U1M4.LW.Access and Join Methods Part 1

Report

Mastykina Elizaweta

1. Table access full scan

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

Creation of the table t2:

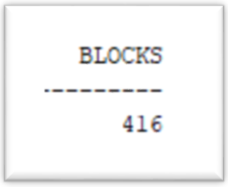
```
CREATE TABLE t2 AS
 SELECT TRUNC( rownum / 100 ) id, rpad( rownum,100 ) t_pad
   FROM dual
  CONNECT BY rownum < 100000;
```

Creation of the index:

```
CREATE INDEX t2_idx1 ON t2
  ( id );
```

Allocated Blocks Count:

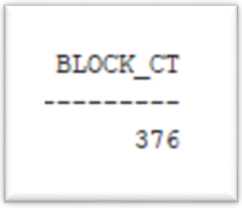
```
  SELECT BLOCKS FROM user_segments WHERE segment_name =
'T2';
```



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

Used Blocks Count:

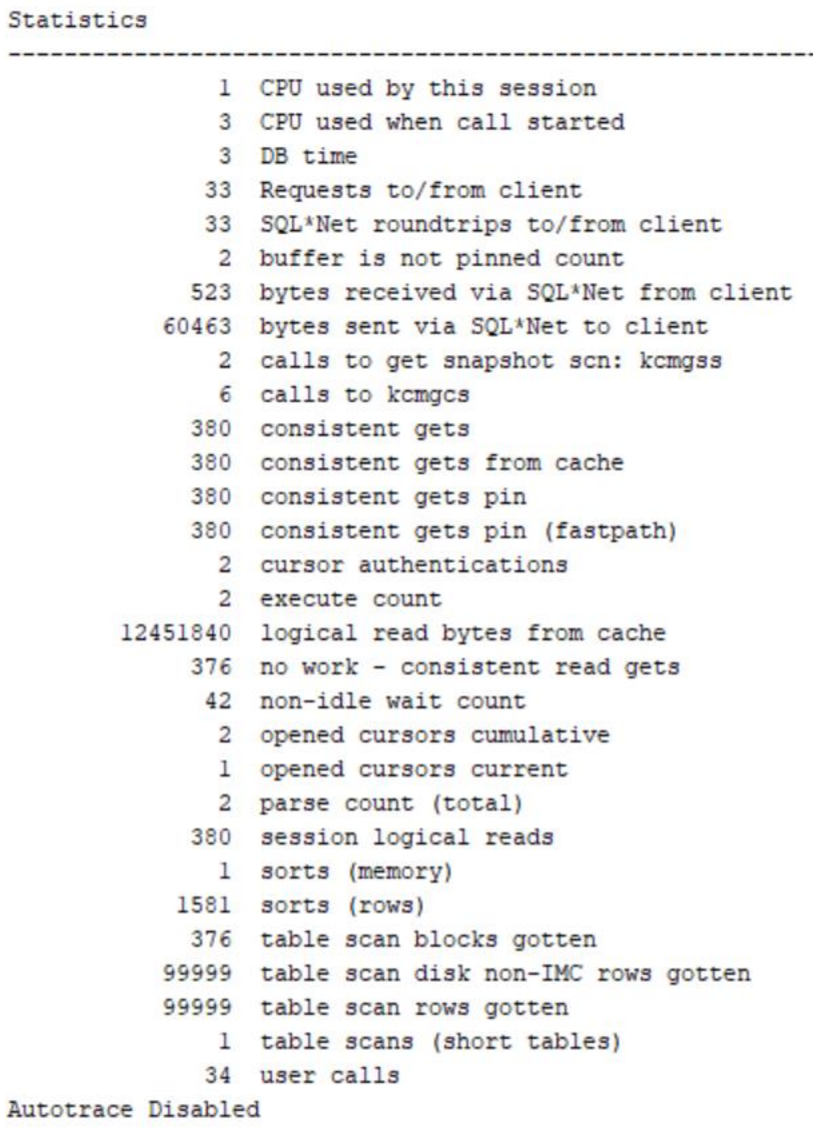
```
  SELECT COUNT(Distinct
(dbms_rowid.rowid_block_number(rowid))) block_ct FROM t2 ;
```



