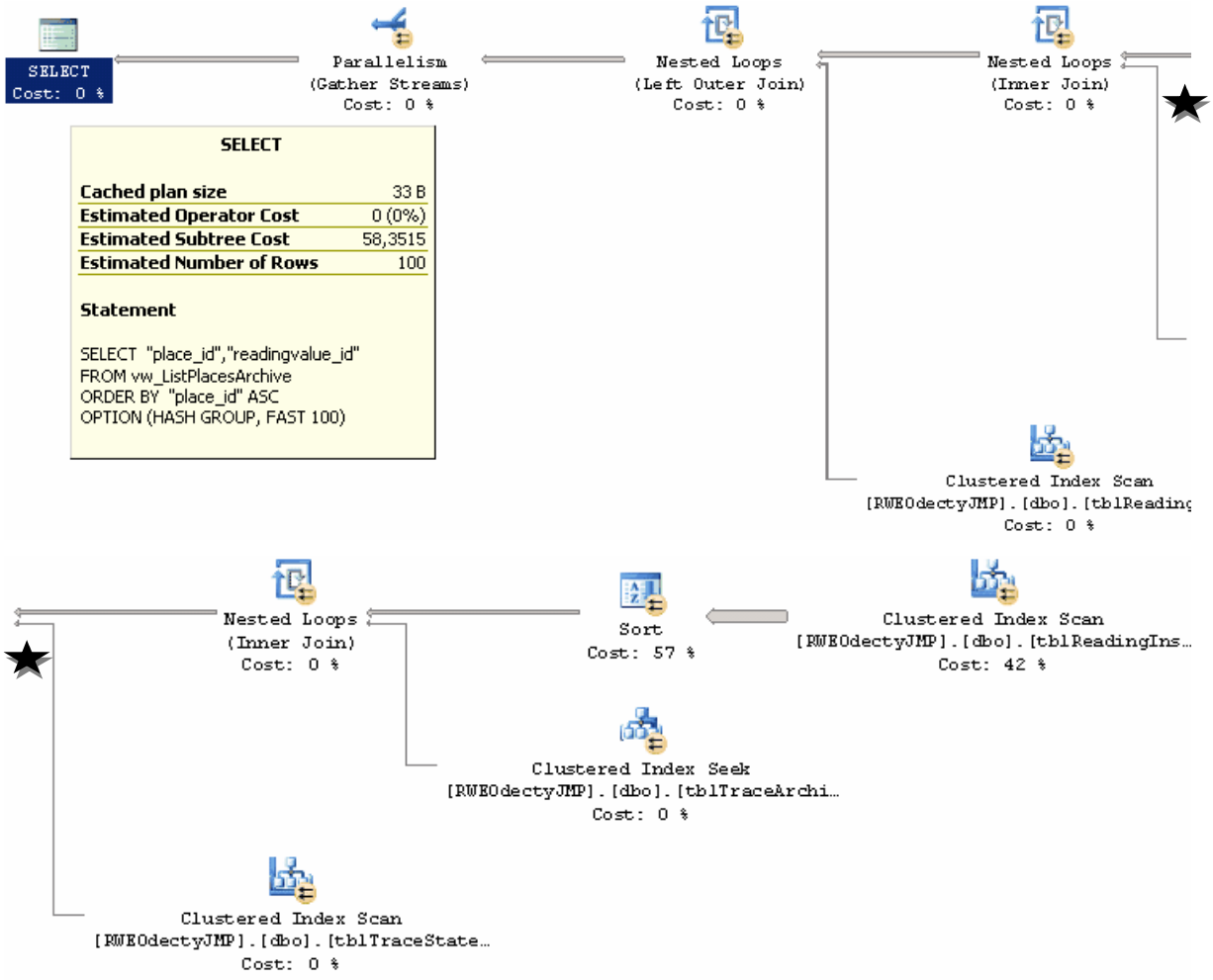
Pohled *vw_ListPlacesArchive* obsahuje SELECT nad několika tabulkami pomoci LEFT OUTER JOIN. Jak lze vyčíst z plánu, velice náročnou operací je SORT. Optimalizátor správně využil paralelismu, viz operace se znakem . V případě, že by server nedisponoval vícero procesory, pak by výsledný plán vypadal následovně:



Obr. 7.6 Prováděcí plán pohledu *vw_ListPlacesArchive* při sekvenčním běhu

Na obr. 7.6 lze vidět výslednou cenu dotazu provedeného sekvenčně (pomocí nastaveného parametru *MAXDOP* na hodnotu 1). Rozdíl odhadované ceny činí téměř 30%.

Urychlení provedení tohoto dotazu lze realizovat přidáním query hints *FAST*, viz následující obrázek:



Obr.7.7 Prováděcí plán pohledu *vw_ListPlacesArchive* při použití parametru *FAST* 100

Na tomto plánu lze vidět, že odhadovaná cena je o zhruba 20 jednotek nižší, je ovšem daná menším počtem řádků započítaných do výsledku dotazu.

7.4 Správa indexů

S optimalizací dotazů úzce souvisí i správa indexů. Prvním sledovaným příznakem v analýze byla míra jejich využití. Málo využívané indexy totiž zatěžují SQL Server z pohledu správy statistik pro optimalizátor a je tedy vhodné je odstranit:

```
DROP INDEX [NazevIndexu] ON [NazevObjektu] WITH ( ONLINE = ON )
```

Pokud se smaže nonclustered index, pak je jeho definice odstraněna z metadat a datové stránky příslušného indexu (B-strom) jsou smazány z databázových souborů. V případě zrušení clustered indexu jsou datové záznamy, jež byly uloženy na úrovni listů daného indexu, přesunuty do struktury hromada (heap).

Parametr *ONLINE = ON* určuje, zda tabulka asociovaná s příslušným indexem bude dostupná pro dotazy a případné modifikace během této operace. Tato vlastnost je možná pouze v SQL Server 2005 Enterprise edici a pouze v případě odstranění clustered indexu. Lze použít i při opětovném sestavení indexu.

Dalším krokem analýzy byla detekce fragmentace nejvíce využívaných indexů. V případě velké míry je doporučeno znovu sestavení indexu (= REBUILD, smazání a nové vytvoření indexu), v případě menší míry postačí reorganizace (= defragmentace listů indexu). Znovu sestavení indexu jsem provedl následovně:

```
ALTER INDEX [NazevIndexu] ON [NazevObjektu]
REBUILD WITH (
  PAD_INDEX  = OFF,
  FILFACTOR  = 0,
  STATISTICS_NORECOMPUTE  = OFF,
  ALLOW_ROW_LOCKS  = ON,
  ALLOW_PAGE_LOCKS  = ON,
  SORT_IN_TEMPDB = OFF,
  ONLINE = ON)
```

Použil jsem tyto parametry (lze je použít také při vytváření nového indexu):

- *PAD_INDEX* – určuje tzv. výplň indexu střední úrovně datových stránek. Úzce souvisí s následujícím parametrem *FILLFACTOR*, pokud není nastaven, pak je zaplnění rovno téměř plné kapacitě.
- *FILLFACTOR* – udává v % míru zaplnění listové úrovně indexu. Implicitní hodnota je 0, jež znamená optimalizované zaplnění indexu. Jiná hodnota znamená skutečné

procento zaplnění (např. hodnota 100 určuje vytvoření clustered indexů na plných datových stránkách, v případě nonclustered indexů na plných datových listech).

- *STATISTIC_NORECOMPUTE* – určuje, zda budou přepočítány distribuční statistiky.
- *ALLOW_ROW_LOCKS, ALLOW_PAGES_LOCKS* – povolení zámků nad řádky, resp. stránkami při přístupu k indexu.
- *SORT_IN_TEMPDB* – určuje, zda se uloží výsledky setřízení do databáze tempdb. Pokud je tato databáze na jiném disku než uživatelské, pak může dojít ke snížení času při vytváření či znovu sestavení indexu.

