UNIVERZITA KOMENSKÉHO

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Katedra Informatiky

DIPLOMOVÁ PRÁCA

Meno: Branislav Belas

Odbor: Programové a počítačové systémy

Vedúci dipl. práce: RNDr. Ján Šturc, CSc.

BRATISLAVA 2005

UNIVERZITA KOMENSKÉHO

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Katedra Informatiky

Optimalizácia dotazov nad star schémami

Meno: Branislav Belas

Odbor: Programové a počítačové systémy

Vedúci dipl. práce: RNDr. Ján Šturc, CSc.

BRATISLAVA 2005

Čestne prehlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedenej literatúry a iných zdrojov.

....

Branislav Belas

Ďakujem svojmu diplomovému vedúcemu RNDr. Jánovi Šturcovi, CSc. za podnetnú diskusiu a cenné rady pri písaní tejto diplomovej práce.

OBSAH

1 Úvod ... 7
2 Optimalizátor ... 9
2.1 Úvod do spracovania dotazu ... 9
2.2 Všeobecný popis optimalizátora ... 9
2.3 Operácie optimalizátora ... 11
2.3.1 Výber prístupu a cieľa ... 11
2.3.2 Inicializačný parameter OPTIMIZER_MODE ... 12
2.3.3 CBO štatistiky v data dictionary ... 13
2.3.4 Optimalizácia SQL dotazov pre rýchly čas odozvy ... 13
2.4 Cost-based optimalizátor ... 14
2.4.1 Prepisovač dotazov (Query Transformer) ... 14
2.4.1.1 Spájanie pohľadov ... 15
2.4.1.2 Spájanie komplexných pohľadov ... 16
2.4.2 Kalkulant (Estimator) ... 17
2.4.2.1 Selektivita ... 17
2.4.2.2 Kardinalita ... 17
3 Indexy ... 18
3.1 Typy indexov ... 18
3.1.1 B*-tree indexy ... 18
3.1.2 Bitmapové indexy ... 19
4 Základné prístupové metódy ... 21
4.1 ROWID ... 21
4.2 Full table prehľadávanie ... 21
4.3 Indexové prehľadávanie ... 21
4.4 Fast full indexové prehľadávanie ... 22
4.5 Skip scan indexové prehľadávanie ... 22

5 Triedenia 23
6 Spojenia 24
6.1 Typy spojení 25
6.1.1 Spájanie tabuliek pomocou Nested loop 25
6.1.2 Sort-merge spojenie 26
6.1.3 Hash spojenie 28
7 Hinty 29
7.1 Ovplyvňovanie optimalizátora pomocou hintov 29
7.1.1 Definovanie kompletnej množiny hintov pre dotaz 30
7.2 Rozdelenie hintov 31
7.2.1 Hinty pre definovanie prístupu a cieľa optimalizácie 31
7.2.2 Hinty na definovanie metódy prístupu k tabuľke 33
7.2.3 Hinty pre transformáciu dotazu 38
7.2.4 Hinty na definovanie poradia spájania tabuliek 41
7.2.5 Hinty na definovanie metód spojenia tabuliek 42
7.2.6 Hinty na paralelné vykonávanie dotazu 45
8 Optimalizácia star schém 47
8.1 Technika Nested loop 49
8.2 STAR hint 50
8.3 Star transformácia 51
8.4 Spájacie bitmapové indexy 53
9 Záver 56
10 Použitá literatúra 57

1 Úvod

Na tému optimalizácia dotazov bolo v uplynulých desiatkach rokov napísaných nespočet článkov a stovky kníh. Je to široká oblasť, ktorú možno skúmať teoreticky, ako aj na reálnych databázových systémoch v praxi. Ak by sme sa chceli v diplomovej práci venovať optimalizácii dotazu všeobecne, museli by sme byť buď veľmi stručný, alebo by práca presahovala stovky strán. Preto sme sa sústredili na optimalizáciu dotazov nad star schémami v prostredí databázového systému Oracle9i. Na konkrétnom príklade ukazujeme rôzne efektívne metódy optimalizácie typického star dotazu z prostredia dátového skladu.
Optimalizácia dotazov je jednou z každodenných úloh databázových administrátorov alebo skôr databázových vývojárov. Pri transformácii dát, ktorá je súčasťou spracovania dátového skladu, sa vývojár stretáva s úskaliami optimalizácie dotazov nad star schémami. Práve star schéma je totiž jedným zo základných dizajnov, ktoré sa v dátových skladoch používajú. Obrovská mohutnosť dnešných databáz, neuveriteľné množstvo spracúvaných informácií, to všetko umocňuje význam optimalizácie dotazov. Pre databázových odborníkov je to však výzva a optimalizácia sa môže stať jednou z mála úloh, kde môžu popri každodennej rutine naplno využiť svoju kreativitu a inteligenciu.
Moderné databázové systémy im dnes dovoľujú takmer úplnú kontrolu nad spôsobom vykonania konkrétneho dotazu. Napríklad pomocou tzv. hintov, pridaných do kódu dotazu je môžné ovplyvniť poradie spracovaných tabuliek, určiť spôsob prístupu ku každej z nich ako aj jednotlivé metódy spojenia tabuliek v dotaze. Databázové systémy takto ponúkajú silný nástroj, ktorý v rukách skúseného vývojára vyrieši veľa netriviálnych problémov pri spracovaní dát.
V úvodných kapitolách práce popisujeme detailne činnosť optimalizátora pri spracovaní dotazu. Patria sem aj operácie, ktoré optimalizátor vykonáva automaticky a programátor ich zväčša nemôže nijak ovplyvniť. Napriek tomu je pre efektívnu optimalizáciu dôležité vedieť, aké operácie optimalizátor vykonáva samostatne.
Ďalšie kapitoly sú venované základným metódam prístupov k tabuľke a technikám spájania tabuliek. Popisujeme základné typy indexov ako aj väčšinu možných prístupových ciest, ktorými optimalizátor prehľadáva tabuľky.
V 7. kapitole sa podrobne venujeme najdôležitejším hintom. Ich použitie sa pri dobre fungujúcom optimalizátore dotazov môže zdať zbytočné, avšak v praxi sa využívajú veľmi často. Jednou z príčin je to, že cost-based optimalizátor potrebuje pre svoje efektívne

fungovanie štatistiky o objektoch. Napríklad v tzv. stagingovej časti dátového skladu sa používajú často dočasné tabuľky na transformáciu dát. Tieto sa málokedy analyzujú kôli nadbytočnej réžii. A v takýchto prípadoch nám hinty dovoľujú vykonať časť práce za optimalizátor a poradiť mu správnu metódu, ako dotaz nad takouto tabuľkou spracovať. Aj pri existujúcich štatistikách sa však môže stať, že optimalizátor nedokáže zvoliť efektívny plán vykonania. Ak má programátor detailné informácie o dátach v tabuľke, dokáže pomocou hintov vytvoriť plán vykonania dotazu, ktorý je oveľa efektívnejší, ako plán vygenerovaný optimalizátorom.

Dátový sklad je súbor technológií na podporu rozhodovania, ktorých cieľom je umožniť vedúcim pracovníkom, manažérom, analytikom a pod. robiť lepšie a rýchlejšie rozhodnutia. Jedným zo základných dizajnov v dátových skladoch je star schéma. Optimalizácii dotazov nad touto štruktúrou sa podrobne venujeme v 8. kapitole. Na príklade konkrétnej star schémy popisujeme viaceré spôsoby jej optimalizácie a porovnávame výsledky. Pokusná star schéma príkladu má 3 dimenzie a faktovú tabuľku s 10 mil. riadkami. Výsledky behov dotazov boli namerané v čistom databázovom prostredí bez vedľajších procesov, ktoré by mohli ovplyniť výsledky.

2 Optimalizátor

V tejto kapitole sa pokúsime popísať, ako optimalizátor spracúva dotazy, jeho optimalizačné metódy a proces výberu konkrétneho plánu spracovania SQL dotazu.

2.1 Úvod do spracovania dotazu

Optimalizátor používa tieto komponenty pri spracovaní SQL dotazu:

Parser – syntaktická aj sémantická analýza dotazu

Optimalizátor používa buď metódy založené na cene dotazu (cost-based optimalizátor) alebo tzv. metódu pravidiel (Rule-based optimalizátor). Jeden zo zvolených prístupov nám vyprodukuje správny a efektívny spôsob vykonania dotazu.

Generátor kódu (Row Source Generator) pripraví plán vykonania dotazu na základe optimálneho plánu, ktorý vyprodukoval optimalizátor

SQL Execution Engine – generuje výsledky pomocou plánu vykonania dotazu

2.2 Všeobecný popis optimalizátora

Optimalizátor ma za úlohu vygenerovať najefektívnejší plán, ako vykonať SQL dotaz. Pracuje s viacerými parametrami, ktoré môžu ovplyvniť spôsob vykonania SQL dotazu. Táto analýza je veľmi dôležitý krok vo vykonávaní dotazu, lebo môže znamenať niekoľkonásobné zrýchlenie dotazu. Dokonca niekedy môže znamenať rozdiel medzi vykonateľným dotazom a dotazom, ktorý vďaka zlému plánu nie je možné vykonať.
SQL dotaz môže používať mnoho prístupových ciest a metód spojenia, ako napr.:

- FULL TABLE SCAN
- INDEX SCAN
- NESTED LOOP
- HASH JOIN

Optimalizátor Oracle zahŕňa dva rôzne prístupy k optimalizácii. Jedným je cost-based optimalizátor (CBO) a druhým rule-based optimalizátor (RBO). Nové verzie optimalizátora zdokonalujú cost-based prístup a rule-based optimalizácia sa tu nachádza hlavne z dôvodu spätnej kompaibility.
Výber prístupu optimalizácie môžu ovplyvniť viaceré faktory. Vypočítavaním cost-based štatistík možno uprednostniť CBO alebo naopak manuálnym hintom v kóde dotazu prikážeme použitie RBO. Ďalším faktorom môže byť potreba aplikácie vrátiť rýchlo výsledok dotazu alebo naopak optimalizácia na väčšiu priepustnosť.
Niekedy má tvorca aplikácie alebo dizajnér lepší prehľad o údajoch uložených v databáze ako samotný optimalizátor. Ak nemá cost-based optimalizátor dosť aktuálne štatistické dáta o objektoch, môže dizajnér použiť tzv. hinty. Tie umožňujú detailne popísať prístupové cesty k objektom a spôsob ich vzájomného spojenia. Hinty teda dovoľujú dizajnérovi dotazu vytvoriť úplne vlastný plán spracovania dotazu a mať tak kontrolu nad celým postupom vykonania.

Viacero noviniek v databáze Oracle9i potrebuje nevyhnutne cost-based optimalizátor:

Particiované tabuľky a indexy
Indexovo-organizivané tabuľky
Reverse key indexy
Function-based indexy
Parameter SAMPLE príkazu SELECT
Paralelné dotazy a paralelné DML príkazy
Hviezdicové (star) transformácie a hviezdicové spojenia
Prepisovanie dotazu s materializovaným pohľadom
Hash spojenia
Bitmapové indexy a bitmapové indexy spojenia
Index skip prehľadávanie

Použitie akejkoľvek z týchto vymoženostní aktivuje CBO, aj keď je mód optimalizátora nastavený na RULE.

2.3 Operácie optimalizátora

Pri spracovaní každého SQL dotazu vykonáva optimalizátor viacero oprácií:

Vyhodnocovanie výrazov a podmienok - prvým krokom optimalizátora je vyhodnotiť čím najviac výrazov a podmienok obsahujúcich konštanty.

