|  |
| --- |
| EPAM Systems, RD Dep.  MTN.\*NIX.07 Oracle DB. Introduction to DWH |
| MTN.\*NIX.07 Lab - Access and Join Methods Part 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial status of document | [**Kiryl Bucha**](mailto:Kiryl_bucha@epam.com) | 16-JAN-2012 |  |  |
|  |  |  |  |  |  |

*Contents*

[1. Table access full scan 3](#_Toc315003714)

[1.1. Task 1: Full Scans and the High-water Mark and Block reading 3](#_Toc315003715)

[2. Index Scan types 4](#_Toc315003716)

[2.1. Task 2: Index Clustering factor parameter 4](#_Toc315003717)

[2.2. Task 3: Index Unique Scan 5](#_Toc315003718)

[2.3. Task 4: Index Range Scan 5](#_Toc315003719)

[2.4. Task 5: Index Skip Scan 6](#_Toc315003720)

# 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:

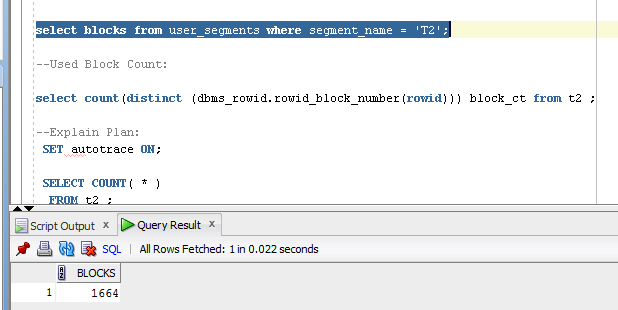
# CREATE INDEX t2\_idx1 ON t2

( id );

**Step 3:**

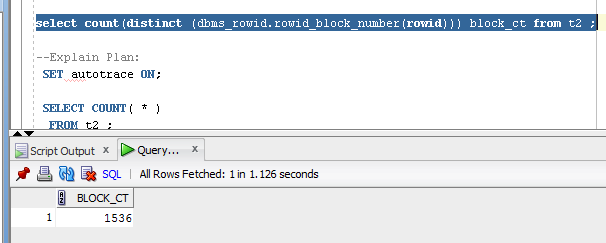
Block count:

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



Used Block Count:

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

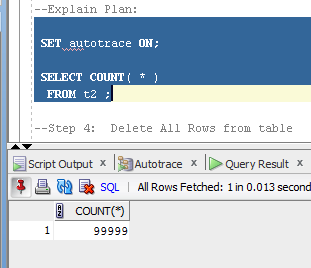


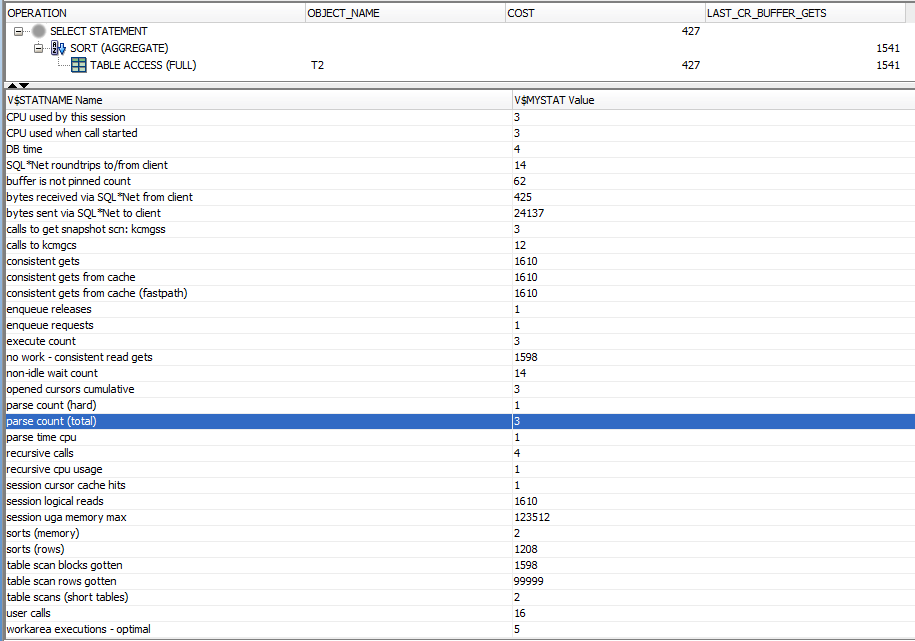
Explain Plan:

# SET autotrace ON;

# SELECT COUNT( \* )

FROM t2 ;

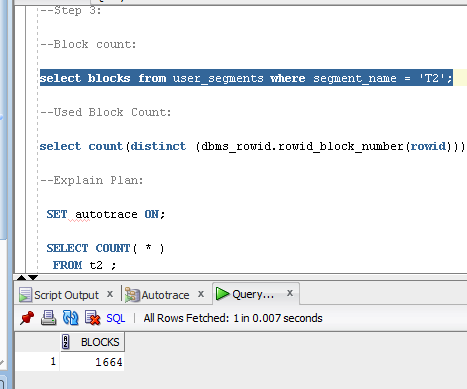


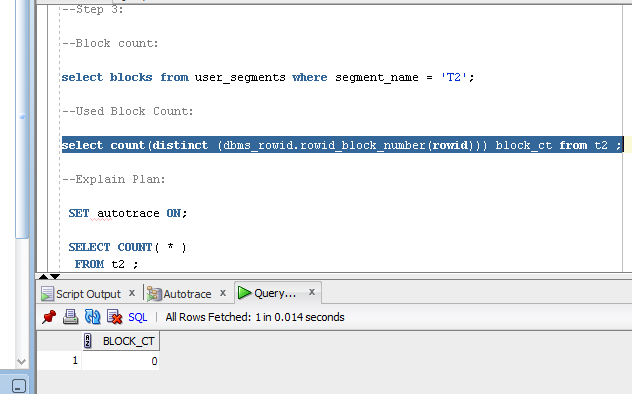


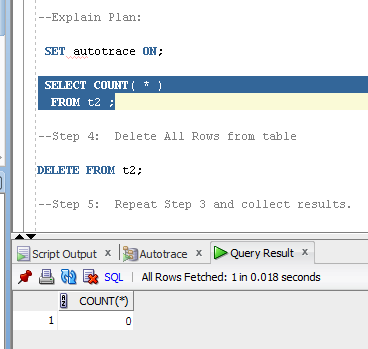
Step 4: Delete All Rows from table

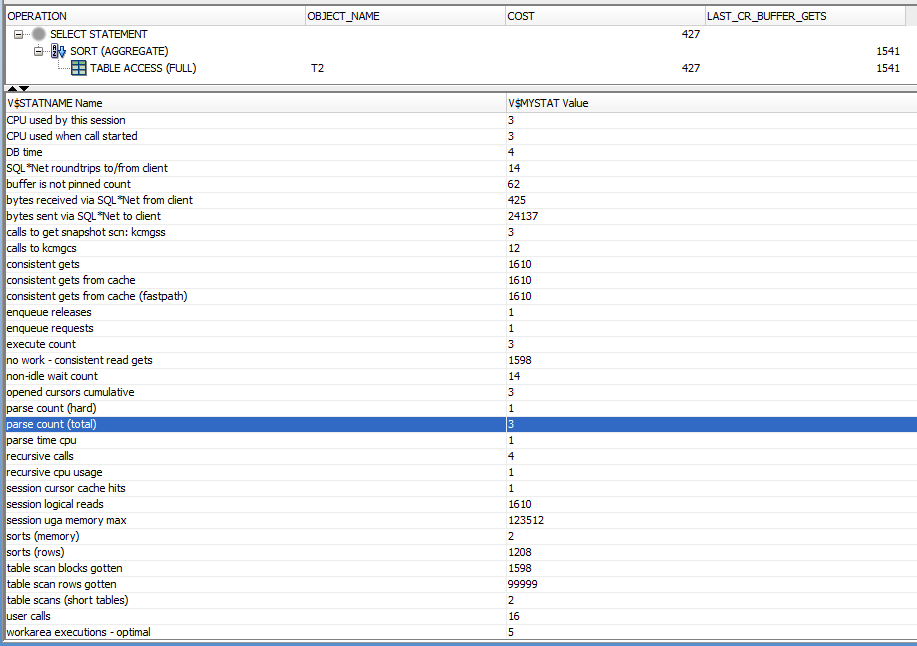
# DELETE FROM t2;

