OPTIMALIZÁTOR SQL DOTAZŮ - ÚVOD DO PROBLEMATIKY

Dušan Kajzar, Magdaléna Chmelařová

Slezská univerzita v Opavě, Filozoficko - přírodovědecká fakulta, Bezručovo nám. 13,
746 01 Opava

Abstract
The article deals with the query optimizer. This theme belongs to the area of performance and tuning database servers. In the introduction we mention the importance of the process called query optimizer and symptoms of problems which show the necessity of monitoring the query plans. Other chapters list the strategies used by the query optimizer and some of these strategies are discussed in detail. The conclusion concerns all the possibilities of diagnosing and using its results for different professional groups of workers from the area of information systems and technologies of the company.

1. Úvod

Pomocí SQL dotazu zakomponovaného v aplikaci formuluje programátor požadavek vůči databázovému serveru. Říká mu, jaké informace, ze kterých databázových tabulek, jsou požadovány, co klient potřebuje vědět. Neříká již, jak má databázový systém dotaz zpracovat. Otázkou, jakým způsobem má být databázovým systémem dotaz efektivně zpracován, se zabývá tzv. optimalizátor SQL dotazů (query optimizer).

Optimalizátor SQL dotazů je tedy nedílnou součástí databázových systémů. Jde o proces, jehož úkolem je analyzovat jednotlivé SQL dotazy a

- určit potřebné zdroje pro zpracování dotazu, tj. tabulky, indexy, pracovní tabulky, alokaci paměti, ...
- optimalizovat SQL dotaz, tj. provést kroky vedoucí k redukci zdrojů na minimum a vybrat nejméně náročné postupy práce s vybranými zdroji

Tématicky patří problematika optimalizátoru SQL dotazů do rozsáhlé oblasti ladění výkonnosti databázového systému. Víme, že na výkonnost databázového systému má rozhodující vliv aplikační a databázový design. Znalosti principů práce optimalizátoru v daném databázovém systému a faktorů, jež mají na jeho rozhodování vliv, jsou tedy důležitou podmínkou pro

- sestavování efektivních SQL dotazů
- návrh databázového designu
- testování a hodnocení výkonnosti aplikace a databázového serveru, odhalování a odstraňování úzkých míst ve zpracování

V našem článku chceme formou úvodu do problematiky seznámit čtenáře:

- s funkcí optimalizátoru SQL dotazů v databázovém systému
- s některými faktory, které mají vliv na jeho práci při sestavování tzv. query plánů
- s možnostmi pro diagnostiku v oblasti ladění SQL dotazů
- s využitím informací o práci optimalizátoru programátorem, designerem databáze a systémovým administrátorem

Vycházíme přitom ze zkušeností při práci v databázovém systému Sybase, verze 11.5.

2. Příznaky problémů při zpracování SQL dotazů

Problémy při zpracování SQL dotazů se mohou projevovat následovně:

- SQL dotaz je prováděn mnohem pomaleji, než očekáváte vzhledem k velikosti tabulky a existenci indexů
- SQL dotaz je prováděn mnohem pomaleji, než obdobné dotazy
- provádění daného SQL dotazu je náhle mnohem pomalejší, než obvykle
- provádění SQL dotazu v rámci uložené procedury je pomalejší, než jeho provádění mimo uloženou proceduru
- monitorovacími metodami je zjištěno, že při provádění SQL dotazu není používán index, přestože podle vás vhodný index existuje

Možné příčiny těchto problémů pochopíme snáze po seznámení se s principy práce optimalizátoru SQL dotazů a s možnostmi diagnostiky v této oblasti.

3. Kroky systému při zpracování SQL dotazů

Po obdržení SQL dotazu od klienta provádí databázový server následující kroky:

- Text SQL dotazu je podroben větnému rozboru, je zkontrolována syntaxe dotazu a ověřeno, zda veškeré objekty, na něž se SQL dotaz odkazuje, existují.
  Tento krok zajišťuje proces zvaný parser, výsledkem jeho práce je tzv. syntaktický strom SQL dotazu.
- SQL dotaz zpracovaný parserem je optimalizován. Tento krok zajišťuje optimalizátor SQL dotazů. Během optimalizace zvažuje optimalizátor náročnost jednotlivých variant zpracování SQL dotazu. Tuto náročnost bodově hodnotí. Výsledkem jeho práce je optimální postup zpracování dotazu - tzv. query plán (plán provádění dotazu).
  Query plán je uspořádaná množina kroků potřebných k provedení dotazu, včetně přístupových metod k tabulkám a indexům.
- Vybraný query plán je kompilován.
- Kompilovaný kód query plánu daného SQL dotazu je spuštěn. Výsledek dotazu zasílá databázový server klientovi, který SQL dotaz podal.

