

Piotr Sajda PS5 👍

With 20 years of Oracle performance tuning experience and just as many years of windsurfing, I’m probably the best windsurfer among the Ora-tuning geeks and definitely the most “out-there” performance tuning specialist on the water!

Blog: piotrsajda.com/oracle-blog

E-mail: piotr.sajda@gmail.com

LinkedIn: https://www.linkedin.com/in/piotr-sajda-246304/

CLUF / CF / Clustering Factor

Almost legendary

Oh, let‘s clarify it beyond any doubt
and reach out to the Oracle DOC

Oracle documentation makes it clear, but...

CLUSTERING FACTOR:

Indicates the amount of order of the rows in the table based on the values of the index.
If the value is near the number of blocks, then the table is very well ordered. In this case, the index entries in a single leaf block tend to point to rows in the same data blocks.
If the value is near the number of rows, then the table is very randomly ordered. In this case, it is unlikely that index entries in the same leaf block point to rows in the same data blocks.

Oracle documentation makes it clear

if you already know it ☺

CLUSTERING FACTOR:

Indicates the amount of order of the rows in the table based on the values of the index.

If the value is near the number of blocks, then the table is very well ordered. In this case, the index entries in a single leaf block tend to point to rows in the same data blocks.
If the value is near the number of rows, then the table is very randomly ordered. In this case,
it is unlikely that index entries in the same leaf block point to rows in the same data blocks.

Have you heard this before ?

“Oh, our SQL does not behave because of the
HIGH CLUSTERING FACTOR!
How do we know it ?
Because we found an extremally high clustering factor (number) in dba_indexes”

“... it’s obvious, we need to rebuild that index !”

Feel home … #1

Select * from TSCLUF where COL = <value>

```
Execution Plan
----------------------------------------------------------
Plan hash value: 4266397642

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |   485 |   127K|    23   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| TSCLUF        |   485 |   127K|    23   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | IDX_ID_SORTED |   485 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```



1 – get to the index

2 – get to the table

OR maybe :

1 – get to the index

2 – get to the table to retrieve the same data

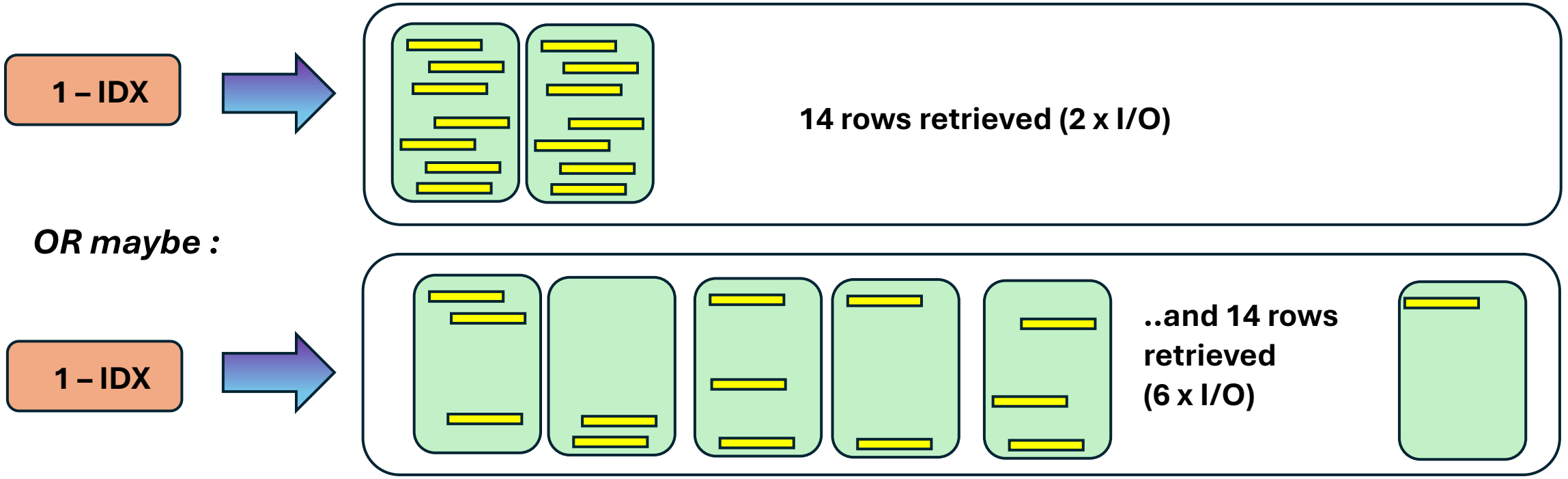
Feel home … (row level)

Select * from TSCLUF where COL = <value>

Execution Plan

```
----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |   485 |   127K|    23   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID | TSCLUF        |   485 |   127K|    23   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | IDX_ID_SORTED |   485 |       |     4   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------
```







Select COLOR From TSCLUF Where MYSTR = ‘ABC’

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |   485 |   127K|    23   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| TSCLUF        |   485 |   127K|    23   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | IDX_STR__SORT |   485 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Wait, could this represent a LOW CLUF…?
Yes!

Index Leaf Block	
rowidA	**ABC**
rowidB	**ABC**
rowidC	**ABC**
rowid…	*ABD*



Table Block – all 3 rows in 1 block (nice…)					
rowidA	**ABC**	Blue	*Y*	*10*	*Nice*
rowidC	**ABC**	Green	*Y*	*1*	*Ugly*
rowidB	**ABC**	White	*N*	*400*	*OK*
Rowid	*SAB*	*Purple*	*Y*	*32*	*Donou*

1 x I/O …and you have it all!

Select COLOR From TSCLUF Where MYSTR = ‘ABC’

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |   485 |   127K|    63   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| TSCLUF        |   485 |   127K|    63   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | IDX_STR__MESS |   485 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Could this then represent a HIGH CLUF…?
Indeed!

Index Leaf Block

rowidA	**ABC**
rowidB	**ABC**
rowidC	**ABC**
rowid…	*ABD*

Oopss.. 3 x I/O …and you have it all..

Table Block #34

rowidA	**ABC**	Blue	*Y*	*10*	*Nice*
rowidX	**ABS**	Black	*N*	*98*	*Ugly*
rowidY	**XYZ**	Blue	*N*	*20*	*OK*
…	….	….	…	…	…

Table Block #900

rowidW	**QQQ**	Blue	*Y*	*10*	*Nice*
rowidC	**ABC**	Green	*Y*	*1*	*Ugly*
rowidY	**XYZ**	Blue	*N*	*20*	*OK*
…	….	….	…	…	…

Table Block #10232

rowidT	**DDB**	Blue	*Y*	*10*	*Nice*
rowidX	**ABS**	Black	*N*	*98*	*Ugly*
rowidB	**ABC**	Blue	*N*	*400*	*OK*
…	….	….	…	…	…

I’ll tell you somethin’ …

… what I shouldn’t

Let‘s see it „raw“ - create table + 2 indexes

```
SQL>
create table TSCLUF as
select rownum as ID_SORTED,                          -- nicely sorted
trunc( dbms_random.value(1,10e6) ) as ID_MESS,   -- messy!
rpad('X',256,'X') as pad
from dual connect by level <= 10e6 ;

Table created.

SQL> create index IDX_ID_SORTED on TSCLUF ( ID_SORTED ) ;
Index created.


SQL> create index IDX_ID_MESS on TSCLUF ( ID_MESS ) ;
Index created.
```

How are the values of these two columns „shaped“ ?

```
SQL> select * from
     (
       select
         rownum id_sorted,
         trunc( dbms_random.value(1,10e6) ) ID_MESS
       from dual connect by level <= 10e6
     )
    where rownum <= 20 ;
```



Let’s make a sample

ID_SORTED	ID_MESS
1	4682246
2	6691
3	176981
4	5294352
5	2334970
6	2902623
7	4793470
8	1614453
9	294588
10	3108646
11	95708
12	4333752
13	3582110
14	5229318
15	8280
16	4420173
17	123315
18	1557922
19	71484
20	1540524

20 rows selected.

ID_SORTED → presorted

ID_MESS → a mess..!

.. And so they will be put in the “TSCLUF” table → blocks

Let‘s see it „raw“ - gather stats

```
SQL> exec DBMS_STATS.GATHER_TABLE_STATS (ownname => '&OWNER’ ,
                                         tabname => '&tablename’,
                                         cascade => true,
                                         estimate_percent => &estimate_percent,
                                        granularity => 'ALL’,
                                         degree => &parallel_slaves);
Enter value for owner: TSTUS
Enter value for tablename: TSCLUF
Enter value for estimate_percent: 100
Enter value for parallel_slaves: 2

PL/SQL procedure successfully completed.
```

Let‘s see it „raw“ - examine the outcome of stats

OWNER	TABLENAME	INDEX_NAME	CLUSTF	TabBlks	TabRows	NDV	LEAF_BLOCKS	MB
DELME	TESTCL	IDX_ID_MESS	9,999,973	385,478	10,000,000	6355456	23371	192
DELME		IDX_ID_SORTED	384,616	385,478	10,000,000	9914368	24002	192
	**********							----------
	sum							384

TABLE :
Number of **blocks** in the Table (TESTCL) : 385,478
Number of **rows** in the Table (TESTCL) : 10,000,000

Both INDEXES :
Clustering Factor in IDX_ID_SORTED (calculated by dbms_stats) : 384,616
Clustering Factor in IDX_ID_MESS (calculated by dbms_stats) : 9,999,973

LOW CLUF

Let's dump'em ..

We will dump :
Index Leaf Block → First Table Block based on that leaf

Dump of Index Tree → Leaf - LOW CLUF

```
from dba_objects :

OBJECT                OBJECT_ID   OBJECT_TYPE
--------------------- ---------- ---------------
TSTUS.IDX_ID_SORTED   181826     INDEX
```

```
SQL> select  DBMS_UTILITY.DATA_BLOCK_ADDRESS_FILE(288260284) "#File",
             DBMS_UTILITY.DATA_BLOCK_ADDRESS_BLOCK(288260284) "#Block"
            from dual;

#File     "#Block"
--------- -----------------
68        3047612
```

```
SQL> alter session set events 'immediate trace name treedump level 181826';

----- begin tree dump
branch: 0x112e80bb 288260283 (0: nrow: 38, level: 2)
   branch: 0xc618836 207718454 (-1: nrow: 672, level: 1)
      leaf: 0x112e80bc 288260284 (-1: row:485.485 avs:824)
      leaf: 0x112e80bd 288260285 (0: row:479.479 avs:816)
      leaf: 0x112e80be 288260286 (1: row:479.479 avs:816)
      leaf: 0x112e80bf 288260287 (2: row:479.479 avs:816)
      leaf: 0x112e80c0 288260288 (3: row:479.479 avs:816)
      leaf: 0x112e80c1 288260289 (4: row:478.478 avs:830)
      leaf: 0x112e80c2 288260290 (5: row:479.479 avs:816)
      leaf: 0x112e80c3 288260291 (6: row:479.479 avs:816)
      leaf: 0x112e80c4 288260292 (7: row:479.479 avs:816)
      leaf: 0x112e80c5 288260293 (8: row:478.478 avs:830)
(...)
(...)
```

```
SQL> alter system dump datafile 68 block 3047612 ;

System altered.
```

We are expecting to see the first 485 indexed values in the leaf :

Dump of Index Leaf Block - LOW CLUF

Index Leaf Block Dump :

