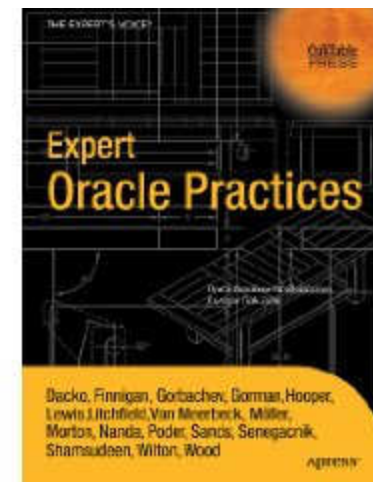

Why does Optimizer hates my SQL!

By
Riyaj Shamsudeen



Who am I?

- 17 years using Oracle products/DBA
- OakTable member
- Certified DBA versions 7.0,7.3,8,8i &9i
- Specializes in RAC, performance tuning, Internals and E-business suite
- Chief DBA with OraInternals
- Co-author of “Expert Oracle Practices” ‘2010
- Email: rshamsud at gmail.com
- Blog : orainternals.wordpress.com



Disclaimer

These slides and materials represent the work and opinions of the author and do not constitute official positions of my current or past employer or any other organization. This material has been peer reviewed, but author assume no responsibility whatsoever for the test cases.

If you corrupt your databases by running my scripts, you are solely responsible for that.

This material should not be reproduced or used without the authors' written permission.

Agenda

■ **Selectivity**

- Cardinality & histograms
- Correlation issues
- Few more reasons behind optimizer decisions.
 - **Bind peeking issues**
 - **Lack of statistics**
 - **Lack of system statistics**
 - **Missing statistics on indexes**
 - **Missing statistics on columns**
 - **Arithmetic comparison issues.**

Selectivity

Selectivity is a measure, with values ranging between 0 and 1, indicating how many elements will be selected from a fixed set of elements.

Selectivity – Example

- Selectivity of predicate

dept =:b1

is $1/\text{NDV} = 1/3 = 0.333$

- Selectivity of predicate

(dept=:b1 or code=:b2)

$= \sim (1/3) + (1/3) - (1/3) * (1/3)$

$= 0.555$

- Selectivity of predicate

(dept=:b1 and code=:b2)

$= \sim (1/3) * (1/3)$

$= 0.111$

Dept	Code	Emp
10	TX	Scott
10	TX	Mary
10	TX	Larry
20	CA	Juan
20	CA	Raj
20	CA	Pele
20	CA	Ronaldinho
30	FL	Ronaldo

NDV : # of distinct values

Selectivity

- Lower the selectivity, better the predicates are.
- Probability concepts will be applied in calculating selectivity of many predicates.

$$p(A \text{ and } B) = p(A) * p(B)$$

$$p(A \text{ or } B) = p(A) + p(B) - p(A) * p(B)$$

Agenda - Part I

- Selectivity
- **Cardinality & histograms**
- Correlation issues
- Few more reasons behind optimizer decisions.
 - Bind peeking issues
 - Lack of statistics
 - Lack of system statistics
 - Missing statistics on indexes
 - Missing statistics on columns
 - Arithmetic comparison issues.

Cardinality

- Cardinality is a measure, an estimate of number of rows expected from a row source.

- In a very simple sense:

$$\text{Cardinality} = \text{Selectivity} * \text{Number of rows}$$

- Cardinality estimates are very essential and need to be accurate for the optimizer to choose optimal execution plan.

Cardinality – No histogram

- Cardinality of predicate

dept =:b1

= (Selectivity * Num_rows)

= (1/3) * 8 = 2.6 rows

= 3 rows

predicate	Estimate	Actual	Change
dept=10	3	3	=
dept=20	3	4	↑
dept=30	3	1	↓

Dept	Emp
10	Scott
10	Mary
10	Larry
20	Karen
20	Jill
20	Pele
20	Ronaldinho
30	Ronaldo

- Under estimation of cardinality is a root cause for many SQL performance issues.

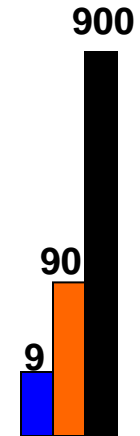
Cardinality & Skew

```
drop table t1;
create table t1 (color_id number, color varchar2(10) );

insert into t1
select l1,
       case
         when l1 <10 then 'blue'
         when l1 between 10 and 99 then 'red'
         when l1 between 100 and 999 then 'black'
       end case
from   (select level l1 from dual connect by level <1000)
/

commit;

Begin
  dbms_stats.gather_table_stats ('cbo2','t1',
  estimate_percent =>100, method_opt =>' for all columns size 1');
End;
/
```



Cardinality : w/o histograms

```
select color, count(*) from t1 group by color order by 2  
COLOR          COUNT(*)
```

```
-----  
blue           9  
red            90  
white          900
```

```
select count(*) from t1 where color='blue';
```

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

```
-----  
| Id  | Operation                | Name | Rows  | Bytes | Cost (%CPU)| Time  |  
-----  
|  0  | SELECT STATEMENT         |      |      |      |    3 (100)|      |  
|  1  |   SORT AGGREGATE         |      |    1  |    6  |           |      |  
|*  2  |    TABLE ACCESS FULL    | T1   |   333 |  1998 |    3 (0) | 00:01 |  
-----
```

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

```
-----  
2 - filter("COLOR"='blue')
```

Histograms

If the column values have skew and if there are predicates accessing that column, then use histograms. But, use them sparingly!

Cardinality : with histograms

Begin

```
dbms_stats.gather_table_stats ('cbo2','t1', estimate_percent =>100,  
method_opt =>' for all columns size 10');
```

end;

/

PL/SQL procedure successfully completed.

```
select count(*) from t1 where color='blue';
```

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

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	6	3 (0)	00:00:01
1	SORT AGGREGATE		1	6		
* 2	TABLE ACCESS FULL	T1	9	54	3 (0)	00:00:01

Predicate Information (identified by operation id):

```
2 - filter("COLOR"='blue')
```

14 rows selected.

Histograms ...

- If there are few distinct values then use that many # of buckets. Else use size auto.
- In many cases, 100% statistics might be needed.
- Collecting histograms in all columns is not recommended and has a side effect of increasing CPU usage. Also, statistics collection can run longer.

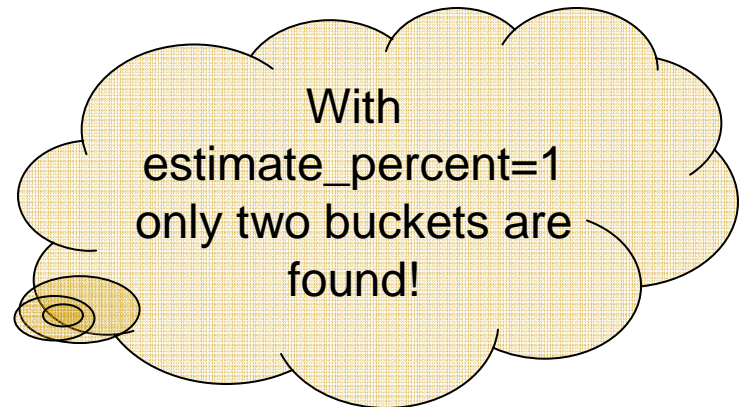
Histograms & estimates

```
begin
  dbms_stats.gather_table_stats ('cbo2','t1', estimate_percent =>1,
    method_opt =>' for all columns size 10');
end;
/
```

```
1 select column_name , endpoint_number from dba_tab_histograms
2* where table_name='T1' and column_name='COLOR'
SQL> /
```

Column Name	ENDPOINT_NUMBER
COLOR	0
COLOR	1

2 rows selected.