Z výše uvedeného plyne, že vstupem pro proces optimalizátoru je syntaktický strom SQL dotazu a výstupem je plán provádění dotazu, tj. query plán. V dalším textu uvedeme strategie, které optimalizátor během analýzy SQL dotazu používá.

4. Strategie používané optimalizátorem SQL dotazů

K analýze a vytýčení možných variant zpracování dotazu má optimalizátor vestavěné algoritmy speciálních strategií. Provádí:

- analýzu SQL dotazu z hlediska jednotlivých strategií
- vytyčuje možné varianty zpracování SQL dotazu
- vytýčené varianty hodnotí podle vestavěných kritérií náročnosti zpracování
- provádí integraci získaných výsledků
- vybírá variantu, která je z hlediska stanovených kriterií nejméně náročná

Optimalizátor SQL dotazů se vždy snaží o výběr takové přístupové metody ke zdrojům, která minimalizuje náročnost zpracování dotazu. Náročnost je zvažována vhledem k době odezvy systému na dotaz a vzhledem k propustnosti systému, tj. množství zpracovaných transakcí za časovou jednotku. Prakticky to znamená snahu o minimalizaci zejména počtu fyzických přístupů na disky během zpracování dotazu, počtu logických čtení, tj. čtení z datové cache, minimalizaci režie spojené s reformátováním (tj. s pomocnou pracovní tabulkou resp.

pracovním indexem, který systém musí během zpracování dotazu v některých případech vytvořit) apod. Na těchto skutečnostech jsou založená kritéria hodnocení náročnosti variant.

Studium strategií optimalizátoru můžeme rozdělit do následujících témat:

- podmínky výběru - klauzule "where" v SQL dotazu
- pokrytí dotazu indexem
- zpracování podmínek s operátory "or" a "in" v SQL dotazu
- hodnocení efektivnosti při použití indexu
- zpracování agregačních funkcí v SQL dotazu
- zpracování klauzule "order by" - setřídění výsledků SQL dotazu
- problematika spojování tabulek - tzv. joins
- zpracování poddotazů v SQL dotazu
- zpracování klauzulí "group by" a "distinct"
- modifikace dat v tabulce - tzv. update
- zpracování SQL dotazů v uložených procedurách
- práce s datovou cache
- strategie zamykání zdrojů
- práce s kurzory

V tomto článku se z důvodů rozsáhlosti problematiky nemůžeme věnovat všem tématům podrobně. Všimněme si tedy principiálně jen témat souvisejících bezprostředně s výběrem indexu a hodnocením efektivnosti použití daného indexu.

Zaměříme-li se na problematiku přístupu ke větám databázové tabulky, pak optimalizátor může na základě svých rozhodovacích algoritmů zvažovat v podstatě dvě metody přístupu:

- pomocí přímého prohledávání zdrojové tabulky (tzv. table scan), nebo
- použitím indexu

Jde tedy o řešení otázek typu: Použít pro zpracování dotazu index ? Pokud ano, pak který ?
Jeden z prvních kroků, které optimalizátor v tomto směru provede, je analýza výběrových podmínek v klauzuli "where" SQL dotazu. Zabývejme se tedy nejprve podmínkami výběru v klauzuli "where", poté pokrytím SQL dotazu indexem a "or" strategií. Nakonec, v podkapitole "hodnocení efektivnosti při použití indexu", si povíme o významu systémových statistických tabulek pro hodnocení náročnosti varianty zpracování SQL dotazu.

