Report

Laboratory Work 4

Dmitry Ladutsko

July 19, 2022

## 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;



Picture 1.1 – Creation Result

Step 2:

CREATE INDEX t2\_idx1 ON t2 ( id );



Picture 1.2 – Creation Result

Step 3:

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



Picture 1.3 - Query Result

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



Picture 1.4 - Query Result

SET autotrace ON;



Picture 1.5 - Result

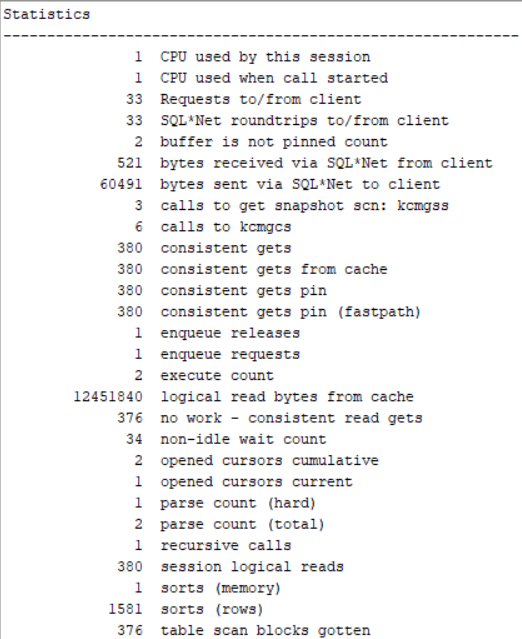
SELECT COUNT( \* )

FROM t2 ;

Text

Description automatically generated

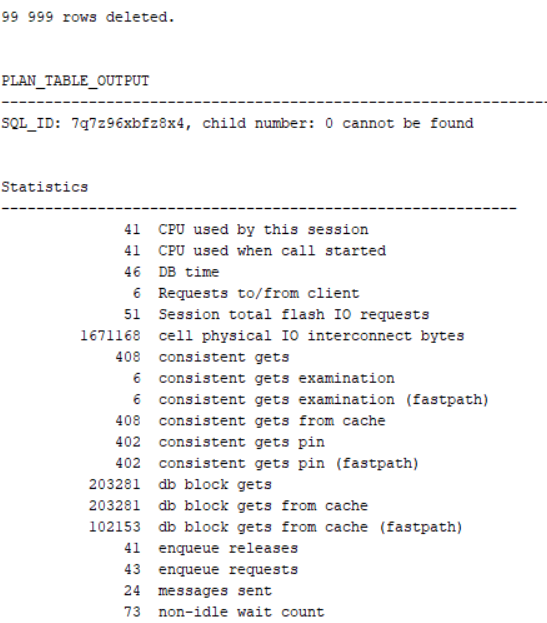
Picture 1.6 – Quantity of Rows



Picture 1.7 - Statistics

Step 4:

DELETE FROM t2;

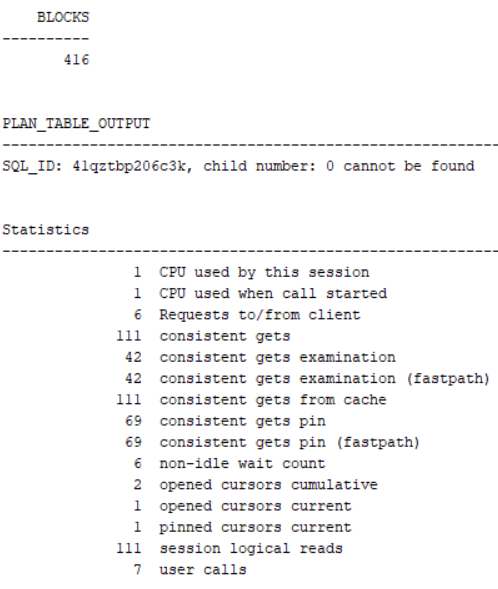


Picture 1.8 – Deleting and Statistics

Step 5:

