Five Hints for Optimal SQL

Jonathan Lewis
jonathanlewis.wordpress.com
www.jlcomp.demon.co.uk

Who am I?

Independent Consultant
32+ years in IT
27+ using Oracle
Strategy, Design, Review, Briefings, Educational, Trouble-shooting
Oracle author of the year 2006
Select Editor’s choice 2007
UKOUG Inspiring Presenter 2011
ODTUG 2012 Best Presenter (d/b)
UKOUG Inspiring Presenter 2012
UKOUG Lifetime Award (IPA) 2013
Member of the Oak Table Network
Oracle ACE Director
O1 visa for USA
The Problem

• "Left-deep"
• Bad choice of transformation
• Restrictions on transformations
• *Bad Estimates*
• *Bad join order*

Left-deep tree

The optimizer’s strategy is always to work towards a ‘left-deep’ tree. We traverse the tree from the bottom left corner going across then up.
But sometimes the left-deep tree cannot give a good path, and we have to push Oracle into a “bushy” tree – usually with /*+ no_merge */ hints.

Example (a)

```sql
select ...
from  t1,
   (   select ..., count(t3.v2)
       from  t2, t3
       where exists (   
           select ... where t2.v1 = ...
       )
       and  t3.n1 = t2.n1
       group by ...
   ) v1,
   t4
where  
   v1.n1 = t1.id
and  t4.id(+) = v1.n2
;
```

We may have a query that looks like this, which we have written with this layout because it shows (to the human eye) a good strategy for executing it.
Example (b)

- "Shaping" the query plan

- Your vision
  - ( ( t1, ( ( t2, subquery ), t3 ) ), t4 )

- Oracle's plan
  - ( ( ( t1, t2 ), t3 ), {unnested subquery} ), t4 )

We may have to tell the optimizer what our vision is because it will have a strong bias to get rid of the inline view and filter subquery if it can.

Example (c)

```
select /*+ push_pred(v1) */
from t1,
 ( select /*+ no_merge */ ...
from t2, t3
where exists ( 
    select /*+ no_unnest push_subq */ ...
    )
and t3.n1 = t2.n1
group by ...
) v1,
 t4
where
  v1.n1 = t1.id
and t4.(+d) = v1.n2
;
```

At the high level, this small number of inline hints will describe our vision without trying to micro-manage the optimizer every step of the way.
Example (d)

```
select /*+ qb_name(main) push_pred(v1@main) 
    no_merge(@inline) 
    no_unnest(@subq1) push_subq(@subq1) */ 
from t1, 
  ( select /*+ qb_name(inline) */ ... 
    from t2, t3 
    where exists ( 
      select /*+ qb_name(subq1) */ ... 
    ) 
    and t3.n1 = t2.n1 
    group by ... 
  ) v1, 
  t4 
where v1.n1 = t1.id and t4.id(+) = v1.n2;
```

By adding query block names to the query we can put all the remaining hints at the start of the statement.

Example (e)

```
LEADING(@SEL$F0E3EC4B T2@INLINE T3@INLINE) 
INDEX_RS_ASC(@SEL$F0E3EC4B T2@INLINE (T2.N1)) 
USE_NL(@SEL$F0E3EC4B T3@INLINE) 
INDEX_RS_ASC(@SEL$F0E3EC4B T3@INLINE (T3.N1)) 
PUSH_SUBQ(@SUBQ1) 
INDEX(@SUBQ1 T5@SUBQ1 (T5.ID))  
OUTLINE(@MAIN) OUTLINE(@INLINE) OUTLINE(@SUBQ1) 
LEADING(@MAIN T1@MAIN V1@MAIN T4@MAIN) 
FULL(@MAIN T1@MAIN) 
OUTLINE_LEAF(@MAIN) 
OUTLINE_LEAF(@SEL$F0E3EC4B) 
OUTLINE_LEAF(@SUBQ1) 
USE_NL(@MAIN V1@MAIN) 
NO_ACCESS(@MAIN V1@MAIN) 
PUSH_PRED(@MAIN V1@MAIN 1) 
USE_HASH(@MAIN T4@MAIN) 
FULL(@MAIN T4@MAIN)  
ALL_ROWS 
DB_VERSION('12.1.0.2') 
OPTIMIZER_FEATURES_ENABLE('12.1.0.2') 
IGNORE_OPTIM_EMBEDDED_HINTS
```

