# Table access full scan

# CREATE TABLE t2 AS

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

# FROM dual

# CONNECT BY rownum < 100000;

# COMMIT;

# CREATE INDEX t2\_idx1 ON t2

# ( id );

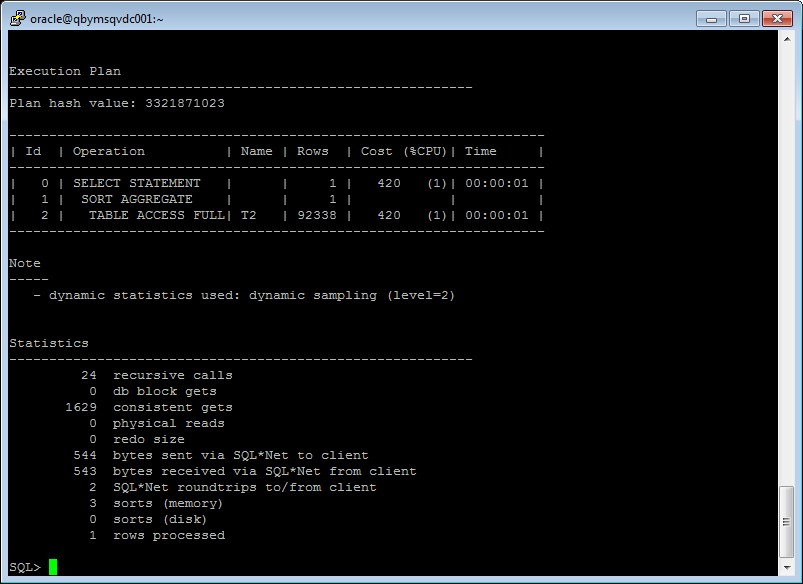
# select blocks from user\_segments where segment\_name = 'T2';



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



SELECT COUNT( \* ) FROM t2 ;



DELETE FROM t2;

COMMIT;

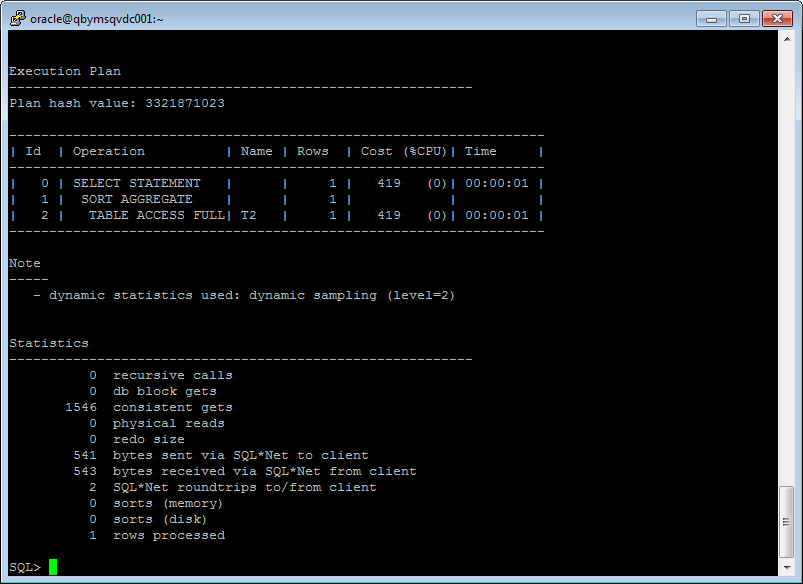
# select blocks from user\_segments where segment\_name = 'T2';



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



SELECT COUNT( \* ) FROM t2 ;



INSERT INTO t2

( ID, T\_PAD )

VALUES

( 1,'1' );

COMMIT;

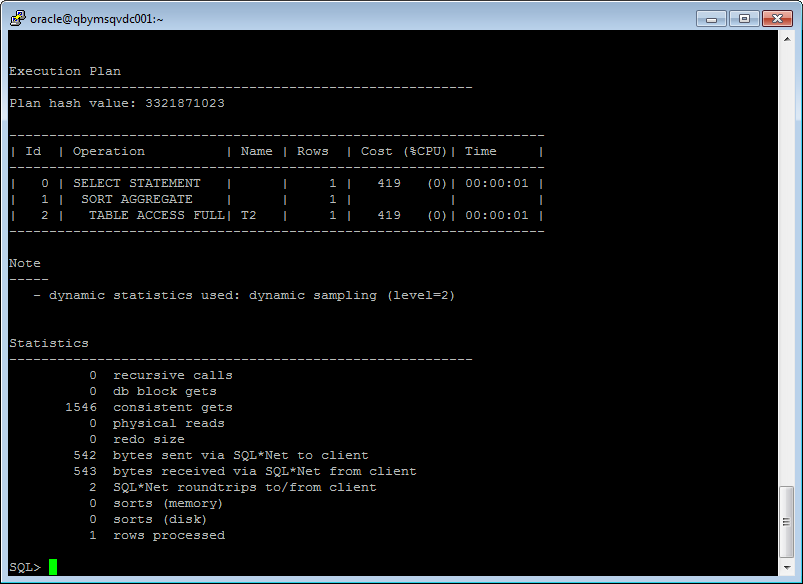
# select blocks from user\_segments where segment\_name = 'T2';



