# 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

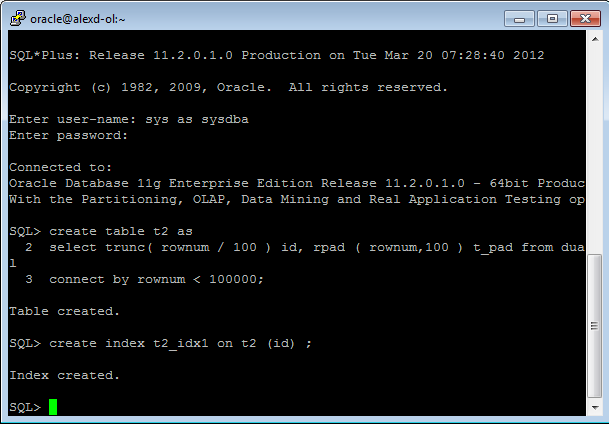
FROM dual

CONNECT BY rownum < 100000;

Step 2:

# CREATE INDEX t2\_idx1 ON t2

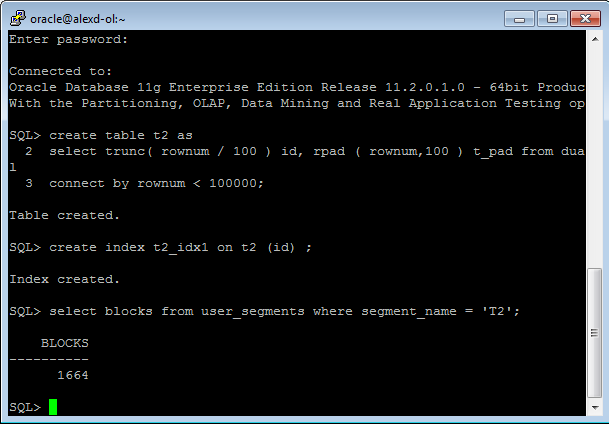
( id );



**Step 3:**

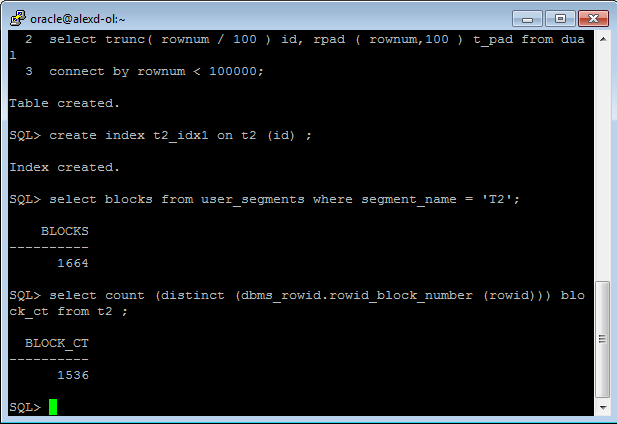
Block count:

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



Used Block Count:

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

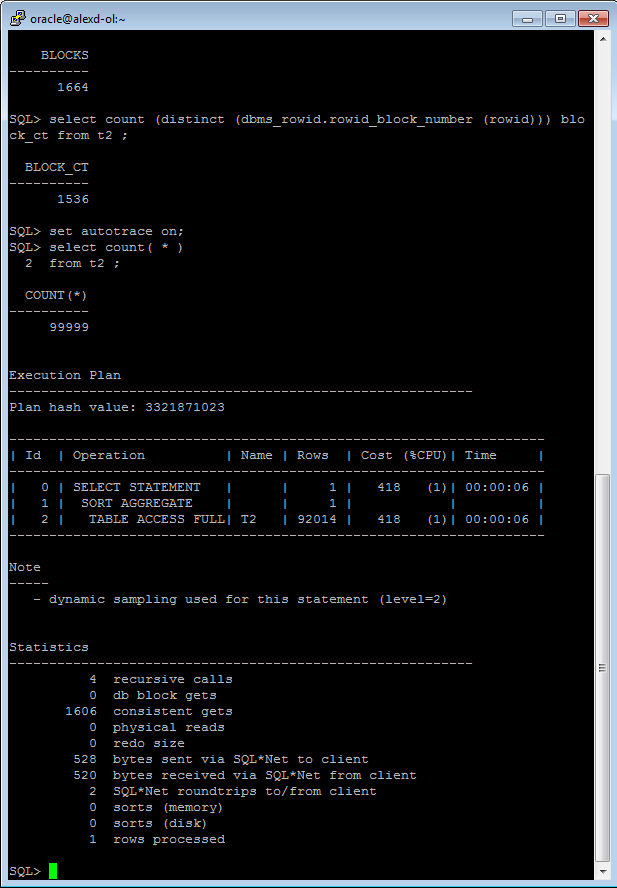


Explain Plan:

# 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

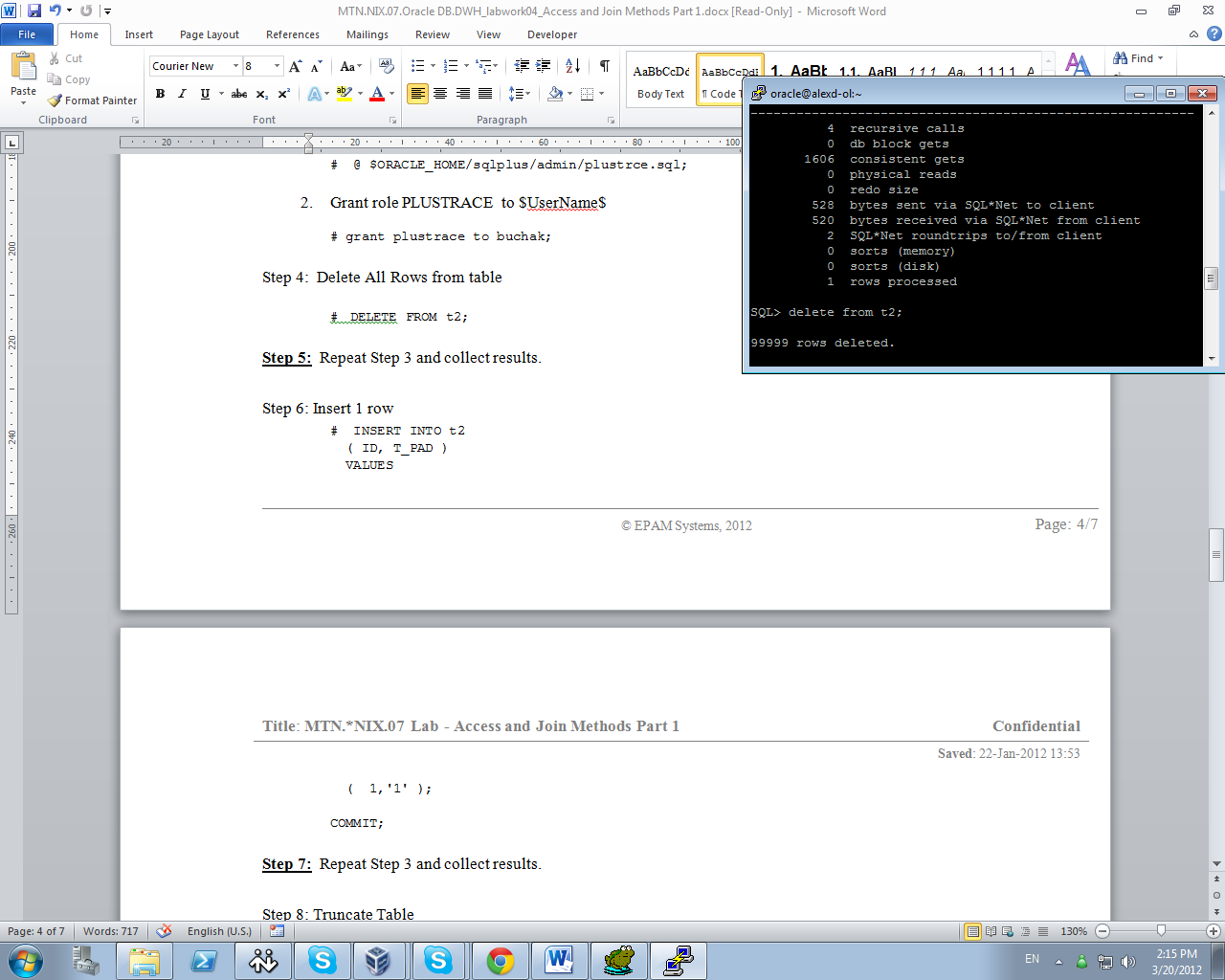
0 sorts (memory)

