

Optimalizace práce optimalizátoru Oracle10g

Ivan Halaška, Karel Richta
katedra počítačů, FEL ČVUT Praha
{halaska,richta}@fel.cvut.cz

Motto:

Databáze obsahuje data
Uživatelé se na data ptají
Hloupý systém přehrabuje data bez rozmyslu
Chytrý systém obsahuje optimalizátor
Optimalizátor optimalizuje – plánuje efektivní provedení příkazů
Chytrý optimalizátor si něco pamatuje
Někdy se snaží být moc chytrý a sestavuje mnoho plánů - to mu může
trvat dlouho
Můžeme mu ale poradit
Proč tedy optimalizovat práci optimalizátoru?
Protože jen hlupák si nenechá poradit.
Proč zrovna Oracle10g?
Protože ho známe.
Jedná se o malý výlet do světa práce SŘBD.

Obsah

- Volba optimálního plánu
- Sdílení dříve vytvořených plánů
- Podklady pro rozhodování optimalizátoru – statistiky
- Možnosti snížení režie pořizování aktuálních statistik
- Možnosti snížení režie rozhodování optimalizátoru
 - Napovídání pomocí tzv. „hint“ nápovědy
 - Znovupoužitelné osnovy plánů provedení
- Upřesnění podkladů pro optimalizátor v podobě SQL profilů

Úvod

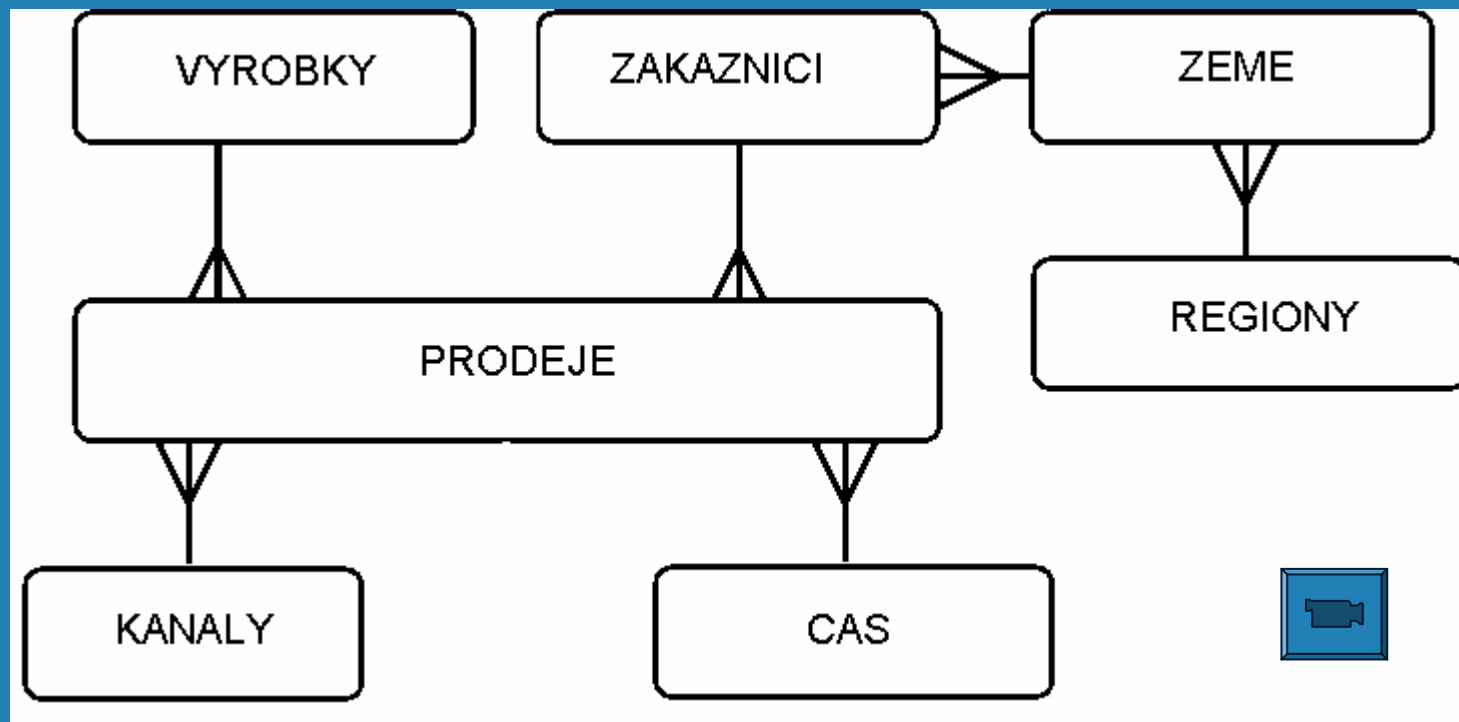
- Úkolem optimalizátoru relačního SŘBD je vybrat pro zadaný příkaz v jazyce SQL jeden z možných plánů provedení a to pokud možno plán nejlepší.
- Optimalizátor se rozhoduje na základě analýzy příkazu a jemu dostupných informací o objektech, na které se příkaz odkazuje.
- Vedle kvality tohoto rozhodování je neméně důležitým problémem efektivnost sběru a zpracování informací potřebných pro kvalitní rozhodnutí.
- Jak to dělá optimalizátor SŘBD Oracle?