```
row#0[8020] flag: -------, lock: 0, len=12
col 0; len 2; (2):  c1 02              --> SORTED_ID = 1 (decimal value)
col 1; len 6; (6):  11 71 ad d3 00 00
row#1[8008] flag: -------, lock: 0, len=12
col 0; len 2; (2):  c1 03              --> SORTED_ID = 2 (decimal value)
col 1; len 6; (6):  11 71 ad d3 00 01
row#2[7996] flag: -------, lock: 0, len=12
col 0; len 2; (2):  c1 04              --> SORTED_ID = 3 (decimal value)
col 1; len 6; (6):  11 71 ad d3 00 02
row#3[7984] flag: -------, lock: 0, len=12
col 0; len 2; (2):  c1 05
col 1; len 6; (6):  11 71 ad d3 00 03
row#4[7972] flag: -------, lock: 0, len=12
col 0; len 2; (2):  c1 06
col 1; len 6; (6):  11 71 ad d3 00 04
(…)
(…) - skipped ..
(…)
row#484[1830] flag: -------, lock: 0, len=13
col 0; len 3; (3):  c2 05 56         --> SORTED_ID = 485 (decimal value)
col 1; len 6; (6):  11 71 ad de 00 10
```

Quick (reversed) check :

```
SQL> select dump(1,16) from dual ;

DUMP(1,16)
-----------------
Typ=2 Len=2: c1,2      --> matching the row#0, col0
```

```
SQL> select dump(2,16) from dual ;

DUMP(2,16)
-----------------
Typ=2 Len=2: c1,3     --> matching the row#1, col0
```

```
SQL> select dump(485,16) from dual ;

DUMP(485,16)
--------------------
Typ=2 Len=3: c2,5,56   --> matching the row#484, col0
```

.. and there we have it!

Finally we reach to the table! - LOW CLUF

Magic number : 19

```
SQL> select
count(distinct DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid))
"amount_blks"
from TSTUS.TSCLUF
where id_sorted <= 485 ;
```

```
amount_blks
-----------
         19
```



Ah, the values of ID_SORTED ranging from **1 to 485** which fill the first leaf of the **index**, are distributed across **19 physical blocks of the table**.

Wait… *what* do we “have it” ?

Magic number : 19

```
SQL> select
count(distinct DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid))
"amount_blks"
from TSCLUF
where id_sorted <= 485 ;
```

```
amount_blks
-----------
         19
```



It does not tell us much as long as we do not compare this number with the corresponding number from HIGH CLUF

.. still with the table - LOW CLUF

```
SQL> select
   DBMS_ROWID.ROWID_RELATIVE_FNO(rowid) as "#File",
   DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid) "#Block",
   count(*) "Number_rows_in_this_block"
from  TSCLUF
where id_sorted <= 485
group by DBMS_ROWID.ROWID_RELATIVE_FNO(rowid),
DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid)
order by 1,2 ;
```

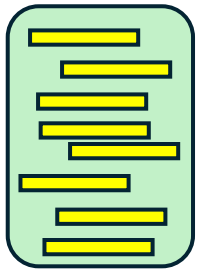
#File	#Block	Number_rows_in_this_block
54	**2646760**	**26**
54	2646761	26
54	2646762	26
54	2646763	26
54	2646764	26
54	2646765	26
54	2646766	26
54	2646767	26
69	3255763	26
69	3255764	26

(…)

19 rows selected. → *ah, see - it's number of blocks from the previous slide!*

What did we get from this query?

In each of these 19 table blocks you'll find **26 rows** that satisfy this predicate: **where id_sorted <= 485**



.. Won‘t leave you (table)! - LOW CLUF

```
SQL> break on #Block skip 1

SQL> select
rowid,
DBMS_ROWID.ROWID_RELATIVE_FNO(rowid) as "#File",
DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid) "#Block" ,
DBMS_ROWID.ROWID_ROW_NUMBER(rowid) "#Row",
ID_SORTED
from TSTUS.TSCLUF
where id_sorted <= 485
order by "#Block", "#Row" ;
```

We can expect 19 different blocks with 26 rows (#Row ranging from 0-25) in each block with altogether 485 rows.

This is indeed what we achieved :



ROWID	#File	#Block	#Row	ID_SORTED
AAAsZBAA2AAKGLoAAA	54	2646760	0	131
AAAsZBAA2AAKGLoAAB	54		1	132
AAAsZBAA2AAKGLoAAC	54		2	133
(...)				
AAAsZBABFAAMa3TAAA	69	3255763	0	1
AAAsZBABFAAMa3TAAB	69		1	2
AAAsZBABFAAMa3TAAC	69		2	3
AAAsZBABFAAMa3TAAD	69		3	4
AAAsZBABFAAMa3TAAE	69		4	5
AAAsZBABFAAMa3TAAF	69		5	6
AAAsZBABFAAMa3TAAG	69		6	7
AAAsZBABFAAMa3TAAH	69		7	8
AAAsZBABFAAMa3TAAI	69		8	9
AAAsZBABFAAMa3TAAJ	69		9	10
AAAsZBABFAAMa3TAAK	69		10	11
AAAsZBABFAAMa3TAAL	69		11	12
AAAsZBABFAAMa3TAAM	69		12	13
AAAsZBABFAAMa3TAAN	69		13	14
AAAsZBABFAAMa3TAAO	69		14	15
AAAsZBABFAAMa3TAAP	69		15	16
AAAsZBABFAAMa3TAAQ	69		16	17
AAAsZBABFAAMa3TAAR	69		17	18
AAAsZBABFAAMa3TAAS	69		18	19
AAAsZBABFAAMa3TAAT	69		19	20
AAAsZBABFAAMa3TAAU	69		20	21
AAAsZBABFAAMa3TAAV	69		21	22
AAAsZBABFAAMa3TAAW	69		22	23
AAAsZBABFAAMa3TAAX	69		23	24
AAAsZBABFAAMa3TAAY	69		24	25
AAAsZBABFAAMa3TAAZ	69		25	**26**

Block 8KB

485 rows selected --> ah, look it is : 19 blocks x 26 rows/block (sometimes less rows)

.. „let‘s dump‘em!“ (**table block**) - LOW CLUF

```
SQL>  alter system dump datafile 69 block 3255763 ;

System altered.
```

Reminder :

```
SQL> create table TSTUS.TSCLUF as
select rownum as ID_SORTED,
-- nicely sorted
trunc(dbms_random.value(1,10e6)) as ID_MESS, -- messy!
rpad('X',256,'X') as pad
from dual connect by level <= 10e6 ;
```

the values for “col 0” represent the ID_SORTED
and are indeed *sorted* within the table block

Dump of a single block of **the TABLE** :

```
tab 0, row 0, @0x1e71
tl: 271 fb: --H-FL--  lb: 0x0  cc: 3
col  0: [ 2]  c1 02          --> value: 1 (integer)
col  1: [ 5]  c4 04 0c 58 36
col  2: [256]
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
(...)
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
58 58 58 58 58 58
tab 0, row 1, @0x1d62
tl: 271 fb: --H-FL--  lb: 0x0  cc: 3
col  0: [ 2]  c1 03          --> value: 2 (integer)
col  1: [ 5]  c4 04 30 02 3d
col  2: [256]
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
(...)
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
58 58 58 58 58 58

… more entries … …
… until the last one: row 25 (26th entry) :

tab 0, row 25, @0x3fb
tl: 271 fb: --H-FL--  lb: 0x0  cc: 3
col  0: [ 2]  c1 1b          --> value: 26 (integer)
col  1: [ 5]  c4 07 4c 22 51
col  2: [256]
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
(...)
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
58 58 58 58 58 58
end_of_block_dump
```

Block 8KB

Very demanding quiz –

```
SQL>  alter system dump datafile 69 block 3255763 ;

System altered.
```

Reminder :

```
SQL> create table TSTUS.TSCLUF as
select rownum as ID_SORTED,

trunc(dbms_random.value(1,10e6))
as ID_MESS, -- messy!

rpad('X',256,'X') as pad

from dual connect by level <= 10e6 ;
```

?

```
Dump of a single block of the TABLE :

tab 0, row 0, @0x1e71
tl: 271 fb: --H-FL-- lb: 0x0  cc: 3
col  0: [ 2]  c1 02
col  1: [ 5]  c4 04 0e 58 36
col  2: [256]
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
(...)
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
58 58 58 58 58 58
tab 0, row 1, @0x1d62
tl: 271 fb: --H-FL-- lb: 0x0  cc: 3
col  0: [ 2]  c1 03
col  1: [ 5]  c4 04 30 02 3d
col  2: [256]
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
(...)
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
58 58 58 58 58 58

… more entries … …
… until the last one: row 25 (26th entry) :

tab 0, row 25, @0x3fb
tl: 271 fb: --H-FL-- lb: 0x0  cc: 3
col  0: [ 2]  c1 1b
col  1: [ 5]  c4 07 4e 22 51
col  2: [256]
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
(...)
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
58 58 58 58 58 58
end_of_block_dump
```

HIGH CLUF

Let’s dump’em ..

We will dump again:
Index Leaf Block → FirstTable Block based on that leaf

Dump Index Tree and its First Leaf - HIGH CLUF

```
from dba_objects :

OBJECT               OBJECT_ID   OBJECT_TYPE
-------------------- ---------- ---------------
IDX_ID_MESS          181827     INDEX
```

```
SQL> select  DBMS_UTILITY.DATA_BLOCK_ADDRESS_FILE(284573884) "#File",
             DBMS_UTILITY.DATA_BLOCK_ADDRESS_BLOCK(284573884) "#Block"
             from dual;

             #File      "#Block"
             --------- -----------------
             67         3555516
```

```
SQL> alter session set events 'immediate trace name treedump level 181827';
----- begin tree dump
branch: 0x10f640bb 284573883 (0: nrow: 40, level: 2)
   branch: 0xc142b14 202648340 (-1: nrow: 638, level: 1)
      leaf: 0x10f640bc 284573884 (-1: row:485.485 avs:821)
      leaf: 0x10f640bd 284573885 (0: row:479.479 avs:816)
      leaf: 0x10f640be 284573886 (1: row:478.478 avs:830)
      leaf: 0x10f640bf 284573887 (2: row:479.479 avs:818)
      leaf: 0x10f640c0 284573888 (3: row:478.478 avs:829)
      leaf: 0x10f640c1 284573889 (4: row:478.478 avs:827)
      leaf: 0x10f640c2 284573890 (5: row:479.479 avs:820)
      leaf: 0x10f640c3 284573891 (6: row:479.479 avs:821)
      leaf: 0x10f640c4 284573892 (7: row:478.478 avs:829)
(…)
      leaf: 0xda462ef 228877039 (338: row:422.422 avs:827)
      leaf: 0xda462f0 228877040 (339: row:422.422 avs:822)
      leaf: 0xda462f1 228877041 (340: row:41.41 avs:7299)
----- end tree dump
```

```
SQL> alter system dump datafile 67 block 3555516 ;

System altered.
```

On the next slide we are supposed to see the first 485 indexed values in the index leaf :

Make sure: Index Leaf Block dump - HIGH CLUF

```
Index Leaf Block Dump :

row#0[8020] flag: -------, lock: 0, len=12
col 0; len 2; (2):  c1 02                  --> MESS_ID = 1
col 1; len 6; (6):  0d a4 3f a3 00 15

row#1[8008] flag: -------, lock: 0, len=12
col 0; len 2; (2):  c1 07                  --> MESS_ID = 6
col 1; len 6; (6):  11 71 fc b3 00 0e
row#2[7996] flag: -------, lock: 0, len=12

col 0; len 2; (2):  c1 08                  --> MESS_ID = 7
col 1; len 6; (6):  10 f6 a3 d6 00 0f
row#3[7984] flag: -------, lock: 0, len=12
col 0; len 2; (2):  c1 08
col 1; len 6; (6):  10 f6 ef b9 00 0e
(..)
(..)
row#483[1840] flag: -------, lock: 0, len=13
col 0; len 3; (3):  c2 05 62
col 1; len 6; (6):  0d 9d 84 4a 00 08
row#484[1827] flag: -------, lock: 0, len=13
col 0; len 3; (3):  c2 05 62               --> MESS_ID = 497
col 1; len 6; (6):  11 35 3c ce 00 18
----- end of leaf block Logical dump -----
```

Quick (reversed) check :

```
SQL> select dump(1,16) from dual ;

DUMP(1,16)
-----------------
Typ=2 Len=2: c1,2      --> matching the row#0, col0
```

```
SQL> select dump(6,16) from dual ;

DUMP(2,16)
-----------------
Typ=2 Len=2: c1,7     --> matching the row#1, col0
```

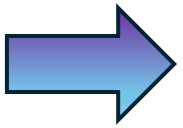
```
SQL> select dump(497,16) from dual ;

DUMP(485,16)
--------------------
Typ=2 Len=3: c2,5,62   --> matching the row#484, col0
```

#Blks in Table for first Idx Leaf - HIGH / LOW CLUF

```
SQL> select
count(distinct DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid))
"amount_blks"
from TSCLUF
where id_sorted <= 485 ;

amount_blks
-----------
         19
```

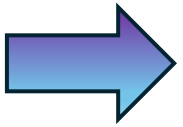


LOW Clustering Factor :

Ah, the values of ID_SORTED ranging from **1 to 485** (which fill the first leaf of the index), are distributed across just **19 physical table blocks**.

