SQL Tuning tips and tricks
Agenda

- Using the right tools
- Functions, friends or foes?
- Data type dilemmas
- Optimizer hints
- Influencing the execution plan without adding hints
Using the right tools

Expected index range scan but got full table scan?

- Query – How many customers do we have in a specific zipcode

```
SELECT count(cust_first_name)
FROM customers2
WHERE zipcode = :n;
```

- Customers2 table has b-tree index on zipcode
- There is a histogram on the zipcode column due to data skew
Using the right tools

Expected index range scan but got full table scan?

Set bind variable :n to 94065
Using the right tools

Correct plan is achieved when literal value is used

Why is the bind variable version getting the wrong plan?

Using literal value gets the expected plan
Why is there a TO_NUMBER function on the bind variable n after it was defined as a number?

Why is simple equality predicate being applied as filter and no as access?
Bad plan is actually caused by using autotrace

Autotrace is not aware of bind at all

Hence TO_NUMBER on n

No bind peeking takes place
Using the right tools

Cardinality estimate
doesn’t use histogram

Calculated using

\[ \text{ROW\_NUM} \rightarrow 20,000 \]

\[ \text{NDV} \rightarrow 2 \]

```
SQL> var n number
SQL> exec :n :=94065;
PL/SQL procedure successfully completed.

SQL> set autotrace traceonly explain
SQL> SELECT count(cust_email)
2   FROM customers2
3   WHERE zipcode = :n;

Execution Plan
Plan hash value: 2704912892

<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>21</td>
<td>152 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>2</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*2</td>
<td>TABLE ACCESS FULL</td>
<td>CUSTOMERS2</td>
<td>10000</td>
<td>205K</td>
<td>152 (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - filter("ZIPCODE"=TO_NUMBER(:N))
```
Using the right tools

Solution – use DBMS_XPLAN.DISPLAY_CURSOR

```sql
SQL> SELECT count(cust_email) 2 FROM customers2 3 WHERE zipcode = :n;
COUNT(CUST_EMAIL)  
-------------------  
1

SQL> select * from table(dbms_xplan.display_cursor());
```

Execute the statement with the bind then run DBMS_XPLAN command.
### Using the right tools

Using the right tools can help optimize SQL queries and improve performance. For instance, the following SQL query:

```sql
SQL> SELECT count(cust_email)
  2  FROM customers2
  3  WHERE zipcode = :n;

COUNT(CUST_EMAIL)
-----------------
1
```

Displays the count of customers based on the ZIP code. To see the actual bind value under the plan, you can use the `dbms_xplan.display_cursor` procedure with the `format` option:

```sql
SQL> select * from table(dbms_xplan.display_cursor(format=> 'typical +peeked_binds'))
```

This will show additional format options that can reveal the actual bind value used in the query execution plan. For example, the query might look like:

```
PLAN_TABLE_OUTPUT
```

This will help you understand how the query is being executed and potentially optimize it further.
Agenda

- Using the right tools
- Functions, friends or foes?
- Data type dilemmas
- Optimizer hints
- Influencing the execution plan without adding hints
Function Abuse

- Cardinality estimation issues
- May reduce access paths
- Can increase CPU needs (repeated function calls)
- Could lead to partition elimination elimination
Cardinality Estimation Issues

ops$tkyte%ORA11GR2> create table t as
  2   select *
  3 from all_objects
  4 /

Table created.
Cardinality Estimation Issues

```sql
ops$tkyte%ORA11GR2> select count(*)
  2   from t
  3   where created >= to_date( '5-sep-2010', 'dd-mon-yyyy' )
  4     and created < to_date( '6-sep-2010', 'dd-mon-yyyy' )
  5  /

  COUNT(*)
----------
    65925

ops$tkyte%ORA11GR2> select count(*), 0.01 * count(*), 0.01 * 0.01 * count(*)
  2   from t
  3  /

  COUNT(*) 0.01*COUNT(*) 0.01*0.01*COUNT(*)
---------- ---------- ----------
    72926    729.26     7.2926
```
Cardinality Estimation Issues

ops$tkyte%ORA11GR2> exec dbms_stats.gather_table_stats( user, 'T' );
PL/SQL procedure successfully completed.

• Why did I wait till here to gather statistics?
Cardinality Estimation Issues

```sql
ops$tkyte%ORA11GR2> select count(*)
2    from t t2
3   where created >= to_date('5-sep-2010', 'dd-mon-yyyy')
4     and created < to_date('6-sep-2010', 'dd-mon-yyyy')
5   /
6 COUNT(*)
7
----------
8 65925
9
ops$tkyte%ORA11GR2> select * from table(dbms_xplan.display_cursor);
10  ---------------------------------------------------------------------------
11 | Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
12 ---------------------------------------------------------------------------
13 |   0 | SELECT STATEMENT   |      |       |       |   291 (100)|          |
14 |   1 |  SORT AGGREGATE    |      |     1 |     8 |            |          |
15 |*  2 |   TABLE ACCESS FULL| T    |
16 |   511K|   291   (1)| 00:00:04 |
17  ---------------------------------------------------------------------------
18 Predicate Information (identified by operation id):
19  ---------------------------------------------------------------------------
20  2 - filter("CREATED"<TO_DATE(' 2010-09-06 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "CREATED">=TO_DATE(' 2010-09-05 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
21  ---------------------------------------------------------------------------
```
Cardinality Estimation Issues

ops$tkyte%ORA11GR2> select count(*)
   2 from t t1
   3 where trunc(created) = to_date( '5-sep-2010', 'dd-mon-yyyy' )
   4 /
   COUNT(*)
----------
  65925
ops$tkyte%ORA11GR2> select * from table(dbms_xplan.display_cursor);
---------------------------------------------------------------------------
<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></td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>294 (100)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*  2</td>
<td>TABLE ACCESS FULL</td>
<td>T</td>
<td>729</td>
<td>5832</td>
<td>294 (2)</td>
<td>00:00:04</td>
</tr>
</tbody>
</table>
---------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------------------------------
  2 - filter(TRUNC(INTERNAL_FUNCTION("CREATED"))=TO_DATE(' 2010-09-05
  00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
Cardinality Estimation Issues

```
ops$tkyte%ORA11GR2> select count(*)
  2    from t t1
  3   where trunc(created) = to_date('5-sep-2010', 'dd-mon-yyyy')
  4     and substr( owner, 1, 3 ) = 'SYS'
  5 /  
----------
   33535
ops$tkyte%ORA11GR2> select * from table(dbms_xplan.display_cursor);
```

```
<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></td>
<td></td>
<td>292 (100)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>T</td>
<td>7</td>
<td>98</td>
<td>292 (1)</td>
<td>00:00:04</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