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



SELECT COUNT( \* ) FROM t2 ;



TRUNCATE TABLE t2;

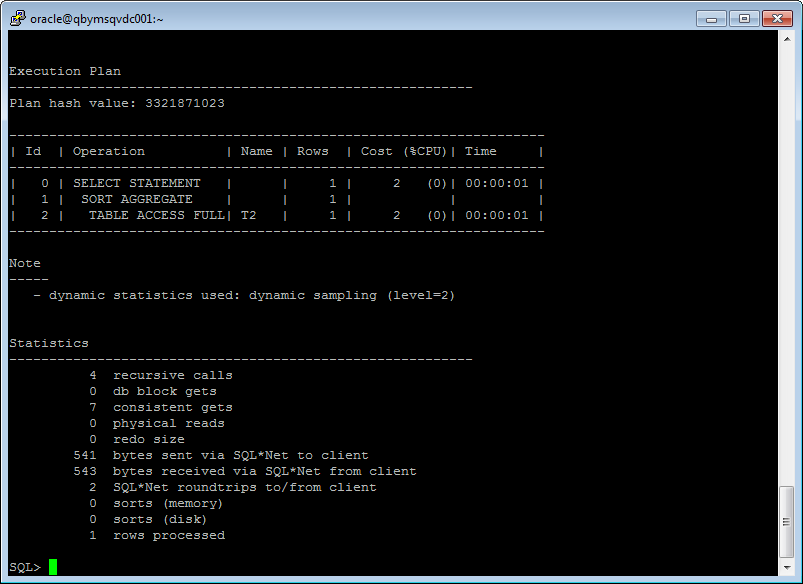
# select blocks from user\_segments where segment\_name = 'T2';



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



SELECT COUNT( \* ) FROM t2 ;



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 1664 | 1536 | 99999 | 1629 | The consistent gets Oracle metric is the number of times a consistent read (a logical RAM buffer I/O) was requested to get data from a data block. We inserted 99999 rows into the table. For this required 1536 blocks, but oracle reserved 1664 block for the table, because the segment size increases discretely. |
| 2 | 1664 | 0 | 0 | 1546 | After deleting and committing rows from t2 are still in their places in blocs and the table still has links with this blocks. In spite of the fact that the table is seemed to be empty, the request will be generated as for table, filled with data. |
| 3 | 1664 | 1 | 1 | 1546 | The request will be generated as for table, filled with data because High Water Mark are at the same place (As at the previous situation). |
| 4 | 8 | 0 | 0 | 7 | The “truncate” command moves the High Water Mark of the table back to zero. |

# Index Scan types

CREATE TABLE t2 AS

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

FROM dual

CONNECT BY rownum < 100000;

COMMIT;

CREATE INDEX t2\_idx1 ON t2

( id );

CREATE TABLE t1 AS

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

FROM dual

CONNECT BY rownum < 100000;

COMMIT;

CREATE INDEX t1\_idx1 ON t1

( id );

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 );

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

i.clustering\_factor,

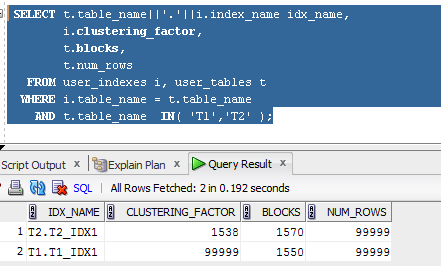
t.blocks,

t.num\_rows

FROM user\_indexes i, user\_tables t

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

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



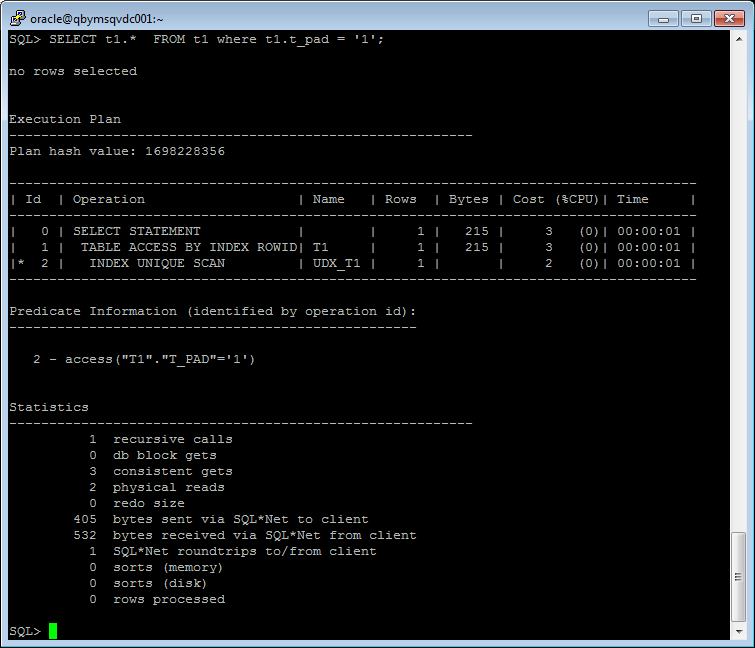
Фактор кластеризации указывает оптимизатору как расположены строки, содержащие индексированные значения, - в небольшом наборе смежных блоков или же разбросаны по многочисленным блокам таблицы.