Opětovné vytvoření indexu odstraní fragmentaci, upraví diskové místo zhuštěním indexových stránek dle nastaveného FILLFACTORU a přeskupí indexové záznamy ve stránkách.

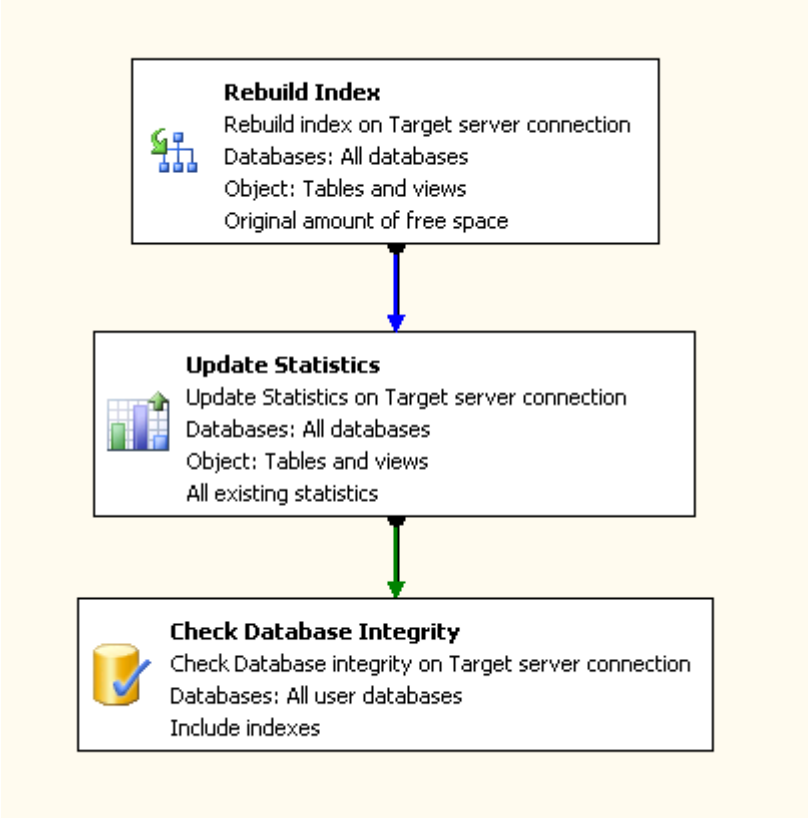
Reorganizaci indexu jsem provedl pomocí příkazu:

```
ALTER INDEX [NazevIndexu] ON [NazevObjektu]
REORGANIZE WITH (
  LOB_COMPACTION = ON )
```

Tuto operaci lze upřesnit jediným parametrem *LOB_COMPACTION*, jenž specifikuje, zda velikost všech datových stránek zahrnující velké objekty (datové typy LOB jsou image, text, ntext, (n)varchar, varbinary a xml) bude zredukována. Povolením tohoto parametru lze dosáhnout lepšího využití diskového místa. Proces reorganizace vyžaduje minimum systémových prostředků, defragmentuje listovou úroveň daného indexu fyzickým uspořádáním stránek indexu dle logického pořadí.

SQL Server 2005 umožňuje vytvořit a používat tzv. plány údržby, jež automatizovaným způsobem provádí základní úkoly pro správu databáze, včetně indexů. Na obr. 7.7 je uvedeno schéma základního plánu na serveru SQL1, kde jsem použil následující kroky:

- Rebuild Index – přestavění indexu pro zlepšení výkonnosti při prohledávání indexů. Dojde k optimalizaci distribuce dat a volného prostoru v indexových stránkách pro jejich budoucí rychlejší růst
- Update Statistics – zaktualizuje statistky optimalizátoru dotazů týkající se distribuce hodnot dat v tabulkách. Zlepší se tím schopnost optimalizátoru najít vhodnou strategii přístupu k datům za účelem zvýšení výkonu dotazů
- Check Database Integrity – provede interní kontrolu konzistence datových a indexových stránek v dané databázi.



Obr. 7.8 Schéma plánu údržby

Tento plán údržby má nastaven každodenní spouštění vždy o půl noci, kdy je databázový systém nejméně využíván. Je výsledkem analýzy, díky které jsem zjistil, že chybí mnoho potřebných statistik pro optimalizátor a že všechny clustered indexy jsou během dne hojně využívány, což má za následek vyšší fragmentaci.

7.5 Partitioning

Vzhledem k tomu, že všechny analyzované servery zpracovávají pouze aktuální data, nebyla potřeba implementovat rozdělení tabulek – partitioning. V daném enterprise prostředí se ovšem připravuje nasazení nového SQL Serveru, jenž bude využit zejména pro analýzu a dolování dat. V tomto případě, kdy aplikace nad danou databází bude pracovat s velkým množstvím dat, může rozdělení tabulek v databázi zrychlit přístup a manipulaci s nimi. Proto jsem si tento způsob ladění schématu databáze vyzkoušel v testovacím prostředí.

Microsoft SQL Server podporuje horizontální dělení, tabulky tak lze rozdělit na několik částí (partition) dle různých kritérií, nejčastěji podle časového hlediska. Jednotlivé části pak představují daný časový úsek (např. kalendářní měsíc). V testovacím prostředí jsem zvolil rozdělení tabulky, jež byla naplněna několika tisíci záznamy obsahující náhodné hodnoty (pomocí funkce *rand()*) typu integer.

Proces rozdělení tabulky zahrnuje čtyři kroky:

1. Vytvoření jedné nebo více filegroup

   Slouží pro seskupení jednoho nebo více databázových souborů umožňující je umístit na jiné disky. Pro testovací účely jsem vytvořil tři filegroupy SG1 až SG3.

2. Vytvoření partition funkce

   Slouží pro definici rozdělení hodnot do částí. Protože jsem použil testovací tabulku s hodnotami v rozmezí 0 – 1000, zvolil jsem pravidelné rozložení do čtyř částí, viz TSQL příkaz:

```
CREATE PARTITION FUNCTION PF1 (int)
AS
RANGE RIGHT FOR VALUES (250,500,750)
```

3. Vytvoření partition schématu

   Namapování rozdělené tabulky na příslušné fyzické soubory, resp. filegroupy. Lze zvolit namapování všech částí do jedné skupiny, popř. je specifikovat jako v mém případě:

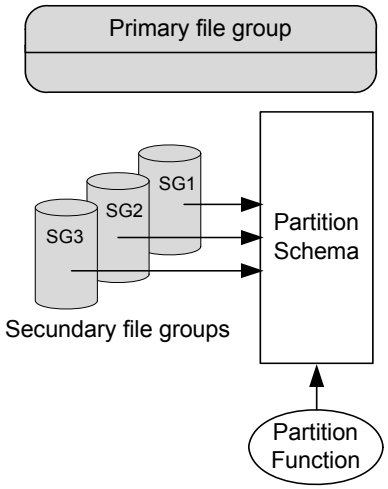
```
CREATE PARTITION SCHEME PS1
AS
PARTITION PF1 TO ([SG1],[SG2],[SG3],[PRIMARY])
```

4. Vytvoření tabulky se specifikací schématu

   Syntaxe je stejná jako při vytváření klasické tabulky jen s tím rozdílem, že na konci příkazu je uvedena specifikace schématu a příslušného sloupce:

```
CREATE TABLE tab_part (id int, test int)
ON PS1 (test)
```

Celý tento proces lze znázornit pomocí obr. 7.9:



Obr. 7.9 Schéma partitioning

Dotaz nad rozdělenou tabulkou se pak provádí klasicky beze změny jako v případě nerozdělené tabulky, výběr hodnot z příslušných částí provádí automaticky SQL Server. Lze ovšem specifikovat hodnoty pouze z námi zvolené části pomocí příkazu *$partition.<nazev schematu>*. Následující příkaz vypíše součet všech záznamů v jednotlivých částech tabulky:

```
SELECT
  $partition.PF1(test) AS 'Partition',
  count(*) AS 'Pocet zaznamu'
FROM tab_part
GROUP BY $partition.PF1(test)
ORDER BY $partition.PF1(test)
```

Výsledkem je rovnovážné rozložení hodnot (a tedy potvrzení kvality náhodného generátoru čísel), viz obr. 7.10.