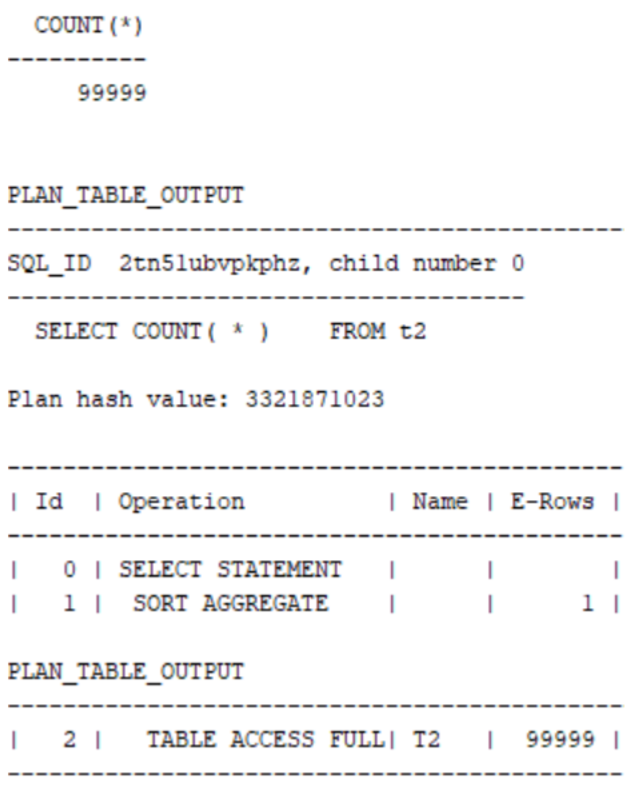
```
  BLOCK_CT
----------
       376
```

Execution of the full table scan:

```
SET autotrace ON;

SELECT COUNT( * )
 FROM t2 ;

SET autotrace OFF;
```



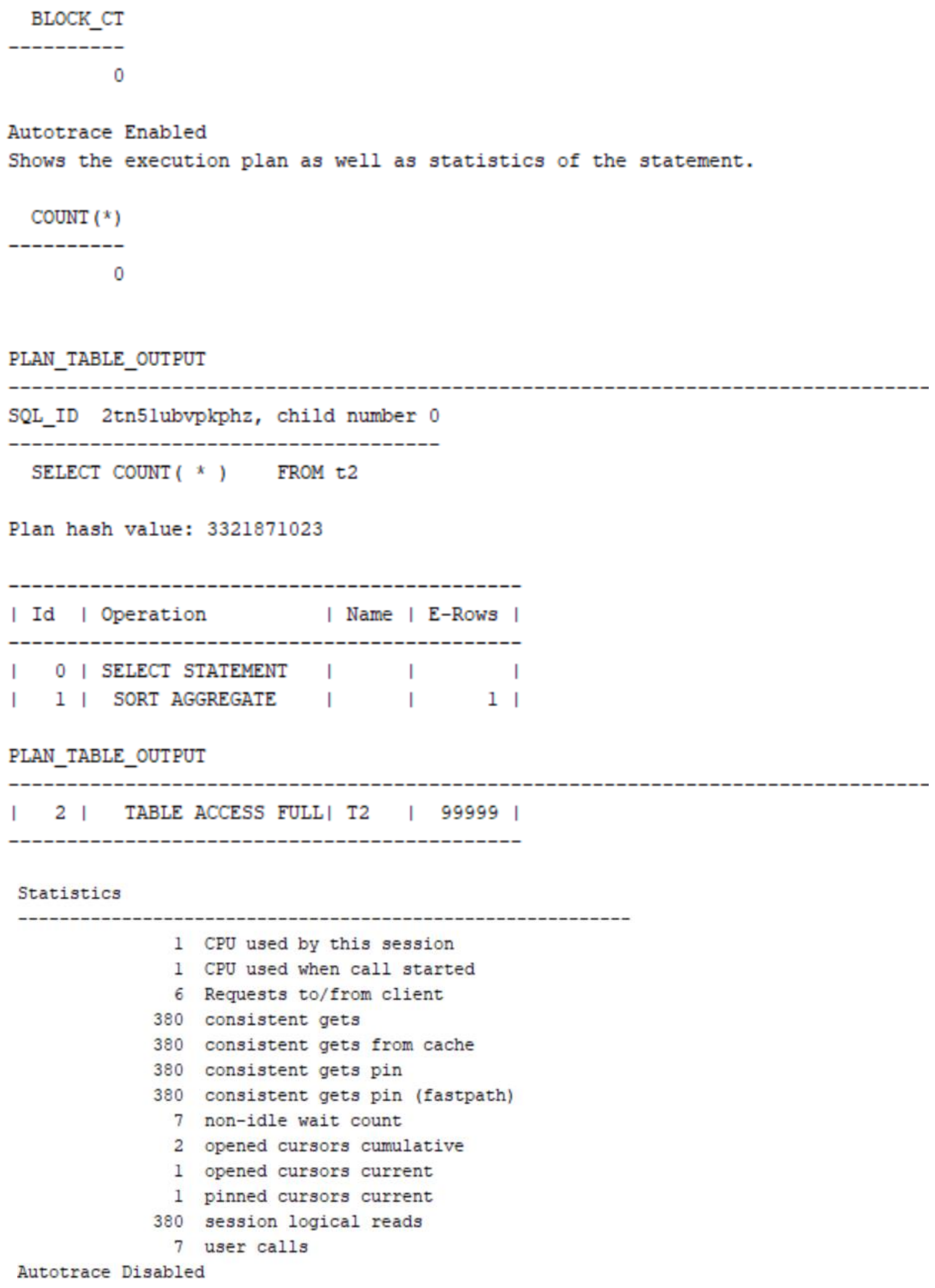
```
Statistics
-----------------------------------------------------------
               1  CPU used by this session
               3  CPU used when call started
               3  DB time
              33  Requests to/from client
              33  SQL*Net roundtrips to/from client
               2  buffer is not pinned count
             523  bytes received via SQL*Net from client
           60463  bytes sent via SQL*Net to client
               2  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)
               2  cursor authentications
               2  execute count
        12451840  logical read bytes from cache
             376  no work - consistent read gets
              42  non-idle wait count
               2  opened cursors cumulative
               1  opened cursors current
               2  parse count (total)
             380  session logical reads
               1  sorts (memory)
            1581  sorts (rows)
             376  table scan blocks gotten
           99999  table scan disk non-IMC rows gotten
           99999  table scan rows gotten
               1  table scans (short tables)
              34  user calls
Autotrace Disabled
```



```
  COUNT(*)
----------
     99999


PLAN_TABLE_OUTPUT
---------------------------------------------
SQL_ID  2tn51ubvpkphz, 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 |
---------------------------------------------
```