Оптимизатор выбирает index scan при работе с таблицей T2, но full table scan при выполнении запроса к таблице T1. Это происходит потому, что фактор кластеризации для таблицы Т1 значительно больше, фактора кластеризации для таблицы Т2.

Фактор кластеризации – это главный параметр при определении оптимизатором способа чтения таблиц.

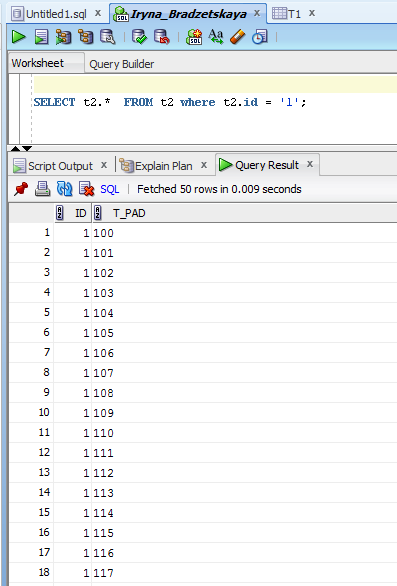
Индекс таблицы Т2 имеет лучшую производительность при исполнении Select clause.

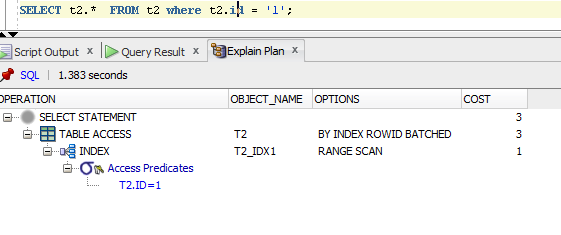
# Index Unique Scan



Изначально unique index был создан для столбца t\_pad. В запросе использовался фильтр для t\_pad на одно конкретное значение. Оптимизатор решил использовать именно Index Unique Scan для поиска rowid через дерево индексов, а также для доступа к данным таблицы по этому rowid.

# Index Range Scan





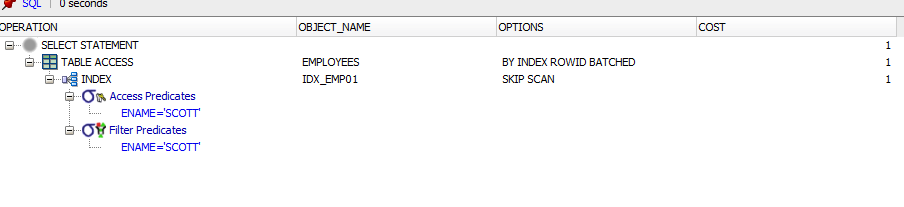
Для столбца id был создан неуникальный индекс. Однако в запросе присутствует фильтр на id для конкретного значения. Таким образом, оптимизатор решил использовать Index Range Scan. Оптимизатор выполняет поиск блоков id через дерево индекса, а также предоставляет доступ к данным таблицы по найденным rowid.

# Index Skip Scan

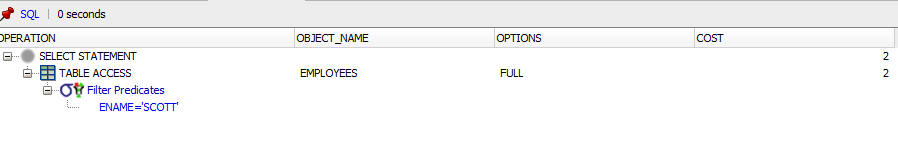
CREATE INDEX idx\_emp01 ON employees

( empno, ename, job );

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



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



|  |  |  |  |
| --- | --- | --- | --- |
| № | Count of Rows | Consistent gets | Description |
| 1 | 1 | 3 | Skip Scan Этот запрос работает лучше. Это происходит потому, что не нужно использовать full scan. В этом случае происходит логическое расщепление индекса на несколько субиндексов. Количество логических субиндексов равно количеству проиндексированных столбцов таблицы. |
| 2 | 1 | 4 | Full Scan  Этот запрос работает медленнее. Это происходит потому, что при поиске просматриваются все строки таблицы. |