```
2 - filter((SUBSTR("OWNER",1,3)='SYS' AND TRUNC(INTERNAL_FUNCTION("CREATED"))=TO_DATE(' 2010-09-05 00:00:00' 'syyyy-mm-dd hh24:mi:ss')))"
```
Cardinality Estimation Issues

ops$tkyte%ORA11GR2> select count(*)
2    from t t1
3   where trunc(.created) = to_date( '5-sep-2010', 'dd-mon-yyyy' )
4     and substr( owner, 1, 3 ) = 'SYS'
5     and mod(object_id,100000) > 1
6     / COUNT(*)
7------------------
8  33535
ops$tkyte%ORA11GR2> select * from table(dbms_xplan.display_cursor);
---------------------------------------------------------------------------
<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></td>
<td></td>
<td>292 (100)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>T</td>
<td>1</td>
<td>19</td>
<td>292 (1)</td>
<td>00:00:04</td>
</tr>
</tbody>
</table>
---------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------------------------------
| 0 | SELECT STATEMENT     | | | | 292 (100) | |
| 1 | SORT AGGREGATE       | | 1 | 19    |            | |
|* 2 | TABLE ACCESS FULL    | T | 1 | 19    | 292 (1)   | 00:00:04 |
---------------------------------------------------------------------------

2 - filter((SUBSTR("OWNER",1,3)='SYS' AND MOD("OBJECT_ID",100000)>1
    AND TRUNC(INTERNAL_FUNCTION("CREATED"))=TO_DATE(' 2010-09-05 00:00
        'syyyy-mm-dd hh24:mi:ss')))
Compile with warnings...

```
SQL> alter session set plsql_warnings='enable:all';

SQL> create or replace procedure p
    2  as
    3  begin
    4    dbms_output.put_line( 'hello world' );
    5  exception
    6      when others
    7        then null;
    8  end;
    9 /

Warning: Procedure created with compilation errors.
c##tkyte%CDB1> show errors
Errors for PROCEDURE P:

LINE/COL  ERROR
--------- -----------------------------------------------
6/6  PLS-06009: procedure "P" OTHERS handler does not end in RAISE or RAISE_APPLICATION_ERROR
```
Increased CPU

```
ops$tkyte%ORA11GR2> create or replace procedure p authid definer
  2  as
  3      l_date  varchar2(30) := '01-jan-2011';
  4      l_start number := dbms_utility.get_cpu_time;
  5  begin
  6      for i in 1 .. 10
  7          loop
  8              for x in ( select owner, object_name
  9                  from big_table.big_table
 10                  where created = l_date )
 11              loop
 12                  null;
 13              end loop;
 14          end loop;
 15      dbms_output.put_line( 'CPU: ' ||
 16                      to_char( dbms_utility.get_cpu_time-l_start ) );
 17  end;
 18 /
SP2-0804: Procedure created with compilation warnings
ops$tkyte%ORA11GR2> exec p
CPU: 132
```
Increased CPU

...  
7      loop
8      for x in ( select owner, object_name 
9                     from big_table.big_table 
10                     where created = l_date )
11      loop
12           null;
13      end loop;
...  

ops$tkyte%ORA11GR2> show errors procedure p  
Errors for PROCEDURE P:

LINE/COL  ERROR
---------  --------------------------------------------------------
10/36      PLW-07204: conversion away from column type may result in sub-optimal query plan
Increased CPU

ops$tkyte%ORA11GR2> create or replace procedure p authid definer
  2  as
  3      l_date date := to_date('01-jan-2011','dd-mon-yyyy');
  4      l_start number := dbms_utility.get_cpu_time;
  5  begin
  6      for i in 1 .. 10
  7      loop
  8         for x in ( select owner, object_name
  9                           from big_table.big_table
  10                      where created = l_date )
  11          loop
  12             null;
  13          end loop;
  14      end loop;
  15  dbms_output.put_line( 'CPU: ' ||
  16                      to_char( dbms_utility.get_cpu_time-l_start ) );
  17  end;
  18  /
Procedure created.
ops$tkyte%ORA11GR2> exec p
CPU: 94  30% less CPU in this case
Reduced Access Paths

ops$tkyte%ORA11GR2> create table t
  2  ( x varchar2(20) constraint t_pk primary key,
  3    y varchar2(30)
  4  );
Table created.

ops$tkyte%ORA11GR2> insert into t
  2  select user_id, username
  3    from all_users;
47 rows created.

ops$tkyte%ORA11GR2> commit;
Commit complete.
Reduced Access Paths

```sql
ops$tkyte%ORA11GR2> create or replace procedure p authid definer
  2   as
  3     l_rec t%rowtype;
  4     l_key number := 5;
  5   begin
  6       select * into l_rec from t where x = l_key;
  7       for x in (select plan_table_output
  8           from TABLE( dbms_xplan.display_cursor()))
  9           loop
 10           dbms_output.put_line( x.plan_table_output );
 11           end loop;
 12   end;
 13 /
SP2-0804: Procedure created with compilation warnings
```
Reduced Access Paths

...  
5    begin
6      select * into l_rec from t where x = l_key;
7      for x in (select plan_table_output ... 

ops$tkyte%ORA11GR2> show errors
Errors for PROCEDURE P:

LINE/COL  ERROR
-----------
6/42    PLW-07204: conversion away from column type may result in sub-optimal query plan
Reduced Access Paths

ops$tkyte%ORA11GR2> exec p
SQL_ID 18796jgha0hwz, child number 0
-------------------------------------
SELECT * FROM T WHERE X = :B1

Plan hash value: 1601196873

---------------------------------------------------------------------
| Id | Operation                | Name | Rows | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------
|  0 | SELECT STATEMENT         |      |       |       |     3 (100)|          |
|*  1 | TABLE ACCESS FULL       | T    |     1 |    29 |     3   (0)| 00:00:01 |
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------------------------
1 - filter(TO_NUMBER("X")=:B1)
Reduced Access Paths

```sql
ops$tkyte%ORA11GR2> create or replace procedure p authid definer
2 as
3   l_rec t%rowtype;
4   l_key varchar2(5) := '5';
5 begin
6   select * into l_rec from t where x = l_key;
7   for x in (select plan_table_output
8       from TABLE( dbms_xplan.display_cursor()))
9   loop
10      dbms_output.put_line( x.plan_table_output );
11   end loop;
12 end;
13 /
Procedure created.