Delete All Rows from table and repeat previous steps:

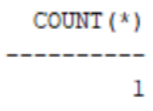
```
 DELETE FROM t2;

 SELECT COUNT(Distinct
(dbms_rowid.rowid_block_number(rowid))) block_ct FROM t2 ;

 SELECT autotrace ON;

 SELECT COUNT( * )
  FROM t2 ;

 SET autotrace OFF;
```



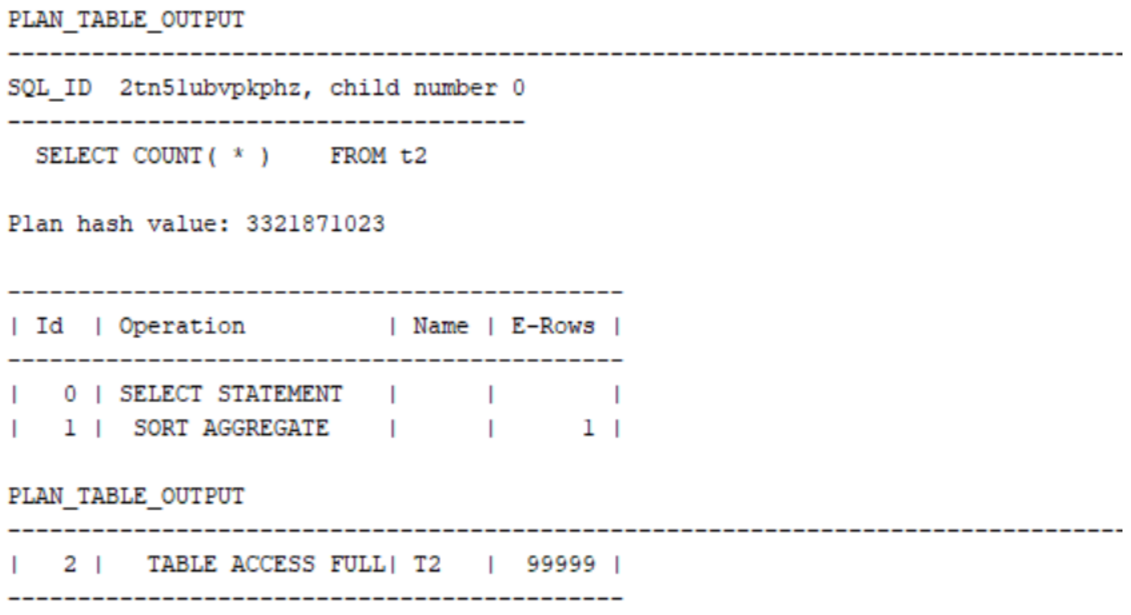
```
  BLOCK_CT
----------
         0

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

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


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
SQL_ID  2tn51ubvpkphz, 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  CPU used by this session
               1  CPU used when call started
               6  Requests to/from client
             380  consistent gets
             380  consistent gets from cache
             380  consistent gets pin
             380  consistent gets pin (fastpath)
               7  non-idle wait count
               2  opened cursors cumulative
               1  opened cursors current
               1  pinned cursors current
             380  session logical reads
               7  user calls
 Autotrace Disabled
```