--1

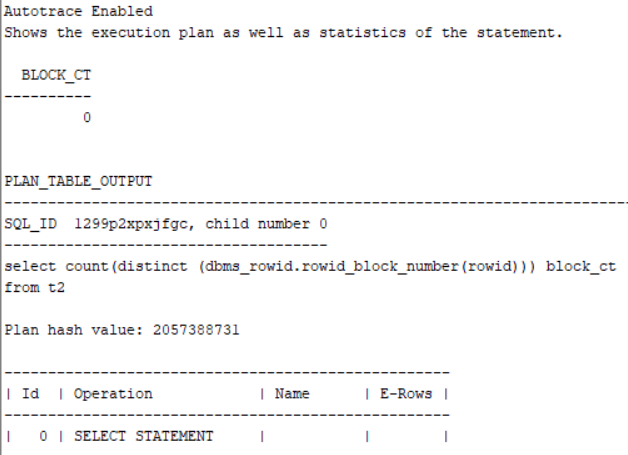
select blocks from user\_segments where segment\_name = 'T2';



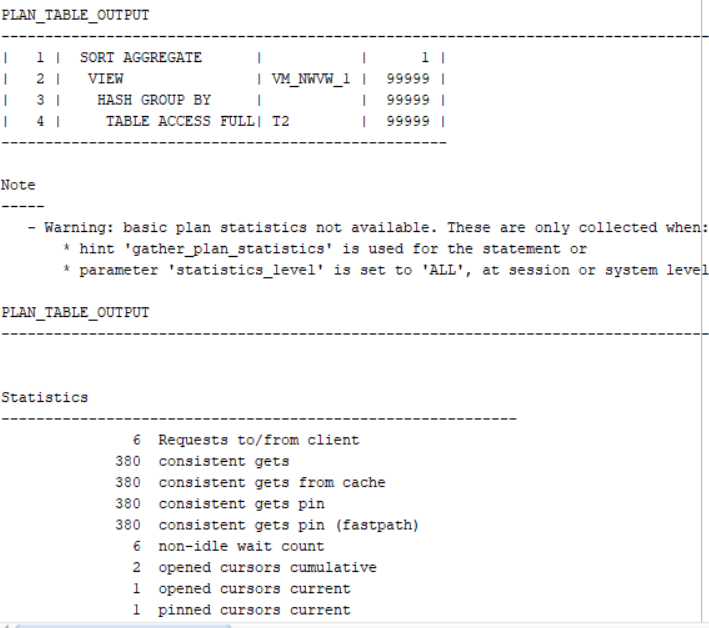
Picture 1.9 – Count of Blocks

--2

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



Picture 1.10 – Count of Used Blocks



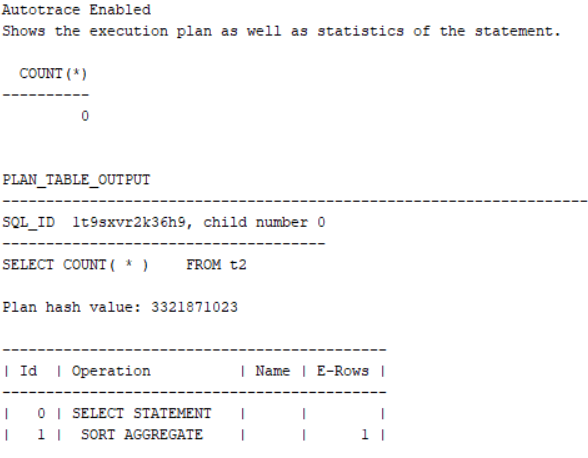
Picture 1.11 – Quantity of Consistent Gets

--3

SELECT COUNT( \* )

FROM t2 ;

Graphical user interface, text

Description automatically generated

Picture 1.12 – Quantity of Consistent Gets and Count of Rows

Step 6:

INSERT INTO t2

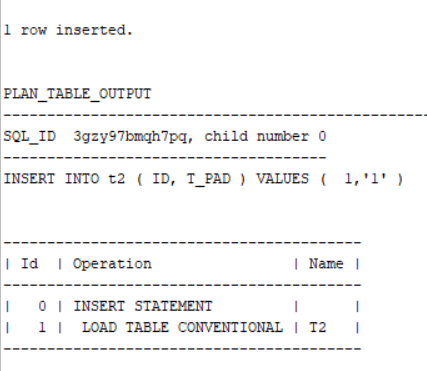
( ID, T\_PAD )

VALUES

( 1,'1' );

COMMIT;

