Debreceni Egyetem
Informatikai Kar

Betekintés az SQL hangolásba Oracle környezetben

Témavezető:
Dr. Juhász István
Egyetemi adjunktus

Készítette:
Thurzó Ákos György
Programtervező informatikus

Debrecen
2009

Tartalomjegyzék

Bevezetés

Napjaink modern informatikai rendszereinek működéséhez elengedhetetlen a megfelelő és hatékony adatkezelés. A legtöbb nagy rendszer az adatait egy adatbázis-kezelő rendszer segítségével menedzseli. Az Oracle adatbázis-kezelő rendszer jelenleg piacvezető a relációs adatbázis-kezelő rendszerek között, az Oracle DBMS-sel (Database Management System – adatbázis-kezelő rendszer) kapcsolatos tudásra tehát valódi piaci igény mutatkozik. Azért is választottam ezt a témát, mert ebben a témában nagyon kevés a magyar nyelvű írásos anyag.

Az Oracle adatbázis-kezelőhöz két csomag is létezik, amely a teljesítményhangolás témaköréhez kapcsolódik, a diagnosztikai (Diagnostic Pack) és a hangoló csomag (Tuning Pack). A diagnosztikai csomag segítségével az Oracle önmagát figyeli az ADDM (Automatic Database Diagnostic Monitor) keretrendszer segítségével. Az ADDM a folyamatos teljesítményanalízis alapján felhívja az adatbázis adminisztrátor, vagy a fejlesztő figyelmét azokra a szűk keresztmetszetekre, amelyek a rendszer életében gondot okoznak, vagy okozhatnak. A figyelmeztetéssel nem csupán azt jelzi, hogy mivel van a probléma, hanem egy következő lépést is ajánl a megoldás felé vezető úton. Legtöbbször azt, hogy a hangoló csomagban elérhető tanácsadói eszközök közül futtassunk le egyet, például a Tuning Advisor, vagy az Access Advisor nevű eszközt. Ezek az eszközök pedig egy ajánlást tesznek arra, hogy milyen módosítást lenne érdemes végrehajtani, amit akár egy gombnyomással el is fogadhatunk. Ha ezeket az eszközöket használjuk, akkor az a szakmai tudásunk rovására mehet, mivel gyakorlatilag minden lépést automatizál a hangolás során. Az Oracle újabb és újabb verziói egyre inkább az automatizálás az irányába mozdulnak el. Azon kívül, hogy a fentebb említett két csomag plusz költséget jelent egy Oracle adatbázis-kezelő megvásárlásakor, mivel rendkívül komplex, és korántsem tökéletes, így bonyolult esetben nem feltétlenül működik úgy, ahogy elvárja tőle a használója. Azért érdemes továbbra is tanulni a hangolást, mert ilyen esetekben tudni kell, hogy mit tegyen az ember. Érteni kell, hogy hogyan működik a rendszer.

A hangolás érdekes feladat, mert teljes mértékben ismernünk kell a rendszert, amelynek hangolását végezzük. A hangolási folyamat nem csak rengeteg stabil elméleti és gyakorlati tudást, tapasztalatot, hanem egy általános nyitott gondolkozást is igényel. E nélkül ugyanis szinte lehetetlen kitörni a logikai keretekből és új aspektusokból megvizsgálni az adott esetet.

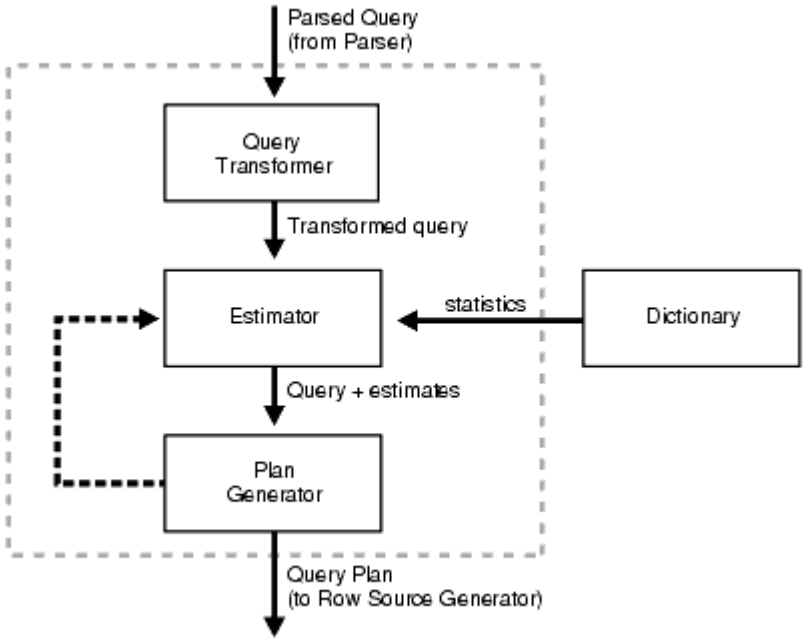
Azért választottam a címben a „betekintés” szót, mert egy betekintést szeretnék nyújtani az SQL hangolás világába. A dolgozatommal szeretném bemutatni az SQL optimalizáló célját, szerepét és működését, a végrehajtási terv szerkezetét, műveleteit. Rá szeretnék világítani a statisztikák szerepére, különböző formáira. Be szeretném mutatni a nyomkövetési technikákat, valamint egy bevezetést szeretnék adni a nyomkövetés eredményeképpen előálló adatok értelmezéséhez. Az elméleti tudnivalókat igyekszem a legtöbb esetben gyakorlati példákkal megfűszerezni.

Az optimalizáló

Az SQL egy deklaratív nyelv, így ahhoz, hogy az SQL-ben írt utasításokat végre tudja hajtani a számítógép, át kell alakítani úgy, hogy procedurális elemeket, „kézzel fogható” műveleteket kell készíteni az egyes utasításrészekből. Egy SQL utasítást rengeteg féle módon végre lehet hajtani úgy, hogy az eredményhalmaz matematikai úton bizonyíthatóan megegyezzen minden esetben. A sokféle végrehajtási lehetőségből segít választani egyet az optimalizáló. Az a célja, hogy egy SQL utasításból egy olyan műveletsort állítson elő, amely valamilyen szempontból optimálisnak tekinthető. A műveletsort, amelyet előállít az optimalizáló *végrehajtási tervnek* nevezzük.

A lekérdezés optimalizáló a következő komponensekből áll:

- Query Transformer – Lekérdezés átalakító
- Estimator – Becslő
- Plan Generator – Végrehajtási terv generátor



A lekérdezés átalakítónak az a feladata, hogy lehetőség szerint átalakítsa a lekérdezést annak érdekében, hogy egy jobb tervet lehessen generálni a lekérdezésből.

Query Transformer – lekérdezés átalakító

Ahogy a képen is látszik a lekérdezés átalakító a parszolt lekérdezést kapja meg bemenetként, amit lekérdezés blokkok egy halmaza reprezentál. Ezek a blokkok egymásba vannak ágyazva, vagy valamilyen kapcsolatban vannak egymással.

A következő átalakítási technikák közül választhat a lekérdezés átalakító:

- *View Merging (nézet összeolvasztás)* – Minden hivatkozott nézet a lekérdezésben egy külön lekérdezés blokk-ként jelenik meg. Ez azt jelenti, hogy az optimalizáló külön végrehajtási tervet generál a hivatkozott nézethez a felhasználási környezettől függetlenül. Lehetséges, hogy ez a módszer nem eredményez optimális végrehajtási tervet, ezért a lekérdezés átalakító összeolvaszthatja a tartalmazó lekérdezés blokkot és a tartalmazott lekérdezés blokkot. Ekkor már nem szükséges egy független végrehajtási tervet készíteni, mert a lekérdezés egészét kell tekinteni végrehajtási terv generálás közben.
- *Predicate Pushing (feltétel áthelyezés)* – Nem összeolvasztott nézetek esetében használhatjuk a feltétel áthelyezés átalakítási technikát. A tartalmazó lekérdezés blokkban szereplő feltételt áthelyezhetjük a tartalmazott lekérdezés blokkba, így a belső nézet hatékonyságát növelhetjük.
- *Subquery Unnesting (allekérdezés kibontás)* – Ha egy lekérdezés allekérdezést tartalmaz, akkor az allekérdezéshez külön végrehajtási tervet generálhat az optimalizáló. A másik lehetőség, hogy alkalmazza az allekérdezés kibontás átalakítási technikát, ami azt jelenti, hogy az allekérdezést összekapcsolássá (join) alakítja.
- *Query Rewrite with Materialized Views (lekérdezés átírás materializált nézetekkel)* – Az optimalizáló átírhatja az eredeti lekérdezést úgy, hogy egy materializált nézetet használjon ahelyett, hogy ténylegesen lefutna a lekérdezés. Természetesen ezt csak akkor teheti meg, ha talál olyan materializált nézetet, aminek a definíciójában szereplő lekérdezés kompatibilis az általunk éppen futtatni kívánt lekérdezéssel, vagyis a csere logikailag végrehajtható.

Estimator – becslő

A becslő különböző mérőszámokat generál:

- Selectivity – szelektivitás
- Cardinality – számosság
- Cost – költség

Ezek a mérőszámok összekapcsolódnak, az egyik befolyásolja a másikat. A becslő végcélja, hogy megbecsülje a végrehajtási terv egészére vonatkozó költséget. A költségek becsléséhez nagy segítségére lehetnek a becslőnek az objektumokról tárolt statisztikák. Általánosan véve mondhatjuk, hogy minél több és pontosabb információ, statisztika, metaadat áll a rendelkezésünkre, annál pontosabb becslést tudunk adni egy feladat végrehajtásának költségét illetően.

Selectivity – szelektivitás

A szelektivitás egy nullától egyig terjedő intervallumba eső szám. Mindig egy feltételhez kötődik, és azt reprezentálja, hogy sorok egy halmazából hány felel meg a feltételnek. A 0 szelektivitás azt jelöli, hogy nem fog egyetlen sor sem megfelelni a halmazból, az 1 szelektivitás pedig azt, hogy minden sor meg fog felelni a feltételnek.
Ha az optimalizáló nem rendelkezik statisztikákkal az adott forráshalmazról, akkor dinamikus mintavételezés vagy egy belső alapértelmezés segítségével tud szelektivitást számolni.
Ha az optimalizáló rendelkezésére állnak a statisztikák, akkor például egy egyenlőségre történő feltételhez kapcsolódóan (oszlop = érték) a szelektivitás az oszlop különböző értékeinek a reciproka lehet. Még pontosabb szelektivitási értéket kaphatunk, ha hisztogram is tartozik az adott oszlophoz és azzal számolunk az oszlopban szereplő különböző értékek helyett.

Cardinality – számosság

A számosság a sorok számát mutatja sorok egy halmazában. Többfélre számosságról is beszélhetünk:

- Alap számosság (Base Cardinality): a sorok száma egy táblában. Ezt a tábla elemzésével tudjuk könnyen elérhetővé tenni az optimalizáló számára. Ha nincs elemezve a tábla, akkor a becslő a tábla által elfoglalt extentek számából képes megbecsülni a sorok számát.
- Effektív számosság (Effective Cardinality): azon sorok száma, amelyeket ténylegesen kiválasztottunk egy lekérdezéssel a táblából. Ez függ azoktól a feltételektől, amelyek a táblához kapcsolódnak. Tulajdonképpen az effektív számosság az alap számosságból és a táblára vonatkozó összes feltétel szelektivitásából megmondható. Ha nincs feltétel a táblához kapcsolódóan, akkor az effektív számosság megegyezik az alap számossággal.
- Összekapcsolási számosság (Join Cardinality): Két sorhalmaz összekapcsolása után kapott sorhalmaz sorainak a száma. Mivel az összekapcsolás Descartes-szorzata két sorhalmaznak szűrőként hozzávéve az eredményhez egy összekapcsolási feltételt, ezért az összekapcsolási számosság a két sorhalmaz számosságának és a kapcsolófeltétel szelektivitásának a szorzata
- Egyediségi számosság (Distinct Cardinality): Egy sorhalmaz egy oszlopában lévő különböző értékeknek a száma.
- Csoport számosság (Group Cardinality): Egy GROUP BY operátor alkalmazása utáni sorhalmaz számossága. Ez nagyban függ a GROUP BY után szereplő oszlop eltérőségi számosságától.

Cost – költség

A költség a munka egységét, vagy a felhasznált erőforrásokat reprezentálja. Az optimalizáló a munka egységének a diszk I/O-t, a CPU felhasználást és a memória felhasználást tekinti. A költség tehát egy olyan szám, ami egy becsült értéke a szükséges diszk I/O, CPU és memória felhasználásnak egy adott művelethez kapcsolódóan.

Plan Generator – végrehajtási terv generátor

A végrehajtási terv generátor számos végrehajtási tervet generál, majd ezek közül a legkisebb költségűt választja. Gondoljunk csak bele mennyi féle végrehajtási lehetőség van, ha csak azt tekintjük, hogy három táblát össze akarunk kapcsolni – azt általában nem írjuk bele a lekérdezésbe, hogy mi legyen az összekapcsolások sorrendje, így tehát az optimalizálónak kell

döntenie, hogy a 6 lehetőség közül melyiket válassza.
A Plan Generator természetesen nem úgy működik, hogy legenerálja az összes lehetséges végrehajtási tervet, majd azok közül a legkisebb költségűt választja, hanem különböző vágásokat hajt végre a generálások közben, ha az aktuálisan generált terv már túl nagy költségűnek bizonyul. A vágás jól használható, ha a legelső generált terv esetén olyan összekapcsolási sorrendet kapunk, amelynek a költsége közel van az optimális költséghez. Az, hogy elsőre találjunk egy ilyen összekapcsolási sorrendet, egy nehéz feladat.
A végrehajtási terv generátor először a legbelsőbb lekérdezés blokkhoz generál végrehajtási terveket, majd ha ezt már optimalizálta, akkor halad kifelé.

Az optimalizáló a végrehajtási terv generálása közben a következő műveleteket kell végrehajtania:

- Kifejezések és feltételek kiértékelése
- Utasítás transzformáció
- Optimalizálási cél meghatározása
- Hozzáférés módja (Access Path)
- Összekapcsolási sorrend (Join Order): azoknál az utasításoknál, amelyeknél több táblát kell összekapcsolni, el kell dönteni ezeknek az összekapcsolási sorrendjét.
- Összekapcsolási mód (Join Method): minden összekapcsoláshoz ki kell választani az összekapcsolási lehetőségek közül egyet.

