**Alexander Korytko**

Task 4

# Table access full scan

## 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;

Step 2:

# CREATE INDEX t2\_idx1 ON t2

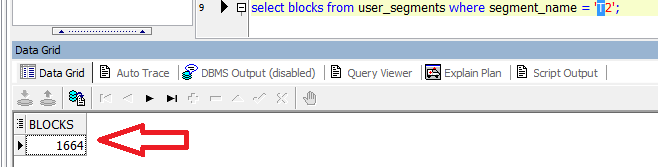
( id );

**Step 3:**

Block count:

# select blocks from user\_segments where segment\_name = 'T2';

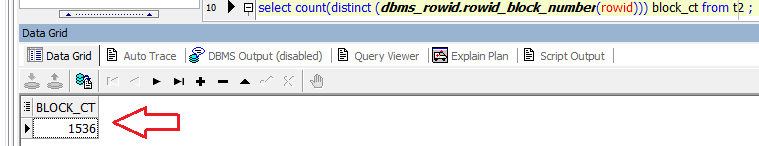
**Result:**



Used Block Count:

# select count(distinct (dbms\_rowid.rowid\_block\_number(rowid))) block\_ct from t2 ;

**Result:**



Explain Plan:

# SET autotrace ON;

# SELECT COUNT( \* )

FROM t2 ;

Statistics

----------------------------------------------------------

0 recursive calls

0 db block gets

1541 consistent gets

0 physical reads

0 redo size

528 bytes sent via SQL\*Net to client

519 bytes received via SQL\*Net from client

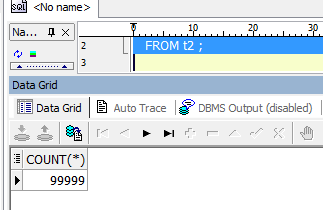
2 SQL\*Net roundtrips to/from client

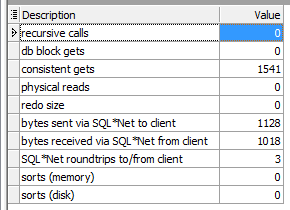
0 sorts (memory)

0 sorts (disk)

1. rows processed

**Result:**

****

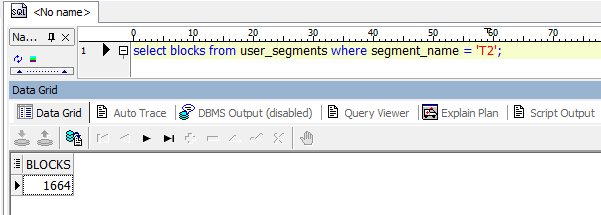


Step 4: Delete All Rows from table

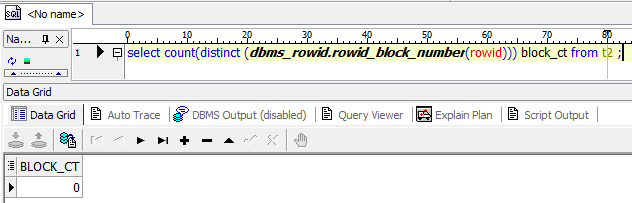
# DELETE FROM t2;

**Step 5:** Repeat Step 3 and collect results.

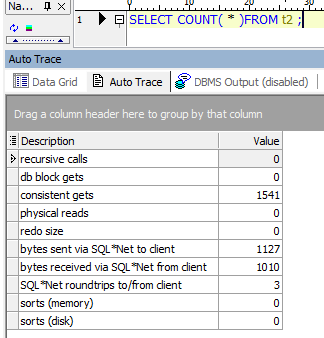
select blocks from user\_segments where segment\_name = 'T2';



select count(distinct (dbms\_rowid.rowid\_block\_number(rowid))) block\_ct from t2 ;



SELECT COUNT( \* )FROM t2 ;



Step 6: Insert 1 row

# INSERT INTO t2

( ID, T\_PAD )

VALUES

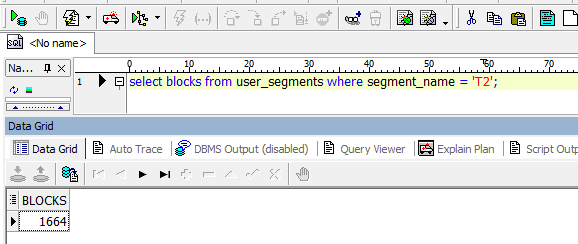
( 1,'1' );

COMMIT;

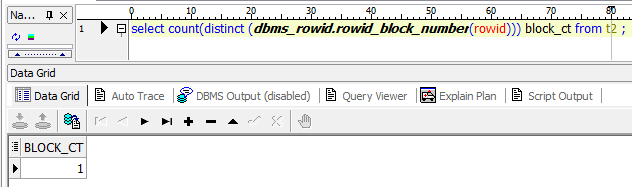
**Step 7:** Repeat Step 3 and collect results.