ops$tkyte%ORA11GR2> show errors
No errors.
```
Reduced Access Paths

ops$tkyte%ORA11GR2> exec p

SQL_ID  18796jgha0hwz, child number 1
-------------------------------------
SELECT * FROM T WHERE X = :B1

Plan hash value: 1303508680

<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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>T</td>
<td>1</td>
<td>29</td>
<td>1 (100)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>INDEX UNIQUE SCAN</td>
<td>T_PK</td>
<td>1</td>
<td></td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):
2 - access("X"=:B1)
Partition Elimination Eliminated

ops$tkyte%ORA11GR2> CREATE TABLE t
  2  (  
  3   dt  date,  
  4   x   int,  
  5   y   varchar2(30)  
  6  )  
  7  PARTITION BY RANGE (dt)  
  8  (  
  9      PARTITION part1 VALUES LESS THAN(to_date('31-jan-2011', 'dd-mon-yyyy')),  
10      PARTITION part2 VALUES LESS THAN(to_date('28-feb-2011', 'dd-mon-yyyy'))  
11  )  
12 /  

Table created.
ops$tkyte%ORA11GR2> create or replace procedure p authid definer
  2  as
  3    l_date timestamp := timestamp'2011-01-15 00:00:00.000';
  4    l_count number;
  5  begin
  6      select count(*) into l_count from t where dt = l_date;
  7
  8      for x in (select plan_table_output
  9                  from TABLE(dbms_xplan.display_cursor() ) )
 10      loop
 11        dbms_output.put_line( '.'||x.plan_table_output );
 12      end loop;
 13  end;
 14 /

SP2-0804: Procedure created with compilation warnings
Partition Elimination Eliminated

... 5 begin
6 select count(*) into l_count from t where dt = l_date;
7 ...

SP2-0804: Procedure created with compilation warnings

ops$tkyte%ORA11GR2> show errors
Errors for PROCEDURE P:

LINE/COL ERROR
-----------------
6/47 PLW-07204: conversion away from column type may result in sub-optimal query plan
## Partition Elimination Eliminated

**SQL_ID**: 0t5m83d3m67q7, child number 0

```sql
SELECT COUNT(*) FROM T WHERE DT = :B1
```

Plan hash value: 3225603066

<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>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td></td>
<td>1</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PARTITION RANGE ALL</td>
<td></td>
<td>1</td>
<td>9</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>* 3</td>
<td>TABLE ACCESS FULL</td>
<td>T</td>
<td>1</td>
<td>9</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

3 - filter(INTERNAL_FUNCTION("DT")= :B1)
Partition Elimination Eliminated

ops$tkyte%ORA11GR2> create or replace procedure p authid definer
 as
 l_date date := to_date( '2011-01-15', 'yyyy-mm-dd' );
 l_count number;
 begin
 select count(*) into l_count from t where dt = l_date;
 for x in (select plan_table_output
 from TABLE( dbms_xplan.display_cursor() ) ) loop
 dbms_output.put_line( '.'||x.plan_table_output );
 end loop;
 end;
 /
 Procedure created.
 ops$tkyte%ORA11GR2> show errors
 No errors.
Partition Elimination Eliminated

```
.SQL_ID 0t5m83d3m67q7, child number 1
-------------------------------------
SELECT COUNT(*) FROM T WHERE DT = :B1
-------------------------------------
.Plan hash value: 3660200434

<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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PARTITION RANGE SINGLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>KEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>KEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

3 - filter("DT"=:B1)
Partition Elimination Eliminated

```
ops$tkyte%ORA11GR2> alter session set Plsql_Warnings = 'error:all';

ops$tkyte%ORA11GR2> create or replace procedure p authid definer
  2     as
  3     l_date timestamp := timestamp'2011-01-15 00:00:00.000';
  4     l_count number;
  5     begin
  6     select count(*) into l_count from t where dt = l_date;
  7     for x in (select plan_table_output
  8              from TABLE( dbms_xplan.display_cursor() ) )
  9     loop
 10     dbms_output.put_line( '.'||x.plan_table_output );
 11     end loop;
 12     end;
 13     /
```
Agenda

- Using the right tools
- Functions, friends or foes?
- Data type dilemmas
- Optimizer hints
- Influencing the execution plan without adding hints
Data type dilemmas

Expected index range scan but got fast full index scan

- Query – Simple IAS part of an ETL process

```
INSERT /*+ APPEND gather_plan_statistics */
INTO t1 (row_id, modification_num, operation, last_upd)
SELECT row_id, 1, 'I', last_upd
FROM t2
WHERE t2.last_upd > systimestamp;
```

- T2 has a b-tree index on the last_upd column
Data type dilemmas

Expected index range scan but got fast full index scan

Only 1 non-equality access predicate

So why is our access predicate applied as a filter?

What does the `INTERNAL_FUNCTION` mean?
Data type dilemmas

Expected index range scan but got fast full index scan

- INTERNAL_FUNCTION typically means a data type conversion has occurred
- Predicate is “t2.last_upd > systimestamp”
- What data type is the last_upd column

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW_ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td></td>
<td>CHAR(2000)</td>
</tr>
<tr>
<td>MODIFICATION_NUM</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>OPERATION</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>LAST_UPD</td>
<td>NOT NULL</td>
<td>DATE</td>
</tr>
</tbody>
</table>
Data type dilemmas

Why is the cardinality estimate wrong?

- Presence of the INTERNAL_FUNCTION cause the Optimizer to guess the cardinality estimate
  - Optimizer has no way of knowing how function effects data in LAST_UPD column
  - Without a function based index or extended statistics the Optimizer has to guess
  - Guess is 5% of the rows in the table

- 5% of 823296 is 41,164.8 or 41,165
Data type dilemmas

Solution correct data type mismatch

- Changing the predicate from SYSTIMESTAMP to SYSDATE

```
INSERT /*+ APPEND gather_plan_statistics */ INTO t1
   (row_id, modification_num, operation, last_upd)
SELECT row_id, 1 , 'I', last_upd
FROM t2
WHERE t2.last_upd > sysdate;
```
Data type dilemmas

Solution correct data type mismatch

- Changing the predicate from SYSTIMESTAMP to SYSDATE

Predicate Information (identified by operation id):

\[
2 - \text{access("T2","LAST_UPD">SYSDATE@!)}
\]
Data type dilemmas

Expected to get partition pruning via a join but didn’t

- Query – calculate total amount sold that was returned same day

```sql
SELECT sum(amount_sold)
FROM sh.sales s, sh.sales_returns sr
WHERE s.time_id = sr.time_id
AND sr.time_id = to_date('31-DEC-01', 'dd-mon-yyyy');
```

- Sales table is range partitioned on time_id
- Sales table has 4 years of data in quarterly partitions
Data type dilemmas

Expected to get partition pruning via a join but didn’t

SELECT count(s.amount_sold) FROM sales s, sales_return sr WHERE s.time_id = sr.time_id AND sr.time_id = '31-DEC-01'

Plan hash value: 890024704

Predicate Information (identified by operation id):

2 - access("SR"."TIME_ID"=INTERNAL_FUNCTION("S"."TIME_ID"))
3 - filter("SR"."TIME_ID"=TO_TIMESTAMP('31-DEC-01'))
5 - filter(INTERNAL_FUNCTION("S"."TIME_ID")=TO_TIMESTAMP('31-DEC-01'))

Getting transitive predicate but INTERNAL_FUNCTION on partitioned column prevents pruning
Function needed because the join columns have different data types
Data type dilemmas

Solution – ensure join columns have the same data type

```
SELECT count(s.amount_sold) FROM sales s, sales_return sr WHERE
s.time_id = sr.time_id AND sr.time_id = '31-DEC-01'
```

