COST BASED QUERY TRANSFORMATIONS
CONCEPT AND ANALYSIS USING 10053 TRACE

Introduction
This paper is to explore cost based query transformation introduced in 10g and enhanced in 11g. Trace files generated from 10053 events are analyzed to further explore and analyze these transformations.

This paper is designed to provide an outline of features. Not an exhaustive educational material. Scripts are provided wherever possible to help improve the understanding of these features. Presentation must be used in conjunction with this paper to further understanding of this new feature.

What is Query transformation
Cost based optimizer rewrites SQL using various techniques discussed in the paper. These are known as transformations and many such transformations are possible for a given query. Some transformations are cheaper than others and Optimizer further evaluates cost for these transformations to find cheapest transformation.

Introduction to logical optimizer
Until 10g, there is one optimizer component that evaluates cost for various join permutations, join cost etc. This is considered as physical optimization as mostly physical aspects of the cost is considered. Oracle version 10g introduces transformations which are logical rewrites of original SQL. A distinction must be made between two layers of optimizer, one logical and another physical optimizer.

Further logical optimizer calls physical optimizer modules for each transformation to evaluate cost for that transformation. Module kkoqbc is the physical optimizer module and called by logical optimizer for each transformation.

Example query
Following query is a correlated subquery and will be used in initial part of this paper. This query finds all employees whose salary is greater than average salary in their department. Further predicates are added to check that location exists in locations table and the employee employed at least for the past 3650 days.

```sql
SELECT /*+ qb_name (e1_outer) */ * FROM emp e1 WHERE salary >
    (SELECT /*+ qb_name (e2_inner) */ AVG(salary) FROM emp e2, dept d1
    WHERE e1.dept_id = e2.dept_id AND e2.dept_id = d1.dept_id AND
    EXISTS (SELECT /*+ qb_name (l1_inner) */ 1 FROM locations l1
    WHERE l1.location_id=d1.location_id )
) AND e1.hire_date > SYSDATE - (10*365)
```

Query graph for the above query is represented below. For rows from emp table, inner subquery is executed to find average salary for that dept_id. This is a multilevel subquery.
Transformed query
As an example, above original query was transformed to the following query by cost based logical optimizer. This transformation is known as group by placement. Original query is a correlated subquery: For each row from outer row source (emp) inner query is executed. Transformed query is a non-correlated subquery. A new variation of subquery has been created and aggregation step moved before joining to emp table.

```
Select /*+ qb_name (e1_outer) */ * from 
emp e1,
(select /*+ qb_name (e2_inner) */
d1.dept_id, avg(salary) avg_salary
from emp e2, dept d1
where e2.dept_id = d1.dept_id and
exists
(select /*+ qb_name (l1_inner) */ 1 from locations l1
where l1.location_id=d1.location_id )
group by dept_id ) gbp1
where
 e1.hire_date > sysdate - (10*365)  and
 e1. salary > gbp1.avg_salary and
 e1.dept_id = gbp1.dept_id
```

This is just one of the transformation and many such transformations possible. These transformations can not be achieved in a single step. Optimizer steps through many intermediate query states before completing the transformation.

**Why do transformations must be costed?**
There are many such transformations possible. Optimal transformation depends upon data distribution, properties of data and SQL.

Consider original SQL using correlated subquery: For each row from outer emp table, correlated sub-query executed once. Let us assume that cost for each execution of subquery is 100.

So approximate cost for original subquery is: Cost for outer emp table rows + # of rows in outer row source (N) X Cost for executing inner subquery

Consider transformed query: subquery with alias gbp1 executed only once and let us also assume that cost for that gbp1 subquery is 10000.

Approximate cost for transformed subquery is:
Cost for outer emp table rows + Cost for executing inner subquery (gbp1) + # of rows in outer row source (N) X join cost
For comparison purposes, first common term can be removed and total cost for each step compared in the table below. It is visible that optimal transformation depends upon # of rows in the table.

<table>
<thead>
<tr>
<th># of rows</th>
<th>Original query</th>
<th>Transformed query (join cost of 0.1 per row)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 X 100 = 100</td>
<td>1X10000+0.1 = 10000.1</td>
</tr>
<tr>
<td>100</td>
<td>100 X 100 = 10,000</td>
<td>1X10000+100 X 0.1 = 10,010</td>
</tr>
<tr>
<td>10000</td>
<td>10000 X 100 = 1,000,000</td>
<td>1X10000+10,000 X 0.1 = 11,000</td>
</tr>
</tbody>
</table>

**Phases**

Optimizer transforms the query using various techniques and following diagram is an illustration of how these techniques applied in phases. After each transformation, physical optimizer kkoqbc is called to calculate optimal execution plan for the transformed query and cost saved. Cost for these transformations tracked as cost annotations and transformation with lowest cost is chosen as the final execution plan.

There are certain heuristic transformation such as Common Subexpression Elimination (CSE), Order BY Elimination (OBYE), Join Elimination (JE) etc and need not be costed.

Following sections of the paper walks through 10053 trace file for this example SQL and shows how these transformations are applied to this SQL, in a complex, atomic steps.

---

1 Not all transformations must be costed. Some transformation need not be costed as results of those transformations will improve performance.
**Subquery unnest**

Subquery unnest is a transforms subquery operations in to a join operation. First phase for this SQL is transforming exists operator in to a semi join. Query graph below illustrates this step. Semi join is an execution step and different from regular join operator.

**Trace lines for Subquery unnest**

Following trace lines from 10053 trace shows above transformation. Query block L1_INNER is transformed in to a semi join in this step.

Registered qb: SEL$D72FB22B 0x21ee71cc (SUBQUERY UNNEST E2_INNER; L1_INNER)

signature (): qb_name=SEL$D72FB22B nbfros=3 flg=0
fro(0): flg=0 objn=71162 hint_alias="D1"@"E2_INNER"
fro(1): flg=0 objn=71164 hint_alias="E2"@"E2_INNER"
fro(2): flg=0 objn=71160 hint_alias="L1"@"L1_INNER"

Transformed query shown below using this step. Semi join is shown as S= step below.

```
select /*+ qb_name (e1_outer) */ * from
  emp e1 where
  salary >
  (select /*+ qb_name (e2_inner) */
   avg(salary) from
    emp e2, dept d1, locations l1
   where e1.dept_id = e2.dept_id and
     e2.dept_id = d1.dept_id and
     l1.location_id S= d1.location_id )
and e1.hire_date > sysdate - (10*365)
```

**Subquery unnest #2**

Next step is unnesting inner subquery with ‘salary>’ operator in to a join operator. This is achieved in multiple steps.

