|  |
| --- |
| MTN.\*NIX.07 Lab - Access and Join Methods Part 1 |

# 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:

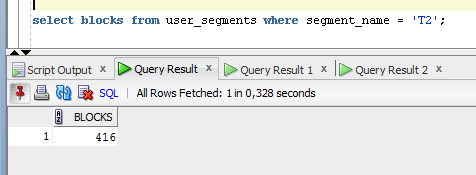
# CREATE INDEX t2\_idx1 ON t2

( id );

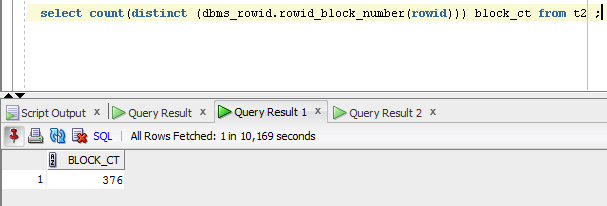
**Step 3:**

Block count:

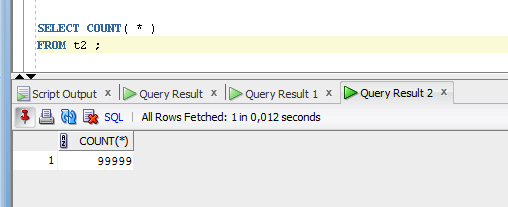
# select blocks from user\_segments where segment\_name = 'T2';



Used Block Count:



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



Explain Plan:

# SET autotrace on statistics

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

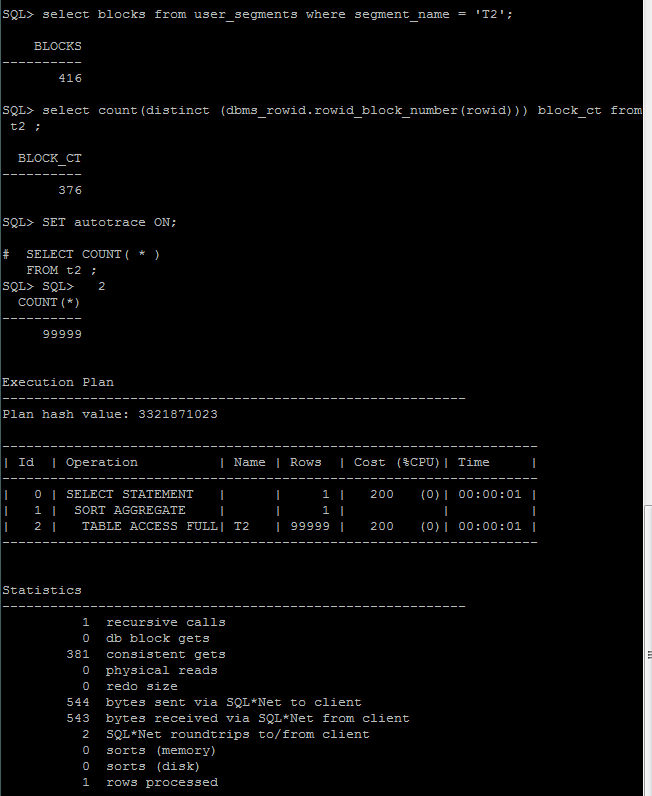
519 bytes received via SQL\*Net from client

2 SQL\*Net roundtrips to/from client

0 sorts (memory)

0 sorts (disk)

1. rows processed



**NOTE:** If you received next error: Check PLUSTRACE role is enabled. Please make next steps:

1. Run next script connected as sysdba:

# @ $ORACLE\_HOME/sqlplus/admin/plustrce.sql;

1. Grant role PLUSTRACE to $UserName$

# grant plustrace to VY;

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:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| select | 416 | 376 | 99999 | 381 | Oracle allocates table blocks for header and data. Count of blocks is greater than count of used blocks because oracle increases the size of table by allocating a few blocks at a time (extents) |
| delete |  |  |  | 387 | Data was deleted from table. Physically data is in blocks, but it is no mark about this in table header. |
| select | 416 | 0 | 0 | 381 | Table has a mark that it doesn’t contain data, but blocks are allocated and ready to be filled |
| Insert - select | 416 | 1 | 1 | 381 | Oracle write row into one of the allocated blocks. In table header is a mark were data was written. |
| truncate | 6 | 0 | 0 | 3 | Trancate – table header has a mark that these are no allocated blocks for data of this table. 6 blocks – system data. |