Transformácia dotazov - náročnejšie SQL dotazy, napr. obsahujúce vnorené poddotazy a pohľady, prepisuje optimalizátor na ekvivalentný dotaz zo spojením.

Výber správneho prístupu pre optimalizátor - optimalizátor si vyberie medzi cost-based optimalizáciou a rule-based optimalizáciou. Ďalej rozhodne, aký cieľ použije pri vyhodnocovaní dotazu.

Výber prístupových ciest - k dátam v tabuľkách je možné pristúpiť viacerými spôsobmi. Pre každú tabuľku v dotaze preto optimalizátor vyberie efektívnu prístupovú cestu ktorou bude získavať dáta.

Výber poradia spojení - spájanie viac ako troch tabuliek je možné vykonať v rozdielnom poradí. Optimalizátor vyberie pár tabuliek, ktorý sa spojí ako prvý, a potom tabuľku, ktorá sa pripojí k výsledku. Spojenie tabuliek je vždy binárna operácia. Pri spájaní viac ako dvoch tabuliek sa spoja najprv prvé dve a výsledná množina riadkov sa spája s ďalšou tabuľkou.

Výber metódy spojenia - pre každé spojenie vyberie optimalizátor metódu, akou spojenie vykoná.

2.3.1 Výber prístupu a cieľa

CBO optimalizátor je štandardne nastavený tak, že jeho cieľom je najväčší prietok dát. To znamená spracovať všetky riadky dotazu, použitím najmenšieho množstva zdrojov. Čiže optimalizátor predpokladá, že ako výsledok potrebujete celú množinu riadkov a nestačí vám prvých pár.

Optimalizátor môže mať ako cieľ nastavený aj minimálny čas odozvy. Vtedy zvolí najlepší prístup, ktorým za použitia minimálneho množstva zdrojov spracuje prvý riadok dotazu.
Plán spracovania dotazu sa môže líšiť podľa zvoleného cieľa. Optimalizácia na prietok dát má často za následok zvolenie full-table prístup k tabuľke namiesto prístupu cez indexy. Ak je naopak cieľom minimálny čas odozvy, optimalizátor zvolí spojenie pomocou nested-loop.
Majme napríklad dotaz, ktorý sa dá vykonať pomocou nested-loop spojenia aj sort-merge spojenia. Spojenie sort-merge spojí všetky riadky zrejme rýchlejšie, ako neste-loop. Prvý riadok zase spracuje rýchlejšie nested-loop.

Pri výbere prístupu a cieľa pre SQL dotaz je optimalizátor ovplyvňený týmito faktormi:

- inicializačný parameter OPTIMIZER_MODE
- štatistiky CBO optimalizátora v Data Dictionary
- SQL hinty na zmenu cieľa

2.3.2 Inicializačný parameter OPTIMIZER_MODE

Tento parameter nastavuje štandardné správanie sa optimalizátora pri výbere cieľa pre SQL dotaz. Možné nastavenia sú:

CHOOSE – optimalizátor automaticky vyberie medzi cost-based prístupom a rule-based prístupom, podľa toho, či existujú štatistiky pre spracovávaný objekt. CHOOSE je štandardné nastavenie. Ak sa v data dictionary nachádzajú štatistiky aspoň k jednej z tabuliek ku ktorým sa pristupuje, optimalizátor zvolí cost-based prístup a cieľom je najväčšia priepustnosť.
Ak data dictionary obsahuje len časť štatistík, optimalizátor zvolí znovu cost-based prístup a chýbajúce štatistiky musia byť odhadnuté. To môže mať za následok výber neefektívneho plánu spracovania, čo sa negatívne prejaví na výkone spracovania. Ak data dictionary neobsahuje žiadne štatistiky o objektoch, ktoré spracováva dotaz, optimalizátor vyberie rule-based prístup.

ALL_ROWS – tento parameter prinúti optimalizátor použiť cost-based prístup a cieľom bude najväčšia prispustnosť. Tento prístup a cieľ je použitý aj v prípade, že neexistujú štatistiky k objektom.

FIRST_ROWS_n – bez ohľadu, či existujú štatistiky k objektom alebo nie, optimalizátor zvolí cost-based prístup. Cieľom je minimálny čas odozvy, za ktorý dotaz spracuje prvých n riadkov. Číslo n môže byť: 1, 10, 100 alebo 1000.

FIRST_ROWS – optimalizátor použije spojenie cost-based prístupu a heuristík, aby čím najrýchlejšie spracoval prvých pár riadkov dotazu. Tento parameter môže mať za následok vygenerovanie veľmi neefektívneho prístupového plánu a nachádza sa tu len kôli spätnej kompatibilite.

RULE – optimalizátor použije rule-based prístup.

V prípade, že optimalizátor používa cost-based prístup a chýbajú mu štatistiky pre nejaký objekt v dotaze, musí odhadnúť potrebné štatistiky pomocou vnútorných informácií o objekte. Napríklad štatistické informácie ako počet riadkov tabuľky sa dájú odhadnúť pomocou počtu alokovaných blokov pre danú tabuľku, atď.

2.3.3 CBO štatistiky v data dictionary

CBO používa štatistiky, ktoré su uložené v data dictionary. Na zhromaždovanie štatistík sa používa balík DBMS_STATS alebo príkaz ANALYZE.
Štatistiky poskytujú CBO rôzne informácie o unikátnosti údajov a ich ditribúcií. Pomocou týchto informácií môže optimalizátor výpočítať ceny plánov spracovania s veľkou presnosťou. Potom dokáže CBO zvoliť efekívny plán s najmenšou cenou.

2.3.4 Optimalizácia SQL dotazov pre rýchly čas odozvy

CBO optimalizuje dotazy pre rýchlu odozvu, ak je parameter OPTIMIZER_MODE nastavený na hodnotu FIRST_ROWS_n, kde n je 1, 10, 100, alebo 1000, alebo FIRST_ROWS.

Ak je čislo *n* malé, CBO generuje plány, ktoré používajú nested loop spojenie s vyhľadávaním pomocou indexov. Pre väčšie *n* optimalizátor použije hash-join spojenie a full table prehľadávanie.

2.4 Cost-based optimalizátor

Cost-based optimalizátor má za úlohu vybrať najefektívnejší plán spracovania SQL dotazu. Rozhoduje sa na základe dostupných prístupových ciest a na základe existujúcich štatistík o objektoch (tabuľky, indexy). Rozhodovanie CBO je možné ovplyvniť tzv. SQL hintami, ktoré su uvedené v poznámke k SQL dotazu.

CBO vykonáva nasledujúce kroky:

1. Optimalizátor generuje viacero potencionálnych plánov spracovania na základe dostupných prísupových ciest a SQL hintov.
2. Určí cenu každého plánu pomocou štatistík uložených v data dictionary. Optimalizátor zohľadňuje pri určovaní ceny dotazu aj voľnú pamäť servera, momentálne zaťaženie CPU a zložitosť I/O operácií.
3. Optimalizátor porovná ceny vygenerovaných plánov a vyberie plán s najmenšou cenou.

Komponenty cost-based optimalizátora

CBO sa skladá z nasledujúcich komponentov:

- Prepisovač dotazov (Query Transformer)
- Ohodnocovač dotazov (Estimator)
- Generátor plánov (Plan Generator)

2.4.1 Prepisovač dotazov (Query Transformer)

Vstupom pre prepisovač je dotaz, ktorý dostane od Parsera. Hlavnou úlohou prepisovača je zistiť, či je možné prepísať dotaz na taký, ktorý sa bude dať vykonať lepším plánom. Prepisovač má k dispozícii štyri techniky transformovania dotazu:

- spájanie pohľadov
- pretláčanie predikátov
- rozbaľovanie poddotazov

- prepisovanie dotazov, ktoré obsahujú materializovaný pohľad

Prepisovač môže použiť akúkoľvek kombináciu týchto techník na prepísanie dotazu.

2.4.1.1 Spájanie pohľadov

Spájanie pohľadov je najjednoduchšia forma transformácie dotazov. Dá sa aplikovať na dotazy, ktoré obsahujú pohľad. Často je možné odstrániť odkaz na pohľad spojením definície pohľadu s dotazom.

Príklad dotazu s pohľadom:

```
CREATE VIEW TEST_VIEW AS
SELECT ENAME, DNAME, SAL FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO;
SELECT ENAME, DNAME FROM TEST_VIEW WHERE SAL > 10000;
```

Bez SQL transformácií by sme museli pospájať všetky riadky tabuľky EMP so všetkými riadkami tabuľky DEPT. Až potom by sme odfiltrovali riadky spĺňajúce podmienku SAL > 10000.

Použitím metódy spájania pohľadov, môžeme tento dotaz transformovať takto:

```
SELECT ENAME, DNAME FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND E.SAL > 10000;
```

Pri vykonávaní transformovaného dotazu sa môže aplikovať predikát `SAL > 10000` ešte pred spájaním tabuliek EMP a DEPT. Transformáciou sme teda získali podstatné zvýšenie výkonu, lebo spájame výrazne menšie množstvo dát. SQL transformácie prinášajú veľké zrýchlenie už pri jednoduchých dotazoch. Na strome algebraického výrazu znázorňuje tento proces pretláčanie selekcií, čo najbližšie k listom.

2.4.1.2 Spájanie komplexných pohľadov

Na rozdiel od jednoduchých dotazov, pohľady, ktoré obsahujú GROUP BY alebo DISTINCT nie je až také ľahké spojiť. Optimalizátor preto poskytuje viacero techník, ako spojiť aj zložitejšie pohľady.

Zoberme si ako príklad pohľad, ktorý ráta priemerú mzdu pre každé oddelenie:

```
CREATE VIEW AVG_SAL_VIEW AS
SELECT DEPTNO, AVG(SAL) AVG_SAL_DEPT FROM EMP
GROUP BY DEPTNO
```

Dotaz, ktorý hľadá priemernú mzdu pre každé oddelenie v Oaklande:

```
SELECT DEPT.NAME, AVG_SAL_DEPT
FROM DEPT, AVG_SAL_VIEW
WHERE DEPT.DEPTNO = AVG_SAL_VIEW.DEPTNO
AND DEPT.LOC = 'OAKLAND'
```

Pohľad a dotaz môžeme pretransformovať nasledovne:

```
SELECT DEPT.NAME, AVG(SAL)
FROM DEPT, EMP
WHERE DEPT.DEPTNO = EMP.DEPTNO
AND DEPT.LOC = 'OAKLAND'
GROUP BY DEPT.ROWID, DEPT.NAME
```

Výsledkom je zase podstatné zvýšenie výkonu dotazu. Namiesto toho, aby sme vykonali operáciu GROUP BY na celej tabuľke EMP, transformovaný dotaz spojí tabuľku a aplikuje predikát ešte pred náročnou operáciou GROUP BY. Čím selektívnejšia podmienka (v našom prípade DEPT.LOC = 'OAKLAND') sa presunie pred operáciu GROP BY, tým je pretransformovaný dotaz efektívnejší oproti pôvodnému.

2.4.2 Kalkulant (Estimator)

Oceňovač alebo kalkulant má za úlohu určiť hodnoty troch základných mier pre dotaz, ako napríklad selektivita, kardinalita alebo cena. Tieto miery sú od seba závislé, t.j. jedna sa vypočíta z druhej. Konečným cieľom pre oceňovač je určiť celkovú cenu daného plánu. Používa pritom štatistiky objektov, ktoré sú obsiahnuté v dotaze. Kvalita štatistík ovplyvňuje presnosť odhadu jednotlivých mier.