Plan hash value: 462494559

<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>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td></td>
<td>783 (100)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HASH JOIN</td>
<td></td>
<td>494K</td>
<td>7724K</td>
<td>783 (4)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PART JOIN FILTER CREATE</td>
<td>:BF0000</td>
<td>629</td>
<td>5032</td>
<td>738 (3)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS FULL</td>
<td>SALES_RETURN</td>
<td>629</td>
<td>5032</td>
<td>738 (3)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PARTITION RANGE SINGLE</td>
<td></td>
<td>786</td>
<td>6288</td>
<td>42 (5)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>786</td>
<td>6288</td>
<td>42 (5)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("S","TIME_ID"="SR","TIME_ID")
4 - filter("SR","TIME_ID"='31-DEC-01')
6 - filter("S","TIME_ID"='31-DEC-01')

KEY means dynamic pruning at execution time
AP means And Pruning, caused by bloom filter

Now get transitive predicate without data type conversion hence pruning
Agenda

- Using the right tools
- Functions, friends or foes?
- Data type dilemmas
- Optimizer hints
  - What are Optimizer hints
  - How do you determine which hint is needed
  - Why hints are not obeyed
- Influencing the execution plan without adding hints
Optimizer hints

Expectations

- This material will not instantly make you an Optimizer hint expert!
- Adding hints won’t magically improve every query you encounter

Optimizer hints should only be used with extreme care
What are hints?

Overview

- Hints allow you to influence the Optimizer when it has to choose between several possibilities
- A hint is a directive that will be followed when applicable
- Can influence everything from the Optimizer mode used to each operation in the execution
- Automatically means the Cost Based Optimizer will be used
  - Only exception is the RULE hint but it must be used alone
What are hints?
Example - directions to the mall
What are hints?

Example - directions to the mall but don’t take 6th St

![Map showing directions to Westfield San Francisco Centre](image-url)
What are hints?

Example - directions to the mall if you have insider information
What are hints?
Hints only evaluated when they apply to a decision that has to be made

- Should I walk or drive to the mall?
  - Best plan would be to walk
- Should I go up 4th, 5th, or 6th street?
  - Best plan would be to go up 4th street
- Should I go in the front or the back door of the mall?
  - Best plan would be to go in the back door
- Telling me the cheapest parking is at 5th and Mission garage is irrelevant since I decided to walk
Two different classes of hints

Hint mechanism is not exclusively used by the Optimizer

Two types of hints
- Non-Optimizer hints
- Optimizer hints

Non-Optimizer Hints

Optimizer Hints
Not all hints influence the Optimizer

Overview

- The hint mechanism is not exclusively used by the Optimizer
- Several other functional area use hints too
  - Direct path load can be controlled by APPEND hint
  - Parallel statement queuing can be controlled by STATEMENT_QUEUING hint
  - Data management in buffer cache can be influenced by CACHE hint
  - What SQL statements get monitored by SQL Monitor can be controlled by MONITOR hint
This session focuses on Optimizer hints

Hints can be used to influence call aspects of the Optimizer

Two different classes of hints
Hints influencing query transformations

Overview

- First thing the Optimizer does is try to transform (rewrite) your statement
  - This allows additional join methods and join orders to be used
- Some transformations are always done but some are cost-based
- Hints can be used to influence the transformations the Optimizer does
  - NO_QUERY_TRANSFORMATION
  - MERGE
  - USE_CONCAT
  - REWRITE
  - STAR_TRANSFORMATION
  - UNNEST
Hints can also influence all aspects of a plan

Overview

- Hints to influence cardinality
  - DYNAMIC_SAMPLING
  - CARDINALITY

- Hints to influence join methods
  - USE_NL_WITH_INDEX
  - USE_HASH

- Hints to influence access paths
  - FULL
  - INDEX

- Hints to influence join order
  - LEADING
  - ORDERED

- Most hints have corresponding negative hint preceded by word ‘NO_’
- More information on hints can be found in chapter 3 of SQL Reference Guide
Hints Classifications

Overview

- **Single-table** - hints that are specified on one table or view
  - FULL, INDEX or USE_NL
- **Multi-table** - hint that can be specified on one or more tables or views
  - LEADING or ORDERED
- **Query block** - hints that operate on single query blocks
  - STAR_TRANSFORMATION or UNNEST
- **Statement** – hints that apply to the entire SQL statement
  - ALL_ROWS or OPTIMIZER_FEATURES_ENABLE
How to use Optimizer hints

Overview

- Hints are inserted in a SQL statement in the form of a comment with an additional + sign
- They go immediately after the keyword (SELECT, INSERT, etc)

Note: Hint syntax is correct but it is not a valid hint so it is ignored & treated as comment.
How to use Optimizer hints

Overview

- Hints and comments can be combined
- But best practice is to keep comment and hints in different blocks
  - Comments can be put anywhere in a SQL statement not just after keyword

```
SQL> Select /*+ FULL(s) This is a hint */ a comment */ count(*) From sales s;
```

```
SQL> Select /*+ This_is_a_comment_and_hint FULL(s) */ count(*) From sales s;
```

```
SQL> Select /*+ FULL(S) */ count(*) From sales s /* Comment in separate block */;
```
How to use Optimizer hints

Correctly identifying the object in the hint

- Which one of the following hints will trigger the pk_emp index to be used in this query?

  Select /*+ index(scott.emp pk_emp) */ * From emp e;

  Select /*+ index(emp pk_emp) */ * From emp e;

  Select /*+ index(pk_emp) */ * From emp e;

  None of them
How to use Optimizer hints

Correctly identifying the object in the hint

- If you use a table alias in the query than you must specify the table alias name in the hint

- Otherwise the hint is ignored

```
Select /*+ index(e pk_emp) */ * / * From emp e;
```
How to use Optimizer hints

Hints only apply to the query block in which they appear

The dept table only appears in the sub-query, which is treated as separate query block. Hint has no effect.
How to use Optimizer hints

Hints only apply to the query block in which they appear

The hint on dept now has an effect as it appears in the correct query block, the sub-query

Only exception are statement level hints
How to use Optimizer hints

Query block names

- Oracle automatically names each query block in a SQL statement
  - sel$1, ins$2, upd$3, del$4, cri$5, mrg$6, set$7, misc$8
  - Displayed using ‘+alias’ format parameter in DBMS_XPLAN procedures
- Query block names can be used to specify which block a hint applies to
  - /*+ FULL(@SEL$2 D) */
- The QB_NAME hint can be used to explicitly labels each query block
  - /*+ QB_NAME(my_name_for_block) */
How to use Optimizer hints

Query block names

```sql
SQL> Select /*+ FULL(e) FULL(@MY_SUBQ d) */ e.ename, e.deptno
2  From emp e
3  Where e.deptno = (Select /*+ QB_NAME(MY_SUBQ) */ d.deptno
4       From dept d
5       Where d.loc='CHICAGO');
```
How to use Optimizer hints

How do I know if my hints are used or not?