Histograms & estimates

```
SQL> explain plan for select count(*) from t1 where color='blue';
```

Explained.

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

Plan hash value: 3724264953

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	6	3 (0)	00:00:01
1	SORT AGGREGATE		1	6		
* 2	TABLE ACCESS FULL	T1	333	1998	3 (0)	00:00:01

Predicate Information (identified by operation id):

```
2 - filter("COLOR"='blue')
```

10g method_opt

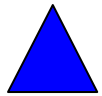
- Method_opt parameter in dbms_stats package defaults to 'for all columns size auto'.
- Dbms_stats can automatically collect histograms for columns with skewed distribution.

Agenda - Part I

- Selectivity
- Cardinality & histograms
- **Correlation issues**
- Few more reasons behind optimizer madness.
 - Bind peeking issues
 - Lack of statistics
 - Lack of system statistics
 - Missing statistics on indexes
 - Missing statistics on columns
 - Arithmetic comparison issues.

Correlation explained

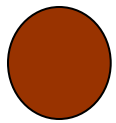
Optimizer assumes no correlation between predicates.



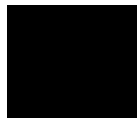
In this example,

All triangles are blue.

All circles are red.



All squares are black.



Predicates shape='CIRCLE' and color='RED'
are correlated.

But optimizer assumes no-correlation.

Correlation ...

```
drop table t1;
create table t1
  (color_id number, color varchar2(10), shape varchar2(10) );

insert into t1
select l1,
       case
         when l1 <10 then 'blue'
         when l1 between 10 and 99 then 'red'
         when l1 between 100 and 999 then 'black'
       end case,
       case
         when l1 <10 then 'triangle'
         when l1 between 10 and 99 then 'circle'
         when l1 between 100 and 999 then 'rectangle'
       end case
from
  (select level l1 from dual connect by level <1000)
/

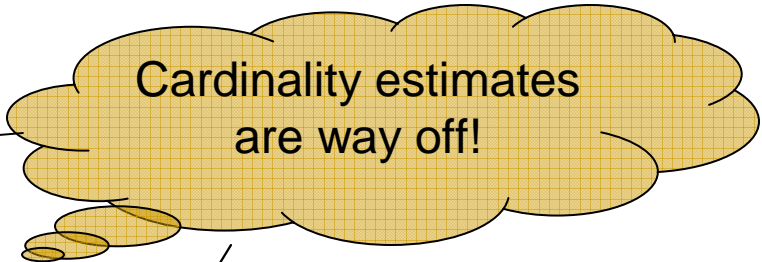
commit;

exec dbms_stats.gather_table_stats ('cbo2','t1', estimate_percent =>100,
  method_opt =>' for all columns size 1');
```

Correlation & cardinality

```
1* select color, shape, count(*) from t1 group by color,shape
SQL> /
```

COLOR	SHAPE	COUNT(*)
blue	triangle	9
red	circle	90
black	rectangle	900



```
explain plan for select count(*) from t1
  where color='blue' and shape='triangle';
```

```
select * from table(dbms_xplan.display);
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	16	3 (0)	0:00:01
1	SORT AGGREGATE		1	16		
* 2	TABLE ACCESS FULL	T1	111	1776	3 (0)	0:00:01

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

```
2 - filter("COLOR"='blue' AND "SHAPE"='triangle')
```

(No)Correlation ..why?

- Selectivity of first single column predicate
color = 'blue' is 1/3.

- Selectivity of next single column predicate
shape='triangle' is 1/3.

- Combined selectivity of both predicates are

$$\text{sel}(p1) * \text{sel}(p2) = (1/3)*(1/3)=1/9 \quad [\text{Probability theory}]$$

- Cardinality estimates, then, becomes

$$999 * (1/9) = 111$$

Optimizer assumes no
Correlation between
Predicates.

Correlation w/ Histograms..

```
alter session set optimizer_dynamic_sampling=0;
```

```
exec dbms_stats.gather_table_stats ('cbo2','t1', estimate_percent =>100,  
method_opt =>' for all columns size 5');
```

```
explain plan for select count(*) from t1  
where color='blue' and shape='triangle';
```

```
select * from table(dbms_xplan.display);
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	16	3 (0)	0:00:01
1	SORT AGGREGATE		1	16		
* 2	TABLE ACCESS FULL	T1	1	16	3 (0)	00:00:01

With histograms, row
Estimates are farther away
from reality

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

```
2 - filter("SHAPE"='triangle' AND "COLOR"='blue')
```

So what do we do?

- Until version 11g, this is a real problem. There is no easy way to fix this. Column statistics might need to be manually adjusted.
- In version 10g, optimizer_dynamic_sampling at level 4 can be used to mitigate this.
- Version 11g provides extended statistics to resolve this correlation issue. Refer my blog entry <http://orainternals.wordpress.com/2008/03/21/correlation-between-column-predicates/> for more information on this topic.

Extended statistics

- Dbms_stats package provides a function to create extended statistics.
- Following code creates an extended statistics on (color,shape) capturing correlation between the columns color and shape.

```
SELECT dbms_stats.create_extended_stats( ownname=>user,  
    tabname => 'T1',extension => '(color, shape )' ) AS c1_c2_correlation  
FROM dual;
```

```
C1_C2_CORRELATION
```

```
-----  
SYS_STUAOJW6_2K$IUXLR#$DK235BV
```

Virtual column

- Creating extended statistics adds a hidden, virtual column to that table.

```
select owner, table_name, column_name, hidden_column, virtual_column
from dba_tab_cols where table_name='T1' and owner='CBO2'
order by column_id;
```

OWNER	TABLE	COLUMN_NAME	HID	VIR
...				
CBO2	T1	SHAPE	NO	NO
CBO2	T1	SYS_STUA0JW6_2K\$IUXLR#\$DK235BV	YES	YES

- Following code collects statistics on virtual column with histogram.

```
begin
  dbms_stats.gather_Table_stats( user, 'T1',
    estimate_percent => 100,
    method_opt => 'for all columns size 254');
end;
/
```

Better estimates

- For other combinations also estimate is very close.

```
SQL> explain plan for select count(*) from t1 where color='black' and  
      shape='rectangle';
```

Explained.

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

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	16	4 (0)	00:00:01
1	SORT AGGREGATE		1	16		
* 2	TABLE ACCESS FULL	T1	900	14400	4 (0)	00:00:01

Predicate Information (identified by operation id):

```
2 - filter("COLOR"='black' AND "SHAPE"='rectangle')
```

Better estimates

...2

- Creating extended statistics adds a hidden, virtual column to that table.

```
explain plan for select count(*) from t1 where  
color='blue' and shape='triangle';  
select * from table(dbms_xplan.display);
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	16	4 (0)	00:00:01
1	SORT AGGREGATE		1	16		
* 2	TABLE ACCESS FULL	T1	9	144	4 (0)	00:00:01

Predicate Information (identified by operation id):

```
2 - filter("SHAPE"='triangle' AND "COLOR"='blue')
```

Non-existing values

- For non-existing combinations also, estimates are much better.