2.4.2.1 Selektivita

Selektivita predstavuje pomer nejakej podmonižiny riadkov k celej množine riadkov. Selektivita je zviazaná s určitým predikátom, napr.

```
LAST_NAME = ‘Smith‘
```

alebo s kombináciou predikátov, ako napr.

```
LAST_NAME = ‘Smith‘ AND JOB_TYPE = ‘Clerk‘
```

Predikát filtruje istú podmnožinu riadkov s celej množiny. Selektivita teda určuje aké množstvo riadokov prejde týmto filtrom. Môže nadobúdať hodnoty od 0 po 1. Selektivita 0 znamená, že žiaden riadok nevyhovuje predikátu a teda neprejde filtrom. Naopak ak je selektivita 1, filtrom prešli všetky riadky.

2.4.2.2 Kardinalita

Kardinalita je jednoducho povedané počet riadkov v nejakej množine riadkov. Napríklad kardinalita tabuľky je celkový počet jej riadkov.

3 Indexy

3.1 Typy indexov

Indexy sú databázové objekty, ktoré sú fyzicky nezavislé od dát. Môžu sa použiť na prístup k dátam, ktoré vyžaduje SQL dotaz alebo na overovanie platnosti integritných reštrikcií. Indexy môžu byť unikatné alebo neunikátne. Unikátny index nám zaručuje, že žiadne dva prvky indexu nemajú rovnakú hodnotu. Existujú aj tzv. zložené indexy, ktoré sa vytvárajú na viacerých stĺpcoch tabuľky.

Pre štandardné indexy používa Oracle B*-tree indexy, vyvážené pre rovnakú dobu prístupu. Ďalším typom indexov sú reverzné indexy, ktoré obracajú poradie bytov hodnôt v atribúte. Tie môžu byť efektívne využité pri monotónne rastúcich hodnotách stĺpca, pričom staré hodnoty sú časom vymazávané. Množinu indexov uzatvárajú bitmapové, doménové a funkčné indexy.

3.1.1 B*-tree indexy

Každý B*-tree index obsahuje koreň, ako základný štartovací bod. Podľa počtu prvkov môže obsahovať jeden alebo viac uzlov. Listy tohto stromu obsahujú samotné hodnoty a ROWID, ktoré určuje riadok v príslušnom dátovom segmente. Indexy sú vždy vyvážené, pričom sa môžu rozširovať. V niektorých situáciách sa môže stať, že výška B*-stromu zbytočne narastie. Vtedy je potrebné index zreorganizovať.

Vnútorné uzly stromu slúžia na vyhľadávanie a obsahujú

- minimálny prefix kľúča, aby bolo možné rozlíšiť dve hodnoty,
- ukazovateľ na blok, ktorý obsahuje kľúč

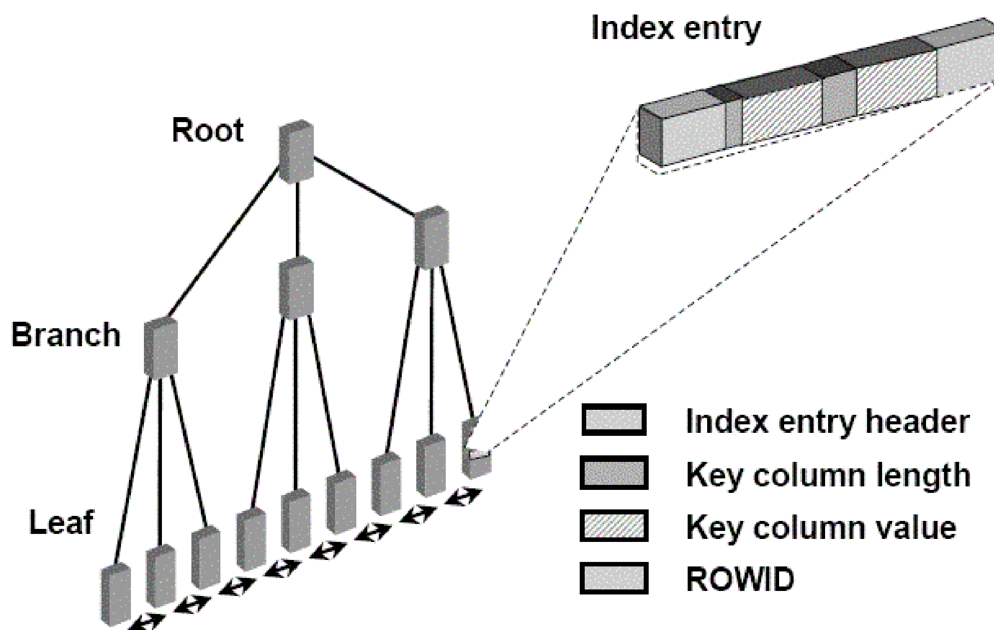
Ak majú bloky n kľúčov, potom obsahujú n+1 ukazovateľov, pričom počet kľúčov a ukazovateľov je obmädzený velkosťou blokov. Index neobsahuje odkazy na hodnoty NULL.

Akonáhle Oracle prečíta informáciu o ROWID z listov indexu, môže pristupovať fyzicky priamo k riadkom v tabuľke.

Ak sa Oracle rozhodne použiť index na prístup k dátam v tabuľke urobí nasledovné kroky:

- prehladá index na výskyt atribútov, ktoré vyžaduje SQL dotaz

- ak všetky požadované atribúty obsahuje samotný index, dáta sú získané priamo z indexu bez toho, aby bol použitý dátový segment
- ak dotaz obsahuje aj iné atribúty, zistí sa adresa príslušných riadkov z indexu a dátové bloky sú vyhľadané cez ROWID



Obr. 1: B*-tree index

3.1.2 Bitmapové indexy

Základnou úlohou indexov je poskytnúť ukazovateľ na riadky v tabuľke, ktoré obsahujú nejakú požadovanú hodnotu. Klasický b-tree index uchováva ku každej hodnote zoznam s ROWID. Tieto ROWID konkrétne určujú riadky s danou hodnotou. V bitmapovom indexe sa namiesto zoznamu s ROWID používa bitmapa pre každú jednu hodnotu stĺpca. Každý bit v bitmape tak zodpovedá jednému riadku v tabuľke. Ak je bit nastavený na 1, znamená to, že daný riadok nadobúda práve túto jednu hodnotu. Na konverziu bitu na konkrétnu hodnotu ROWID sa používa mapovacia funkcia. Ak má atribút len málo rôznych hodnôt, bitmapový

index je veľmi efektívny na priestor. Navyše z bitmapami sa dajú vykonávať rýchle bitové operácie. Ak je v podmienke WHERE dotazu kombinácia nejakých dvoch stĺpcov, nad ktorými sú postavené bitmapové indexy, riadky dotazu vyhovujúce podmienkam dostaneme veľmi efektívne. Optimalizátor pomocou jednoduchých bitových operácií AND, OR, NOT skombinuje bitové mapy a získa jednu výslednú, ktorú premapuje konkrétný zoznam riadkov (ROWID) a vygeneruje výsledok.

Bitmapové indexy sa často využívajú v prostredí dátových skladov. Zrýchľujú ad-hoc dotazy a majú nízke nároky na priestor aj vďaka dobrej kompresii. B-tree indexy nad veľkými tabuľkami majú často väčšiu veľkosť ako samotné dáta v tabuľke. Naproti tomu bitmapové indexy majú zlomok veľkosti dát v tabuľke.
Dotazy, ktoré obsahujú podmienky s predikátmi <, >, <=, >= sú efektívnejšie spracované pomocou klasických b-tree indexov.
Najefektívnejšie sú bitmapové indexy nad atribútmi, ktoré nadobúdajú málo rôznych hodnôt. Ak počet rôznych hodnôt atribútu tvorí menej ako 1% všetkých riadkov tabuľky, potom je atribút vhodným kandidátom pre bitmapový index.

4 Základné prístupové metódy

4.1 ROWID

Oracle používa ROWID na identifikáciu konkrétneho riadku v databáze. Každý riadok má svoje jedinečné ROWID, ktoré ho určuje. Všetky tabuľky majú jeden pseudostĺpec, ktorý sa volá ROWID a skladá sa zo štyroch častí.
- databázový segment
- relatívne číslo súboru, ktoré je unikátne vzhľadom na tablespace
- číslo dátového bloku
- číslo riadku v dátovom bloku

Kombináciou týchto prvkov dostávame jedinečne identifikovaný riadok v databáze. Ak zadáme vo WHERE podmienke dotazu priamo konkrétne ROWID, je to najrýchlejší spôsob prístupu ku konkrétnemu riadku v databáze.

4.2 Full table prehľadávanie

Full table prehľadávanie tabuľky znamená sekvenčné čítanie tabuľky bez použitia indexov alebo iných prostriedkov. Je to najzákladnejšia metóda prístupu k tabuľke avšak ani zdaleka nie najpomalšia. Oracle umožňuje čítanie viacerých blokov z databázového súboru v jednej vstupno-výstupnej operácii. Pri full table prehľadávaní je možné využiť paralelizmus a rozdeliť prehľadávanie na viaceré podprocesy. Viaceré zdroje uvádzajú hranicu 5% ako výhodnú pre použitie full table prehľadávania resp. prístupu k tabuľke pomocou indexu. To znamená, že ak očakávame, že náš dotaz vyberie z tabuľky viac ako 5% jej riadkov, je výhodné použiť full table prehľadávanie. Z toho je zrejmé, že prístup k tabuľke pomocou indexov je efektívny len pre veľmi malý počet vyberaných riadkov. Toto tvrdenie platí však len pre klasické indexy a nie bitmapové indexy, kde je situácia o niečo iná.

4.3 Indexové prehľadávanie

Prístup k tabuľke pomocou indexov môže veľmi výrazne ovplivniť rýchlosť SQL dotazu. Pri tejto metóde však môže byť čítané nie len z indexových blokov ale aj dátových blokov.

Tie nebývajú často sekvenčné a preto nie je možné využiť viacblokové čítanie ako pri full table prehľadávaní. Všeobecné pravidlo pre B*-tree indexy hovorí, že použitie indexu pri selekcii je výhodné, ak vyselektujeme menej ako 5% riadkov z tabuľky. Toto je samozrejme veľmi všeobecné pravidlo a skutočná hranica závisí od dát.

4.4 Fast full indexové prehľadávanie

Fast full indexové prehľadávanie je alternatíva k full table prehľadávaniu. Používa sa v prípade, že index obsahuje všetky potrebné stĺpce, ktoré selektujeme. Fast full index prístup je rýchlejší ako klasický indexový pristup lebo využíva viacblokové operácie. Tiež môže byť paralelizovaný ako full table prehľadávanie. Tento spôsob však nevracia nutne všetky riadky v zotriedenom poradí. Ak je navyše v dotaze použitý nejaký predikát, ktorý obmedzuje množinu riadkov, nedá sa táto metóda prístupu použiť.

4.5 Skip scan indexové prehľadávanie

Spojený index, ktorý sa skladá z viacerých stĺpcov je zvyčajne možné použiť len v prípade, že dotaz obsahuje podmienku na všetky stĺpce indexu. Skip scan prehľadávanie je však možné použiť aj vtedy, keď nie je známa hodnota prefixového stĺpca v indexe. Optimalizátor vtedy použije algoritmus skip scan na získanie ROWID riadkov, ktoré nepoužívajú prefixový stĺpec indexu. Skip scan algoritmus tak redukuje potrebu vytváranie indexu, ktorý sa používa len sporadicky.