---

Riyaj Shamsudeen
HOTSOS’ 2008
But, first inner subquery must be moved in to a view for subsequent transformation.

Registered qb: SEL$58CDACD2 0x21ee4a5c (SUBQ INTO VIEW FOR COMPLEX UNNEST SEL$D72FB22B)

Then e1_outer is added in to that view and a new query block is created.

Registered qb: SEL$4ADFCC1B 0x21ee5968 (VIEW ADDED E1_OUTER)

Following step converts this query block by unnesting vw_sq_1 view.

Registered qb: SEL$825981F0 0x21ee5968 (SUBQUERY UNNEST SEL$4ADFCC1B; SEL$D72FB22B)

Transformed query shown below after above subquery unnest step.

```sql
select /*+ qb_name (e1_outer) */ e1.* from
emp e1,
(select/*+ qb_name (e2_inner) */
avg(salary) avg1,
e2.dept_id item_0
from emp e2, dept d1, locations l1
where e2.dept_id = d1.dept_id and
l1.location_id = d1.location_id
group by e2.dept_id ) vw_sq_1
where e1.salary > vw_sq_1.avg1
and e1.hire_date > sysdate - (10*365)
and e1.dept_id = vw_sq_1.item_0
```

Riyaj Shamsudeen
HOTSOS’ 2008
FPD – Filter Push Down

Filter predicates are pushed down to query block and applied at their query blocks. With unnesting step above filter predicates must be pushed down so that it can be applied at more optimum level. Query graph printed below to improve understanding of trace lines.

![Query Graph](image)

Trace lines for FPD in Query block SEL$825981F0

Following trace lines shows that FPD step is applied at query block SEL$825981F0.

```sql
FPD: Considering simple filter push in query block SEL$825981F0 (#1)
"E1"."SALARY">"VW_SQ_1"."AVG(SALARY)" AND "E1"."DEPT_ID"="VW_SQ_1"."ITEM_0" AND
"E1"."HIRE_DATE">SYSDATE@!-3650
```

Optimizer also tries to generate transitive predicates from check constraints.

```sql
try to generate transitive predicate from check constraints for query block SEL$825981F0
(#1)
finally: "E1"."SALARY">"VW_SQ_1"."AVG(SALARY)" AND "E1"."DEPT_ID"="VW_SQ_1"."ITEM_0" AND
"E1"."HIRE_DATE">SYSDATE@!-3650
```

Trace lines for FPD in Query block SEL$58CDACD2

Following trace lines shows that FPD step is applied at query block SEL$825981F0.

```sql
FPD: Considering simple filter push in query block SEL$58CDACD2 (#2)
"E2"."DEPT_ID"="D1"."DEPT_ID" AND "L1"."LOCATION_ID"="D1"."LOCATION_ID"
```

```sql
try to generate transitive predicate from check constraints for query block SEL$58CDACD2
(#2)
finally: "E2"."DEPT_ID"="D1"."DEPT_ID" AND "L1"."LOCATION_ID"="D1"."LOCATION_ID"
```

Costing query block – calling physical optimizer

Next step of the transformation calls kkoqbc physical optimizer module to evaluate cost for this transformed SQL. Calls to physical modules are done at a query block granularity. As shown in trace file below, inner view with alias vw_sq_1 is costed. Typical lines from 10053 trace file showing table columns, indices and cost to access rows etc printed.

```sql
SU: Costing transformed query.
CBQT: Looking for cost annotations for query block SEL$58CDACD2, key=SEL$58CDACD2_00000202_2
CBQT: Could not find stored cost annotations. ...(1)
```

```sql
kkoqbc: optimizing query block SEL$58CDACD2 (#2)
```

Riyaj Shamsudeen
HOTSOS’ 2008
Cost annotations – Reducing calls to physical optimizer

It is possible to have explosion of transformations as number of tables in the subquery section increases. This, in turn, can lead to very high number of calls to physical optimizer increasing parse time and CPU usage spent for parsing. It is also quite obvious that same query block can be parsed repeatedly.

Impact of this is reduced by keeping track of costed query block. Similar query block can have minor, but critical changes and may be costed again. Each query block variation is attached with a unique signature. Logical optimizer reuses already optimized query blocks if signature matches.

From the trace lines printed above, at line (1) before calling physical optimizer, cost annotations with a signature of SEL$58CDACD2_00000202_2 is checked to see if that query block with that signature is already costed 3.

```
Trying or-Expansion on query block SEL$58CDACD2 (#2)
Transfer Optimizer annotations for query block SEL$58CDACD2 (#2)
Final cost for query block SEL$58CDACD2 (#2) - All Rows Plan:
  Best join order: 2
  Cost: 489.7812  Degree: 1  Card: 99990.0000  Bytes: 2547500
  Resc: 489.7812  Resp: 489.7812
```

Above trace lines shows that cost annotations are stored and final cost for that annotation is saved. This is only within the parsing session, not permanent. Following table illustrates how these cost annotations are stored, as listed in [2]. Undocumented (and so unsupported) parameter “_optimizer_reuse_cost_annotations” controls this behavior. By default, it is set to true 4.

<table>
<thead>
<tr>
<th>QB identifier</th>
<th>State</th>
<th>QB type</th>
<th>Cost</th>
<th>Cardinality</th>
<th>selectivity</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. SEL$58CDACD2_00000202_2</td>
<td></td>
<td></td>
<td>489</td>
<td>663</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. SEL$825981F0_00000000_0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Costing next query block SEL$825981F0

Next outer query block is costed below.

```
kkoqbc: optimizing query block SEL$825981F0 (#1)
```

```
QUERY BLOCK SIGNATURE

signature (optimizer): qb_name=SEL$825981F0 nbfros=2 flg=0
  fro(0): flg=0 objn=71164 hint_alias="E1"@"E1 OUTER"
  fro(1): flg=1 objn=0 hint_alias="VW_SQ_1"@"SEL$4ADFC1B"
```

---

3 Signature seems to have three parts to it. Various tables and their states are represented in this signature. Minor changes such reordering of tables, level of group by etc creates new and unique signatures.

