Report

Laboratory Work 4

Dmitry Ladutsko

July 19, 2022

Task 1: Full Scans and the High-water Mark and Block reading

Step 1:

CREATE TABLE t2 AS

SELECT TRUNC(rownum / 100) id, rpad(rownum,100) t_pad

FROM dual

CONNECT BY rownum < 100000;

```
Table T2 created.
```

Picture 1.1 – Creation Result

Step 2:

CREATE INDEX t2_idx1 ON t2 (id);

```
Index T2_IDX1 created.
```

Picture 1.2 – Creation Result

Step 3:

select blocks from user_segments where segment_name = 'T2';

```
    BLOCKS
----------
       416
```

Picture 1.3 - Query Result

select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;

```
  BLOCK_CT
----------
       376
```

Picture 1.4 - Query Result

SET autotrace ON;

```
Autotrace Enabled
Shows the execution plan as well as statistics of the statement.
```

Picture 1.5 - Result

SELECT COUNT(*)

FROM t2 ;

	COUNT(*)
1	99999

Picture 1.6 – Quantity of Rows

```
Statistics
----------------------------------------------------------
          1  CPU used by this session
          1  CPU used when call started
         33  Requests to/from client
         33  SQL*Net roundtrips to/from client
          2  buffer is not pinned count
        521  bytes received via SQL*Net from client
      60491  bytes sent via SQL*Net to client
          3  calls to get snapshot scn: kcmgss
          6  calls to kcmgcs
        380  consistent gets
        380  consistent gets from cache
        380  consistent gets pin
        380  consistent gets pin (fastpath)
          1  enqueue releases
          1  enqueue requests
          2  execute count
   12451840  logical read bytes from cache
        376  no work - consistent read gets
         34  non-idle wait count
          2  opened cursors cumulative
          1  opened cursors current
          1  parse count (hard)
          2  parse count (total)
          1  recursive calls
        380  session logical reads
          1  sorts (memory)
       1581  sorts (rows)
        376  table scan blocks gotten
```

Picture 1.7 - Statistics

Step 4:

DELETE FROM t2;

```
99 999 rows deleted.


PLAN_TABLE_OUTPUT
------------------------------------------------------------------
SQL_ID: 7q7z96xbfz8x4, child number: 0 cannot be found


Statistics
----------------------------------------------------------
         41  CPU used by this session
         41  CPU used when call started
         46  DB time
          6  Requests to/from client
         51  Session total flash IO requests
    1671168  cell physical IO interconnect bytes
        408  consistent gets
          6  consistent gets examination
          6  consistent gets examination (fastpath)
        408  consistent gets from cache
        402  consistent gets pin
        402  consistent gets pin (fastpath)
     203281  db block gets
     203281  db block gets from cache
     102153  db block gets from cache (fastpath)
         41  enqueue releases
         43  enqueue requests
         24  messages sent
         73  non-idle wait count
```

Picture 1.8 – Deleting and Statistics

Step 5:

--1

select blocks from user_segments where segment_name = 'T2';

```
    BLOCKS
----------
       416


PLAN_TABLE_OUTPUT
------------------------------------------------------------
SQL_ID: 41qztbp206c3k, child number: 0 cannot be found


Statistics
------------------------------------------------------------
               1  CPU used by this session
               1  CPU used when call started
               6  Requests to/from client
             111  consistent gets
              42  consistent gets examination
              42  consistent gets examination (fastpath)
             111  consistent gets from cache
              69  consistent gets pin
              69  consistent gets pin (fastpath)
               6  non-idle wait count
               2  opened cursors cumulative
               1  opened cursors current
               1  pinned cursors current
             111  session logical reads
               7  user calls
```

Picture 1.9 – Count of Blocks

--2

select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;

```
Autotrace Enabled
Shows the execution plan as well as statistics of the statement.

  BLOCK_CT
----------
         0


PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
SQL_ID  1299p2xpxjfgc, child number 0
-------------------------------------
select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct
from t2

Plan hash value: 2057388731

-----------------------------------------------------
| Id  | Operation             | Name      | E-Rows |
-----------------------------------------------------
|   0 | SELECT STATEMENT      |           |        |
```