0 sorts (disk)

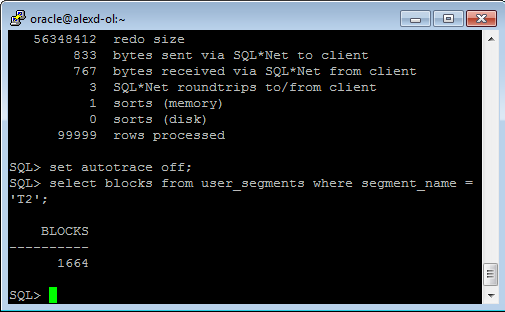
1. rows processed

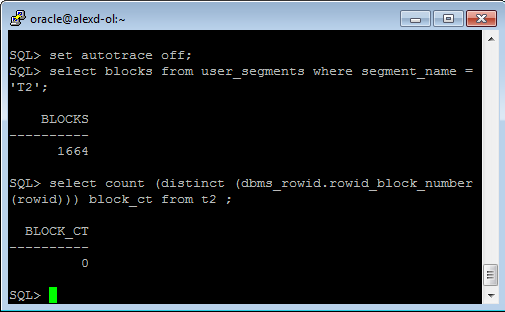
Step 4: Delete All Rows from table

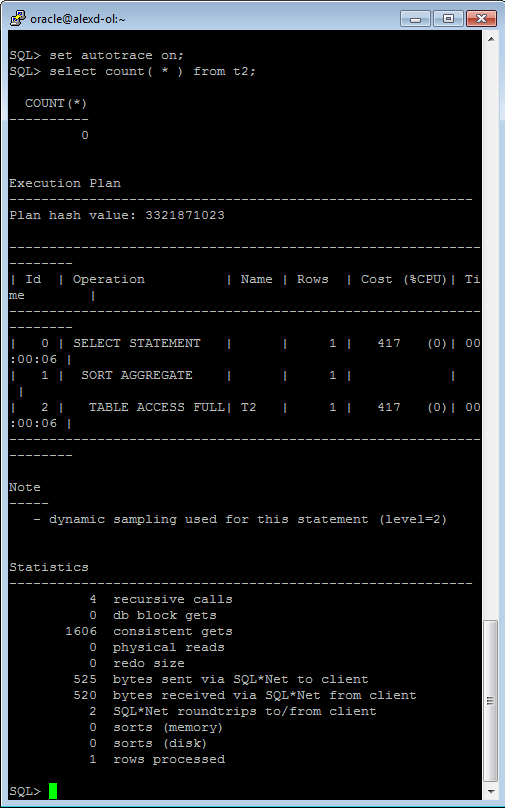
# DELETE FROM t2;



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







Step 6: Insert 1 row

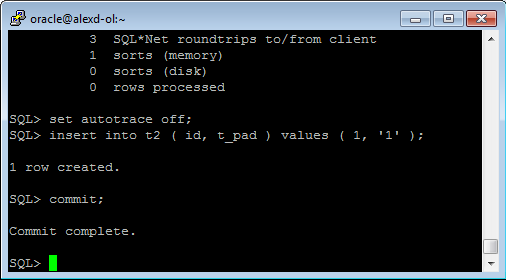
# INSERT INTO t2

( ID, T\_PAD )

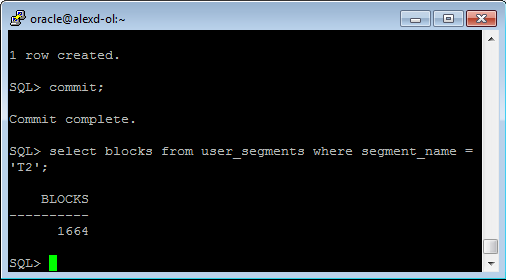
VALUES

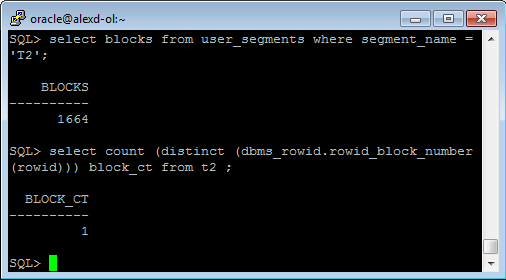
( 1,'1' );