4 Underscore parameters are not supported by Oracle Corporation unless specifically evaluated. Oracle support must be contacted before using any underscore parameters, in a production environment. There parameters are listed here to improve understanding, not as a guidance to use them.
Interleaved CVM

Next phase is merging complex views to further transform this SQL. Following picture shows how output of prior transformation is transformed into simple join, by Complex View Merging (CVM) step.

```
select /*+ qb_name (e1_outer) */ * from
    emp e1,
    (select /*+ qb_name (e2_inner) */
        avg(salary) ,
        dept_id item_0
    from
        emp e2, dept d1, locations l1
    where  e2.dept_id = d1.dept_id
    and
    l1.location_id = d1.location_id
    group by dept_id)   "vw_sq_1"
where
    e1.salary >
    vw_sq_1."avg(salary)"
    and e1.hire_date > sysdate -
    (10*365)
    and e1.dept_id =
    vw_sq_1.item_0

select /*+ qb_name (e1_outer) */ e1.* from
    emp e1, emp e2, dept d1, locations l1
where
    e1.dept_id = e2.dept_id and
    e1.hire_date > sysdate -
    (10*365)
    and e2.dept_id = d1.dept_id and
    l1.location_id = d1.location_id
Group by
    e2.Dept_id, e1.hire_date, e1.salary ,
    e1.dept_id, e1.emp_name, e1.emp_id
Having e1.salary > avg(e2.salary)
```

Following are the 10053 trace lines for the above step.

- CVM: CBQT Marking query block SEL$58CDACD2 (#2) as valid for CVM.
- CVM: Merging complex view SEL$58CDACD2 (#2) into SEL$825981F0 (#1).
- CVM: result SEL$825981F0 (#1)
- Registered qb: SEL$8370D25A 0x21ee0968 (VIEW MERGE SEL$825981F0; SEL$58CDACD2)

Following query graph transformation might better illustrate above Complex view merging phase.
Costing transformation

Above transformation is costed calling physical optimizer kkoqbe module. Note below that all four row sources are printed in single line.

Registered qb: SEL$8370D25A 0x21ee0968 (VIEW MERGE SEL$825981F0; SEL$58CDACD2)

<table>
<thead>
<tr>
<th>QUERY BLOCK SIGNATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>signature (): qb_name=SEL$8370D25A nbfros=4 flg=0</td>
</tr>
<tr>
<td>fro(0): flg=0 objn=71164 hint_alias=&quot;E1&quot;@&quot;E1_OUTER&quot;</td>
</tr>
<tr>
<td>fro(1): flg=0 objn=71162 hint_alias=&quot;D1&quot;@&quot;E2_INNER&quot;</td>
</tr>
<tr>
<td>fro(2): flg=0 objn=71164 hint_alias=&quot;E2&quot;@&quot;E2_INNER&quot;</td>
</tr>
<tr>
<td>fro(3): flg=0 objn=71160 hint_alias=&quot;L1&quot;@&quot;L1_INNER&quot;</td>
</tr>
</tbody>
</table>

Join Predicate Push Down (JPPD)

Join predicates can be pushed down to inner query block further filtering rows, reducing cost. As shown below, JPPD is considered for this SQL, but not applied since no valid join conditions were found to push down. This feature will be explained using a different query structure later.

JPPD: Checking validity of push-down from query block SEL$825981F0 (#1) to query block SEL$58CDACD2 (#2)

Check Basic Validity for Non-Union View for query block SEL$58CDACD2 (#2)

JPPD: Bypassed: No valid join condition found.

Transformation: Moving subquery as a column level subquery

Next transformation is converting the inline view to a column level subquery and costing this transformation.

This transformation visible in trace lines as bind variables are used in the inner subquery block.
SU: Starting iteration 2, state space = (2) : (0)

FPD: Considering simple filter push in query block SEL$D72FB22B (#2)
"E2"."DEPT_ID"=:B1 AND "E2"."DEPT_ID"="D1"."DEPT_ID" AND 
"L1"."LOCATION_ID"="D1"."LOCATION_ID"
try to generate transitive predicate from check constraints for query block 
SEL$D72FB22B (#2)
finally: "E2"."DEPT_ID"=:B1 AND "E2"."DEPT_ID"="D1"."DEPT_ID" AND 
"L1"."LOCATION_ID"="D1"."LOCATION_ID" AND "D1"."DEPT_ID"=:B2

This subquery is rearranged into a column level subquery.

select /*+ qb_name (e1_outer) *//* from emp e1,
  (select avg(salary), dept_id item_0
    from emp e2, dept d1, locations l1
    where e1.dept_id = e2.dept_id
    and e2.dept_id = d1.dept_id
    and l1.location_id = d1.location_id
    group by dept_id ) "vw_sq_1"
where
  e1.salary >
  vw_sq_1."avg(salary)"
  and e1.hire_date > sysdate -
    (10*365)
  and e1.dept_id =

select emp_id, emp_name, dept_id, salary, hire_Date
from ( select e1.*,
  (select avg(salary) from
    emp e2, dept d1, locations l1
    where e1.dept_id = e2.dept_id
    and e2.dept_id = d1.dept_id
    and l1.location_id = d1.location_id
  ) avg_sal
  from emp e1 )
where salary > avg_sal and hire_date> sysdate-3650

Cost for this column level subquery calculated below as approximately 173.

Final cost for query block SEL$D72FB22B (#2) - All Rows Plan:
  Best join order: 1
  Cost: 173.2842  Degree: 1  Card: 1.00000  Bytes: 21
  Resc: 173.2842  Resc_io: 172.0000  Resc_cpu: 28447119

Costing outer query block
Outer query block is costed using the above column level subquery. Notice that cost is very high, since inner subquery is called for each row.

Final adjusted join cardinality: 1521, sq. fil. factor: 20.000000
Trying or-expansion on query block E1_OUTER (#1)
Transfer Optimizer annotations for query block E1_OUTER (#1)
Final cost for query block E1_OUTER (#1) - All Rows Plan:
  Best join order: 1
  Cost: 2694538.1787  Degree: 1  Card: 1521.0000  Bytes: 973216
  Resc: 2694538.1787  Resc_io: 2674566.1836  Resc_cpu: 442397934575
  Resp: 2694538.1787  Resp_io: 2674566.1836  Resc_cpu: 442397934575
**SJC – Set to Join Conversion, Predicate Movement, Join elimination**

These transformations will be explained with a different SQL later to simplify understanding.

**GBP – Group By Placement**

Group by operators if applied at an optimal step can reduce number of rows from a row source, reduce cost and can improve performance. This transformation considers applying group by operator at various levels.