- Any valid hint will be used
- Can check if a hint is valid in hint section of 10053 trace

ERR indicates if there is an error with hint

USED indicates the hint was used during the evaluation of the part of the plan it pertains to
Doesn’t mean the final plan will reflect it
How to use Optimizer hints

Example showing how hints are used

Un-hinted SQL statement

```sql
SQL> Select c.cust_first_name, c.cust_last_name, sum(s.amount_sold)
2    From customers c, sales s
3    Where c.cust_id=s.cust_id
4    And c.cust_city='Los Angeles'
5    And c.cust_state_province='CA' And c.country_id=52790
6    And s.time_id='09-NOV-00'
7    Group by c.cust_first_name, c.cust_last_name;
```
How to use Optimizer hints

Example showing how hints are used

Default plan is a hash join between sales and customers

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>250 (100)</td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td>1</td>
<td>64</td>
<td>250 (4)</td>
</tr>
<tr>
<td>* 2</td>
<td>HASH JOIN</td>
<td></td>
<td>4</td>
<td>256</td>
<td>249 (4)</td>
</tr>
<tr>
<td>* 3</td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>CUSTOMERS</td>
<td>3</td>
<td>138</td>
<td>226 (3)</td>
</tr>
<tr>
<td>4</td>
<td>PARTITION RANGE SINGLE</td>
<td></td>
<td>535</td>
<td>9630</td>
<td>23 (18)</td>
</tr>
<tr>
<td>* 5</td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>SALES</td>
<td>535</td>
<td>9630</td>
<td>23 (18)</td>
</tr>
</tbody>
</table>

We want the query to use a nested loops join
How to use Optimizer hints

Example showing how hints are used

Hinted SQL statement

```sql
SQL> explain plan for
2  Select /*+ USE_NL(s) */ c.cust_first_name, c.cust_last_name, sum(s.amount_sold)
3  From   customers c, sales s
4  Where  c.cust_id=s.cust_id
5  And    c.cust_city='Los Angeles'
6  And    c.cust_state_province='CA' And c.country_id=52790
7  And    s.time_id='09-NOV-00'
8  Group by c.cust_first_name, c.cust_last_name;
```
How to use Optimizer hints

Example showing how hints are used

Even with hint we get hash join plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td>HASH JOIN</td>
<td>1</td>
<td>64</td>
<td>250 (100)</td>
</tr>
<tr>
<td></td>
<td>HASH JOIN</td>
<td></td>
<td>4</td>
<td>256</td>
<td>249 (4)</td>
</tr>
<tr>
<td></td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>CUSTOMERS</td>
<td>3</td>
<td>138</td>
<td>226 (3)</td>
</tr>
<tr>
<td></td>
<td>PARTITION RANGE SINGLE</td>
<td></td>
<td>535</td>
<td>9630</td>
<td>23 (18)</td>
</tr>
<tr>
<td></td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>SALES</td>
<td>535</td>
<td>9630</td>
<td>23 (18)</td>
</tr>
</tbody>
</table>

Why did it not use the hint?
How to use Optimizer hints

Example showing how hints are used

- Lets look in the 10053 trace file

```
Dumping Hints
============
atom_hint=(@=0x13c6e7e20 err=0 resol=1 used=1 token=924 org=1 lvl=3 txt=USE_NL ("S") )
============ END SQL Statement Dump ==============
```

- Hint is valid and was used
- Why did it not change the plan?
- We only hinted the join method we didn’t hint the join order
- Hint only valid when sales is on right side
- Hint considered when join order was customer, sales but not when it was sales, customer
How to use Optimizer hints

Example showing how hints are used

Hinted SQL statement with both join method and join order hints

```
SSQL> explain plan for
2 Select /*+ ORDERED USE NL(s) */ c.cust_first_name, c.cust_last_name, sum(s.amount_sold)
3 From customers c, sales s
4 Where c.cust_id=s.cust_id
5 And c.cust_city='Los Angeles'
6 And c.cust_state_province='CA' And c.country_id=52790
7 And s.time_id='09-NOV-00'
8 Group by c.cust_first_name, c.cust_last_name;
```
How to use Optimizer hints

Example showing how hints are used

Hinted plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>64</td>
<td>292 (7)</td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td>1</td>
<td>64</td>
<td>292 (7)</td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td>4</td>
<td>256</td>
<td>291 (6)</td>
</tr>
<tr>
<td>* 3</td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>CUSTOMERS</td>
<td>3</td>
<td>138</td>
<td>226 (3)</td>
</tr>
<tr>
<td>4</td>
<td>PARTITION RANGE SINGLE</td>
<td></td>
<td>1</td>
<td>18</td>
<td>22 (19)</td>
</tr>
<tr>
<td>* 5</td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>SALES</td>
<td>1</td>
<td>18</td>
<td>22 (19)</td>
</tr>
</tbody>
</table>
How to use Optimizer hints

Guaranteeing the same plan every time

- Partial hints can’t guarantee the same plan every time
- Only way to guarantee the same plan every time is with a full outline
- A full outline is a complete set of hints for all aspects of a plan
- Full outline for a plan can be displayed using ‘+outline’ option with FORMAT parameter in DBMS_XPLAN.DISPLAY_CURSOR

Select *
From table(DBMS_XPLAN.DISPLAY_CURSOR(format=>’+outline’));
How to use Optimizer hints

Guaranteeing the same plan every time

```sql
SQL> select * from table(dbms_xplan.display_cursor(format=>'TYPICAL +outline'));

PLAN_TABLE_OUTPUT

SQL_ID fbgb04j4qspn8, child number 0

select e,empno, e,ename from emp e where empno <8000

Plan hash value: 169057108

<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></td>
<td></td>
<td>2 (100)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>EMP</td>
<td>14</td>
<td>140</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>*</td>
<td>INDEX RANGE SCAN</td>
<td>PK_EMP</td>
<td>14</td>
<td></td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Outline Data

/*****
BEGIN_OUTLINE_DATA
IGNORE_OPTIM_EMBEDDED_HINTS
OPTIMIZER_FEATURES_ENABLE(‘11.2.0.3’)
DB_VERSION(‘11.2.0.3’)
ALL_ROWS
OUTLINE_LEAF(0,”SEL$1”)
INDEX_RS_ASC(0,”SEL$1” “E”0,”SEL$1” (“EMP”, ”EMPNO”))
END_OUTLINE_DATA
*/
```

Full outline for the plan
How to use Optimizer hints
Guaranteeing the same plan every time

Easier to maintain a full outline using SQL Plan Management
Changing the Optimizer mode

- The following hints control the Optimizer mode
  - ALL_ROWS (default mode)
  - FIRST_ROWS(n)
  - RULE
- FIRST_ROWS(n) choose plan that returns the first n rows most efficiently
  - Use of old FIRST_ROWS hint is not recommended
    - Not fully cost based
- RULE* hint reverts back to Rule Based Optimizer (RBO)

*Not recommended as RBO is de-supported and severely limits plan options
Changing the Optimizer mode

Default Optimizer mode example

- 40% of the rows in the employee table have department_id = 50
- Default plan is a full table scan

```sql
SQL> Select employee_id, last_name, salary
2   From employees
3   Where department_id =50;

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*1</td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>EMPLOYEES</td>
<td>45</td>
<td>1260</td>
</tr>
</tbody>
</table>
```
Changing the Optimizer mode
FRIST_ROWS(n) hint example