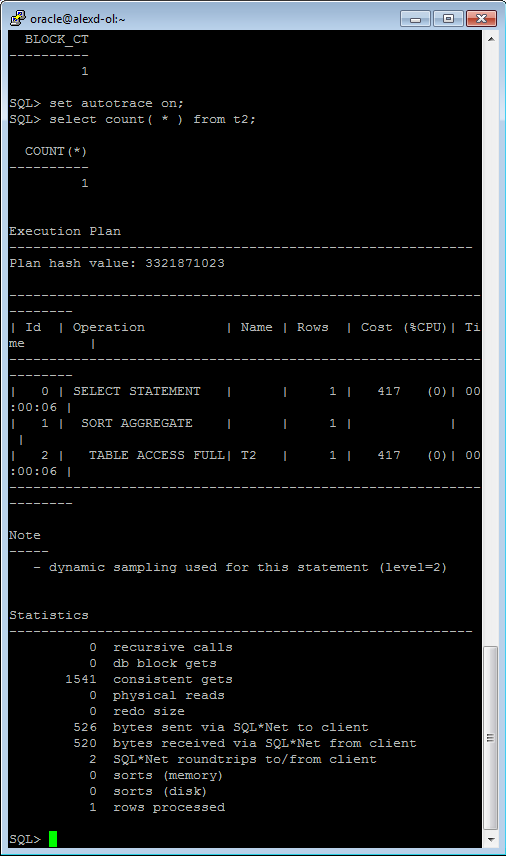
COMMIT;