Použitý příklad

- Jako demonstrační příklad budeme používat jednoduchý fragment modelu, který Oracle dodává jako součást instalace.
- Jedná se o data se záznamy o prodejních případech:

„Daný výrobek byl prodán danému zákazníkovi v daný čas, prostřednictvím daného prodejního kanálu. Daný zákazník patří do jedné země, v dané zemi může být mnoho zákazníků. Daná země patří do jednoho regionu, v dané regionu může být mnoho zemí.“

Schéma pro použitý příklad



Reprezentace modelu

- V databázi tomuto modelu odpovídá spojení jedné velké tabulky faktů PRODEJE, která obsahuje záznamy o prodejních případech, s několika malými (dimenzionálními) tabulkami (spojení do hvězdy).
- „Vraní nohy“ ve schématu naznačují kardinalitu vztahu 1:N, která je relačně implementována pomocí cizích klíčů v tabulce PRODEJE. Ty se odkazují na primární klíče jednotlivých dimenzionálních tabulek.
- Tabulky mají primární klíče (VYROBEK_ID, ZAKAZNIK_ID, ZEME_ID, REGION_ID, KANAL_ID a CAS_ID) a nad nimi jsou vytvořeny unikátní indexy typu B-strom. Nad cizími klíči v tabulce PRODEJE jsou vytvořeny bitmapové neunikátní indexy.

Provádění SQL příkazů v SŘBD Oracle

- Po zadání SQL příkazu se nejprve hledá dostatečně podobný příkaz ve sdílené paměti příkazů (podle tzv. “hash code”).
- Pokud se příkaz ve sdílené paměti najde, využijí se uložené informace, např. i exekuční plán příkazu.
- Pokud se nenajde, je příkaz analyzován a vybrán exekuční plán pro jeho provedení.
- Motivace je: analyzovat příkaz jednou, ale opakovaně využít výsledků analýzy při exekuci příkazu.

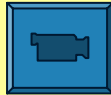
Plán provedení SQL příkazu

- Výsledkem analýzy je plán provedení daného SQL příkazu.
- Plán je zaznamenáván v podobě stromu elementárních operací.
- Každý krok plánu představuje jeden uzel stromu operací. Výsledek jedné dílčí operace je předán ke zpracování operaci, která je ve stromu spjata s uzlem – předchůdcem.

Příklad 1: Ilustrace plánů provedení

Zadejme co nejjednodušší dotaz:

```
select count(*) "Počet zákazníků"  
from ZAKAZNICI;  
  
Počet zákazníků  
-----  
55500
```



Tabulka zákazníků (ZAKAZNICI) má primární klíč s odpovídajícím unikátním indexem (ZAKAZNICI_PK). Navíc má bitmapový index nad sloupcem POHLAVI (ZAKAZNICI_POHLAVI_BIX).

Plán provedení

(Příklad 1)

Optimalizátor se rozhoduje mezi neznámým počtem plánů, uvažme tři z nich:

- **První možný plán** je založen na úplné prohlídce všech bloků datového segmentu tabulky zákazníků.
- **Druhý možný plán** prochází klasický index typu B-strom nad primárním klíčem (spočtou se elementy v listových uzlech indexového stromu, požadované informace o počtu řádků tabulky lze zjistit i „pouhým“ spočtením elementů v indexu).
- **Třetí možný plán** spočívá v prohlížení bitmapového indexu nad sloupcem POHLÁVI (spočítá se, kolik bitových pozic má bitová mapa indexu).

Plán provedení I. (Příklad 1)

<i>Id</i>	<i>Id_předchůdce</i>	<i>Operace</i>	<i>Jméno objektu</i>	<i>Cena</i>
0		SELECT STATEMENT		329
1	0	SORT AGGREGATE		
2	1	TABLE ACCESS FULL	ZAKAZNICI	328

Naplánováno úplné prohledání datového segmentu

Plán provedení II. (Příklad 1)

<i>Id</i>	<i>Id_před- chůdce</i>	<i>Operace</i>	<i>Jméno objektu</i>	<i>Cena</i>
0		SELECT STATEMENT		117
1	0	SORT AGGREGATE		
2	1	INDEX FULL SCAN	ZAKAZNICI_PK	117

Naplánováno prohledání indexu nad primárním klíčem.
Cena je přibližně třikrát příznivější.

