# 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 | 1612 | Create + Insert 99999 row: наш Insert устанавливаем HWM. Понадобилось 1612 согласованных чтений из блока (consistent gets). |
| 2 | 1664 | 0 | 0 | 1541 | Delete 99999 row: Как видим после удаления данных HWM не сбрасывается и по прежнему необходимо большое количество чтений. |
| 3 | 1664 | 1 | 1 | 1541 | Insert 1 row: Вставив одну строку HWM также не сбросился, читаем много раз. |
| 4 | 8 | 0 | 0 | 5 | Truncate: Операция truncate сбросила значение HWM, читаем мало раз. |

# 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

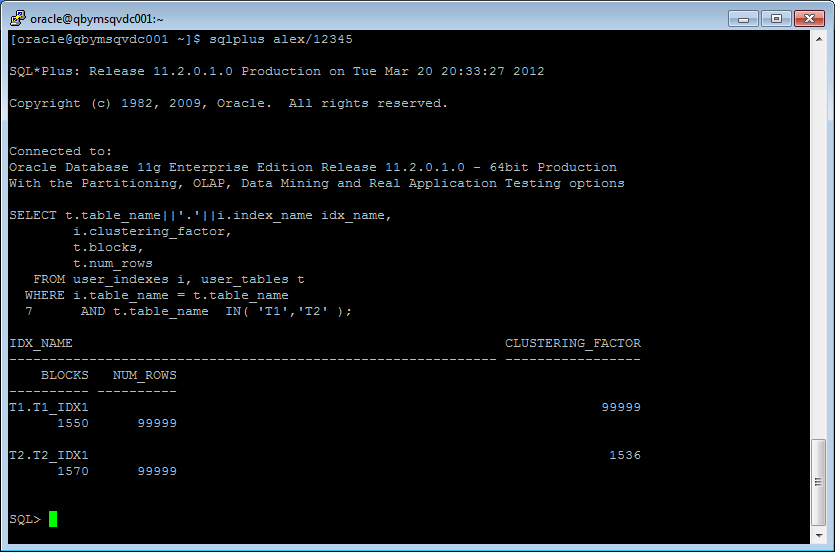
WHERE i.table\_name = t.table\_name

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

**Task Results:**

Expected:

* Screenshot of the step 5;



* Description of the parameter clustering factor;

Значение CLUSTERING FACTOR (показатель кластеризации) - показатель того, насколько упорядочена таблица в соответствии с индексом.

Если значение близко к общему количеству блоков, значит, таблица очень хорошо упорядочена. В этом случае записи индекса в одном листовом блоке обычно указывают на строки, находящиеся в одних и тех же блоках данных.

Если значение близко к общему количеству строк, значит, таблица весьма не упорядочена. В этом случае маловероятно, что записи индекса в одном листовом блоке указывают на те же блоки данных.

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

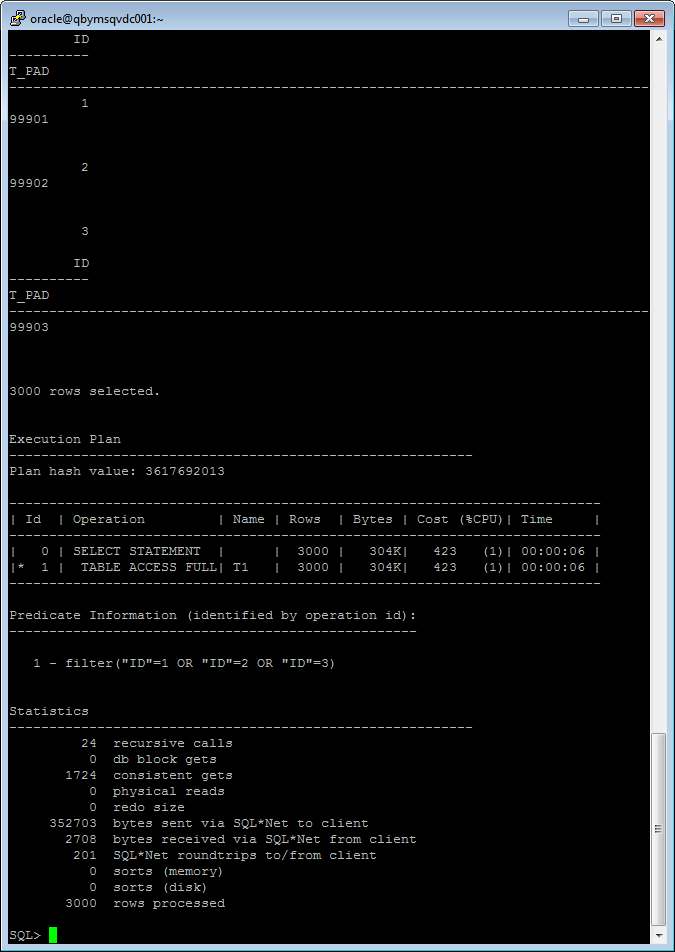
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T1 struture:   |  |  | | --- | --- | | ID | T\_PAD | | 1  2  ..  99 | …  …  …  … | | 1  2  ..  99 | …  …  …  … | | T2 struture:   |  |  | | --- | --- | | ID | T\_PAD | | 0  …  …  0 | …  …  …  … | | 1  …  …  1 | …  …  …  … | |