# 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,

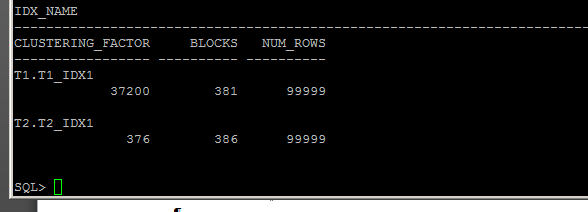
t.num\_rows

FROM user\_indexes i, user\_tables t

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

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

**Task Results:**



Parameter clustering factor indicates the amount of order of the rows in the table based on the values of the index:

• If the value is near the number of blocks, then the table is very well ordered. In

this case, the index entries in a single leaf block tend to point to rows in the

same data blocks.

• If the value is near the number of rows, then the table is very randomly

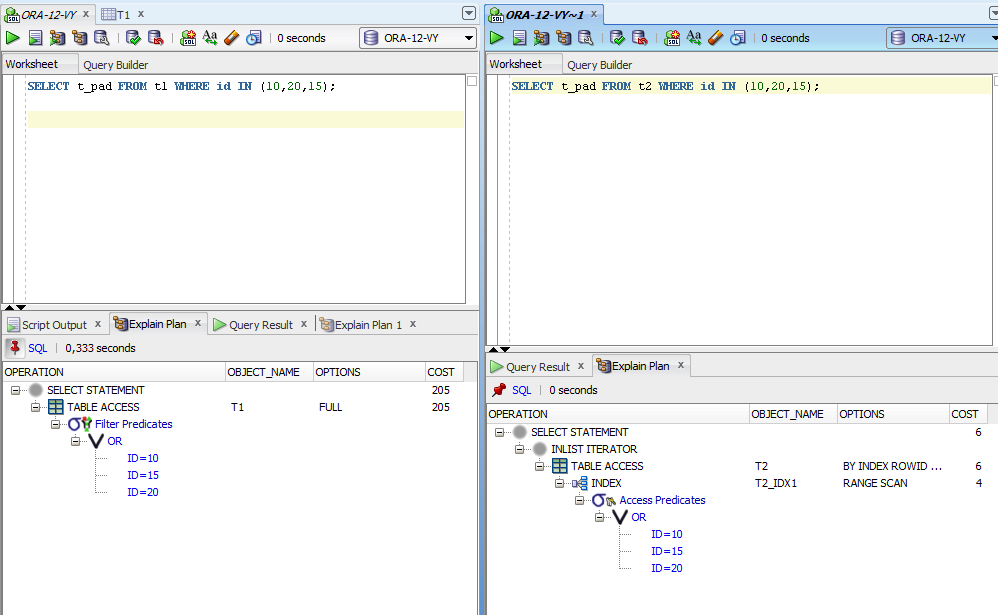
ordered. In this case, it is unlikely that index entries in the same leaf block

point to rows in the same data blocks.

Such delta between clustering factors caused by the different amount of values in column id in tables t1 and t2. Oracle range scans through the index structure, if it discovers the next row in the index is on the same database block as the prior row, it does not perform another I/O to get the table block from the buffer cache. It already has a handle to one and just uses it. However, if the next row is *not* on the same block, then it will release that block and perform another I/O into the buffer cache to retrieve the next block to be processed.

As a result indexes consisting of smaller list of value (in relation to the size of the table) will be more effective.

Screenshot shows the example approving this statement. Index on table t2 is more effective.