Plán provedení III. (Příklad 1)

<i>Id</i>	<i>Id_před- chůdce</i>	<i>Operace</i>	<i>Jméno objektu</i>	<i>Cena</i>
0		SELECT STATEMENT		3
1	0	SORT AGGREGATE		
2	1	BITMAP CONVERSION COUNT		2
3	2	BITMAP INDEX FAST FULL SCAN	ZAKAZNICI_POHLAVI_BIX	

Naplánována prohlídka bitmapového indexu. Cena je o dva řády příznivější oproti prvnému plánu.

Volba (sub)optimálního plánu

- Plány jsou uspořádány od nejdražšího k nejlevnějšímu.
- Odhadnutá cena pro každý plán vyjadřuje zejména počet logických čtení databázových bloků.
- Nejlevnější plán za daných podmínek je vybrán jako (sub)optimální.
- V uvedeném příkladu je to třetí plán procházející bitmapový index.

Sdílení dříve vytvořených plánů I.

- Pro každý SQL příkaz, který se má provést, se v operační paměti serveru vytvoří speciální kurzor. Ten má dvě části.
 - První část je jedinečná a vzniká v oblasti paměti vyhrazené pro dané uživatelské připojení (session).
 - Druhá část kurzoru, říkáme jí **kontext kurzoru** je ve sdílené paměťové oblasti serveru (cursor cache).
 - Do kontextu kurzoru se uloží zejména: zdrojový text příkazu, seznam objektů, na které příkaz odkazuje a plán provedení příkazu.
 - Kontextová část kurzoru může být sdílena více kurzory.

Sdílení dříve vytvořených plánů II.

- V první fázi zpracování nového příkazu optimalizátor vyhodnotí, zda je možné pro příkaz využít některý z již vytvořených kontextů kurzoru.
- V případě, že ano, ušetří se místo ve sdílené oblasti operační paměti a ušetří se výpočetní kapacita, kterou by bylo nutné vynaložit při nové analýze příkazu.
- Jestliže neexistuje použitelný kontext kurzoru, založí se kontext nový a proběhne úplná analýza nového příkazu. Jejím výsledkem bude nový kontext kurzoru s novým plánem provedení.
- Nalezený nebo vytvořený plán je v dalších fázích zpracování příkazu aplikován.

Sdílení dříve vytvořených plánů III.

- Při hledání, zda je možné použít některý existující kontext kurzoru se optimalizátor rozhoduje na základě porovnání textu příchozího příkazu s texty příkazů, které jsou uloženy v zapamatovaných kontextech kurzorů.
- Nakolik striktně má nový příkaz textově odpovídat příkazu v kurzoru je možno nastavit pomocí úrovně podobnosti. Ta má v podstatě dvě zásadní varianty označované jako „EXACT“ a „SIMILAR“.
- Existují techniky, které upravují příchozí příkazy do „konfekční podoby“, čímž se zvýší pravděpodobnost, že dva příkazy mající stejný význam, budou odhaleny jako příkazy, pro něž je možné použít stejný kontext kurzoru.

Sdílení dříve vytvořených plánů IV.

- U kontextu kurzoru, jehož příkaz je dostatečně podobný příchozímu příkazu, se dále prověřuje, zda stejná jména v textově podobných příkazech odkazují ke stejným databázovým objektům.
- Uplatňuje se předpoklad, že po dobu existence sdíleného kurzoru se nezměnily podmínky, za jejichž platnosti byl pro příkaz zvolen (sub)optimální plán, zaznamenaný do kontextu kurzoru.
- V případě, že dojde ke změně u některého objektu, kurzor se stane automaticky neplatným a není nadále využíván. Změnou může být například to, že pro tabulku byly nově posbírány statistiky nebo byla znovu přeložena funkce volaná v příkazu.
- Kontext kurzoru vydrží ve sdílené paměti po omezenou dobu. Dříve nebo později bude jím obsazená paměť použita pro jiný účel.

Sdílení dříve vytvořených plánů V.

- K formulaci nového plánu přistoupí optimalizátor teprve jako k poslední možnosti.
- Pokud je k tomu donucen, sestaví pro tento účel všechny v úvahu přicházející plány.
- Pro každý plán odhadne cenu provedení a do kontextu kurzoru zaznamená plán nejlevnější.
- Při odhadu ceny plánu potřebuje optimalizátor statistiky. Sběr statistik představuje určitou režii systému.



Možnosti snížení režie sběru statistik I.

- Volba přístupové cesty k vybrané množině řádků zahrnuje rozhodování o tom, zda bude či nebude naplánováno použití toho kterého indexu.
- Jedním ze základních východisek je odhad **selektivity** výběrové podmínky použité v příkazu (kolik procent řádků z celkového množství bude vybráno do odpovědi).
- Zhruba řečeno, čím lepší je selektivita dotazu, tím výhodnější bude použít příslušný index.
- Jsou rozlišovány sloupce s rovnoměrným a nerovnoměrným rozložením zastoupených hodnot.

Možnosti snížení režie sběru statistik II.