The 13 hints on the left are the optimizer's micro-management. The outline_leaf hints on the right show it the final query blocks it optimizes.
The List

• (no_)merge
  – Three forms, easiest in-line, can be inside a with subquery
  – Naming the query block, in the inline view, naming the view

• (no_)push_pred
  – Push join predicate inside (non-mergeable) complex view - some limitations

• (no)_unnest
  – Two forms, easiest in the subquery

• (no_)push_subq
  – changed syntax in 10g, now two forms, easiest in the subquery

• driving_site()
  – Optimizer can choose driving site based on cost of network traffic

Complex View Merging

create or replace view avg_val_view as
select
  id_parent, avg(val) avg_val_t1
from t2
group by
  id_parent;

select
  /*+ no_merge(avg_val_view) push_pred(avg_val_view) */
  t1.vc1, avg_val_t1
from t1,
  avg_val_view
where
  t1.vc2 = 'XYZ'
and avg_val_view.id_parent = t1.id_parent
;
### Unhinted

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>23</td>
<td>2024</td>
<td>39</td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td>23</td>
<td>2024</td>
<td>39</td>
</tr>
<tr>
<td>*2</td>
<td>HASH JOIN</td>
<td></td>
<td>32</td>
<td>2816</td>
<td>8</td>
</tr>
<tr>
<td>*3</td>
<td>TABLE ACCESS FULL</td>
<td>T1</td>
<td>1</td>
<td>81</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS FULL</td>
<td>T2</td>
<td>1024</td>
<td>7168</td>
<td>5</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):
2 - `access("ID_PARENT"="T1"."ID_PARENT")`
3 - `filter("T1"."VC2"='XYZ')`

The optimizer has joined then aggregated - which could be a disaster if it got the estimates wrong.

### no_merge() hint only

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>95</td>
<td>40</td>
</tr>
<tr>
<td>*1</td>
<td>HASH JOIN</td>
<td></td>
<td>1</td>
<td>95</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>JOIN FILTER CREATE</td>
<td>:BF0000</td>
<td>1</td>
<td>69</td>
<td>2</td>
</tr>
<tr>
<td>*3</td>
<td>TABLE ACCESS FULL</td>
<td>T1</td>
<td>1</td>
<td>69</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>VIEW</td>
<td>AVG_VAL_VIEW</td>
<td>32</td>
<td>832</td>
<td>37</td>
</tr>
<tr>
<td>5</td>
<td>HASH GROUP BY</td>
<td>AVG_VAL_VIEW</td>
<td>32</td>
<td>224</td>
<td>37</td>
</tr>
<tr>
<td>6</td>
<td>JOIN FILTER USE</td>
<td>:BF0000</td>
<td>1024</td>
<td>7168</td>
<td>5</td>
</tr>
<tr>
<td>*7</td>
<td>TABLE ACCESS FULL</td>
<td>T2</td>
<td>1024</td>
<td>7168</td>
<td>5</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):
1 - `access("AVG_VAL_VIEW"."ID_PARENT"="T1"."ID_PARENT")`
3 - `filter("T1"."VC2"='XYZ')`
7 - `filter(SYS_OP_BLOOM_FILTER(:BF0000,"ID_PARENT"))`

The Bloom filter appeared for the first time in this example when I finally got to 11gR2.
no_merge() + push_pred()

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>82</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>82</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>T1</td>
<td>1</td>
<td>69</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>VIEW PUSHED PREDICATE</td>
<td>AVG_VAL_VIEW</td>
<td>1</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TABLE ACCESS FULL</td>
<td>T2</td>
<td>32</td>
<td>224</td>
<td>5</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):
2 - filter("T1"."VC2"='XYZ')
4 - filter(COUNT(*)>0)
6 - filter("ID_PARENT"="T1"."ID_PARENT")

Forcing predicate pushdown in this example was actually a bad idea; but you can see the join predicate has moved inside the view.