Optimalizálási cél meghatározása

Kétféle optimalizálási célról beszélhetünk adatbázis környezetben: válaszidőre (response time), vagy áteresztő képességre (throughput). Általában véve az a cél, hogy nagyobb legyen a végrehajtási sebesség, illetve, hogy az eredményt minél hamarabb visszakapjuk. Ez természetesen együtt jár azzal is, hogy az utasításvégrehajtás során kevesebb erőforrást használunk fel.

Ha válaszidőre történik az optimalizálás, akkor az a cél, hogy a lekérdezés eredményének első néhány sorát kapjuk meg gyorsan, majd a többit ráérünk később is. Ennél a célnál tehát a részeredmények visszakapásának sebessége számít leginkább. Ez általános optimalizálási cél például egy felhasználói felületen. Az áteresztő képességre történő optimalizálási célnál pedig az számít optimális végrehajtásnak, ha az SQL utasítás egésze gyorsabban fejeződik be. Itt nem érdekes a részeredmények visszakapásának sebessége, csak a teljes utasítás végrehajtásának sebessége. Gondoljunk csak bele hogyan éreznénk magunkat, mint felhasználó, ha egy mai webes kereső nem válaszidőre lenne optimalizálva – végigvárhatnánk míg az összes találatot megkeresi, miközben valószínűleg az első néhány találat között ott van amit mi keresünk.

Az optimalizálási célt globálisan az OPTIMIZER_MODE inicializációs paraméterrel tudjuk beállítani. Ennek értékei lehetnek CHOOSE, ALL_ROWS, FIRST_ROWS_*n*, FIRST_ROWS vagy RULE.

CHOOSE

Az optimalizáló választ a költség alapú és a szabály alapú megközelítés között attól függően, hogy rendelkezésre állnak-e statisztikák, vagy nem. Ez az alapértelmezett beállítás. Ha az adatszótár tartalmaz statisztikát legalább egy érintett tálbláról, akkor az optimalizáló a költség alapú megközelítést választja, és áteresztő képességre optimalizál. Ha csak néhány statisztika áll rendelkezésre, az optimalizáló még úgy is a költség alapú optimalizálást választ, de a kevés vagy nem megfelelő statisztika könnyen vezethet nem hatékony végrehajtási tervhez.

Ha az adatszótárban nincs statisztika azokhoz az objektumokhoz, amelyeket felhasználunk, akkor szabály alapú optimalizálás fog történni.

ALL_ROWS

Az optimalizáló mindenféleképpen költség alapú optimalizálási módot választ függetlenül attól, hogy rendelkezésre állnak-e statisztikák vagy sem. A cél az áteresztő képesség fokozása.

FIRST_ROWS_*n*

Az optimalizáló itt is mindenképpen költség alapú optimalizálást választ. A cél a leggyorsabb válaszidő, az első *n* sor leggyorsabb megkapása; *n* lehetséges értékei 1, 10, 100, vagy 1000.

FIRST_ROWS

Költség és heurisztika keverékét használja az optimalizáló ahhoz, hogy az első néhány sort a leggyorsabban tudja visszaadni.

RULE

Ennek az értéknek a hatására az optimalizáló mindenképpen szabály alapú optimalizálási módot választ függetlenül attól, hogy rendelkezésre állnak-e statisztikák vagy sem.

A végrehajtási terv (Execution Plan)

Minden SQL utasítás végrehajtásához szükség van egy végrehajtási tervre. A kész terv alapján fog végrehajtódni az utasítás. De hogyan is néz ki egy ilyen terv, illetve hogyan nézhetjük meg a végrehajtási tervet?

A végrehajtási terv lépések kombinációja. Egy lépés egy műveletet (operation) jelent. A végrehajtási terv minden felhasznált objektumhoz egy hozzáférési módot tartalmaz, valamint táblák összekapcsolásánál a kapcsolási sorrendet és módot.

Lássunk egy példát a végrehajtási tervre. Legyen adott a következő SQL utasítás:

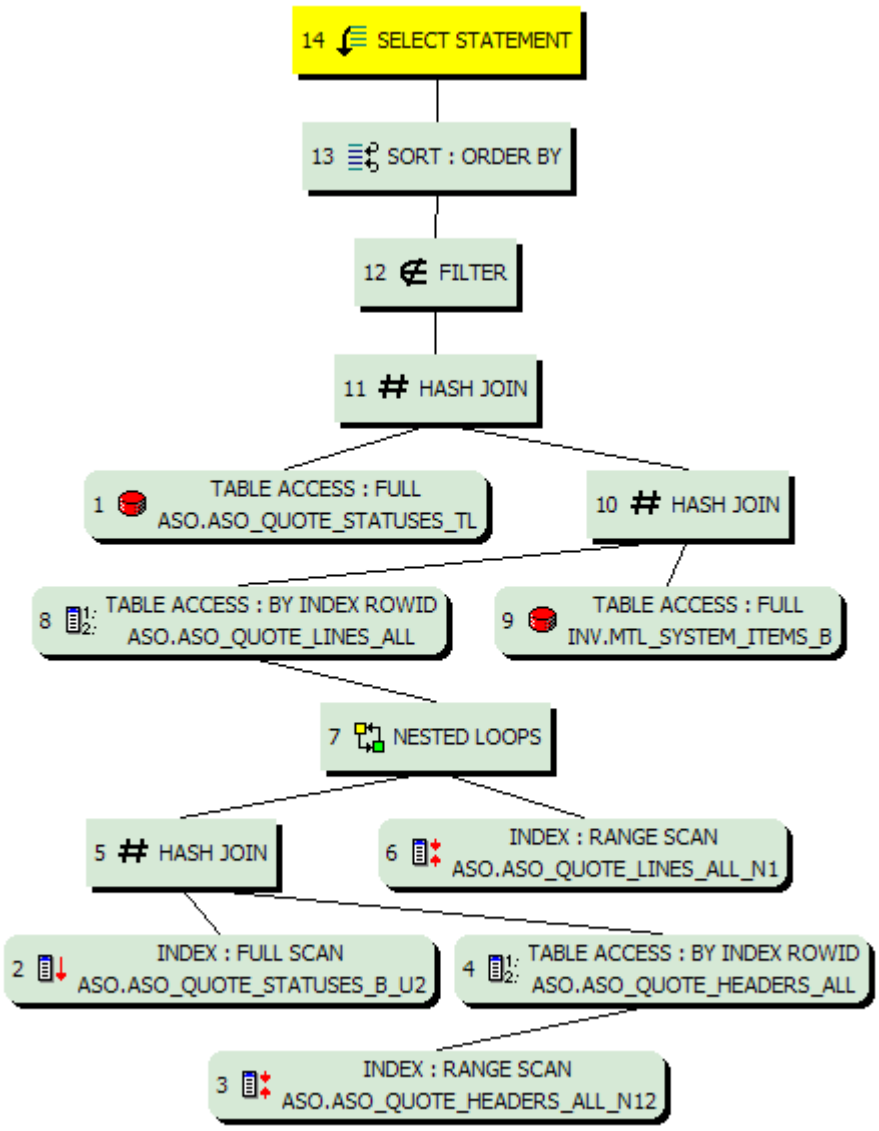
```
SELECT h.quote_number quote_number
     , h.quote_version quote_version
     , s.meaning status
     , l.quantity quantity
     , i.segment1 item
  FROM aso_quote_headers_all h
     , aso_quote_lines_all l
     , aso_quote_statuses_vl s
     , mtl_system_items_b i
 WHERE h.quote_header_id = l.quote_header_id
   AND h.quote_status_id = s.quote_status_id
   AND l.inventory_item_id = i.inventory_item_id
   AND h.creation_date between :low and :high
 ORDER BY h.quote_number, l.line_number;
```

A hozzátartozó végrehajtási terv:

```
-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                          |  Name                       | Rows  | Bytes |TempSpc| Cost (%CPU)|
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                             |   106K|  9164K|       | 57514   (6)|
|   1 |  SORT ORDER BY                     |                             |   106K|  9164K|    20M| 57514   (6)|
|*  2 |   FILTER                           |                             |       |       |       |            |
|*  3 |    HASH JOIN                       |                             |   106K|  9164K|       | 55145   (6)|
|*  4 |     TABLE ACCESS FULL              | ASO_QUOTE_STATUSES_TL       |    50 |  1100 |       |     4  (25)|
|*  5 |     HASH JOIN                      |                             |   106K|  6873K|       | 55117   (5)|
|   6 |      TABLE ACCESS BY INDEX ROWID   | ASO_QUOTE_LINES_ALL         |    10 |   180 |       |     9  (12)|
|   7 |       NESTED LOOPS                 |                             | 25803 |  1259K|       | 23009   (2)|
|*  8 |        HASH JOIN                   |                             |  2509 | 80288 |       |  2605   (2)|
|   9 |         INDEX FULL SCAN            | ASO_QUOTE_STATUSES_B_U2     |    50 |   250 |       |     2  (50)|
|  10 |         TABLE ACCESS BY INDEX ROWID| ASO_QUOTE_HEADERS_ALL       |  2509 | 67743 |       |  2603   (2)|
|* 11 |          INDEX RANGE SCAN          | ASO_QUOTE_HEADERS_ALL_N12   | 40139 |       |       |    26  (12)|
|* 12 |        INDEX RANGE SCAN            | ASO_QUOTE_LINES_ALL_N1      |    10 |       |       |            |
|  13 |      TABLE ACCESS FULL             | MTL_SYSTEM_ITEMS_B          |   569K|  8899K|       | 31970   (7)|
-----------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
   2 - filter(TO_DATE(:Z)<=TO_DATE(:Z))
   3 - access("B"."QUOTE_STATUS_ID"="T"."QUOTE_STATUS_ID")
   4 - filter("T"."LANGUAGE"=:B1)
   5 - access("L"."INVENTORY_ITEM_ID"="I"."INVENTORY_ITEM_ID")
   8 - access("H"."QUOTE_STATUS_ID"="B"."QUOTE_STATUS_ID")
  11 - access("H"."CREATION_DATE">=:Z AND "H"."CREATION_DATE"<=:Z)
  12 - access("H"."QUOTE_HEADER_ID"="L"."QUOTE_HEADER_ID")
```

Amit először látnunk kell, hogy minden sorban egy-egy művelet van. A műveletek fajtáiról később szó lesz. Hogy értelmezni tudjuk a műveletek sorrendjét érdemes egy grafikus ábrázolási módot szemügyre venni:



A végrehajtási tervre gondolhatunk úgy is, mint egy fa adatszerkezetre. Minden részfa gyökerének végrehajtása előtt végre kell hajtani a belőle ágazó műveleteket reprezentáló elemeket. Az ábrán a műveletek előtt szereplő szám a műveletek végrehajtásának sorendjére utal. A fenti táblázatban egyéb becsült adatokat láthatunk arra vonatkozóan, hogy hány sort fog eredményezni az adott művelet (Rows), mekkora méretű lesz az adott művelet eredményhalmaza (Bytes), mennyi ideiglenes táblaterületet fog felhasználni az adott művelet (TempSpc), és mennyi az adott művelet kötsége (Cost). A táblázat után az SQL utasításban szereplő feltételeket látjuk felsorolva. Alapvetően kétfajta feltételt különböztethetünk meg:

- Hozzáférési feltétel (Access predicate) – a művelet végrehajtása közben kerül kiértékelésre, csak a feltételnek megfelelő sorok kerülnek be az eredményhalmazba.
- Szűrőfeltétel (Filter predicate) – az eredményhalmazra utólagosan kerül kiértékelésre a szűrőfeltétel, amely alapján tovább szűkülhet az eredményhalmaz.

Mikor az optimalizáló elkészíti az éppen futtatni kívánt SQL utasítás végrehajtási tervét, akkor azt el tudjuk érni néhány dinamikus performancia nézeten keresztül, például a V$SQL_PLAN-en keresztül. Négy azonosító oszlopot emelnék ki először:

- ADDRESS – a szülő kurzor a kezelőjének a címe
- HASH_VALUE – a szülő utasítás hash értéke a library cache-ben.
- CHILD_NUMBER – a gyerek kurzor száma, amelyik az adott végrehajtási tervet használja.
- SQL_ID – a szülő kurzor SQL azonosítója a library cache-ben.

Az ADDRESS és a HASH_VALUE oszloppal a V$SQL_PLAN nézethez kapcsolhatjuk a V$SQLAREA nézetet, ahol a kurzorral kapcsolatos információkat érhetjük el.

Az ADDRESS, HASH_VALUE és CHILD_NUMBER oszlopokkal pedig V$SQL nézethez kapcsolhatjuk a V$SQL_PLAN nézetet, ahol láthatjuk például az SQL szövegét az SQL_TEXT oszlopban.

Ha tehát egy éppen futó, vagy nem rég futtatott SQL végrehajtási tervét szeretnénk megnézni manuálisan, akkor a V$SQL nézetben megtehetjük ezt az SQL_TEXT oszlop alapján, majd az ADDRESS, HASH_VALUE és CHILD_NUMBER oszloppal tovább tudunk menni a V$SQL_PLAN nézetre, ahol már láthatjuk a végrehajtási tervet.