- Pro sloupce s rovnoměrným rozložením hodnot lze selektivitu zadaného predikátu odhadnout na základě znalosti počtu řádků tabulky, počtu různých hodnot zastoupených ve sloupci, min. a max. hodnoty Tyto údaje optimalizátor nazývá **základními statistikami**.
- Pro sloupce s nerovnoměrným zastoupením jednotlivých hodnot je selektivita dotazu pro každou hodnotu jiná. Měli bychom pro každou hodnotu znát kolikrát je ve sloupci zastoupena. Takováto statistika se nazývá **histogram**.
- U indexu je pro odhady ceny potřebná například znalost hloubky jeho B-stromu a faktor shluknutí řádků se stejnou hodnotou indexovaného klíče (shluknutí do společných datových bloků).

Možnosti snížení režie sběru statistik III.

- Při odhadu selektivity a dalších faktorů vedoucích ke stanovení ceny plánu optimalizátor využívá, vedle základních statistik a histogramů, posbíraných pro datové objekty (tabulky, jejich sloupce a indexy), i **systemové statistiky** dokumentující hospodaření s operační pamětí a výpočetní kapacitou procesoru.
- Sběr statistik představuje nezanedbatelnou režii a může tedy ovlivnit průchodnost systému pro další prováděné úlohy.
- Použití neaktuálních statistik může vést k volbě neadekvátního plánu provedení SQL příkazu, což také může vést ke snížení průchodnosti systému.
- Problém je jak rozhodnout, které statistiky jsou zastaralé (stale) a musí být tudíž aktualizovány.
- Zatímco dříve se spíše uvažovalo o tom, zda je nutné znovu analyzovat danou tabulku nebo index, zvyšující se výpočetní síla serverů nabízí extenzivní uvažování. Předmětem sběru statistik jsou pak spíše „všechny objekty daného schématu“ nebo „všechny objekty celé databáze“.

Možnosti snížení režie sběru statistik IV.

- Základní potřebou je průběžně udržovat statistiky aktuální, ale s co nejmenším vlivem na průchodnost ostatních úloh.
- Je třeba naplánovat aktualizaci statistik na období klidového provozu a aktualizovat pouze ty statistiky, které si to zaslouží, které od posledního sběru **zastaraly**.
- Pro potřebu vyhodnocení, zda statistiky nad danou tabulkou ztratily na aktuálnosti, lze zapnout automatické sledování změn dat v tabulce. Statistika nad tabulkou může být například prohlášena za neaktuální, když se data v tabulce od posledního sběru změnila v rozsahu cca 10%.
- Proces pro periodickou aktualizaci zastaralých statistik nad objekty celé databáze je zařazen do fronty úloh při vytvoření databáze. Jeho spuštění je naplánováno na období, které je pomocí parametrů databáze prohlášeno za období s nízkým transakčním provozem. Implicitně to je noc a celé víkendové dny.

Možnosti snížení režie sběru statistik V.

Předpokládejme, že udržujeme několik instancí databázové aplikace u zákazníků. Lze zvolit tuto strategii:

- Režijně náročný sběr statistik je možné provádět pouze na referenční instanci.
- Statistiku lze pravidelně exportovat z datového slovníku do uživatelských tabulek, tyto přenést do databází ostatních instancí aplikace a odtud importovat do tamního datového slovníku.
- Tam lze potom ušetřit výpočetní kapacitu a statistiky vůbec nesbírat. Případně zabránit opětovnému sběru importovaných statistik tím, že je navíc **zamkneme**.

Napovídání pomocí tzv. „hint“ nápovědy

- Práci optimalizátoru můžeme urychlit tím, že mu připomeneme, jak se rozhodl v minulosti.
- Jedním z prostředků, který systém Oracle správci aplikace a programátorovi nabízí, jsou tzv. „hint“ nápovědy.
- Pomocí nich je možné vytyčit pro rozhodování optimalizátoru určité mantinely. Použití některých přístupových cest optimalizátoru nedoporučíme, použití jiných mu doporučíme.
- Nejde o striktní příkazy, jsou to spíše *tipy*, podle kterých se optimalizátor může řídit, ale také nemusí. Z hlediska jazyka SQL je nápověda „hint“ realizována komentářem, který má speciální umístění a vlastní syntaxi.

Příklad 2: Nápořveda typu „hint“ I.

Vraťme se k dotazu z příkladu 1 - jak přimějeme optimalizátor k volbě některého z uvedených tří plánů?

```
select count(*) from ZAKAZNICI z;
```



- Optimalizátor o své vůli zvolí nejlevnější plán:

<i>Id</i>	<i>Id_před- chůdce</i>	<i>Operace</i>	<i>Jméno objektu</i>	<i>Cena</i>
0		SELECT STATEMENT		3
1	0	SORT AGGREGATE		
2	1	BITMAP CONVERSION COUNT		2
3	2	BITMAP INDEX FAST FULL SCAN	ZAKAZNICI_PO HLAVI_BIX	

Příklad 2: Náповěda typu „hint“ II.