```
SQL> select
count(distinct DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid) )
"amount_blks"
from TSCLUF
where id_mess <= 485 ;

amount_blks
-----------
        469
```



HIGH Clustering Factor :

The values of ID_MESS ranging from **1 to 485** (which fill the first leaf of the index), are distributed across as many as **469 physical table blocks !**

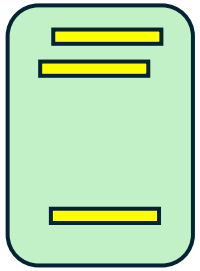
If you think that this is going to look like this ...

..then you're right !

HIGH CLUF vs. LOW CLUF (table block)

HIGH Clustering Factor

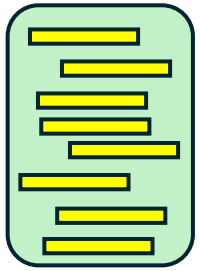
Content of a single table block
(**sparsely** populated)



*The values of ID_MESS ranging from **1 to 485** which fill the first leaf of the **index**, are distributed across **469 physical table blocks**.*

LOW Clustering Factor

Content of a single table block
(**densely** populated)



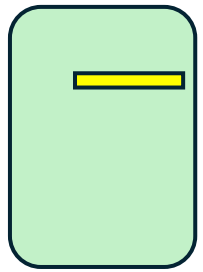
*The values of ID_SORTED ranging from **1 to 485** which fill the first leaf of the **index**, are distributed across **19 physical table blocks**.*

#rows/block for HIGH CLUF vs. LOW CLUF

```
SQL> select
   DBMS_ROWID.ROWID_RELATIVE_FNO(rowid) as "#File",
   DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid) "#Block",
   count(*) "Number_rows_in_this_block"
from  TSTUS.TSCLUF
where ID_MESS <= 485
group by DBMS_ROWID.ROWID_RELATIVE_FNO(rowid),
DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid)
order by "#File", "#Block" ;
```

#File	#Block	Number_rows_in_this_block
45	**2048617**	**1**
45	2048794	1
45	2048926	1
45	2049315	1
46	2057193	1
50	2352355	1
50	2352375	1
50	2352679	1
(…)		
71	2159697	1
71	2162949	1
71	2162979	1

469 rows selected. → *ah, see - it's also the number of blocks!*

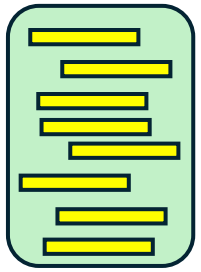


Do you remember ..?

```
SQL> select
   DBMS_ROWID.ROWID_RELATIVE_FNO(rowid) as "#File",
   DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid) "#Block",
   count(*) "Number_rows_in_this_block"
from  TSTUS.TSCLUF
where id_sorted <= 485
group by DBMS_ROWID.ROWID_RELATIVE_FNO(rowid),
DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid)
order by 1,2 ;
```

#File	#Block	**Number_rows_in_this_block**
54	**2646760**	**26**
54	2646761	26
54	2646762	26
54	2646763	26
54	2646764	26
54	2646765	26
54	2646766	26
54	2646767	26
69	3255763	26
69	3255764	26
(…)		

19 rows selected. → *ah, see - it's also the number of blocks!*



I couldn’t resist –

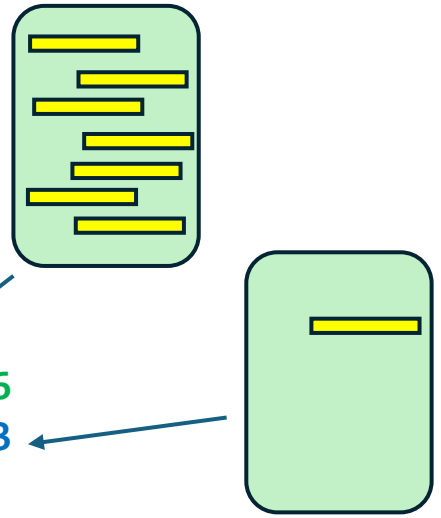
let’s take a look at the stats again

Let‘s examine the outcome of the stats again

OWNER	TABLENAME	INDEX_NAME	CLUSTF	TabBlks	TabRows	NDV	LEAF_BLOCKS	MB
DELME	TESTCL	IDX_ID_MESS	9,999,973	385,478	10,000,000	6355456	23371	192
DELME		IDX_ID_SORTED	384,616	385,478	10,000,000	9914368	24002	192
	**********							----------
	sum							384

TABLE :
Number of **blocks** in the Table (TESTCL) : 385,478
Number of **rows** in the Table (TESTCL) : 10,000,000

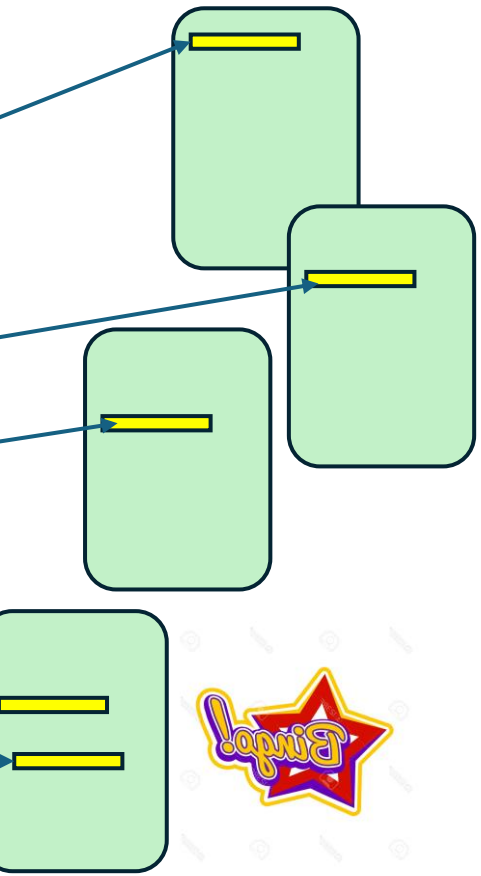
Both INDEXES :
Clustering Factor in IDX_ID_SORTED (calculated by dbms_stats) : 384,616
Clustering Factor in IDX_ID_MESS (calculated by dbms_stats) : 9,999,973



HIGH CLUF – number of rows / block

```
SQL> break on #Block skip 1
SQL> select rowid,
            DBMS_ROWID.ROWID_RELATIVE_FNO(rowid) as "#File",
            DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid) "#Block" ,
            DBMS_ROWID.ROWID_ROW_NUMBER(rowid) "#Row",
            ID_MESS
     from TSTUS.TSCLUF  where id_mess  <= 485
     order by "#File“, "#Block", "#Row“
```

ROWID	#File	#Block	#Row	ID_MESS
AAAsZBAA2AAF0DzAAC	54	1523955	2	118
AAAsZBAA2AAF0D/AAJ	54	1523967	9	183
AAAsZBABHAAG+SQAAP	71	1827984	15	102
AAAsZBAA2AAHYRmAAD	54	1934438	3	458
(…)				
AAAsZBABDAAN1saAAO	**67**	**3627802**	**14**	**381**
AAAsZBABDAAN1saAAR	**67**		**17**	**8**
AAAsZBABDAAN1tlAAO	67	3627877	14	40



Closer to the reality -
let's do the tiny little wee Index Range Scan

SQL with :
1 Index Leaf Block and <some blks> from the table - let's see the <some>

Closer to the reality

Let’s go through the cases represented by SELECT statements and examine the **true I/O effort** measured with the number of physical reads which are behind a query retrieving its data from the table, by the means of :

ID_SORTED (Low CLUF)

ID_MESS (High CLUF)

.. in the predicate (WHERE) section

LOW CLUF: estimation of I/O impact

Set 10046 trace and flush buffer cache

```
SQL> exec
dbms_monitor.session_trace_enable(session_id=>&sid_to_trace,serial_num=>&serial_to_trace,bind
s=>true,waits=>true);
Enter value for sid_to_trace: 2907
Enter value for serial_to_trace: 40588

SQL> alter system flush buffer_cache ;

System altered.

SQL> select * from TSTUS.TSCLUF where id_sorted <= 485 ;

485 rows selected.

Elapsed: 00:00:00.02

Execution Plan
----------------------------------------------------------
Plan hash value: 4266397642

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |   485 |   127K|    23   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| TSCLUF        |   485 |   127K|    23   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | IDX_ID_SORTED |   485 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

```
Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
         88  consistent gets
         23  physical reads
          0  redo size
     143555  bytes sent via SQL*Net to client
        784  bytes received via SQL*Net from client
         34  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        485  rows processed
```

*Yes, it does not tell us much as long as we do not compare this number with the corresponding number from **high CLUF***

LOW CLUF: estimation of I/O impact

Physical reads performed on behalf of index IDX_ID_SORTED (object_id = 181826) :

```
# cat CDBTST1_ora_187041_lowcluf.trc | grep WAIT | grep physical  | grep 181826

WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 867 cellhash#=3612651036 diskhash#=4260375198 bytes=8192 obj#=181826
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 680 cellhash#=439186241 diskhash#=3459493587 bytes=8192 obj#=181826
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 415 cellhash#=3612651036 diskhash#=4260375198 bytes=8192 obj#=181826
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 333 cellhash#=3612651036 diskhash#=4260375198 bytes=8192 obj#=181826
```

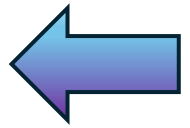
Why 4 not 3 I/Os? → https://piotrsajda.com/oracle-blog/why-this-extra-physical-read-while-performing-index-scan/