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









Step 6: Insert 1 row

# INSERT INTO t2

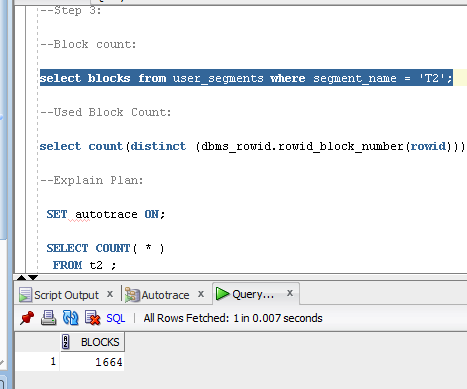
( ID, T\_PAD )

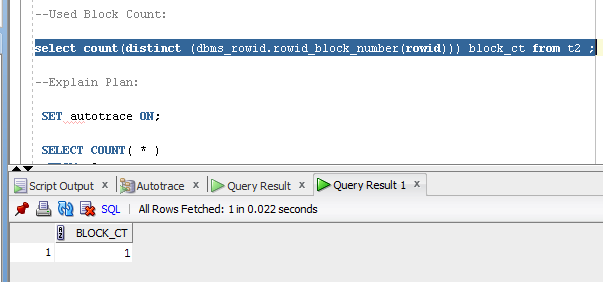
VALUES

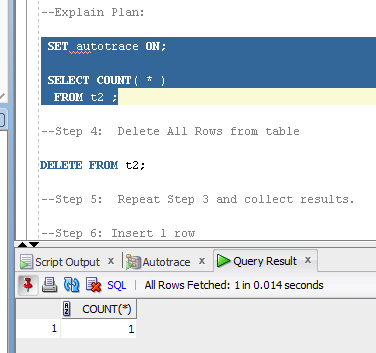
( 1,'1' );

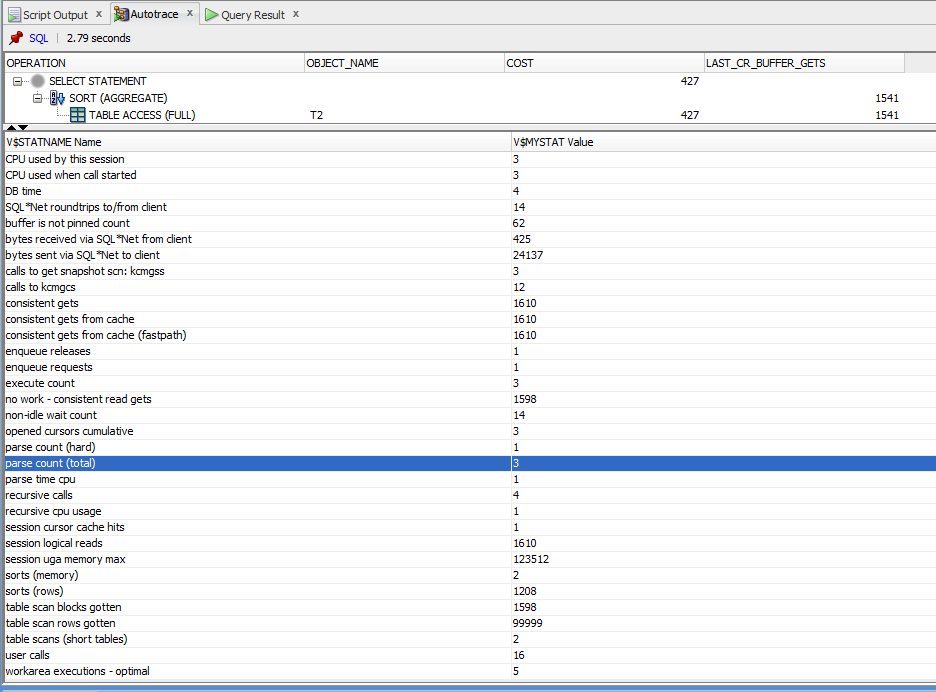
COMMIT;