Inline view

```sql
select
t1.id, max_small
from
t1,
(
    select /*+ no_merge push_pred */
        id_p, max(small_num_c) max_small
    from t2
    group by
        id_p
) v2
where
    v2.id_p = t1.id
and
    t1.small_num_p between 10 and 20
;
```

The mechanism applies equally well to in-line views, whether you write them in-line as above, or format them with subquery factoring.
Subquery Factoring

```sql
with max_view as (  
  select /*+ no_merge push_pred cardinality(NNN) */  
    id_p, max(small_num_c) max_small  
  from t2  
  group by  
    id_p  
)  
select  
  t1.id, max_small  
from  
  t1,  
  max_view v2  
where  
  v2.id_p = t1.id  
and  
  t1.small_num_p between 10 and 20
```

Generally (in modern versions of Oracle) when factored subquery is moved inline the plan you get matches the original inline plan.

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>60</td>
<td>1260</td>
<td>427</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>60</td>
<td>1260</td>
<td>427</td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>T1</td>
<td>60</td>
<td>480</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>VIEW PUSHED PREDICATE</td>
<td></td>
<td>1</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>* 4</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>T2</td>
<td>5</td>
<td>40</td>
<td>7</td>
</tr>
<tr>
<td>* 7</td>
<td>INDEX RANGE SCAN</td>
<td>T2_PK</td>
<td>5</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
```

Predicate Information (identified by operation id):

- 2 - filter("T1"."SMALL_NUM_P">=10 AND "T1"."SMALL_NUM_P"<=20)
- 4 - filter(COUNT(*)>0 AND 20>=10)
- 7 - access("ID_P"="T1"."ID")

In this case the only difference between the plan from the stored view and the inline/CTE version is the absence of the stored view name at operation 3.
No_merge limitation (a)

```sql
select
    round(sum(ceil(len/8100)) * 8/1024,0)    used_mb
from

    (select
        /*+ no_merge */
        dbms_lob.getlength(c1) len
    from
tbl
)
where
    len > &lob_limit

;```

No_merge limitation (b)

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>VIEW</td>
<td></td>
<td>6</td>
<td>78</td>
<td>2</td>
</tr>
<tr>
<td>* 3</td>
<td>TABLE ACCESS FULL</td>
<td>TBL</td>
<td>6</td>
<td>522</td>
<td>2</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

3 - filter("DBMS_LOB"."GETLENGTH"("C1")>4000)