	Partition	Pocet zaznamu
1	1	69158
2	2	68643
3	3	68561
4	4	68761

Obr. 7.10 Počet záznamů v jednotlivých částech rozdělené tabulky

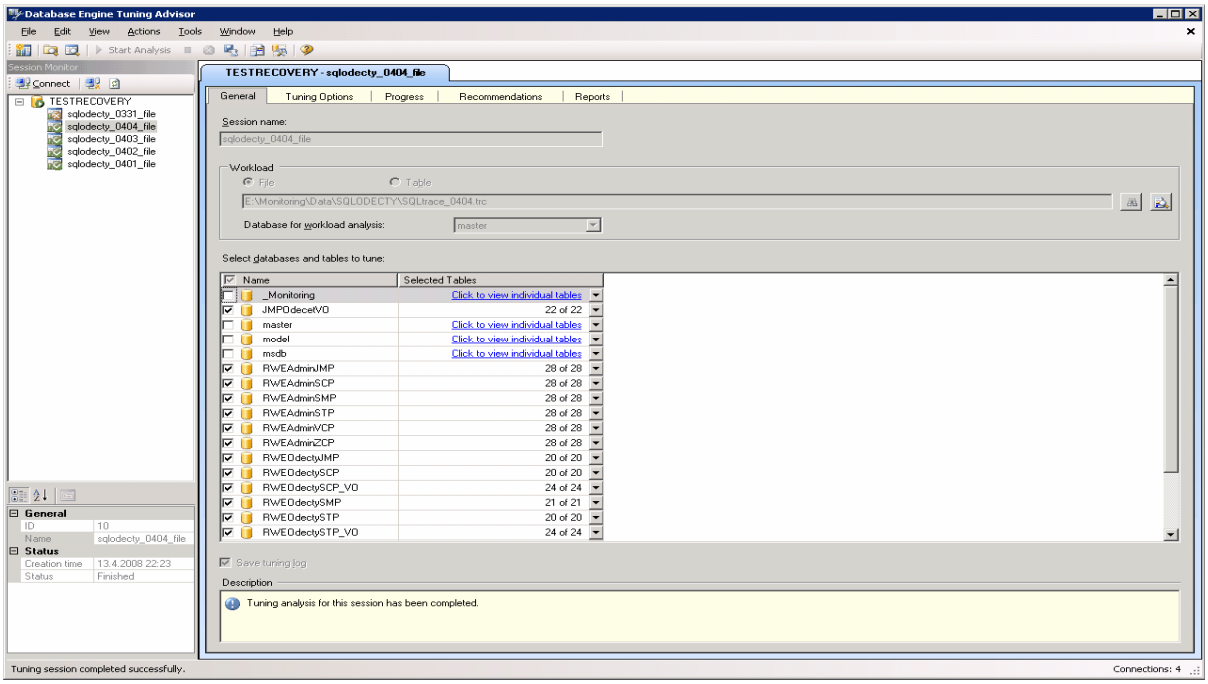
Rozdělení tabulek lze použít vždy jen na základě hodnoty jednoho sloupce a kromě číselných typů je možné použít i typ datum a znak. Rozdělit lze také indexy příslušné tabulky, kdy v případě použití stejné partition funkce dojde k navýšení výkonu, neboť tabulka i index jsou vzájemně uspořádány (všechna data a indexy jsou rozděleny stejným algoritmem).

Otestoval jsem rychlost přístupů k rozdělené a klasické tabulce, ale rozdíly nebyly tak markantní. To bylo ovšem dáno zejména typem dat, počtem procesorů a také skutečností, že ne všechny filegroupy byly umístěny na různých discích. V případě dotazů nad mohutnou tabulkou (o velikosti řádově 100 GB), která je rozdělena do několika filegroup umístěných na vlastních discích a v případě systému, disponujícím vícero procesory (minimálně 4), lze dosáhnout navýšení výkonu.

7.6 Použití nástroje Database Tuning Advisor

Nástroj Database Tuning Advisor (DTA) je vhodný pro analýzu a optimalizaci dotazů. Jako vstup lze použít zátěž (stopu) zaznamenanou SQL Profilerem, tabulku nebo pouze daný pomalu vykonávající dotaz. DTA umožňuje několik základních nastavení rozdělené do sekcí, zde uvádím příklad mého nastavení:

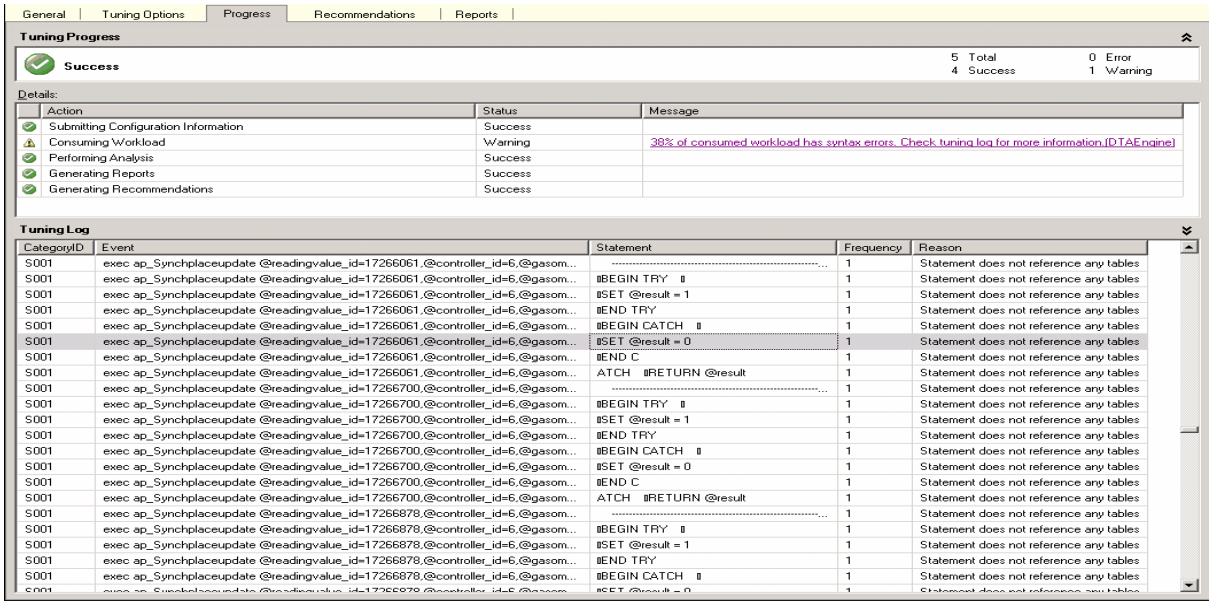
- Fyzické struktury použité v databázi: indexy (oba typy, clustered i nonclustered) i indexované pohledy – DTA vezme v potaz všechny indexovatelné struktury SQL serveru
- Strategie rozdělení - partitioning: zvolil jsem možnost navrhnout partitioning
- Fyzické struktury zachovatelné v naší databázi: neponechat žádné struktury, tzn. navrhnout případně nové optimální možnosti pro indexy.