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





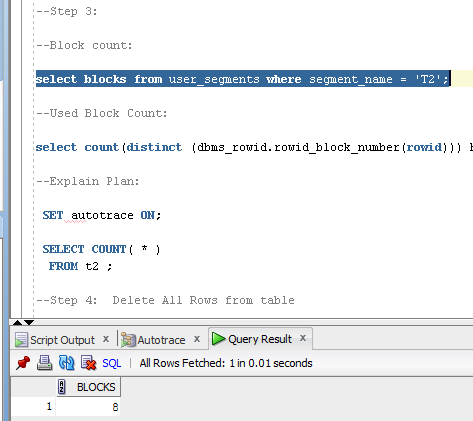


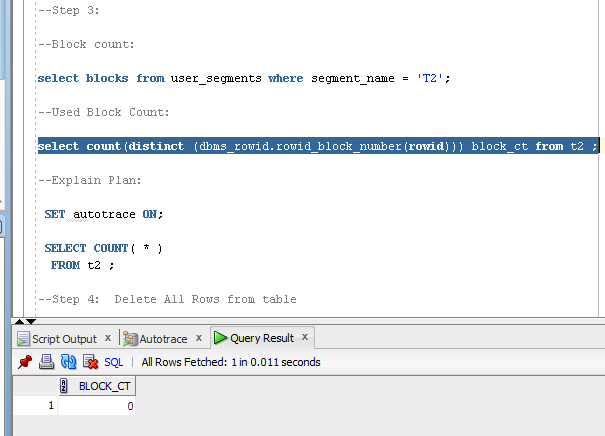


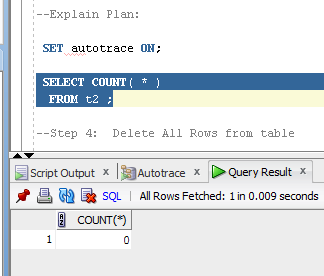
Step 8: Truncate Table

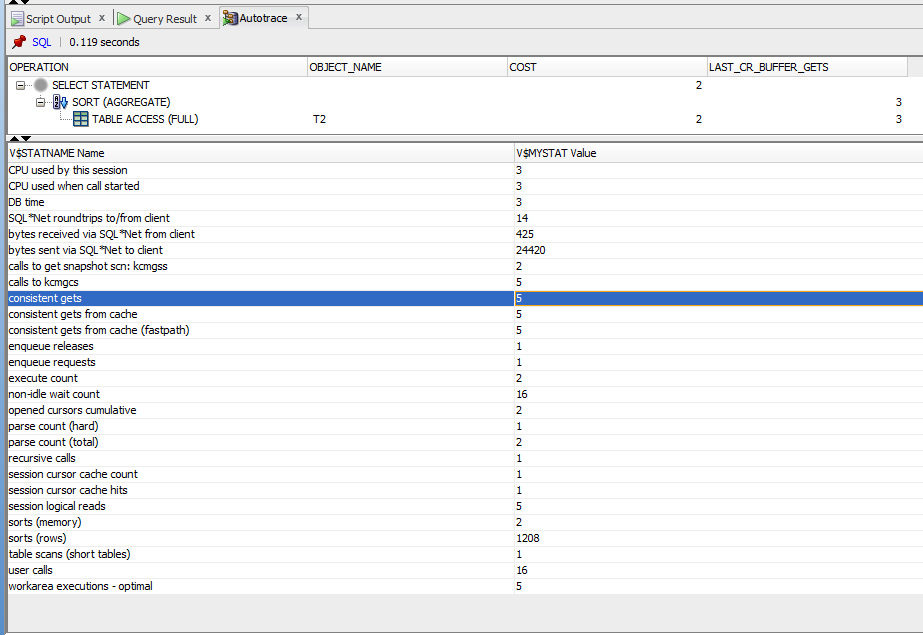
# TRUNCATE TABLE t2;