Plan changed because the assumption is you are going to stop fetching after first 10 rows.
Changing the Optimizer mode

RULE hint

- RULE hint specifies that the Rule Based Optimizer (RBO) be used

The RULE hint is ignored if

- Other hints are specified in the stmt
- One or more partitioned tables are used
- One or more IOTs are used
- One or more Materialized views exist
- A SAMPLE clauses is specified in a SELECT statement
- A spreadsheet clause is specified
- Parallel execution is used
- Grouping sets are used
- Group outer-join is used
- A create table with a parallel clause
- A left or full outer join (ANSI) is specified
- Flashback cursor (AS OF) is used
- ……….
RULE hint ignored when partitioned table is used

SQL> select /*+ RULE */ count(*) From sales s;
COUNT(*)
---------
918843

RULE hint is ignored because SALES is a partitioned table has to use CBO.
Changing the Optimizer mode

**RULE hint**

- RULE hint prevents bitmap index from being used and triggers full scan

```sql
SQL> Select /*+ RULE */ count(*) From non_partitioned_sales;
```

**RULE hint is used but prevents bitmap index from being used thus triggers a full table scan**

Note

- rule based optimizer used (consider using cbo)
Changing initialization parameter for a query

OPT_PARAM hint

- Allows value for init.ora Optimizer parameters to be changed for a specific query
- Useful way to prevent setting non-default parameter value system-wide
- Only the following Optimizer influencing init.ora parameters can be set:
  - OPTIMIZER_DYNAMIC_SAMPLING
  - OPTIMIZER_INDEX_CACHING
  - OPTIMIZER_INDEX_COST_ADJ
  - OPTIMIZER_USE_PENDING_STATISTICS
  - Optimizer related underscore parameters
  - STAR_TRANSFORMATION_ENABLED
  - PARALLEL_DEGREE_POLICY
  - PARALLEL_DEGREE_LIMIT
Changing initialization parameter for a query

OPT_PARAM hint example

Cardinality under-estimated due to complex expression

Extended statistics would help
Changing initialization parameter for a query

**OPT_PARAM hint example**

```sql
SQL> Select /*+ OPT_PARAM('OPTIMIZER_USE_PENDING_STATISTICS' 'TRUE') */ count(*)
    2   From employees e
    3   Where e.job_id='SA_REP'
    4   And (e.salary*e.commission_pct)*12 > 13000;

          COUNT(*)
         ----
            22

SQL> Select * from table(dbms_xplan.display_cursor());

PLAN_TABLE_OUTPUT
-----------------------------------------
SQL_ID  0b375b2r77x0, child number 1
-----------------------------------------
Select /*+ OPT_PARAM('OPTIMIZER_USE_PENDING_STATISTICS' 'TRUE') */
    2   count(*) From employees e Where e.job_id='SA_REP' And
    3   (e.salary*e.commission_pct)*12 > 13000

Plan hash value: 1756361130

<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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/* 2</td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>EMPLOYEES</td>
<td>22</td>
<td>264</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
```

Extended statistics created & statistics re-gathered as pending statistics OPT_PARAM hint enables pending statistics for only this statement
Changing Optimizer features enable

This parameter gets its own hint

- OPTIMIZER_FEATURES_ENABLE parameter allows you to switch between optimizer versions
- Setting it to previous database version reverts the Optimizer to that version
  - Disables any functionality that was not present in that version
- Easy way to work around unexpected behavior in a new release
- Hint allows you to revert the Optimizer for just a single statement
Changing Optimizer features enable

Example

```sql
SQL> explain plan for
    2  Select object_type, count(*)
    3   From t
    4   Group by object_type;

Explaned.

SQL>
SQL> select * from table(dbms_xplan.display(format=>'+outline'));

PLAN_TABLE_OUTPUT

Plan hash value: 47235625

<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>49</td>
<td>490</td>
<td>212 (8%)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td>49</td>
<td>490</td>
<td>212 (8%)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>T</td>
<td>88766</td>
<td>868Ki</td>
<td>201 (2%)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
```
Changing Optimizer features enable

Example

```sql
SQL> explain plan for
2  Select /*+ optimizer_features_enable('9.2.0') */ object_type, count(*)
3  From t
4  Group by object_type;

Explanation:

Hash GROUP BY introduced in 10g not an option for 9.2 Optimizer so traditional sort based GROUP BY selected
```
Agenda

- Using the right tools
- Functions, friends or foes?
- Data type dilemmas
- Optimizer hints
  - What are Optimizer hints
  - How do you determine which hint is needed
  - Why hints are not obeyed
- Influencing the execution plan without adding hints
Determining which hints are need

Run SQL Tune Advisor

- Ideally you should not have to manually intervene if the Optimizer picks a suboptimal plan
- SQL Tuning Advisor available via Oracle Tuning Pack
  - Monitors high load SQL via AWR
  - Runs night tuning task to find more efficient plan
  - More efficient plans implemented via SQL Profile
  - A SQL Profile is contains auxiliary information that helps mitigates defects in the Optimizer or the inputs too it
  - Can be manually invoked for problematic SQL statements
Agenda

- Using the right tools
- Functions, friends or foes?
- Data type dilemmas
- Optimizer hints
  - What are Optimizer hints
  - How do you determine which hint is needed
  - Why hints are not obeyed
- Influencing the execution plan without adding hints
Why are Optimizer hints not obeyed?

Syntax and Spelling

- Which one of the following hints will trigger the pk_emp index to be used in this query?

Select /*+ ind(e pk_emp) */ * From emp e;
Select /*+ index(e emp_pk) */ * From emp e;
Select /*+ index(e pk_emp) */ * From emp e;
Why are Optimizer hints not obeyed?

Invalid hint

- Specifying an index hint on a table with no indexes

```
SQL> explain plan for
2  Select /*+ index(p) */ p.promo_name, p.promo_cost
3  From  my_promotions p
4  Where p.promo_category = 'TV'
5  And  p.promo_begin_date = '05-OCT-99';

Plan hash value: 1905114768

PL/SQL procedure successfully completed.
```

Invalid hint because no indexes exist on the table.
Why are Optimizer hints not obeyed?

Illegal hint

- Specifying a hash join hint for non-equality join

```sql
SQL> explain plan for
2 Select /*+ USE_HASH(e s) */ e.first_name, e.last_name
3 From employees e, salary_grade s
4 Where e.salary between s.low_sal and s.high_sal;

Explained.

SQL>
```

Illegal hint because a hash join can't be used for a non-equality join predicate.
Why are Optimizer hints not obeyed?

Invalid hint combinations

- Specifying a parallel hint for an index range scan

```
SQL> explain plan for
2   select /*+index(e empno_pk_ind) parallel(e 8) */ empno, ename
3   from emp e
4   where empno<7700;

Explaned.
```

Invalid hint combination because an index range scan can't be parallelized on non-partitioned table.
Why are Optimizer hints not obeyed?