Obr. 7.11 Úvodní obrazovka nástroje DTA

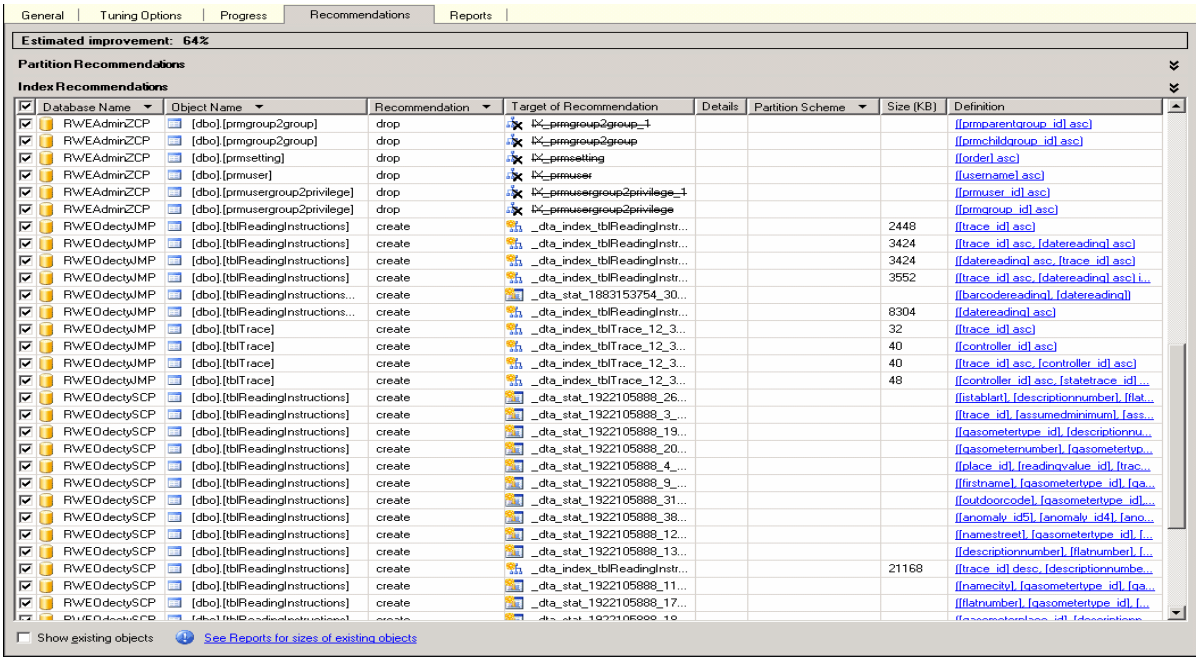
Na obr. 7.11 lze vidět úvodní obrazovku DTA, kde jsem zvolil stopu SQL Profileru a databáze, jež budou zahrnuty do ladění. DTA po načtení stopy provede její analýzu, viz obr. 7.9, na kterém lze vidět tzv. Tuning log, kde jsou zaznamenané události, jež nebudou zahrnuty do ladění, např.:

- Dotaz nezahrnuje žádnou tabulku – příkazy SET nebo DECLARE
- Dotaz je proveden nad velmi malou tabulkou, jež obsahuje 10 datových stránek či méně.
- Nesprávná syntaxe – příkazy zahrnující klíčová slova BEGIN, TRY, INSERTED.



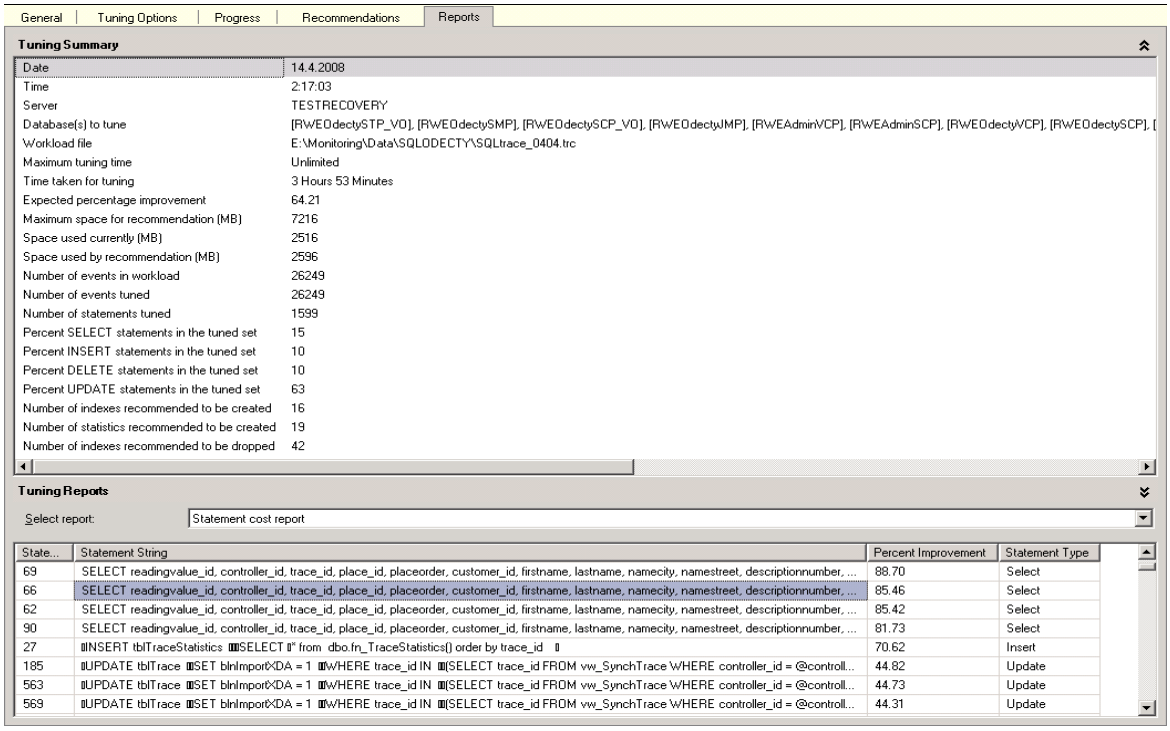
Obr. 7.12 Aktuální činnost DTA

Následně probíhala nejnáročnější část – vytvoření optimalizačních doporučení a výsledného reportu, v případě největší stopy trvala celá tato akce téměř 6 hodin.



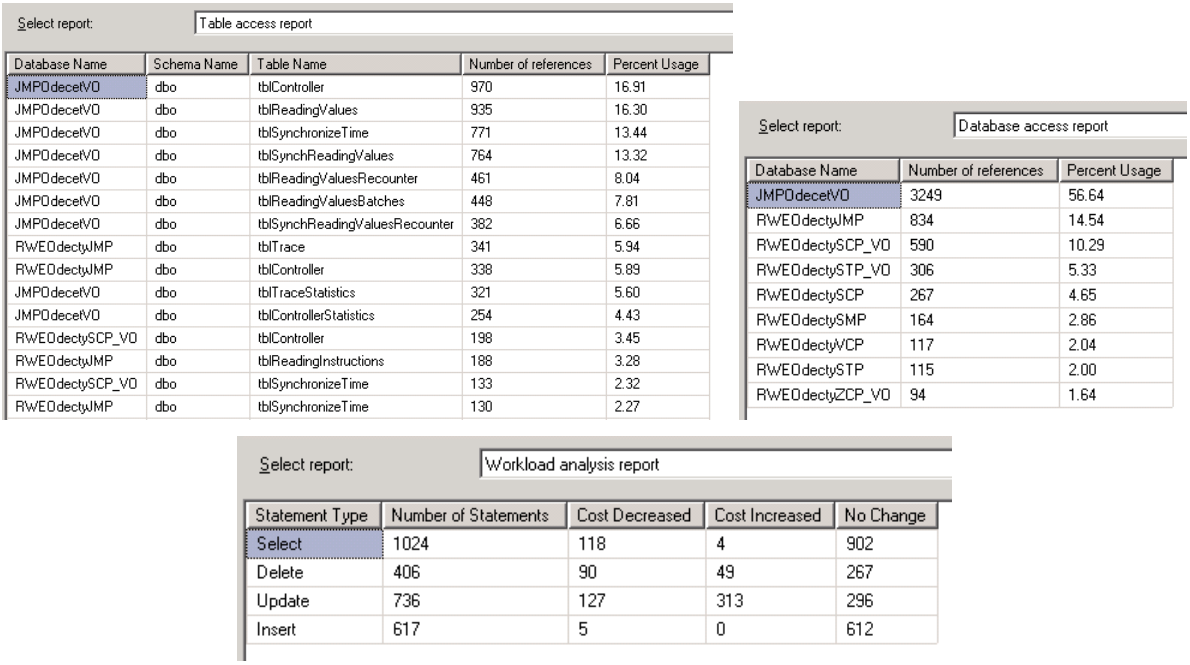
Obr. 7.13 DTA - optimalizační doporučení

Na obr. 7.13 lze vidět výsledná doporučení s odhadovaným zlepšením výkonnosti, v našem případě 64%. Tato doporučení zahrnují operace, jako jsou odstranění indexů, vytvoření nových, popř. vytvoření chybějících statistik optimalizátoru.