Physical reads performed on behalf of table TSCLUF (object_id = 181825) :

```
# cat CDBTST1_ora_187041_lowcluf.trc | grep WAIT | grep physical  | grep 181825

WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 779 cellhash#=1290911885 diskhash#=1644083320 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 316 cellhash#=1290911885 diskhash#=1644083320 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 262 cellhash#=1290911885 diskhash#=1644083320 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 311 cellhash#=1290911885 diskhash#=1644083320 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 233 cellhash#=1290911885 diskhash#=1644083320 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 666 cellhash#=3018416126 diskhash#=3925360322 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 263 cellhash#=3018416126 diskhash#=3925360322 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 296 cellhash#=3018416126 diskhash#=3925360322 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 178 cellhash#=3018416126 diskhash#=3925360322 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 197 cellhash#=3018416126 diskhash#=3925360322 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 196 cellhash#=3018416126 diskhash#=3925360322 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 151 cellhash#=3018416126 diskhash#=3925360322 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 171 cellhash#=3018416126 diskhash#=3925360322 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 365 cellhash#=1290911885 diskhash#=1644083320 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 282 cellhash#=1290911885 diskhash#=1644083320 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 269 cellhash#=1290911885 diskhash#=1644083320 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 311 cellhash#=1290911885 diskhash#=1644083320 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 178 cellhash#=1290911885 diskhash#=1644083320 bytes=8192 obj#=181825
WAIT #140088361959968: nam='cell single block physical read: flash cache' ela= 200 cellhash#=1290911885 diskhash#=1644083320 bytes=8192 obj#=181825
```

10046 trace



Index part

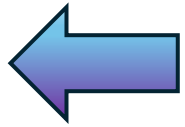
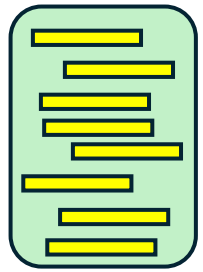


Table part



x 19

```
# cat CDBTST1_ora_187041_lowcluf.trc | grep WAIT | grep physical  | grep 181826 | wc -l
4
```

```
# cat CDBTST1_ora_187041_lowcluf.trc | grep WAIT | grep physical  | grep 181825 | wc -l
19
```

HIGH CLUF: estimation of I/O impact

Set 10046 trace and flush buffer cache

```
SQL> exec
dbms_monitor.session_trace_enable(session_id=>&sid_to_trace,serial_num=>&serial_to_trace,bind
s=>true,waits=>true);
Enter value for sid_to_trace: 2907
Enter value for serial_to_trace: 31646

SQL> alter system flush buffer_cache ;

System altered.


SQL> select * from TSTUS.TSCLUF where ID_MESS <= 485 ;

470 rows selected.

Elapsed: 00:00:00.08

Execution Plan
----------------------------------------------------------
Plan hash value: 3542504988

-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |             |   486 |   127K|   490   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TSCLUF      |   486 |   127K|   490   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | IDX_ID_MESS |   486 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

```
Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
        505  consistent gets
        472  physical reads
          0  redo size
      13563  bytes sent via SQL*Net to client
        765  bytes received via SQL*Net from client
         33  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        470  rows processed
```

HIGH CLUF: estimation of I/O impact

Physical reads performed on behalf of index IDX_ID_MESS (object_id = 181827) :

```
# cat CDBTST1_ora_239127_hi_cluf.trc | grep WAIT | grep physical | grep 181827

WAIT #139803498625344: nam='cell single block physical read: flash cache' ela= 2449 cellhash#=4259318641 diskhash#=1265020198 bytes=8192 obj#=181827
WAIT #139803498625344: nam='cell single block physical read: flash cache' ela= 703 cellhash#=1290911885 diskhash#=2137534647 bytes=8192 obj#=181827
WAIT #139803498625344: nam='cell single block physical read: flash cache' ela= 311 cellhash#=4259318641 diskhash#=1265020198 bytes=8192 obj#=181827
```

Physical reads performed on behalf of table (object_id = 181825) :

```
# cat CDBTST1_ora_239127_hi_cluf.trc | grep WAIT | grep physical | grep 181825

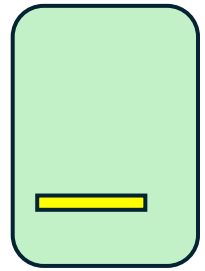
WAIT #139803498625344: nam='cell single block physical read: flash cache' ela= 2135 cellhash#=439186241 diskhash#=3981647790 bytes=8192 obj#=181825
WAIT #139803498625344: nam='cell single block physical read: flash cache' ela= 742 cellhash#=1810516308 diskhash#=1028073414 bytes=8192 obj#=181825
WAIT #139803498625344: nam='cell single block physical read: flash cache' ela= 1997 cellhash#=3018416126 diskhash#=3263235976 bytes=8192 obj#=181825
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 375 cellhash#=0 diskhash#=0 blocks=13 obj#=181825 tim=48088787822650
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 302 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787827672
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 303 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787830565
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 300 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787833128
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 303 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787835531
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 384 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787838367
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 400 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787840232
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 284 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787841973
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 238 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787843783
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 278 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787845159
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 308 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787846260
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 286 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787848015
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 265 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787849530
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 272 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787851241
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 294 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787852730
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 279 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787854184
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 315 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787855098
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 311 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787855887
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 302 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787856696
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 300 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787857460
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 369 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787858267
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 300 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787859060
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 256 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787859853
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 270 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787860593
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 317 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787861333
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 317 cellhash#=0 diskhash#=0 blocks=14 obj#=181825 tim=48088787862223
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 334 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787863024
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 409 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787863918
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 289 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787864682
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 311 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787865484
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 308 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787866654

WAIT #139803498625344: nam='cell list of blocks physical read' ela= 385 cellhash#=0 diskhash#=0 blocks=4 obj#=181825 tim=48088787867339
```

10046 trace

Index part

Table part



x 469

```
Execution Plan
----------------------------------------------------------
Plan hash value: 3542504988

-------------------------------------------------------------------------------------------
| Id  | Operation                           | Name        | Rows  | Bytes | Cost (%CP
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |             |   486 |   127K|   490   (
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TSCLUF      |   486 |   127K|   490   (
|*  2 |   INDEX RANGE SCAN                  | IDX_ID_MESS |   486 |       |     4   (
-------------------------------------------------------------------------------------------
```

HIGH CLUF: estimation of I/O impact

```
Physical reads performed on behalf of index IDX_ID_MESS (object_id = 181827) :

# cat CDBTST1_ora_239127_hi_cluf.trc | grep WAIT | grep physical | grep 181827

WAIT #139803498625344: nam='cell single block physical read: flash cache' ela= 2449 cellhash#=4259318641 diskhash#=1265020198 bytes=8192 obj#=181827
WAIT #139803498625344: nam='cell single block physical read: flash cache' ela= 703 cellhash#=1290911885 diskhash#=2137534647 bytes=8192 obj#=181827
WAIT #139803498625344: nam='cell single block physical read: flash cache' ela= 311 cellhash#=4259318641 diskhash#=1265020198 bytes=8192 obj#=181827

Physical reads performed on behalf of table (object_id = 181825) :

# cat CDBTST1_ora_239127_hi_cluf.trc | grep WAIT | grep physical | grep 181825

WAIT #139803498625344: nam='cell single block physical read: flash cache' ela= 2135 cellhash#=439186241 diskhash#=3981647790 bytes=8192 obj#=181825
WAIT #139803498625344: nam='cell single block physical read: flash cache' ela= 742 cellhash#=1810516308 diskhash#=1028073414 bytes=8192 obj#=181825
WAIT #139803498625344: nam='cell single block physical read: flash cache' ela= 1997 cellhash#=3018416126 diskhash#=3263235976 bytes=8192 obj#=181825
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 375 cellhash#=0 diskhash#=0 blocks=13 obj#=181825 tim=48088787822650
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 302 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787827672
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 323 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787830565
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 332 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787833128
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 303 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787835531
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 384 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787838367
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 400 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787840232
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 284 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787841973
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 238 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787843703
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 278 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787845159
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 308 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787846860
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 2[illegible] cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787848015
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 2[illegible] cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787849530
WAIT #139803498625344: nam='cell list of blocks physical read' ela= [illegible] cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787851241
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 294 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787852730
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 279 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787854184
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 315 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787855098
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 311 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787855887
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 302 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787856696
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 300 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787857460
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 369 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787858267
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 300 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787859060
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 256 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787859853
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 270 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787860593
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 317 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787861333
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 317 cellhash#=0 diskhash#=0 blocks=14 obj#=181825 tim=48088787862223
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 334 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787863024
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 409 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787863918
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 289 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787864682
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 311 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787865484
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 308 cellhash#=0 diskhash#=0 blocks=15 obj#=181825 tim=48088787866654
WAIT #139803498625344: nam='cell list of blocks physical read' ela= 385 cellhash#=0 diskhash#=0 blocks=4 obj#=181825 tim=48088787867339
```

Index part

10046 trace

Table part

```
# cat CDBTST1_ora_239127_hi_cluf.trc | grep WAIT | grep physical | grep 181825 |
grep "blocks=15" | wc -l
29

# cat CDBTST1_ora_239127_hi_cluf.trc | grep WAIT | grep physical | grep 181825 |
grep "blocks=13" | wc -l
1

# cat CDBTST1_ora_239127_hi_cluf.trc | grep WAIT | grep physical | grep 181825 |
grep "blocks=4" | wc -l
1

# cat CDBTST1_ora_239127_hi_cluf.trc | grep WAIT | grep physical | grep 181825 |
grep "blocks=14" | wc -l
1


-------
Total physical reads performed on behalf of the table:

29*15 + 1*13 + 1*14 + 1*4 + 3(single reads) = 469
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |             |   486 |   127K|   490   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TSCLUF      |   486 |   127K|   490   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | IDX_ID_MESS |   486 |       |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
        505  consistent gets
        472  physical reads
```

HIGH & LOW CLUF: comparison of I/O impact

HIGH CLUF :

```
SQL> select * from TSTUS.TSCLUF where ID_MESS <= 485 ;

470 rows selected.

Elapsed: 00:00:00.08

-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |             |   486 |   127K|   490   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TSCLUF      |   486 |   127K|   490   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | IDX_ID_MESS |   486 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
        505  consistent gets
        472  physical reads
```

LOW CLUF :

```
SQL> select * from TSTUS.TSCLUF where id_sorted <= 485 ;

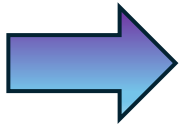
485 rows selected.

Elapsed: 00:00:00.02

-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |   485 |   127K|    23   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID | TSCLUF        |   485 |   127K|    23   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | IDX_ID_SORTED |   485 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
         88  consistent gets
         23  physical reads
```

High CLUF: 3 + “469” physical I/O
Low CLUF: 4 + 19 physical I/O



~20 times less I/O

What about Time, Memory and CPU utilization ?

Even closer to the reality

Let’s go through the cases represented by SELECT statements and examine the **true Time , CPU and Memory (Buffer Cache) effort**

ID_SORTED (Low CLUF)

ID_MESS (High CLUF)

.. in the predicate (WHERE) section – this time with much bigger range: **10% of total amount of rows in the table** (1e6 / 10e6)

Even closer to the reality: **Time Elapsed**

Let’s start from the comparison of: Time Elapsed for

ID_SORTED (Low CLUF)

ID_MESS (High CLUF)

*Each time I have **flushed the buffer cache** in order to force physical I/Os*

HIGH & LOW CLUF: comparison of Time Elapsed

```
SQL> select * from TSTUS.TSCLUF T where id_sorted <= 1e6 ;

1000000 rows selected.