Picture 1.10 – Count of Used Blocks

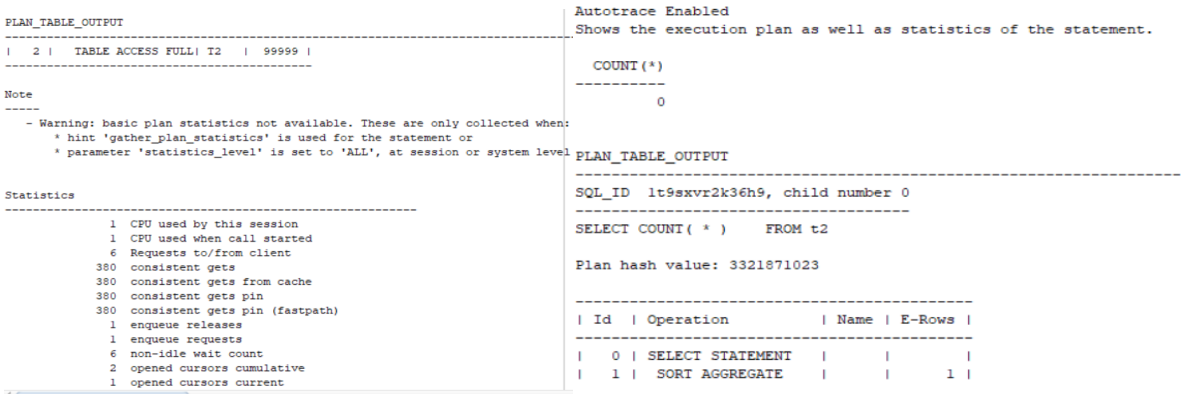
```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
|   1 |  SORT AGGREGATE      |           |      1 |
|   2 |   VIEW               | VM_NWVW_1 |  99999 |
|   3 |    HASH GROUP BY     |           |  99999 |
|   4 |     TABLE ACCESS FULL| T2        |  99999 |
-----------------------------------------------------

Note
-----
   - Warning: basic plan statistics not available. These are only collected when:
       * hint 'gather_plan_statistics' is used for the statement or
       * parameter 'statistics_level' is set to 'ALL', at session or system level

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------


Statistics
------------------------------------------------------------
               6  Requests to/from client
             380  consistent gets
             380  consistent gets from cache
             380  consistent gets pin
             380  consistent gets pin (fastpath)
               6  non-idle wait count
               2  opened cursors cumulative
               1  opened cursors current
               1  pinned cursors current
```

Picture 1.11 – Quantity of Consistent Gets

--3

SELECT COUNT(*)

FROM t2 ;



Picture 1.12 – Quantity of Consistent Gets and Count of Rows

Step 6:

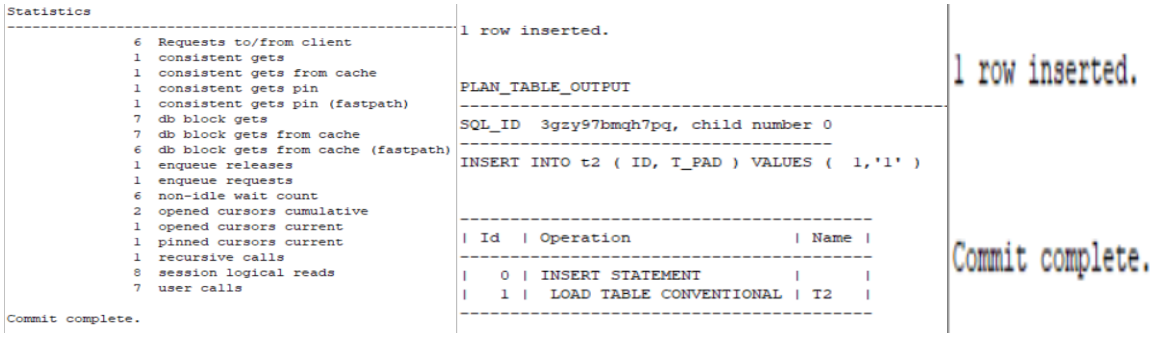
INSERT INTO t2

(ID, T_PAD)

VALUES

(1,'1');

COMMIT;



Picture 1.12 – Insertion

Step 7:

--1

SET autotrace ON;