4.1 Podmínky výběru - klauzule "where" v SQL dotazu

Analýza výběrových podmínek v klauzuli "where" SQL dotazu je jedním z prvních kroků, které optimalizátor provede. Výsledkem analýzy je separace tzv. podmínek typu SARG od podmínek ostatních a odhad náročnosti přístupových metod pro každou podmínku typu SARG v klauzuli "where". Vysvětlíme nejprve pojem podmínka typu SARG.
Podmínka typu SARG (zkratka pro search argument) je takovou podmínkou ve tvaru <sloupec> <operátor> <výraz> resp. <sloupec> is null, pro kterou je současně splněno:

- existuje index pro sloupec tabulky, jenž se v podmínce vyskytuje
- na daném sloupci nesmí být aplikována funkce nebo jiná operace
- aritmetický operátor je platným SARG operátorem
  (zde řadíme operátory =, <, >, <=, >=, is null)
- na pravé straně aritmetického operátoru musí být konstanta nebo výraz, jenž se dá výpočtem převést na konstantu
- sloupec a konstanta na pravé straně aritmetického operátoru jsou téhož datového typu

Jaký je význam uvedených podmínek ? Při splnění těchto podmínek pro daný SQL dotaz může optimalizátor efektivně využít pro zpracování dotazu vhodný index.
Aritmetický operátor nerovnosti (!=) není SARG operátorem. Při výskytu operátoru != nemůže optimalizátor pomocí indexu (přesněji pomocí systémových statistických tabulek k indexu - viz podkapitola 4.4) jednoduše odhadnout počet vět, které vyhovují podmínce "where", ani vymezit počáteční pozici pro vyhledávání pomocí indexu. Použití indexu může tedy být zamítnuto jako zbytečné resp. nevýhodné.

Příklad:
Mějme následující SQL dotaz a předpokládejme, že pro tabulku *seznam* existuje index s klíčem *prijmeni* :

```
SELECT prijmeni, jmeno, telefon FROM seznam
    WHERE prijmeni ="Novak" AND funkce="5472"
```

Podle výše uvedeného je výběrovou podmínkou typu SARG v našem příkladu podmínka
prijmeni ="Novak",
nikoliv ovšem
funkce="5472" (neexistuje index pro sloupec *funkce*).
V tomto případě by však optimalizátor pravděpodobně pro zpracování dotazu index podle sloupce *prijmeni* zvolil. Nelze však vyloučit ani přímé prohledávání tabulky *seznam* bez použití indexu. Např. při velkém počtu občanů se stejným příjmením v tabulce *seznam* v závislosti i na velikosti této tabulky.

V klauzuli "where" lze spojit podmínky SARG logickým operátorem "and", výsledkem je složená podmínka typu SARG. V našem příkladu ovšem již složená podmínka prijmeni="Novak" AND funkce="5472" není typu SARG.

Další příklady podmínek typu SARG (předpokládejme existenci indexů pro uvedené sloupce):

```
hmotnost > 500
hmotnost > 5 * 200    ..... výraz je převoditelný na konstantu
hmotnost > 500 AND cena < 6000
```

Příklady podmínek, jež nejsou typu SARG:

```
prijmeni = rodne_prijmeni ..... porovnávání hodnot dvou sloupců
substring(prijmeni, 1,3) = "Nov" ..... aplikace funkce nad sloupcem prijmeni
2 * hmotnost > 300 ..... operace se sloupcem hmotnost
hmotnost != 800    ..... výskyt operátoru !=
hmotnost = 900 OR hmotnost = 1200 ... logický operátor "or" není SARG operátorem
```

Příklady podmínek ekvivalentních s podmínkami typu SARG, tj. podmínek, které optimalizátor transformuje na typ SARG:

```
cena between 5000 AND 8000 .... transformováno na: cena >=5000 AND cena<=8000
prijmeni like "Nov%" ... transformováno na: prijmeni >="Nov" AND prijmeni<"Now"
```

Závěr k této strategii:
Vzhledem k výše uvedenému při sestavování SQL dotazu zvažme, zda je možné upravit klauzuli "where" SQL dotazu tak, aby buď celá, nebo alespoň její části byly podmínky typu SARG. Někdy je vhodné přidat do klauzule "where" redundantní podmínky, aby optimalizátor měl více možností pro rozhodování.