Elapsed: 00:00:16.25

-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |  1000K|   256M| 40883   (1)| 00:00:02 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TSCLUF        |  1000K|   256M| 40883   (1)| 00:00:02 |
|*  2 |   INDEX RANGE SCAN                  | IDX_ID_SORTED |  1000K|       |  2409   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
     171311  consistent gets
     40'690  physical reads
    1000000  rows processed
```

```
SQL> select /*+ index(T IDX_ID_MESS) */ * from TSTUS.TSCLUF T where
id_mess <= 1e6 ;

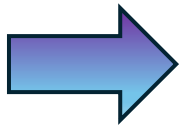
1000329 rows selected.

Elapsed: 00:00:33.36

-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |             |  1000K|   256M|  1002K  (1)| 00:00:40 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TSCLUF      |  1000K|   256M|  1002K  (1)| 00:00:40 |
|*  2 |   INDEX RANGE SCAN                  | IDX_ID_MESS |  1000K|       |  2346   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
    1069104  consistent gets
    361'957  physical reads
    1000329  rows processed
```

Low CLUF: 16 seconds
High CLUF: 33 seconds



~2 times faster

HIGH & LOW CLUF: comparison of Time Elapsed

```
SQL> alter session set "_optimizer_batch_table_access_by_rowid" = false ;

SQL> select * from TSTUS.TSCLUF T where id_sorted <= 1e6 ;

1000000 rows selected.

Elapsed: 00:00:16.12

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |  1000K|   256M| 40883   (1)| 00:00:02 |
|   1 |  TABLE ACCESS BY INDEX ROWID| TSCLUF        |  1000K|   256M| 40883   (1)| 00:00:02 |
|*  2 |   INDEX RANGE SCAN           | IDX_ID_SORTED |  1000K|       |  2409   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
     171311  consistent gets
     40'690  physical reads
    1000000  rows processed
```

```
SQL> alter session set "_optimizer_batch_table_access_by_rowid" = false ;

SQL> select /*+ index(T IDX_ID_MESS) */ * from TSTUS.TSCLUF T where
id_mess <= 1e6 ;

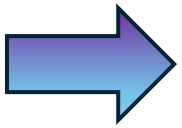
1000329 rows selected.

Elapsed: 00:00:48.24

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |  1000K|   256M|  1002K  (1)| 00:00:40 |
|   1 |  TABLE ACCESS BY INDEX ROWID| TSCLUF      |  1000K|   256M|  1002K  (1)| 00:00:40 |
|*  2 |   INDEX RANGE SCAN           | IDX_ID_MESS |  1000K|       |  2346   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
    1069093  consistent gets
    361'957  physical reads
    1000329  rows processed
```

Low CLUF: 16 seconds
High CLUF: 48 seconds



3 times faster ! (I/O batching did a good job)

“Gotcha”

LOW & HIGH CLUF: comparison of I/O

```
SQL> select * from TSTUS.TSCLUF T where id_sorted <= 1e6 ;

1000000 rows selected.

Elapsed: 00:00:16.12

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |  1000K|   256M| 40883   (1)| 00:00:02 |
|   1 |  TABLE ACCESS BY INDEX ROWID| TSCLUF        |  1000K|   256M| 40883   (1)| 00:00:02 |
|*  2 |   INDEX RANGE SCAN           | IDX_ID_SORTED |  1000K|       |  2409   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
     171311  consistent gets
     40'690  physical reads
    1000000  rows processed
```

```
SQL> select /*+ index(T IDX_ID_MESS) */ * from TSTUS.TSCLUF T where
id_mess <= 1e6 ;

1000329 rows selected.

Elapsed: 00:00:48.24


-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |  1000K|   256M|  1002K  (1)| 00:00:40 |
|   1 |  TABLE ACCESS BY INDEX ROWID| TSCLUF      |  1000K|   256M|  1002K  (1)| 00:00:40 |
|*  2 |   INDEX RANGE SCAN           | IDX_ID_MESS |  1000K|       |  2346   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
    1069093  consistent gets
     361'957  physical reads
    1000329  rows processed
```

x 9 less I/O

But before I said (& was lying ...?)

High CLUF: 3 + "469" physical I/O

Low CLUF: 4 + 19 physical I/O

~20 times less I/O

Short reminder: HIGH & LOW CLUF for first blocks

HIGH CLUF :

```
SQL> select * from TSCLUF where ID_MESS <= 485 ;

470 rows selected.

Elapsed: 00:00:00.08

-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |             |   486 |   127K|   490   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TSCLUF      |   486 |   127K|   490   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | IDX_ID_MESS |   486 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
        505  consistent gets
        472  physical reads
```

LOW CLUF :

```
SQL> select * from TSCLUF where id_sorted <= 485 ;

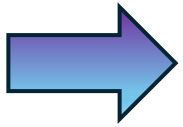
485 rows selected.

Elapsed: 00:00:00.02

-------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |               |   485 |   127K|    23   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| TSCLUF        |   485 |   127K|    23   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | IDX_ID_SORTED |   485 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
         88  consistent gets
         23  physical reads
```

High CLUF: 3 + “469” physical I/O
Low CLUF: 4 + 19 physical I/O



~20 times less I/O

Gotcha: am I lying ? *Yes & No*

Where ID_SORTED <= 485

And that means :

“get me the block with the lowest values of ID_SORTED” (1-485)

All such values of ID_SORTED (col0) occupy just 2 bytes

this is why the first block accomodated 26 rows

```
tab 0, row 0, @0x1e71
col  0: [ 2]  c1 02  --> value: 1 (integer)
col  1: [ 5]  c4 04 0e 58 36
col  2: [256]
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
(...)
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
58 58 58 58 58 58
```

First Row in Table

…
…
…

26 rows

```
end_of_block_dump
```

Where ID_SORTED <= 1000000

With ID_SORTED close to 1e6 the col0 will now occupy 5 bytes - so no chance for 26 rows in such block, rather closer to ~20

```
tab 0, row 0
col  0: [ 5]  c1 02 d7 0e 31  → value somewhere around 1e6
col  1: [ 4]  c4 04 0e 58
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
(...)
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
58 58 58 58 58 58
```

Somhere close to 1000000's row

…
…
…

Definitely not 26 rows, maybe 20 ?

```
end_of_block_dump
```

Note: no such anomalies with e.g. DATE type

Even closer to the reality: **Memory**

Comparison of: Memory utilized (buffer cache)

ID_SORTED (Low CLUF)

ID_MESS (High CLUF)

*Each time I have **flushed the buffer cache** in order to ensure the fresh baseline*

HIGH & LOW CLUF: comparison of Memory Utilized

Size on disk of all involved segments:

IDX_ID_SORTED : 192 MB
IDX_ID_MESS : 192 MB
TSCLUF : 3012 MB

```
SQL> alter system flush buffer_cache ;
SQL> select * from TSCLUF T where id_sorted <= 1e6 ;

1000000 rows selected.

-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |  1000K|   256M| 40883   (1)| 00:00:02 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TSCLUF        |  1000K|   256M| 40883   (1)| 00:00:02 |
|*  2 |   INDEX RANGE SCAN                  | IDX_ID_SORTED |  1000K|       |  2409   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------

From v$bh :

OWNER      OBJECT_NAME       OBJECT_TYPE          B_CACHE          MB_CACHE
---------- ----------------- -------------------- ---------------- ----------
TSTUS      IDX_ID_SORTED     INDEX                18,251,776       17
TSTUS      TSCLUF            TABLE                315,080,704      300
```

Low CLUF

```
SQL> alter system flush buffer_cache ;
SQL> select /*+index(T IDX_ID_MESS)*/ * from TSCLUF T where id_mess <= 1e6 ;

1000329 rows selected.

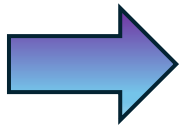
-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |             |  1000K|   256M|  1002K  (1)| 00:00:40 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TSCLUF      |  1000K|   256M|  1002K  (1)| 00:00:40 |
|*  2 |   INDEX RANGE SCAN                  | IDX_ID_MESS |  1000K|       |  2346   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------

From v$bh :

OWNER      OBJECT_NAME       OBJECT_TYPE          B_CACHE          MB_CACHE
---------- ----------------- -------------------- ---------------- ----------
TSTUS      IDX_ID_MESS       INDEX                18,276,352       17
TSTUS      TSCLUF            TABLE                2,946,875,392    2810
```

High CLUF

Low CLUF: 300 MB cached
High CLUF: 2810 MB cached !



~10 x more memory utilized
(& wasted?) in case of High CLUF

HIGH & LOW CLUF: comparison of Memory Utilized

Remember – we‘re performing I/Os „blockwise“ !

Low CLUF

```
SQL> alter system flush buffer_cache ;
SQL> select * from TSCLUF T where id_sorted <= 1e6 ;

1000000 rows selected.

From v$bh :

OWNER      OBJECT_NAME       OBJECT_TYPE          B_CACHE          MB_CACHE
---------- ----------------- -------------------- ---------------- ----------
TSTUS      IDX_ID_SORTED     INDEX                18,251,776       17
TSTUS      TSCLUF            TABLE                315,080,704      300
```

High CLUF

```
SQL> alter system flush buffer_cache ;
SQL> select /*+index(T IDX_ID_MESS)*/ * from TSCLUF T where id_mess <= 1e6 ;

1000329 rows selected.

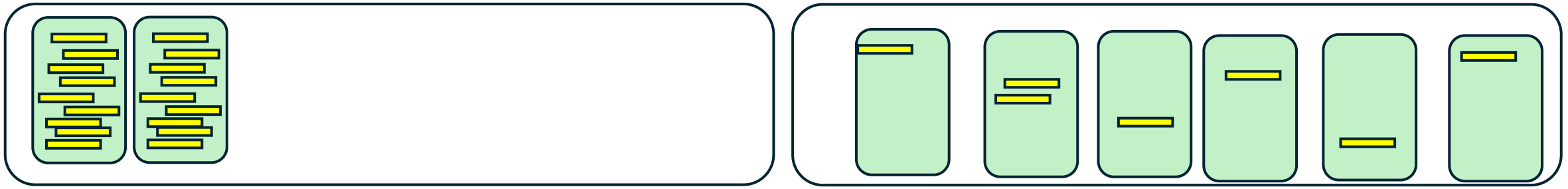
From v$bh :