Használhatjuk az EXPLAIN PLAN utasítást is végrehajtási terv generálására. Ekkor az optimalizáló végrehajtja ugyanazokat a lépéseket, mintha futtatni szeretnénk egy SQL utasítást, csak éppen megáll a végrehajtási terv generálása végén, nem kezdi el végrehajtani azt. Az

eredményt pedig egy adott táblába helyezi el – alapértelmezés szerint ez a tábla a PLAN_TABLE nevű tábla. Az EXPLAIN PLAN utasítás legegyszerűbb szerkezete:

```
EXPLAIN PLAN FOR SQL_UTASÍTÁS
```

Példa:

```
SQL> EXPLAIN PLAN FOR
  2  UPDATE aso_quote_headers_all
  3     SET price_frozen_date = sysdate
  4   WHERE quote_number = :qnum
  5     AND quote_version = :ver;
Explained.
```

Miután a végrehajtási terv a megfelelő táblában van, lekérdezhetjük azt egy egyszerű lekérdezéssel, vagy használhatjuk a DBMS_XPLAN csomag DISPLAY nevű függvényét:

```
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

-------------------------------------------------------------------------------------------
| Id  | Operation            |  Name                     | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------------
|   0 | UPDATE STATEMENT     |                           |     1 |    12 |     4  (25)|
|   1 |  UPDATE              | ASO_QUOTE_HEADERS_ALL     |       |       |            |
|*  2 |   INDEX UNIQUE SCAN  | ASO_QUOTE_HEADERS_ALL_U1  |     1 |    12 |     3  (34)|
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("ASO_QUOTE_HEADERS_ALL"."QUOTE_NUMBER"=TO_NUMBER(:Z) AND
              "ASO_QUOTE_HEADERS_ALL"."QUOTE_VERSION"=TO_NUMBER(:Z))
```

Műveletek és a hozzáférés módja (Access Path)

Minden olyan objektum esetén – amelyet az utasítás végrehajtása során felhasználunk valamilyen módon – meg kell határozni a hozzáférés módját. A hozzáférés módja azt jelenti, hogy hogyan érjük el az adatot az adatbázisban. Lekérdezésnél alapesetben figyelembe kell venni a FROM utasításrészben szereplő objektumokat és a hozzájuk tartozó WHERE utasításrészben szereplő kifejezéseket. A megfelelő hozzáférés módjának meghatározásához szükség van statisztikákra. A statisztikák alapján az optimalizáló arról is tud dönteni, hogy milyen műveletet hajtson végre. Ha tudjuk például, hogy egy lekérdezés eredményhalmazának számossága töredéke lesz az alaphalmaz számosságának, akkor érdemes valamilyen indexet felhasználni a végrehajtás során, ahelyett, hogy az egész alaphalmazt bejárnánk.

Hozzáférési módok közül néhány:

- Teljes tábla olvasás (Full Table Scan)
- Index olvasás (Index Scan)
- Sorazonosító olvasás (Rowid Scan)
- Táblaminta olvasás (Sample Table Scan)

Teljes tábla olvasás (Full Table Scan)

A teljes tábla olvasás azt jelenti, hogy egy adott táblát a végrehajtás során az elejétől a végéig bejárunk, és minden egyes sorra külön-külön eldöntjük, hogy megfelel-e a WHERE utasításrészben szereplő feltételeknek.

Példa teljes tábla olvasásra:

```
SELECT *
  FROM aso_quote_statuses_b;
```

```
-------------------------------------------------------------------------------
| Id  | Operation            |  Name                | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                      |    50 |  3600 |     5  (20)|
|   1 |  TABLE ACCESS FULL   | ASO_QUOTE_STATUSES_B |    50 |  3600 |     5  (20)|
-------------------------------------------------------------------------------
```

Teljes táblaolvasást akkor választ az optimalizáló, ha egy adott táblán nincs index (vagy nincs olyan index, amelyiket használhatná), vagy a táblában lévő adatok nagy részére szükség lesz az eredményhalmazban, esetleg túl kicsi a tábla ahhoz, hogy megérje az indexhez „nyúlni”. Ha nincs indexelve az adott tábla nyilvánvalóan nem tudunk gyorsítani azon a döntési folyamaton, hogy melyik sorra lesz szükségünk. A másik eset, ha a tábla nagy része benne lesz az eredményhalmazban. Ekkor nem célszerű a meglévő indexeket használni, hiszen minden indexből történő olvasáshoz egy újabb I/O művelet szükséges, hogy a sorazonosító (rowid) alapján ki tudjuk olvasni a megfelelő sort. Ha ezt a legtöbb sorra végre kell hajtani, akkor ez egy plusz I/O igény lenne. Tovább csökkentve az I/O igényt, az Oracle-ben a DB_FILE_MULTIBLOCK_READ_COUNT nevű paraméterben meghatározható, hogy egyetlen I/O művelet során hány blokkot olvassunk be egyszerre.

Sorazonosító olvasás (Rowid Scan)

Egy sor sorazonosítója meghatározza, hogy az adott sor egy adatfájl melyik adatblokkjában és azon belül pontosan milyen pozícióban érhető el. Ha egyetlen sort keresünk, akkor a sorazonosító olvasás a leggyorsabb módja, hogy megtaláljuk az adott sort, mivel a sorazonosító egyértelműen meghatározza annak pontos helyét. Általában a sorazonosító olvasás a második lépés az index olvasás után, ha az index nem tartalmazza az eredményhalmazban látni kívánt oszlopokat.

Példa a sorazonosító olvasásra:

```
SELECT update_allowed_flag
  FROM aso_quote_statuses_b
 WHERE status_code = 'FAXED';
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                   |  Name                     | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                           |     1 |    15 |     2  (50)|
|   1 |  TABLE ACCESS BY INDEX ROWID| ASO_QUOTE_STATUSES_B      |     1 |    15 |     2  (50)|
|*  2 |   INDEX UNIQUE SCAN         | ASO_QUOTE_STATUSES_B_U1   |     1 |       |            |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("ASO_QUOTE_STATUSES_B"."STATUS_CODE"='FAXED')
```

Index olvasás (Index Scan)

Az index egy olyan objektum, ami egy vagy több oszlopot és a hozzájuk tartozó sorazonosítót tartalmazza. Az index olvasás egy olyan művelet, amely bejárja az indexet az indexelt oszlopok alapján, majd visszaadja azoknak a sorazonosítók a halmazát, amelyek megfeleltek a keresési kritérium(ok)nak.

Példa index olvasásra:

```
SELECT status_code
  FROM aso_quote_statuses_b;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation            |  Name                     | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                           |    50 |   650 |     2  (50)|
|   1 |  INDEX FULL SCAN     | ASO_QUOTE_STATUSES_B_U1   |    50 |   650 |     2  (50)|
-------------------------------------------------------------------------------------
```

Az indexek használatával jó esetben növelhetjük az utasítás végrehajtásának sebességét. Általában kisebb az index objektum, mint az az objektum, amit indexelünk, ezért az index bejárása gyorsabb lesz. Ennek ellenére nem minden esetben érdemes index olvasást használni a végrehajtás során, mert ahogy azt a teljes tábla olvasásnál írtam, lehet, hogy több I/O műveletet vesz igénybe egy index olvasás és egy sorazonosító olvasás, mint egy teljes tábla olvasás.

Ha egy olyan műveletet hajtunk végre, amely során egy táblából csak indexelt oszlopot akarunk látni az eredményhalmazban, akkor az optimalizáló választhatja az index olvasást úgy, hogy azt nem követi sorazonosító olvasás, hanem egyszerűen az indexből olvassa ki azokat az oszlopértékeket, amelyekre szükség van.

Az index olvasásnak számos fajtája van indextől és végrehajtandó művelettől függően:

- Index Unique Scan*:* Az Index Unique Scan legfeljebb egy sorazonosítót ad vissza. Akkor lehet ezt a műveletet használni, ha garantált az értékek különbözősége.
- Index Range Scan: Az Index Range Scan egy általános index olvasási művelet. Ebben az esetben nincs szükség az értékek egyediségére. Ebben a hozzáférési módban a feltételben szereplő operátor lehet egyenlőség, kisebb, nagyobb, vagy akár like is, ha az nem százalékjellel kezdődik – ekkor ugyanis nem tudjuk felhasználni az indexet a keresésre.
- Index Skip Scan: Index Skip Scan összetett (composite) index estén használható akkor, ha a prefix oszlop nem szerepel a feltételek között.
- Full Scan: Indexen történő teljes keresést hajt végre. Akkor van értelme ennek a műveletnek, ha az összes szükséges oszlop benne van az adott indexben, mert akkor azt lehet, hogy gyorsabban tudja kiolvasni, mint teljes tábla olvasással.
- Fast Full Index Scan: Ez a művelet hasonló az Index Full Scan-hez, viszont itt több blokkos olvasást (multiblock read) használ a rendszer, így ez gyorsabb végrehajtást eredményez. Nagy előnye az Index Full Scan-hez képest, hogy párhuzamosítható a művelet.
- Index Joins: Több index hash-join művelettel történő összekapcsolását jelenti. Abban az esetben használható, ha az összes oszlop, amire a lekérdezésben szükség van, szerepel az indexekben. Ekkor nincs szükség a háttérben lévő tábla használatára, elég az indexeket összekapcsolni.
- Bitmap Joins: A Bitmap Join egy bittéképet használ a kulcs értékeknek és egy összerendelő függvényt, ami átalakít minden bitpozíciót egy sorazonosítóvá. A bittérképek hatékonyan kapcsolhatóak össze, mert alkalmazhatjuk rájuk a bitenkénti műveleteket.

Először azt gondolhatnánk, hogy az indexek használata minden esetben teljesítményjavulást okoz, de ez nincs mindig így. Ahogy azt a teljes tábla olvasás műveletnél írtam olyan esetben nem érdemes az indexet használni, amikor az alaptábla sorainak nagy részére szükség lesz az eredményhalmazban. Most, hogy már ismerjük mind a teljes tábla olvasást, mind az index olvasásokat, nézzünk egy példát arra, hogy mennyi plusz költséget jelent ha index olvasást használnánk olyan esetben, amikor valójában nincs szükség rá. Tekintsük a következő lekérdezést, ami lekérdezi azoknak a társaságoknak a nevét, amelyek az aktuális naptól korábban készültek:

```
SELECT party_name
  FROM hz_parties
 WHERE creation_date < TRUNC(sysdate);
```

A lekérdezés eredményhalmaza valószínűleg nagy részét tartalmazni fogja az eredeti hz_parties táblának, aminek néhány millió soros nagyságrendű a számossága. A lekérdezés végrehajtási terve:

```
-------------------------------------------------------------------------
| Id  | Operation            |  Name       | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |             |  4453K|   178M| 67248  (17)|
|*  1 |  TABLE ACCESS FULL   | HZ_PARTIES  |  4453K|   178M| 67248  (17)|
-------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("HZ_PARTIES"."CREATION_DATE"<TRUNC(SYSDATE@!))
```

A Rows oszlopban láthatjuk, hogy az optimalizáló becslése alapján a lekérdezés 4,5 millió sort fog eredményezni, a költség pedig 67248. Az egyetlen lényegi művelet a végrehajtási tervben a hz_parties táblára vonatkozó teljes tábla olvasás, amelyre egy szűrőfeltételt alkalmazandó végrehajtáskor.

Ha egy ajánlást (hint) teszünk a lekérdezés szövegébe arra vonatkozóan, hogy ha lehetséges,

akkor használja a hz_parties tábla HZ_PARTIES_N100 nevű indexét, akkor a következőképpen módosul a lekérdezés szövege és végrehajtási terve:

```
SELECT --+ index(hz_parties HZ_PARTIES_N100)
       party_name
  FROM hz_parties
 WHERE creation_date < TRUNC(sysdate);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation                    |  Name            | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |  4453K|   178M|  2179K  (2)|
|   1 |  TABLE ACCESS BY INDEX ROWID| HZ_PARTIES       |  4453K|   178M|  2179K  (2)|
|*  2 |   INDEX FULL SCAN            | HZ_PARTIES_N100  |  4453K|       | 21616  (11)|
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------

   2 - access("HZ_PARTIES"."CREATION_DATE"<TRUNC(SYSDATE@!))
       filter("HZ_PARTIES"."CREATION_DATE"<TRUNC(SYSDATE@!))
```

Láthatjuk, hogy az előzőhöz hasonlóan becslés szerint 4,5 millió sort fog eredményezni a lekérdezés, tehát csak a végrehajtás logikája változik. Észre is vehetjük, hogy az előbbi 67248-as költség index használata esetén több, mint 2 millióra (!!) növekedik. A teljes index olvasás csupán 21616 költségű, de mivel az index 4,5 millió sorazonosítót ad vissza, ezért a TABLE ACCESS BY INDEX ROWID – ami a sorazonosító alapján történő hozzáférést jelenti – művelet rendkívül költséges lesz, ami már nem tud versenybe szállni a teljes tábla olvasással.

Több tábla összekapcsolása

Egyszerre csak két táblát lehet összekapcsolni. Ha kettőnél többet szerenénk összekapcsolni, akkor az úgy fog történni, hogy először két táblát kapcsolunk össze a végrehajtás során, majd ennek az összekapcsolásnak az eredményhalmazát kapcsoljuk össze a harmadikkal és így tovább.

Összekapcsolási módok:

- Nested Loop
- Hash Join
- Sort-merge Join

Nested Loop

A Nested Loop összekapcsolási módot akkor érdemes használni, ha kis méretű adathalmazokat akarunk összekapcsolni és a kapcsolási feltétel segítségével gyorsan el tudjuk érni a második táblát. A két táblát megkülönböztetjük az alapján, hogy melyiket nevezzük ki külső, és melyiket belső táblának. A Nested Loop Join az egyszerű egymásba ágyazott ciklus logikát követi: A külső tábla minden sorára megnézzük, hogy a belső tábla valamely sora hozzákapcsolható-e.

Példa a Nested Loop-ra:

```
SELECT h.quote_number
     , s.update_allowed_flag
  FROM aso_quote_headers_all h
     , aso_quote_statuses_b s
 WHERE h.quote_status_id = s.quote_status_id
   AND s.status_code = 'FAXED';
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                    |  Name                       | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                             | 40139 |  1175K| 10085   (2)|
|   1 |  NESTED LOOPS                |                             | 40139 |  1175K| 10085   (2)|
|   2 |   TABLE ACCESS BY INDEX ROWID| ASO_QUOTE_STATUSES_B        |     1 |    20 |     2  (50)|
|*  3 |    INDEX UNIQUE SCAN         | ASO_QUOTE_STATUSES_B_U1     |     1 |       |            |
|   4 |   TABLE ACCESS BY INDEX ROWID| ASO_QUOTE_HEADERS_ALL       | 40139 |   391K| 10084   (2)|
|*  5 |    INDEX RANGE SCAN          | ASO_QUOTE_HEADERS_ALL_N6    | 40139 |       |            |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("S"."STATUS_CODE"='FAXED')
   5 - access("H"."QUOTE_STATUS_ID"="S"."QUOTE_STATUS_ID")
```

Hash Join

Akkor érdemes használni ezt az összekapcsolási módot, ha nagy méretű adathalmazokat kapcsolunk össze, vagy az adathalmazok nagy részét összekapcsoljuk. A Hash Join csak equijoin összekapcsolásoknál alkalmazható. A végrehajtás során egy hash tábla épül fel a kisebbik adathalmaz kapcsolási kulcsa alapján. A következő lépés, hogy bejárjuk a nagyobbik adathalmazt, majd a kapcsolási kulcs alapján megtaláljuk az összekapcsolt sorokat. Akkor érdemes Hash Join-t használni, ha a hash tábla elfér a memóriában. Ha ez mégsem így lenne, akkor a megnövekedő diszk I/O műveletek miatt nagy mértékben csökkenhet a hatékonyság.