```
SQL> explain plan for select count(*) from t1 where color='blue' and  
      shape='rectangle';
```

Explained.

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

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	16	4 (0)	00:00:01
1	SORT AGGREGATE		1	16		
* 2	TABLE ACCESS FULL	T1	5	80	4 (0)	00:00:01

Predicate Information (identified by operation id):

```
2 - filter("COLOR"='blue' AND "SHAPE"='rectangle')
```

Agenda - Part I

- Selectivity
- Cardinality & histograms
- Correlation issues
- **Few issues behind optimizer decisions.**
 - Bind peeking issues
 - Lack of statistics
 - Lack of system statistics
 - Missing statistics on indexes
 - Missing statistics on columns
 - Arithmetic comparison issues.

#1. Bind peeking...

- Optimizer peeks in to bind variables at time of hard parsing.
- Optimizer uses values from the bind variable to optimize the statement. This can affect range predicates and statements with like clause.
- If there are histograms on that column, then execution plan can be different depending upon the time of hard parse (Example follows).
- Parameter `_optim_peek_user_binds` controls this behavior and defaults to true.

Bind peeking example

```
variable v_color varchar2(12)
exec :v_color := 'blue';
```

```
select count(*) from t1 where color=:v_color;
select * from table
(dbms_xplan.display_cursor);
```

SQL_ID 209g46tpf1gnq, child number 0

select count(*) from t1 where color=:v_color

Plan hash value: 2432955788

Id	Operation	Name	Rows
0	SELECT STATEMENT		
1	SORT AGGREGATE		1
* 2	INDEX RANGE SCAN	T1_I1	9

Predicate Information

2 - access("COLOR"=:V_COLOR)

```
exec :v_color := 'black';
```

```
select count(*) from t1 where color=:v_color;
select * from table
(dbms_xplan.display_cursor);
```

SQL_ID 209g46tpf1gnq, child number 0

select count(*) from t1 where color=:v_color

Plan hash value: 2432955788

Id	Operation	Name	Rows
0	SELECT STATEMENT		
1	SORT AGGREGATE		1
* 2	INDEX RANGE SCAN	T1_I1	9

Predicate Information

2 - access("COLOR"=:V_COLOR)

Bind peeking issues

Adaptive cursor sharing feature in 11g mitigates this issue (somewhat..)

```
exec sys.dbms_shared_pool.purge('234CD2C8,1793113750','C',1);
```

```
SQL> exec :v_color :='black';
```

PL/SQL procedure successfully completed.

```
SQL> select * from table
      (dbms_xplan.display_cursor);
SQL_ID  209g46tpf1gnq, child number 0
-----
select count(*) from t1 where color=:v_color
```

```
-----
| Id  | Operation                               | Name  | Rows |
-----
|  0  | SELECT STATEMENT                         |       |      |
|  1  | SORT AGGREGATE                           |       |    1 |
|*  2  | INDEX FAST FULL SCAN                     | T1_I1 |  900 |
-----
```

Predicate Information :

```
-----
2 - filter("COLOR"=:V_COLOR)
```

```
SQL> exec :v_color :='blue';
```

PL/SQL procedure successfully completed.

```
select * from table
      (dbms_xplan.display_cursor);
SQL_ID  209g46tpf1gnq, child number 0
-----
```

```
-----
| Id  | Operation                               | Name  | Rows |
-----
|  0  | SELECT STATEMENT                         |       |      |
|  1  | SORT AGGREGATE                           |       |    1 |
|*  2  | INDEX FAST FULL SCAN                     | T1_I1 |  900 |
-----
```

Predicate Information :

```
-----
2 - filter("COLOR"=:V_COLOR)
```

Adaptive Cursor Sharing

Initially, execution plan for both bind Variables are the same.

```
variable v_color varchar2(10)
```

```
exec :v_color := 'blue';
```

PL/SQL procedure successfully completed.

```
SQL> select distinct shape from t1 where color=:v_color;
```

```
SHAPE
-----
triangle
```

```
select * from table(dbms_xplan.display_cursor);
SQL_ID d5d8tfnunpcca, child number 0
```

```
-----
select distinct shape from t1 where color=:v_color
```

```
Plan hash value: 3899492605
```

```
variable v_color varchar2(10)
```

```
exec :v_color := 'black';
```

PL/SQL procedure successfully completed.

```
SQL> select distinct shape from t1 where color=:v_color;
```

```
SHAPE
-----
rectangle
```

```
select * from table(dbms_xplan.display_cursor);
SQL_ID d5d8tfnunpcca, child number 0
```

```
-----
select distinct shape from t1 where color=:v_color
```

```
Plan hash value: 3899492605
```

Id	Operation	Name	Rows	Id	Operation	Name	Rows
0	SELECT STATEMENT			0	SELECT STATEMENT		
1	HASH UNIQUE		3	1	HASH UNIQUE		3
2	TABLE ACCESS BY INDEX ROWID	T1	9	2	TABLE ACCESS BY INDEX ROWID	T1	9
* 3	INDEX RANGE SCAN	T1_COLOR	9	* 3	INDEX RANGE SCAN	T1_COLOR	9

Predicate Information (identified by operation id):

```
3 - access("COLOR"=:V_COLOR)
```

Predicate Information (identified by operation id):

```
3 - access("COLOR"=:V_COLOR)
```

Adaptive Cursor Sharing

In few seconds and few executions later, SQL evolved in to a bind-aware statement.

```
variable v_color varchar2(10)
```

```
exec :v_color := 'blue';
```

PL/SQL procedure successfully completed.

```
SQL> select distinct shape from t1 where color=:v_color;
```

SHAPE

triangle

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

```
SQL_ID d5d8tfnunpcca, child number 1
```

select distinct shape from t1 where color=:v_color

Plan hash value: 3899492605

Id	Operation	Name	Rows
0	SELECT STATEMENT		
1	HASH UNIQUE		3
2	TABLE ACCESS BY INDEX ROWID	T1	9
* 3	INDEX RANGE SCAN	T1_COLOR	9

Predicate Information (identified by operation id):

```
3 - access("COLOR"=:V_COLOR)
```

```
variable v_color varchar2(10)
```

```
exec :v_color := 'black';
```

PL/SQL procedure successfully completed.

```
SQL> select distinct shape from t1 where color=:v_color;
```

SHAPE

rectangle

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

```
SQL_ID d5d8tfnunpcca, child number 2
```

select distinct shape from t1 where color=:v_color

Plan hash value: 2134347679

Id	Operation	Name	Rows
0	SELECT STATEMENT		
1	HASH UNIQUE		3
* 2	TABLE ACCESS FULL	T1	900

Predicate Information (identified by operation id):

```
2 - filter("COLOR"=:V_COLOR)
```

#2. Lack of object statistics

- Without object statistics, optimizer assumes few basic statistics.

- # of distinct keys 100 and Average Row length 100

- From table length, calculates # of rows in the table.

$$\# \text{ of rows} = \sim (\# \text{ of blocks}) * (\text{block_size}) / 100$$

$$= \sim 5 * 8192 / 100$$

$$= 409 \text{ rows (for 5 blocks table)}$$

- So, single column predicate cardinality is:

$$409 / 100 = \sim 4 \text{ rows.}$$

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT				1	
1	SORT AGGREGATE		1	7		
2	INDEX RANGE SCAN	T1_I1	4	28	1	00:00:01

Predicate Information:

2 - access("COLOR"='black')

In 10g..

- This problem is partly resolved by Version 10g defaults. In version 10g, optimizer_dynamic_sampling is set to 2.
- If there are no statistics on tables, Optimizer will dynamically sample rows and calculate statistics.
- This may be a viable option if # of rows in the table is volatile, such as interface tables.
- Cardinality is accurate with dynamic sampling.