Obr. 7.14 DTA - výsledný report

Výsledný report obsahuje několik různých statistik zaznamenané zátěže, např. procento využitelnosti indexů a databáze, počet přístupů k daným tabulkám a sloupcům, počet příkazů, viz obr. 7.15:



Select report: Table access report

Database Name	Schema Name	Table Name	Number of references	Percent Usage
JMPOdecetVO	dbo	tblController	970	16.91
JMPOdecetVO	dbo	tblReadingValues	935	16.30
JMPOdecetVO	dbo	tblSynchronizeTime	771	13.44
JMPOdecetVO	dbo	tblSynchReadingValues	764	13.32
JMPOdecetVO	dbo	tblReadingValuesRecounter	461	8.04
JMPOdecetVO	dbo	tblReadingValuesBatches	448	7.81
JMPOdecetVO	dbo	tblSynchReadingValuesRecounter	382	6.66
RWEOdectyJMP	dbo	tblTrace	341	5.94
RWEOdectyJMP	dbo	tblController	338	5.89
JMPOdecetVO	dbo	tblTraceStatistics	321	5.60
JMPOdecetVO	dbo	tblControllerStatistics	254	4.43
RWEOdectySCP_VO	dbo	tblController	198	3.45
RWEOdectyJMP	dbo	tblReadingInstructions	188	3.28
RWEOdectySCP_VO	dbo	tblSynchronizeTime	133	2.32
RWEOdectyJMP	dbo	tblSynchronizeTime	130	2.27

Select report: Database access report

Database Name	Number of references	Percent Usage
JMPOdecetVO	3249	56.64
RWEOdectyJMP	834	14.54
RWEOdectySCP_VO	590	10.29
RWEOdectySTP_VO	306	5.33
RWEOdectySCP	267	4.65
RWEOdectySMP	164	2.86
RWEOdectyVCP	117	2.04
RWEOdectySTP	115	2.00
RWEOdectyZCP_VO	94	1.64

Select report: Workload analysis report

Statement Type	Number of Statements	Cost Decreased	Cost Increased	No Change
Select	1024	118	4	902
Delete	406	90	49	267
Update	736	127	313	296
Insert	617	5	0	612

Obr. 7.15 DTA – ukázky výsledných analýz

Vyzkoušel jsem i příkazový řádek nástroje DTA, dosažené výsledky ovšem odpovídaly reportům uvedeným výše.

```
dta -E -D DB1 -if Report.sql -s SQL1 -of SQL1_0401.sql -ox
SQL1_0401.xml -fa IDX_IV -fp NONE -fk NONE
```

7.6.1 Výsledný report

DTA v prvním případě doporučil odstranění nepoužívaných indexů. Poté navrhl vytvoření několika nonclustered indexů a vytvoření chybějících statistik pro optimalizátor.

Výsledná doporučení lze přímo aplikovat na danou databázi, uložit ve formě TSQL skriptu, popř. otestovat volbou „*Evaluate Recommendations*“, kdy se spustí nová analýza zátěže ale s již aplikovanými doporučeními.

Porovnal jsem všechna vytvořená doporučení a vytvořil výstup, jenž obsahoval příkazy shodné ve všech skriptech. Výsledkem byl skript, který jsem aplikoval na testovací databáze a který zahrnoval doporučení, jež byla výstupem všech zaznamenaných zátěží.

8 Závěr

Po nastudování teorie optimalizace databázových systémů, kdy jsem se zaměřil především na fyzickou organizaci dat (indexy a jejich druhy, přístupové metody), princip optimalizátoru a optimalizační nástroje konkrétních databázových systémů, jsem měl dostatek informací na konkrétní analýzu a ladění databázových systému Microsoft SQL Server 2005 v enterprise prostředí.

Vzhledem ke skutečnosti, že všechny analyzované systémy jsou v produkčním prostředí 24/7 a je na nich požadováno nejpřísnější SLA (smlouva se zákazníkem), byla prvním krokem domluva spuštění a nasazení monitoringu. Proběhlo tak několik telekonferencí, jejichž výsledkem bylo vytvoření tzv. monitorovací strategie. Ta určovala, kdy přesně bude sledování systému započato a ukončeno, s jakou četností budou sbírány data čítačů systémového nástroje Perfmon, popř. co vše bude zaznamenáno na SQL Serveru.

Po týdenním sbírání dat přišla na řadu jejich analýza. Jednalo se o velice náročný proces na čas, neboť tato data dosahovala velkých objemů a jejich následné načtení do pomocných nástrojů a analýza tudíž trvala nepřípustnou dobu. Výstupem v tomto kroku byly různé grafy a tabulky, zobrazující analýzy systémové úrovně (čas procesoru, využití paměti, vytížení V/V prostředků aj.) i analýzy týkající se přímo SQL Serveru (detekce nejpomalejších dotazů a dávek, využití a fragmentace indexů aj.). Kombinací analýz obou daných úrovní jsem se snažil zjistit, zda je eventuální slabé místo detekované na systémové úrovni způsobeno aspektem úrovně databázové a naopak, kolik systémových prostředků spotřebovávají nejpomalejší a nejnáročnější dotazy. Ověřil jsem také zabezpečení serverů a práva uživatelů, kteří se k daným serverům připojují. Zde nebyly zjištěny žádné kritické nedostatky, uživatelé mají nastavena adekvátní práva a přístupy a administrátorské (neomezené) oprávnění má přiděleno minimum ověřených uživatelů.

Následovalo vytvoření testovacího prostředí, na které jsem nahrál všechny analyzované databáze. Výstupy v předchozím kroku byly vstupem pro samotný proces optimalizace, kdy jsem testoval různé varianty ladění dotazů (vytvoření chybějících indexů, přepis dotazů do optimálnější podoby), zkoušel možnosti nastavení SQL serveru (paralelní zpracování, rozšíření paměti apod.) a testoval zaznamenanou zátěž nástrojem Database Tuning Advisor. Vznikly tak reporty a doporučení, jež byly následně poslány aplikačním správcům daných databázových systémů.

Z důvodu zdlouhavých business procesů v daném enterprise prostředí, kdy veškeré změny na produkčních systémech podléhají schvalování na úrovni managementu, nedošlo v době odevzdání této práce k implementaci navrhnutých optimalizací, a tudíž jsem musel vzít jako výsledek optimalizace testy provedené na simulovaném prostředí. Zde ovšem nešlo přesně nasimulovat stejnou zátěž jako v případě zaznamenané v analýze (zejména nemožná simulace počtu připojených uživatelů a jejich aktivit - dotazů). Proto bych rád uvedl novou vlastnost databázového systému Oracle 11g, jež je pro tento účel ideálně stvořená. Jedná se o tzv. Real Application Testing, který umožňuje provést

zachycení zátěže, následně ji přehrát na testovacím systému s různými optimalizačními změnami a tím jednoduše provést simulaci reálného prostředí.