OWNER      OBJECT_NAME       OBJECT_TYPE          B_CACHE          MB_CACHE
---------- ----------------- -------------------- ---------------- ----------
TSTUS      IDX_ID_MESS       INDEX                18,276,352       17
TSTUS      TSCLUF            TABLE                2,946,875,392    2810
```

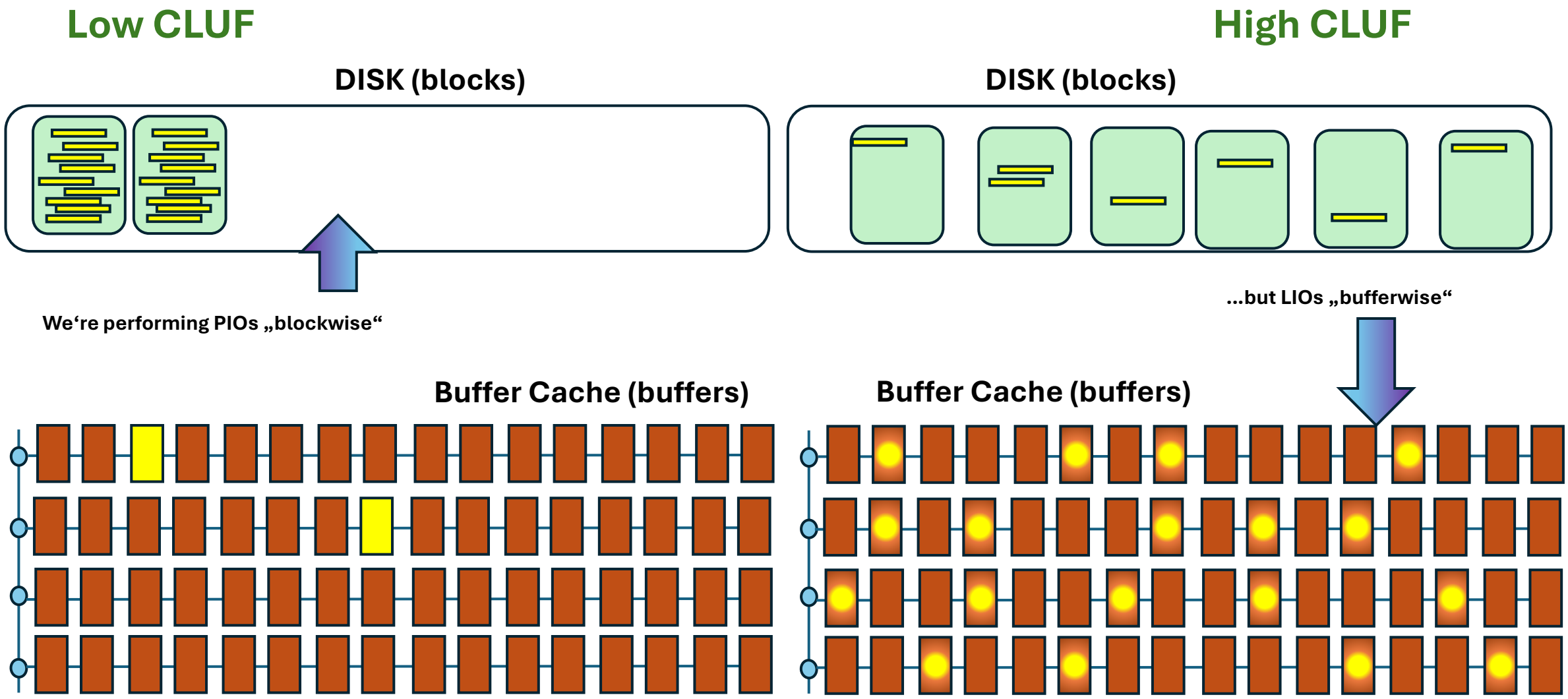
How are the rows with ID_SORTED & ID_MESS scattered on disk ?

logically and physically retrieved ~10% of the table

logically retrieved ~10%, but physically ~90% of the table



HIGH & LOW CLUF: comparison of Memory Utilized



Even closer to the reality: **CPU**

Comparison of: CPU cycles utilized

ID_SORTED (Low CLUF)

ID_MESS (High CLUF)

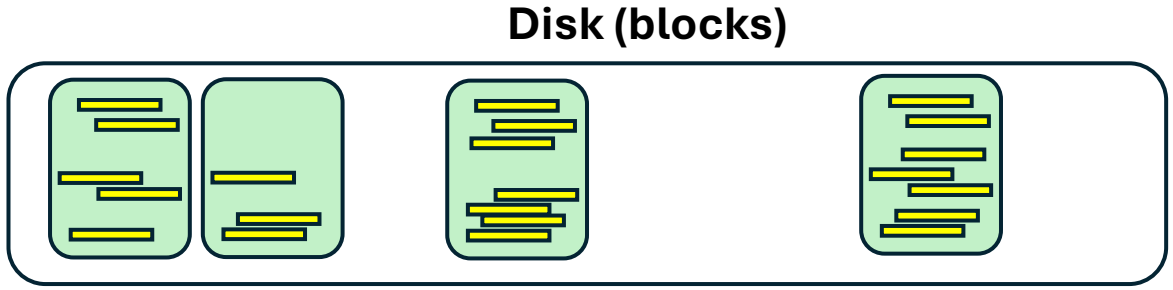
No ***flushing of the buffer cache*** *! ! !*

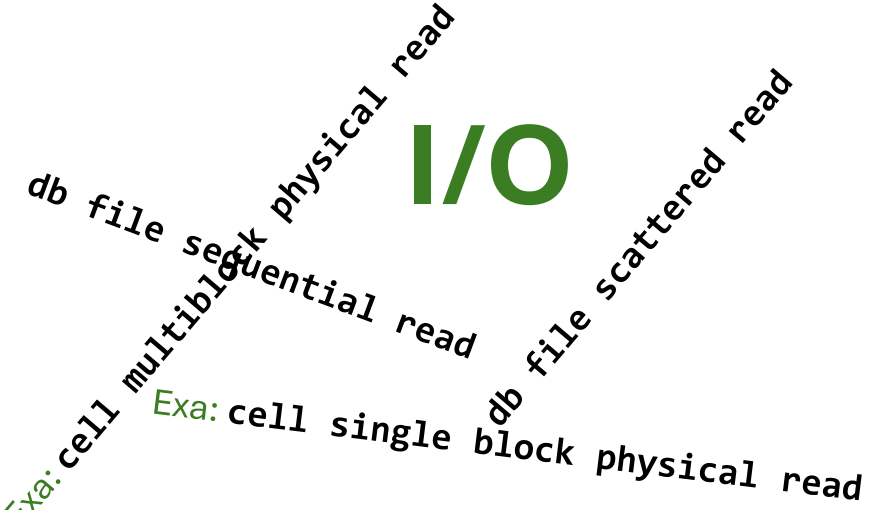
OK, but we have them in buffer cache.
So the second execution of this SQL will go fast!

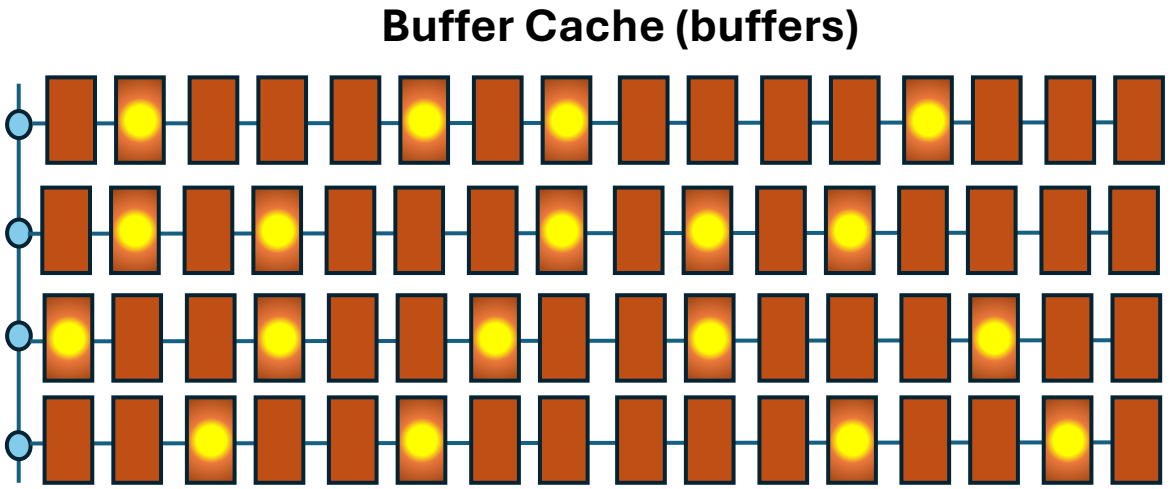
So, why bother ?

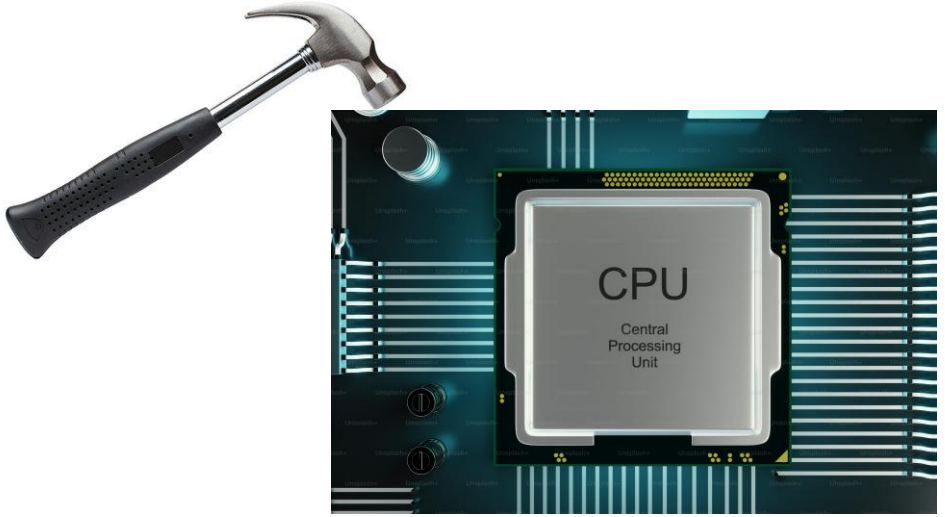
HIGH & LOW CLUF: comparison of Memory Utilized

Doesn‘t matter whether Low or High CLUF …









HIGH & LOW CLUF: comparison of CPU Utilized

Both execs are done with the previously populated cache + 10046 is set (will use tkprof)

```
SQL> select * from TSCLUF T where id_sorted <= 1e6 ;

1'000'000 rows selected.

Statistics
----------------------------------
    171'311  consistent gets
          0  physical reads → only LIOs (CPU)
tkprof :

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    66668      1.52       1.93          0     171311          0     1000000
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    66670      1.53       1.93          0     171311          0     1000000

Rows     Row Source Operation
-------  ---------------------------------------------------------------------
1000000  TABLE ACCESS BY INDEX ROWID TSCLUF (cr=171'311 time=876'584 us )
1000000   INDEX RANGE SCAN IDX_ID_SORTED    (cr=68'746  time=320'505 us )
```

Low CLUF

```
SQL> select /*+index(T IDX_ID_MESS)*/ * from TSCLUF T where id_mess <= 1e6 ;

1'000'329 rows selected.