```
-----  
Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |  
-----  
0 | SELECT STATEMENT | | 1 | 7 | 3 (0) | 00:00:01 |  
1 | SORT AGGREGATE | | 1 | 7 | | | |  
* 2 | TABLE ACCESS FULL | T1 | 900 | 6300 | 3 (0) | 00:00:01 |  
-----
```

predicate information (identified by operation id):

```
-----  
2 - filter("COLOR"='black')
```

#3. System statistics

- ❑ System statistics provide optimizer statistics about performance of single block and multi block reads. By default, there are only noworkload system statistics.

```
1* select * from aux_stats$  
SQL> /
```

SNAME	PNAME	PVAL1	PVAL2
SYSSTATS_INFO	STATUS		COMPLETED
SYSSTATS_INFO	DSTART		04-05-2009 18:44
SYSSTATS_INFO	DSTOP		04-05-2009 18:44
SYSSTATS_INFO	FLAGS		0
SYSSTATS_MAIN	CPUSPEEDNW	1513.096	
SYSSTATS_MAIN	IOSEKTIM		10
SYSSTATS_MAIN	TOTERSPEED		4096
SYSSTATS_MAIN	SREADTIM		
SYSSTATS_MAIN	MREADTIM		
SYSSTATS_MAIN	CPUSPEED		
SYSSTATS_MAIN	MBRC		
SYSSTATS_MAIN	MAXTHR		
SYSSTATS_MAIN	SLAVETHR		

```
13 rows selected.
```

Effect of system statistics

```
explain plan for select v1 from t1 where n3 between 10 and 30;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1362	138K	19 (0)	00:00:01
* 1	TABLE ACCESS FULL	T1	1362	138K	19 (0)	00:00:01

```
begin
  dbms_stats.set_system_stats ('cpuspeed',588);
  dbms_stats.set_system_stats ('sreadtim',5);
  dbms_stats.set_system_stats ('mreadtim',30);
  dbms_stats.set_system_stats ('mbrcc',12);
end;
/
```

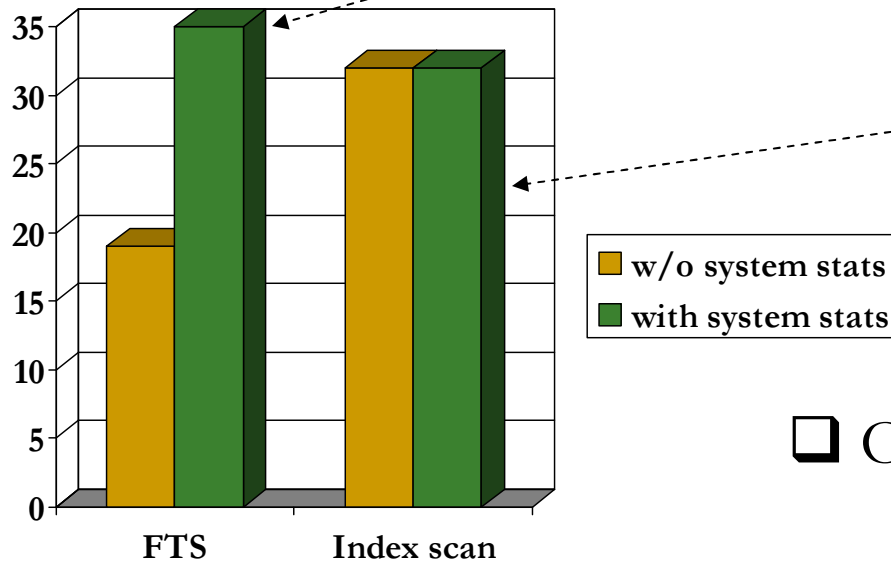
Plan with noworkload system statistics.

Plan with system statistics.

```
explain plan for select v1 from t1 where n3 between 10 and 30;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1362	138K	32 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	T1	1362	138K	32 (0)	00:00:01
* 2	INDEX RANGE SCAN	T1_N3	1362		4 (0)	00:00:01

Cost difference



❑ Cost of Full Table scan went up from 19 to 35 after adding system statistics.

❑ Cost of Index range scan remained the same.

❑ Optimizer chose index range scan.

❑ Adjusting Optimizer parameters `optimizer_index_caching` and `optimizer_index_cost_adj` are another method, but system statistics are preferred.

aux_stats\$

□ Aux_stats\$ shows values for sreadtim, mreadtime etc..

```
SQL> select * from aux_stats$;
```

SNAME	PNAME	PVAL1	PVAL2
---	---	---	---
SYSSTATS_INFO	STATUS		COMPLETED
SYSSTATS_INFO	DSTART		04-05-2009 19:24
SYSSTATS_INFO	DSTOP		04-05-2009 19:24
SYSSTATS_INFO	FLAGS		1
SYSSTATS_MAIN	CPUSPEEDNW	1513.096	
SYSSTATS_MAIN	IOSEEKTIM		10
SYSSTATS_MAIN	IOTFRSPEED	4096	
SYSSTATS_MAIN	SREADTIM	5	
SYSSTATS_MAIN	MREADTIM	30	
SYSSTATS_MAIN	CPUSPEED	588	
SYSSTATS_MAIN	MBRC	12	
SYSSTATS_MAIN	MAXTHR		
SYSSTATS_MAIN	SLAVETHR		

13 rows selected.

Agenda - Part II

■ Few more issues continued...

- Missing statistics on indexes
- Missing statistics on columns
- Column type mismatch
- Index on non-selective columns
- Index efficiency
- Null and index
- Arithmetic comparison issues.

■ Hints

#4: Index & statistics

❑ It is a common mistake to add index to an existing table, but not collect statistics on that index.

❑ This problem is prevalent in 9i.

```
drop table t1;
```

```
create table t1 (n1 number, n2 number,  
                n3 number, v1 varchar2(100));
```

```
insert into t1  
select l1, mod(l1, 110), round(l1,110),  
       lpad(l1, 100, 'x')
```

```
from  
  (select level l1 from dual  
   connect by level <= 100000);
```

```
commit;
```

Index on column n2
will have high
clustering factor!

Index with statistics

Index access through t1_n2 is costly and optimizer chose FTS correctly.

```
create index t1_n2 on t1(n2);

Begin
  dbms_stats.gather_table_stats(
    user, 't1' , estimate_percent =>100,
    cascade =>true);
End;
/
```

```
Access Path: index (AllEqRange)
Index: T1_N2
resc_io: 912.00 resc_cpu: 6868703
ix_sel: 0.009091
ix_sel_with_filters: 0.009091
Cost: 914.34 Resp: 914.34 Degree: 1
```

```
explain plan for select v1 from t1 where n2=10;
```

```
select * from table(dbms_xplan.display);
```

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT				892	
1	TABLE ACCESS FULL	T1	909	93K	892	00:00:05

```
Predicate Information:
```

```
1 - filter("N2"=10)
```

Index with out statistics

Index chosen even though execution of this plan is costly.