Insert 1 row and execute a full table scan and noted the consistent gets (logical block reads):

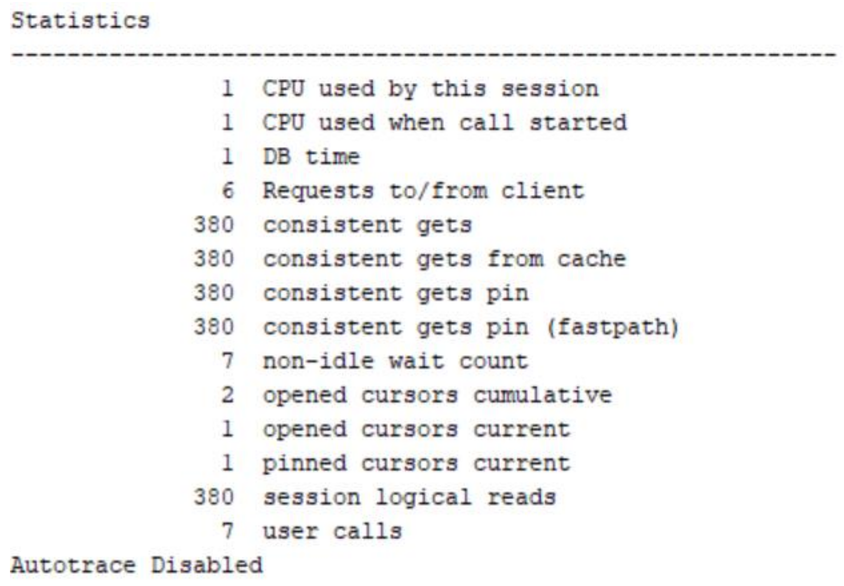
```
INSERT INTO t2
  ( ID, T_PAD )
  VALUES
  (  1,'1' );

COMMIT;


  SELECT COUNT(Distinct
(dbms_rowid.rowid_block_number(rowid))) block_ct FROM t2 ;

  SET autotrace ON;

  SELECT COUNT( * )
   FROM t2 ;

  SET autotrace OFF;
```



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



```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------
SQL_ID  2tn51ubvpkphz, 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  CPU used by this session
              1  CPU used when call started
              1  DB time
              6  Requests to/from client
            380  consistent gets
            380  consistent gets from cache
            380  consistent gets pin
            380  consistent gets pin (fastpath)
              7  non-idle wait count
              2  opened cursors cumulative
              1  opened cursors current
              1  pinned cursors current
            380  session logical reads
              7  user calls
Autotrace Disabled
```