Table

Description automatically generatedText

Description automatically generated

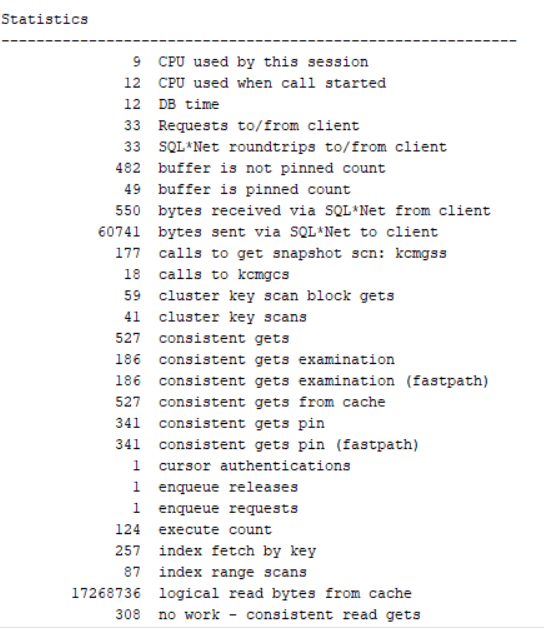
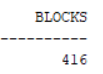
Picture 1.12 – Insertion

Step 7:

--1

SET autotrace ON;

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

Table

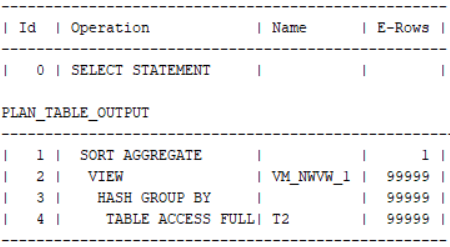
Description automatically generated with low confidence

Picture 1.13 – Count of Blocks

--2

SET autotrace ON;

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

Graphical user interface, text

Description automatically generated

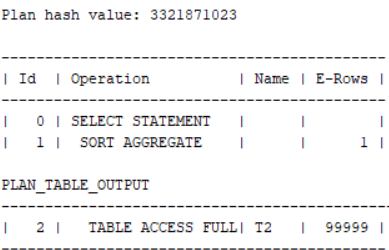
Picture 1.14 - Count of Used Blocks

--3

SET autotrace ON;

SELECT COUNT( \* )

FROM t2 ;

Text

Description automatically generated with medium confidence

Picture 1.15 – Quantity of Consistent Gets and Count of Rows

Step 8:

TRUNCATE TABLE t2;

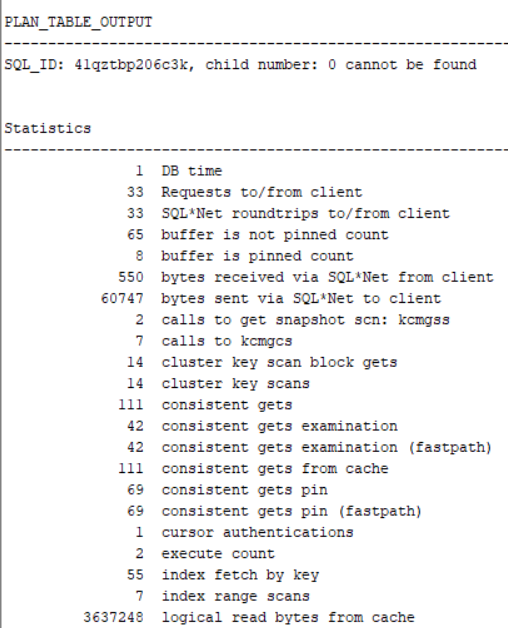


Step 9:

--1

SET autotrace ON;

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

Text

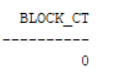
Description automatically generated

Picture 1.16 – Count of Blocks

--2

SET autotrace ON;

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

Text

Description automatically generatedTable

Description automatically generated

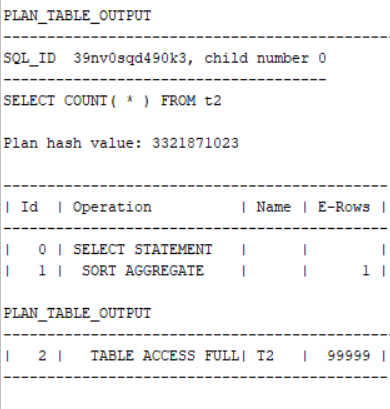
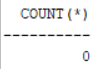
Picture 1.17 - Count of Used Blocks