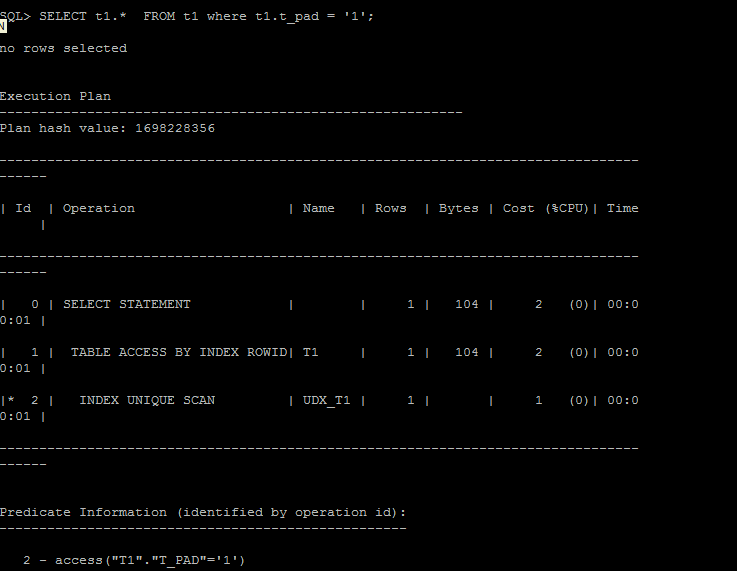
## Task 3: Index Unique Scan

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:

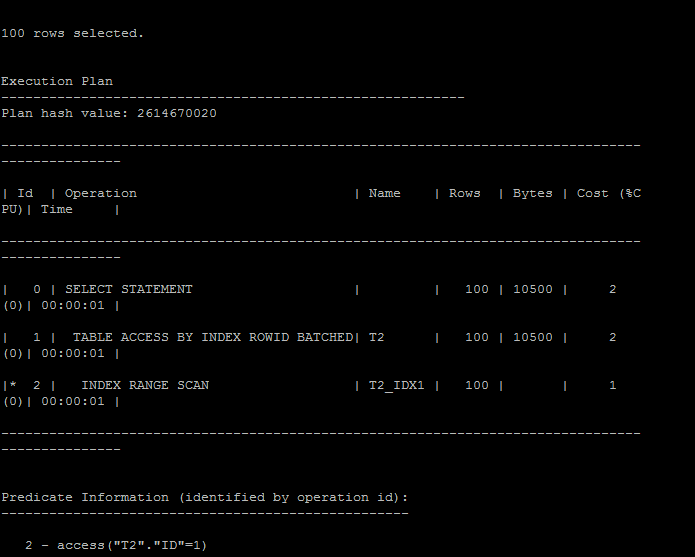
* Screenshot of the step 2;
* Description of process: How oracle read block on step 2;

Oracle uses index udx\_t1 for a unique scan in table T1. Oracle invokes unique index scan and quickly reads every block of the index touching the table itself.

## Task 4: Index Range Scan

**Step 1:**

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



**Task Results:**

Oracle refers the index table first before hitting the actual data table. When oracle searches for the value in this index table, it follows random search (i.e. binary search) to go to the corresponding first required index record. Using this binary search, Oracle is intelligent enough to go to the “first” index record directly in the index table. After that, it does only the sequential search to retrieve the remaining index records since index values are kept in the sorted order so it doesn’t need to go for the random search for the others .