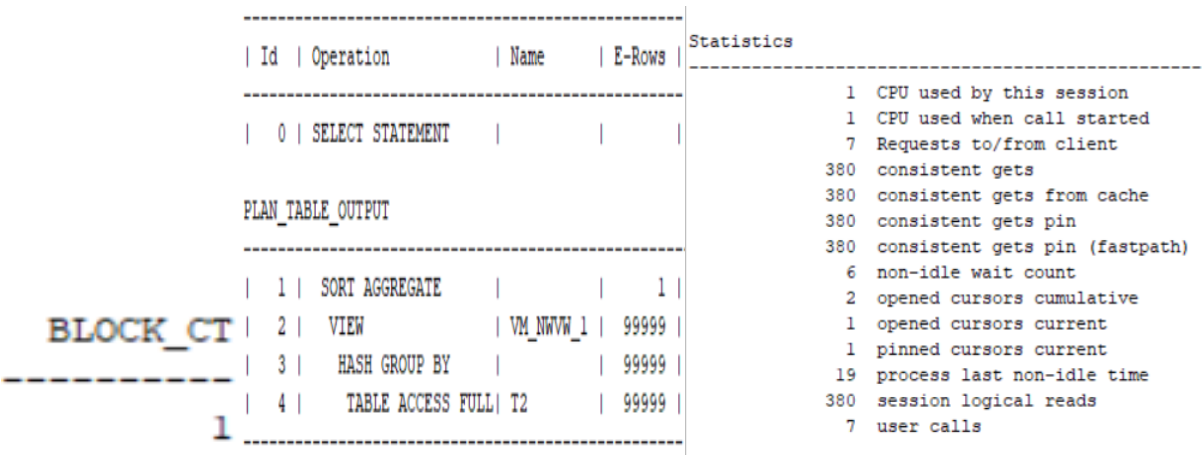
select blocks from user_segments where segment_name = 'T2';

Picture 1.13 – Count of Blocks

--2

SET autotrace ON;

select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;



```
----------------------------------------------
| Id  | Operation          | Name     | E-Rows |
----------------------------------------------
|   0 | SELECT STATEMENT   |          |        |

PLAN_TABLE_OUTPUT
----------------------------------------------
|   1 |  SORT AGGREGATE    |          |      1 |
|   2 |   VIEW             | VM_NWVW_1 |  99999 |
|   3 |    HASH GROUP BY   |          |  99999 |
|   4 |     TABLE ACCESS FULL| T2     |  99999 |
----------------------------------------------

BLOCK_CT
----------
         1

Statistics
----------------------------------------------------------
          1  CPU used by this session
          1  CPU used when call started
          7  Requests to/from client
        380  consistent gets
        380  consistent gets from cache
        380  consistent gets pin
        380  consistent gets pin (fastpath)
          6  non-idle wait count
          2  opened cursors cumulative
          1  opened cursors current
          1  pinned cursors current
         19  process last non-idle time
        380  session logical reads
          7  user calls
```

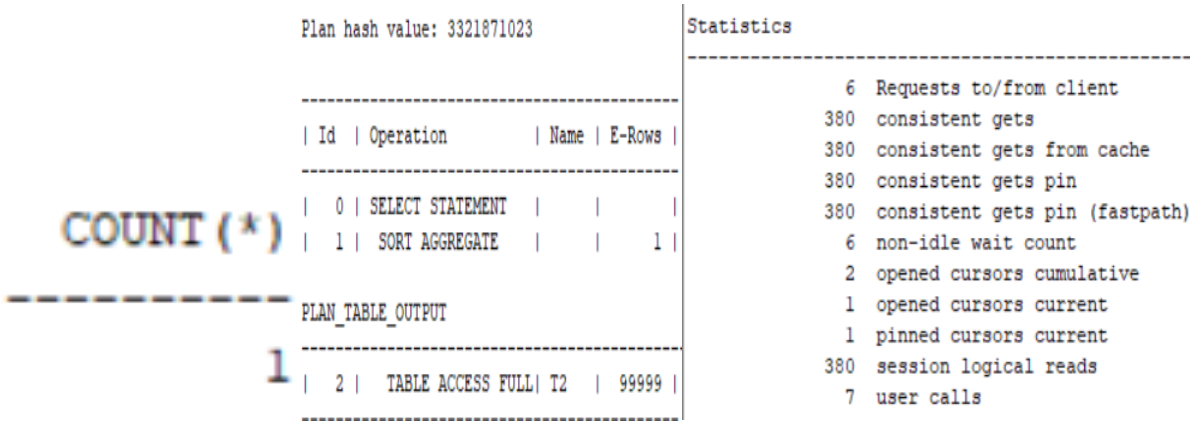
Picture 1.14 - Count of Used Blocks

--3

SET autotrace ON;