```
Begin
  dbms_stats.delete_index_stats
    (user, 't1_n2');
End;
/
```

```
Access Path: index (AllEqRange)
Index: T1_N2
resc_io: 9.00 resc_cpu: 74373
ix_sel: 0.009091
ix_sel_with_filters: 0.009091
Cost: 9.03 Resp: 9.03 Degree: 1
```

```
explain plan for select v1 from t1 where n2=10;
```

```
select * from table(dbms_xplan.display);
```

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT				9	
1	TABLE ACCESS BY INDEX ROWID	T1	909	93K	9	00:00:01
2	INDEX RANGE SCAN	T1_N2	909		1	00:00:01

```
Predicate Information:
```

```
2 - access("N2"=10)
```

10g...

- ❑ Again, 10g fixes this issue.

- ❑ In create index statement “compute statistics” is the default and so index will be created with statistics.

- ❑ Parameter `_optimizer_compute_index_stats` controls this behavior and statistics collection forced during create index or rebuild of an index.

#5. Columns with out statistics

□ Another common issue is to add column to a table, but forget to collect statistics on that new column.

```
alter table t_large3 add (n1_new number);
```

```
update t_large3 set n1_new=n1;  
1000000 rows updated.
```

```
commit;
```

```
explain plan for select * from t_large3  
where n1_new =:b1;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		9996	4978K	38988 (1)	00:03:15
* 1	TABLE ACCESS FULL	T_LARGE3	9996	4978K	38988 (1)	00:03:15

```
dbms_stats.gather_table_stats(user,'t_large3',estimate_percent=>100,cascade=>true);
```

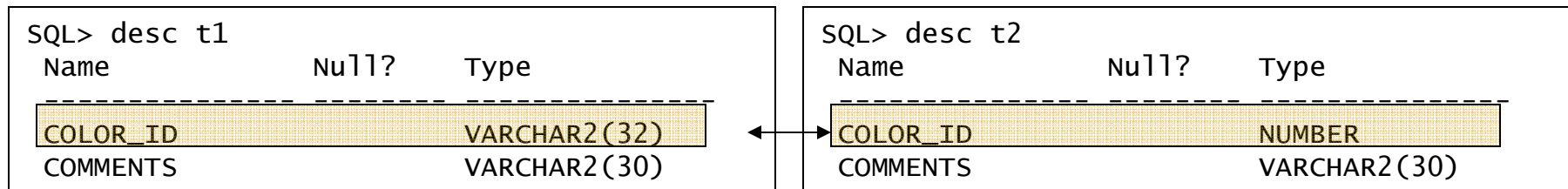
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	515	38988 (1)	00:03:15
* 1	TABLE ACCESS FULL	T_LARGE3	1	515	38988 (1)	00:03:15

Without column level statistics, row estimates far off..

#6. Column type mismatch

- It is recommended to have join columns of same data type.
- If there is a mismatch in data type, then `to_number` or `to_char` functions need to be applied. Applying a function over a column disallows use of index on that column.

SQL> desc t1			SQL> desc t2		
Name	Null?	Type	Name	Null?	Type
COLOR_ID		VARCHAR2(32)	COLOR_ID		NUMBER
COMMENTS		VARCHAR2(30)	COMMENTS		VARCHAR2(30)



Column type mismatch...

```
select t1.* from t1, t2 where t1.color_id = t2.color_id and t1.color_id=10;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	13	5 (0)	00:00:01
1	NESTED LOOPS					
2	NESTED LOOPS		1	13	5 (0)	00:00:01
* 3	TABLE ACCESS FULL	T2	1	4	3 (0)	00:00:01
* 4	INDEX RANGE SCAN	T1_COLOR_ID	1		1 (0)	00:00:01
5	TABLE ACCESS BY INDEX ROWID	T1	1	9	2 (0)	00:00:01

Query Block Name / Object Alias (identified by operation id):

```
1 - SEL$1  
3 - SEL$1 / T2@SEL$1  
4 - SEL$1 / T1@SEL$1  
5 - SEL$1 / T1@SEL$1
```

Index not chosen..

Predicate Information (identified by operation id):

```
3 - filter(TO_NUMBER("T2"."COLOR_ID")=10)  
4 - access("T1"."COLOR_ID"=I0)  
  
filter("T1"."COLOR_ID"=TO_NUMBER("T2"."COLOR_ID"))
```

Due to data type mismatch, to_number conversion function applied over the join column.

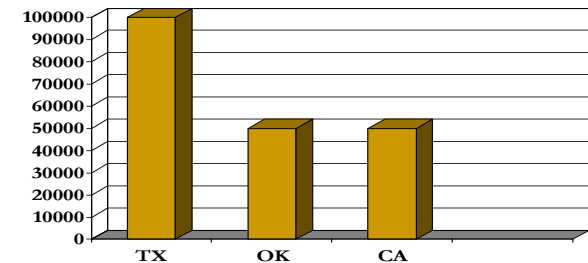
#7. Index(btree) on non-selective columns

- Indexed columns should have good selectivity, for btree indices. If selectivity is closer to one, then cost of access through index can be more than cost of Full table scan.
- In the example below, selectivity of column state_code is poor.

```
create table sel_t3  
  ( state_code varchar2(2), first_name varchar2(30), last_name varchar2(30) );
```

```
insert into sel_t3  
select case when l1 between 0 and 100000 then 'TX'  
           when l1 between 100001 and 150000 then 'OK'  
           when l1 between 150001 and 200000 then 'CA'  
           end ,  
       rpad ( 'Jill', 30,'x'),rpad ( 'Somebody', 30,'x') from  
       (select level l1 from dual connect by level <=100000)  
/  
commit;
```

```
create index sel_t3_i1 on sel_t3 ( state_code);
```



Index(btree) on non-selective columns...

```
explain plan for
select first_name||','||last_name from sel_t3
where state_code='TX';
```

Id	Operation	Name	Rows
0	SELECT STATEMENT		99915
* 1	TABLE ACCESS FULL	SEL_T3	99915

Predicate Information (identified by operation id):

1 - filter("STATE_CODE"='TX')

```
Index: SEL_T3_I1
resc_io: 1241.00 resc_cpu: 48209447
ix_sel: 0.504751
```

ix_sel_with_filters: 0.504751

Cost: 1243.18 Resp: 1243.18 Degree: 1

- Index selectivity with filters is 0.5, meaning over half the rows in the table qualify for this predicate condition.

Access with non-selective columns

Set lines 120 pages 0

```
Select * from table(dbms_xplan.display);
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		1	214	9 (23)
1	SORT ORDER BY		1	214	9 (23)
* 2	TABLE ACCESS BY INDEX ROWID	ORDER_LINES	1	155	4 (25)
3	NESTED LOOPS		1	214	8 (13)
* 4	TABLE ACCESS BY INDEX ROWID	ORDER_HEADERS	1	59	5 (20)
* 5	INDEX RANGE SCAN	ORDER_HEADERS_N2	20		4 (25)
* 6	INDEX RANGE SCAN	ORDER_STAGING_LINES_N1	1		3 (34)

Predicate Information (identified by operation id):

- 2 - filter("OL"."ORG_NUMBER"=:Z AND "OL"."ORG_LINE_NUMBER"=:TO_NUMBER(:Z))
- 4 - filter("OH"."ORG_NUMBER"=:Z AND "OH"."PO_ID"=:TO_NUMBER(:Z))
- 5 - access("OH"."HEADER_STATUS"='PROCESSED')
- 6 - access("OH"."SRC_HEADER_ID"="OL"."SRC_HEADER_ID")

With selective index

```
drop index ORDER_HEADERS_N2;;
create index ORDER_HEADERS_N2 on ORDER_HEADERS
(header_status, org_number, po_id) compress 1 compute statistics;
```