Například:

1)
```
SELECT pj.nazev_prodejny, pr.nazev_produktu, dod.nazev_dodavatele
    FROM prodejny pj, produkty pr, dodavatele dod
```

```
        WHERE pj.dodavatel_id = dod.dodavatel_id
                AND pj.produkt_id = pr.produkt_id
                AND pj.dodavatel_id >= 2000
2) SELECT pj.nazev_prodejny, pr.nazev_produktu, dod.nazev_dodavatele
        FROM prodejny pj, produkty pr, dodavatele dod
        WHERE pj.dodavatel_id = dod.dodavatel_id
                AND pj.produkt_id = pr.produkt_id
                AND dod.dodavatel_id >=2000
3) SELECT pj.nazev_prodejny, pr.nazev_produktu, dod.nazev_dodavatele
        FROM prodejny pj, produkty pr, dodavatele dod
        WHERE pj.dodavatel_id = dod.dodavatel_id
                AND pj.produkt_id = pr.produkt_id
                AND pj.dodavatel_id >=2000
                AND dod.dodavatel_id >=2000
```

Předpokládejme, že celé klauzule ve všech třech případech jsou složenými podmínkami typu SARG. Aniž bychom případy podrobněji rozebírali úvahami o velikosti tabulek, kardinalitě vztahů apod., uveďme, že případ 3) dává obecně optimalizátoru více možností pro optimalizaci dotazu (pro volbu pořadí prohledávání tabulek).

4.2 Pokrytí dotazu indexem

Říkáme, že SQL dotaz je pokrýván indexem, jestliže některý index obsahuje všechny informace nutné ke zpracování dotazu. Databázový server tedy čte pouze index, nikoliv datové stránky vlastní tabulky. To předpokládá:

- všechny sloupce v klauzuli "where" jsou klíčovými sloupci některého indexu
- všechny položky seznamu vybíraných položek příkazu SELECT jsou klíčovými sloupci téhož indexu

Taková ideální situace však málokdy nastane. Optimalizátor opět provádí odhad náročnosti různých variant řešení. Pro ilustraci uveďme následující příklady. Budeme předpokládat existenci indexu k tabulce *zamestnanci* se složeným klíčem "prijmeni, jmeno, funkce".

Příklady:

1) SELECT funkce FROM zamestnanci WHERE prijmeni="Novak" AND jmeno="Jan"
   Tento dotaz je plně pokrytý indexem, sloupce v klauzuli "where" (prijmeni, jmeno) jsou klíčovými položkami indexu, sloupec *prijmeni* je přitom prvním klíčovou položkou. Položka *funkce* v seznamu vybíraných položek je obsažena v klíči indexu.
2) SELECT jmeno, prijmeni FROM zamestnanci WHERE funkce="5075"
   Zde klauzule "where" neobsahuje první klíčovou položku indexu. Pro optimalizátor to bude patrně znamenat použití metody čtení tzv. listové úrovně (leaf level) indexu.
3) SELECT jmeno, prijmeni, telefon FROM zamestnanci WHERE funkce="5075"
   Výskyt navíc sloupce *telefon* v seznamu vybíraných položek již pravděpodobně povede k použití metody čtení tabulky *zaměstnanci* bez použití indexu (tj. k metodě table scan).
4) SELECT funkce, telefon FROM zamestnanci WHERE prijmeni="Novak" AND
   jmeno = "Jan"
   Klauzule "where" obsahuje vedoucí klíče indexu (prijmeni, jmeno), bude tedy použit index pro lokalizaci požadované věty. Navíc bude čtena i datová stránka tabulky *zamestnanci*, z důvodu zjištění hodnoty položky *telefon*.

4.3 Zpracování podmínek s operátory "or" a "in" v SQL dotazu

Jde o klauzule typu:
WHERE <sloupec1> <operátor > <hodnota1> OR <sloupec2> <operátor> <hodnota2> ....
nebo
WHERE <sloupec> IN <hodnota1, hodnota2, ...>
Klauzuli s operátorem "in" proces parser převádí na případ první.