8.1 Návrhy na zlepšení

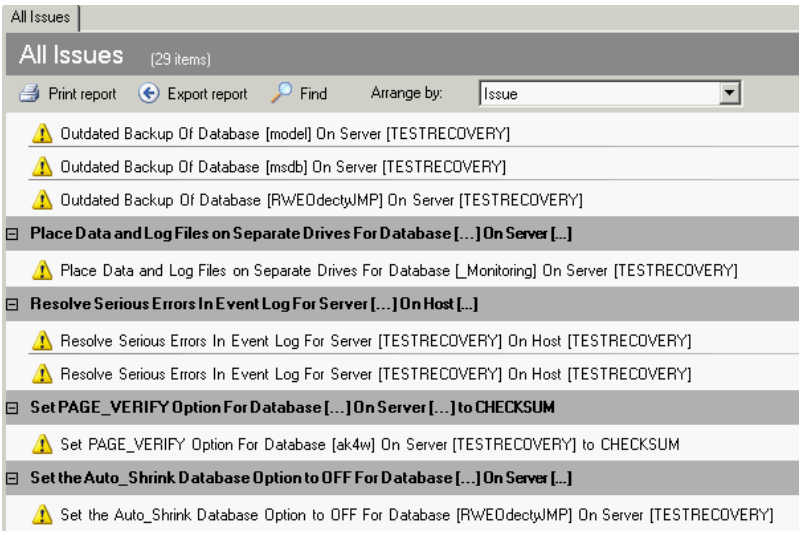
Existuje mnoho možností, jak zefektivnit a zautomatizovat jednotlivé procesy optimalizace. V případě monitorovací fáze lze využít několik nástrojů firmy Microsoft, např. SQLIO (Disk Subsystem Benchmark Tool), jenž je určen pro simulaci V/V zátěže pomocí operací SQL Serveru (čtení a zápis různě velkých datových stránek, záloha a obnova databází aj.). Podobným nástrojem je utilita SQLIOSim, která simuluje aktivitu SQL Serveru na diskovém subsystému.

Pro diagnostiku výkonnostních problému, týkající se přímo SQL Serveru, existuje také několik nástrojů:

- SQLDiag: nástroj ke kolekci logů, zátěže a čítačů, jenž může běžet jako služba či konzole.
- Best Practices Analyzer (BPA): umožňuje sbírat data z OS Windows a konfiguraci SQL Serveru a díky přednastaveným doporučením dokáže zjistit, zda se v daném databázovém prostředí nachází slabé místo.
- Performance Dashboard Reports: slouží k monitoringu a k řešení výkonnostních problémů.
- Health and History Tool (SQLH2): podobně jako BPA provádí kolekci dat a následné spouštění reportů vůči nim pro zjištění, jak je daný SQL Server využíván.
- RML Utilities: umožňuje zjistit, které aplikace, databáze, případně uživatel (login) využívá nejvíce systémových prostředků či jak se bude daný systém chovat v případě aplikace různých hotfixů a jiných konfiguračních změn.

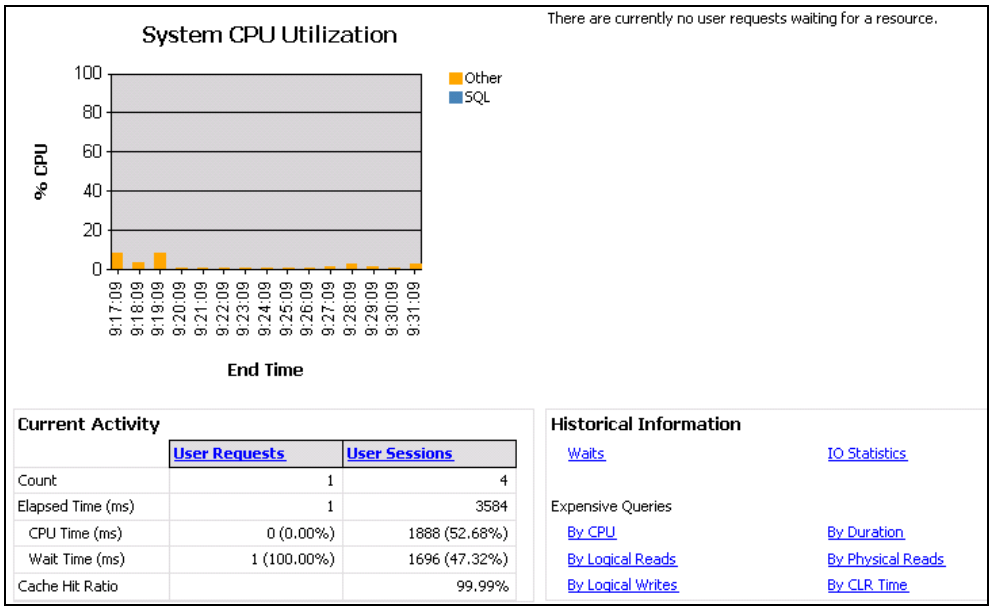
Tyto nástroje ovšem vyžadují instalaci na daný server a vzhledem k již uvedeným záležitostem ohledně nemožnosti změny na serveru bez jeho předchozího schválení, jsem tyto utility nemohl vyzkoušet na produkčních systémech. Proto jsem je nasadil pouze na testovacím prostředí, ověřil jejich funkčnost a míru prospěšnosti pro daný účel.

Vyzdvihl bych zde především BPA, jenž automatizovaným způsobem otestuje prostředí SQL Serveru a zobrazí přehledný report s vysvětleními, jak vyřešit eventuální problémy. Dokáže ověřit základní konfiguraci serveru, jeho zabezpečení a případné výkonnostní nedostatky. Na obr. 8.1 lze vidět příklad výstupu nástroje BPA:



Obr. 8.1 Výsledný report nástroje BPA

Přehledný výstup vhodný pro následnou analýzu dokáže zobrazit i nástroj Performance Dashboard, jenž zobrazuje reporty dle předdefinovaných šablon pomocí integrace do Management studia. Umožňuje tak zobrazit nejvíce vytěžující dotazy s jejich prováděcím plánem, množstvím spotřebovaných systémových prostředků, případně chybějícími indexy a další vhodné statistiky.



Obr. 8.2 Úvodní obrazovka nástroje Performance Dashboard

Tyto nástroje umožňují provádět analýzu na vícero serverech najednou a jsou tudíž vhodným rozšířením v enterprise prostředí.

8.2 Zhodnocení

Proces optimalizace se skládá z několika úzce propojených kroků, z nichž zejména analýza a sledování serverů jsou časově velice náročné. Monitorování daného prostředí nelze totiž úspěšně realizovat během kratšího časového úseku, ale je důležité jej provádět pravidelně a po delší dobu. Jen tak lze získat ideální vzorek dat pro určení eventuálních slabých míst, jež jsou vstupem pro následnou konkrétní optimalizaci. Vzhledem k této skutečnosti byl vhodným nástrojem Microsoft Operations Manager (MOM), jenž mimo jiné ukládá všechna zaznamenaná výkonnostní data do databáze (Reporting) a lze tak získat přehledy za delší časový horizont (maximálně ovšem měsíc zpětně).

Pomocí týdenního sledování systémů a pomocí reportů nástroje MOM jsem zjistil, že analyzované servery v enterprise prostředí disponují adekvátními systémovými prostředky k dané zátěži, jíž jsou vystaveny. Našly se ovšem místa, která nebyla optimálně navržena a která jsem se snažil vyladit, popř. eliminovat.

Díky této práci jsem získal mnoho zkušeností, počínaje komunikace se zákazníky a domluvy technologických procesů, přes návrh monitorovací strategie až po realizaci a testování samotných optimalizačních kroků. V posledním případě jsem vyzkoušel a ověřil téměř všechny možnosti, které proces optimalizace, popř. samotný SQL Server umožňuje.