Explain plan for SELECT
Set lines 120 pages 0

```
Select * from table(dbms_xplan.display);
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		1	206	9 (23)
1	SORT ORDER BY		1	206	9 (23)
* 2	TABLE ACCESS BY INDEX ROWID	ORDER_LINES	1	155	4 (25)
3	NESTED LOOPS		1	206	8 (13)
4	TABLE ACCESS BY INDEX ROWID	ORDER_HEADERS	1	51	5 (20)
* 5	INDEX RANGE SCAN	ORDER_HEADERS_N2	1		4 (25)
* 6	INDEX RANGE SCAN	ORDER_STAGING_LINES_N1	1		3 (34)

Predicate Information (identified by operation id):

- 2 - filter("OL"."ORG_NUMBER"=:Z AND "OL"."ORG_LINE_NUMBER"=TO_NUMBER(:Z))
- 5 - access("RSH"."HEADER_STATUS"='PROCESSED' AND "RSH"."ORG_NUMBER"=:Z AND "RSH"."PO_ID"=TO_NUMBER(:Z))
- 6 - access("RSH"."SRC_HEADER_ID"="RSL"."SRC_HEADER_ID")

#8. Index efficiency?

Why would the optimizer not choose t1_n2 index?

```
DESC T1
...
N2      NUMBER
N3      NUMBER
...

select count(distinct(n2)) n2_count,
       count(distinct(n3)) n3_count
from t1;

  N2_COUNT  N3_COUNT
-----
         62         63
```

1 row selected.

```
create index t1_n2 on t1(n2);
create index t1_n3 on t1(n3);

Begin
  dbms_stats.gather_table_stats
    ( user, 't1' ,
      estimate_percent =>100,
      cascade =>true);
End;
/
```

```
explain plan for
select n1 from t1 where n2=:b1;
```

Id	Operation	Name	Rows	Cost (%CPU)
0	SELECT STATEMENT		62	19 (0)
* 1	TABLE ACCESS FULL	T1	62	19 (0)

Predicate Information (identified by operation id):

1 - filter("N2"=TO_NUMBER(:B1))

```
explain plan for
select n1 from t1 where n3=:b1;
```

Id	Operation	Name	Rows	Cost (%CPU)
0	SELECT STATEMENT		61	3 (0)
1	TABLE ACCESS BY INDEX ROWID	T1	61	3 (0)
* 2	INDEX RANGE SCAN	T1_N3	61	1 (0)

Predicate Information (identified by operation id):

2 - access("N3"=TO_NUMBER(:B1))

Index properties..

```
drop table t1;
create table t1
(n1 number,
 n2 number,
 n3 number,
 v1 varchar2(100) );

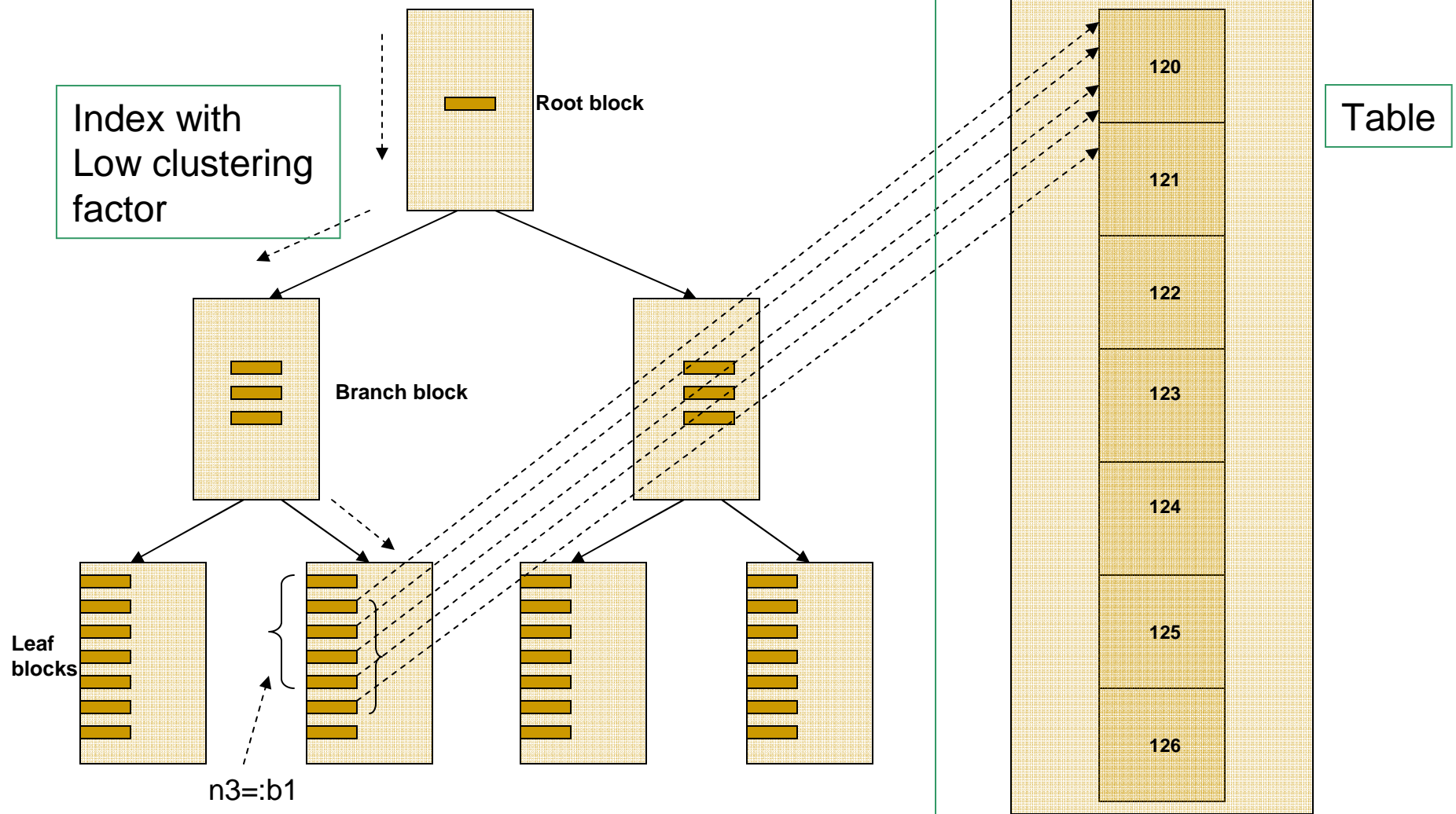
insert into t1
select l1,
       mod(l1, 62) n2,
       round (l1/62) n3,
       lpad(l1, 100,'x') v1
from
  (select level l1 from dual
   connect by level
   <=3844);