Как видим таблицы имеют разную структуру. В T2 данные упорядочены в соответствии с ID, в результате чего получили CLUSTERING FACTOR~= Count of Used Blocks => Таблица хорошо упорядочена

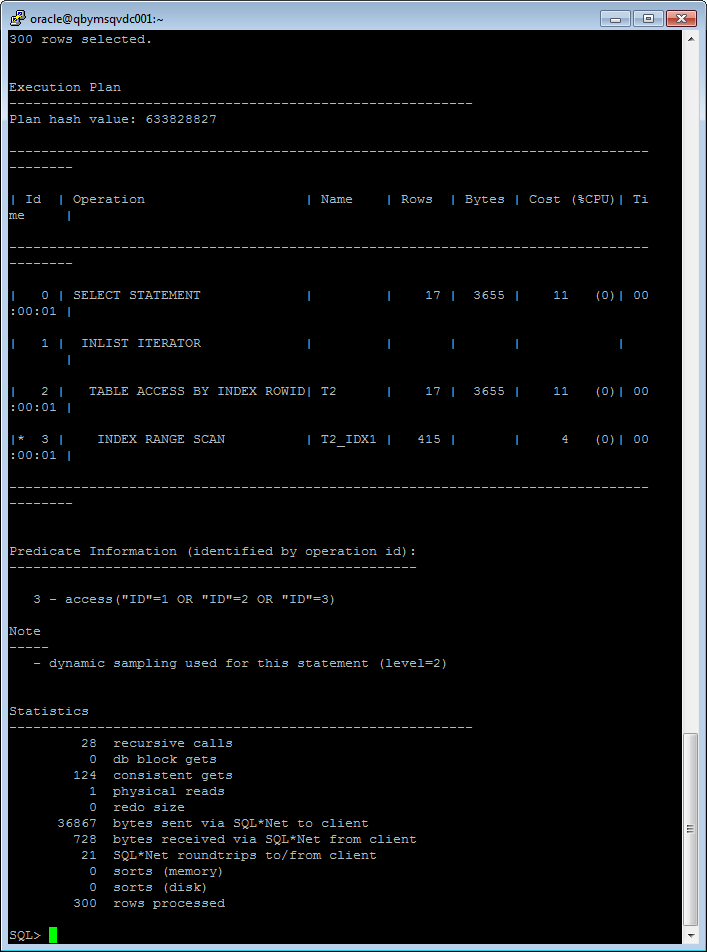
В T1 данные упорядочены другим образом, в результате чего CLUSTERING FACTOR= NUM\_ROWS=> Таблица плохо упорядочена.

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

Select \* from t1 where id in (1,2,3);



Select \* from t2 where id in (1,2,3);



## Task 3: Index Unique Scan

Step 1:

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

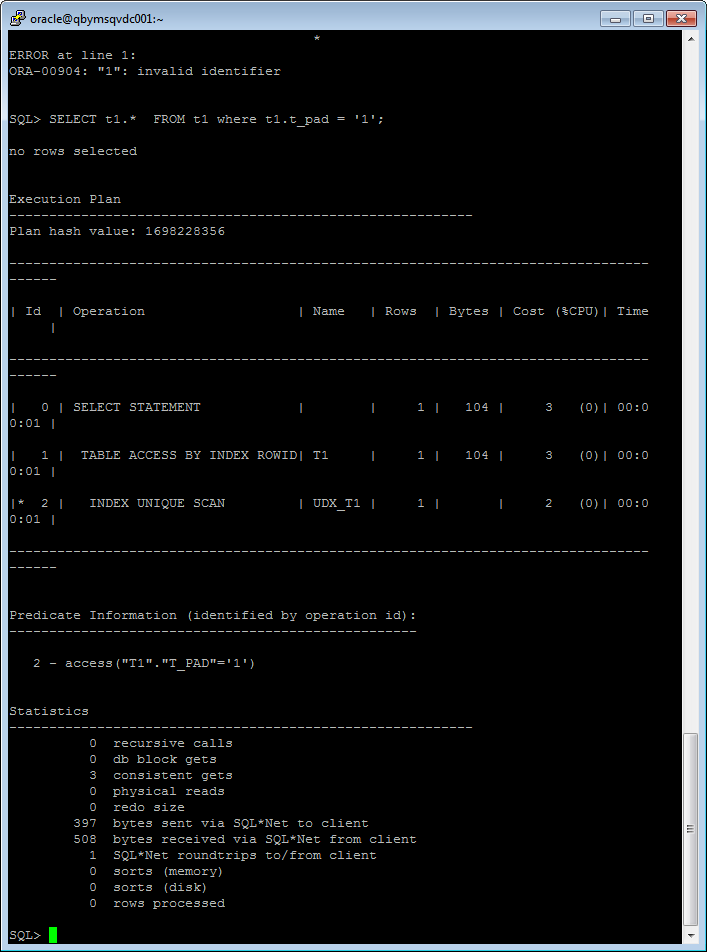
**Step 2**

# 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;

Метод поиска единственного значения через уникальный индекс. Всегда возвращается одно значение.

## Task 4: Index Range Scan

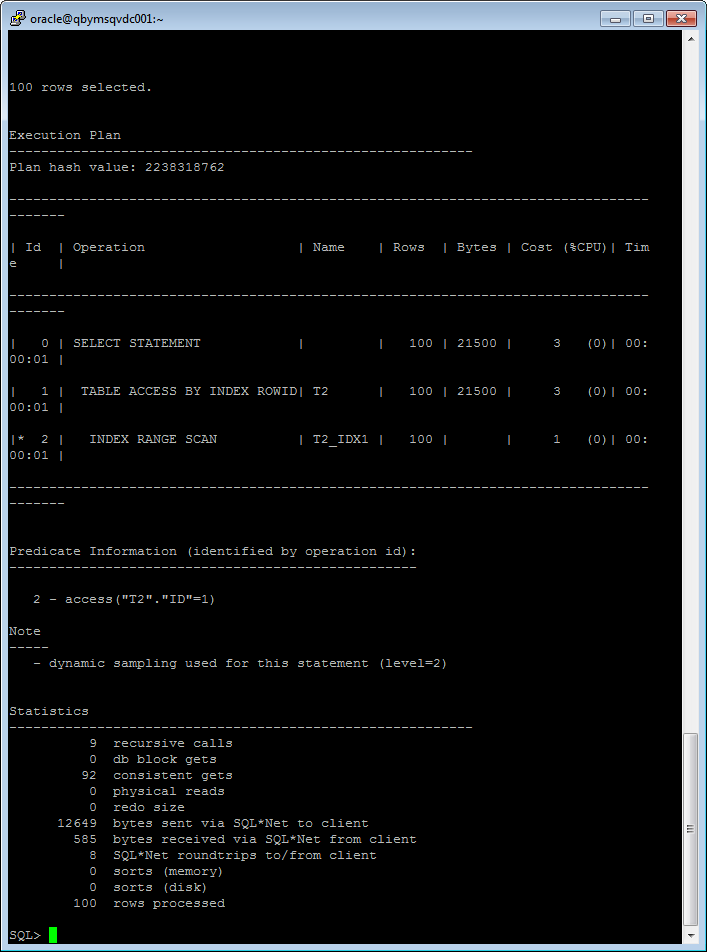
**Step 1:**

# 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;

Метод применяется для поиска множества значений при помощи индекса. Используется при поиске диапазонов значений при помощи SQL операторов between, >, <, <>, >=, <=. Для неуникального индекса может выдавать множество значений и для условия равенства.