Only inner subquery is considered in this transformation as group by operator exists in that query block. Notice that there are only only three tables in this subquery and GBP is using exhaustive search to probe all possible transformation. Following query graph is showing that group by operator applied at various levels, in a nutshell.

---

*Cost-Based Group By Placement*

GBP: Checking validity of GBP for query block SEL$58CDACD2 (#2)

GBP: Checking validity of group-by placement for query block SEL$58CDACD2 (#2)

GBP: Using search type: exhaustive

GBP: Considering group-by placement on query block SEL$58CDACD2 (#2)

GBP: Starting iteration 1, state space = (3,4,5) : (0,0,0)

GBP: Transformed query

FPD: Considering simple filter push in query block SEL$58CDACD2 (#2)

"E2"."DEPT_ID"="D1"."DEPT_ID" AND "L1"."LOCATION_ID"="D1"."LOCATION_ID"

try to generate transitive predicate from check constraints for query block SEL$58CDACD2 (#2)

finally: "E2"."DEPT_ID"="D1"."DEPT_ID" AND 
"L1"."LOCATION_ID"="D1"."LOCATION_ID"
From above 4 graphs, group by operator is applied at various levels. GBP#1 applies gb (group by) operator in all tables, GBP#2 applies gb operator at emp level, GBP #3 applies gb operator at dept, locations level, GBP#4 combines GBP#2 and GBP#3. These possible transformations are discussed below.

Due to minimal # of tables involved in this subquery, exhaustive search is used. It is possible to use different search techniques such as linear, two-pass etc and Optimizer automatically switches to a different search technique, if number of tables are higher.

GBP #1
In this transformation, group by operator is applied after joining all three tables, represented above as #1.

```
select avg(salary), dept_id from
  emp e2,
  dept d1,
  locations l1
where
  e2.dept_id = d1.dept_id and
  d1.location_id = l1.location_id
GROUP BY dept_id
```

Trace lines shows three row sources in a normal join condition.

GBP #2
In this transformation Group by operator applied at emp table level and then dept d1 and locations l1 joined.

Riyaj Shamsudeen
HOTSOS' 2008
This transformation happens in multiple steps.

GBP: Starting iteration 2, state space = (3,4,5) : (0,C,0)

Both row sources D1 & L1 moved in to a new query block.

Registered qb: SEL$24BEC10C 0x21ea6b00 (QUERY BLOCK TABLES CHANGED SEL$58CDACD2)
QUERY BLOCK SIGNATURE
signature (): qb_name=SEL$24BEC10C nbfros=2 flg=0
  fro(0): flg=0 objn=71162 hint_alias="D1"@"E2_INNER"
  fro(1): flg=0 objn=71160 hint_alias="L1"@"L1_INNER"

Row source E2 is split and moved in to a new query block and a new view created as VW_GBC_2.

Registered qb: SEL$6543C244 0x21ea0f64 (SPLIT/MERGE QUERY BLOCKS SEL$24BEC10C)
QUERY BLOCK SIGNATURE
signature (): qb_name=SEL$6543C244 nbfros=1 flg=0
  fro(0): flg=0 objn=71164 hint_alias="E2"@"E2_INNER"

Above two query blocks are joined to create new query block.

Registered qb: SEL$E003ED3F 0x21ea6b00 UNKNOWN QUERY BLOCK ORIGIN SEL$58CDACD2; SEL$58CDACD2; LIST 2)
QUERY BLOCK SIGNATURE
signature (): qb_name=SEL$E003ED3F nbfros=3 flg=0
  fro(0): flg=0 objn=71162 hint_alias="D1"@"E2_INNER"
  fro(1): flg=0 objn=71160 hint_alias="L1"@"L1_INNER"
  fro(2): flg=5 objn=0 hint_alias="VW_GBC_2"@"SEL$E003ED3F"

Calling physical optimizer to cost transformed query

First inner query block aggregating emp e2 alone costed calling physical optimizer. Note the row source has only e2.

GBP: Costing transformed query.
CBQT: Looking for cost annotations for query block SEL$6543C244, key = SEL$6543C244_00001200_3
CBQT: Could not find stored cost annotations.
kkqbc: optimizing query block SEL$6543C244 (#3)
QUERY BLOCK SIGNATURE
signature (optimizer): qb_name=SEL$6543C244 nbfros=1 flg=0

select vw_gbc_2.dept_id, avg_salary
from
  ( select avg(salary) avg_salary, dept_id item_1
      from emp e2
      group by dept_id ) vw_gbc_2,
  dept d1,
  locations l1
where
  vw_gbc_2.dept_id = d1.dept_id and
  and d1.location_id = l1.location_id

Riyaj Shamsudeen  
HOTSOS’ 2008
fro(0): flg=0 objn=71164 hint_alias="E2"@"E2_INNER"

Trying or-Expansion on query block SEL$6543C244 (#3)
Transfer Optimizer annotations for query block SEL$6543C244 (#3)
Final cost for query block SEL$6543C244 (#3) - All Rows Plan:
  Best join order: 1
  Cost: 514.8488  Degree: 1  Card: 100000.0000  Bytes: 300000
  Resc: 514.8488  Resc_io: 509.0000  Resc_cpu: 129556005
  Resp: 514.8488  Resp_io: 509.0000  Resp_cpu: 129556005
kkoqbc-subheap (delete addr=0x0C31E560, in-use=9860, alloc=12356)
kkoqbc-end:

Whole query is costed next by calling physical optimizer.

QUERY BLOCK SIGNATURE
---------------------
signature (optimizer): qb_name=SEL$E003ED3F nbfros=3 flg=0
  fro(0): flg=0 objn=71162 hint_alias="D1"@"E2_INNER"
  fro(1): flg=0 objn=71160 hint_alias="L1"@"L1_INNER"
  fro(2): flg=1 objn=0 hint_alias="VW_GBC_2"@"SEL$E003ED3F"

Trying or-Expansion on query block SEL$E003ED3F (#2)
Transfer Optimizer annotations for query block SEL$E003ED3F (#2)
Final cost for query block SEL$E003ED3F (#2) - All Rows Plan:
  Best join order: 3
  Cost: 672.7886  Degree: 1  Card: 9990.0000  Bytes: 589410
  Resc: 672.7886  Resc_io: 664.0000  Resc_cpu: 194675802
  Resp: 672.7886  Resp_io: 664.0000  Resp_cpu: 194675802
kkoqbc-subheap (delete addr=0x0C31F224, in-use=28556, alloc=31680)
kkoqbc-end:
  call(in-use=124192, alloc=180120), compile(in-use=404584, alloc=447744),
  execution(in-use=468220, alloc=470204)
  kkoqbc: finish optimizing query block SEL$E003ED3F (#2)
CBQT: Saved costed qb# 3 (SEL$6543C244), key = SEL$6543C244_00001200_3
CBQT: Saved costed qb# 2 (SEL$E003ED3F), key = SEL$E003ED3F_00000042_0
GBP: Updated best state, Cost = 672.79

Cost annotations are saved for these two query block for further reuse.