Contradictory hints

- If two hints contradict each other, they will both be ignored.

```
SQL> explain plan for
    2   SELECT /*+ FULL(e) INDEX(e depthno 1k ind) */ e.empno, e.ename
    3   FROM emp e
    4   WHERE e.empno<7300;

Explained.
SQL> select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT

Plan hash value: 460664205

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID BATCHED</td>
<td>EMP</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>EMPNO_PK_IND</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Conflicting hints you can’t do a full table scan and index lookup on same table.
Why are Optimizer hints not obeyed?

Hint becomes invalid due to transformation

- Ordered hint dictates the join order as the order of tables in FROM clause.
Why are Optimizer hints not obeyed?

Hint becomes invalid due to transformation

- Actual join order used

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 1</td>
<td>FILTER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SORT GROUP BY</td>
<td></td>
<td>1</td>
<td>83</td>
</tr>
<tr>
<td>* 3</td>
<td>HASH JOIN</td>
<td>DEPARTMENTS</td>
<td>3265</td>
<td>264K</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>DEPT_LOCATION_IX</td>
<td>21</td>
<td>147</td>
</tr>
<tr>
<td>* 5</td>
<td>INDEX RANGE SCAN</td>
<td></td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>* 6</td>
<td>HASH JOIN</td>
<td></td>
<td>3296</td>
<td>244K</td>
</tr>
<tr>
<td>7</td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>EMPLOYEES</td>
<td>107</td>
<td>749</td>
</tr>
<tr>
<td>* 8</td>
<td>HASH JOIN</td>
<td></td>
<td>107</td>
<td>7383</td>
</tr>
<tr>
<td>9</td>
<td>1 TABLE ACCESS STORAGE FULL</td>
<td>EMPLOYEES</td>
<td>107</td>
<td>3852</td>
</tr>
<tr>
<td>10</td>
<td>2 TABLE ACCESS STORAGE FULL</td>
<td>JOBS</td>
<td>19</td>
<td>627</td>
</tr>
</tbody>
</table>
Why are Optimizer hints not obeyed?

Hint becomes invalid due to transformation

- NO_MERGE hint prevents transformation from taking place

```sql
SQL> SELECT /*+ NO_MERGE(v) ORDERED */ e1.last_name, j.job_title, e1.salary, v.avg_salary
2  FROM employees e1
3  jobs j,
4  (SELECT e2.department_id, avg(e2.salary) avg_salary
5   FROM employees e2, departments d
6   WHERE d.location_id=1700
7   AND e2.department_id = d.department_id
8   GROUP BY e2.department_id) v
9  WHERE e1.job_id = j.job_id
10 AND e1.department_id = v.department_id
11 AND e1.salary > v.avg_salary
12 ORDER BY e1.last_name;
```
Why are Optimizer hints not obeyed?

Hint now valid because transformation prevented

- Actual join order used
Agenda

- Using the right tools
- Functions, friends or foes?
- Data type dilemmas
- Optimizer hints
- Influencing the execution plan without adding hints
If you can hint it, baseline it

Alternative approach to hints

- It's not always possible to add hints to third party applications
- Hints can be extremely difficult to manage over time
- Once added never removed

Solution

- Use SQL Plan Management (SPM)
- Influence the execution plan without adding hints directly to queries
- SPM available in EE, no additional options required
If you can hint it, baseline it

SQL Plan Management

Users → Parse → GB → HJ → Execute → Plan Acceptable

Plan history

Plan baseline

GB
HJ
HJ
HJ
If you can hint it, baseline it

SQL Plan Management

Users → Parse → Plan history → Plan baseline

GB, NL, HJ
If you can hint it, baseline it

SQL Plan Management

Parse

Plan history

Plan baseline

Execute

Plan Acceptable
Influence execution plan without adding hints

Example Overview

- Simple two table join between the SALES and PRODUCTS tables

```
Current Plan

GROUP BY

HASH JOIN

TABLE ACCESS PRODUCTS

TABLE ACCESS SALES

 Desired Plan

GROUP BY

HASH JOIN

INDEX RANGE SCAN

PROD_SUPP_ID_INDX

TABLE ACCESS SALES
```
Influence execution plan without adding hints

Step 1. Execute the non-hinted SQL statement

```sql
SQL> VARIABLE sup_id number
SQL> exec :sup_id := 1;

PL/SQL procedure successfully completed.

SQL>
SQL> SELECT p.prod_name, sum(s.amount_sold) amt
    2   FROM Sales s, Products p
    3   WHERE s.prod_id=p.prod_id
    4   AND p.supplier_id = :sup_id
    5   group by p.prod_name;

<table>
<thead>
<tr>
<th>PROD_NAME</th>
<th>AMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Envoy External 6X CD-ROM</td>
<td>645586.12</td>
</tr>
<tr>
<td>Model SM26273 Black Ink Cartridge</td>
<td>617732.28</td>
</tr>
<tr>
<td>Model K8822S Cordless Phone Battery</td>
<td>582640.54</td>
</tr>
<tr>
<td>Bounce</td>
<td>244595.65</td>
</tr>
</tbody>
</table>
```
Influence execution plan without adding hints
Default plan is uses full table scans followed by a hash join

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td>72</td>
<td>5688</td>
</tr>
<tr>
<td>2</td>
<td>HASH JOIN</td>
<td></td>
<td>72</td>
<td>5688</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>PRODUCTS</td>
<td>72</td>
<td>3816</td>
</tr>
<tr>
<td>4</td>
<td>VIEW</td>
<td>VW_GBC_5</td>
<td>72</td>
<td>1872</td>
</tr>
<tr>
<td>5</td>
<td>HASH GROUP BY</td>
<td></td>
<td>72</td>
<td>648</td>
</tr>
<tr>
<td>6</td>
<td>PARTITION RANGE ALL</td>
<td></td>
<td>918K</td>
<td>8075K</td>
</tr>
<tr>
<td>7</td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>SALES</td>
<td>918K</td>
<td>8075K</td>
</tr>
</tbody>
</table>
Influence execution plan without adding hints

Step 2. Find the SQL_ID for the non-hinted statement in V$SQL

```
SQL> SELECT sql_id, sql_fulltext
2  FROM v$sql
3  WHERE sql_text like '%SELECT p.prod_name%';
```

<table>
<thead>
<tr>
<th>SQL_ID</th>
<th>SQL_FULLTEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bn5p8hp266tah</td>
<td>SELECT p.prod_name, sum(s.amount_sold) amt FROM Sales s, Products p</td>
</tr>
</tbody>
</table>

Influence execution plan without adding hints

Step 3. Create a SQL plan baseline for the non-hinted SQL statement