5 Triedenia

Jazyk SQL obsahuje viaceré komponenty, ktoré implicitne používajú triedenie. Napr. operácie **DISTINCT**, **GROUP BY**, **UNION**, **MINUS** a **INTERSECT** vždy vyvolajú triedenie. Kedže triedenie hodnôt nejakého stĺpca tabuľky môže byť veľmi drahá operácia, je výhodné sa triedeniam vyhýbať pokiaľ je to možné. Jednou z metód, ako sa vyhnúť triedeniu je použitie indexov. Najmä spájané indexy nám môžu veľmi pomôcť pri redukovaní triedenia. Pretože sa index musí čítať sekvenčne, strácame pritom schopnosť čítať viac blokov v jednej vstupno-výstupnej operácii. Prehľadávanie tabuľky použitím index fast full nám nezaručuje, že riadky budú vyberané v správnom poradí.

6 Spojenia

Ak je za klauzulou **FROM** v dotaze uvedených viacero tabuliek, ide o dotaz so spojením. Optimalizátor používa spojenia napríklad aj v prípade poddotazov.

Príklad:

```
SELECT c.cust_last_name, c.cust_first_name,
       co.country_id, co.country_name
FROM customers c JOIN countries co
     ON (c.country_id = co.country_id) -- spájací predikát
 AND co.country_id = 'JP' -- nespájací predikát
OR c.cust_id = 205; -- jednoriadkový predikát
```

spájací predikát – predikát za klauzulou WHERE, ktorý obsahuje stĺpce dvoch tabuliek spojenia

nespájací predikát – predikát za klauzulou WHERE, ktorý obsahuje iba stĺpce z jednej tabuľky

jednoriadkový predikát – predikát rovnosti nad stĺpcom, ktorý je buď primárny kľúč alebo je typu UNIQUE. Alebo je na stĺpci postavený UNIQUE INDEX bez ďalších obmedzení. Optimalizátor v tomto prípade vie, že predikát vyberá najviac jeden riadok z tabuľky.

equijoin – najčastejší typ spojenia, pri ktorom je medzi stĺpcami spájaných tabuliek znamienko rovnosti (=)

nonequijoin – spájací predikát obsahuje nejaký iný operátor porovnania, napr. <, >, <=, >= alebo <>

6.1 Typy spojení

Operácia spojenia kombinuje riadky z dvoch tabuliek a vytvorí jednu množinu riadkov. Oracle pozná tieto typy spojenia:

- Nested loop
- Sort-Merge
- Hash
- Cluster
- Full outer spojenia

6.1.1 Spájanie tabuliek pomocou Nested loop

Pri spájaní tabuliek pomocou nested loop sa jedna tabuľka definuje ako vonkajšia (vedúca) a druhá ako vnútorná. Spojenie sa potom vykoná tak, že pre každý riadok z vedúcej tabuľky sa vyhľadajú jeho páry vo vnútornej tabuľke:

- pre každý ridok vonkajšej tabuľky
    - pre každý riadok vnútornej tabuľky
        - over, či sa riadky zhodujú

Štandardný plán vykonania pre nested loop spojenie:

```
NESTED LOOPS
      TABLE ACCESS (…) OF vonkajšia_tabuľka
      TABLE ACCESS (…) OF vnútorná_tabuľka
```

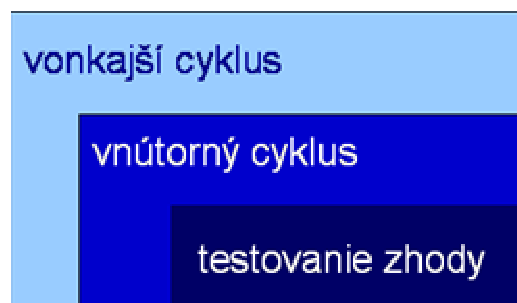
Nestep loop sa často používa v kobinácii s prístupom cez indexy. V tomto prípade bude plán vykonania nasledovný:

```
NESTED LOOPS
      TABLE ACCESS (BY ROWID) OF vonkajšia_tabuľka
            INDEX (... SCAN) OF index_vonkajšej_tab
      TABLE ACCESS (BY ROWID) OF vnútorná_tabuľka
            INDEX (RANGE SCAN) OF index_vnútornej_tab (NON-UNIQUE)
```

Vo väčšine prípadov sa na prístup k vonkajšej tabuľke používa full table prehľadávanie.

Pri nonequijoin spojeniach nie je možné použiť hash na spojenie tabuliek. V závislosti od nespájacích predikátov dotazu potom môže plán vykonania obsahovať nested loop s full table prehľadávaním vonkajšej aj vnútornej tabuľky. To znamená, že pre každý riadok vonkajšej tabuľky sa vykoná full table scan vnútornej tabuľky, čo je veľmi neefektívny spôsob spojenia.

Nested loop je možné použiť prakticky pri akomkoľvek spojení dvoch tabuliek bez ohľadu na predikát spojenia. Kôli jeho slabej výkonnosti pri spájaní väčších tabuliek sa používa len vtedy, ak nie je možné použiť hash, sort-merge alebo cluster spojenie.



Obr. 2: Schéma nested loop spojenia

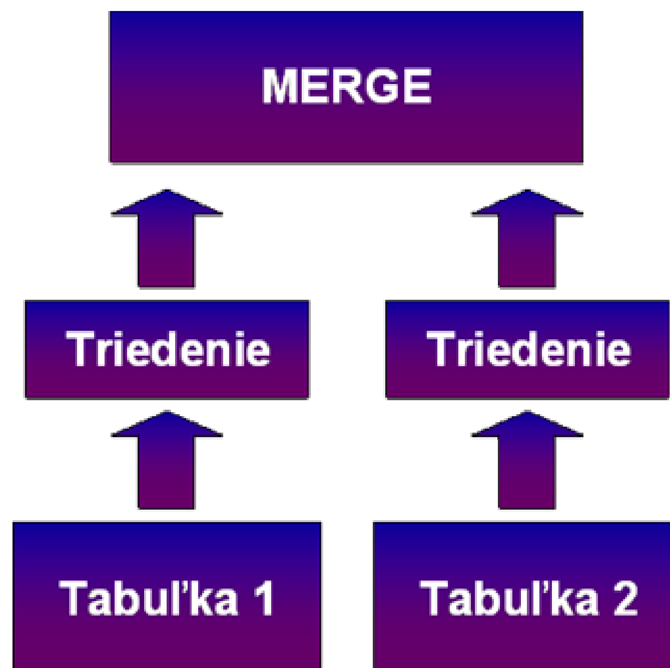
6.1.2 Sort-merge spojenie

Pri spájaní tabuliek použitím sort-merge sa obe spájané tabuľky najprv utriedia podľa stĺpcov, cez ktoré sa spájajú. Ak bolo zotriedenie niektorej z tabuliek vykonané už v predošlej operácií, tabuľka sa už nemusí znovu triediť. Práve triedenie je najzložitejšou časťou tohoto typu spájania, najmä ak ho nie je možné vykonať v pamäti. Po utriedení sa obidve tabuľky skombinujú a kontroluje sa zhoda v spájacom predikáte.

Základný plán vykonania:

```
MERGE (JOIN)
      SORT(JOIN)
            TABLE ACCESS OF tabuľkaA
      SORT(JOIN)
            TABLE ACCESS OF tabuľkaB
```

Operácie TABLE ACCESS nad oboma tabuľkami je možné vykonať použitím indexov, ak index obsahuje nejaké nespájacie predikáty. Väčšinou sa ale na prístup k tabuľkám pri tomto spojení používa full table scan. Na poradí, v akom sú tabuľky zotriedované, nezáleží. Neexistuje tu žiadna vonkajšia alebo vnútorná tabuľka. Sort-merge spojenie je výkonnejšie ako nested loop v prípade, že počet riadkov vyhovujúcich podmienke spojenia tvorí väčšinu tabuľky. V pripade malého počtu zhodných riadkov býva rýchlejší nested loop s použitím indexu.



Obr. 3: Schéma sort-merge spojenia

6.1.3 Hash spojenie

Na využitie hash spájania tabuliek je potrebný cost-based optimalizátor. Podobne ako sort-merge spojenie, hash je možný len v prípade equijoin (=) spájania. Hash vykonáva tieto kroky:

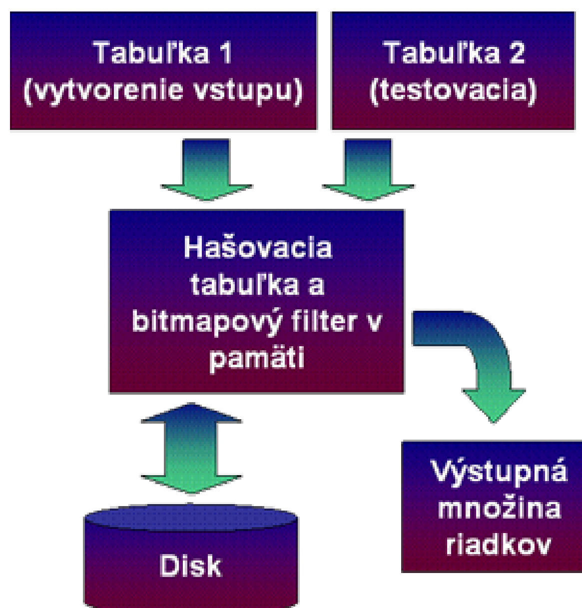
- full table scan oboch tabuliek, pričom sa každá z nich rozdelí na toľko partícií, ako je potrebné, podľa dostupnej pamäte
- na najmenšej partícii sa vytvorí hashovacia tabuľka a použije ďalšiu partíciu na overovanie hashovacej tabuľky

Všetky partície, ktoré nebolo možné umiestniť do pamäte, sú uložené na disk. Počet partícií, na ktoré boli tabuľky rozdelné závisí od veľkosti dostupnej pamäte.

Plán vykonania hash spojenia:

HASH JOIN

    TABLE ACCESS OF (…) tabuľkaA
    TABLE ACCESS OF (…) tabuľkaB



Obr. 4: Schéma hash spojenia

7 Hinty

7.1 Ovplyvňovanie optimalizátora pomocou hintov

Hinty nám dovoľujú robiť rozhodnutia, ktoré obyčajne robí za nás optimalizátor. Ako dizajnér aplikácie môžeme mať niekedy lepšie informácie o objektoch v databáze ako samotný optimalizátor. Napríklad môžeme vedieť, že konkrétny index je selektívnejší ako iný, pre niektoré dotazy. Tieto vedomosti nám umožňujú zvoliť efektívnejší plán vykonania dotazu ako optimalizátor. Hinty dovoľujú presne definovať, aký plán vykonania má optimalizátor použiť na vykonanie konkrétneho dotazu.

Použitím hintov je možné ovplyvniť:

- spôsob prístupu k optimalizácii dotazu
- cieľ cost-based optimalizátora pre SQL dotaz
- prístupovú cestu ku každej tabuľke v dotaze
- poradie spájania tabuliek
- spôsob spojenia dvoch tabuliek

Všetky hinty okrem hintu RULE majú za následok použitie cost-based optimalizátora na vykonanie dotazu. Ak nemá optimalizátor k dispozícii štatistiky, budú použité štandardné hodnoty.