Literatura

[1] Kifer, M., Bernstein, A., Lewis, P., M.: Database Systems. An Application-Oriented Approach. 2nd edition. Part three. Pearson. 2006, s. 319 – 455

[2] Zendulka, J., Rudolfová, I.: Databázové systémy. Studijní opora, 2006, Fakulta informačních technologií VUT v Brně.

[3] Stanek, R., W.: Microsoft SQL Server 2005. Kapesní rádce administrátora. Computer Press. 2007

[4] Microsoft Official Course: 2780b Maintaining a Microsoft SQL Server 2005 Database. Microsoft Corporation, 2007

[5] Loney, K., Bryla, B.: Mistrovství v Oracle Database 10g. Computer Press. 2006

[6] Oracle Database Performance Tuning Guide, 10.12.2007, http://download.oracle.com/docs/cd/B14117_01/server.101/b10752/toc.htm

[7] Roy, S., Sugiyama, M.: Sybase Performance Tuning. Prentice Hall. 1996

[8] Hitchcock, B.: Sybase Database Administrator's Handbook. Prentice Hall. 1995

[9] Anderson, G., W.: Client/Server Database Design with SYBASE: A High-Performance and Fine- Tuning Guide. McGraw-Hill.1996

[10] Taylor, B.: The Official New Features Guide to Sybase ASE 15. Wordware Publishing, Inc. 2006

[11] Knight, B: Professional SQL Server 2005 Administration. Wiley Publishing, Inc. 2007, kapitola 11 – 15

[12] Microsoft IT Academy Online Learning Program
– course 2937: Administering and Monitoring Microsoft SQL Server 2005.
– workshop 2980-2985: Building a Monitoring Solution for Microsoft SQL Server 2005 Performance Issues and Optimizing.
31.4.2008, https://itacademy.microsoftelearning.com

[13] SQL Server Performance Tuning Articles, 31.4.2008, http://www.sql-server-performance.com/articles/per/main.aspx

[14] Sunil, A: Troubleshooting Performance Problems in SQL Server 2005. Microsoft Corporation. 2005

[15] SQL Server 2005 Books Online, 31.4.2008, http://technet.microsoft.com/en-us/library/ms130214.aspx

Příloha 1: Skripty pro analýzu SQL Serveru

Analýza využití procesoru a detekce náročných dotazů

```
-- Vypis vsech CPU a uloh
SELECT scheduler_id, current_tasks_count, runnable_tasks_count
FROM sys.dm_os_schedulers
WHERE scheduler_id < 255

-- Detekce nejpomalejsich dotazu
SELECT
  SUBSTRING(st.text,(qs.statement_start_offset/2)+1,((CASE
statement_end_offset
    WHEN -1 THEN datalength(st.text) ELSE qs.statement_end_offset END
    - qs.statement_start_offset)/2) + 1) AS 'Prikaz',
  db_name(dbid) AS 'Databaze',
  object_name(objectid,dbid) AS 'Fce/Proc',
  total_elapsed_time / execution_count AS 'Prum. provadeci cas',
  execution_count, total_worker_time, total_elapsed_time,
  last_execution_time, creation_time,
  total_physical_reads, total_logical_reads, total_logical_writes
  dbid, objectid
FROM
  sys.dm_exec_query_stats AS qs CROSS APPLY
  sys.dm_exec_sql_text(qs.sql_handle) st
WHERE object_name(objectid,dbid) IS NULL -- dotazy mimo proc. a davky
ORDER BY total_elapsed_time / execution_count DESC
--ORDER BY total_worker_time DESC
--ORDER BY total_physical_reads DESC

-- Detekce nejpomalejsich procedur a davek
SELECT
  CASE WHEN dbid = 32767 THEN 'Resource' ELSE db_name(dbid) END AS
'Databaze',
  object_name(objectid,dbid) AS 'Fce/Proc',
  SUM(usecounts) AS 'Pocet pouziti',

SUBSTRING(CONVERT(char(23),DATEADD(ms,SUM(total_elapsed_time)/1000,0),121),
12,23)
    AS 'Celk. provadeci cas',
```

```
SUBSTRING(CONVERT(char(23),DATEADD(ms,SUM(total_elapsed_time/usecounts)/100
0,0),121),12,23)
    AS 'Prum. provadeci cas',
  SUM(total_logical_reads) / SUM(usecounts) * 1.0 AS 'Prum. pocet cteni',
  SUM(total_logical_writes) / SUM(usecounts) * 1.0 AS 'Prum. pocet zapisu',
  SUM(total_worker_time) / SUM(usecounts) * 1.0 AS 'CPU vytizeni',
  SUM(plan_generation_num) AS 'Pocet rekompilaci',
  dbid, objectid
FROM
  sys.dm_exec_cached_plans cp CROSS APPLY
  sys.dm_exec_sql_text(cp.plan_handle) JOIN
    (SELECT
       SUM(total_elapsed_time) AS total_elapsed_time,
       SUM(total_logical_reads) AS total_logical_reads,
         SUM(total_logical_writes) AS total_logical_writes,
         SUM(total_worker_time) AS total_worker_time,
         SUM(plan_generation_num) AS plan_generation_num,
         plan_handle
     FROM sys.dm_exec_query_stats
     GROUP BY plan_handle)
  qs ON cp.plan_handle = qs.plan_handle
WHERE objtype = 'Proc'
GROUP BY dbid, objectid
ORDER BY SUM(total_elapsed_time) / SUM(usecounts) * 1.0 DESC;
--ORDER BY SUM(total_worker_time) / SUM(usecounts) * 1.0 DESC;
--ORDER BY SUM(total_logical_reads) / SUM(usecounts) * 1.0 DESC;
--ORDER BY SUM(total_logical_writes) / SUM(usecounts) * 1.0 DESC;
--ORDER BY SUM(plan_generation_num) DESC;

-- Detekce paralelnich pozadavku
SELECT
  r.session_id,
  r.request_id,
  MAX(ISNULL(exec_context_id, 0)) AS number_of_workers,
  r.sql_handle,
  r.statement_start_offset,
  r.statement_end_offset,
  r.plan_handle
FROM
  sys.dm_exec_requests r
```

```
  JOIN sys.dm_os_tasks t ON r.session_id = t.session_id
  JOIN sys.dm_exec_sessions s ON r.session_id = s.session_id
WHERE
  s.is_user_process = 0x1
GROUP BY
  r.session_id, r.request_id,
  r.sql_handle, r.plan_handle,
  r.statement_start_offset, r.statement_end_offset
HAVING MAX(ISNULL(exec_context_id, 0)) > 0

-- Nalezeni planu jez by mohly bezet paralelne
SELECT
  db_name(qp.dbid) AS 'Databaze',
  object_name(qp.objectid,qp.dbid) AS 'Fce/Proc',
  cp.*
FROM
  sys.dm_exec_cached_plans cp
  CROSS APPLY sys.dm_exec_query_plan(cp.plan_handle) qp
  CROSS APPLY sys.dm_exec_sql_text(cp.plan_handle) AS st
WHERE
  cp.cacheobjtype = 'Compiled Plan' AND qp.query_plan.value
  ('declare namespace
p="http://schemas.microsoft.com/sqlserver/2004/07/showplan";
    max(//p:RelOp/@Parallel)', 'float') > 0

-- Info o optimalizatoru (pustit za obdobi)
SELECT *
FROM sys.dm_exec_query_optimizer_info
```