GBP #3
Next Group by placement technique applies group by operator at dept, locations and then emp e2
joined. An additional group by operator applied finally for data consistency. As evident, group by
operator is tried at various levels and transformed SQL costed using physical optimizer.

```sql
select
  sum(c2.salary*vw_gbf_3.item_2)/
  sum(decode(c2.salary, null, 0, vw_gbf_3.item_3))
  avg(salary),
  c2.dept_id item_1
from
  ( select d1.dept_id item_1, count(*) item_2,
      count(*) item_3
    from dept d1, locations l1
    where l1.location_id = d1.location_id
    group by d1.dept_id ) vw_gbf_3,
  emp e2
where c2.dept_id = vw_gbf_3.item_1
  group by d2.dept_id
```

Riyaj Shamsudeen
HOTSOS’ 2008
Steps for this transformation:
This transformation happens in multiple steps.

GBP: Starting iteration 3, state space = (3,4,5) : (F,0,F)

Step 1: A new query block created with just E2

Registered qb: SEL$50B3ADB4 0x21e9dd44
(Query BLOCK TABLES CHANGED SEL$58CDACD2)
QUERY BLOCK SIGNATURE
signature (): qb_name=SEL$50B3ADB4 nbfros=1 flg=0
    fro(0): flg=0 objn=71164 hint_alias="E2"@"E2_INNER"

Step 2: A new query block created with just D1 & L1 and group by operator applied at this step.

Registered qb: SEL$22220E47 0x21ealbe4
(SPLIT/MERGE QUERY BLOCKS SEL$50B3ADB4)
QUERY BLOCK SIGNATURE
signature (): qb_name=SEL$22220E47 nbfros=2 flg=0
    fro(0): flg=0 objn=71162 hint_alias="D1"@"E2_INNER"
    fro(1): flg=0 objn=71160 hint_alias="L1"@"L1_INNER"

Step 3: A new query block created joining above two query blocks.

Registered qb: SEL$35B1C696 0x21e9dd44
(UNKNOWN QUERY BLOCK ORIGIN SEL$58CDACD2; SEL$58CDACD2; LIST 3)
QUERY BLOCK SIGNATURE
signature (): qb_name=SEL$35B1C696 nbfros=2 flg=0
    fro(0): flg=0 objn=71164 hint_alias="E2"@"E2_INNER"
    fro(1): flg=5 objn=0 hint_alias="VW_GBF_3"@"SEL$35B1C696"

Calling physical optimizer to cost transformed QBP#3

Adding more metrics to older version of SQL increases logical reads, almost doubling logical reads.
Sales_so_far metric is added and this column keeps running total of sales_qty for an item and location.

-- Query block from step 2 costed below:

| signature (optimizer): qb_name=SEL$22220E47 nbfros=2 flg=0
|    fro(0): flg=0 objn=71162 hint_alias="D1"@"E2_INNER"
|    fro(1): flg=0 objn=71160 hint_alias="L1"@"L1_INNER"

Trying or-Expansion on query block SEL$22220E47 (#4)
Transfer Optimizer annotations for query block SEL$22220E47 (#4)
Final cost for query block SEL$22220E47 (#4) - All Rows Plan:
Best join order: 1
Cost: 52.0502  Degree: 1  Card: 9990.0000  Bytes: 159840
Resp: 52.0502  Resp_io: 51.0000  Resp_cpu: 23263193
kkoqbc-end:

Query block created in step 3 above costed below.

| QUERY BLOCK SIGNATURE
| signature (optimizer): qb_name=SEL$35B1C696 nbfros=2 flg=0
|    fro(0): flg=0 objn=71164 hint_alias="E2"@"E2_INNER"
|    fro(1): flg=1 objn=0 hint_alias="VW_GBF_3"@"SEL$35B1C696"

Trying or-Expansion on query block SEL$35B1C696 (#2)
Transfer Optimizer annotations for query block SEL$35B1C696 (#2)
Final cost for query block SEL$35B1C696 (#2) - All Rows Plan:
Best join order: 1
Cost: 727.0640  Degree: 1  Card: 99900.0000  Bytes: 4795200

Riyaj Shamsudeen
HOTSOS’ 2008
GBP #4
In this transformation, combination of GBP #2 and GBP #3 tried. In GBP #2, group by operator was applied at emp e2 and then D1&L1 were joined. In GBP #3, group by operator applied in D1&L1 and then emp e2 was joined. This transformation is a combination of both GBP #2 and GBP #3.

Notice that avg function has been converted to sum function to calculate average.

**Calling physical optimizer to cost transformed QBP#4**

Physical optimizer called for these query blocks to calculate final cost for this transformation.

Costing for vw_gbc_4 view:

```java
signature (optimizer): qb_name=SEL$6543C244 nbfros=1 flg=0
fro(0): flg=0 objn=71164 hint_alias="E2"@"E2.Inner"
```

Trying or-Expansion on query block SEL$6543C244 (#5)
Transfer Optimizer annotations for query block SEL$6543C244 (#5)

Riyaj Shamsudeen
HOTSOS’ 2008
Final cost for query block SEL$6543C244 (#5) - All Rows Plan:

Best join order: 1
Cost: 514.8488  Degree: 1  Card: 100000.0000  Bytes: 3000000
Resc: 514.8488  Resc_to: 509.0000  Resc_cpu: 129556005
Resp: 514.8488  Resp_to: 509.0000  Resp_cpu: 129556005
kkoqbc-subheap (delete addr=0x0C329494, in-use=9880, alloc=12356)
kkoqbc-end:

Costing for vw_gbf_5 view:

signature (optimizer): qb_name=SEL$7E9F6985 nbfros=2 flg=0
  fro(0): flg=0 objn=71162 hint_alias="D1"@"E2_INNER"
  fro(1): flg=0 objn=71160 hint_alias="L1"@"L1_INNER"