## 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 );

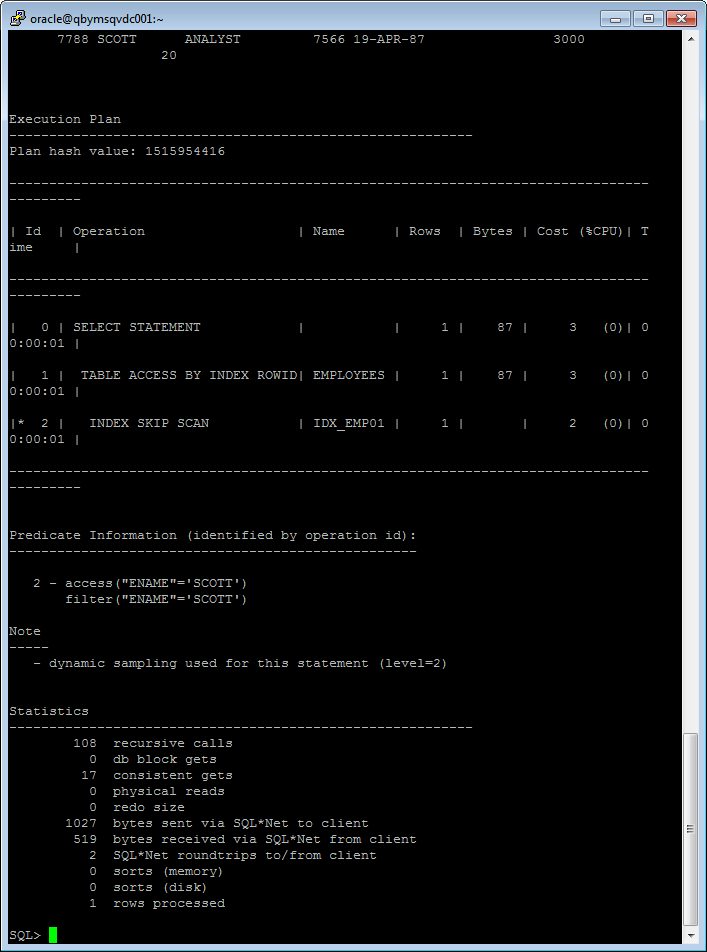
**Step 3:**  Get trace and statistic of explain plan

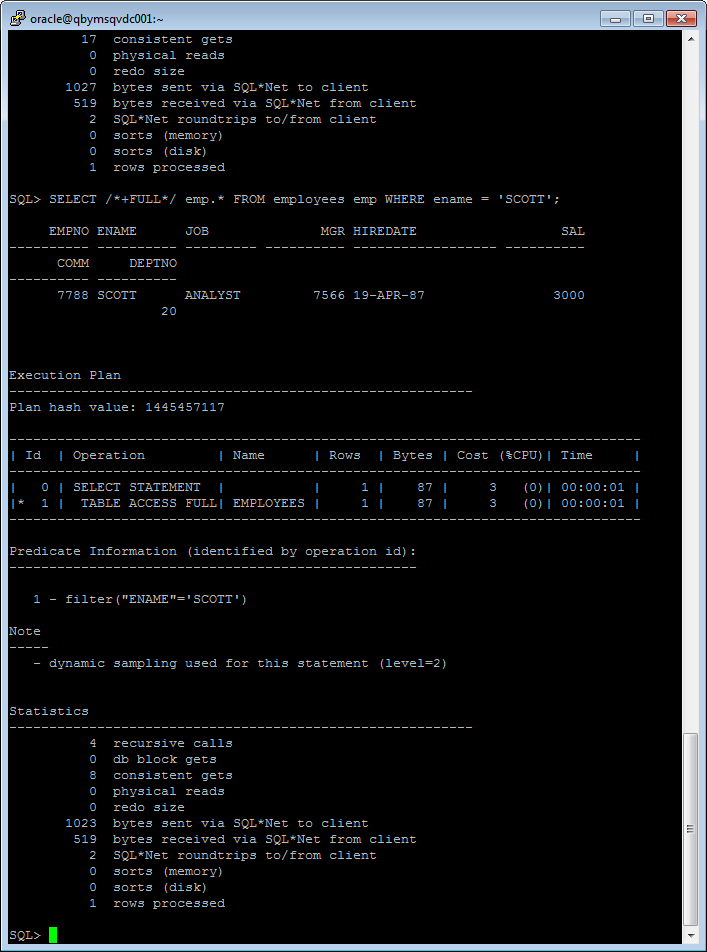
# SELECT /\*+INDEX\_SS(emp idx\_emp01)\*/ emp.\* FROM employees emp where ename = 'SCOTT';

# 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;

NDEX SKIP SCAN, который позволяет использовать составной индекс для поиска данных, даже если в условиях поиска отсутствует лидирующий столбец.

Практически, выполняется INDEX RANGE SCAN для каждого различного значения лидирующего столбца.

INDEX SKIP SCAN эффективен в тех случаях, когда лидирующий столбец имеет минимальное количество различных значений. INDEX SKIP SCAN - это не идеальное и эффективнейшее решение для всех поисков по составному ключу без лидирующего столбца.

* 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 | 1516 | 0 | 3 | Index Unique Scan |
| 2 | 1664 | 1536 | 100 | 92 | Range index |
| 3 | 8 | 1 | 1 | 17 | Index Skip Scan (/\*+INDEX\_SS) |
| 4 | 8 | 1 | 1 | 8 | Index Skip Scan (/\*+FULL\*/) |