**RESULTS:**

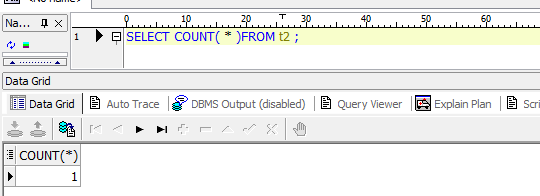
select blocks from user\_segments where segment\_name = 'T2';

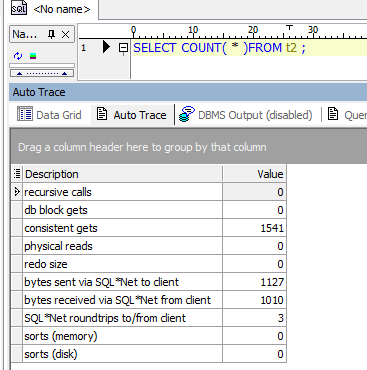


select count(distinct (dbms\_rowid.rowid\_block\_number(rowid))) block\_ct from t2 ;



SELECT COUNT( \* )FROM t2 ;





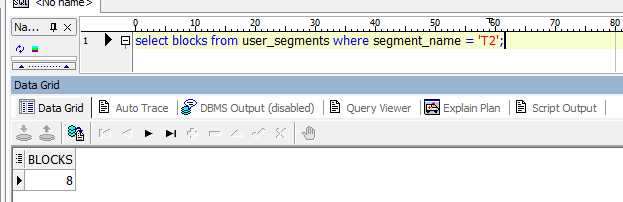
Step 8: Truncate Table

# TRUNCATE TABLE t2;

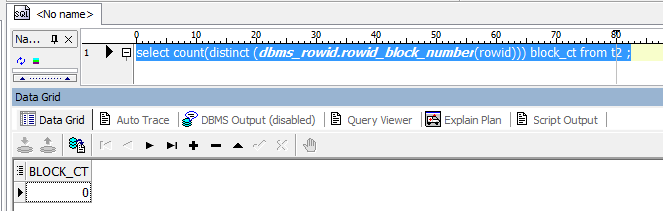
**Step 9:**  Repeat Step 3 and collect results.

**RESULTS:**

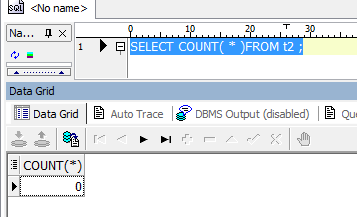
select blocks from user\_segments where segment\_name = 'T2';

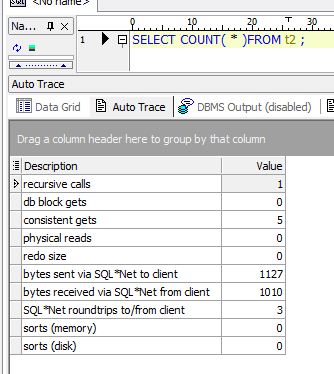


select count(distinct (dbms\_rowid.rowid\_block\_number(rowid))) block\_ct from t2 ;



SELECT COUNT( \* )FROM t2 ;





**Task Results:**

Expected:

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 1664 | 1536 | 99999 | 1541 | **Step 3** – Create table and select Data. Result: Count of Rows 99999 and HWM is MAX now HWM=1541 |
| 2 | 1664 | 0 | 0 | 1541 | **Step 5** – Delete Rows from Table. Result: Count of Rows =0 but Consistent gets = 1541 because **memory is still dedicated for our segment** even if Count of Used Blocks = 0 |
| 3 | 1664 | 1 | 1 | 1541 | **Step 7** – Insert 1 Row to our Table. Result: Count of Rows =1 now, Consistent gets = 1541 because **memory is still dedicated for last MAX Count of Rows** even Count of Used Blocks = 1 |
| 4 | 8 | 0 | 0 | 5 | **Step 9** – Truncate our Table. Result: Count of Rows =0 , Consistent gets = 5 because **memory not dedicated more for our Max Count Items and now it\s =5,** Count of Used Blocks = 8 (8 because there is still some information about our table) |

# Index Scan types

## Task 2: Index Clustering factor parameter

Step 1: Create table t2 as on task 1 step 1-2

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 );

Step 2: Create table t1 as listed below

# CREATE TABLE t1 AS

SELECT MOD( rownum, 100 ) id, rpad( rownum,100 ) t\_pad

FROM dual

CONNECT BY rownum < 100000;

Step 3:

# CREATE INDEX t1\_idx1 ON t1

( id );

Step 4: 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 );

**Step 5:** Select Clustering Factor

# SELECT t.table\_name||'.'||i.index\_name idx\_name,

i.clustering\_factor,

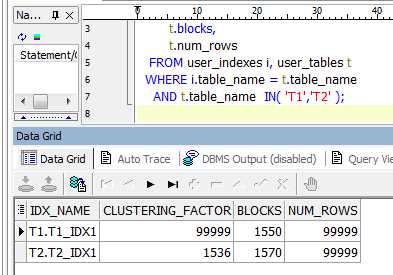
t.blocks,

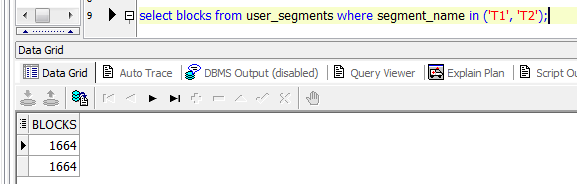
t.num\_rows

FROM user\_indexes i, user\_tables t

WHERE i.table\_name = t.table\_name

AND t.table\_name IN( 'T1','T2' );





**Task Results:**

Expected:

* Screenshot of the step 5;
* Description of the parameter clustering factor;

The clustering factor statistic of an index helps the optimizer generate the cost of using the index and is a measure of how well ordered the table data is as related to the indexed values. Recall that index entries are stored in sorted order while table data is stored in random order. Unless an effort has been made to specifically load data into a table in a specific order, you are not guaranteed where individual rows of data will end up. For example, rows from the orders table that share the same order\_date may not all reside in the same blocks. They are likely to be scattered randomly across the blocks in the table. The clustering factor of an index indicates to the optimizer if data rows containing the same indexed values will be located in the same or a small set of contiguous blocks, or if rows will be scattered across numerous table blocks.

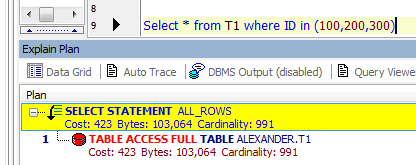
* Explanation: why for indexes t1\_idx1 and t2\_idx1 we have different values ;

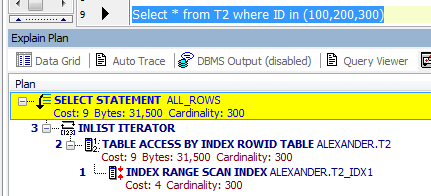
In table T2 we have a lot of duplicate values (for example, 100 items with ID= 0) and in T1 in Table 2, all the different ID (ID=1,2,3,4,5,6,7,8,9,10 and etc), that’s why clustering factor in T1 > than clustering factor in T2

* Which Index has best selective performance in execution Select clause filtered by IN ( , list of values, );

The best selective performance will be in table T2, because in case we select all items from Table T2 we a should get FULL ACCESS to our Table and its COST more. See screenshots for more details

**Results:**





## Task 3: Index Unique Scan

**Step 1:**

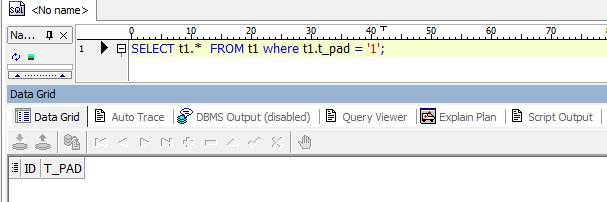
# CREATE UNIQUE INDEX udx\_t1 ON t1( t\_pad );

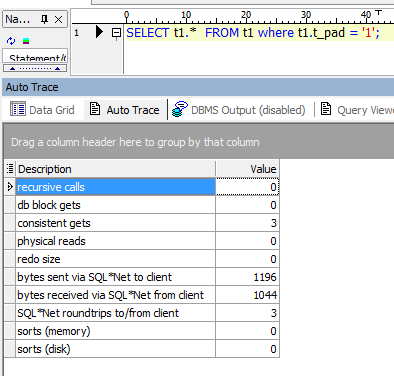
**Step 2**

# SELECT t1.\* FROM t1 where t1.t\_pad = '1';

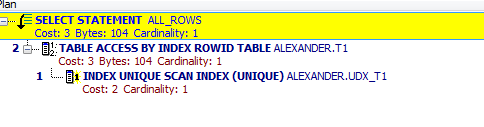
**Task Results:**

* Screenshot of the step 2;





* Description of process: How oracle read block on step 2;