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

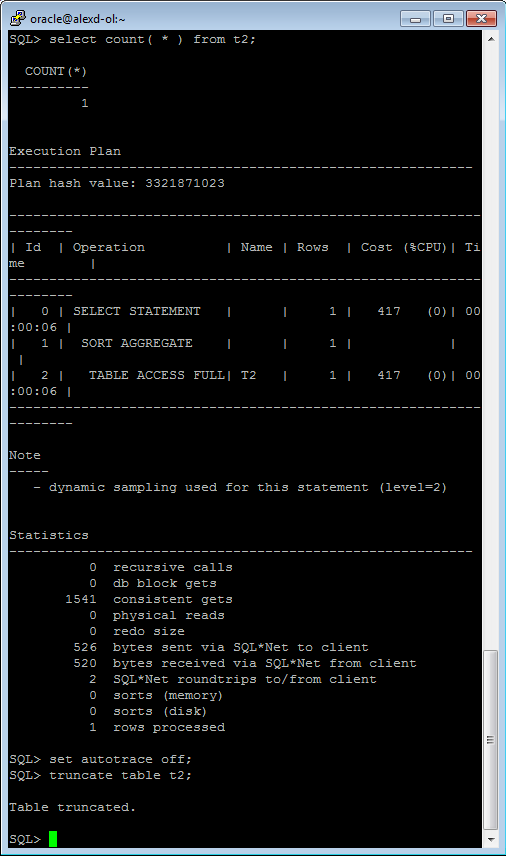




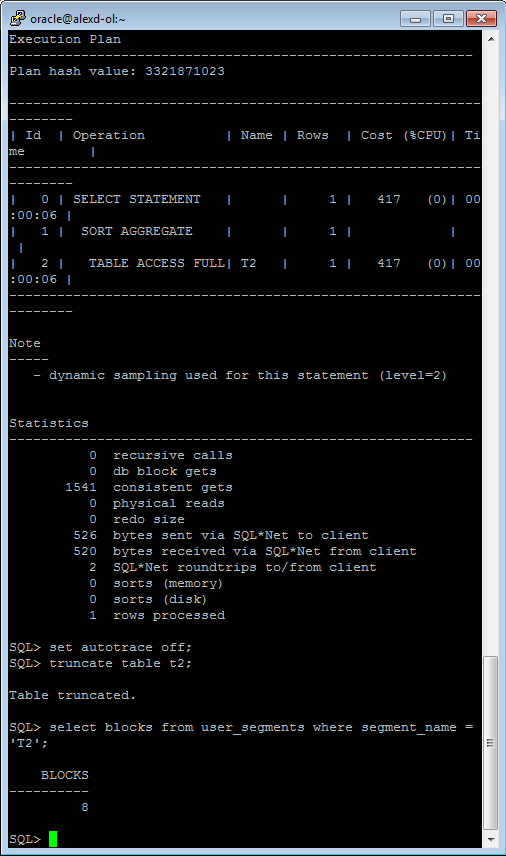


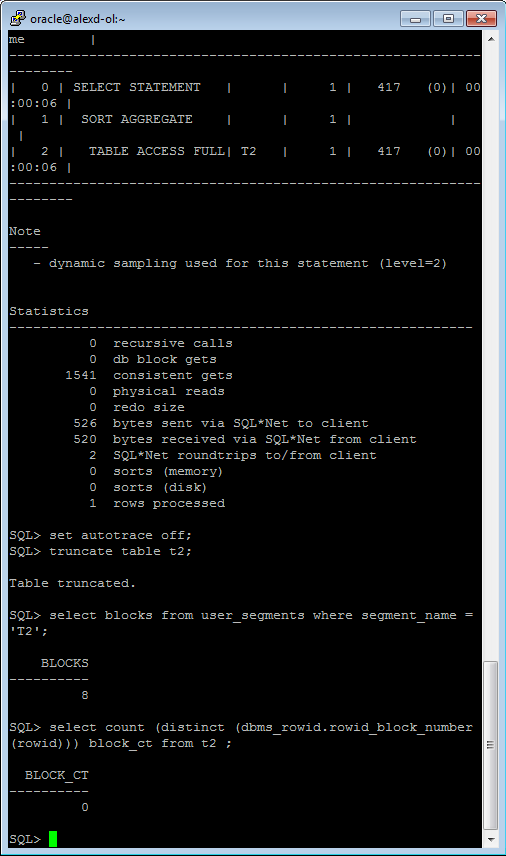
Step 8: Truncate Table

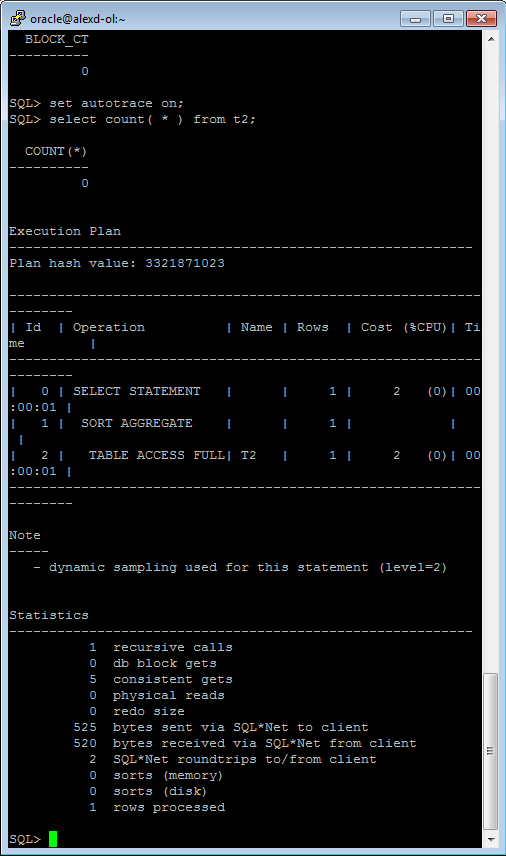
# TRUNCATE TABLE t2;



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







**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 | 1606 | We have hard task for db(cons 1606) |
| 2 | 1664 | 0 | 0 | 1606 | We have deleted all informations, but oracle doesn’t know it, the water marks exists. |
| 3 | 1664 | 1 | 1 | 1541 | We a making full scan of our table, the water marks exists. |
| 4 | 8 | 0 | 0 | 5 | The table had been truncated. the watermarks deleted, blocks are free. |

# Index Scan types

## Task 2: Index Clustering factor parameter

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

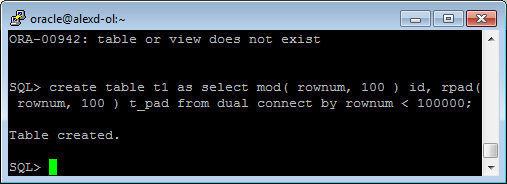
Step 2: Create table t1 as listed below

# CREATE TABLE t1 AS

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

FROM dual

CONNECT BY rownum < 100000;



Step 3:

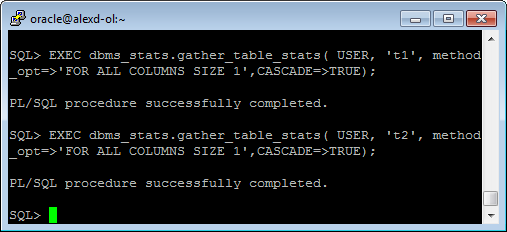
# CREATE INDEX t1\_idx1 ON t1

( 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,

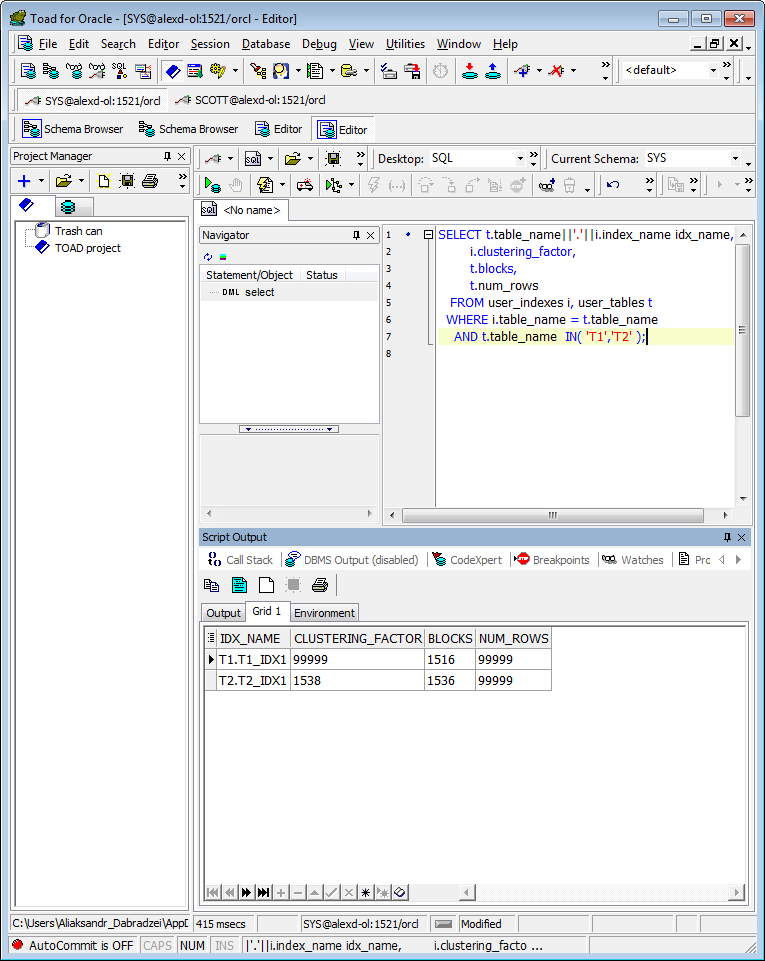
t.blocks,

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;
* Explanation: why for indexes t1\_idx1 and t2\_idx1 we have different values ;
* Which Index has best selective performance in execution

In Oracle the clustering factor of an index is a single number that is supposed to represent the correlation between the order of the index and the order of the corresponding table.

The table data is scattered in relation to the index order, and it tend to be ignored by the cost-based optimizer Table 2 we have less clustered factor and better performance.