Truncate Table and execute a full table scan and noted the consistent gets (logical block reads):

```
TRUNCATE TABLE t2;
SELECT COUNT(Distinct
(dbms_rowid.rowid_block_number(rowid))) block_ct FROM t2 ;

  SET autotrace ON;

  SELECT COUNT( * )
   FROM t2 ;

  SET autotrace OFF;
```

```
Table T2 truncated.


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

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

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


PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
SQL_ID  2tn51ubvpkphz, 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  DB time
               6  Requests to/from client
               1  consistent gets
               1  consistent gets from cache
               1  consistent gets pin
               1  consistent gets pin (fastpath)
               1  enqueue releases
               1  enqueue requests
               7  non-idle wait count
               2  opened cursors cumulative
               1  opened cursors current
               1  pinned cursors current
               1  recursive calls
               1  session logical reads
               7  user calls
Autotrace Disabled
```

№	Count of Blocks	Count of Used Blocks	Count of Rows	Consistent gets	Description
1-3 (After creation)	416	376	99999	380	The HVM remains the same, all blocks up to the HWM are read in and scanned, even if they are empty
4-5 (After deletion)	416	0	0	380	
6-7 (After insertion)	416	1	1	380	
8-9 (After truncation)	6	0	0	1	The space was deallocated and the HWM was reset

This access method implies

iterate over all rows of the table, excluding those that do not satisfy

where predicate. It is applied either when

predicate conditions are absent in the index, or when the index does not exist.

2. Index Scan types

2.1. Task 2: Index Clustering factor parameter

Created table t2:

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

Created index:

```
CREATE INDEX t2_idx1 ON t2
 ( id );
```

Created table t1:

```
CREATE TABLE t1 AS

SELECT MOD( rownum, 100 ) id, rpad( rownum,100 ) t_pad

 FROM dual

CONNECT BY rownum < 100000;
```

Created index:

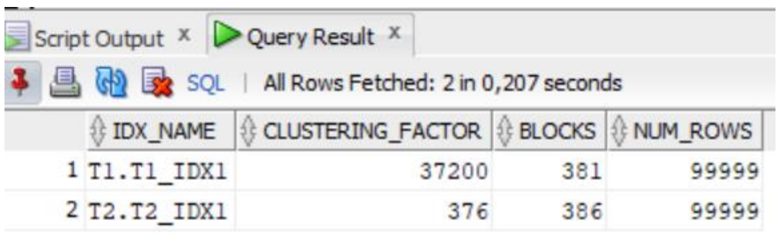
```
    CREATE INDEX t1_idx1 ON t1( id );
```

Calculate statistic for both tables:

```
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 );
```

Computed Index Clustering factor:

```
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' );
```



All Rows Fetched: 2 in 0,207 seconds

	IDX_NAME	CLUSTERING_FACTOR	BLOCKS	NUM_ROWS
1	T1.T1_IDX1	37200	381	99999
2	T2.T2_IDX1	376	386	99999

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.

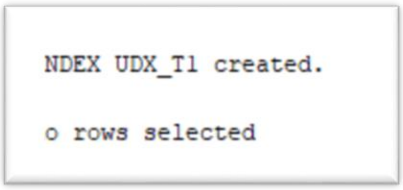
The table rows are synchronized with the index when the clustering factor is close to the number of data blocks and the column value is not row-ordered when the clustering factor approaches the number of rows in the table.

We have different clustering for T1 and T2 tables because they have different structure of column ID. Our index is based on column ID and we can see that in T2 rows with the same id are next to each other, when in T1 they are distributed.

2.2. Task 3: Index Unique Scan

Created index:

```
CREATE UNIQUE INDEX udx_t1 ON t1( t_pad );
SELECT t1.* FROM t1 where t1.t_pad = '1';
```