## Task 5: Index Skip Scan

Step 1:

# CREATE TABLE employees AS

SELECT \*

FROM scott.emp;

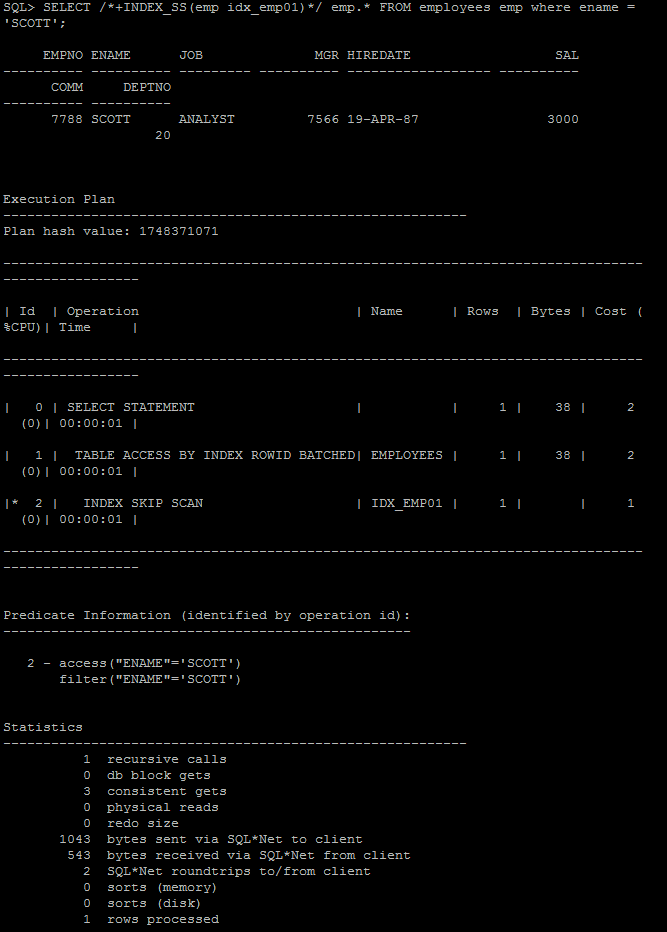
Step 2:

# CREATE INDEX idx\_emp01 ON employees

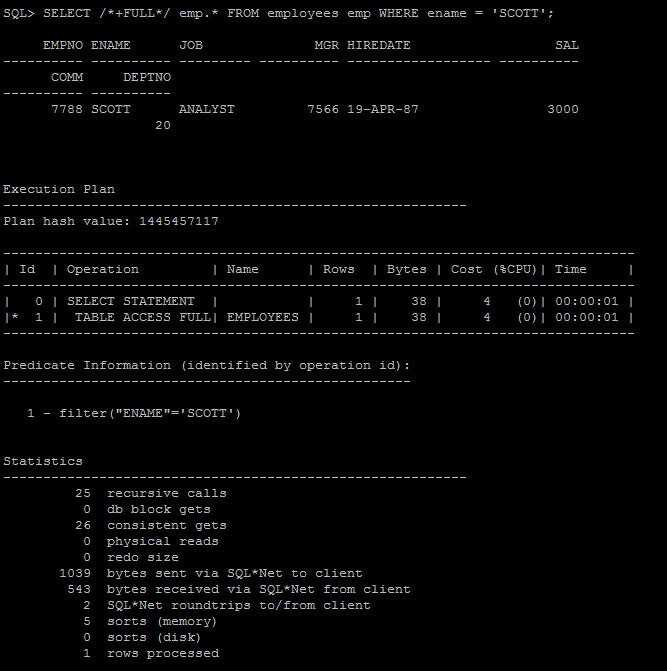
( empno, ename, job );

**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.

|  |  |
| --- | --- |
| Index Skip Scan | The database "skips" through a single index as if it were searching separate indexes. Skip scanning is beneficial if there are few distinct values in the leading column of a composite index and many distinct values in the nonleading key of the index.  Oracle refers the index table first before hitting the actual data table. Since we are looking for the needed column value, oracle hits index table first and get the corresponding ROWIDs.. When oracle searches for the value in this index table, it follows random search (i.e. binary search) to go to the corresponding index records. |
| Full Index Scan | Oracle refers the index table first before hitting the actual data table. Oracle can go for the “full index scan” since ORDER BY clause is not mentioned. After traversing through all the index records in the index table, Oracle outputs all the date from the table according to the ROWID-s from index table. |