Workarounds

```sql
select rownum rn, dbms_lob.get_length()...

with subq as (select /*+ materialize */ ...```
SELECT ...
FROM SIEBEL.S_ACT_EMP T1,
     SIEBEL.CX_CORR_DATE T3,
     SIEBEL.S_CONTACT T5,
     SIEBEL.S_ASSET T7,
     SIEBEL.S_FN_APPR T9,
     SIEBEL.S_CONTACT_X T11,
     SIEBEL.S_PROJITEM T13,
     SIEBEL.S_ADDR_PER T6,
     SIEBEL.S_EVT_ACT_FNX T14,
     SIEBEL.S_ADDR_PER T17,
     SIEBEL.S_OPTY T19,
     SIEBEL.S_EVT_ACT X T2,
     SIEBEL.S_EVT_ACT_SS T4,
     SIEBEL.S_ADDR_PER T6,
     SIEBEL.S_EVT_ACT SS T4,
     SIEBEL.S_EVT_ACT_FNX T14,
     SIEBEL.S_EVT_ACT T20
WHERE ...
ORDER BY T20.CREATED DESC

A Siebel example (but it could be Oracle Financials, Peoplesoft, SAP etc.
When a query has a lot of tables the optimizer can easily start badly.

SELECT ...
FROM (select /*+ no_merge */ ...
from SIEBEL.S_EVT_ACT T20,
     SIEBEL.S_EVT_ACT_X T2, SIEBEL.S_EVT_ACT_SS T4,
     SIEBEL.S_EVT_ACT_FNX T14, SIEBEL.S_EVT_ACT T20
) v1,
     SIEBEL.S_ACT_EMP T1,
     SIEBEL.CX_CORR_DATE T3,
     SIEBEL.S_ADDR_PER T6,
     SIEBEL.S_EVT_ACT T20,
     SIEBEL.S_ADDR_PER T6,
     SIEBEL.S_EVT_ACT_SS T4,
     SIEBEL.S_EVT_ACT_FNX T14, SIEBEL.S_EVT_ACT T20
WHERE ...
ORDER BY v1.CREATED DESC

This approach tends to double the length of the SQL (most of the select list
appears twice). A simple leading(t20, t2, t4, t14, t16) might be sufficient.
"Unmerge" - everything changes (a)

select
    p.product_name,
    max(decode(s.year,2010, s.quantity)) quan_2010,
    max(decode(s.year,2011, s.quantity)) quan_2011,
    max(decode(s.year,2012, s.quantity)) quan_2012
from
    products p,
    sales s
where
    s.product_id = p.product_id
group by
    p.product_name
;

Interestingly a recent version of the optimizer added the capability to do the inverse of complex view merging for aggregates.

"Unmerge" - everything changes (b)

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>3</td>
<td>144</td>
<td>26</td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td>3</td>
<td>144</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td>3</td>
<td>144</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCTS</td>
<td>4</td>
<td>36</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>VIEW PUSHED PREDICATE</td>
<td>VW_GBC_1</td>
<td>1</td>
<td>39</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>FILTER</td>
<td></td>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>SALES</td>
<td>3</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>INDEX RANGE SCAN</td>
<td>SAL_I1</td>
<td>3</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):
5  - filter(COUNT(*)>0)
8  - access("S"."PRODUCT_ID"="P"."PRODUCT_ID")

So you need to know about "group by placement" or the no_place_group_by hint or /*+ opt_params("optimizer_group_by_placement",false) */
In this case we start with a subquery which is a correlated subquery with aggregation.

```sql
select
    outer.*
from
    emp outer
where
    outer.sal > (  
        select
            avg(inner.sal)
        from emp inner
        where inner.dept_no = outer.dept_no
    )
;
```

Unnesting (b)

```sql
select  -- notional unnesting transformation
    outer.*
from
    (  
        select  dept_no, avg(sal) av_sal  
        from emp
        group by dept_no
    ) inner,
    emp outer
where
    outer.dept_no = inner.dept_no
and
    outer.sal > inner.av_sal;
```
Unnesting (c)

```
select -- possible view merge transformation
    outer.dept_no dept_no, outer.sal sal,
    outer.emp_no emp_no, outer.padding padding
from
    test_user.emp inner, test_user.emp outer
where
    inner.dept_no = outer.dept_no

group by
    inner.dept_no, outer.rowid,
    outer.padding, outer.emp_no,
    outer.sal, outer.dept_no

having
    outer.sal > avg(inner.sal)
```

Unnesting (d)

In the subquery

```
/*+ no_unnest */

/*+ unnest no_merge */

/*+ unnest merge */
```

In the containing query

```
/*+ no_unnest(@qb_name) */

/*+ unnest(@qb_name) no_merge(@qb_name) */

/*+ unnest(@qb_name) merge(@qb_name) */
```
Subquery pushdown (a)