Hinty ovplyvňujú optimalizáciu v bloku dotazu, v ktorom sú uvedené. Blok dotazu môže byť napr.:

- jednoduchý príkaz `SELECT`, `UPDATE alebo DELETE`
- poddotaz komplexného dotazu
- časť zloženého dotazu (napr. pomocou príkazu UNION)

Hinty sú definované ako súčasť komentára so znakom „+“. Dva rôzne tvary syntaxu sú nasledovné:

```
{DELETE|INSERT|SELECT|UPDATE} /*+ hint [text] [hint[text]]... */
```

alebo

```
{DELETE|INSERT|SELECT|UPDATE} --+ hint [text] [hint[text]]...
```

kde:

DELETE, **INSERT**, **SELECT** a **UPDATE** sú kľúčové slová, ktoré začínajú blok dotazu. Komentáre, ktoré obsahjú hinty môžu nasledovať len po týcho príkazoch. Ak je hint definovaný nesprávne alebo so syntaktickou chybou, tak ho optimalizátor ignoruje.

7.1.1 Definovanie kompletnej množiny hintov pre dotaz

V niektorých prípadoch je nutné definovať každú operáciu v dotaze pomocou hintov, aby ste získali optimálny plán vykonania. Keď optimalizujete komplexný dotaz obsahujúci viacero tabuliek a uvediete len hint INDEX pre niektorú z tabuliek, optimalizátor doplní ostatné operácie podľa svojich možností. Môže sa stať, že hintom definovaný index vôbec nebude zohladnený, lebo je nepoužiteľný v kombinácií s metódou spojenia danej tabuľky.

V nasledujúcom dotaze sa pomocou hintov definuje poradie tabuliek, v ktorom sa budú spájať, použité metódy spojenia aj prístupové cesty k tabuľkám:

```
SELECT /*+ ORDERED INDEX (b, jl_br_balances_n1) USE_NL (j b)
           USE_NL (glcc glf) USE_MERGE (gp gsb) */
 b.application_id ,
 b.set_of_books_id ,
 b.personnel_id,
 p.vendor_id Personnel,
 p.segment1 PersonnelNumber,
 p.vendor_name Name
FROM  jl_br_journals j,
      jl_br_balances b,
      gl_code_combinations glcc,
      fnd_flex_values_vl glf,
      gl_periods gp,
      gl_sets_of_books gsb,
      po_vendors p
WHERE  ...
```

7.2 Rozdelenie hintov

Hinty optimalizátora sa delia podľa [2] do nasledujúcich kategorií:

- hinty pre definovanie prístupu a cieľa optimalizácie
- hinty na definovanie metódy prístupu k tabuľke
- hinty pre transformáciu dotazu
- hinty na definovanie poradia spájania tabuliek
- hinty na definovanie metód spojenia tabuliek
- hinty pre paraelné vykonávanie dotazu

7.2.1 Hinty pre definovanie prístupu a cieľa optimalizácie

Hinty z tejto kategórie umožňujú zvoliť medzi použitím rule-based optimalizátora a cost-based optimalizátora. Pri použití cost-based optimalizátora ďalej dovoľujú výber spôsobu optimalizácie dotazu pre najväčšiu priepustnosť alebo najrýchlejšiu dobu odozvy.

Hinty:

- ALL_ROWS
- FIRST_ROWS(n)
- CHOOSE
- RULE

Ak je v SQL dotaze uvedený konkrétný prístup a cieľ optimalizácie, tak ho optimalizátor použije aj v prípade, že k objektom neexistujú štatistiky. Chýbajúce štatistiky sa nahradia preddefinovanými štandardnými hodnotami, čo môže viesť k neefektívnej optimalizácii.

ALL_ROWS

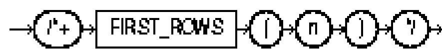
Tento hint implicitne použije cost-based optimalizátor na dotaz, ktorý optimalizuje s cieľom čo najväčšej prispustnosti dát. To zamená vykonať kompletný dotaz a vygenerovanie celéj množiny riadkov použitím minimálnych prostriedkov.

Napríklad:

```
SELECT /*+ ALL_ROWS */ empno, ename, sal, job
FROM emp
WHERE empno = 7566;
```

FIRST_ROWS(n)

Hint FIRST_ROWS(*n*) (kde *n* je kladné celé číslo) alebo FIRST_ROWS definuje optimalizáciu dotazu pre rýchlu odozvu. Hint FIRST_ROWS(n) ponúka väčšiu presnosť. Optimalizátor sa snaží vygenerovať plán, ktorý vracia prvých *n* riadkov výsledku čo najefektívnejšie.



Napríklad:

```
SELECT /*+ FIRST_ROWS(10) */ empno, ename, sal, job
FROM emp
WHERE deptno = 20;
```

V tomto príklade má každé oddelenie viacero zamestnancov. Užívateľ pomocou hintu definuje požiadavku, že chce prvých 10 zamestnancov získať čo najrýchlejšie.

CHOOSE

Hint CHOOSE je štandardným nastavením optimalizátora a nemusí byť uvedený. Oznamuje optimalizátoru, že má vybrať spôsob prístupu k optimalizácii. Optimalizátor sa na základe existencie štatistík k použitým objektom rozhodne pre spôsob prístupu. Ak existujú štatistiky aspoň k jednej tabuľke, uvedenej v dotaze, tak sa použije cost-based prístup k optimalizáii dotazu. Ak optimalizátor nemá k dispozícii štatistiky k žiadnemu objektu, použije rule-based prístup k optimalizácii dotazu.

Napríklad:

```
SELECT /*+ CHOOSE */ empno, ename, sal, job
FROM emp
WHERE empno = 7566;
```

RULE

Hint RULE definuje použitie rule-based prístupu k optimalizácii dotazu ako v tomto príklade:

```
SELECT --+ RULE
empno, ename, sal, job
FROM emp
WHERE empno = 7566;
```

7.2.2 Hinty na definovanie metódy prístupu k tabuľke

Každý z nasledujúcich hintov definuje metódu prístupu k tabuľke:

- FULL
- ROWID
- INDEX
- INDEX_ASC
- INDEX_COMBINE
- INDEX_DESC
- INDEX_FFS
- NO_INDEX

Ak zadefinujete hint pre konkrétnu metódu prístupu k tabuľke, optimalizátor túto metódu použije len v prípade, že je túto prístupovú metódu možné zvoliť. V opačnom prípade hint ignoruje.

FULL

Hint FULL prinúti optimalizátor aby použil full table prehľadávanie ako metódu prístupu k tabuľke uvedenej v parametri.

Príklad:

```
SELECT /*+ FULL(A) nepoužije index na atribúte accno */ accno, bal
FROM accounts a
WHERE accno = 7086854;
```

Optimalizátor vykoná full table prehľadávanie tabuľky `accounts` aj v prípade, že existuje index na atribúte `accno`, ktorý by mohol byť použitý.

ROWID

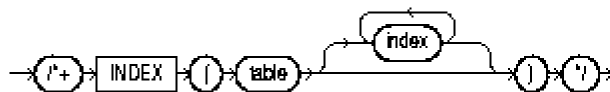
Hint `ROWID` definuje prehľadávanie tabuľky cez ROWID.

V nasledujúcom príklade vyhľadávame riadok pomocou rozsahu ROWID:

```
SELECT /*+ROWID(emp)*/ *
FROM emp
WHERE rowid > 'AAAAtkAABAAAFNTAAA' AND empno = 155;
```

INDEX

Hint `INDEX` zvolí prehľadávanie tabuľky použitím indexu, ktorý môžeme definovať ako parameter. Tento hint je možné použiť v spojení s B-tree indexom, bitmapovým indexom aj spájacým bitmapovým indexom. Pri bitmapových indexoch sa ale doporučuje použitie hintu `INDEX_COMBINE`, ktorý dokáže využiť špeciálne vlastnosti bitmapových indexov.



Ako paramaeter hintu môžeme uviesť aj viacero indexov:

- Ak je paramatrom hintu len jeden index, potom optimalizátor prehľadáva tabuľku použitím tohto indexu a neberie do úvahy full table prehľadávanie ani žiadny iný index.
- Ak je parametrom hintu viacero indexov, potom optializátor rozhoduje o použití indexu na základe ceny prehľadávania danej tabuľky týmito indexami. Môže sa tiež

rozhodnúť pre použitie viacerých indexov súčasne a zvolí prístupovú cestu (indexovú) s najnižšou cenou. Optimalizátor pritom neberie do úvahy full table prehľadávanie alebo prehľadávanie pomocou indexu, ktorý nie je uvedený v zozname ako paramater hintu.

- Ak neuvedieme ako parameter hintu žiadny index, optimalizátor vyberá vhodnú prístupovú cestu spomedzi všetkých možných indexov v tabuľke. Môže takisto zvoliť viacero indexov naraz a zlúčiť výsledky. Neberie však do úvahy full table prehľadávanie.

Napríklad nasledujúci dotaz vyberá meno, výšku a váhu všetkých mužov v nemocnici:

```
SELECT meno, vyska, vaha
FROM pacienti
WHERE pohlavie = 'm';
```

Predpokladajme, že na atribúte `pohlavie` je index a že atribút nadobúda hodnoty len `m` a `z`. Ak je porovnateľne veľa mužských pacientov v nemocnici ako ženských, tak dotaz vracie relatívne vysoké precento riadkov z celej tabuľky. V takomto prípade bude zrejme efektívnejšie full table prehľadávanie než prístup cez indexy. Ak však len malé percento pacientov nemocnice sú muži, potom by prístup použitím indexu mohol byť oveľa rýchlejši ako full table prehľadávanie. Ak neboli nad atribútom `pohlavie` rátané frekvenčné histogrami, potom sa optimalizátor nevie sám správne rozhodnúť pre indexové prehľadávanie tabuľky. Cost-based optimalizátor predpokladá, že každá hodnota má rovnakú šancu na výskyt v kaaždom riadku. Čiže pre atribút s dvomi možnými hodnotami je táto šanca na výskyt 50%. Preto cost-based optimalizátor zvolí full-table prehľadávanie a nie indexový prístup.

Ak máme informácie o hodnote atribútu, ktorá je uvedená v podmienke `WHERE`, že táto hodnota sa vyskytuje len málo percentách riadkov tabuľky, potom môžeme použiť hint `INDEX`. Tým prinútime optimalizátor, aby vykonal indexové prehľadávanie tabuľky namiesto full table prehľadávania.

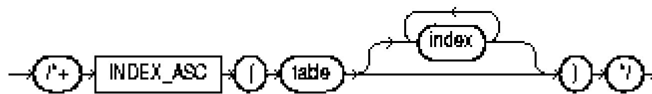
Príklad:

```
SELECT /*+ INDEX(pacienti pohlavie_index) použi index pohlavie_index
pretože v tabuľke pacienti je len málo mužských pacientov.  */
  meno, vyska, vaha
FROM pacienti
WHERE pohlavie = 'm';
```

Hint INDEX je možné použiť aj pre predikát IN-list.

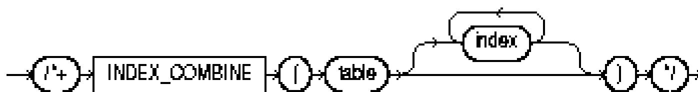
INDEX_ASC

Hint INDEX_ASC definuje indexové prehľadávanie tabuľky ako v predošlom prípade. Ak sa však používa v pláne vykonania index range scan, prehľadávanie indexových hodnôt sa vykoná vo vzostupnom (rastúcom) poradí.



INDEX_COMBINE

Hint INDEX_COMBINE určuje metódu prístupu k tabuľke použitím bimtapových indexov. Ak ako paramater neuvedieme označenie žiadneho indexu, optimalizátor berie do úvahy všetky možné kombinácie bitmapových indexov, ktoré sú k dispozícii v tabuľke. Ak uvedieme ako parameter nejaké konkrétne bitmapové indexy, potom optimalizátor zvolí nejakú efektívnu kombináciu uvedených indexov.