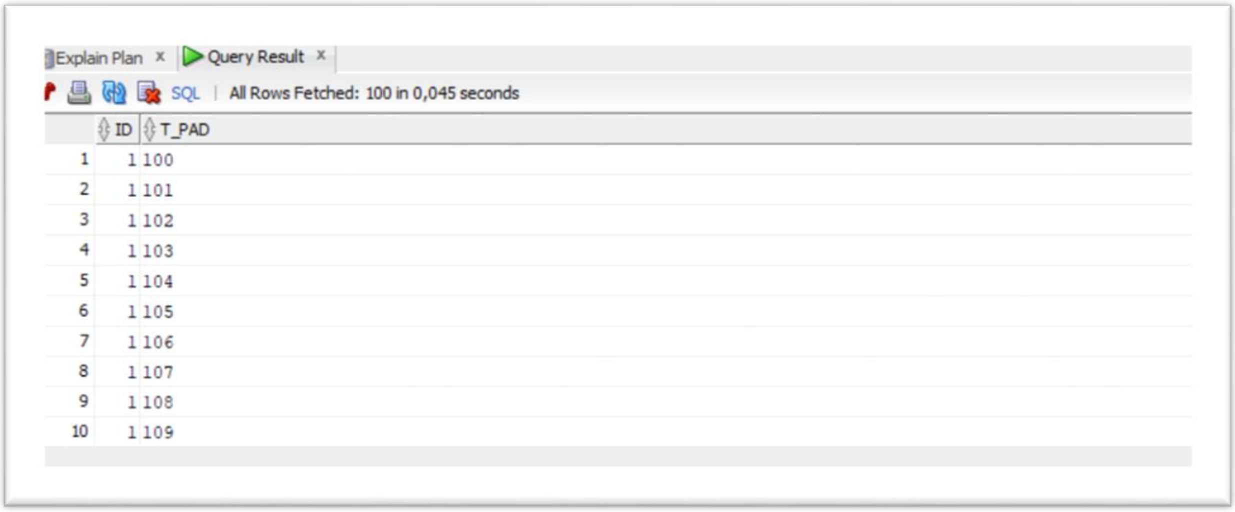
```
NDEX UDX_T1 created.

o rows selected
```

Output is empty because in select statement ‘1’is string but t_pad type is int

The index structure goes from the root to leaf block to a single entry, gets a rowid, which is used to access the table data block containing one row. The TABLE ACCESS BY INDEX ROWID step in the plan specifies access to a table data block.

2.3. Task 4: Index Range Scan

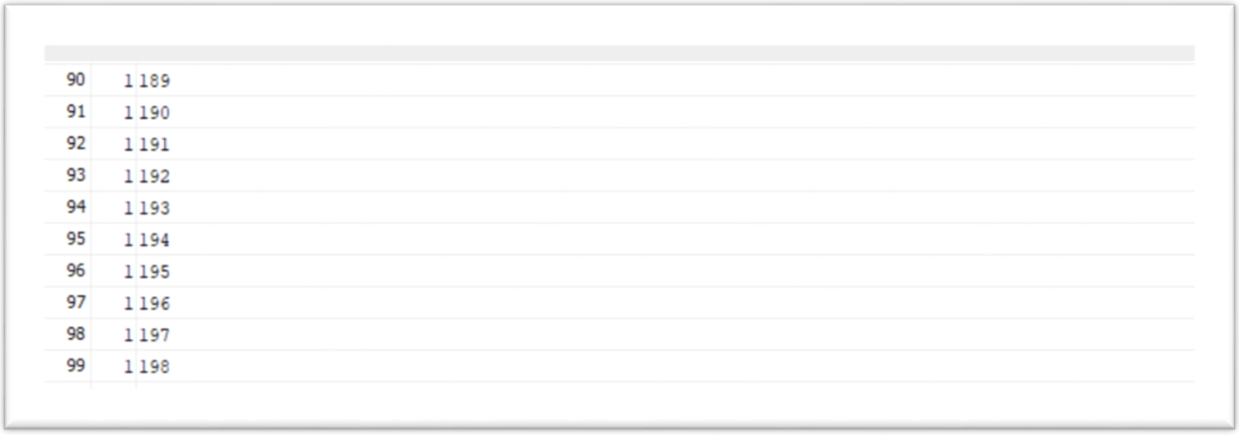
```
SELECT t2.*  FROM t2 where t2.id = '1';
```



Explain Plan | Query Result

All Rows Fetched: 100 in 0,045 seconds

	ID	T_PAD
1	1	100
2	1	101
3	1	102
4	1	103
5	1	104
6	1	105
7	1	106
8	1	107
9	1	108
10	1	109



90	1	189
91	1	190
92	1	191
93	1	192
94	1	193
95	1	194
96	1	195
97	1	196
98	1	197
99	1	198

A range scan goes through the index structure from the root block to the first leaf block containing an entry that matches the given condition. From this starting point, the rowid is retrieved from the index entry and the table data block (TABLE ACCESS BY INDEX ROWID) is retrieved. After the first row is retrieved, the leaf block of the index is accessed again and the next entry is read to obtain the next row ID. This exchange of data between leaf index blocks and data blocks continues until all matching index entries have been read.