- Dejme optimalizátoru pokyn, aby ignoroval nevyhovědnější bitmapový index nad sloupcem ZAKAZNICI.POHLAVI:

```
select  --+NO_INDEX(z ZAKAZNICI_POHLAVI_BIX)  
count(*) from ZAKAZNICI z;
```

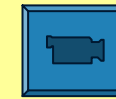


<i>Id</i>	<i>Id_předchůdce</i>	<i>Operace</i>	<i>Jméno objektu</i>	<i>Cena</i>
0		SELECT STATEMENT		117
1	0	SORT AGGREGATE		
2	1	INDEX FULL SCAN	ZAKAZNICI_PK	117

Příklad 2: Nápořveda typu „hint“ III.

Nejdražší z plánů v příkladu 1 si optimalizátor vybere až tehdy, když mu dáme tip, aby ignoroval všechny indexy nad tabulkou ZAKAZNICI.

```
select  --+NO_INDEX(z)  
count(*) from ZAKAZNICI z;
```



<i>Id</i>	<i>Id_před-chůdce</i>	<i>Operace</i>	<i>Jméno objektu</i>	<i>Cena</i>
0		SELECT STATEMENT		329
1	0	SORT AGGREGATE		
2	1	TABLE ACCESS FULL	ZAKAZNICI	328

Nápověda typu „hint“ - souhrn

Pokynů typu hint je velké množství. Pomocí nich dáváme optimalizátoru tipy:

- jaké přístupové cesty má upřednostnit/nemá brát v úvahu
- v jakém pořadí má naplánovat spojení zdrojových tabulek
- jaké techniky spojení má použít pro danou tabulku, když na ni přijde řada.

Pro jeden příkaz je možné zadat kombinaci několika tipů typu „hint“, akceptovány budou tehdy, nebudou-li se vzájemně popírat.

Některé tipy pro volbu módu optimalizátoru

- ALL_ROWS
- FIRST_ROWS_{1|10|100|1000}
- CHOOSE
- RULE

Využívání osnov I.

- Oracle poskytuje prostředky, pomocí nichž se buď jednorázově, nebo po zvolenou dobu, zaznamenávají zpracovávané SQL příkazy společně s plány, které pro ně optimalizátor zvolil.
- Záznamy mají podobu **znovupoužitelných osnov** (tzv. „**outlines**“) a zaznamenávají se do specializované části datového slovníku, kterou tvoří samostatné schéma jménem **OUTLN**.
- Zaznamenané osnovy je možné seskupovat do skupin, které se nazývají **kategorie**. Na databázovém stroji, který slouží jako etalon pro skupinu zákaznických instalací, můžeme při čerstvých statistikách vytvořit různé sady osnov pro období s různým typem zátěže a pro databáze s různou robustností.

Využívání osnov II.

- Celé schéma OUTLN je možné technikou export/import přenést do databází zákazníků.
- V rutinním provozu je možné „zapnout“ používání dané kategorie osnov.
- Po příchodu nového příkazu optimalizátor vyhodnotí, zda je možné pro příkaz využít již existující kontext kurzoru.
- Jestliže ne, vytvoří se nový kontext kurzoru. Pak se prohledá zapnutá kategorie osnov, zda v ní je osnova zpracovávaného příkazu. Jestliže ano, použije se tato osnova jako velmi striktní vodítko při stavbě nového plánu.
- Může se tak ušetřit na režii údržby aktuálních statistik, protože optimalizátor, vedený návodem v osnově je nepotřebuje.
- Může se snížit výpočetní náročnost stavby plánu, protože při použití osnovy optimalizátor nemusí téměř nic rozhodovat.

Využívání osnov III.

Příklad 3: předpokládejme, že máme v databázi uloženu tabulku zákazníků ZAKAZNICI a tabulku zemí ZEME.

```
select
  ze.jmeno_zeme,
  za.zak_mesto,
  za.zak_prijmeni
from ZEME ZE
     join ZAKAZNICI ZA using (zeme_id);
```



Využívání osnov IV.

Příklad 3: Podívejme se jaký plán optimalizátor zvolí, když nemá k dispozici základní statistiky:

```
analyze table ZAKAZNICI delete statistics;  
analyze table ZEME      delete statistics;  
explain plan for  
  select ze.jmeno_zeme  
         , za.zak_mesto  
         , za.zak_prijmeni  
  from ZEME ze join ZAKAZNICI za using (zeme_id);  
Vysvětleno.
```



Využívání osnov V.