V případě výskytu operátoru "or" ve výběrové podmínce optimalizátor zvažuje tyto dvě metody přístupu ke zdrojům:

- prohledávání zdrojové tabulky (tzv. table scan), nebo
- použití speciální "or strategie"

Optimalizátor použije metodu table scan, jestliže:

- klauzule "where" obsahuje sloupec, k němuž neexistuje index, nebo
- odhadovaná náročnost zpracování při použití indexů je větší než náročnost při table scan

Použití "or strategie" znamená:

- Optimalizátor vybere vhodný index pro každou z výběrových podmínek v klauzuli "where" spojených operátorem "or".
  Databázový server je tedy při této strategii schopen použít ke zpracování SQL dotazu více indexů současně.
- Dílčí množiny vět vyhovující jednotlivým výběrovým podmínkám jsou sloučeny ve výslednou množinu, přičemž jsou vyloučené duplicitní věty
  (nutno zabránit opakovanému výběru vět, jejichž hodnoty položek vyhovují současně více dílčím podmínkám spojeným operátorem "or").

Příklad:
SELECT * FROM seznam WHERE pracoviste ="8852" OR prijem >= 15000
Existují-li dva indexy pro sloupce *pracoviste* a *prijem*, optimalizátor může použít pro zpracování dotazu oba současně. Jeden pro výběrovou podmínku pracoviste ="8852", druhý pro výběrovou podmínku prijem >= 15000.

4.4 Hodnocení efektivnosti při použití indexu

Při hodnocení efektivnosti použití konkrétního indexu zvažuje optimalizátor následující kritéria:

- přibližně kolik vět tabulky bude tvořit výstup SQL dotazu
- přibližně kolik datových stránek bude muset být přitom (a při použití konkrétního indexu, pracovního indexu, pracovní tabulky) přečteno

K těmto odhadům slouží speciální tzv. systémové statistické tabulky, které si systém vytváří pro každý index a které uchovávají informace o rozložení hodnot klíčových položek v indexu, o průměrném procentu duplicitních hodnot klíče indexu apod.
Zmíněné statistické informace však databázový server neudržuje on-line aktuální. To by bylo velmi náročné na výkon serveru. K aktualizaci statistických informací pro jednotlivé tabulky a jejich indexy slouží speciální systémové procesy v současných databázových systémech nazývané většinou "update systémových statistik". Spuštění těchto procesů je načasováno na dobu mimo hlavní provoz databázového serveru.
Z toho však plyne jeden možný problém - systémové statistiky mohou vzhledem k dynamice dat v tabulce zastarat a optimalizátor se rozhoduje podle nepřesných informací. Nemusí tedy pro zpracování SQL dotazu zvolit optimální postup.

Minimalizace tohoto nebezpečí je otázkou koordinace práce mezi provozním oddělením a systémovou údržbou. Podkladem pro takovou koordinaci musí být dokumentace k systému vzniklá ve spolupráci návrhářů databáze, programátorů analytiků a systémových administrátorů.

Rozhodování systému o použití indexu může být v některých případech doplněno o analýzu efektivnosti prohledávání indexu obráceným směrem (tzv. backward scan). Jde o případy SQL dotazů s požadavkem na sestupné setřídění výsledků (klauzule "order by .... descending") a SQL dotazy s agregační funkcí "maximum". Tato metoda rozšiřuje databázovému serveru možnost pro použití indexu v případech, kdy by jinak musel vytvořit a setřídit pracovní tabulku (příkladem je požadavek "order by .... descending").
Použití metody zpětného prohledávání však nemusí být vždy efektivní. V praxi můžeme zaznamenat během provozu vyšší výskyt tzv. mrtvých zámků (deadlocks) v systému. Aby bylo možno vyhnout se problémům, umožňuje databázový server vydat zákaz pro používání metody zpětného prohledávání (pomocí parametru serveru).

5. MOŽNOSTI DIAGNOSTIKY PRÁCE OPTIMALIZÁTORU

Databázový server poskytuje nástroje pro diagnostiku práce optimalizátoru SQL dotazů. Hlavním nástrojem je uložená procedura "showplan", která umožňuje výpis posloupnosti jednotlivých kroků zpracování SQL dotazu tak, jak je sestavil optimalizátor.
To znamená výpis:

- použitých zdrojů (tabulek, indexů) při zpracování každého kroku
- použité strategie řešení a přístupové metody (pořadí spojovaných tabulek, pozicování dle indexu resp. table scan, využití indexních klíčů, vytvoření pomocné pracovní tabulky, vytvoření pomocného indexu, strategie agregace, třídění apod.)
- strategie práce s datovou cache během zpracování dotazu