Példa Hash Join-ra:

```
SELECT h.quote_number
     , s.update_allowed_flag
  FROM aso_quote_headers_all h
     , aso_quote_statuses_b s
 WHERE h.quote_status_id = s.quote_status_id;
```

```
-------------------------------------------------------------------------------
| Id  | Operation           |  Name                  | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                        |  1003K|    16M| 14459  (14)|
|*  1 |  HASH JOIN          |                        |  1003K|    16M| 14459  (14)|
|   2 |   TABLE ACCESS FULL | ASO_QUOTE_STATUSES_B   |    50 |   350 |     5  (20)|
|   3 |   TABLE ACCESS FULL | ASO_QUOTE_HEADERS_ALL  |  1003K|  9799K| 14227  (12)|
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("H"."QUOTE_STATUS_ID"="S"."QUOTE_STATUS_ID")
```

Sort-merge Join

Két műveletből áll ez az összekapcsolási mód. Először rendezzük (sort) mindkét adathalmazt, majd összeolvasztjuk (merge) a két adathalmazt a kapcsolófeltétel(ek) alapján. Általában a Hash Join hatékonyabb összekapcsolási mód, mint a Sort-merge Join, mert a rendezési művelet igen költséges. Ha a rendezést el tudjuk hagyni – tehát az adathalmazok már eleve rendezve vannak –, akkor hatékonyabb lehet a Sort-merge Join, mint a Hash Join. Non-equijoin esetében nem használhatunk Hash Join-t, ilyen esetekben sokszor még rendezési műveletekkel együtt is jobban megéri Sort-merge Joint választani.

Rendezések

Akkor kerül sor rendezésre, amikor a végrehajtás során szükség van arra, hogy valamilyen módon rendezett adathalmazzal rendelkezzünk. A rendezések általában erőforrásigényes műveletek, ezért a hatékony végrehajtás érdekében érdemes olyan SQL utasításokat írni, amelyben kevés rendezési művelet van. Ilyenek a DISTINCT, GROUP BY, UNION, MINUS, INTERSECT. Sokszor nem szükséges UNION-t használni halmazok összeillesztésére, megfelel a UNION ALL is, ami nem garantálja ugyan a sorok különbözőségét, de jelentősen gyorsíthatja a végrehajtást, mivel nem igényel rendezési műveletet. Indexek felhasználásával kiiktathatjuk a rendezési műveleteket, mivel az indexek eleve rendezett adathalmazt jelentenek. Fontos azonban tudni, hogy indexek használatával elveszthetjük a több blokkos I/O előnyeit.

A következő rendezési műveletekről beszélhetünk:

- Sort Unique
- Sort Aggregate
- Sort Group by
- Sort Join
- Sort Order by

Sort Unique

Egyediségre vonatkozó rendezési művelet. Akkor fog Sort Unique szerepelni a végrehajtási tervben, ha DISTINCT kulcsszóval előírjuk az egyediséget, vagy valamilyen művelet számára biztosítani kell az értékek egyediségét.

Példa a Sort Unique-ra:

```
SELECT DISTINCT created_by
  FROM aso_quote_statuses_b;
```

```
-------------------------------------------------------------------------------
| Id  | Operation           |  Name                | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                      |    10 |    40 |     6  (34)|
|   1 |  SORT UNIQUE        |                      |    10 |    40 |     6  (34)|
|   2 |   TABLE ACCESS FULL | ASO_QUOTE_STATUSES_B |    50 |   200 |     5  (20)|
-------------------------------------------------------------------------------
```

Sort Aggregate

Ez a művelet valójában nem igényel rendezést, csupán az összesítő műveleteket jelzi.

Példa Sort Aggregate műveletre:

```
SELECT MAX(creation_date), MIN(creation_date)
  FROM aso_quote_statuses_b;
```

```
-------------------------------------------------------------------------------
| Id  | Operation           |  Name                | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                      |     1 |     8 |     5  (20)|
|   1 |  SORT AGGREGATE     |                      |     1 |     8 |            |
|   2 |   TABLE ACCESS FULL | ASO_QUOTE_STATUSES_B |    50 |   400 |     5  (20)|
-------------------------------------------------------------------------------
```

Sort Group by

Erre a műveletre akkor van szükség amikor összesítő műveletet végzünk adatok több különböző csoportján. A rendezéshez el kell különíteni a sorokat különböző csoportokba.

Példa a Sort Group by műveletre:

```
SELECT update_allowed_flag
     , count(update_allowed_flag)
  FROM aso_quote_statuses_b
 GROUP BY update_allowed_flag;
```

```
-------------------------------------------------------------------------------
| Id  | Operation           |  Name                | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                      |     2 |     4 |     6  (34)|
|   1 |  SORT GROUP BY      |                      |     2 |     4 |     6  (34)|
|   2 |   TABLE ACCESS FULL | ASO_QUOTE_STATUSES_B |    50 |   100 |     5  (20)|
-------------------------------------------------------------------------------
```

Sort Join

Sort-Merge Join összekapcsolási mód esetén használja ezt a műveletet, ha az alap adathalmazt rendezni kell.

Sort Order by

Akkor kerül végrehajtásra ez a művelet, ha az eredményhalmaz sorait rendezni kell, és ezt a rendezést nem lehet megoldani indexek felhasználásával.

Példa a Sort Order by műveletre:

```
SELECT status_code
     , creation_date
  FROM aso_quote_statuses_b
 ORDER BY 1, 2;
```

```
-------------------------------------------------------------------------------
| Id  | Operation           |  Name                | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                      |    50 |  1050 |     6  (34)|
|   1 |  SORT ORDER BY      |                      |    50 |  1050 |     6  (34)|
|   2 |   TABLE ACCESS FULL | ASO_QUOTE_STATUSES_B |    50 |  1050 |     5  (20)|
-------------------------------------------------------------------------------
```

Nézzünk egy példát a rendezési műveletekkel kapcsolatban. Adott az a probléma, hogy ki szeretnénk listázni azokhoz a megrendelésekhez tartozó azonosítókat, amiket azon a napon rendeltek, amelyik napon az legfrissebb rendelést is feladták. A lekérdezés talán legegyszerűbb formája és a hozzátartozó végrehajtási terv így néz ki:

```
SELECT order_number
  FROM oe_order_headers_all
 WHERE TRUNC(ordered_date) = (SELECT MAX(TRUNC(ordered_date))
                              FROM oe_order_headers_all);
```

```
-------------------------------------------------------------------------------
| Id  | Operation           |  Name                | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                      | 16134 |   220K| 32287  (18)|
|*  1 |  TABLE ACCESS FULL  | OE_ORDER_HEADERS_ALL | 16134 |   220K| 32287  (18)|
|   2 |   SORT AGGREGATE    |                      |     1 |     8 |            |
|   3 |    TABLE ACCESS FULL| OE_ORDER_HEADERS_ALL |  1613K|    12M| 29373  (10)|
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
   1 - filter(TRUNC("OE_ORDER_HEADERS_ALL"."ORDERED_DATE")= (SELECT /*+ */
              MAX(TRUNC("OE_ORDER_HEADERS_ALL"."ORDERED_DATE")) FROM
              "ONT"."OE_ORDER_HEADERS_ALL" "OE_ORDER_HEADERS_ALL"))
```

Szembetűnő lehet, hogy a WHERE feltételben az ordered_date oszlopra a TRUNC függvényt hívjuk, ami azt eredményezi, hogy a rendszer nem képes az oszlopon lévő indexet használni. Érdemes leellenőrizni, hogy van-e index az adott oszlopon:

```
SELECT *
  FROM dba_ind_columns
 WHERE table_owner = 'ONT'
   AND table_name = 'OE_ORDER_HEADERS_ALL'
   AND column_name = 'ORDERED_DATE';

no rows selected
```

Mivel nincs index az ordered_date oszlopon, ezért nem szükséges átalakítanunk a lekérdezést úgy, hogy képes legyen a rendszer használni az indexet.
Az első dolog amit észrevehetünk a végrehajtási terv értelmezése közben, hogy két teljes tábla olvasás szerepel ugyanarra a táblára. Fel is lehet tenni a kérdést, hogy hogyan lehetne úgy átírni a lekérdezést, hogy ne legyen benne két teljes tábla olvasás? Mielőtt leírnék egy lehetséges megoldást, néhány szó az *analitikus függvények*ről. Az analitikus függvények összesített értéket számolnak sorok egy csoportja alapján. Abban különböznek az összesítő függvényektől, hogy minden csoportra több sort is visszaadhatnak. A sorok egy csoportját az analitikus függvényeknél ablaknak (window) nevezzük. Az ablakot az analitikus függvény utáni OVER kulcsszó után kell meghatározni. Az analitikus függvények végrehajtása az utolsó előtti lépés (az utolsó lépés az ORDER BY utasításrész). Emiatt analitikus függvényeket csak SELECT után, vagy ORDER BY utasításrészben használhatunk.

Nézzük meg tehát az előbbi SQL utasítással ekvivalens megoldást:

```
SELECT order_number
  FROM (SELECT order_number,
               DENSE_RANK() OVER(ORDER BY TRUNC(ordered_date)
                                    DESC NULLS LAST) rnk
          FROM oe_order_headers_all)
 WHERE rnk = 1;
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation               |  Name                | Rows  | Bytes |TempSpc| Cost (%CPU)|
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |                      |  1613K|    40M|       | 40793  (16)|
|*  1 |  VIEW                   |                      |  1613K|    40M|       |            |
|*  2 |   WINDOW SORT PUSHED RANK|                     |  1613K|    21M|    74M| 40793  (16)|
|   3 |    TABLE ACCESS FULL    | OE_ORDER_HEADERS_ALL |  1613K|    21M|       | 29373  (10)|
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
   1 - filter("from$_subquery$_001"."RNK"=1)
   2 - filter(DENSE_RANK() OVER ( ORDER BY TRUNC("OE_ORDER_HEADERS_ALL"."ORDERED_DATE") DESC
              NULLS LAST )<=1)
```

Amint láthatjuk, ez a lekérdezés egyetlen teljes tábla olvasást igényel. A teljes tábla olvasás után egy WINDOW SORT művelet következik, ami az analitikus függvény meghívását jelenti. A DENSE_RANK() analitikus függvény az OVER kulcsszó után megadott window alapján minden sorhoz hozzárendel egy sorszámot. A sorszámozás 1-től indul, és mindig eggyel nő a sorszám, ha az előző sorban lévő kifejezéshez képest – itt TRUNC(ordered_date) – az aktuális sorban lévő kifejezés eltér. Tehát az összes olyan dátum, amelyik azonos, ugyanazt a sorszámot is fogja kapni ebben a lekérdezésben. Mivel csak a legkésőbbi dátumhoz tartozó megrendelésszámokra vagyunk kíváncsiak, ezért – az allekérdezést egy inline nézetként használva – a WHERE feltételben az 1-es sorszámra szűrünk.

A költség oszlopban 40793 szerepel, szemben az eredeti lekérdezés 32287-es költségével. Ezek alapján azt gondolhatnánk, hogy rosszabb lesz a teljesítménye a későbbi lekérdezésnek. Mivel ezek becsült adatok, jó lenne kipróbálni vajon tényleg lassabb az Oracle által nagyobb költségűnek ítélt lekérdezés? Az első lekérdezés autotrace-szel és beállított időzítővel a következő eredményt adta:

```
Elapsed: 00:00:44.32
Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
     184086  consistent gets
     181120  physical reads
          0  redo size
        238  bytes sent via SQL*Net to client
        279  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

A második lekérdezés eredménye:

```
Elapsed: 00:00:34.37
Statistics
----------------------------------------------------------
          5  recursive calls
          4  db block gets
      92049  consistent gets
      91659  physical reads
          0  redo size
        238  bytes sent via SQL*Net to client
        279  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          2  sorts (memory)
          1  sorts (disk)
          2  rows processed
```

Az első lekérdezés végrehajtása tehát 44 másodpercig, míg a második lekérdezés 34 másodpercig tartott. Az első végrehajtása során 181120 fizikai és 184086 logikai olvasás történt, a másodiknál 91659 fizikai és 92049 logikai. Az első művelet nem igényel rendezést, míg a második műveletnél 3 rendezési művelet is végrehajtódik. A v$sql_plan_statistics_all dinamikus performancia nézetet felhasználva bővebb információkhoz juthatunk. Ahogy a végrehajtási tervben láthattuk a TempSpc oszlopban 74MB ideiglenes szegmensbeli lemez felhasználást becsült az optimalizáló, a valójában felhasznált mennyiség pedig a v$sql_plan_statistics_all last_tempseg_size oszlopából kiolvasható, 128 byte, ami 3 fizikai írás műveletet jelent (last_disk_writes oszlop). A WINDOW SORT műveletnél a last_execution oszlopban 1 PASS szerepel, nem pedig OPTIMAL. Akkor lenne OPTIMAL a végrehajtás, ha az egész sorhalmazt sikerült volna csak a memóriában rendezni. Mivel lemezre is kellett írni a művelet során – amely egy ideiglenes szegmens használatát jelenti – ezért látható 1 PASS a végrehajtásnál. Lehetséges még MULTIPASS végrehajtás is, ami azt jelenti, hogy többszöri lemezfelhasználás szükséges a kevés SQL munkaterület miatt. Ha indokolttá válik, akkor a SORT_AREA_SIZE nevű paraméterrel tudjuk beállítani, amelynek az értékét byte-ban kell megadni, és a rendezésre szánt SQL munkaterület méretét határozza meg az SGA-n belül. Ha rendezési művelet kerül végrehajtásra, akkor ezen a területen fog megtörténni. Ha a rendezéshez szükséges hely mérete meghaladja a paraméterben megadott méretet, akkor a SORT_AREA_SIZE - szükséges byte-nyi ideiglenes szegmenst fog igénybe venni a végrehajtás. Az optimalizáló ezt a paramétert is figyelembe veszi a költségszámításnál, ha rendezési művelet, vagy sor-merge join is szerepel a végrehajtási tervben. Nagyobb SORT_AREA_SIZE paraméter kevesebb költséget fog eredményezni. Az Oracle csak akkor ajánlja ennek a paraméternek a használatát, ha a szerver osztott módban (shared server) működik. Helyette használhatjuk az automatikus SQL munkaterület méretezést úgy, hogy beállítjuk PGA_AGGREGATE_TARGET paramétert.

Hogyan hajtja végre az optimalizáló a műveleteket?

Amennyiben lehetőség van rá az optimalizáló teljesen kiértékeli a kifejezéseket, és átalakíthatja egy szemantikailag megegyező formába. Ez a művelet akkor történhet meg, ha például az átalakított kifejezés gyorsabban kiértékelhető, mint az eredeti.

IN-List Iterator

Az optimalizáló használhatja az IN-List Iterator műveletet, ha értékekkel használunk IN operátort az utasításban. Tulajdonképpen az In-List Iterator az IN-ben szereplő összes értékre végrehajtja a végrehajtási tervben utána következő műveletet.

Akkor érdemes az IN-List Iterator-t használni, ha egy létezik egy !!!!!! szelektivitású index az adott oszlophoz. Ha több kifejezés is szerepel az utasításban OR logikai operátorral összekapcsolva ugyanarra az indexre vonatkozóan, akkor az optimalizáló ezt a műveletet használja a CONCATENATION, vagy UNION ALL helyett.

Példa az IN-List Iterator műveletre:

```
SELECT order_number
  FROM oe_order_headers_all
 WHERE cust_po_number IN (:b1, :b2);
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    |  Name                | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                      |    10 |   190 |    10  (10)|
|   1 |  INLIST ITERATOR             |                      |       |       |            |
|   2 |   TABLE ACCESS BY INDEX ROWID| OE_ORDER_HEADERS_ALL |    10 |   190 |    10  (10)|
|*  3 |    INDEX RANGE SCAN          | OE_ORDER_HEADERS_N5  |    10 |       |     2  (50)|
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("OE_ORDER_HEADERS_ALL"."CUST_PO_NUMBER"=:Z OR
              "OE_ORDER_HEADERS_ALL"."CUST_PO_NUMBER"=:Z)
```

Concatenation

A Concatenation művelet olyan műveletek esetén hasznos, amelyekben különböző (nem összekapcsolódó) kifejezések vannak OR logikai operátorral összekapcsolva. A Concatenation az utána következő gyerek műveletek eredményeit összefűzi, és kiegészítő logikai kifejezésekkel biztosítja, hogy ne szerepeljenek duplikált sorok.

Mikor nem érdemes Concatenation műveletet használni?

- Ha az OR-ral összekapcsolt logikai kifejezések ugyanarra az oszlopra vonatkoznak és használható az IN-List Iterator helyette.
- Ha költséges műveleteket kell végrehajtani a Concatenation-t követő lépésekben.

Példa a Concatenation műveletre:

```
SELECT creation_date
  FROM oe_order_headers_all
 WHERE cust_po_number = :b1
    OR order_number = :b2;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                    |  Name                 | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                       |     2 |    54 |    13  (16)|
|   1 |  CONCATENATION               |                       |       |       |            |
|   2 |   TABLE ACCESS BY INDEX ROWID| OE_ORDER_HEADERS_ALL  |     1 |    27 |     8  (13)|
|*  3 |    INDEX RANGE SCAN          | OE_ORDER_HEADERS_U2   |     5 |       |     4  (25)|
|*  4 |   TABLE ACCESS BY INDEX ROWID| OE_ORDER_HEADERS_ALL  |     1 |    27 |     8  (13)|
|*  5 |    INDEX RANGE SCAN          | OE_ORDER_HEADERS_N5   |     5 |       |     4  (25)|
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("OE_ORDER_HEADERS_ALL"."ORDER_NUMBER"=TO_NUMBER(:Z))
   4 - filter(LNNVL("OE_ORDER_HEADERS_ALL"."ORDER_NUMBER"=TO_NUMBER(:Z)))
   5 - access("OE_ORDER_HEADERS_ALL"."CUST_PO_NUMBER"=:Z)
```

Tranzitivitás

Az optimalizáló néhány esetben felfedezhet tranzitivitási feltételeket két logikai kifejezés vonatkozásában.

Ha olyan feltételt talál az optimalizáló, mint a következő

```
WHERE oszlop1 = konstans
  AND oszlop1 = oszlop2
```

akkor az oszlop2 = konstans logikai kifejezéssel kiegészíti a feltételeket. A tranzitivitást csak olyan esetekben cseréli ki az optimalizáló, amikor konstans szerepel az egyik oldalon.

Példa a tranzitivitásra:

```
SELECT h.creation_date
  FROM oe_order_headers_all h
     , oe_order_lines_all l
 WHERE h.header_id = l.header_id
   AND h.header_id = :b1;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                    |  Name                | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                      |     1 |    20 |     4  (25)|
|   1 |  NESTED LOOPS                |                      |     1 |    20 |     4  (25)|
|   2 |   TABLE ACCESS BY INDEX ROWID| OE_ORDER_HEADERS_ALL |     1 |    14 |     4  (25)|
|*  3 |    INDEX UNIQUE SCAN         | OE_ORDER_HEADERS_U1  |     1 |       |     3  (34)|
|*  4 |   INDEX RANGE SCAN           | OE_ORDER_LINES_N1    |     1 |     6 |            |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("H"."HEADER_ID"=TO_NUMBER(:Z))
   4 - access("L"."HEADER_ID"=TO_NUMBER(:Z))
       filter("H"."HEADER_ID"="L"."HEADER_ID")
```

Statisztikák

Az optimalizálónak szüksége van statisztikákra ahhoz, hogy megfelelő döntést tudjon hozni, vagyis megfelelő (optimális) végrehajtási tervet kapjunk. A hiányzó statisztikai adatokat az Oracle alapértelmezésekkel helyettesíti. Az ebben a részben említett statisztikák az adatszótárban tárolódnak (DBA_*, vagy USER_* nézetek). Ezek nem keverendőek össze a dinamikus performancia nézetekkel, amelyek a futás közbeni statisztikákat mutatják a rendszerről.

Nézzük meg milyen statisztikákat gyűjt az adatbázis-kezelő rendszerünk:

- Tábla statisztikák
    - Sorok száma (számosság – cardinality)
    - Blokkok száma
    - Átlagos sorhosszúság
- Oszlop statisztikák
    - Egyedi értékek száma (egyediségi számosság – distinct cardinality)
    - NULL értékek száma az adott oszlopban
    - Eloszlás (hisztogram)
    - Kiterjesztett statisztikák – Olyan statisztikákat jelent, amelyek túlmutatnak az alapvető statisztikákon, ide tartoznak a többoszlopos együttes statisztikák, illetve a kifejezések statisztikái.
- Index statisztikák
    - Levélblokkok száma
    - Szintek száma
    - Fürtözési tényező (Clustering factor) – Egy indexelt tábla indexének azon tulajdonsága, amely megmutatja, hogy az indexelt tulajdonság szempontjából a tábla összetartozó sorai hány blokkban helyezkednek el. A kisebb fürtözési tényező azt jelzi, hogy az összetartozó sorok kevesebb blokkban vannak – ez hatékonyabb indexhasználatot mutat, mivel kisebb fürtözési tényező esetében kevesebb diszk I/O szükséges.

- Rendszer statisztikák
    - I/O hatékonysága és kihasználása
    - CPU hatékonysága és kihasználása

Mivel az adatbázis objektumok folyamatosan változnak, így ezeket a statisztikákat is rendszeresen frissíteni kell ahhoz, hogy kellően pontosan leírják az objektumokat. A statisztikákat automatikusan karban tudja tartani az Oracle, de lehetőséget az ad a kézi karbantartásra is a DBMS_STATS csomaggal. A DBMS_STATS csomag nem csak a statisztikák gyűjtésére használható, de exportálhatjuk is azokat, illetve importálhatjuk is őket. Ez jól jöhet, ha egy éles rendszeren lévő statisztikákat szeretnénk átmozgatni egy teszt rendszerre, megteremtve ezzel az optimalizáló számára a hasonló környezetet.

A statisztikák gyűjtése elsősorban az adatokból vett minták alapján történik. Ha ez nem így lenne, akkor teljes tábla- , index olvasásra lenne szükség, ami meglehetősen erőforrásigényes. A minta méretét kézzel is beállíthatjuk, de érdemes az Oracle-re bízni a minta méretezését.

Oszlop statisztikák és a hisztogram

A legalapvetőbb statisztikus információk egy adott oszlopban lévő értékek eloszlásáról a legkisebb és a legnagyobb érték. Ez természetesen nem elegendő olyan esetekben, amikor ferde az eloszlás. Ha ilyen helyzettel állunk szemben, akkor az egyszerű oszlopstatisztikák mellett érdemes hisztogramot is készíteni a jobb eloszlás meghatározásához. Beállíthatjuk azt is, hogy az Oracle automatikusan kezelje a hisztogramokat, vagyis azokhoz az oszlopokhoz automatikusan hoz létre hisztogramot, ahol ferde az eloszlás.

A hisztogram pontos becslést nyújt az oszlopban lévő adatok eloszlásáról. Pontosabb becslést kaphat az optimalizáló a szelektivitást illetően, így optimális végrehajtási tervet tud készíteni a nem egyenletes, ferde eloszlású oszlopokra is.

Az Oracle két hisztogramfajtát különít el:

- Sűrűséghisztogram (Height-balanced histogram)

- Gyakorisági hisztogram (Frequency histogram)

Egy-egy oszlopnál a használt hisztogram típusa a DBA_TAB_COL_STATISTICS, vagy USER_TAB_COL_STATISTICS adatszótárnézetek HISTOGRAM oszlopából tudható meg. Ennek az oszlopnak az értéke lehet HEIGHT BALANCED, FREQUENCY, vagy NONE. Ez az oszlop csak 10g verziótól érhető el, korábbi verzióknál használhatjuk a következő lekérdezést a hisztogram típusának megállapításához:

```
SELECT CASE
          WHEN NVL (h.row_cnt, 0) = 0
             THEN 'NONE'
          WHEN (h.bucket_cnt > 255
                OR (    h.bucket_cnt > h.distcnt
                    AND h.row_cnt = h.distcnt
                    AND h.density * h.bucket_cnt * 2 <= 1
                   )
               )
             THEN 'FREQUENCY'
          ELSE 'HEIGHT BALANCED'
       END histogram_type
     , o.name object_name
     , c.name column_name
  FROM sys.hist_head$ h
     , sys.obj$ o
     , sys.col$ c
 WHERE h.obj# = o.obj#
   AND h.col# = c.col#
   AND o.obj# = c.obj#
   AND o.name = '&objname';
```

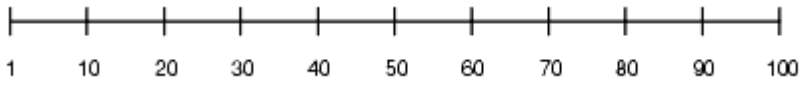
Érdekesség, hogy 10.2.0.4-es verziótól a kettes szorzót kivették a feltételből, 11g Release 1-től pedig a kisebb egyenlő helyett csak kisebb relációs jel van.

Magát a hisztogramot, illetve annak adatait a DBA_TAB_HISTOGRAMS, illetve a USER_TAB_HISTOGRAMS nézeteken keresztül érhetjük el.

Sűrűséghisztogram

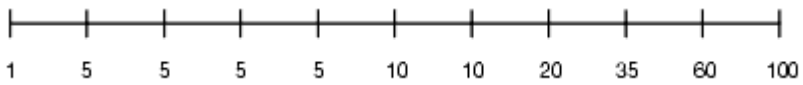
Ebben a hisztogramtípusban az oszlopértékek sávokra vannak osztva, minden sávban ugyanannyi érték van. Ekkor tehát a hasznos információ, amit a hisztogram szolgáltat, hogy hova esik az értékek tartományában a végpont (endpoint).

Egyenletes eloszlásnál ha az értékek 1 és 100 közé esnek és 10 részre bontjuk ezt a tartományt, akkor így nézhet ki a hisztogram:



Mind a 10 tartományban az összes oszlopérték egytizede szerepel.

Ha az adatok nem egyenletes eloszlásúak, akkor a hisztogram például így nézhet ki:



Ebben az esetben láthatóan a legtöbb oszlopérték 5-tel egyenlő.

Például az aso_quote_headers tábla cust_account_id oszlopának sűrűséghisztogramja a következőképpen néz ki:

```
SQL> SELECT endpoint_number
  2       , endpoint_value
  3    FROM dba_histograms
  4   WHERE table_name = 'ASO_QUOTE_HEADERS_ALL'
  5     AND column_name = 'CUST_ACCOUNT_ID';
```

Az eredmény (részlet):

```
ENDPOINT_NUMBER ENDPOINT_VALUE
--------------- --------------
              0           1000
              1           1048
              2           1125
              3           1230
              4           1260
              6           1304
```

A kimenet azt mutatja, hogy az első tartomány (sáv) 1000 és 1048 közé esik, a második tartomány 1048 és 1125 közé, és így tovább. Minden tartományban körülbelül ugyanannyi érték van. Azt, hogy egy adott sávban hány elem szerepel, a sávok maximális száma szabályozza.

Gyakorisági hisztogram

Gyakorisági hisztogram esetében minden oszlopérték külön tartományt (végpontot – endpoint) jelöl. Minden ilyen végponthoz tartozó érték pedig azt tartalmazza, hogy hányszor fordul elő az adott érték az oszlopban. Automatikusan gyakorisági hisztogram jön létre az adott oszlophoz, ha az oszlop egyediségi számossága kisebb vagy egyenlő a megadott maximális tartományok számával.

Például az aso_quote_headers tábla quote_status_id oszlopának gyakorisági hisztogramja a következőképpen néz ki:

```
SQL> SELECT endpoint_number
  2       , endpoint_value
  3    FROM dba_histograms
  4   WHERE table_name = 'ASO_QUOTE_HEADERS_ALL'
  5     AND column_name = 'QUOTE_STATUS_ID';
```

A kimenet pedig a következő (részlet):

```
ENDPOINT_NUMBER ENDPOINT_VALUE
--------------- --------------
           1089          10000
          20325          10002
          22763          10005
          28352          10009
          28975          10010
          28977          10013
          28985          10016
          28997          10017
```

Az endpoint_value oszlop mutatja a quote_status_id oszlopban szereplő értékeket, az endpoint_number oszlop pedig a gyakoriságát tartalmazza az adott értéknek.

Rendszerstatisztikák

Kétféleképpen lehet rendszerstatisztikákat gyűjteni. Az első, ha az Oracle véletlenszerűen olvasási műveleteket hajt végre minden adatfájlon, és ez alapján határozza meg a rendszerstatisztikákat (**noworkload**). A másik fajta statisztikagyűjtési módszer pedig nem hajt végre külön olvasási műveleteket, hanem az adatbázisban végrehajtott normál olvasásokat használja egyben statisztikák gyűjtésére is (**workload**).

A következő táblázat mutatja a rendszerstatisztikákat:

Paraméter neve	Leírás	Kezdőérték	Egység
cpuspeedNW	A noworkload processzorsebességet reprezentálja.	Rendszerinduláskor határozódik meg.	millió ütés / másodperc
ioseektim	Az I/O keresési idő mérőszám a keresési időből, a várakozási időből és az operációs rendszer többletmunkájából adódó időből tevődik össze.	Rendszerinduláskor határozódik meg. Az alapértelmezett érték 10.	ms

iotfrspeed	Az I/O átviteli sebesség azt jelzi, hogy az Oracle mennyi adatot tud beolvasni egyetlen olvasási kéréssel.	Rendszerinduláskor határozódik meg. Az alapértelmezett érték 4096.	byte / ms
cpuspeed	A workload processzorsebességet reprezentálja.	Nincs	millió ütés / másodperc
maxthr	A maximális I/O áteresztőképesség mérőszám a maximális I/O áteresztőképességet reprezentálja, amit az I/O alrendszer szolgáltatni tud.	Nincs	byte / sec.
slavethr	A slave I/O throughput az átlagos parallel slave I/O áteresztőképességet mutatja.	Nincs	Byte / sec.
sreadtim	A single block read time különböző blokkolvasásokból kinyert átlagos válaszidő.	Nincs	ms
mreadtim	A multiblock read time különböző több blokkos olvasások esetén fennálló átlagos válaszidő.	Nincs	ms
mbrc	A multiblock read count a több blokkos olvasásnál beolvasott blokkok száma átlagolva.	Nincs	blokk

A cpuspeedNW, ioseektim, iotfrspeed mérőszámok a noworkload statisztikák, a cpuspped, maxthr, slavethr, sreadtim, mreadtim és mbrc pedig workload statisztikák. Ha mindkétfajta statisztika rendelkezésre áll, akkor az optimalizáló a workload statisztikákat használja fel.

Nyomkövetés

Az Oracle számos eszközt kínál arra, hogy az adatbázis működését nyomon tudjuk követni (**trace**). Ezekből az eszközökből én most az SQL trace technikákat, illetve néhány olyan eszközt fogok bemutatni, amely a nyomkövetést szolgálja.

SQL Trace

Az SQL nyomkövetés hatékonysági információkat szolgáltat a különböző végrehajtott SQL-ekről. A generált információhalmaz a következőkből tevődik össze:

- Parse, Execute, Fetch műveletek száma
- Felhasznált idő
- Fizikai és logikai olvasások
- Feldolgozott sorok száma
- Library Cache információk
- Annak a felhasználónak a felhasználói neve, aki az adott SQL parszolást kérte
- Minden COMMIT és ROLLBACK
- Várakozási események és azok időtartama

A nyomkövetést többféle hatókörre lehet engedélyezni:

- Munkamenet szintű nyomkövetés
- Adatbázis példány szintű nyomkövetés

Ha engedélyezve van a nyomkövetés egy adott munkamenetre, akkor az Oracle egy nyomkövetési fájlt generál, amely tartalmazza a statisztikus információkat a nyomkövetett SQL-ről. Ha a nyomkövetés az egész példány számára be van kapcsolva, akkor az Oracle minden folyamathoz külön nyomkövetési fájlt készít.

Mielőtt engedélyeznénk a nyomkövetést, érdemes leellenőrizni a következő paramétereket:

- TIMED_STATISTICS – ennek a paraméternek a ki-, illetve bekapcsolásával tilthatjuk le, illetve engedélyezhetjük az időzített statisztikák gyűjtését – ilyen például a felhasznált processzoridő, illetve az eltelt idők számítása. Ez egy dinamikus paraméter, munkamenet szinten is beállítható.
- MAX_DUMP_FILE_SIZE – a maximális nyomkövetési fájl méretet határozhatjuk meg ezzel a paraméterrel, operációs rendszer blokkokban. Az alapértelmezett értéke 500. Ez szintén egy dinamikus paraméter, így munkamenet szinten is beállítható.
- USER_DUMP_DEST – Egy abszolút operációs rendszer elérési utat tartalmaz ez a paraméter, amely azt határozza meg, hogy a nyomkövetési fájlok hova kerüljenek elhelyezésre a fájlrendszerben. Ez a paraméter is beállítható munkamenet szinten.

A következő lekérdezéssel megtudhatjuk, hogy hova fogja menteni, illetve mi lesz a nyomkövetési fájl neve egy adott munkameneten belül:

```
SELECT    u_dump.VALUE
       || '/'
       || db_name.VALUE
       || '_ora_'
       || v$process.spid
       || NVL2 (v$process.traceid, '_' || v$process.traceid, NULL)
       || '.trc' "Trace File"
  FROM v$parameter u_dump
       CROSS JOIN v$parameter db_name
       CROSS JOIN v$process
       JOIN v$session ON v$process.addr = v$session.paddr
 WHERE u_dump.NAME = 'user_dump_dest'
   AND db_name.NAME = 'db_name'
   AND v$session.audsid = SYS_CONTEXT ('userenv', 'sessionid');
```

A nyomkövetésnek különböző szintjei (level) léteznek:

- 0 – A nyomkövetés ki van kapcsolva
- 1 – Bekapcsolt nyomkövetés

- 4 – 1-es szint, plusz a gazdakörnyezet változóiról (bind variable) is tartalmaz információkat
- 8 – 1-es szint, plusz a várakozási eseményekről (wait events) is tartalmaz információkat
- 12 – 1-es szint, plusz a gazdakörnyezet változóiról és a várakozási eseményekről is tartalmaz információkat.

Bekapcsolhatjuk a nyomkövetést az adott munkameneten belül az alábbi módszerek valamelyikével:

- A DBMS_MONITOR nevű csomag DATABASE_TRACE_ENABLE nevű eljárásával
- A DBMS_SESSION nevű csomag SET_SQL_TRACE nevű eljárásával
- Az ALTER SESSION SET SQL_TRACE = TRUE; utasítással (1-es szintű nyomkövetés)
- Az ALTER SESSION SET EVENTS='…'; utasítással, ahol az events paraméterben megadhatjuk, hogy milyen eseményeket szeretnénk figyelni
- A DBMS_SYSTEM nevű csomag SET_EV nevű eljárásával.

A nyomkövetés automatikusan kikapcsol, ha az adott munkamenet befejeződik, vagy a munkamenet közben kiadjuk az ALTER SESSION SET SQL_TRACE = FALSE, esetleg az ALTER SESSION SET EVENTS = '…' utasítást, ahol az events paraméterben a nyomkövetés kikapcsolását állítjuk be.

Többféle nyomkövetést állíthatunk be, ha kézzel állítjuk be az eseményeket az ALTER SESSION SET EVENTS vagy a DBMS_SYSTEM.SET_EV eljárással. Az eddigiekben az SQL nyomkövetésről volt szó, ez a 10046-os esemény. Lehetőségünk van azonban számos különböző diagnosztikai és nyomkövetési eseményt beállítani.

Egy példa nyomkövetési állomány elérhető a függelékben.

TKPROF

A TKPROF egy parancssori alkalmazás, amely könnyebben olvasható formába tudja hozni a nyomkövetéskor generált fájlok tartalmát. Ez az eszköz bemenetként tehát egy, vagy több

nyomkövetési fájlt vár, és egy, vagy több új szöveges állományt állít elő. Parancssori argumentumokkal szabályozhatjuk, hogy milyen elemeket generáljon a kimeneti állományba.

A TRCSESS nevű parancssoros eszközzel több nyomkövetési állományt egyesíthetünk, majd a TKPROF eszközt használva egy összesített eredményt kaphatunk több munkamenetről.

A TKPROF kimenetében nem szerepelnek a COMMIT, illetve ROLLBACK utasítások.

A kimenetben például ilyen táblázatokat láthatunk:

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1    100.00      42.78          0          0          0           0
Execute      1      0.00       1.17          0          0          0           0
Fetch     2831 151500.00  428250.88     153855     154017          0       42449
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     2833 151600.00  428294.83     153855     154017          0       42449
```

A táblázat sorai

- *Parse* – A parszolást jelenti, amely eredményeképpen előáll a végrehajtási terv. A művelet magába foglalja a biztonsági ellenőrzéseket, a hivatkozott táblák, oszlopok és egyéb adatbázisbeli objektumok létezésének ellenőrzését.
- *Execute* – Az adott utasítás végrehajtását jelenti.
- *Fetch* – A fetch művelet –, amely csak lekérdezéseknél értelmezendő – néhány sor lekérdezését jelenti a kurzorból.

A táblázat oszlopai

- *Count* – Megmutatja, hogy az utasításon hányszor hajtódott verge az adott művelet (parse, fetch, vagy execute).
- *CPU* – Az adott művelettel eltöltött összes CPU idő másodpercben. Ez az oszlop nullát tartalmaz, ha a TIMED_STATISTICS paraméter nincs bekapcsolva.
- *Elapsed* – Az adott művelettel eltöltött összes idő másodpercben. Ez az oszlop nullát tartalmaz, ha a TIMED_STATISTICS paraméter nincs bekapcsolva.
- *Disk* – Megmutatja, hogy az adott műveleten belül hány fizikai olvasás hajtódott végre, vagyis hány adatblokkot olvasott be összesen a rendszer.
- *Query* – Megmutatja, hogy az adott műveleten belül hány konzisztens módú olvasás történt. Általában lekérdezéseknél történő logikai olvasás hajtódik végre konzisztens módban.

- *Current* – Megmutatja, hogy az adott műveleten belül hány current módú olvasás történt. Current módú olvasás az INSERT, DELETE és UPDATE műveletekre jellemző.
- *Rows* – Az adott lekérdezés összes feldolgozott sorát mutatja. Nem tartalmazza viszont azokat a sorokat, amelyek az SQL utasítás allekérdezésében kerültek feldolgozásra.

Lekérdezések esetében a feldolgozott sorok száma a Fetch sorban látható, míg a DML utasítások esetében az Execute sorban.

Rekurzív hívások (recursive calls)

Néhány esetben ahhoz, hogy egy SQL utasítást végre tudjunk hajtani, az Oracle-nek további SQL utasításokat kell végrehajtania. Az ilyen szükséges plusz utasításokat rekurzív hívásoknak nevezzük. Amikor egy lekérdezést akarunk futtatni, akkor a különböző ellenőrzések szintén ilyen rekurzív hívások formájában jelennek meg, ha a szükséges adatok nincsenek benne a data dictionary cache-ben.

Ha a nyomkövetés be van kapcsolva, akkor az Oracle a nyomkövetési állományba fogja helyezni a rekurzív hívásokkal kapcsolatos információkat. Ezeket a TKPROF segítségével megjeleníthetjük, de akár ki is hagyhatjuk a kimenetből.

Library Cache misses

A Library Cache miss azt jelenti, hogy az SQL utasítás – parszolt formában – nem található meg a library cache memóriaterületen. Ekkor az Oracle-nek parszolnia kell az SQL-t.

A TKPROF kimenetében ez az üzenet a táblázatok alatt egy külön sorban jelenik meg:

```
Misses in library cache during parse: 1
```

Végrehajtási terv TKPROF-ban

A végrehajtási tervet is megjeleníthetjük a TKPROF kimenetben, ezzel is segítve a könnyebb elemzést. Ennek a módja, hogy az EXPLAIN paramétert hozzáadjuk a TKPROF argumentumlistájához. A végrehajtási terv műveletei mellett egy külön oszlopban megjeleníti az adott lépésben feldolgozott sorok számát. Mikor elindítjuk a TKPROF-ot, akkor az csatlakozik az adatbázis példányhoz, majd lekérdezi minden a nyomkövetési állományban szereplő SQL utasítás végrehajtási tervét, majd a végrehajtási tervet összerendelni a sorforrás műveletekkel, és ezt az összerendelt műveletsort fogja szerepeltetni a kimenetben. Tudnunk kell azonban, hogy ez a végrehajtási terv nem feltétlenül az a végrehajtási terv, aminek végrehajtása során a kimenetben szereplő statisztikákat kaptuk.

Egy példa TKPROF kimenet megtalálható a függelékben.

Összefoglalás

Az SQL hangolás alapját az optimalizáló beható ismerete jelenti, így célszerűen a téma körbejárását annak működésével, valamint komponenseivel kezdtem, úgy mint:

- Query Tranformer, azaz lekérdezés átalakító
- Estimator, vagyis becslő
- Plan generator - végrehajtási terv generátor.

A lekérdezés átalakító egy már értelmezett lekérdezést kap meg, és ezen végzi el a transzformációkat, amely a végrehajtás hatékonyságának növelését segítheti elő. Az átalakítóhoz kapcsolódóan még áttekintettük a különböző átalakítási technikákat.

A becslő célja, hogy a végrehajtandó utasításra vonatkozóan a szelektivitás, számosság, költség mérőszámait becsülje meg. Láthattuk, hogy ezek a számok egymástól nem függetlenek, erős összefüggés van közöttük. A becslések elvégzésének célja, a végső futtatás absztrakt költségét megfelelően meghatározza.