Trying or-Expansion on query block SEL$7E9F6985 (#6)
Transfer Optimizer annotations for query block SEL$7E9F6985 (#6) - All Rows Plan:

Best join order: 1
Cost: 52.0502  Degree: 1  Card: 9990.0000  Bytes: 159840
Resc: 52.0502  Resc_to: 51.0000  Resc_cpu: 23263193
Resp: 52.0502  Resp_to: 51.0000  Resp_cpu: 23263193
kkoqbc-subheap (delete addr=0x0C32A158, in-use=19164, alloc=22500)
kkoqbc-end:

Costing whole query block.

signature (optimizer): qb_name=SEL$C43B7E12 nbfros=2 flg=0
  fro(0): flg=1 objn=0 hint_alias="VW_GBF_5"@"SEL$C43B7E12"
  fro(1): flg=1 objn=0 hint_alias="VW_GBC_4"@"SEL$C54E6AB0"

Trying or-Expansion on query block SEL$C43B7E12 (#2)
Transfer Optimizer annotations for query block SEL$C43B7E12 (#2) - All Rows Plan:

Best join order: 1
Cost: 749.9118  Degree: 1  Card: 9990.0000  Bytes: 769230
Resc: 749.9118  Resc_to: 741.0000  Resc_cpu: 197405395
Resp: 749.9118  Resp_to: 741.0000  Resp_cpu: 197405395
kkoqbc-subheap (delete addr=0x0C324424, in-use=18652, alloc=22500)
kkoqbc-end:

call(in-use=167804, alloc=212872), compile(in-use=509756, alloc=517176),
execution(in-use=540080, alloc=544052)
kkoqbc: finish optimizing query block SEL$C43B7E12 (#2)
CBQT: Saved costed qb# 5 (SEL$6543C244), key = SEL$6543C244_00001200_2
CBQT: Saved costed qb# 6 (SEL$7E9F6985), key = SEL$7E9F6985_00001200_2
CBQT: Saved costed qb# 2 (SEL$C43B7E12), key = SEL$C43B7E12_00000042_0
GBP: Updated best state, Cost = 749.91

SJC – Set Join Conversion

Some set operations can be transformed to Join operation. Having join operations opens up more options to optimize, for the optimizer. A new SQL introduced below to explain this transformation.

Select /*+ qb_name (e1_outer) */ * from
  emp e1, dept d1, locations l1
  where
    e1.dept_id = d1.dept_id and
    d1.location_id = l1.location_id and
    e1.salary >100000
intersekt
Select /*+ qb_name (e2_outer) */ * from
  emp e2, dept d2, locations l2
  where
    e2.dept_id = d2.dept_id and
    d2.location_id = l2.location_id and
    hire_date> sysdate-365*10
In the above SQL, there are two branches connected by an intersect operator.

**Transformed SQL**

Above SQL is transformed to following SQL and all row sources have been converted to join operators.

```
select distinct e1.emp_id, e1.emp_name, e1.dept_id,
       e1.salary, e1.hire_Date, d1.dept_id, d1.dept_name,
       d1.location_id, l1.location_id, l1.city_name, l1.state
from
  emp e2, dept d2, locations l2, emp e1, dept d1, locations l1
where
  e1.emp_id = e2.emp_id and
  sys_op_mapNonNull(e1.emp_name) =
    sys_op_mapNonNull(e2.emp_name) and
  e1.dept_id = e2.dept_id and e1.salary = e2.salary and
  e1.hire_date = e2.hire_Date and d1.dept_id = d2.dept_id and
  sys_op_mapNonNull(d1.dept_name) =
    sys_op_mapNonNull(d2.dept_name) and
  d1.location_id = d2.location_id and
  l1.location_id = l2.location_id and
  sys_op_mapNonNull(l1.city_name) =
    sys_op_mapNonNull(l2.city_name) and
  sys_op_mapNonNull(l1.state) =
    sys_op_mapNonNull(l2.state) and
  e1.dept_id = d1.dept_id and d1.location_id = l1.location_id and
  e1.salary > 100000 and e2.dept_id = d2.dept_id and
  d2.location_id = l2.location_id and
  e2.hire_date - sysdate() - 365 * 10
```

Lines (1) above shows that all tables are connected with join operator.

Lines (2) to (4) added due to avoid duplicates in lieu of self join operations. For example, in line (2) above, e1.emp_id = e2.emp_id is joined. In the next line, `sys_op_mapNonNull` function is used to handle null values.

Original predicates are listed from line (4)

Picture below pictorially represents transformed SQL. As shown below, a straightforward query graph is converted to complex query graph with self join between various row sources.

---

5 `Sys_op_mapNonNull` is to handle null values. It is a fact that null != null. Since column `emp_name` above can be null, to simulate null = null condition in the transformed SQL, this function is used. It is basically equivalent to writing `(emp_name is null or emp_name=emp_name)`. Note that this function is applied to all non-null columns.

Riyaj Shamsudeen
HOTSOS' 2008
Parameters
SJC is disabled by default. Setting parameter _convert_set_to_join to true enables this transformation.

Trace lines
Following trace lines shows that how this set operation converted to join operation.

SJC: Considering set-join conversion in query block SET$1 (#0)
*****************************************************************************
Set-Join Conversion (SJC)
*****************************************************************************
SJC: Checking validity of SJC on query block SET$1 (#0)
SJC: Passed validity checks.
SJC: SJC: Applying SJC on query block SET$1 (#0)
Registered qb: SET$09AAA538 0x1ded2d00 (SET QUERY BLOCK SET$1; SET$1)
*****************************************************************************
QUERY BLOCK SIGNATURE
signature (): qb_name=SET$09AAA538 nbfros=2 flg=0
fro(0): flg=1 objn=0 hint_alias="NULL_HALIAS"@"SET$09AAA538"
fro(1): flg=1 objn=0 hint_alias="NULL_HALIAS"@"SET$09AAA538"

Registered qb: SEL$02B15F54 0x1ded2d00 (VIEW MERGE SET$09AAA538; SEL$1 SEL$2)
*****************************************************************************
QUERY BLOCK SIGNATURE
signature (): qb_name=SEL$02B15F54 nbfros=6 flg=0
fro(0): flg=0 objn=73174 hint_alias="D1"@"SEL$1"
fro(1): flg=0 objn=73176 hint_alias="E1"@"SEL$1"

Again, do not use underscore parameters without Oracle support blessing.