Statistics
----------------------------------
  1'069'093  consistent gets
          0  physical reads → only LIOs (CPU)
tkprof :

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch    66690      2.84       3.47          0    1069093          0     1000329
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    66692      2.84       3.48          0    1069093          0     1000329


Rows     Row Source Operation
-------  ---------------------------------------------------------------------
1000329  TABLE ACCESS BY INDEX ROWID TSCLUF (cr=1'069'093 time=2'236'932 us )
1000329   INDEX RANGE SCAN IDX_ID_MESS     (cr=68'767    time=339'050  us  )
```

High CLUF

IDX_ID_SORTED with **time=320'505 us** which makes ~320 ms
TSCLUF with **time=876'584 us** performing **~171k LIOs** - which makes ~877 ms
---
That makes : 320 ms + 877 ms = 1.2 seconds

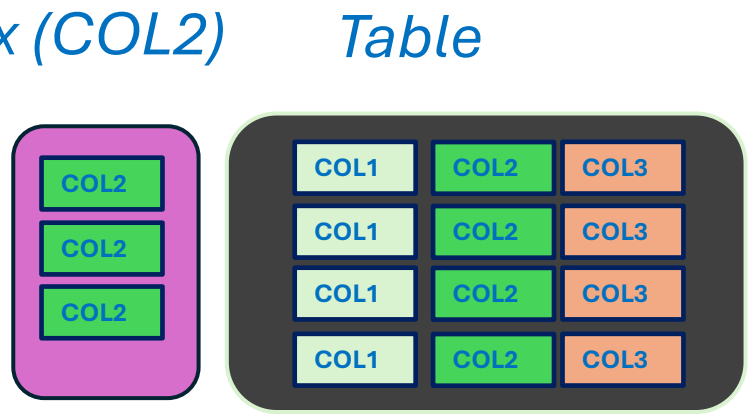
~2 x →

IDX_ID_MESS with **time=339'050 us**, which makes ~339 ms (→ no difference)
TSCLUF with **time=2'236'932 us**, performing **~1e6 LIOs** which makes ~2237 ms
---
That makes: 339 ms + 2237 ms = 2.5 seconds (!)

How to mitigate HIGH CLUF?

Extend the Index

```
SQL> Select COL1, COL2 From TABLE where COL2 > :B01
```



Consider: Create EXT_IDX on TABLE (COL2 , COL1)

→No need to visit TABLE with High CLUF

Recreate (dbms_redefinition?) TABLE with proper **ORDER BY**

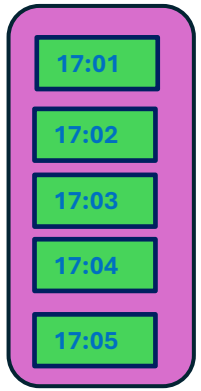
```
SQL> Select * From TABLE where MESSY_COL > :B01
```

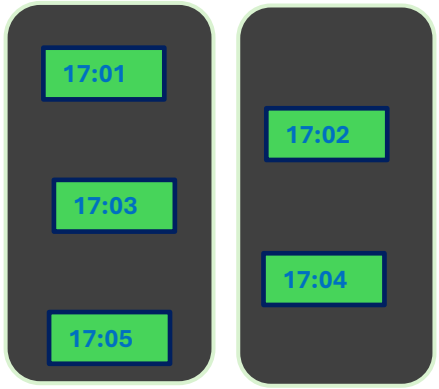
```
Create table NEW_TABLE as select * from TABLE Order by MESSY_COL
+ recreate the indexes
```

Be careful not to make things worse though – you may screw up other SQLs with other predicates (COLs) with up to now fairly decent CLUF !
So, if MESSY_COL is not (close to) unique, consider adding these COLs to ORDER BY
and possibly change the order of sorting

```
Create table NEW_TABLE as select * from TABLE Order by MESSY_COL, <other columns>
+ recreate the indexes
```

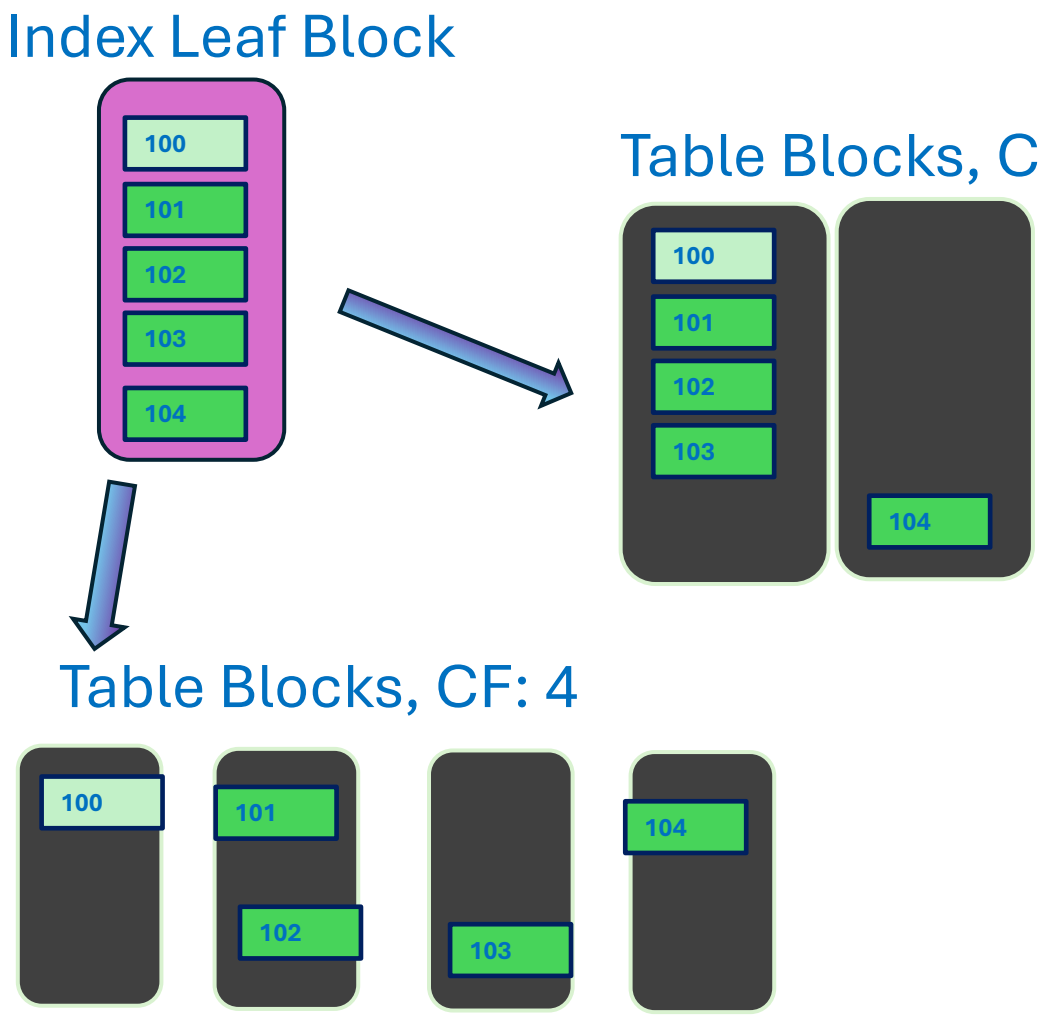
Consider increasing TABLE_CACHED_BLOCKS

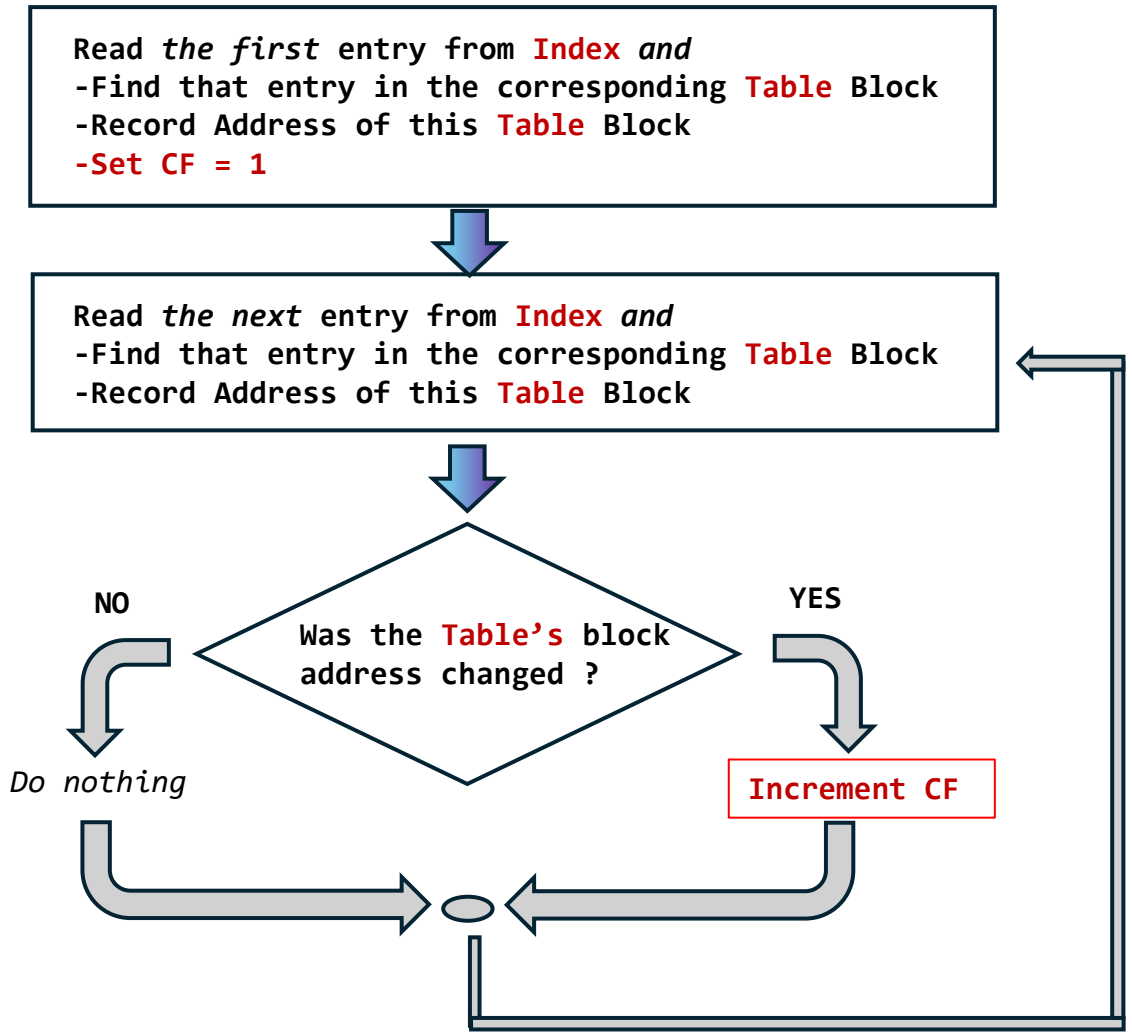




Does the High CLUF problem really exist
or it just *appears* as if it was there ..

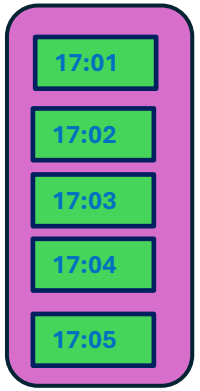
How is CLUF calculated while gathering Stats?





How is the CLUF calculated (**anomaly**)

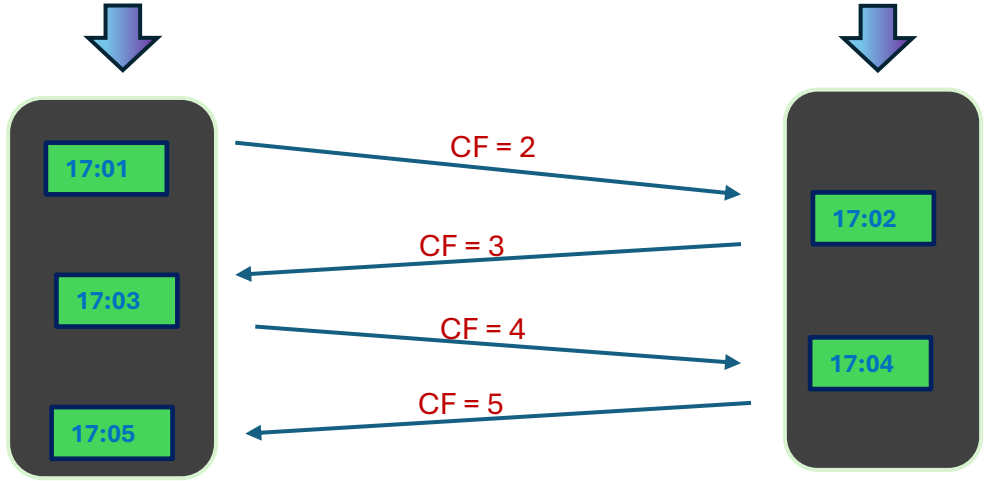
Index Leaf Block



Two concurrent sessions perfoming INSERT
(assume: with 2 seconds gap for each)