--3

SET autotrace ON;

SELECT COUNT( \* )

FROM t2 ;

Text

Description automatically generated with medium confidence

Picture 1.18 – Quantity of Consistent Gets and Count of Rows

Task Result:

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 416 | 376 | 99999 | 380 | Table was filled into 416 blocks, 376 of which were used (40 unused). 380 consistent get operations. Maximal number of rows - 99999 |
| 2 | 416 | 0 | 0 | 380 | We used DELETE operation but still have an opportunity to roll back deleted data. DELETE just saves allocated space and stats, so the number of consistent gets did not change after as you can see below 380 -> 380 |
| 3 | 416 | 1 | 1 | 380 | Even if we INSERT 1 row (or 2,3,4 … 99999) |
| 4 | 6 | 0 | 0 | 1 | We used TRANCATE operator which truncate all the allocated data and stats (w/o an opportunity to roll back data), but oracle still leave 6 blocks to store an information about table |

## Task 2: Index Clustering factor parameter

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 Select clause filtered by IN ( , list of values, );

Step 1***:***

CREATE TABLE t2 AS

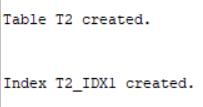
SELECT TRUNC( rownum / 100 ) id, rpad( rownum,100 ) t\_pad

FROM dual

CONNECT BY rownum < 100000;

CREATE INDEX t2\_idx1 ON t2

( id );



Picture 2.1 – Table and Index creating

Step 2***:***

# 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

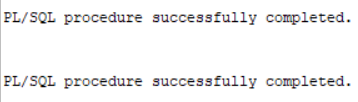
( id );



Step 4***:***

# 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 );



Picture 2.2 – Procedure execution

Step 5:

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

i.clustering\_factor,

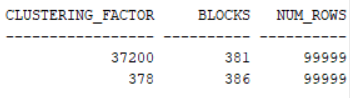
t.blocks,

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**:**

* The clustering factor indicates the amount of order of the rows in the table based on the values ​​of the index. At the screenshot above we can see that clustering factor **approaches the number of blocks** in the table 378 ~ 386. That means that the rows are ordered. If it approaches the number of rows in the table, the rows are randomly ordered. In such a case (clustering factor near the number of rows), it is unlikely that index entries in the same leaf block will point to rows in the same data blocks. If all of the index entries in a given leaf block point to different blocks in the table – the table is **not well ordered** with respect to this index. If we want index being **strong** clustered – we should consider using Index Organized Tables (**IOT**). They force the rows into a specific **physical location** based on their index entry. Even though the correct set up of clustering anyway increases performance and decreases cost.
* Because we used MOD operand in creation of table t1 instead of TRUNC in t2.
* Clustering factor increments if rawid points to different table blocks.

## Task 3: Index Unique Scan

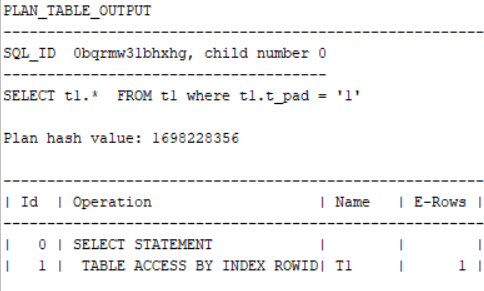
Step 1:

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



Step 2

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

*Text

Description automatically generated*

Picture 3.1 - Query Result

*Note.* Select in 3 task is empty because ‘1’ is a char, but a column type is number.

## Task 4: Index Range Scan

Step 1:

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

A picture containing chart

Description automatically generated

Picture 4.1 - Query Result

## Task 5: Index Skip Scan

I found scott schema in the internet. Separate file named ‘scottschema’ will be attached.