```sql
select
t1.v1
from t1, t3
where t1.n2 = 15
/*
and exists (select --+ no_unnest
null
from t2
where t2.n1 = 15
and t2.id = t1.id
)
*/
and t3.n1 = t1.n1
and t3.n2 = 15
;
```

If a subquery doesn’t unnest it may run early or (as it generally does) it may run late in the execution of the query. (pushed down the tree).

Subquery pushdown (b)

**Execution Plan 11.2.0.4 – without subquery**

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>173</td>
<td>3979</td>
<td>193</td>
</tr>
<tr>
<td>* 1</td>
<td>HASH JOIN</td>
<td></td>
<td>173</td>
<td>3979</td>
<td>193</td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>T1</td>
<td>157</td>
<td>2355</td>
<td>96</td>
</tr>
<tr>
<td>* 3</td>
<td>TABLE ACCESS FULL</td>
<td>T3</td>
<td>157</td>
<td>1256</td>
<td>96</td>
</tr>
</tbody>
</table>

**Execution Plan 11.2.0.4 – with subquery**

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>28</td>
<td>366</td>
</tr>
<tr>
<td>* 1</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>HASH JOIN</td>
<td></td>
<td>173</td>
<td>4844</td>
<td>193</td>
</tr>
<tr>
<td>* 3</td>
<td>TABLE ACCESS FULL</td>
<td>T1</td>
<td>157</td>
<td>3140</td>
<td>96</td>
</tr>
<tr>
<td>* 4</td>
<td>TABLE ACCESS FULL</td>
<td>T3</td>
<td>157</td>
<td>1256</td>
<td>96</td>
</tr>
<tr>
<td>* 5</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>T2</td>
<td>1</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>* 6</td>
<td>INDEX UNIQUE SCAN</td>
<td>T2_PK</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Subquery pushdown (c)

```
select
  t1.v1
from t1, t3
where t1.n2 = 15
and  exists (select --+ no_unnest push_subq
        null
            from t2
        where t2.n1 = 15
            and t2.id = t1.id
      )
  and t3.n1 = t1.n1
  and t3.n2 = 15
;
```

Warning: The syntax for push_subq() changed from 9i to 10g

Subquery pushdown (d)

```
Execution Plan 11.2.0.4 – subquery not pushed

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>28</td>
<td>366</td>
</tr>
<tr>
<td>1</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HASH JOIN</td>
<td></td>
<td>173</td>
<td>4844</td>
<td>193</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>T1</td>
<td>157</td>
<td>3140</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS FULL</td>
<td>T3</td>
<td>157</td>
<td>1256</td>
<td>96</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>T2</td>
<td>1</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>INDEX UNIQUE SCAN</td>
<td>T2_PK</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Execution Plan 11.2.0.4 – pushed subquery