```
Session #1
SQL> Insert into TAB (dt) values ( SYSDATE )
```

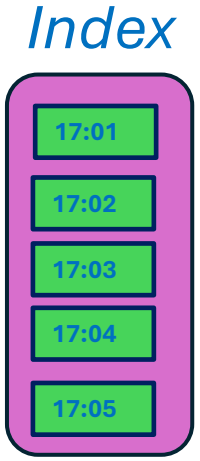
```
Session #2
SQL> Insert into TAB (dt) values ( SYSDATE )
```



Based on the algorithm
CLUF = 5
(!)
occupying just
2 Tbl Blocks ☹

*Table Blocks in **ASSM** managed tablespace*

How is the CLUF calculated with **TABLE_CACHED_BLOCKS > 1**

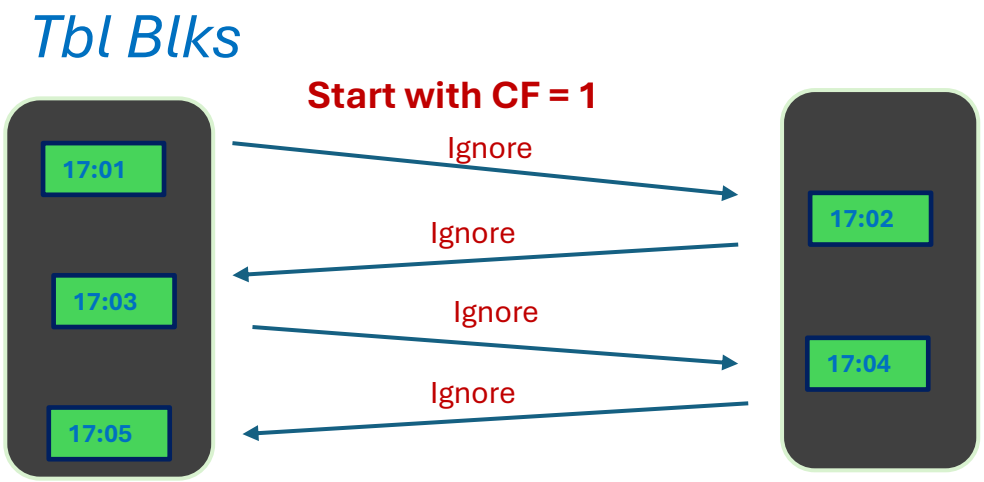


Imagine such SQL :

```
SQL> Select from TAB Where DT >= SYSDATE-1
```

Index would probably do GREAT
But due to the HIGH CLUF (but virtually non-existing problem) :

Full Table Scan (?)

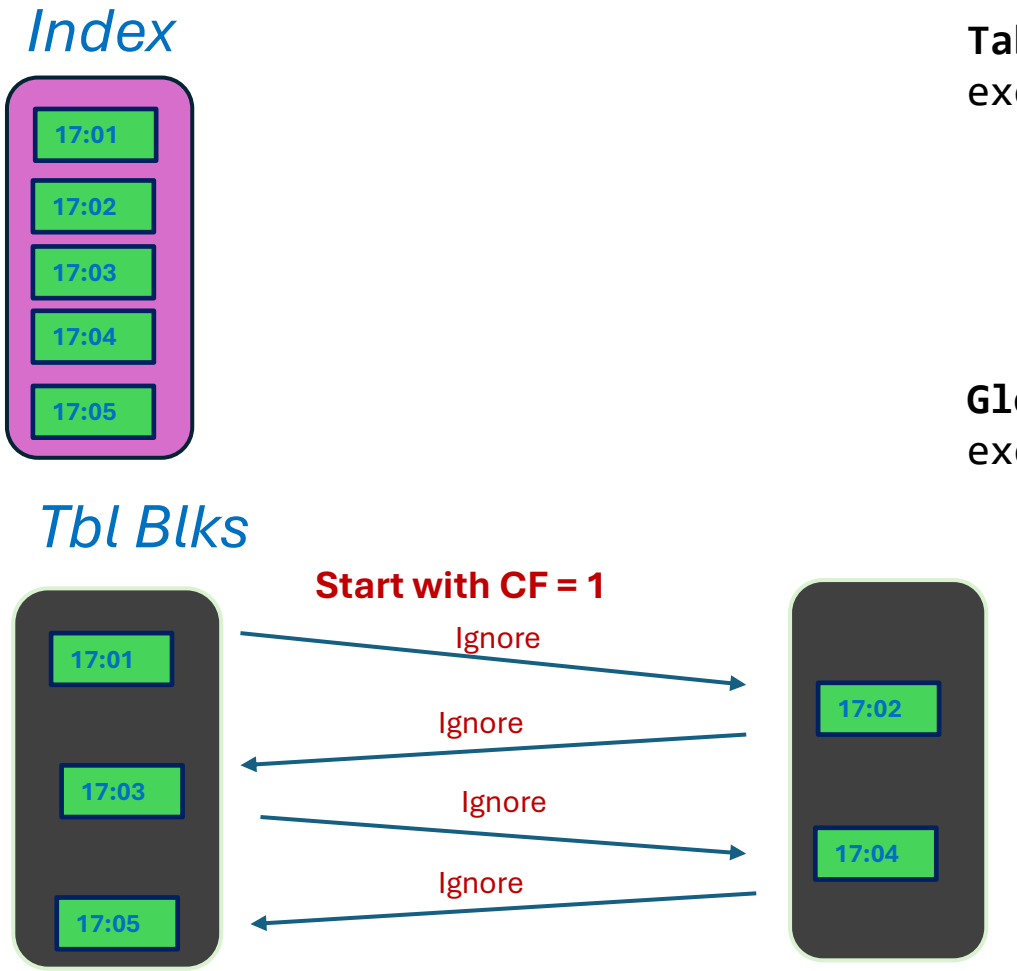


So, assume TABLE_CACHED_BLOCKS = 2
CLUF will be calculated to 1
which much closer the truth than CLUF = 5

How to find out whether that anomaly?

- Use DBMS_ROWID to examine number of blocks, rows/blk for your predicate and compare with total rows/blk
- Set autotrace (for physical reads, rows processed) and experiment with range scans applying that prediate
- Make yourself familiar with costing of Index Range Scans and the impact of CLUF on the final cost (10053)

How to set **TABLE_CACHED_BLOCKS > 1**




```
Table level :
exec dbms_stats.set_table_prefs(
        ownname => 'HR',
        tabname => 'EMPLOYEES',
        pname   => 'TABLE_CACHED_BLOCKS',
        pvalue  => 16);

Global level :
exec dbms_stats.set_global_prefs('TABLE_CACHED_BLOCKS',16);
```

Q & A

Gotcha: Why hint /*+ index */ ?

```
select * from TSTUS.TSCLUF T where id_sorted <= 1e6

1000000 rows selected.

Elapsed: 00:00:14.99
```

```
SQL> select /*+ index(T IDX_ID_MESS) */ * from TSTUS.TSCLUF T where id_mess <= 1e6

1000329 rows selected.

Elapsed: 00:00:33.10
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |  1000K|   256M| 40883   (1)|
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TESTCL        |  1000K|   256M| 40883   (1)|
|*  2 |   INDEX RANGE SCAN                  | IDX_ID_SORTED |  1000K|       |  2409   (1)|
-------------------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                           | Name        | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |             |  1000K|   256M|  1002K  (1)|
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TESTCL      |  1000K|   256M|  1002K  (1)|
|*  2 |   INDEX RANGE SCAN                  | IDX_ID_MESS |  1000K|       |  2346   (1)|
-------------------------------------------------------------------------------------------
```

```
SQL> select * from TSTUS.TSCLUF T where id_mess <= 1e6
```

```
-------------------------------------------------------------------------------------
| Id  | Operation                 | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |        |  1000K|   256M| 66890   (1)| 00:00:03 |
|*  1 |  TABLE ACCESS STORAGE FULL| TESTCL |  1000K|   256M| 66890   (1)| 00:00:03 |
-------------------------------------------------------------------------------------
```

Gotcha: Why hint /*+ index */ ?

```
***************************************
BASE STATISTICAL INFORMATION
***********************
Table Stats::
  Table: TESTCL  Alias: TESTCL
  #Rows: 10000000  SSZ: 0  LGR: 0  #Blks:  385478  AvgRowLen:  269.00
Index Stats::
  Index: IDX_ID_MESS  Col#: 2
  LVLS: 2  #LB: 23371  #DK: 6355456  LB/K: 1.00  DB/K: 1.00  CLUF: 9999973.00
KKEISFLG: 1
  Index: IDX_ID_SORTED  Col#: 1
  LVLS: 2  #LB: 24002  #DK: 9914368  LB/K: 1.00  DB/K: 1.00  CLUF: 384616.00
```

Cost of FTS :

```
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for TESTCL[TESTCL]
  SPD: Return code in qosdDSDirSetup: NOCTX, estType = TABLE

 kkecdn: Single Table Predicate:"TESTCL"."ID_SORTED"<=1e6
  Column (#1): ID_SORTED(NUMBER)
    AvgLen: 6 NDV: 9914368 Nulls: 0 Density: 0.000000 Min: 1.000000 Max: 10000000.000000
  Estimated selectivity: 0.100000 , col: #1
  Table: TESTCL  Alias: TESTCL
    Card: Original: 10000000.000000  Rounded: 1000000  Computed: 1000000.108637  Non
Adjusted: 1000000.108637
Access Path: TableScan
    Cost:  66885.604225  Resp: 66885.604225  Degree: 0
```

Cost of Index Range Scan :

```
****** Costing Index IDX_ID_SORTED
ix_sel: 0.100000  ix_sel_with_filters: 0.100000
    Cost: 40883.160481  Resp: 40883.160481  Degree: 1
  Best:: AccessPath: IndexRange
  Index: IDX_ID_SORTED
         Cost: 40883.160481  Degree: 1  Resp: 40883.160481  Card: 1000000.108637

Cost of Indexe RS (HCLUF)
 ****** Costing Index IDX_ID_MESS
  Access Path: index (RangeScan)
    Index: IDX_ID_MESS
    resc_io: 1002339.000000  resc_cpu: 7528097438
    ix_sel: 0.100000  ix_sel_with_filters: 0.100000
    Cost: 1002539.749265  Resp: 1002539.749265  Degree: 1
  Best:: AccessPath: IndexRange
  Index: IDX_ID_MESS
         Cost: 1002539.749265  Degree: 1  Resp: 1002539.749265  Card: 1000000.973451
Bytes: 0.000000
***************************************
```

Gotcha: Why hint /*+ index */ ?

```
Costing :
```

I/O Cost of Index Range Scan = (I/O cost of index access) **+** (I/O cost of table access based on this index)

cost of index access: Index_LVL + CEIL(#LB * ix_sel)
AND
cost of the table access: CLUF * ix_sel

```
LOW :
09:19:28 SYS@GCTEST1->PDB01> select ceil(24002 * 0.100000) from dual ;

CEIL(24002*0.100000)
--------------------
                2401

09:20:11 SYS@GCTEST1->PDB01> select 2 + ceil(24002 * 0.100000) from dual ;

2+CEIL(24002*0.100000)
----------------------
                  2403

09:20:31 SYS@GCTEST1->PDB01> select round( 384616.00 * 0.100000 ) from dual ;

ROUND(384616.00*0.100000)
-------------------------
                    38462

HIGH CLUF :

38462 + 2403
40865 --> Index Range Scan (COST calculated from 10053)

FTS
09:29:05 SYS@GCTEST1->PDB01> select round( 9999973.00 * 0.100000 ) from dual ;

ROUND(9999973.00*0.100000)
--------------------------
                    999997
999997 + 2403
1002400
```