```sql
SQL> variable cnt number;
SQL>
SQL> execute :cnt := dbms_spn.load_plans_from_cursor_cache(sql_id=>'bn5p8hp266tah');
PL/SQL procedure successfully completed.
SQL>
SQL> column sql_text format a45
SQL> select sql_handle, sql_text, plan_name, enabled from dba_sql_plan_baselines where sql_text like '%SELECT     p.prod_name%';
```

<table>
<thead>
<tr>
<th>SQL_HANDLE</th>
<th>SQL_TEXT</th>
<th>PLAN_NAME</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_10ed3803a09c8fe1</td>
<td>SELECT p.prod_name, sum(s.amount_sold) amt FROM Sales s, Products p WHERE</td>
<td>SQL_PLAN_11v9s0fh9t3z1c47b6be0</td>
<td>YES</td>
</tr>
</tbody>
</table>
Influence execution plan without adding hints

Step 4. Captured Plan is not our desired plan so it should be disabled

```
SQL> exec :cnt :=DBMS_SPM.ALTER_SQL_PLAN_BASELINE(SOQL_HANDLE =>'SQL_10ed3803a09c8fe1', -
>      PLAN_NAME =>'SQL_PLAN_11v9s0fh9t3z1c47b6be0', -
>      ATTRIBUTE_NAME =>'enabled', -
>      ATTRIBUTE_VALUE =>'NO');

PL/SQL procedure successfully completed.

SQL>
SQL> select sql_handle, sql_text, plan_name, enabled from dba_sql_plan_baselines where sql_text like '%SELECT p.prod_name%';

SQL_HANDLE       SQL_TEXT                                                                 PLAN_NAME  ENA
----------------- -----------------------------------------------------------------------------------------------------------------------
SQL_10ed3803a09c8fe1 SELECT p.prod_name, sum(s.amount_sold) asmt FROM Sales s, Products p WHERE                                      SQL_PLAN_11v9s0fh9t3z1c47b6be0 NO
```
### Influence execution plan without adding hints

**Step 5.** Modify the SQL statement to use the hint(s) & execute it

```sql
SQL> SELECT /*+ INDEX(p) */ p.prod_name, sum(s.amount_sold) amt
     2  FROM Sales s, Products p
     3  WHERE s.prod_id=p.prod_id
     4  AND p.supplier_id = :sup_id
     5  group by p.prod_name;
```

<table>
<thead>
<tr>
<th>PROD_NAME</th>
<th>AMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounce</td>
<td>244595.65</td>
</tr>
<tr>
<td>Comic Book Heroes</td>
<td>101214.6</td>
</tr>
<tr>
<td>Envoy External 6X CD-ROM</td>
<td>645586.12</td>
</tr>
<tr>
<td>Finding Fido</td>
<td>78881.08</td>
</tr>
<tr>
<td>Model K88226 Cordless Phone Battery</td>
<td>582640.54</td>
</tr>
</tbody>
</table>
Influence execution plan without adding hints

Step 6. Find SQL_ID & PLAN_HASH_VALUE for hinted SQL stmt

```sql
SQL> SELECT sql_id, plan_hash_value, sql_fulltext
2   FROM v$sql
3   where sql_text like 'SELECT /*+ INDEX(p)%';

<table>
<thead>
<tr>
<th>SQL_ID</th>
<th>PLAN_HASH_VALUE</th>
<th>SQL_FULLTEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>cn29d9b5wp9u7</td>
<td>903671040</td>
<td>SELECT sql_id, plan_hash_value, sql_fulltext</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FROM v$sql</td>
</tr>
<tr>
<td></td>
<td></td>
<td>where sql_text like '</td>
</tr>
<tr>
<td>ac7jyxhg9mj0c</td>
<td>187119048</td>
<td>/*+ INDEX(p) */ p.prod_name, sum(s.amount_sold) amt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FROM Sales s, P</td>
</tr>
</tbody>
</table>
```
Influence execution plan without adding hints

Step 7. Associate hinted plan with original SQL stmt’s SQL HANDLE

```sql
SQL> exec :cnt:=dbms_spm.load_plans_from_cursor_cache

sql_id =>'ac7jyxhg9mj0c',
plan_hash_value => 187119048,
sql_handle =>'SQL_10ed3803a09c8fe1');

PL/SQL procedure successfully completed.
```

Sql_id & plan_hash_value belong to hinted statement

sql_handle is for the non-hinted statement
Influence execution plan without adding hints

Step 8. Confirm SQL stmt has two plans in it’s baseline

<table>
<thead>
<tr>
<th>SQL_HANDLE</th>
<th>SQL_TEXT</th>
<th>PLAN_NAME</th>
<th>ENA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_10ed3803a09c8fe1</td>
<td>SELECT p.prod_name, sum(s.amount_sold) amt FROM Sales s, Products p WHERE</td>
<td>SQL_PLAN_11v9s0fh9t3zn1aa1ba510 YES</td>
<td></td>
</tr>
<tr>
<td>SQL_10ed3803a09c8fe1</td>
<td>SELECT p.prod_name, sum(s.amount_sold) amt FROM Sales s, Products p WHERE</td>
<td>SQL_PLAN_11v9s0fh9t3zn1c47b6be0 NO</td>
<td></td>
</tr>
</tbody>
</table>

Hinted plan only accepted plan for non-hinted SQL stmt
Influence execution plan without adding hints

Step 9. Confirm hinted plan is being used

```
SELECT  p.prod_name, sum(s.amount_sold) amt FROM Sales s, Products p WHERE s.prod_id=p.prod_id AND p.supplier_id = :sup_id group by p.prod_name
```

Plan hash value: 187113948

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>573 (100)</td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td>71</td>
<td>3550</td>
<td>573 (10)</td>
</tr>
<tr>
<td>*</td>
<td>HASH JOIN</td>
<td></td>
<td>72</td>
<td>3600</td>
<td>572 (10)</td>
</tr>
<tr>
<td>3</td>
<td>VIEW</td>
<td>WJ_GBC_5</td>
<td>72</td>
<td>1224</td>
<td>570 (10)</td>
</tr>
<tr>
<td>4</td>
<td>HASH GROUP BY</td>
<td></td>
<td>72</td>
<td>648</td>
<td>570 (10)</td>
</tr>
<tr>
<td>5</td>
<td>PARTITION RANGE ALL</td>
<td></td>
<td>918Ki</td>
<td>8075Ki</td>
<td>530 (3)</td>
</tr>
<tr>
<td>6</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>918Ki</td>
<td>8075Ki</td>
<td>530 (3)</td>
</tr>
<tr>
<td>*</td>
<td>INDEX RANGE SCAN</td>
<td>PROD_SUPP_INDEX</td>
<td>72</td>
<td>2376</td>
<td>1 (0)</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("ITEM_1"="P","PROD_ID")
7 - access("P"."SUPPLIER_ID"=:SUP_ID)

Non-hinted SQL text but it is using the plan hash value for the hinted stmt

Note section also confirms SQL plan baseline used for stmt

- SQL plan baseline SQL_PLAN_11v9s0fh9t3z1aa1ba510 used for this statement
Summary

- Optimizer hints should only be used with extreme caution
- To guarantee the same plan every time must supply a complete outline
- Use _OPTIMIZER_IGNORE_HINTS parameter to test query performance without hints