commit;
```

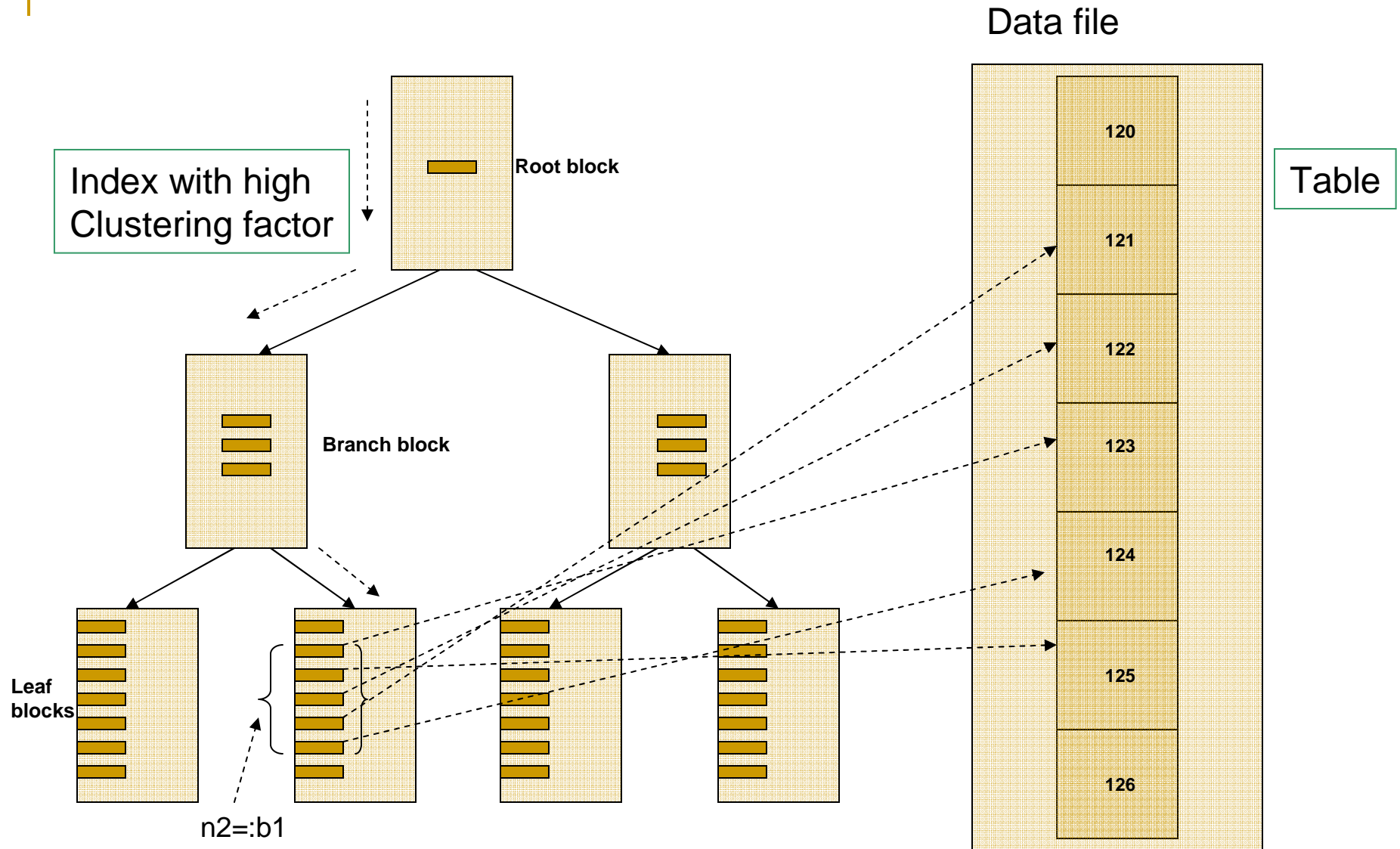
Property	T1_N2	T1_N3
Unique	No	No
Level	1	1
Leaf blocks	8	8
Distinct keys	62	63
# of rows	3844	3844
Avg. leaf block/key	1	1
Avg. datablock/key	61	1
Clustering factor	3843	78



Index efficiency



Index efficiency



Index efficiency

- Optimizer did not choose index on n2 since clustering factor of that index is very high.
- Accessing table through the index will be costlier due to visits to table blocks can increase I/O.
- Avoiding table blocks by adding more columns to that index might be of help.

#9. Cardinality and null

Explain plan for select count(*) from t1 where `n1 is not null;`

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	2	3612 (1)	00:00:19
1	SORT AGGREGATE		1	2		
* 2	TABLE ACCESS FULL	T1	2160	4320	3612 (1)	00:00:19

Predicate Information (identified by operation id):

2 - filter("N1" IS NOT NULL)

How did the optimizer came up with an estimate of 2160?

Table Name	Number of Rows	Blocks	Average Row Len	Global Stats	User Stats	Sample Size	Date
T1	98,750	7,174	502	YES	NO	9,875	04-14-2009

Column Name	Column Details	Distinct Values	Density	Number Buckets	Number Nulls	Sample Size	Date
N1	NUMBER(22)	2,160	0	1	96,590	216	04-14-2009
V1	VARCHAR2(512)	98,750	0	1	0	9,875	04-14-2009

Cardinality and null

For tables with non-null rows at only recent blocks, this estimate could be off by a factor, leading to poor plan.

Sample size of 10% was used.

Optimizer found 216 rows with non-null values for n1

Scaling 216 by 10 yields 2160 rows. So, estimate for 'n1 is not null' is 2160.

Table Name	Number of Rows	Blocks	Average Row Len	Global Stats	User Stats	Sample Size	Sample Date MM-DD-YYYY
T1	98,750	7,174	502	YES	NO	9,875	04-14-2009

Column Name	Column Details	Distinct Values	Density	Number Buckets	Number Nulls	Sample Size	Sample Date MM-DD-YYYY
N1	NUMBER(22)	2,160	0	1	96,590	216	04-14-2009
V1	VARCHAR2(512)	98,750	0	1	0	9,875	04-14-2009

Cardinality and null

- In few cases, sample percent might not be good enough to get good estimate of null values.
- Also, null values are not stored in single column btree indices. So, following SQL might not use indices:
select * from t1 where n1 is null;
select * from t1 where (n1=:b1 or n1 is null);
[Work around: create multi-column indices: (n1, 0)]
- Use of nvl functions and function based index may be needed.

Agenda - Part II

- Few more issues continued...
 - Missing statistics on indexes
 - Missing statistics on columns
 - Column type mismatch
 - Index on non-selective columns
 - Index efficiency
 - Arithmetic comparison issues.

■ Hints

HINTS

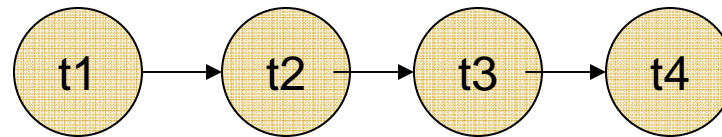
- ❑ Hints are directive to the optimizer and almost always honored by the optimizer.
- ❑ Hints are not honored only if the hints are inconsistent with itself or another hint.
- ❑ Avoid hints, if possible. Many software upgrade performance issues that I have seen is due to hints(bad).
- ❑ In some cases, hints are necessary evils ☹️

ORDERED

❑ ORDERED hint Dictates optimizer an exact sequence of tables to join [top to bottom or L->R canonically speaking].

Select ...

From t1,
t2,
t3,
t4

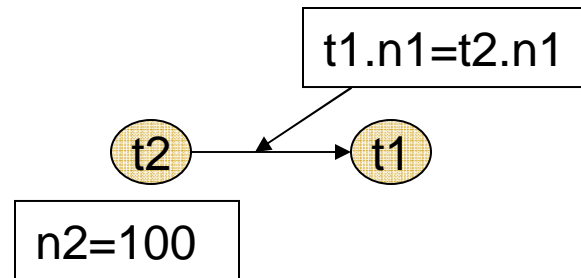


ORDERED