Az optimalizáló témakörében utolsóként betekintettünk a végrehajtási terv generátor komponens működésébe, valamint a műveleteibe. A komponens működés közben számos generált terv közül választja ki az optimálisnak vélt tervet.

Általában véve egy alkalmazás hangolása során szükség van a hangolás céljának meghatározására: ez többnyire vagy a válaszidő csökkentése, vagy az áteresztőképesség növelése. A cél meghatározását inicializációs paraméter vagy hintek segítségével befolyásolhatjuk. Az egyik legáltalánosabb a CHOOSE, amely az optimalizálóra bízza a döntést.

Ezt követően a végrehajtási tervről, és annak szerkezetéről esett szó, valamint hogyan lehet helyesen értelmezni. Leírtam a hozzáférési feltétel, és a szűrőfeltétel szerepét a tervben, majd a VSQL_PLAN, VSQLAREA dinamikus performancia nézetek funkcióit, szerkezetüket.

Az EXPLAIN PLAN utasítás működését egy példán keresztül szemléltettem.

A következő témakör a hozzáférés módja volt, amelyben sorra kielemeztem a legfontosabb hozzáférési módokat, mindegyikre egy-egy példát mutatva:

- Full table scan
- Index scan
- Rowid scan

A hozzáférés módja után a táblák összekapcsolási módjait vettem górcső alá: nested loop, hash join, sort-merge join. A példákból és a hozzájuk fűződő magyarázatokból egyértelműen kikristályosodott az egyes összekapcsolások felhasználási lehetőségei.

A rendezések tárgyalása során az eddig bevált példákkal megtámogatott vizsgálódást folytattam.

A következő fontos témakört a statisztikák adták. Bemutattam, valamint elemeztem a különböző típusait, valamint néhány példával is szolgáltam a könnyebb érthetőséget.

Végül, de nem utolsó sorban a nyomkövetésről, a nyomkövetési technikákról, a nyomkövetési állományból kinyerhető információkról ejtettem néhány szót. Néhány eszköz példáján keresztül világítottam meg milyen hasznos lehet a nyomkövetés a hangolás folyamata során.

Napjaink óriási rendszereiben a tárolt és ez által a feldolgozandó adatok mennyisége rohamos ütemben növekszik, így a hangolás rendkívül nagy szerepet kap. Úgy gondolom, hogy minden Oracle adatbázis környezetben tevékenykedő fejlesztőnek, adatbázis üzemeltetőnek szüksége van arra a tudásra, amelyet ez a szakdolgozat reprezentál. Ennek segítségével jobban megismerheti az általa használt rendszer működését, szerkezetét. Úgy érzem a szakdolgozatommal hasznos és a gyakorlatban is használható betekintést sikerült nyújtanom az SQL hangolásba Oracle környezetben. A dolgozatnak ismeretanyagának elsajátítása után az érdeklődőek egyénileg is bátran belefoghatnak haladóbb dokumentációk feldolgozásába. Remélem dolgozatom elnyerte tetszését és joggal színesíti a magyar nyelvű SQL hangolás témakörben íródott anyagok listáját.

Irodalomjegyzék

[1] Oracle9*i* Database Performance Tuning Guide and Reference, Release 2 (9.2)
http://download.oracle.com/docs/cd/B10501_01/server.920/a96533/toc.htm

[2] Oracle® Database Performance Tuning Guide, 11*g* Release 1 (11.1)
http://download.oracle.com/docs/cd/B28359_01/server.111/b28274/toc.htm

[3] Oracle® Database Concepts, 11*g* Release 1 (11.1)
http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/toc.htm

[4] Oracle® Database Reference, 11g Release 1 (11.1)
http://download.oracle.com/docs/cd/B28359_01/server.111/b28320/toc.htm

[5] Oracle® Database SQL Language Reference, 11*g* Release 1 (11.1)
http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/toc.htm

[6] Understanding Optimization, Kimberly Floss
http://www.oracle.com/technology/oramag/oracle/05-jan/o15tech_tuning.html

[7] Oracle Histograms, Yong Huang
http://yong321.freeshell.org/oranotes/Histogram.html

[8] On reading trace files with PL/SQL, René Nyffenegger
http://www.adp-gmbh.ch/blog/2006/02/05.php

[9] Oracle Internals – Events, Julian Dyke
http://juliandyke.com/Diagnostics/Events/Events.html

[10] Oracle Internals – Trace levels, Julian Dyke
http://juliandyke.com/Diagnostics/Trace/TraceLevels.html

[11] Oracle Internals – Event reference, Julian Dyke
http://www.juliandyke.com/Diagnostics/Events/EventReference.html

[12] Oracle Tuning - The Definitive Reference, Rampant Techpress, Donald K. Burleson, Alexey B. Danchenkov

[13] Oracle is the #1 Relational Database
http://www.oracle.com/database/number-one-database.html

[14] Adatbázis üzemeltetés, Tóth Balázs
Oracle Junior képzési program, 2008. október 9.

Függelék

Példa nyomkövetési állomány

```
Dump file /u01/app/oracle/admin/nie11id3/udump/nie11id3_ora_6329.trc
Oracle9i Enterprise Edition Release 9.2.0.8.0 - 64bit Production
With the Partitioning option
JServer Release 9.2.0.8.0 - Production
ORACLE_HOME = /u01/app/oracle/product/9.2.0
System name:   SunOS
Node name:     migdev
Release:       5.10
Version:       Generic_138888-05
Machine:       sun4u
Instance name: nie11id3
Redo thread mounted by this instance: 1
Oracle process number: 207
Unix process pid: 6329, image: oracle@migdev (TNS V1-V3)

*** 2009-05-04 18:23:16.824
*** SESSION ID:(194.63245) 2009-05-04 18:23:16.735
APPNAME mod='SQL*Plus' mh=3669949024 act='' ah=4029777240
=====================
PARSING IN CURSOR #3 len=67 dep=0 uid=40 oct=42 lid=40
im=3513894248517 hv=986227385 ad='4729a218'
ALTER SESSION SET EVENTS'10046 trace name context forever, level 4'
END OF STMT
EXEC #3:c=0,e=128,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=3513894161700
=====================
PARSING IN CURSOR #3 len=107 dep=0 uid=40 oct=3 lid=40 tim=3513894458798 hv=3046699809
ad='e6f81818'
SELECT flow_status_code
     , COUNT(flow_status_code)
  FROM oe_order_lines_all
 GROUP BY flow_status_code
END OF STMT
PARSE #3:c=0,e=2744,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=3513894458793
BINDS #3:
EXEC #3:c=10000,e=98,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=3513894459669
*** 2009-05-04 18:26:15.602
FETCH #3:c=71310000,e=174385173,p=496132,cr=5478021,cu=0,mis=0,r=1,dep=0,og=4,tim=3514068844891
FETCH #3:c=0,e=48,p=0,cr=0,cu=0,mis=0,r=10,dep=0,og=4,tim=35140
69054808
```

```
STAT #3 id=1 cnt=11 pid=0 pos=1 obj=0 op='SORT GROUP BY 'STAT #3 id=2 cnt=4930386 pid=1 pos=1
obj=2097137 op='TABLE ACCESS FULL OE_ORDER_LINES_ALL '
=====================
PARSING IN CURSOR #3 len=85 dep=0 uid=40 oct=6 lid=40 tim=3514069302253 hv=183895537
ad='40d76fd0'
UPDATE aso_quote_headers_all
   SET attribute15 = 'Y'
 WHERE quote_header_id = 996324
END OF STMT
PARSE #3:c=0,e=5132,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=3514069302249
BINDS #3:
=====================
PARSING IN CURSOR #4 len=2241 dep=1 uid=0 oct=2 lid=0 tim=3514069366585 hv=3618587546
ad='ca7f82d8'
INSERT INTO "CDCMGR"."ASO_QUOTE_HEADERS_ALL_CT"
(operation$,cscn$,commit_timestamp$,source_colmap$,rsid$,timestamp$,"ATTRIBUTE2","RESOURCE_ID","C
ONTRACT_TEMPLATE_ID","CONTRACT_TEMPLATE_MAJOR_VER","CONTRACT_REQUESTER_ID","CONTRACT_APPROVAL_LEV
EL","PUBLISH_FLAG","RESOURCE_GRP_ID","SOLD_TO_PARTY_SITE_ID","DISPLAY_ARITHMETIC_OPERATOR","MAX_V
ERSION_FLAG","QUOTE_DESCRIPTION","MINISITE_ID","ATTRIBUTE_CATEGORY","QUOTE_TYPE","CUST_PARTY_ID",
"INVOICE_TO_CUST_PARTY_ID","PRICING_STATUS_INDICATOR","TAX_STATUS_INDICATOR","PRICE_UPDATED_DATE"
,"TAX_UPDATED_DATE","RECALCULATE_FLAG","PRICE_REQUEST_ID","ATTRIBUTE1","ATTRIBUTE3","ATTRIBUTE4",
"ATTRIBUTE5","ATTRIBUTE6","ATTRIBUTE7","ATTRIBUTE8","ATTRIBUTE9","ATTRIBUTE10","ATTRIBUTE11","ATT
RIBUTE12","ATTRIBUTE13","ATTRIBUTE14","ATTRIBUTE15","SECURITY_GROUP_ID","OBJECT_VERSION_NUMBER","
QUOTE_HEADER_ID","CREATION_DATE","CREATED_BY","LAST_UPDATE_DATE","LAST_UPDATED_BY","LAST_UPDATE_L
OGIN","REQUEST_ID","PROGRAM_APPLICATION_ID","PROGRAM_ID","PROGRAM_UPDATE_DATE","ORG_ID","QUOTE_NA
ME","QUOTE_NUMBER","QUOTE_VERSION","QUOTE_STATUS_ID","QUOTE_SOURCE_CODE","QUOTE_EXPIRATION_DATE",
"PRICE_FROZEN_DATE","QUOTE_PASSWORD","ORIGINAL_SYSTEM_REFERENCE","PARTY_ID","CUST_ACCOUNT_ID","IN
VOICE_TO_CUST_ACCOUNT_ID","ORG_CONTACT_ID","PHONE_ID","INVOICE_TO_PARTY_SITE_ID","INVOICE_TO_PART
Y_ID","ORIG_MKTG_SOURCE_CODE_ID","MARKETING_SOURCE_CODE_ID","ORDER_TYPE_ID","QUOTE_CATEGORY_CODE"
,"ORDERED_DATE","ACCOUNTING_RULE_ID","INVOICING_RULE_ID","EMPLOYEE_PERSON_ID","PRICE_LIST_ID","CU
RRENCY_CODE","TOTAL_LIST_PRICE","TOTAL_ADJUSTED_AMOUNT","TOTAL_ADJUSTED_PERCENT","TOTAL_TAX","TOT
AL_SHIPPING_CHARGE","SURCHARGE","TOTAL_QUOTE_PRICE","PAYMENT_AMOUNT","EXCHANGE_RATE","EXCHANGE_TY
PE_CODE","EXCHANGE_RATE_DATE","CONTRACT_ID","SALES_CHANNEL_CODE","ORDER_ID") VALUES  (:op,:cscn,
to_date('4000-01-01:00:00:00','YYYY-MM-
DD:HH24:MI:SS'),:scm,sys.cdc_rsid_seq$.nextval,SYSDATE,:1,:2,:3,:4,:5,:6,:7,:8,:9,:10,:11,:12,:13
,:14,:15,:16,:17,:18,:19,:20,:21,:22,:23,:24,:25,:26,:27,:28,:29,:30,:31,:32,:33,:34,:35,:36,:37,
:38,:39,:40,:41,:42,:43,:44,:45,:46,:47,:48,:49,:50,:51,:52,:53,:54,:55,:56,:57,:58,:59,:60,:61,:
62,:63,:64,:65,:66,:67,:68,:69,:70,:71,:72,:73,:74,:75,:76,:77,:78,:79,:80,:81,:82,:83,:84,:85,:8
6,:87,:88,:89,:90)
END OF STMT
PARSE #4:c=10000,e=2283,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=0,tim=3514069366581
BINDS #4:
 bind 0: dty=1 mxl=32(02) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
   bfp=ffffffff7fff952b bln=32 avl=02 flg=09
   value="UN"
 bind 1: dty=2 mxl=22(09) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
```

```
   bfp=ffffffff7fff93f8 bln=22 avl=09 flg=09
   value=281474976710655
 bind 2: dty=23 mxl=32(12) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
   bfp=ffffffff7fff94ac bln=32 avl=12 flg=09
   value=
Dump of memory from 0xFFFFFFFF7FFF94AC to 0xFFFFFFFF7FFF94B8
FFFFFFFF7FFF94A0                            00000000              [....]
FFFFFFFF7FFF94B0 00000000 00000001                    [........]
 bind 3: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
   bfp=00000000 bln=32 avl=00 flg=09
 bind 4: dty=2 mxl=22(06) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
   bfp=ffffffff7ce5c398 bln=22 avl=06 flg=09
   value=100011981
 bind 5: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
   bfp=00000000 bln=22 avl=00 flg=09
 bind 6: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
   bfp=00000000 bln=22 avl=00 flg=09
 bind 7: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
   bfp=00000000 bln=22 avl=00 flg=09
 bind 8: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
   bfp=00000000 bln=32 avl=00 flg=09
 bind 9: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
   bfp=00000000 bln=32 avl=00 flg=09
 bind 10: dty=2 mxl=22(06) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
   bfp=ffffffff7ce5d528 bln=22 avl=06 flg=09
   value=100000037
 bind 11: dty=2 mxl=22(05) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
   bfp=ffffffff7ce5d510 bln=22 avl=05 flg=09
   value=3012764
 bind 12: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
   bfp=00000000 bln=32 avl=00 flg=09
 bind 13: dty=1 mxl=32(01) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
   bfp=ffffffff7ce5d4e8 bln=32 avl=01 flg=09
   value="Y"
 bind 14: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
   bfp=00000000 bln=32 avl=00 flg=09
 bind 15: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
   bfp=00000000 bln=22 avl=00 flg=09
 bind 16: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
   bfp=00000000 bln=32 avl=00 flg=09
 bind 17: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
   bfp=00000000 bln=32 avl=00 flg=09
 bind 18: dty=2 mxl=22(04) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
   bfp=ffffffff7ce5d3a0 bln=22 avl=04 flg=09
   value=341075
 bind 19: dty=2 mxl=22(04) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
```

```
  bfp=ffffffff7ce5d388 bln=22 avl=04 flg=09
  value=341075
bind 20: dty=1 mxl=32(01) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=ffffffff7ce5d380 bln=32 avl=01 flg=09
  value="C"
bind 21: dty=1 mxl=32(01) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=ffffffff7ce5d378 bln=32 avl=01 flg=09
  value="C"
bind 22: dty=12 mxl=07(07) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=8 offset=0
  bfp=ffffffff7ce5d370 bln=07 avl=07 flg=09
  value="10/10/2008 9:8:17"
bind 23: dty=12 mxl=07(07) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=8 offset=0
  bfp=ffffffff7ce5d368 bln=07 avl=07 flg=09
  value="10/10/2008 9:8:17"
bind 24: dty=1 mxl=32(01) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=ffffffff7ce5d360 bln=32 avl=01 flg=09
  value="N"
bind 25: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 26: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 27: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 28: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 29: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 30: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 31: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 32: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 33: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 34: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 35: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 36: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 37: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 38: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 39: dty=1 mxl=32(01) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
```

```
  bfp=ffffffff7f555848 bln=32 avl=01 flg=09
  value="Y"
bind 40: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 41: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 42: dty=2 mxl=22(04) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7f555420 bln=22 avl=04 flg=09
  value=996324
bind 43: dty=12 mxl=07(07) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=8 offset=0
  bfp=ffffffff7f555328 bln=07 avl=07 flg=09
  value="10/10/2008 9:7:15"
bind 44: dty=2 mxl=22(04) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7f555310 bln=22 avl=04 flg=09
  value=33317
bind 45: dty=12 mxl=07(07) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=8 offset=0
  bfp=ffffffff7f555308 bln=07 avl=07 flg=09
  value="10/10/2008 9:8:29"
bind 46: dty=2 mxl=22(04) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7f5552f0 bln=22 avl=04 flg=09
  value=33317
bind 47: dty=2 mxl=22(03) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7f5552d8 bln=22 avl=03 flg=09
  value=-1
bind 48: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 49: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 50: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 51: dty=12 mxl=07(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=8 offset=0
  bfp=00000000 bln=07 avl=00 flg=09
bind 52: dty=2 mxl=22(03) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5cac0 bln=22 avl=03 flg=09
  value=183
bind 53: dty=1 mxl=32(32) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=ffffffff7ce5ca70 bln=32 avl=32 flg=09
  value="TP RT License - Dieter Scheufele"
bind 54: dty=2 mxl=22(04) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5ca58 bln=22 avl=04 flg=09
  value=949739
bind 55: dty=2 mxl=22(02) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5ca40 bln=22 avl=02 flg=09
  value=1
bind 56: dty=2 mxl=22(04) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5ca28 bln=22 avl=04 flg=09
```

```
  value=10063
bind 57: dty=1 mxl=32(20) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=ffffffff7ce5c938 bln=32 avl=20 flg=09
  value="Order Capture Quotes"
bind 58: dty=12 mxl=07(07) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=8 offset=0
  bfp=ffffffff7ce5c930 bln=07 avl=07 flg=09
  value="11/9/2008 0:0:0"
bind 59: dty=12 mxl=07(07) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=8 offset=0
  bfp=ffffffff7ce5c928 bln=07 avl=07 flg=09
  value="10/10/2008 0:0:0"
bind 60: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 61: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 62: dty=2 mxl=22(05) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c730 bln=22 avl=05 flg=09
  value=4707921
bind 63: dty=2 mxl=22(04) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c718 bln=22 avl=04 flg=09
  value=341075
bind 64: dty=2 mxl=22(04) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c700 bln=22 avl=04 flg=09
  value=341075
bind 65: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 66: dty=2 mxl=22(05) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c6d0 bln=22 avl=05 flg=09
  value=6718985
bind 67: dty=2 mxl=22(04) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c6b8 bln=22 avl=04 flg=09
  value=497640
bind 68: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 69: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 70: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 71: dty=2 mxl=22(03) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c658 bln=22 avl=03 flg=09
  value=1367
bind 72: dty=1 mxl=32(05) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=ffffffff7ce5c568 bln=32 avl=05 flg=09
  value="ORDER"
bind 73: dty=12 mxl=07(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=8 offset=0
  bfp=00000000 bln=07 avl=00 flg=09
bind 74: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
```

```
  bfp=00000000 bln=22 avl=00 flg=09
bind 75: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 76: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 77: dty=2 mxl=22(03) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c500 bln=22 avl=03 flg=09
  value=1465
bind 78: dty=1 mxl=32(03) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=ffffffff7ce5c4f0 bln=32 avl=03 flg=09
  value="EUR"
bind 79: dty=2 mxl=22(03) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c4d8 bln=22 avl=03 flg=09
  value=9725
bind 80: dty=2 mxl=22(05) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c4c0 bln=22 avl=05 flg=09
  value=-680.75
bind 81: dty=2 mxl=22(03) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c4a8 bln=22 avl=03 flg=09
  value=-7
bind 82: dty=2 mxl=22(04) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c490 bln=22 avl=04 flg=09
  value=1718.41
bind 83: dty=2 mxl=22(01) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c478 bln=22 avl=01 flg=09
  value=0
bind 84: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 85: dty=2 mxl=22(05) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=ffffffff7ce5c448 bln=22 avl=05 flg=09
  value=10762.66
bind 86: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 87: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 88: dty=1 mxl=32(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=00000000 bln=32 avl=00 flg=09
bind 89: dty=12 mxl=07(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=8 offset=0
  bfp=00000000 bln=07 avl=00 flg=09
bind 90: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
bind 91: dty=1 mxl=32(06) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=32 offset=0
  bfp=ffffffff7ce5c3c8 bln=32 avl=06 flg=09
  value="DIRECT"
bind 92: dty=2 mxl=22(00) mal=00 scl=00 pre=00 oacflg=10 oacfl2=1 size=24 offset=0
  bfp=00000000 bln=22 avl=00 flg=09
```

```
EXEC #4:c=0,e=26859,p=2,cr=2,cu=1,mis=0,r=1,dep=1,og=4,tim=3514069393707
EXEC #3:c=10000,e=91411,p=6,cr=6,cu=3,mis=0,r=1,dep=0,og=4,tim=3514069393756
*** 2009-05-04 18:28:31.872
XCTEND rlbk=0, rd_only=0
STAT #3 id=1 cnt=1 pid=0 pos=1 obj=0 op='UPDATE  '
STAT #3 id=2 cnt=1 pid=1 pos=1 obj=2127445 op='TABLE ACCESS BY INDEX ROWID OBJ#(2127445) '
STAT #3 id=3 cnt=1 pid=2 pos=1 obj=2127669 op='INDEX UNIQUE SCAN OBJ#(2127669) '
STAT #4 id=1 cnt=1 pid=0 pos=1 obj=0 op='SEQUENCE  '
```