**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 | 1610 | Step3/After creation of the table t2 |
| 2 | 1664 | 0 | 0 | 1610 | Step5/After delete from t2 |
| 3 | 1664 | 1 | 1 | 1610 | Step7/After insert 1 row into t2 |
| 4 | 8 | 0 | 0 | 5 | Step9/After truncate table t2 |

There are some important aspects about access full scan to be mentioned while analyzing our results:

- Count of blocks is the whole amount of blocks to be accessed by oracle to prepare the result set of data. It is different for steps 3-7 and 9 because, when we create, insert, delete data from table it is accessible for oracle and the whole amount of beginning memory is visible for oracle as occupied for our table, but after truncate operation this amount of memory is not recognized as used and there are only 8 blocks describing the structure of the table is left.

- Consistent gets represent the high-water mark. This means that after we delete data from table2 1610 blocks will be steel read, despite the fact, that there is no data there. Only after truncating the table HWM is changed.

- In this exercise we can compare delete and truncate operations, they return 0 used blocks but number of count blocks (occupied space) with our table is different (explanation above). And compare operations of insert (when create table and after delete) the high-water mark, that was set after creation of the table doesn’t change even if we have only 1 used block so we read 1610-1 empty (no good indeed) .

# 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

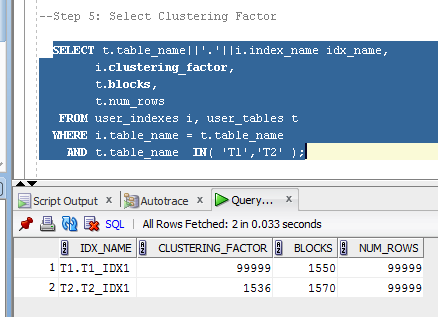
WHERE i.table\_name = t.table\_name

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

**Task Results:**

Expected:

* Screenshot of the step 5;



* Description of the parameter clustering factor;

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.

If an execution plan contains an index range scan including an access to a table by ROWID, the clustering factor tells the cost-based optimizer how "clustered" or "scattered" the data in the table is with respect to the index - that is on average how likely contiguous rows from an index range scan will point to the same table block. Widely scattered data will require to read a different block for every row returned from the index range scan, in contrast in the case of clustered data many of the contiguous rows from the index will point to the same block, making a significant difference to the cost calculated by the cost-based optimizer (and of course making also a significant difference at actual execution time, in particular when many table blocks have to be read from disk).

Usually the clustering factor in case of an index range scan with table access involved represents the largest fraction of the cost associated with the operation, therefore indexes with high clustering factors (meaning that the table data is scattered in relation to the index order) tend to be ignored by the cost-based optimizer and different access paths might be favored instead, like full table scans or the usage of different available indexes.

* 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, );

This results occurs due to the method of table creation, in table 1 we have all unique rows so the clustering factor has bigger value (99999) it means, that the table data is scattered in relation to the index order, and it tend to be ignored by the cost-based optimizer and different access paths might be favored instead, like full table scans or the usage of different available indexes. That’s why for table 2 (where we have many the same values-due to trunk operation) we have less clustered factor and better performance in execution Select clause filtered by IN ( , list of values, ) – less accesses to get result.

## Task 3: Index Unique Scan

Step 1:

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

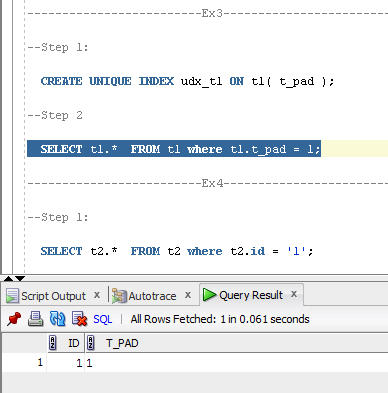
**Step 2**

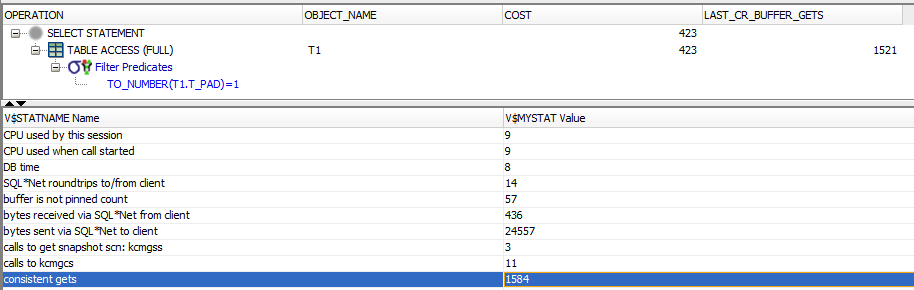
# SELECT t1.\* FROM t1 where t1.t\_pad = '1';

**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. In this cases, the index structure is traversed from root to leaf block to a

single entry, retrieve the rowid, and use it to access the table data block containing the one row. The

The number of block accesses required will always be equal to the height of the index plus one unless there are special circumstances like the row is chained or contains a LOB that is stored elsewhere.

## Task 4: Index Range Scan

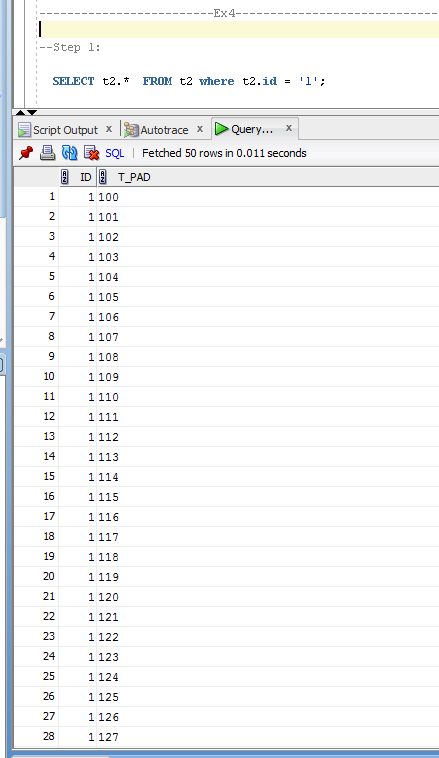
**Step 1:**

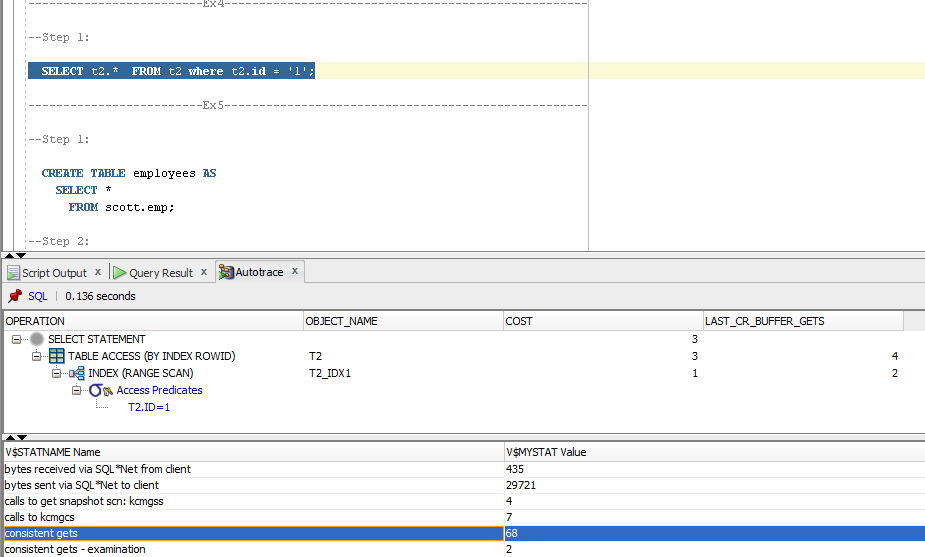
# SELECT t2.\* FROM t2 where t2.id = '1';

**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. The conditions specified can use operators such as <, >, LIKE, BETWEEN and

even =. In order for a range scan to be selected, the range will need to be fairly selective. The larger

the range, the more likely a full scan operation will be chosen instead.