Step 1:

CREATE TABLE employees AS

SELECT \*

FROM scott.emp;



Step 2:

CREATE INDEX idx\_emp01 ON employees

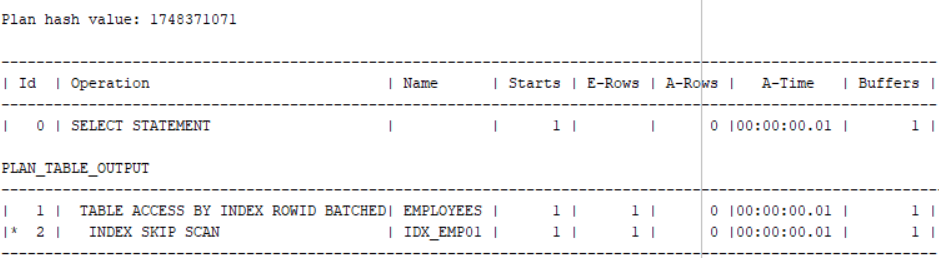
( empno, ename, job );

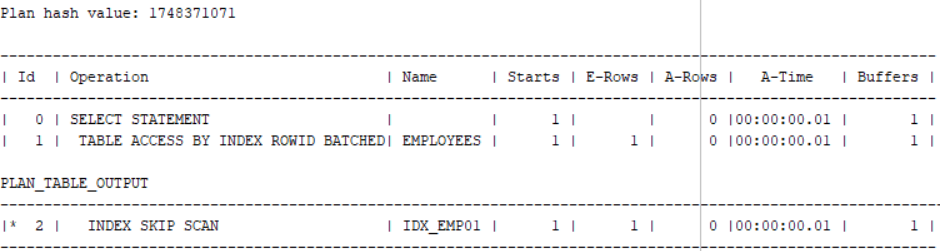


Step 3:

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

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





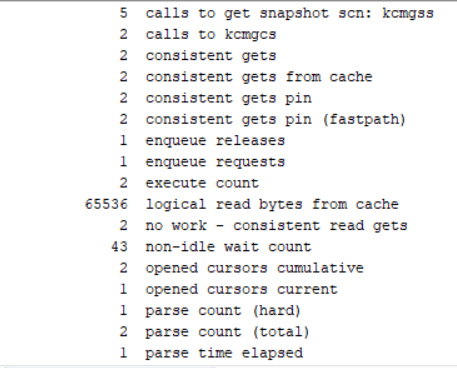
*Note.* Oracle scans the index sub-trees for each of the possible values from leading column.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 6 | 1 | 1 | 2 | Skip scan because optimizer decided it is more sutable for this SELECT |
|  | 6 | 1 | 1 | 2 | /\*+INDEX\_SS(emp idx\_emp01)\*/ (cause of hint) |
|  | 6 | 1 | 1 | 2 | /\*+FULL\*/ (cause of hint) |

SELECT BLOCKS FROM user\_segments WHERE segment\_name = 'EMPLOYEES';



SELECT COUNT(DISTINCT (dbms\_rowid.rowid\_block\_number(rowid))) block\_ct FROM EMPLOYEES ;



Review

**Laboratory Work Summary:**

In this laboratory work we have learned and practiced about how Oracle uses different techniques to access the data needed for the query answer. Access Methods can be divided into two categories: data accessed via a table scan or index access. There are some description below about what methods have we used to access the data at this lab:

**Full Scan:**

A full table scan reads all rows from a table, and then filters out those rows that do not meet the selection criteria.

In a full table scan, the database sequentially reads every formatted block under the high water mark. The database reads each block only once.

**Clustering factor:**

The clustering factor is a number that represents the degree to which data is randomly distributed in a table as compared to the indexed column. In simple terms, it is the number of “block switches” while reading a table using an index.

**Index Unique Scan:**

This scan returns, at most, a single rowid. Oracle performs a unique scan if a statement contains a UNIQUE or a PRIMARY KEY constraint that guarantees that only a single row is accessed.

**Index Range Scan:**

An index range scan is a common operation for accessing selective data.  It can be bounded (bounded on both sides) or unbounded (on one or both sides). Data is returned in the ascending order of index columns. Multiple rows with identical values are sorted in ascending order by rowid.

**Index Skip Scan:**

Index skip scans improve index scans by non-prefix columns. Often, scanning index blocks is faster than scanning table data blocks. Skip scanning lets a composite index be split logically into smaller sub-indexes. In skip scanning, the initial column of the composite index is not specified in the query. In other words, it is skipped. The number of logical sub-indexes is determined by the number of distinct values in the initial column. Skip scanning is advantageous if there are few different values in the leading column of the composite index and many distinct values in the non-leading key of the index.