Riyaj Shamsudeen
HOTSOS’ 2008
Join predicates can be pushed down to various query blocks. If predicates applied at optimal query block, then execution cost can be lower. Following SQL is used to explain this concept.

```
( select /*+ qb_name (v1) */ e1.*, d1.location_id from emp e1, dept d1
where e1.dept_id = d1.dept_id ) v1,
( select/*+ qb_name (v2) */
  avg(salary) avg_sal_dept, d2.dept_id from
  emp e2,  dept d2, locations l2
where
  e2.dept_id = d2.dept_id and
  l2.location_id=d2.location_id
  group by d2.dept_id ) v2
where
  v1.dept_id =v2.dept_id and v1.salary > v2.avg_sal_dept
  and v1.dept_id=100
```

**Trace lines**

Following trace lines shows that one of the predicate is pushed from outer query block to query block v2. Many other predicates can be pushed effectively too.

**JE elimination**

This transformation is a heuristics based transformation. If joins can be eliminated, performance can be improved.

In the SQL below, xists operator is applied to see if location_id exists in the locations table from departments table query block.
exists (select 1 from locations l1 where l1.location_id = d1.location_id );

Plan hash value: 762406906

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>91</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS SEMI</td>
<td></td>
<td>1</td>
<td>91</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>DEPT</td>
<td>1</td>
<td>78</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 3</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C0010444</td>
<td>1</td>
<td>13</td>
<td>0 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Exists operator is implemented as a semi-join in the explain plan. But, if there is a referential key constraint between departments and locations table, then semi-join to locations table can be eliminated. In the explain plan output, semi-join to locations table is missing.

```
alter table dept add constraint dept_fk foreign key (location_id )
    references locations (location_id);
```

```
explain plan for

select d1.* from dept d1 where
exists (select 1 from locations l1 where l1.location_id = d1.location_id );
```

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS FULL</td>
<td>DEPT</td>
<td>1</td>
<td>78</td>
<td>2</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information:
----------------------
1 - filter("D1"."LOCATION_ID" IS NOT NULL)
```

Parameter _optimizer_join_elimination_enabled controls this behavior.

**Trace lines**

JE: Considering Join Elimination on query block SEL$5DA710D3 (#1)

```
Join Elimination (JE)

JE: cfro: DEPT objn:73501 col#:3 dfro:LOCATIONS dcol#:3
Query block (1e68c358) before join elimination:
SQL:****** UNPARSED QUERY IS *****
SELECT "D1"."DEPT_ID" "DEPT_ID","D1"."DEPT_NAME" "DEPT_NAME",
"D1"."LOCATION_ID" "LOCATION_ID"
FROM "CBQT2"."LOCATIONS" "L1","CBQT2"."DEPT" "D1"
WHERE "D1"."LOCATION_ID"="L1"."LOCATION_ID"
```

```
JE: eliminate table: LOCATIONS
Registered qb: SEL$48A72308 0x1e68c358
 (JOIN REMOVED FROM QUERY BLOCK SEL$5DA710D3; SEL$5DA710D3; "L1""@"SEL$2")
```

```
SQL:****** UNPARSED QUERY IS *****
SELECT "D1"."DEPT_ID" "DEPT_ID","D1"."DEPT_NAME"
"DEPT_NAME","D1"."LOCATION_ID"
"LOCATION_ID" FROM "CBQT2"."DEPT" "D1" WHERE "D1"."LOCATION_ID" IS NOT NULL
```

From above trace lines, it is evident that join to locations table is eliminated and rewritten query does not have reference to locations table.

Riyaj Shamsudeen
HOTSOS’ 2008
**PM: Predicate move around**

Predicates can be moved around from one query block to another query block and applied. Following SQL is used to illustrate this feature.

```sql
explain plan for
select /*+ qb_name (v_outer) */ v1.* from
 (select /*+ qb_name (v1) no_merge */ e1.*, d1.location_id from emp e1, dept d1
  where e1.dept_id = d1.dept_id and d1.dept_id=200) v1,
 (select/*+ qb_name (v2) no_merge */ avg(salary) avg_sal_dept, d2.dept_id from
  emp e2, dept d2, locations l2
  where e2.dept_id = d2.dept_id and l2.location_id=d2.location_id and d2.dept_id=100
  group by d2.dept_id
) v2_dept
where
  v1.dept_id = v2_dept.dept_id and v1.salary > v2_dept.avg_sal_dept
  and v2_dept.dept_id=300
;
```

Following trace lines shows that how predicates are pulled up and then pushed down to optimal query blocks.

```
***********************
Predicate Move-Around (PM)
***********************
PM:   Passed validity checks.
PM:   Pulled up predicate "V1"."DEPT_ID"=300
      from query block V1 (#2) to query block V_OUTER (#1)
PM:   Pulled up predicate "V1"."EMP_ID","E1"."DEPT_ID"=300
      from query block V1 (#2) to query block V_OUTER (#1)
PM:   Pulled up predicate "V1"."DEPT_ID"=200
      from query block V1 (#2) to query block V_OUTER (#1)
PM:   Pulled up predicate "V2_DEPT"."DEPT_ID"=100
      from query block V2 (#3) to query block V_OUTER (#1)
PM:   Pushed down predicate "E1"."DEPT_ID"=100
      from query block V_OUTER (#1) to query block V1 (#2)
PM:   Pushed down predicate "D2"."DEPT_ID"=200
      from query block V_OUTER (#1) to query block V2 (#3)
```

**Steps**

There are three strategies in predicate move around transformation.

1. Predicate pull ups: Predicates are pulled out of inner query blocks to outer query blocks.
2. Predicate push down: Predicates are pushed down from outer query blocks to inner query blocks.

**CNT(col) to CNT(*)**

Count of a not null column can be transformed to count(*).