SELECT COUNT(*)

FROM t2 ;

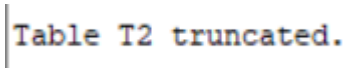


```
Plan hash value: 3321871023

----------------------------------------------
| Id  | Operation          | Name | E-Rows |
----------------------------------------------
|   0 | SELECT STATEMENT   |      |        |
|   1 |  SORT AGGREGATE    |      |      1 |

PLAN_TABLE_OUTPUT
----------------------------------------------
|   2 |   TABLE ACCESS FULL| T2   |  99999 |
----------------------------------------------

  COUNT(*)
----------
         1

Statistics
----------------------------------------------------------
          6  Requests to/from client
        380  consistent gets
        380  consistent gets from cache
        380  consistent gets pin
        380  consistent gets pin (fastpath)
          6  non-idle wait count
          2  opened cursors cumulative
          1  opened cursors current
          1  pinned cursors current
        380  session logical reads
          7  user calls
```

Picture 1.15 – Quantity of Consistent Gets and Count of Rows

Step 8:

TRUNCATE TABLE t2;



```
Table T2 truncated.
```

Step 9:

--1

SET autotrace ON;

select blocks from user_segments where segment_name = 'T2';

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------
SQL_ID: 41qztbp206c3k, child number: 0 cannot be found


Statistics
------------------------------------------------------------
             1  DB time
            33  Requests to/from client
            33  SQL*Net roundtrips to/from client
            65  buffer is not pinned count
             8  buffer is pinned count
           550  bytes received via SQL*Net from client
         60747  bytes sent via SQL*Net to client
             2  calls to get snapshot scn: kcmgss
             7  calls to kcmgcs
            14  cluster key scan block gets
            14  cluster key scans
           111  consistent gets
            42  consistent gets examination
            42  consistent gets examination (fastpath)
           111  consistent gets from cache
            69  consistent gets pin
            69  consistent gets pin (fastpath)
             1  cursor authentications
             2  execute count
            55  index fetch by key
             7  index range scans
       3637248  logical read bytes from cache
       3637248  logical read bytes from cache
            63  no work - consistent read gets
            33  non-idle wait count
             2  opened cursors cumulative
             1  opened cursors current
             2  parse count (total)
           111  session logical reads
             2  shared hash latch upgrades - no wait
             1  sorts (memory)
          1581  sorts (rows)
             8  table scan blocks gotten
            11  table scan disk non-IMC rows gotten
            11  table scan rows gotten
             2  table scans (short tables)
            34  user calls
```

```
BLOCKS
----------
         6
```

Picture 1.16 – Count of Blocks

--2

SET autotrace ON;

select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;

```
Statistics
----------------------------------------------------------
             1  CCursor + sql area evicted
             1  CPU used by this session
             1  CPU used when call started
             1  DB time
            33  Requests to/from client
            33  SQL*Net roundtrips to/from client
             2  buffer is not pinned count
           571  bytes received via SQL*Net from client
         60749  bytes sent via SQL*Net to client
             3  calls to get snapshot scn: kcmgss
             4  calls to kcmgcs
             4  consistent gets
             1  consistent gets examination
             1  consistent gets examination (fastpath)
             4  consistent gets from cache
             3  consistent gets pin
             3  consistent gets pin (fastpath)
             1  db block changes
             1  enqueue releases
             1  enqueue requests
             2  execute count
        131072  logical read bytes from cache
            33  non-idle wait count
             2  opened cursors cumulative
             1  opened cursors current
             1  parse count (hard)
             2  parse count (total)
             1  recursive calls
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------

SQL_ID  1299p2xpxjfgc, child number 0
-------------------------------------
select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct
from t2

Plan hash value: 2057388731

-------------------------------------------------------------
| Id  | Operation          | Name      | E-Rows |
-------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |        |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
|   1 |  SORT AGGREGATE    |           |      1 |
|   2 |   VIEW             | VM_NWVW_1 |  99999 |
|   3 |    HASH GROUP BY   |           |  99999 |
|   4 |     TABLE ACCESS FULL| T2      |  99999 |
-------------------------------------------------------------
```