Pomocí uvedeného nástroje lze monitorovat běh procesů přímo za rutinního provozu databázového serveru. Sledování a studium takových query plánů pro kritické procesy (SQL dotazy) pomáhá nalézat příčiny úzkých míst ve zpracování. Takovýmto způsobem používá nástroje "showplan" jak systémový administrátor zabývající se laděním databázového systému, tak i vývojový pracovník testující chování aplikací na provozních datech a za plného provozu.
Pomocí nástroje "showplan" je možno získat i výpisy query plánů bez spuštění vlastních SQL dotazů (parametrem je SQL dotaz, výsledkem je query plán dotazu). Tímto způsobem používá nástroj "showplan" vývojový pracovník testující efektivnost SQL dotazů v prostředí daného databázového designu.
Další diagnostické nástroje slouží k výpisům statistických údajů o:

- počtu čtení a zápisů na disky během zpracování SQL dotazu
- počtu čtení a zápisů v datové cache během zpracování SQL dotazu
- časech nutných pro parsing, kompilaci a pro vykonání jednotlivých kroků query plánu databázovým serverem

Pokročilejší diagnostika:
Užitím nástroje pro pokročilejší diagnostiku je možno pochopit, proč optimalizátor vybral jako optimální řešení zrovna tuto variantu a nikoliv jinou. Např. proč optimalizátor hodnotil metodu table scan lépe než přístup pomocí indexu, proč vybral jako vhodný přístup k datům přes index *index1* a nikoliv pomocí indexu *index2,* proč se rozhodl pro vytvoření pomocné pracovní tabulky, proč zvolil zrovna toto pořadí procházení tabulek apod.

6. Možnosti pro ovlivnění práce optimalizátoru

V případě zjištěné neefektivnosti při zpracování SQL dotazu a po analýze příčin jde vždy v první řadě o:

- analýzu a úpravu zdrojového textu SQL dotazu
- zhodnocení a případnou úpravu databázového designu

Pro úplnost však dodejme, že existují i metody, pomocí kterých lze databázovému serveru "vnutit" postup práce při zpracování SQL dotazu, tj. který index má použít, jakou strategii zamykání, pořadí spojovaných tabulek atd. Těchto metod může programátor použít po odborném posouzení situace a po zjištění, že optimalizátor nenavrhl (podle zhodnocení programátora) optimální postup zpracování dotazu. Problematika "donucovacích" metod je však již mimo rámec našeho článku. Chceme však před neuváženým použitím takových metod rozhodně varovat z několika důvodů:

- Použití "donucovací" metody je aplikačně závislé, tj. pohled na to, co je optimální postup zpracování dotazu se mění v čase s měnícími se objemy dat a databázovým návrhem. Použitím "donucovací" metody je však postup zpracování dán "natvrdo" přímo v SQL dotazu a optimalizátor je tímto vázán.
- Algoritmy optimalizátoru jsou dnes dobře propracované a praxe ukazuje, že zkušený programátor dává většinou za pravdu (i přes někdy se vyskytující pochybnosti) optimalizátoru.
- Donucovací metody je nutno chápat jako "poslední krok", kterým je v případě nezbytnosti možné zpracování SQL dotazu ovlivnit.

7. Využití znalostí o zpracování SQL dotazů

Závěrem shrneme důležitost informací poskytovaných diagnostickými nástroji k analýze query plánů pro:

- systémového administrátora
  Systémový administrátor využívá tyto informace v rámci testování a ladění výkonnosti databázového serveru. Informace o query plánech kritických procesů jsou užitečným doplňkem dalších speciálních monitorovacích a ladících nástrojů.
- programátora
  Programátor informace využívá při testování efektivnosti jednotlivých SQL dotazů a aplikace jak ve vývojovém prostředí, tak zejména v reálném provozním prostředí.
- designera databáze
  Při znalostech query plánů může designer databáze lépe posoudit vhodnost databázového designu vzhledem k množině SQL dotazů, která s danou databází pracuje.

Litaratura

1. Sybase: Sybase Adaptive Server Enterprise - Performance and Tuning Guide
2. Microsoft: System Administration for MS SQL Server 7.0 - Optimizing Database Performance