Analýza využití paměti SQL Serveru

```
-- Obecne vyuziti pameti
SELECT name, type, SUM(single_pages_kb + multi_pages_kb) AS MemoryUsedInKB
FROM sys.dm_os_memory_clerks
GROUP BY name, type
ORDER BY SUM(single_pages_kb + multi_pages_kb) DESC

-- Informace o vnitrni pameti
DBCC MEMORYSTATUS

-- Kolik zabiraji jednotl. prvky serveru (multipages)
SELECT type, SUM(multi_pages_kb)
FROM sys.dm_os_memory_clerks
WHERE multi_pages_kb != 0
GROUP BY type
ORDER BY SUM(multi_pages_kb) DESC

-- Mnozstvi celkove pameti zabirajici jednotl. komponenty
SELECT
  SUM(multi_pages_kb + virtual_memory_committed_kb +
shared_memory_committed_kb)
FROM sys.dm_os_memory_clerks
WHERE type <> 'MEMORYCLERK_SQLBUFFERPOOL'

-- Info o cache
SELECT
  distinct cc.cache_address,
  cc.name,
  cc.type,
  cc.single_pages_kb + cc.multi_pages_kb AS total_kb,
  cc.single_pages_in_use_kb + cc.multi_pages_in_use_kb AS total_in_use_kb,
  cc.entries_count,
  cc.entries_in_use_count,
  ch.removed_all_rounds_count,
  ch.removed_last_round_count
FROM
  sys.dm_os_memory_cache_counters cc JOIN
  sys.dm_os_memory_cache_clock_hands ch ON
  (cc.cache_address =ch.cache_address)
--WHERE ch.rounds_count > 0 AND ch.removed_all_rounds_count > 0
ORDER BY total_kb DESC
```

Analýza databází a informace o indexech

```
-- Analyza databazi
WITH AnalyzaDB AS (
SELECT
  vfs.database_id AS ID_DB,
  DB_NAME(vfs.database_id) AS Databaze,
  CASE WHEN mf.type = 1 THEN 'LOG' ELSE 'DATA' END AS Typ_souboru,
  SUM(vfs.num_of_bytes_read) AS CteniB,
  SUM(vfs.num_of_reads) AS PocetCteni,
  SUM(vfs.num_of_bytes_written) AS ZapisB,
  SUM(vfs.num_of_writes) AS PocetZapisu,
  SUM(vfs.num_of_bytes_read + vfs.num_of_bytes_written) AS Celkem_VV,
  SUM(vfs.io_stall) AS Cekani_na_VV
FROM
  sys.dm_io_virtual_file_stats(NULL, NULL) AS vfs JOIN
  sys.master_files AS mf ON
    vfs.database_id = mf.database_id AND vfs.file_id = mf.file_id
GROUP BY DB_NAME(vfs.database_id), mf.type, vfs.database_id
)
SELECT
  ROW_NUMBER() over(ORDER BY Cekani_na_VV DESC) AS Poradi,
  ID_DB, Databaze, Typ_souboru,
  cast(1.0 * CteniB/ (1024 * 1024) AS decimal(12, 2)) AS 'Cteni [MB]',
PocetCteni,
  cast(1.0 * ZapisB/ (1024 * 1024) AS decimal(12, 2)) AS 'Zapis [MB]',
PocetZapisu,
  cast(1.0 * Celkem_VV / (1024 * 1024) AS decimal(12, 2)) AS 'Celkem V/V
[MB]',
  cast(Cekani_na_VV / 1000. AS decimal(12, 2)) AS 'Cekani na V/V [sec]',
  cast(100. * Cekani_na_VV / SUM(Cekani_na_VV) over() AS decimal(10, 2)) AS
'Cekani na V/V [%]'
FROM AnalyzaDB
ORDER BY 'Cekani na V/V [sec]' DESC

-- Nejvice vytezujici V/V (logicke operace) pozadavky
SELECT top 5
  (total_logical_reads/execution_count) AS avg_logical_reads,
  (total_logical_writes/execution_count) AS avg_logical_writes,
  (total_physical_reads/execution_count) AS avg_phys_reads,
  Execution_count, statement_start_offset, sql_handle, plan_handle
FROM sys.dm_exec_query_stats
```

```
ORDER BY (total_logical_reads + total_logical_writes) DESC

-- Informace o databazi tempdb
SELECT
  SUM (user_object_reserved_page_count)*8 AS user_objects_kb,
  SUM (internal_object_reserved_page_count)*8 AS internal_objects_kb,
  SUM (version_store_reserved_page_count)*8  AS version_store_kb,
  SUM (unallocated_extent_page_count)*8 AS freespace_kb
FROM sys.dm_db_file_space_usage

-- Vyuziti indexu ve vsech databazich
SELECT *
FROM sys.dm_db_index_usage_stats
WHERE database_id > 4 --AND index_id <> 1
ORDER BY user_seeks DESC, system_seeks DESC

-- Vyuziti indexu v dane databazi
SELECT
  so.name AS 'Tabulka',
  si.name AS 'Index',
  si.type_desc AS 'Typ indexu',
  si.index_id AS 'Index ID',
  us.user_seeks, us.user_scans, us.system_seeks, us.system_scans
FROM
  sys.dm_db_index_usage_stats us INNER JOIN
  sys.objects so ON us.object_id = so.object_id INNER JOIN
  sys.indexes si ON so.object_id = si.object_id
WHERE so.type = 'U'
ORDER BY user_seeks DESC

-- Indexy, ktere nebyly pouzity
SELECT
  object_name(si.object_id),
  si.name,
  us.user_updates,
  us.user_seeks,
  us.user_scans,
  us.user_lookups
FROM
  sys.indexes si LEFT JOIN
  sys.dm_db_index_usage_stats us
```

```
    ON us.object_id = si.object_id AND si.index_id = us.index_id
WHERE
  objectproperty(si.object_id, 'IsIndexable') = 1 AND
  us.index_id is null OR
  (us.user_updates > 0 AND us.user_seeks = 0 AND us.user_scans = 0 AND
us.user_lookups = 0)
ORDER BY object_name(si.object_id) asc

-- Informace o indexech dane tabulky, ktere nebyly vubec vyuzity
SELECT si.name
FROM sys.indexes si
WHERE
  si.object_id=object_id('<tabulka>') AND
  si.index_id NOT IN (
     SELECT us.index_id
     FROM sys.dm_db_index_usage_stats us
     WHERE us.object_id = si.object_id AND si.index_id = us.index_id)

-- Pozadovane (chybejici) indexy
SELECT statement, equality_columns, inequality_columns, included_columns
FROM sys.dm_db_missing_index_details

-- Fragmentace indexu
SELECT top 20 *
FROM sys.dm_db_index_physical_stats(DEFAULT, DEFAULT, DEFAULT, DEFAULT,
'SAMPLED')
ORDER BY avg_fragmentation_in_percent DESC

SELECT
  object_name(i.object_id) AS Tabulka,
  i.name AS 'Index',
  sip.index_level as 'Uroven',
  sip.avg_page_space_used_in_percent AS 'Mira zaplneni',
  sip.avg_fragmentation_in_percent AS 'Externi (logicka) fragmentace',
  sip.fragment_count AS 'Pocet fragmentu',
  sip.avg_fragment_size_in_pages AS 'Pocet stranek na fragment',
  sip.ghost_record_count AS 'Duchove'
FROM
  sys.dm_db_index_physical_stats(db_id(), NULL, NULL, NULL, 'DETAILED') sip
INNER JOIN
  sys.indexes i
```

```
    ON i.object_id = sip.object_id
    AND i.index_id = sip.index_id
--WHERE sip.avg_fragmentation_in_percent > 50
ORDER BY sip.avg_fragmentation_in_percent DESC

/*
Typ zobrazeni informace:
  LIMITED = pouze stranky parent-level, velmi rychle
  SAMPLED = parent-level a vzorek listu
  DETAILED = parent-level p a všechny dat. stranky listu
*/
```

Příloha 2: Obsah CD

Přiložený disk CD-ROM obsahuje:

- Text této práce v elektronické podobě (.doc a .pdf)
- SQL skripty pro analýzu a konfiguraci SQL Serveru
- Šablona pro zápis analyzovaných hodnot