Příklad 3: volba plánu při absenci statistik

Id	Id_předchůdce	Operace	Jméno objektu
0		SELECT STATEMENT	
1	0	NESTED LOOPS	
2	1	TABLE ACCESS FULL	ZAKAZNICI
3	1	TABLE ACCESS BY INDEX ROWID	ZEME
4	3	INDEX UNIQUE SCAN	ZEME_PK_IDX

Predicate Information (identified by operation id):

4 – access ("ZE"."ZEME_ID"="ZA"."ZEME_ID")

note: - rule based optimizer used (consider using cbo)

Využívání osnov VI. - vysvětlení

- Protože nejsou k dispozici statistiky, optimalizátor není schopen odhadovat ceny plánů, které přicházejí v úvahu. Použije proto modul, který se rozhoduje na základě pevně stanovených pravidel vyhodnocujících zápis příkazu (v závěrečné poznámce k plánu je výzva zvážit použití nákladově orientovaného optimalizátoru – cbo = cost based optimizer).
- V plánu bylo zvoleno spojení tabulek technikou vnořených cyklů (NESTED LOOPS). Krok 2 plánu předepisuje, že ve vnějším cyklu se bude jednou úplně procházet tabulka ZAKAZNICI a pro každý řádek této tabulky se dohledá odpovídající řádek z tabulky ZEME s využitím indexu nad primárním klíčem, který se jmenuje ZEME_PK_IDX (krok 4, jeho výsledek bude použit v kroku 3).

Využívání osnov VII. použití statistik

- Uposlechneme pokynu v poznámce pod zobrazeným plánem a posbíráme statistiky pro obě tabulky, čímž umožníme, aby byl nasazen nákladově orientovaný modul optimalizátoru (cbo).

Využívání osnov VIII.

Příklad 3: získání plánu za přítomnosti statistik:

```
analyze table ZAKAZNICI compute statistics;  
analyze table ZEME      compute statistics;  
explain plan for  
  select ze.jmeno_zeme  
         , za.zak_mesto  
         , za.zak_prijmeni  
from ZEME ze join ZAKAZNICI za  
              using(zeme_id);
```



Využívání osnov IX.

Příklad 3: plán provedení získaný za přítomnosti statistik

<i>Id</i>	<i>Id_předchůdce</i>	<i>Operace</i>	<i>Jméno objektu</i>	<i>Cena (%CPU)</i>
0		SELECT STATEMENT		334 (2)
1	0	HASH JOIN		334 (2)
2	1	TABLE ACCESS FULL	ZEME	3 (0)
3	1	TABLE ACCESS FULL	ZAKAZNICI	329 (1)

Predicate Information (identified by operation id):
1 - access ("ZE"."ZEME_ID"="ZA"."ZEME_ID")

Poznámka: protože byl použit cbo, pro plán je známa cena

Využívání osnov X.

Následující příklad ukazuje vytvoření jedné **osnovy** a nastavení podmínek pro optimalizátor tak, aby se rozhodoval podle této osnovy.

Využívání osnov XI.

Příklad 3: vytvoření osnovy

```
create or replace outline DATAKON
for category DATAKON_CAT on
  select ze.jmeno_zeme
         , za.zak_mesto
         , za.zak_prijmeni
from ZEME ze
     join ZAKAZNICI za using (zeme_id);
```

Osnova vytvořena.



Osnova „DATAKON“ pro kategorii „DATAKON_CAT“

Využívání osnov XII.

Pro co se optimalizátor rozhodne, jsou-li zrušeny statistiky a používání osnov je vypnuto?

```
-- zrušení statistik
analyze table ZAKAZNICI delete statistics;
analyze table ZEME      delete statistics;
-- vypnutí využívání osnov
alter session set USE_STORED_OUTLINES = FALSE;
-- analýza dotazu
explain plan for
  select ze.jmeno_zeme
         , za.zak_mesto
         , za.zak_prijmeni
  from ZEME ze join ZAKAZNICI za using (zeme_id);
```



Využívání osnov XIII.

Pro co se optimalizátor rozhodne, jsou-li zrušeny statistiky a používání osnov je *vypnuto*?

<i>Id</i>	<i>Id_před-čůdce</i>	<i>Operace</i>	<i>Jméno objektu</i>
0		SELECT STATEMENT	
1	0	NESTED LOOPS	
2	1	TABLE ACCESS FULL	ZAKAZNICI
3	1	TABLE ACCESS BY INDEX ROWID	ZEME
4	3	INDEX UNIQUE SCAN	ZEME_PK

Predicate Information (identified by operation id):

4 - access("ZE"."ZEME_ID"="ZA"."ZEME_ID")

note: - rule based optimizer used (consider using cbo)

Využívání osnov XIV.

Pro co se optimalizátor rozhodne, jsou-li zrušeny statistiky a *zapneme* používání osnov?

```
-- zapnutí využívání kategorie osnov
alter session set USE_STORED_OUTLINES = DATAKON_CAT;
-- analýza dotazu
explain plan for
  select ze.jmeno_zeme
         , za.zak_mesto
         , za.zak_prijmeni
  from ZEME ze join ZAKAZNICI za using (zeme_id);
Vysvětleno.
```



Využívání osnov XV.

Pro co se optimalizátor rozhodne, jsou-li zrušeny statistiky a *zapneme* používání osnov?