2.4. Task 5: Index Skip Scan

Creation of the table employees:

```
CREATE TABLE employees AS
  SELECT *
```

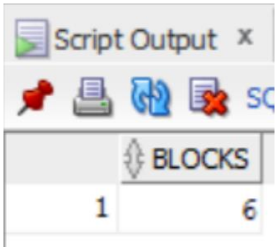
```
        FROM scott.emp;
```

Creation of the index:

```
    CREATE INDEX idx_emp01 ON employees
       ( empno, ename, job );
```

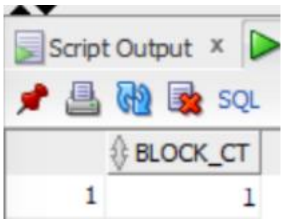
Listed number of allocated blocks and how many blocks contain data:

```
SELECT BLOCKS FROM user_segments WHERE segment_name = 'EMP';
```



	BLOCKS
1	6

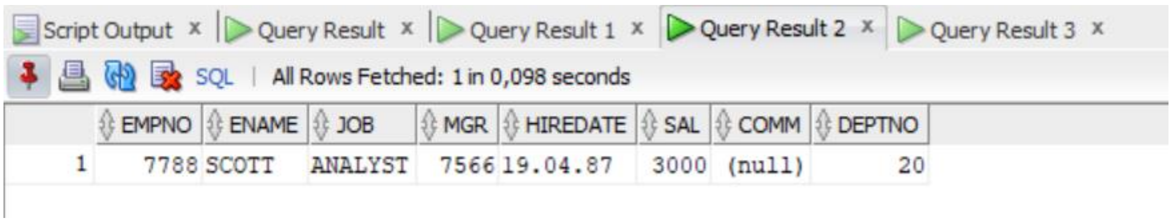
```
SELECT COUNT(DISTINCT (dbms_rowid.rowid_block_number(rowid)))
block_ct

FROM emp ;
```



	BLOCK_CT
1	1

Get trace and statistic of explain plan

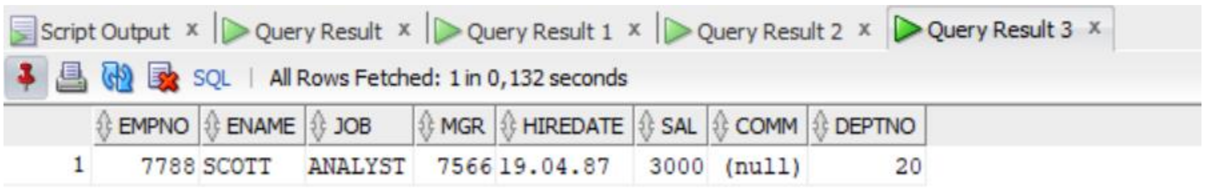
```
  SELECT /*+INDEX_SS(emp idx_emp01)*/ emp.* FROM employees
emp where ename = 'SCOTT';
```



All Rows Fetched: 1 in 0,098 seconds

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7788	SCOTT	ANALYST	7566	19.04.87	3000	(null)	20

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



All Rows Fetched: 1 in 0,132 seconds

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7788	SCOTT	ANALYST	7566	19.04.87	3000	(null)	20

№	Count of Blocks	Count of Used Blocks	Count of Rows	Consistent gets	Description
Full scan	6	1	12	8	Oracle performed SELECT (using the hint /* +FULL(emp)*/) and used the Full Scan algorithm
Index skip scan	6	1	12	2	Oracle performed SELECT (using the hint /*+INDEX_SS(emp idx_emp01)*/) and used the Index Scip Scan algorithm
Without hint	6	1	12	5	Oracle performed SELECT (without using hints) and used the most appropriate Full Scan

The optimizer will tend not to use the index since our predicate did not involve the column X—it might have to inspect each and every index entry in this case (we'll discuss an index skip scan shortly where this is not true). It will typically opt for a full table scan of T instead. That does not preclude the index from being used. If the query was SELECT X,Y FROM T WHERE Y = 5, the optimizer would notice that it did not have to go to the table to get either X or Y (they are in the index) and may very well opt for a fast full scan of the index itself, as the index is typically much smaller than the underlying table.