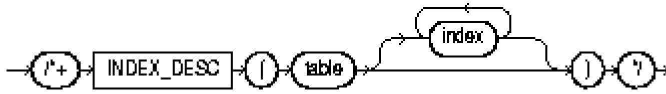


Napríklad kombinácia indexov sal_bmi a hiredate_bmi :

```
SELECT /*+INDEX_COMBINE(emp sal_bmi hiredate_bmi)*/ *
FROM emp
WHERE sal < 50000 AND hiredate < '01-JAN-1990';
```

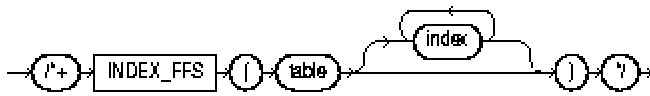
INDEX_DESC

Hint INDEX_ASC definuje indexové prehľadávanie tabuľky ako v predošlom prípade. Ak sa však používa v pláne vykonania index range scan, prehľadávanie indexových hodnôt sa vykoná v zostupnom (klesajúcom) poradí.



INDEX_FFS

Ak použijeme hint INDEX_FFS optimalizátor vykoná fast full indexové prehľadávanie danej tabuľky. Túto metódu je možné použiť v prípade, že vyberáme len atribúty, ktoré sú súčasťou indexu. Je to veľmi rýchla metóda prístupu, lebo prehľadávame len samotný index a nepristupujeme k dátam tabuľky.

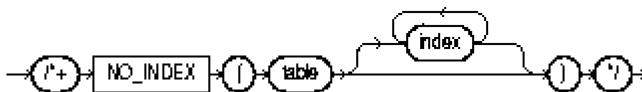


Napríklad tu je možné použiť fast full indexové prehľadávanie, lebo vyberáme len atribút empno, ktorý je súčasťou indexu emp_empno:

```
SELECT /*+INDEX_FFS(emp emp_empno)*/ empno
FROM emp
WHERE empno > 200;
```

NO_INDEX

Hint NO_INDEX vylučuje použitie jedného alebo viacerých indexov pri optimalizácii.



Ak je parametrom hintu len jeden index, potom optimalizátor neberie do úvahy pri optimalizácii len tento jediný index. Ostatné indexy, ktoré nie sú uvedené môžu byť použité v pláne vykonania. Ak je parametrom hintu viacero indexov, potom žiadny z nich nebude

uvažovaný pri optimalizácii dotazu. Ak je hint uvedený bez parametrov, potom optimalizátor neuvažuje použitie žiadneho z indexov v tabuľke.

Ak sú ako hinty jedného dotazu uvedené NO_INDEX hint aj index hint (INDEX, INDEX_ASC, INDEX_DESC, INDEX_COMBINE, or INDEX_FFS) a oba používajú rovnaký index, potom sú oba hinty ignorované a optimalizátor berie do úvahy použitie indexov ako keby nebol v doatze uvedený žiadny hint.

Napríklad:

```
SELECT /*+NO_INDEX(emp emp_empno)*/ empno
FROM emp
WHERE empno > 200;
```

7.2.3 Hinty pre transformáciu dotazu

Nasledujúce hinty určujú použitie rôznich metód transformácie dotazov.

- USE_CONCAT
- NO_EXPAND
- REWRITE
- MERGE
- STAR_TRANSFORMATION
- FACT
- NO_FACT

USE_CONCAT

Ak je vo WHERE podmienke dotazu uvedených viacero podmienok spojených pomocou OR, potom použitím hintu USE_CONCAT optimalizátor prepíše dotaz na viacero dotazov zjednotených použitím UNION ALL. Štandardne sa táto transformácia vykonáva len v prípade, že cena zloženého dotazu je menšia, ako cena dotazu bez transformácie.

Čiže použitím hintu sa nasledujúci dotaz

```
SELECT /*+USE_CONCAT*/ *
FROM emp
WHERE empno > 50 OR sal < 50000;
```

pretransformuje na zložený dotaz

```
SELECT  *
FROM emp
WHERE empno > 50;
CONCATENATION
SELECT  *
FROM emp
WHERE sal < 50000;
```

NO_EXPAND

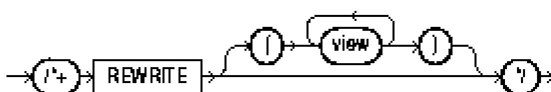
Hintom NO_EXPAND prinútime optimalizátor, aby vykonal namiesto OR expanzie (ako v predošlom príklade) IN-list iteráciu.

Napríklad:

```
SELECT /*+NO_EXPAND*/ *
FROM emp
WHERE empno = 50 OR empno = 100;
```

REWRITE

Pri použití hintu REWRITE sa optimalizátor pokúsi prepísať dotaz použitím materializovaných pohľadov. Hint sa dá použiť s pohľadmi ako parametrami alebo bez nich. Ak je ako parameter uvedený pohľad, tak sa optimalizátor pokúsi prepísať dotaz použitím tohto pohľadu bez ohľadu na cenu.



MERGE

Hint MERGE dovoľuje spájanie pohľadov s dotazmi. Ak dotaz pohľadu obsahuje GROUP BY príkaz alebo DISTINCT, potom môže optimalizátor zlúčiť tento pohľad z volajúcim dotazom. Spájanie dotazov nie je cost-based operácia, preto je nutné uviesť hint vo volajúcom dotaze. Ak hint nie je uvedený, optimalizátor zvolí iný prístup.

Napríklad:

```
SELECT /*+MERGE(v)*/ e1.ename, e1.sal, v.avg_sal
FROM emp e1,
  (SELECT deptno, avg(sal) avg_sal
   FROM emp e2
   GROUP BY deptno) v
WHERE e1.deptno = v.deptno AND e1.sal > v.avg_sal;
```

STAR_TRANSFORMATION

Použitie star transformácie ako techhniky na optimalizáciu dotazov nas star schémemi detailne popisujeme v samostatnej kapitole. Hint STAR_TRANSFORMATION dáva inštrukcie optimalizátoru, aby použil najlepší možný plán s použitím star transformácie. Aj keď je hint uvedený správne, môže sa stať, že plán vykonania nebude obsahovať star transformáciu. Optimalizátor zohľadňuje prepísanie dotazu na poddotazy pre dimenzie. Ak je porovnávaná cena transformovaného dotazu väčšia ako cena netransformovaného, star transformácia sa nepoužije.

FACT

Hint FACT je používaný pri optimalizácii star schém na určenie faktovej tabuľky.

NO_FACT

Hint `NO_FACT` naopak učuje tauľku, ktorá by nemala byť ohľadňovaná ako faktová pri star transformácii dotazu.

7.2.4 Hinty na definovanie poradia spájania tabuliek

Tieto hinty je možné použiť na definovanie poradia, v ktorom sa budú spájať tabuľky dotazu:

- ORDERED
- STAR

ORDERED

Veľmi dôležitý hint `ORDERED`, ktorý povie optimalizátoru, aby spájal tabuľky v takom poradí, v akom sú uvedené za príkazom `FROM`.

Ak nepoužijeme hint `ORDERED`, poradie spojenia tabuliek v dotaze určí optimalizátor sám na základe svojich informácii. Ak máme detailné informácie o počtoch riadkov, ktoré sa z daných tabuliek selektujú, môžeme vedieť lepšie odhadnúť vnútornú a vonkajšiu tabuľku spojenia lepšie ako optimalizátor.

STAR

Hint `STAR` definuje použitie STAR plánu vykonania, ak je to možné. Tento postup na rozdiel od star transformácie spracuváva najväčšiu tabuľku ako poslednú. Spája ju pomocou nested-loop spojenia a použitím zložených indexov s ostatnými tabuľkami (dimenziami). Tento hint je možné použiť len ak sú spájané aspoň tri tabuľky a na najväčšej z nich je zložený index obsahujúci aspoň tri atribúty. Optimalizátor berie do úvahy aj rôzne kombinácie spojení malých tabuliek.

Efektívny plán vykonania tejto transformácie je možné docieliť analyzovaním tabuliek, ktoré sa spájajú. Na základe takto získaných štatistík má optimalizátor možnosť vygenerovať dobrý plán vykonania.

7.2.5 Hinty na definovanie metód spojenia tabuliek

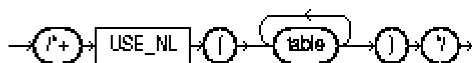
Nasledujúce hinty sa používajú na výber konkrétnej metódy spojenia dvoch tabuliek v dotaze:

- USE_NL
- USE_MERGE
- USE_HASH
- DRIVING_SITE
- LEADING
- HASH_AJ, MERGE_AJ, and NL_AJ
- HASH_SJ, MERGE_SJ, and NL_SJ

Použitie hintov `USE_NL` a `USE_HASH` sa odporúča v kombinácii s hintom `ORDERED`, ktorý definuje presné poradie spájaných tabuliek. Optimalizátor použije tieto hinty vždy keď je uvedená tabuľka tzv. vnútorná tabuľka spojenia. Ak je uvedená tabuľka vonkajšou tauľkou spojenia, hint bude ignorovaný.

USE_NL

Použitím tohto hintu je možné donútiť optimalizátor aby pripájal danú tabuľku k inej použitím nested-loop spojenia. Tabuľka uvedená ako parameter je vnútornou tabuľkou spojenia.



Nasledujúci dotaz spája dve tabuľky `accounts` a `customers`:

```
SELECT accounts.balance, customers.last_name, customers.first_name
FROM accounts, customers
WHERE accounts.custno = customers.custno;
```

Ako sme popísali vyššie, optimalizátor je štandardne nastavený na optimalizáciu dotazov pre najlepšiu priepustnosť dát. Preto si vyberie medzi spojením tabuliek pomocou nested loop spojenia, sort-merge spojenia alebo použije hash spojenie. Vybere nejlepšiu metódu, ktorá vygeneruje všetky riadky výsledku čím najrýchlejšie.

Ak však chceme zoptimalizovať dotaz tak, aby vracal prvé riadky výsledku čím najrýchlejšie a nezaujíma nás celý výsledok, môže použit hint. Prinútime optimalizátor, aby pripojil tabuľku pomocou nested loop spojenia, ktoré zrejme vygeneruje prvých pár riadkov dotazu rýclejšie ako napr. hash spojenie alebo sort-merge spojenie.

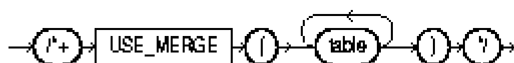
V nasledujúcom príklade sa použitím nested loop spojenia pripája tabuľka `customers`, ako vnútorná tabuľka spojenia:

```
SELECT /*+ ORDERED USE_NL(customers) to get first row faster */
accounts.balance, customers.last_name, customers.first_name
FROM accounts, customers
WHERE accounts.custno = customers.custno;
```

Nested loop spojenie dokáže vygenerovať prvých pár riadkov výsledku obyčajne rýhlejšie ako napr. sort-merge spojenie. Metóda nested loop vie vrátiť prvé riadky už po načítaní jedného riadku z vonkajšej tabuľky a nájdení ekvivalentného riadku vo vnútornej tabuľke. Pri použití metódy sort-merge sa najprv prehľadávajú a zoraďujú obe spájané tabuľky a až potom je možné vygenerovať prvé riadky výsledku.

USE_MERGE

Ak použijeme hint `USE_MERGE`, optimalizátor pripojí danú tabuľku použitím metódy sort-merge.