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

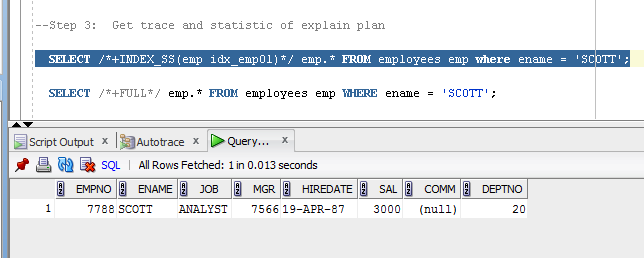
# SELECT /\*+INDEX\_SS(emp idx\_emp01)\*/ emp.\* FROM employees emp where ename = 'SCOTT';

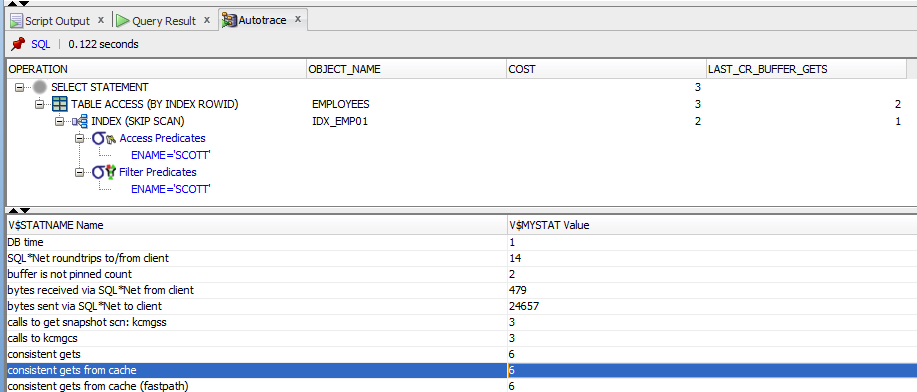
# SELECT /\*+FULL\*/ emp.\* FROM employees emp WHERE ename = 'SCOTT';

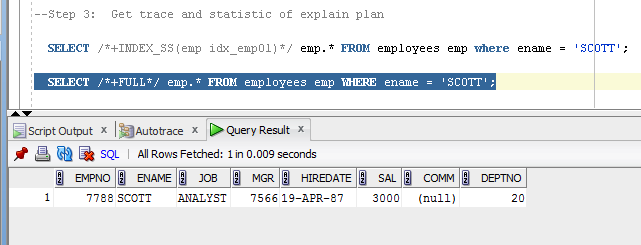
**Task Results:**

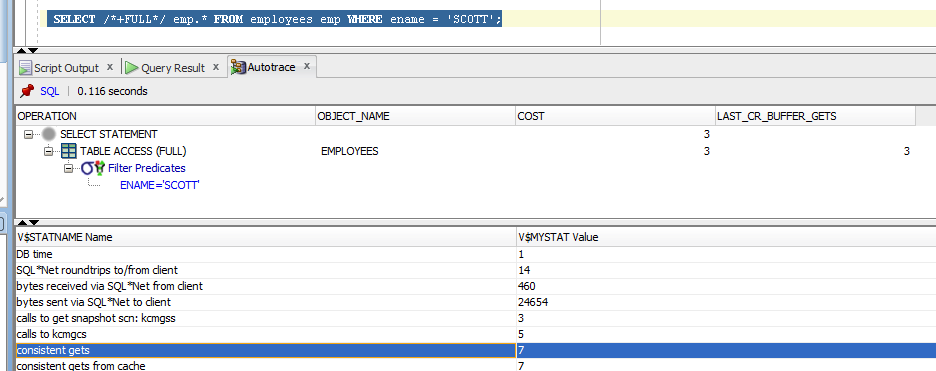
Expected:

* 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 | 68 | Range Scan |
| 2 | 1664 | 1536 | 99999 | 1584 | Unique scan |
| 3 | 8 | 1 | 14 | 7 | Index scip/full |
| 4 | 8 | 1 | 14 | 6 | Index scip/index |

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.