```
BLOCK_CT
----------
         0
```

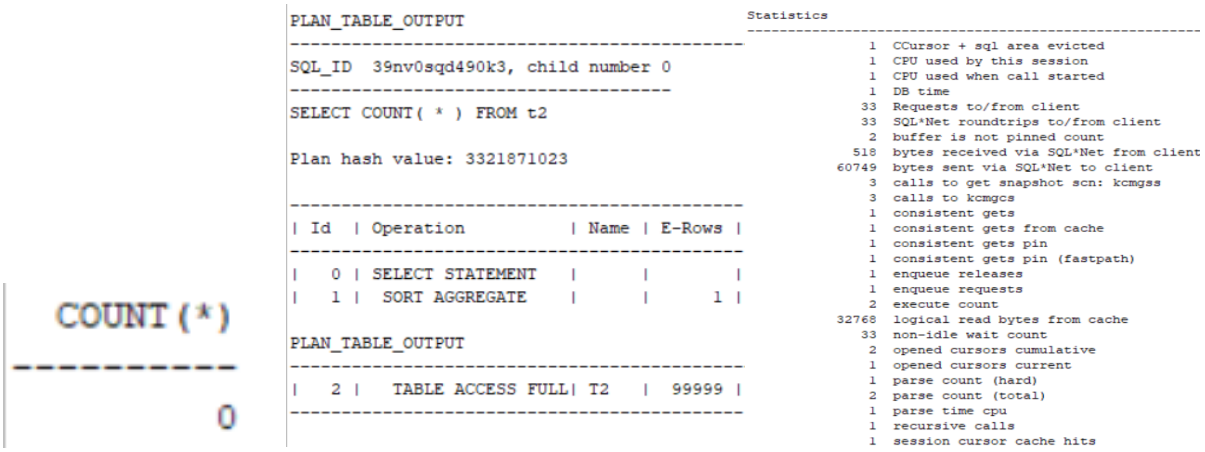
Picture 1.17 - Count of Used Blocks

--3

SET autotrace ON;

SELECT COUNT(*)

FROM t2 ;



```
  COUNT(*)
----------
         0
```

```
PLAN_TABLE_OUTPUT
-----------------------------------------
SQL_ID  39nv0sqd490k3, child number 0
-------------------------------------
SELECT COUNT( * ) FROM t2

Plan hash value: 3321871023

-----------------------------------------
| Id  | Operation          | Name | E-Rows |
-----------------------------------------
|   0 | SELECT STATEMENT   |      |        |
|   1 |  SORT AGGREGATE    |      |      1 |

PLAN_TABLE_OUTPUT
-----------------------------------------
|   2 |   TABLE ACCESS FULL| T2   |  99999 |
-----------------------------------------
```

```
Statistics
----------------------------------------------------------
          1  CCursor + sql area evicted
          1  CPU used by this session
          1  CPU used when call started
          1  DB time
         33  Requests to/from client
         33  SQL*Net roundtrips to/from client
          2  buffer is not pinned count
        518  bytes received via SQL*Net from client
      60749  bytes sent via SQL*Net to client
          3  calls to get snapshot scn: kcmgss
          3  calls to kcmgcs
          1  consistent gets
          1  consistent gets from cache
          1  consistent gets pin
          1  consistent gets pin (fastpath)
          1  enqueue releases
          1  enqueue requests
          2  execute count
      32768  logical read bytes from cache
         33  non-idle wait count
          2  opened cursors cumulative
          1  opened cursors current
          1  parse count (hard)
          2  parse count (total)
          1  parse time cpu
          1  recursive calls
          1  session cursor cache hits
```

Picture 1.18 – Quantity of Consistent Gets and Count of Rows

Task Result:

Summary table with all result and text description of analyses these results.

№	Count of Blocks	Count of Used Blocks	Count of Rows	Consistent gets	Description
1	416	376	99999	380	Table was filled into 416 blocks, 376 of which were used (40 unused). 380 consistent get operations. Maximal number of rows - 99999
2	416	0	0	380	We used DELETE operation but still have an opportunity to roll back deleted data. DELETE just saves allocated space and stats, so the number of consistent gets did not change after as you can see below 380 -> 380
3	416	1	1	380	Even if we INSERT 1 row (or 2,3,4 … 99999)
4	6	0	0	1	We used TRANCATE operator which truncate all the allocated data and stats (w/o an opportunity to roll back data), but oracle still leave 6 blocks to store an information about table

Task 2: Index Clustering factor parameter

Expected:

- Screenshot of the step 5;
- Description of the parameter clustering factor;
- Explanation: why for indexes *t1_idx1* and *t2_idx1* we have different values ;
- Which Index has best selective performance in execution Select clause filtered by IN (, list of values,);

Step 1*:*