Példa TKPROF kimenet

```
TKPROF: Release 9.2.0.1.0 - Production on Mon May 4 23:42:31 2009

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

Trace file: pelda_nie11id3_ora_6329.trc
Sort options: exeela  fchela
********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
********************************************************************************

SELECT flow_status_code
     , COUNT(flow_status_code)
  FROM oe_order_lines_all
 GROUP BY flow_status_code

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.01       0.00          0          0          0           0
Fetch        2     71.31     174.38     496132    5478021          0          11
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4     71.32     174.38     496132    5478021          0          11

Misses in library cache during parse: 1
```

```
Optimizer goal: CHOOSE
Parsing user id: 40

Rows     Row Source Operation
-------  ---------------------------------------------------
     11  SORT GROUP BY
4930386   TABLE ACCESS FULL OE_ORDER_LINES_ALL

********************************************************************************

UPDATE aso_quote_headers_all
   SET attribute15 = 'Y'
 WHERE quote_header_id = 996324

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.01       0.06          4          4          2           1
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2      0.01       0.06          4          4          2           1

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 40

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  UPDATE
      1   TABLE ACCESS BY INDEX ROWID OBJ#(2127445)
      1    INDEX UNIQUE SCAN OBJ#(2127669) (object id 2127669)

********************************************************************************

INSERT INTO "CDCMGR"."ASO_QUOTE_HEADERS_ALL_CT" (operation$,cscn$,
  commit_timestamp$,source_colmap$,rsid$,timestamp$,"ATTRIBUTE2",
  "RESOURCE_ID","CONTRACT_TEMPLATE_ID","CONTRACT_TEMPLATE_MAJOR_VER",
  "CONTRACT_REQUESTER_ID","CONTRACT_APPROVAL_LEVEL","PUBLISH_FLAG",
  "RESOURCE_GRP_ID","SOLD_TO_PARTY_SITE_ID","DISPLAY_ARITHMETIC_OPERATOR",
  "MAX_VERSION_FLAG","QUOTE_DESCRIPTION","MINISITE_ID","ATTRIBUTE_CATEGORY",
```

```
  "QUOTE_TYPE","CUST_PARTY_ID","INVOICE_TO_CUST_PARTY_ID",
  "PRICING_STATUS_INDICATOR","TAX_STATUS_INDICATOR","PRICE_UPDATED_DATE",
  "TAX_UPDATED_DATE","RECALCULATE_FLAG","PRICE_REQUEST_ID","ATTRIBUTE1",
  "ATTRIBUTE3","ATTRIBUTE4","ATTRIBUTE5","ATTRIBUTE6","ATTRIBUTE7",
  "ATTRIBUTE8","ATTRIBUTE9","ATTRIBUTE10","ATTRIBUTE11","ATTRIBUTE12",
  "ATTRIBUTE13","ATTRIBUTE14","ATTRIBUTE15","SECURITY_GROUP_ID",
  "OBJECT_VERSION_NUMBER","QUOTE_HEADER_ID","CREATION_DATE","CREATED_BY",
  "LAST_UPDATE_DATE","LAST_UPDATED_BY","LAST_UPDATE_LOGIN","REQUEST_ID",
  "PROGRAM_APPLICATION_ID","PROGRAM_ID","PROGRAM_UPDATE_DATE","ORG_ID",
  "QUOTE_NAME","QUOTE_NUMBER","QUOTE_VERSION","QUOTE_STATUS_ID",
  "QUOTE_SOURCE_CODE","QUOTE_EXPIRATION_DATE","PRICE_FROZEN_DATE",
  "QUOTE_PASSWORD","ORIGINAL_SYSTEM_REFERENCE","PARTY_ID","CUST_ACCOUNT_ID",
  "INVOICE_TO_CUST_ACCOUNT_ID","ORG_CONTACT_ID","PHONE_ID",
  "INVOICE_TO_PARTY_SITE_ID","INVOICE_TO_PARTY_ID","ORIG_MKTG_SOURCE_CODE_ID",
  "MARKETING_SOURCE_CODE_ID","ORDER_TYPE_ID","QUOTE_CATEGORY_CODE",
  "ORDERED_DATE","ACCOUNTING_RULE_ID","INVOICING_RULE_ID",
  "EMPLOYEE_PERSON_ID","PRICE_LIST_ID","CURRENCY_CODE","TOTAL_LIST_PRICE",
  "TOTAL_ADJUSTED_AMOUNT","TOTAL_ADJUSTED_PERCENT","TOTAL_TAX",
  "TOTAL_SHIPPING_CHARGE","SURCHARGE","TOTAL_QUOTE_PRICE","PAYMENT_AMOUNT",
  "EXCHANGE_RATE","EXCHANGE_TYPE_CODE","EXCHANGE_RATE_DATE","CONTRACT_ID",
  "SALES_CHANNEL_CODE","ORDER_ID")
VALUES
  (:op,:cscn,   to_date('4000-01-01:00:00:00','YYYY-MM-DD:HH24:MI:SS'),:scm,
  sys.cdc_rsid_seq$.nextval,SYSDATE,:1,:2,:3,:4,:5,:6,:7,:8,:9,:10,:11,:12,
  :13,:14,:15,:16,:17,:18,:19,:20,:21,:22,:23,:24,:25,:26,:27,:28,:29,:30,:31,
  :32,:33,:34,:35,:36,:37,:38,:39,:40,:41,:42,:43,:44,:45,:46,:47,:48,:49,:50,
  :51,:52,:53,:54,:55,:56,:57,:58,:59,:60,:61,:62,:63,:64,:65,:66,:67,:68,:69,
  :70,:71,:72,:73,:74,:75,:76,:77,:78,:79,:80,:81,:82,:83,:84,:85,:86,:87,:88,
  :89,:90)


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.00          0          0          0           0
Execute      1      0.00       0.02          2          2          1           1
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2      0.01       0.02          2          2          1           1

Misses in library cache during parse: 1
```

```
Optimizer goal: CHOOSE
Parsing user id: SYS   (recursive depth: 1)

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SEQUENCE

********************************************************************************
ALTER SESSION SET EVENTS'10046 trace name context forever, level 4'


all     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        0      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        1      0.00       0.00          0          0          0           0

Misses in library cache during parse: 0
Misses in library cache during execute: 1
Optimizer goal: CHOOSE
Parsing user id: 40


********************************************************************************

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.00       0.00          0          0          0           0
Execute      3      0.02       0.06          4          4          2           1
Fetch        2     71.31     174.38     496132    5478021          0          11
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        7     71.33     174.45     496136    5478025          2          12

Misses in library cache during parse: 2
Misses in library cache during execute: 1
```

```
OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.00          0          0          0           0
Execute      1      0.00       0.02          2          2          1           1
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2      0.01       0.02          2          2          1           1
Misses in library cache during parse: 1


   3  user  SQL statements in session.
    1  internal SQL statements in session.
    4  SQL statements in session.
********************************************************************************
Trace file: pelda_nie11id3_ora_6329.trc
Trace file compatibility: 9.00.01
Sort options: exeela  fchela
       1  session in tracefile.
       3  user  SQL statements in trace file.
       1  internal SQL statements in trace file.
       4  SQL statements in trace file.
       4  unique SQL statements in trace file.
     294  lines in trace file.
```