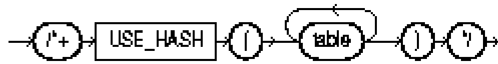


V príklade sú ako parameter uvedené dve tabuľky, ktoré budú spojené použitím sort-merge spojenia:

```
SELECT /*+USE_MERGE(emp dept)*/ *
FROM emp, dept
WHERE emp.deptno = dept.deptno;
```

USE_HASH

Hint USE_HASH definuje ako metódu spojenia tabuliek hash spojenie.



Napríklad:

```
SELECT /*+use_hash(emp dept)*/ *
FROM emp, dept
WHERE emp.deptno = dept.deptno;
```

LEADING

Hint LEADING slúži na definovanie vedúcej tabuľky pri spájaní dvoch tabuliek. Tabuľka uvedená v parametri hintu bude prehľadávaná ako prvá, tzv. vonkajšia tabuľka.

Ak je v dotaze uvedených viacero hintov LEADING, všetky budú optimalizátorom ignorované. Hint ORDERED má väčšiu prioritu ako LEADING hint.

HASH_AJ, MERGE_AJ a NL_AJ

Hinty MERGE_AJ, HASH_AJ, a NL_AJ sa vkladajú do poddotazu, ktorý nasleduje za klauzulou NOT IN. MERGE_AJ používa sort-merge anti-join, HASH_AJ používa hash anti-join a NL_AJ používa nested loop anti-join. Operácia anti-join sa používa na výpočet rozdielu dvoch množín, napr. pri použití NOT IN predikátu.

HASH_SJ, MERGE_SJ a NL_SJ

Hinty HASH_SJ, MERGE_SJ a NL_SJ sa vkladajú do poddotazu za klauzulou EXISTS. HASH_SJ používa na výpočet výsledku hash semi-join, MERGE_SJ používa sort merge semi-join a NL_SJ používa nested loop semi-join.

Napríklad:

```
SELECT * FROM dept
WHERE exists (SELECT /*+HASH_SJ*/ *
  FROM emp
  WHERE emp.deptno = dept.deptno
        AND sal > 200000);
```

Použitie hintu prinúti optimalizátor prepísať poddotaz na špeciálne spojenie medzi oboma tabuľkami v dotaze. Toto špeciálne spojenie funguje tak, že aj keď existuje v poddotaze viacero riadkov, ktoré sú ekvivalentné s riadkom hlavného dotazu, do výsledku sa generuje len jeden riadok.

Poddotaz je možné prepísať na semi-join len v niektorých prípadoch a existujú isté obmädzenia:

- v poddotaze môže byť iba jedna tabuľka
- vonkajší blok dotazu nesmie byť sám poddotazom
- poddotaz musí byt spojený s vonkajším dotazom pomocou ekvivalencie
- v poddotaze nesmie byť použitý príkaz GROUP BY, CONNECT BY alebo ROWNUM

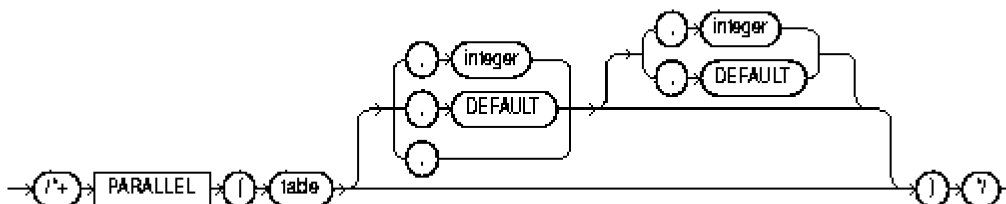
7.2.6 Hinty na paralelné vykonávanie dotazu

Nasledujúce hinty slúžia na ovplyvňovanie paralelného behu dotazu:

- PARALLEL
- NOPARALLEL

PARALLEL

Hint PARALLEL slúži na definovanie počtu súbežných procesov, ktoré budú vykonávať daný dotaz. Hint je možné použiť v kombinácii s príkazmi SELECT, INSERT, UPDATE alebo DELETE.



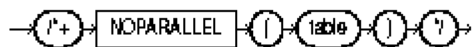
Hintu je možné ako parameter zadať stupeň paralelizmu a spôsob, akým sa budú procesy deliť na inštancie.

V nasledujúcom príklade definuje hint paralelné prehľadávanie tabuľky použitím piatich súbežných procesov:

```
SELECT /*+ FULL(scott_emp) PARALLEL(scott_emp, 5) */ ename
FROM scott.emp scott_emp;
```

NOPARALLEL

Hint NOPARALLEL zakazuje paralelné operácie nad tabuľkou. V prípade, že je použitý aj hint PARALLEL, tak má NOPARALLEL väčšiu prioritu.

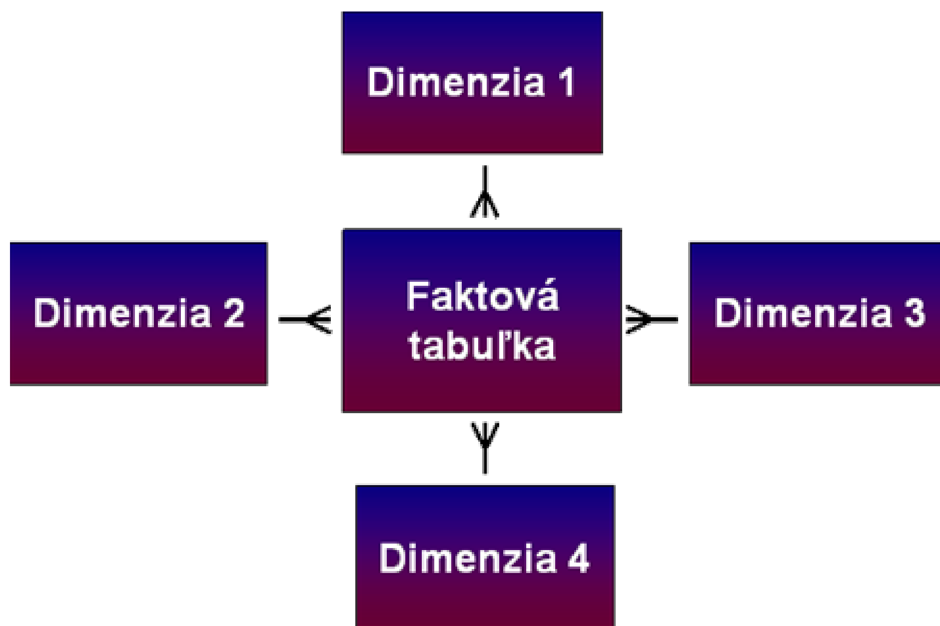


Nasledujúci príklad ukazuje použitie hintu NOPARALLEL:

```
SELECT /*+ NOPARALLEL(scott_emp) */ ename
FROM scott.emp scott_emp;
```

8 Optimalizácia star schém

Jedným zo základných dizajnov v dátových skladoch je star schéma. Jadro hviezdy tvorí jedna alebo viacero veľkých, faktových tabuliek. Na toto jadro sa napája viacero menších tabuliek, tzv. dimenzí. Každá dimenzia obsahuje detailnejšie informácie o nejakom atribúte z faktovej tabuľky. Úlohou optimalizátora je efektívne pripojiť viacero dimenzií na faktovú tabuľku. Toto spojenie je možné vykonať viacerými spôsobmi, pričom niektoré prístupy môžu byť niekoľkonásobne rýchlejšie ako iné.

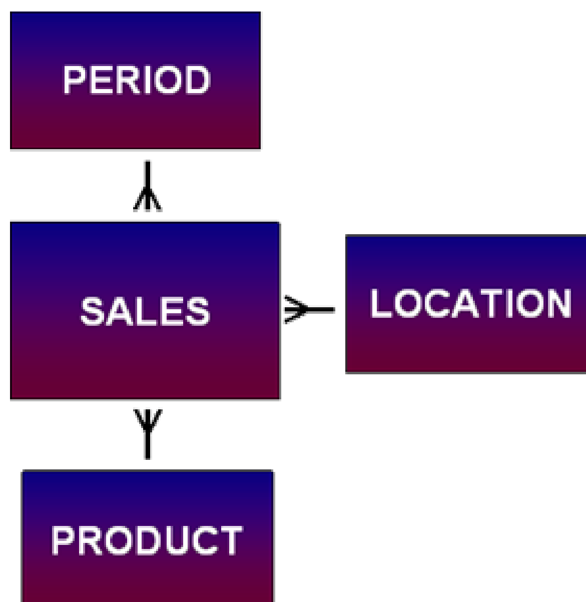


Obr. 5: Všeobecná star schéma

Uvedieme viacero prístupov k optimalizácii star schém a postupný vývoj týchto techník.

Náš pokusný SQL dotaz bude nasledovný: Koľko piva a kávy sa predalo v Dallase počas decembra 1998?

```
SELECT prod.category_name,
       sum (fact.sales_unit) Units,
       sum (fact.sales_retail) Retail
FROM   sales                 fact,
       period                per,
       location              loc,
       product               prod
WHERE  fact.period_id      = per.period_id
  AND  fact.location_id    = loc.location_id
  AND  fact.product_id     = prod.product_id
  AND  per.levelx          = 'DAY'
  AND  per.period_month    = 12
  AND  per.period_year     = 1998
  AND  loc.levelx          = 'STORE'
  AND  loc.city            = 'DALLAS'
  AND  loc.state           = 'TX'
  AND  prod.levelx         = 'ITEM'
  AND  prod.category_name in ('BEER','COFFEE')
GROUP BY prod.category_name;
```



Obr. 6: Star schéma príkladu

8.1 Technika Nested loop

Prvým spôsobom optimalizácie star schém je použitie Nested loop spojenia. Faktová tabuľka sa najprv spojí s prvou dimenziou a následne sa pripájajú ďalšie dimenzie využitím Nested loop spojenia. Pri spájaní dimenzií s veľkou faktovou tabuľkou výkonnosť dramaticky klesá.

Plán vykonania dotazu nad star schémou použitím Nested loop:

Description	CPU cost	IO cost	Object name	Cost
SELECT STATEMENT, GOAL = CHOOSE		133895		133895
SORT GROUP BY		133895		133895
NESTED LOOPS		133814		133814
NESTED LOOPS		8814		8814
NESTED LOOPS		34		34
TABLE ACCESS FULL		1	LOCATION	1
TABLE ACCESS FULL		33	PERIOD	33
TABLE ACCESS FULL		1756	SALES	1756
TABLE ACCESS BY INDEX ROWID		1	PRODUCT	1
INDEX UNIQUE SCAN			PRODUCT_PK	

Používať nested loop spojenie na pripájanie faktovej tabuľky, ktorá obsahuje často viac ako 10 milónov záznamov je mierne povedané neefektívne. Je to často nevykonatelná operácia.

Výsledky behu dotazu nad faktovou tabuľkou s 10 mil. riadkov použitím Nested loop:

```
18,450 physical reads
969,712 logical reads
12,271 CPUs used by session
427.251 seconds elapsed time
```

Dotazy nad star schémami sú v prostredí dátových skladov veľmi časté, preto tento pomalý prístup nemožno v žiadnom prípade doporučiť. Uplatnenie snáď môže nájsť len vo veľmi malých databázach, ale na prístup k obsiahlejšej star schéme by nemal byť nikdy používaný.

Ešte dodávame vysvetlenie štatistík, ktorými sme merali výkonnosť dotazu:

`physical reads` – celkový počet dátových blokov, ktoré bolo nutné načítať z diskov pri vykonávaní dotazu

`logical reads` – celkový počet logických čítaní (z diskov aj z pamäte), ktoré boli nutné na vykonanie dotazu

`CPUs used by session` – celkový čas CPU, ktorý ubehol od začiatku požiadavky po jej koniec. Meria sa v 10-tkach milisekúnd. Ak je operácia vykonaná rýchlejšie, ako 10 milisekúnd, do štatistík sa zapíše 0.