<i>Id</i>	<i>Id_před-čůdce</i>	<i>Operace</i>	<i>Jméno objektu</i>	<i>Cena (%CPU)</i>
0		SELECT STATEMENT		334 (2)
1	0	HASH JOIN		334 (2)
2	1	TABLE ACCESS FULL	ZEME	3 (0)
3	1	TABLE ACCESS FULL	ZAKAZNICI	329 (1)

Predicate Information (identified by operation id):
1 - access("ZE"."ZEME_ID"="ZA"."ZEME_ID")

Využívání osnov XVI.

- Příklad měl pomoci vytvořit si představu o konstrukci osnovy, v praxi se ovšem spíše použije technika masivního záznamu osnov pro všechny plány, které optimalizátor použil v intervalu vymezeném příkazy:

```
alter system set create_stored_outlines = kategorie;  
... běh aplikace  
alter system set create_stored_outlines = FALSE;
```

Příklad 4: přínos osnovy

- V předchozím příkladu jsme ilustrovali, jak optimalizátor přimět k tomu, aby zvolil plán, ke kterému by v dané situaci normálně nedospěl.
- V tomto příkladu se podíváme na to, jak použití osnovy ovlivní dobu analýzy příkazu. Zkusíme složitější dotaz, aby optimalizátor měl více práce při volbě plánu.

Příklad 4: přínos osnovy II.

- Mějme dotaz, u něhož je analýza poněkud složitější. Vytvoříme podmínky pro volbu optimálního plánu, plán zaznamenáme do osnovy.
- Za nadále platných optimálních podmínek necháme příkaz dvakrát zpracovat. Jednou při zapnutí osnovy, podruhé při jejím vypnutí.
- V obou případech optimalizátor dospěje ke stejnému plánu. Porovnáme dobu, kterou k tomu potřeboval.

Příklad 4: přínos osnovy III.

Dotaz je postaven tak, aby nevybral žádné řádky, takže fáze „Execute+Fetch“ budou ve srovnání s fází „Parse“ trvat mnohem kratší dobu:

```
select p.prodano_kusu
       , v.vyrobek_nazev
       , k.kanal_popis
from VYROBKY v
     join PRODEJE p on (v.vyrobek_id=p.vyrobek_id and
                       v.vyrobek_id > 145)
     join KANALY k on (k.kanal_id=p.kanal_id and
                       k.kanal_id in(1,2,3))
     join ZAKAZNICI z on (z.zak_id=p.zak_id and
                          z.zak_mesto = 'Asten');
```



Příklad 4: přínos osnovy IV.

Plán provedení dotazu:

<i>Id</i>	<i>Operace</i>	<i>Jméno objektu</i>	<i>Cena (%CPU)</i>
0	SELECT STATEMENT		32 (100)
1	NESTED LOOPS		32 (0)
2	NESTED LOOPS		31 (0)
3	NESTED LOOPS		30 (0)
4	PARTITION RANGE ALL		29 (0)
5	TABLE ACCESS BY LOCAL INDEX ROWID	PRODEJE	29 (0)
6	BITMAP CONVERSION TO ROWIDS		
7	BITMAP INDEX RANGE SCAN	PRODEJE_VYR_BIX	
8	TABLE ACCESS BY INDEX ROWID	KANALY	1 (0)
9	INDEX UNIQUE SCAN	KANALY_PK	0 (0)
10	TABLE ACCESS BY INDEX ROWID	VYROBKY	1 (0)
11	INDEX UNIQUE SCAN	VYROBKY_PK	0 (0)
12	TABLE ACCESS BY INDEX ROWID	ZAKAZNICI	1 (0)
13	INDEX UNIQUE SCAN	ZAKAZNICI_PK	0 (0)

Příklad 4: přínos osnovy V.

Vytvoření osnovy:

```
create or replace outline datakon_  
for category datakon_cat on  
select . . .
```

Trasování zpracování příkazu bez použití osnovy:

```
alter session set TRACEFILE_IDENTIFIER='bez_osnovy';  
alter session set SQL_TRACE=TRUE;  
alter session set USE_STORED_OUTLINES = FALSE;  
explain plan for  
select . . . ;  
alter session set SQL_TRACE=FALSE;
```



Příklad 4: přínos osnovy VI.

Trasování zpracování příkazu s použitím osnovy:

```
alter session set TRACEFILE_IDENTIFIER = 's_osnovou';  
alter session set SQL_TRACE=TRUE;  
alter session set USE_STORED_OUTLINES = DATAKON_CAT;  
explain plan for  
select . . .  
alter session set SQL_TRACE=FALSE;
```



Příklad 4: přínos osnovy VII.