Step 1 – Oracle Scan table with Unique Index

Step 2 – Oracle access our table by Index RowId

## Task 4: Index Range Scan

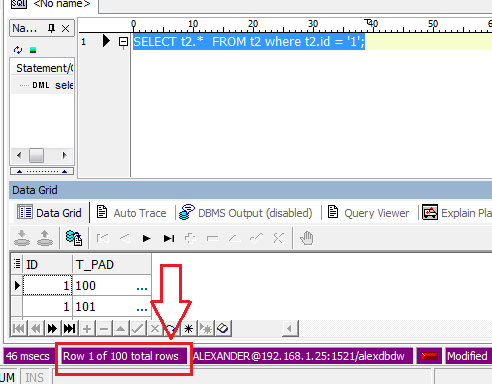
**Step 1:**

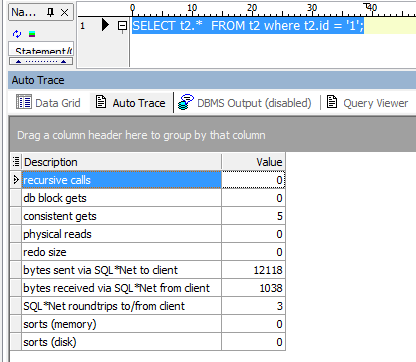
# SELECT t2.\* FROM t2 where t2.id = '1';

**Task Results:**

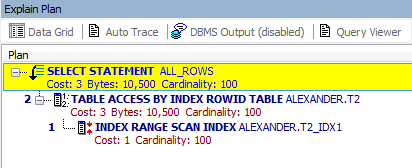
Expected:

* Screenshot of the step 1;





* Description of process: How oracle read block on step 1;



Step 1 – Index range scan Index in table T2

Step 2 – Access t2 by Index RowId

## Task 5: Index Skip Scan

Step 1:

# CREATE TABLE employees AS

SELECT \*

FROM scott.emp;

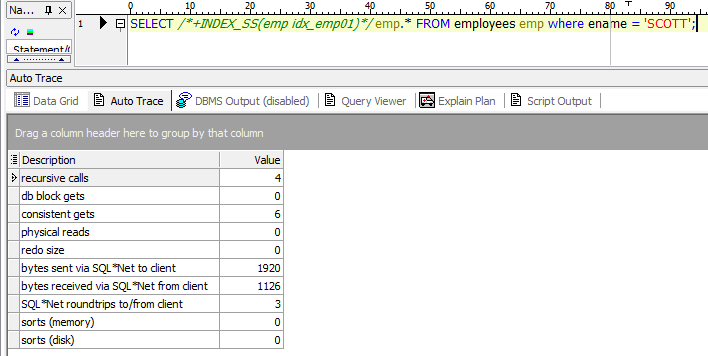
Step 2:

# CREATE INDEX idx\_emp01 ON employees

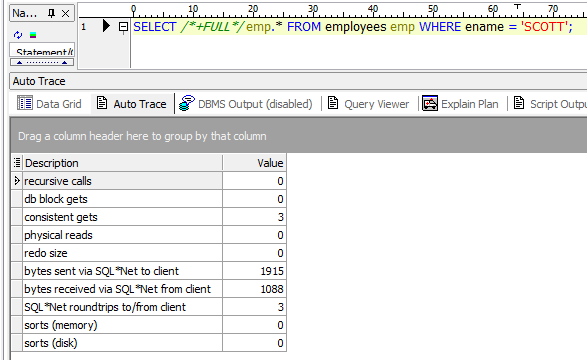
( empno, ename, job );

**Step 3:**  Get trace and statistic of explain plan

# SELECT /\*+INDEX\_SS(emp idx\_emp01)\*/ emp.\* FROM employees emp where ename = 'SCOTT';



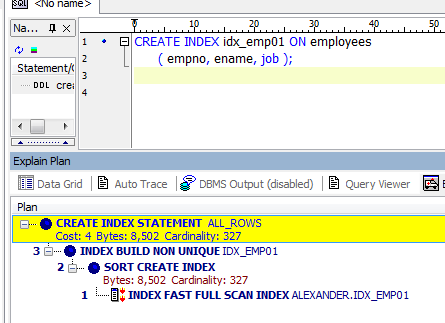
# SELECT /\*+FULL\*/ emp.\* FROM employees emp WHERE ename = 'SCOTT';



**Task Results:**

Expected:

* 2 Screenshots of the step 3
* Description of process: How oracle analyses index that was created on step 2;



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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 1664 | 1536 | 99999 | 3 | Index Unique Scan |
| 2 | 1664 | 1536 | 99999 | 5 | Range index |
| 3 | 1664 | 1536 | 99999 | 6 | Index Skip Scan (/\*+INDEX\_SS) |
| 4 | 1664 | 1536 | 99999 | 3 | Index Skip Scan (/\*+FULL\*/) |