```
CREATE TABLE t2 AS
 SELECT TRUNC( rownum / 100 ) id, rpad( rownum,100 ) t_pad
  FROM dual
 CONNECT BY rownum < 100000;
CREATE INDEX t2_idx1 ON t2
 ( id );
```

```
Table T2 created.

Index T2_IDX1 created.
```

Picture 2.1 – Table and Index creating

Step 2*:*

```
#  CREATE TABLE t1 AS
 SELECT MOD( rownum, 100 ) id, rpad( rownum,100 ) t_pad
  FROM dual
 CONNECT BY rownum < 100000;
```

```
Table T1 created.
```

Step 3*:*

```
# CREATE INDEX t1_idx1 ON t1
 ( id );
```

```
Index T1_IDX1 created.
```

Step 4*:*

```
# EXEC dbms_stats.gather_table_stats(USER,'t1',method_opt=>'FOR ALL COLUMNS SIZE
1',CASCADE=>TRUE );

#  EXEC dbms_stats.gather_table_stats( USER,'t2',method_opt=>'FOR ALL COLUMNS SIZE
1',CASCADE=>TRUE );
```

```
PL/SQL procedure successfully completed.


PL/SQL procedure successfully completed.
```

Picture 2.2 – Procedure execution

Step 5:

```
#  SELECT t.table_name||'.'||i.index_name idx_name,
    i.clustering_factor,
    t.blocks,
    t.num_rows
  FROM user_indexes i, user_tables t
 WHERE i.table_name = t.table_name
  AND t.table_name  IN( 'T1','T2' );
```

IDX_NAME	CLUSTERING_FACTOR	BLOCKS	NUM_ROWS
T1.T1_IDX1	37200	381	99999
T2.T2_IDX1	378	386	99999

Task Results:

- The clustering factor indicates the amount of order of the rows in the table based on the values of the index. At the screenshot above we can see that clustering factor **approaches the number of blocks** in the table 378 ~ 386. That means that the rows are ordered. If it approaches the number of rows in the table, the rows are randomly ordered. In such a case (clustering factor near the number of rows), it is unlikely that index entries in the same leaf block will point to rows in the same data blocks. If all of the index entries in a given leaf block point to different blocks in the table – the table is **not well ordered** with respect to this index. If we want index being **strong** clustered – we should consider using Index Organized Tables (**IOT**). They force the rows into a specific **physical location** based on their index entry. Even though the correct set up of clustering anyway increases performance and decreases cost.
- Because we used MOD operand in creation of table t1 instead of TRUNC in t2.
- Clustering factor increments if rawid points to different table blocks.

Task 3: Index Unique Scan

Step 1:

 # CREATE UNIQUE INDEX udx_t1 ON t1(t_pad);

```
INDEX UDX_T1 created.
```

Step 2

 # SELECT t1. FROM t1 where t1.t_pad = '1';*

```
no rows selected

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
SQL_ID  0bqrmw3lbhxhg, child number 0
-------------------------------------
SELECT t1.*  FROM t1 where t1.t_pad = '1'

Plan hash value: 1698228356

--------------------------------------------------------------------------------
| Id  | Operation                   | Name  | E-Rows |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |       |        |
|   1 |  TABLE ACCESS BY INDEX ROWID| T1    |      1 |

Statistics
----------------------------------------------------------
          1  CPU used by this session
          1  CPU used when call started
          1  DB time
         33  Requests to/from client
         33  SQL*Net roundtrips to/from client
          1  buffer is not pinned count
        533  bytes received via SQL*Net from client
      60509  bytes sent via SQL*Net to client
          2  calls to get snapshot scn: kcmgss
          2  calls to kcmgcs
          2  consistent gets
          2  consistent gets examination
          2  consistent gets examination (fastpath)
          2  consistent gets from cache
          2  execute count
          1  index fetch by key
```

Picture 3.1 - Query Result

Note. Select in 3 task is empty because ‘1’ is a char, but a column type is number.

Task 4: Index Range Scan

Step 1:

 # SELECT t2. FROM t2 where t2.id = '1';*

ID	T_PAD
1	100
1	101
1	102
1	103
1	104
1	105
1	106
1	107
1	108
1	109
1	110

Picture 4.1 - Query Result

Task 5: Index Skip Scan

I found scott schema in the internet. Separate file named 'scottschema' will be attached.

Step 1:

CREATE TABLE employees AS

SELECT *

FROM scott.emp;