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>9</td>
<td>252</td>
<td>195</td>
</tr>
<tr>
<td>1</td>
<td>HASH JOIN</td>
<td></td>
<td>9</td>
<td>252</td>
<td>193</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>T1</td>
<td>8</td>
<td>160</td>
<td>96</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>T2</td>
<td>1</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>INDEX UNIQUE SCAN</td>
<td>T2_PK</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS FULL</td>
<td>T3</td>
<td>157</td>
<td>1256</td>
<td>96</td>
</tr>
</tbody>
</table>
```

Pushed subqueries break the "first child first" rule of thumb for reading plans. (A "missing" filter operation that distorts the shape of the plan.)
Distributed Effects (a)

create table dist_home (dist_away) as
select
    rownum   id,
    rpad(rownum,10) small_vc,
    rpad(rownum,200) large_vc
from all_objects
where rownum <= 2000;

alter table dist_home (dist_away)
    add constraint dh_pk (da_pk) primary key (id);

create public database link test@loopback using 'test';

Distributed Effects (a)

select
    /*+ driving_site (dh) */
    dh.small_vc,
    da.large_vc -- note required columns
from
    dist_home     dh,
    dist_away@test@loopback da
where
    dh.small_vc like '12%' -- small amount of data
and    da.id = dh.id
;
### Distributed Effects (b)

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Inst</th>
<th>IN-OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>DIST_HOME</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 3</td>
<td>REMOTE</td>
<td>DIST_AWAY</td>
<td>1</td>
<td>TEST</td>
<td>R-&gt;S</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

- 2 - filter("DH"."SMALL_VC" LIKE '12%')

Remote SQL Information (identified by operation id):

- 3 - SELECT "ID","LARGE_VC" FROM "DIST_AWAY" "DA" WHERE "ID"=:1
  (accessing 'TEST.LOCALDOMAIN@LOOPBACK')

### Distributed Effects (c)

Change the filter predicate to select more data:

\[ dh\.small_vc \text{ like } '1\%' \]

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Inst</th>
<th>IN-OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>216</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 1</td>
<td>HASH JOIN</td>
<td></td>
<td>216</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>DIST_HOME</td>
<td>216</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 3</td>
<td>REMOTE</td>
<td>DIST_AWAY</td>
<td>2000</td>
<td>TEST</td>
<td>R-&gt;S</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

- 1 - access("DA"."ID"="DH"."ID")
- 2 - filter("DH"."SMALL_VC" LIKE '1\%')

Remote SQL Information (identified by operation id):

- 3 - SELECT "ID","LARGE_VC" FROM "DIST_AWAY" "DA"
  (accessing 'TEST.LOCALDOMAIN@LOOPBACK')
Distributed Effects (d)

“Tune” the hash join by changing the driving site to the away database

/*+ driving_site (da) */

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Inst</th>
<th>IN-OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>216</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 1</td>
<td>HASH JOIN</td>
<td></td>
<td>216</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>REMOTE</td>
<td>DIST_HOME</td>
<td>216</td>
<td>!</td>
<td>R-&gt;S</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>DIST_AWAY</td>
<td>2000</td>
<td>TEST</td>
<td></td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):
1 - access("A1"."ID"="A2"."ID")

Remote SQL Information (identified by operation id):
2 - SELECT "ID","SMALL_VC" FROM "DIST_HOME" "A2"
   WHERE "SMALL_VC" LIKE '1%' (accessing '!')

Multi-table distributed join (a)

select
  sale_date, product, site, qty, profit
from
  sales@&m_target sal,
  sites sit,
  products@&m_target prd
where
  sit.id = sal.site
and  prd.id = sal.product
and  prd.promoted > cast ('&m_test_date' as date)
;
## Multi-table distributed join (b)

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Inst</th>
<th>IN-OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>3632</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>3632</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td>10000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>REMOTE</td>
<td>SALES</td>
<td>10000</td>
<td>TEST</td>
<td>R-&gt;S</td>
</tr>
<tr>
<td>* 4</td>
<td>INDEX UNIQUE SCAN</td>
<td>SI_PK</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>REMOTE</td>
<td>PRODUCTS</td>
<td>1</td>
<td>TEST</td>
<td>R-&gt;S</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

- 4 - access("SIT"."ID"="SAL"."SITE")

Remote SQL Information (identified by operation id):

- 3 - SELECT "SALE_DATE","SITE","PRODUCT","QTY","PROFIT" FROM "SALES" "SAL" (accessing 'TEST.LOCALDOMAIN@LOOPBACK' )

## Multi-table distributed join (c)

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Inst</th>
<th>IN-OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>3632</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>3632</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>REMOTE</td>
<td></td>
<td>45</td>
<td>TEST</td>
<td>R-&gt;S</td>
</tr>
<tr>
<td>* 3</td>
<td>INDEX UNIQUE SCAN</td>
<td>SI_PK</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

- 3 - access("SIT"."ID"="SAL"."SITE")

Remote SQL Information (identified by operation id):

Summary

Hinting is not a desirable strategy

Always name your query blocks

There are a few fairly safe (structural) hints

  A few more than in the title, e.g: qb_name(), leading()
  no_merge() may be the most useful

If you have to hint it, document it

If you can hint it, baseline it -- and document it