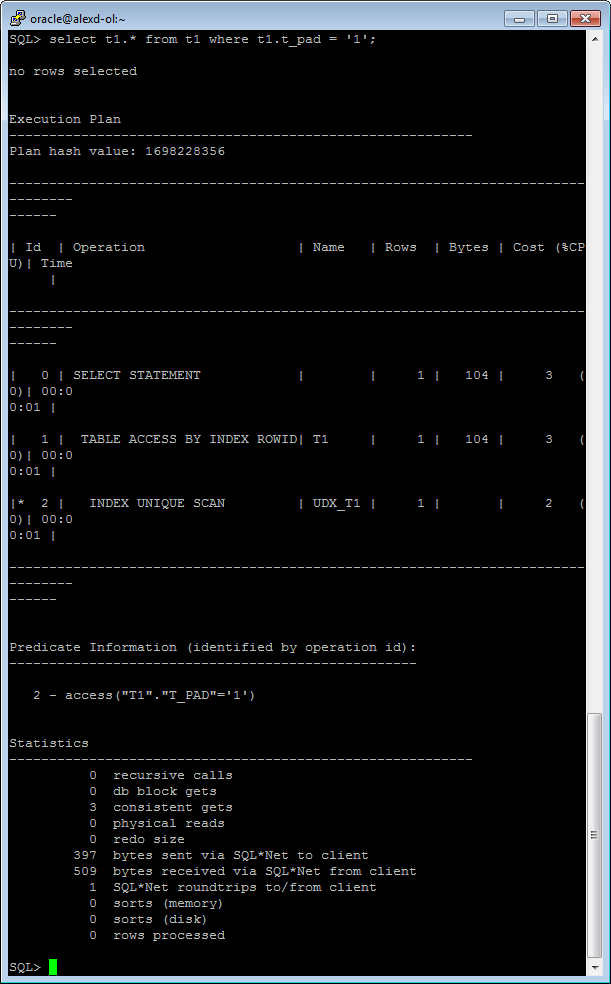
## Task 3: Index Unique Scan

Step 1:

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

**Step 2**

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



**Task Results:**

Expected:

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

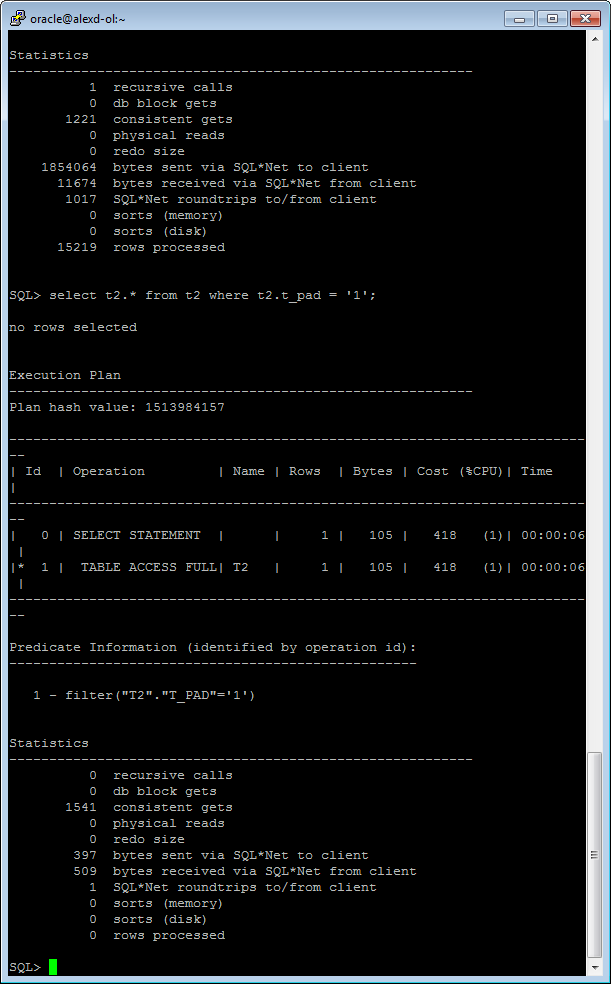
An index unique scan is chosen when a predicate contains a condition using a column defined with a

UNIQUE or PRIMARY KEY index. These types of indexes guarantee that only one row will ever be returned for a specified value.

## 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;

An index range scan is chosen when a predicate contains a condition that will return a range of data.

The index can be unique or non-unique as it is the condition that determines whether or not multiple

rows will be returned or not. A range scan will traverse the index structure from the root block to the first leaf block containing

an entry matching the specified condition. From that starting point, a rowid will be retrieved from the

index entry and the table data block will be retrieved (TABLE ACCESS BY INDEX ROWID). After the first

row is retrieved, the index leaf block will be accessed again and the next entry will be read to retrieve

the next rowid. This back-and-forth between the index leaf blocks and the data blocks will continue

until all the matching index entries have been read.

## Task 5: Index Skip Scan

Step 1:

# CREATE TABLE employees AS

SELECT \*

FROM scott.emp;