Výsledek trasování zpracování příkazu bez použití osnovy:

call	count	cpu	elapsed	disk	query	current	rows
Parse	100	8.82	10.25	0	4500	0	0
Execute	100	0.01	0.01	0	0	0	0
Fetch	100	0.12	0.15	44	4400	0	0
total		8.96	10.43	44	8900	0	0

Výsledek trasování zpracování příkazu s použitím osnovy:

call	count	cpu	elapsed	disk	query	current	rows
Parse	100	5.31	7.78	0	4173	0	0
Execute	100	0.00	0.01	0	0	0	0
Fetch	100	0.10	0.12	0	4400	0	0
total		5.42	7.92	0	8573	0	0

Vyhodnocení:

- V obou případech bude zvolen stejný plán provedení
- Pro poněkud složitější dotaz je už režie fáze „Parse“ v případě použití osnovy nižší.
- Čím složitější příkaz, tím bude úspora výraznější.

Používání SQL profilů I.

- Verze Oracle 10g přinesla nový typ databázového objektu – **profil SQL příkazu**.
- Zatímco dříve jsme měli aktuálně (online) k dispozici pouze časové statistiky o provedení SQL příkazů zaznamenané v operační paměti (jsou dostupné přes dynamické pohledy V\$), ve verzi 10g se statistiky o zpracování SQL příkazů ukládají do persistentního úložiště označovaného AWR (Automatic Workload Repository). Zde jsou uchovávány po dobu *N* dnů (implicitně 7).

Používání SQL profilů II.

- Pomocí nástrojů automatizovaného ladění lze s využitím informací o historii zpracování daného příkazu vytvořit jeho persistentní *profil*.
- Vedle statistik vytvořených pro tabulky, na které příkaz odkazuje, je profil daného příkazu další informací, kterou optimalizátor využije ke zpřesnění svých odhadů cen jednotlivých plánů přicházejících v úvahu.
- SQL profil nenahrazuje osnovu plánu. Osnova vede optimalizátor tvrdě k jedné volbě. Profil „jen“ poskytuje optimalizátoru informaci o historii zpracování daného příkazu - s jakými plány byl proveden, jaké byly časové charakteristiky provedení příkazu apod. Optimalizátor tuto informaci použije při stavbě nového plánu.

Závěr

- Vedle kvality výsledku rozhodování optimalizátoru při volbě optimálního plánu provedení příkazu je též důležitou otázkou vytvoření optimálních podmínek pro ono rozhodování.
- Problémem může být nejen to, za jak dlouho a s jakým čerpáním sdílených zdrojů bude vlastní příkaz posléze proveden, ale i to, s jakými režijními náklady optimalizátor k výsledku dospěje k volbě (sub)optimálního plánu.
- Oracle poskytuje prostředky, pomocí nichž lze režii optimalizátoru snížit.

Konec



Užitečné URL:

[http://www.Oracle.com/technology/
documentation/database10g.html](http://www.Oracle.com/technology/documentation/database10g.html)

Použitá literatura:

Courseware 10g: SQL Tuning Workshop
© Oracle corp.

Využívání osnov XVII.

Výpis osnovy Datakon z datového slovníku:

```
NO_EXPAND( @"SEL$58A6 D7F6" )
NO_SWAP_JOIN_INPUTS( @"SEL$58A6 D7F6" "ZA" @"SEL$1")
PQ_DISTRIBUTE(@"SEL$ 58A6D7F6" "ZA"@ "SEL$ 1" NONE NO NE)
USE_HASH(@ "SEL$58A6D 7F6" "ZA"@ "SEL$1")
LEADING(@ " SEL$58A6D7 F6" "ZE"@ "SEL$1" "C U"@ "SEL$1" )
NO_STAR_TRANSFORMATION(@"SEL$5 8A6D7F6" )
NO_FACT(@ " SEL$58A6D7 F6" "ZA"@ " SEL$1")
NO_FACT(@ " SEL$58A6D7 F6" "ZE"@ " SEL$1")
FULL(@"SEL $58A6D7F6" "ZA"@ "SEL $1")
FULL(@"SEL $58A6D7F6" "ZE"@ "SEL $1")
NO_REWRITE (@"SEL$58A 6D7F6" )
NO_REWRITE (@"SEL$58A 6D7F6" )
NO_REWRITE (@"SEL$58A 6D7F6" )
MERGE (@"S EL$1" )
```

Některé tipy pro volbu přístupové cesty

- FULL
- ROWID
- INDEX
- INDEX_ASC
- INDEX_COMBINE
- HASH

Některé tipy pro přeformulování dotazu

- USE_CONCAT
- NO_EXPAND
- REWRITE
- [NO_]REWRITE
- EXPAND_GSET_TO_UNION
- [NO_]MERGE
- STAR_TRANSFORMATION
- [NO_]FACT

Některé tipy pro volbu pořadí tabulek

- STAR
- ORDERED

Některé tipy pro algoritmus spojení

- USE_NL
- USE_MERGE
- USE_HASH
- HASH_AJ
- MERGE_AJ
- NL_AJ
- HASH_SJ
- MERGE_SJ
- NL_SJ