```

explain plan for
select /*+ ORDERED */
  t1.n1, t2.n2 from
  t_large2 t2,
  t_large t1
where t1.n1 = t2.n1 and t2.n2=100
/

```



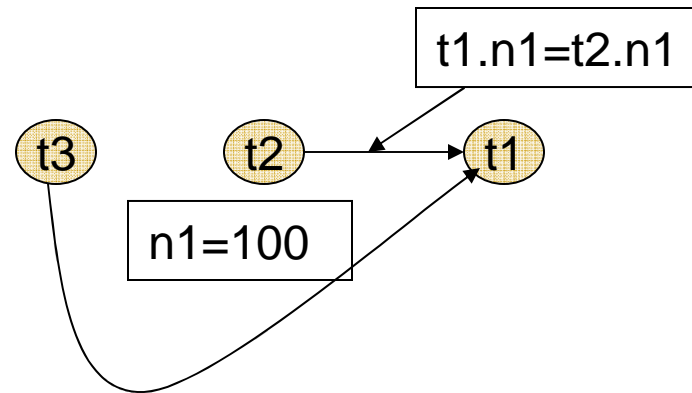
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	13	1401 (5)	00:00:08
1	NESTED LOOPS		1	13	1401 (5)	00:00:08
* 2	INDEX FAST FULL SCAN	T_LARGE2_N2	1	8	1399 (6)	00:00:07
* 3	INDEX RANGE SCAN	T_LARGE_N1	1	5	2 (0)	00:00:01

ORDERED

Optimizer did exactly what it is told to!

❑ Later developer added another table to the join..

```
explain plan for
select /*+ ORDERED */
  t1.n1, t2.n2 from
  t_large3 t3,
  t_large2 t2,
  t_large t1
where t1.n1 = t2.n1 and t2.n2=100
  and t1.n1=t3.n1
/
```



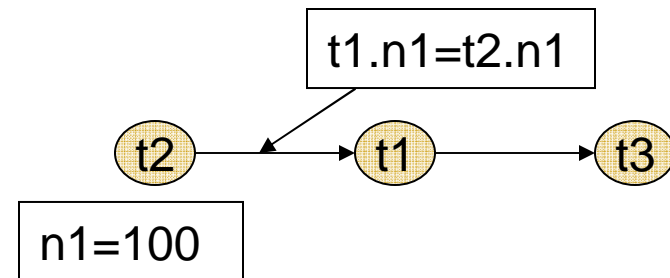
Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT		1	18		1397M (6)	999:59:59
* 1	HASH JOIN		1	18	11M	1397M (6)	999:59:59
2	MERGE JOIN CARTESIAN		499K	6345K		1397M (6)	999:59:59
3	INDEX FAST FULL SCAN	T_LARGE3_N1	999K	4880K		1161 (4)	00:00:06
4	BUFFER SORT		1	8		1397M (6)	999:59:59
* 5	INDEX FAST FULL SCAN	T_LARGE2_N2	1	8		1398 (6)	00:00:07
6	INDEX FAST FULL SCAN	T_LARGE_N1	1100K	5371K		1176 (5)	00:00:06

Ordered & leading

If you must, use leading instead of ordered..

```

explain plan for
select /*+ leading (t2) */
  t1.n1, t2.n2 from
  t_large3 t3,
  t_large2 t2,
  t_large t1
where t1.n1 = t2.n1 and t2.n2=100
      and t1.n1=t3.n1
/
  
```



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	18	1403 (5)	00:00:08
1	NESTED LOOPS		1	18	1403 (5)	00:00:08
2	NESTED LOOPS		1	13	1401 (5)	00:00:08
* 3	INDEX FAST FULL SCAN	T_LARGE2_N2	1	8	1399 (6)	00:00:07
* 4	INDEX RANGE SCAN	T_LARGE_N1	1	5	2 (0)	00:00:01
* 5	INDEX RANGE SCAN	T_LARGE3_N1	1	5	2 (0)	00:00:01

Index hint

Index hint specifies what index to use. Use hints such as cardinality. If you must use index hint, specify columns instead of index name.

```
explain plan for select /*+ index (t, t_large_n2) */ n2 from t_large t where n1=:b1;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	8	3 (0)	00:00:01
* 1	INDEX RANGE SCAN	T_LARGE_N2	1	8	3 (0)	00:00:01

```
drop index t_large_n2;  
create index t_large_n2 on t_large(n2, n1);
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	8	12 (0)	00:00:01
* 1	INDEX SKIP SCAN	T_LARGE_N2	1	8	12 (0)	00:00:01

```
explain plan for select /*+ cardinality(t,1) */ n2 from t_large t where n1=:b1;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	8	4 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	T_LARGE	1	8	4 (0)	00:00:01
* 2	INDEX RANGE SCAN	T_LARGE_N1	1		3 (0)	00:00:01

Agenda - Part II

- Few more issues continued...
 - Missing statistics on indexes
 - Missing statistics on columns
 - Column type mismatch
 - Index on non-selective columns
 - Index efficiency
 - Arithmetic comparison issues.
- Hints
- Few guidelines

Guidelines – CBO setup

- ❑ Setup optimizer parameters correctly.

- ❑ Setup optimizer_features_enable to correct software version.
Use four digits of base version for e.g. 10.2.0.4

- ❑ Collect system statistics
 - ❑ Make sure sreadtim/mreadtim are matching with workload.

 - ❑ MBRC – Multi block read count is set properly.

Guidelines – Parameter setup

- ❑ Setup `db_cache_size` properly. In most cases, SGA is too small for the workload.
- ❑ Setup `Compatible` parameter to correct software version.
- ❑ Setup `pga_aggregate_target` to higher value. CBO uses PGAT in cost calculations.

Guidelines – Object statistics

- ❑ Use higher estimate_percent for smaller tables and modest estimate_percent for large tables.
- ❑ Add optimal indices to the table. For DSS type applications use bitmap indices.
- ❑ For tables with volatile data, use dynamic sampling or lock the statistics with representative data.

Guidelines – Histograms

- ❑ Collect histograms on columns with skewed data and if those columns are specified in predicates.
- ❑ If collecting histograms, try to use `estimate_percent` of 100%.
- ❑ Stop collecting histogram on all columns.

Guidelines – Hints

- ❑ Avoid use of hints. If you must, use soft hints such as leading, first_rows etc.
- ❑ Hints might work today, but will be problematic during future upgrade.
- ❑ If optimizer is not choosing the optimal plan, understand root cause and resolve that. I know, Easier said than done.
- ❑ If the root cause can not be resolved, use outlines or sql profiles to resolve performance issue of that SQL statement.

Guidelines – Use SQL*Trace, tkprof and profiler

- ❑ Last, but not least, to resolve an SQL performance issue, use sqltrace or profiler to understand the bottleneck.
- ❑ Resolve that bottleneck to tune that SQL, rather than randomly trying various options.
- ❑ Structures of tables, indices also important. Use features such as bitmap indices, partitioning, compressed indices etc to improve index efficiency.

References

1. Oracle support site. Metalink.oracle.com. Various documents
2. Internal's guru Steve Adam's website
www.ixora.com.au
3. Jonathan Lewis' website
www.jlcomp.daemon.co.uk
4. Julian Dyke's website
www.julian-dyke.com
5. 'Oracle8i Internal Services for Waits, Latches, Locks, and Memory'
by Steve Adams
6. Randolph Geist : <http://oracle-randolf.blogspot.com>
7. Tom Kyte's website
[Asktom.oracle.com](http://asktom.oracle.com)