`seconds elapsed time` – celkový čas, ktorý ubehol od spustenia dotazu po jeho koniec. Za koniec výkonávania dotazu pokladáme vygenerovanie všetkých riadkov výsledku dotazu.

8.2 STAR hint

Druhou metódou, ako vykonať dotaz nad star schémou, je použitie tzv. `STAR` hintu. V tomto prípade sa najprv spoja kartézskym súčinom všetky riadky dimenzií. Vytvoria tak tabuľku, ktorá sa spojí s veľkou faktovou tabuľkou. Kartézsky súčin všetkých dimenzií, najmä ak sú veľké alebo je ich veľký počet, je príliš drahá operácia.

Plán vykonania dotazu nad star schémou použitím `STAR` hintu:

Description	CPU cost	IO cost	Object name	Cost
SELECT STATEMENT, GOAL = CHOOSE		21072170		21072170
SORT GROUP BY		21072170		21072170
NESTED LOOPS		21072089		21072089
MERGE JOIN CARTESIAN		89		89
MERGE JOIN CARTESIAN		39		39
TABLE ACCESS BY INDEX ROWID		8	PERIOD	8
BITMAP CONVERSION TO ROWIDS				
BITMAP AND				
BITMAP INDEX SINGLE VALUE			PERIOD_B3	
BITMAP INDEX SINGLE VALUE			PERIOD_B2	
BUFFER SORT		31		31
TABLE ACCESS FULL		1	LOCATION	1
BUFFER SORT		88		88
TABLE ACCESS FULL		10	PRODUCT	10
TABLE ACCESS FULL		1756	SALES	1756

Výsledky behu dotazu nad faktovou tabuľkou so 10 mil. riadkov použitím `STAR` hintu:

```
8,257 physical reads
178,144,751 logical reads
```

```
39,142 CPUs used by session
326.782 seconds elapsed time
```

8.3 Star transformácia

Star transformácia je spôsob optimalizácie dotazov nad star schémami, ktorý využíva bitmapove indexy a hash spojenia. Nad každým atribútom faktovej tabuľky, ktorý je cudzím kľúčom nejakej dimenzie, musí byť vytvorený bitmapový index.

Plán vykonania dotazu nad star schémou použitím star transformácie:

Description	IO cost	Object name	Cost
SELECT STATEMENT, GOAL = CHOOSE	683		683
TEMP TABLE GENERATION			
TEMP TABLE GENERATION			
SORT GROUP BY	683		683
HASH JOIN	679		679
HASH JOIN	668		668
TABLE ACCESS FULL	1	ORA_TEMP_1_30	1
TABLE ACCESS BY INDEX ROWID	666	SALES	665
BITMAP CONVERSION TO ROWIDS			
BITMAP AND			
BITMAP MERGE			
BITMAP KEY ITERATION			
TABLE ACCESS FULL	16	ORA_TEMP_1_30	16
BITMAP INDEX RANGE SCAN		SALES_B1	
BITMAP MERGE			
BITMAP KEY ITERATION			
TABLE ACCESS FULL	1	LOCATION	1
BITMAP INDEX RANGE SCAN		SALES_B2	
BITMAP MERGE			
BITMAP KEY ITERATION			
TABLE ACCESS FULL	10	PRODUCT	10
BITMAP INDEX RANGE SCAN		SALES_B3	
TABLE ACCESS FULL	10	PRODUCT	10

Výsledky behu dotazu nad faktovou tabuľkou s 10 mil. riadkami použitím star transformácie:

```
4,128 physical reads
22,584 logical reads
112 CPUs used by session
4.241 seconds elapsed time
```

Tajomstvo efektivity tohto prístupu je v tom, že Oracle najprv prepíše dotaz a použije poddotazy pre každú dimenziu. Dotaz sa nasledne vykoná v dvoch krokoch. Najprv použije bitmapové indexy vo faktovej tabuľke aj v dimenziách tak, aby zredukoval množinu riadkov. Potom spojí množinu riadkov z faktovej tabuľky s dimenziami. Čiže na prístup k faktovej tabuľke sa používajú len efektívne bitmapové indexy, ktoré dramataicky zredukujú množinu čitaných riadkov. Až výsledná menšia množina vstupuje do hash spojenia. Klasické b-tree indexy sa pri tomto prístupe vôbec nepoužívajú.

Prepísaný dotaz po star transformácii môže vyzerať nasledovne:

```
SELECT ...
FROM   SALES             fact
WHERE  fact.period_id    in (
        SELECT period_id
        FROM   period
        WHERE  levelx       = 'DAY'
        AND    period_month = 12
        AND    period_year  = 1998 )
  AND  fact.location_id  in (
        SELECT location_id
        FROM   location
        WHERE  levelx = 'STORE'
        AND    city   = 'DALLAS'
        AND    state  = 'TX' )
  AND  fact.product_id   in (
        SELECT product_id
        FROM   product
        WHERE  levelx        = 'ITEM'
        AND    category_name in ('BEER','COFFEE') )
...;
```

Pri optimalizácii star schémy použitím star transformácie optimalizátor niekedy použije dočasnú tabuľku. To však neplatí vždy a všeobecný plán vykonania dotazu obsahujúceho star transformáciu je nasledovný:

```
Hash join
      Table access by index ROWID pre faktovú tabuľku
            Bitmap AND
                  Bitmap MERGE
                        Table access by index ROWID pre dimenziu
                              Bitmap "AND"
                                    Bitmap index scan
                                    Bitmap index scan
                        Bitmap index range scan faktovej tabuľky
            Bitmap MERGE
      Table access by index ROWID pre dimenziu
            Bitmap "AND"
                  Bitmap index scan
                  Bitmap index scan
Bitmap index range scan faktovej tabuľky
```

Celá star transformácia je závislá na indexoch pre dimenzie a faktovú tabuľku. Plán vykonania obsahuje len výlučne bitmapové indexy, žiadne b-tree indexy. Ak plán obsahuje aj b-tree indexy, nejde o star tranformáciu a čas behu dotazu bude zrejme dramaticky vyšší.

8.4 Spájacie bitmapové indexy

Ďalšia technika na optimalizáziu star schém je založená na špeciálnych spájacích bitmapových indexoch. Spájací bitmapový index vytvorí index nad stĺpcom tabuľky použitím stĺpca z inej tabuľky. Hlavný trik je v tom, že spájací index si uchováva už vypočítaný výsledok spojenia v efektívnej indexovej štruktúre. Tento prístup by mal byť tým najrýchlejším a najefektívnejším spôsobom optimalizácie dotazov nad star schémami, lebo spájacie bitmapové indexy boli vytvorené práve pre tento účel. Použitie tohto spôsobu indexovania však nie je jednoduché a môže mať nežiadúce účinky.

Najprirodzenejší postup, ako použiť bitmapové spájacie indexy je nahradiť bitmapové indexy s predošlej star transformácie spájacími indexami. Namiesto bitmapových indexov na faktovej tabuľke, ktoré sú vytvorené na stĺpcoch cudzích kľúčov dimenzií, vytvoríme bitmapové spájacie indexy.

Napr. namiesto príkazu:

```
CREATE BITMAP INDEX SALES_B1 ON SALES (PERIOD_ID)
```

použijeme príkaz na vytvorenie spájacích indexov:

```
CREATE BITMAP INDEX SALES_BJ1 ON SALES (PER.PERIOD_ID)
FROM SALES FACT, PERIOD PER
WHERE POS.PERIOD_ID = PER.PERIOD_ID
```

Výsledky behu dotazu nad faktovou tabuľkou s 10 mil. riadkami použitím bitmapových spájacích indexov:

```
96,626 physical reads
120,145 logical reads
970 CPUs used by session
38.97 seconds elapsed time
```

Podľa výsledokv je zrejmé, že sa situácia oproti star tranformácii zhoršila a nie zlepšila. Optimalizátor síce používa nové bitmapové spájacie indexy, ale plán vykonania tiež obsahuje nested loop spojenie a b-tree indexy. V dátových skladoch sa však snažíme použitiu b-tree indexov vyhnúť, pokiaľ je to možné. Najefektívnejší spôsob je vykonávať dotaz výhradne použitím bitmapových indexov.

Preto riešenie bude nasledovné. Nenahradíme bitmapové indexy zo star transformácie novými spájacími indexami, ale ich doplníme o tieto indexy. Optimalizátor tak bude mať možnosť vybrať si v každej situácii, ktorý prístup je efektívnejší. Pre každý konkrétny dotaz sa potom môže rozhodnúť, či použije spájacie bitampové indexy alebo normálne bitmapové indexy.

Výsledky behu dotazu nad faktovou tabuľkou s 10 mil. riadkami použitím spájacích bitmapových indexov aj normálnych bitmapových indexov:

```
7,121 physical reads
27,981 logical reads
279 CPUs used by session
13.850 seconds elapsed time
```

Plán vykonania poslednej optimalizácie je podobný, ako plán pre star transformáciu. Chýba tu krok transformácie dočasnej tabuľky, ktorý je potrebný pri star transformácii. Full table prehľadávanie dočasnej tabuľky je nahradené za bitmap range scan spájacieho bitmapového indexu. Plán vykonania má však stále štruktúru star tranfomácie a výsledky bitmapového prehľadávania spracúva bitmap merge proces.

9 Záver

Optimalizácia dotazu je často veľmi zložitý proces, ktorý môže a nemusí vygenerovať efektívny plán vykonania dotazu. V diplomovej práci sme popísali viaceré procesy a metódy, ktoré sú súčasťou optimalizátora v databázovom systéme Oracle9i. Porovnali sme dostupné metódy na optimalizáciu dotazov nad štruktúrou star schém. Výsledky behov dotazov jasne ukazujú, že najefektívnejším prístupom na optimalizovanie dotazov nad star schémami je metóda, ktorá využíva star transformáciu. Dáva najlepší pomer medzi fyzickými a logickými vstupno-výstupnými operáciami, generuje najmenšie zaťaženie procesora a čas behu dotazu.

Metóda	Physical reads	Logical reads	CPUs used	Seconds elapsed
Nested loop	18,450	969,712	12,271	427.251
STAR hint	8,257	178,144,751	39,142	326.782
Star transformácia	**4,128**	**22,584**	**112**	**4.241**
Spájacie bitmapové indexy	7,121	27,981	279	13.850

Tabuľka 1: Metódy optimalizácie star schém

10 Použitá literatúra

[1] Sitansu S. Mittra: *Database Performance Tuning and Optimization*
Springer-Verlag New York, Inc., 2003

[2] Oracle Corp.: *Oracle9i Database Documentation*
Oracle Technology Network, 2002

[3] Donald K. Burleson: *Donald Burleson Articles*
http://www.dba-oracle.com/articles.htm#burleson_arts

[4] Donald K. Burleson: *Cost Control: Inside the Oracle Optimizer*
http://www.oracle.com/technology/oramag/webcolumns/2003/techarticles/

[5] Bert Scalzo: *Oracle DBA Guide to Data Warehousing and Star Schemas*
Prentice Hall PTR, 2003

[6] Joseph B. Green: *Oracle DBA Survival Guide*
Sams Publishing, 1995

[7] Bonnie O'neill: *Oracle Data Warehousing Unleashed*
Macmillan Computer Publishing, 1997

[8] Humphries Hawkins: *Data Warehousing Architecture and Implementation*
Prentice Hall PTR, 1998