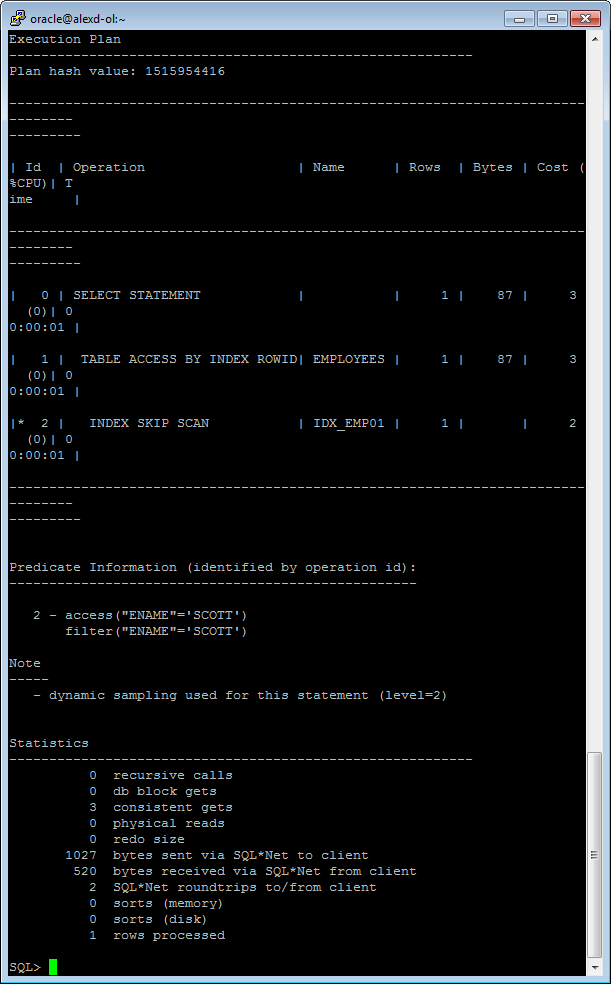
Step 2:

# CREATE INDEX idx\_emp01 ON employees

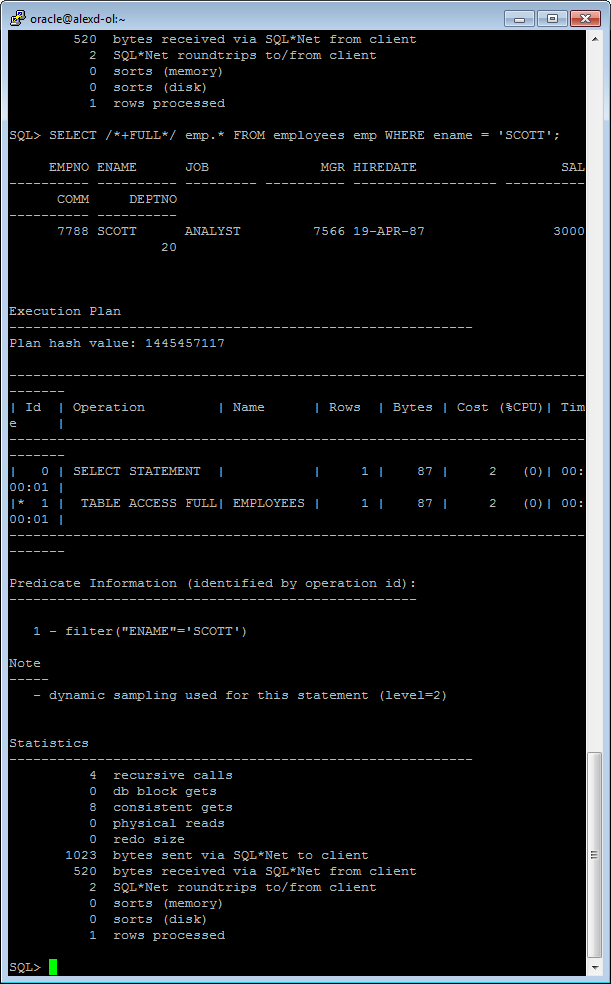
( empno, ename, job );

**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 | Unique index |
| 2 | 1664 | 1536 | 99999 | 1541 | Range index |
| 3 | 1664 | 1536 | 99999 | 3 | Index skip (index\_ss) |
| 4 | 1664 | 1536 | 99999 | 8 | Index skip (full access) |

An index skip scan is chosen when the predicate contains a condition on a non-leading column in an

index and the leading columns are fairly distinct. A skip scan works by logically

splitting a multi-column index into smaller subindexes. The number of logical subindexes is

determined by the number of distinct values in the leading columns of the index. Therefore, the more

distinct the leading columns are, the more logical subindexes would need to be created. If too many

subindexes would be required, the operation won’t be as efficient as simply doing a full scan.

However, in the cases where the number of subindexes needed would be smaller, the operation can be

many times more efficient than a full scan as scanning smaller index blocks can be more efficient

than scanning larger table blocks.