```
Table EMPLOYEES created.
```

Step 2:

CREATE INDEX idx_emp01 ON employees

(empno, ename, job);

```
Index IDX_EMP01 created.
```

Step 3:

SELECT /*+INDEX_SS(emp idx_emp01)*/ emp.* FROM employees emp where ename = 'SCOTT';

SELECT /*+FULL*/ emp.* FROM employees emp WHERE ename = 'SCOTT'

```
Plan hash value: 1748371071

-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |           |      1 |        |      0 |00:00:00.01 |       1 |

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------------
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMPLOYEES |      1 |      1 |      0 |00:00:00.01 |       1 |
|*  2 |   INDEX SKIP SCAN                   | IDX_EMP01 |      1 |      1 |      0 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------------------

Plan hash value: 1748371071

-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |           |      1 |        |      0 |00:00:00.01 |       1 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMPLOYEES |      1 |      1 |      0 |00:00:00.01 |       1 |

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------------
|*  2 |   INDEX SKIP SCAN                   | IDX_EMP01 |      1 |      1 |      0 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------------------
```

Note. Oracle scans the index sub-trees for each of the possible values from leading column.

	Count of Blocks	Count of Used Blocks	Count of Rows	Consistent gets	Description
1	6	1	1	2	Skip scan because optimizer decided it is more sutable for this SELECT
	6	1	1	2	`/*+INDEX_SS(emp idx_emp01)*/ (cause of hint)`
	6	1	1	2	`/*+FULL*/ (cause of hint)`

SELECT BLOCKS FROM user_segments WHERE segment_name = 'EMPLOYEES';

	BLOCKS
1	6

SELECT COUNT(DISTINCT (dbms_rowid.rowid_block_number(rowid))) block_ct FROM EMPLOYEES ;

	BLOCK_CT
1	1

	BLOCK_CT
1	1

```
         5  calls to get snapshot scn: kcmgss
         2  calls to kcmgcs
         2  consistent gets
         2  consistent gets from cache
         2  consistent gets pin
         2  consistent gets pin (fastpath)
         1  enqueue releases
         1  enqueue requests
         2  execute count
     65536  logical read bytes from cache
         2  no work - consistent read gets
        43  non-idle wait count
         2  opened cursors cumulative
         1  opened cursors current
         1  parse count (hard)
         2  parse count (total)
         1  parse time elapsed
```

Review

Laboratory Work Summary:

In this laboratory work we have learned and practiced about how Oracle uses different techniques to access the data needed for the query answer. Access Methods can be divided into two categories: data accessed via a table scan or index access. There are some description below about what methods have we used to access the data at this lab:

Full Scan:

A full table scan reads all rows from a table, and then filters out those rows that do not meet the selection criteria.

In a full table scan, the database sequentially reads every formatted block under the high water mark. The database reads each block only once.

Clustering factor:

The clustering factor is a number that represents the degree to which data is randomly distributed in a table as compared to the indexed column. In simple terms, it is the number of "block switches" while reading a table using an index.

Index Unique Scan:

This scan returns, at most, a single rowid. Oracle performs a unique scan if a statement contains a UNIQUE or a PRIMARY KEY constraint that guarantees that only a single row is accessed.

Index Range Scan:

An index range scan is a common operation for accessing selective data. It can be bounded (bounded on both sides) or unbounded (on one or both sides). Data is returned in the ascending order of index columns. Multiple rows with identical values are sorted in ascending order by rowid.

Index Skip Scan:

Index skip scans improve index scans by non-prefix columns. Often, scanning index blocks is faster than scanning table data blocks. Skip scanning lets a composite index be split logically into smaller sub-indexes. In skip scanning, the initial column of the composite index is not specified in the query. In other words, it is skipped. The number of logical sub-indexes is determined by the number of distinct values in the initial column. Skip scanning is advantageous if there are few different values in the leading column of the composite index and many distinct values in the non-leading key of the index.