```sql
explain plan for
select count(emp_id) from emp e1
where exists ( select 1 from emp e2, dept d2 where e2.dept_id = d2.dept_id
  and e1.dept_id = e2.dept_id )
  and e1.hire_date > sysdate-90
```

Riyaj Shamsudeen
HOTSOS’ 2008
Trace lines

CNT: Considering count(col) to count(*) on query block SEL$1 (#0)
*************************
Count(col) to Count(*) (CNT)
*************************
CNT: Converting COUNT(EMP_ID) to COUNT(*).
CNT: COUNT() to COUNT(*) done.

Parameters

Parameter: _optimizer_cost_based_transformation

Above parameter controls cost based transformation. Various values possible:

- two_pass
- on
- exhaustive
- linear
- iterative
- off

Setting this parameter value to off disables cost based transformation.

Other Parameters

Many transformation discussed in this paper can be selectively disabled by altering these parameters at session level.

_unnest_subquery = true # Unnest subquery
_eliminate_common_subexpr = true # Eliminate common sub expression
_pred_move_around = true # Predicate move around
_convert_set_to_join = false # Convert set to join
_optimizer_order_by_elimination_enabled = true # Order by elimination check
_optimizer_distinct_elimination = true # Distinct elimination
_optimizer_multi_level_push_pred = true # push predicates multiple level
_optimizer_group_by_placement = true # Consider group by placement
_optimizer_reuse_cost_annotations = true # Reuse cost annotations

Memory usage

Memory allocated during transformation is allocated as sub heaps. Sub heaps can be deallocated without dependency to parent heaps. These sub heaps are needed only during parsing can be deallocated before execution step.

Following parameters controls how these sub heaps are allocated and freed.

_optimizer_use_subheap = true # Use sub heap
_optimizer_free_transformation_heap = true # Free heaps after transformation
_optimizer_or_expansion_subheap = true

Following trace lines shows that sub heaps are allocated and deallocated at the end of each transformation. This reduces memory footprint needed in case of complex SQLs.

```
kkoqbc-subheap (create addr=0x090AC150)
...
kkoqbc-subheap (delete addr=0x090AC150, in-use=28616, alloc=31212)
```
Model & CBQT

CBQT is disabled for SQL with Model SQL.

```sql
select a.* from (
    select item, location, week, inventory, sales_so_far, sales_qty, rcpt_qty
    from item_data
    model return updated rows
    partition by (item)
    dimension by (location, week)
    measures (0 inventory, sales_qty, rcpt_qty, 0 sales_so_far)
    rules sequential order (inventory [location, week] = nvl(inventory [cv(location), cv(week)-1], 0)
        - sales_qty [cv(location), cv(week)],
        + rcpt_qty [cv(location), cv(week)],
        sales_so_far [location, week] = sales_qty [cv(location), cv(week)] + nvl(sales_so_far [cv(location), cv(week)-1], 0)
    )
    order by item, location, week
) a, locations l
where a.location = l.location_id;
```

Trace lines

<table>
<thead>
<tr>
<th>Query transformations (QT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBQT bypassed for query block SEL$1 (#0): contained query block.</td>
</tr>
<tr>
<td>CBQT: Validity checks failed for 4um58uddcnx2k.</td>
</tr>
</tbody>
</table>

Analytical functions & CBQT

CBQT is enabled for analytic functions though.

```sql
explain plan for
select a.*, i.item
from (select item, location, week, sales_qty, rcpt_qty,
    sum(sales_qty) over (partition by item, location
    order by week
    rows between unbounded preceding and current row
    ) running_sales_total
) a, item_data i
where a.item = i.item
```

Trace lines

| Check Basic Validity for Non-Union View for query block SEL$2 (#0) |
| CBQT: Validity checks passed for 6rd24ujfu08s5. |
| CSE: Considering common sub-expression elimination in query block SEL$1 (#0) |

CTAS & CBQT

CBQT is disabled for CBQT.

```sql
create table cbqt_test as
select /*+ qb_name (e1_outer) */ * from emp e1 where salary >
    (select/*+ qb_name (e2_inner) */ avg(salary) from emp e2, dept d1
    where e1.dept_id = e2.dept_id and e2.dept_id = d1.dept_id
    and exists
```

Riyaj Shamsudeen
HOTSOS’ 2008
(select /*+ qb_name (l1_inner) */ 1 from locations l1
  where l1.location_id=d1.location_id )
and e1.hire_date > sysdate - (10*365)

Trace lines
******************************************************************************
Predicate Move-Around (PM)  
******************************************************************************
PM:      PM bypassed: Not a SELECT statement

Cost-Based Group By Placement
******************************************************************************
GBP: Checking validity of GBP for query block SEL$58CDACD2 (#2)
GBP: Checking validity of group-by placement for query block SEL$58CDACD2 (#2)
GBP: Bypassed: create table.

Performance comparison

_optimizer_cost_based_transformation=linear  _optimizer_cost_based_transformation=OFF
Elapsed: 00:00:05.60    Elapsed: 00:00:39.61

Statistics

<table>
<thead>
<tr>
<th></th>
<th>0 recursive calls</th>
<th>33 recursive calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>db block gets</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>consistent gets</td>
<td>11206</td>
<td>14481</td>
</tr>
<tr>
<td>physical reads</td>
<td>10892</td>
<td></td>
</tr>
<tr>
<td>redo size</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bytes sent via SQL*Net to client</td>
<td>6556650</td>
<td>6556650</td>
</tr>
<tr>
<td>bytes received via SQL*Net from client</td>
<td>110911</td>
<td>110911</td>
</tr>
<tr>
<td>SQL*Net roundtrips to/from client</td>
<td>10047</td>
<td>10047</td>
</tr>
<tr>
<td>sorts (memory)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>sorts (disk)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rows processed</td>
<td>150689</td>
<td>150689</td>
</tr>
</tbody>
</table>

Riyaj Shamsudeen
HOTSOS’ 2008
Reference

[1] Cost based query transformation in Oracle – VLDB Sept 06
   Rafi Ahmed, Allison Lee, Andrew Witkowski, Dinesh Das, Hong Su, Mohamed Zait
   Thierry Cruanes

   Rafi Ahmed, Oracle Corporation

   Alon Yitzchak Levy, Inderpal Singh Mumick,

[4] Cost Based Oracle Fundamentals By Jonathan Lewis, APRESS publication,
   ISBN 1-59059-636-6


About the author

Riyaj Shamsudeen has 15+ years of experience in Oracle and 14+ years as an Oracle
DBA/ERP Financials DBA. He currently works for Cingular (New AT&T), specializes in
performance tuning and database internals. He has authored few articles such as internals of
locks, internals of hot backups, redo internals etc. He also teaches in community colleges in
Dallas such as North lake college. He was a board member for DOUG (Dallas Oracle User
Group).

When he is not dumping the database blocks, he can be seen playing soccer with his kids.
Appendix #1: Environment details

Windows XP
Oracle version 11.1.0.6
No special configurations such as RAC/Shared server etc.
Locally managed tablespaces
No ASM
No ASSM

And

Linux CentOS 4.3
Oracle version 11g
No special configurations such as RAC/Shared server etc.
Locally managed tablespaces
No ASM
No ASSM