# Table access

## Task 1: Full Scan, High-Water Mark and Consistent Gets

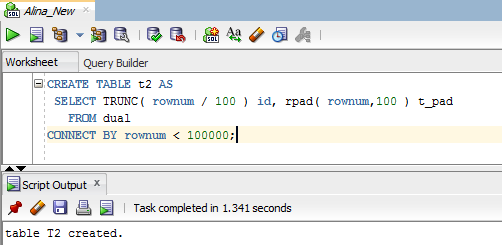
### Step 1 создание таблицы

CREATE TABLE t2 AS

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

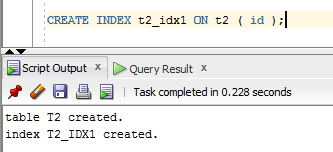
FROM dual

CONNECT BY rownum < 100000;



### Step 2 создание индекса для таблицы.

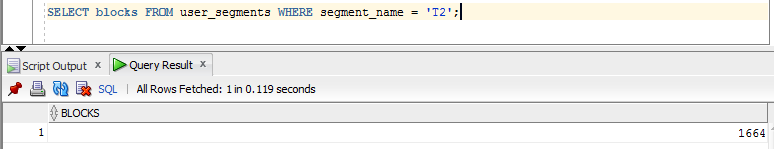
CREATE INDEX t2\_idx1 ON t2 ( id );



### Step 3

Number of blocks used by segment:

SELECT blocks FROM user\_segments WHERE segment\_name = 'T2';

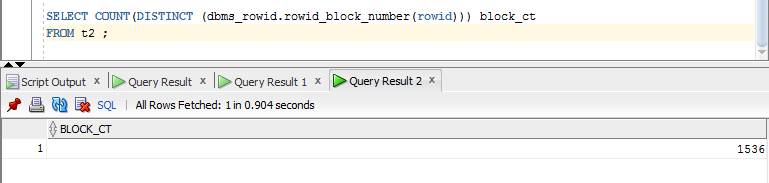


Общее количество блоков,предназначенное для данного сегмента для данной таблицы.

Number of blocks used by real data:

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

FROM t2 ;



Общее количество блоков, которые используются для данных таблицы.

Use autotrace feature of SQL Developer (F6):

SELECT COUNT( \* ) FROM t2;

|  |  |
| --- | --- |
| consistent gets | 1539 |
| consistent gets direct | 1536 |
| consistent gets from cache | 3 |
| consistent gets pin | 3 |
| consistent gets pin (fastpath) | 3 |
| CPU used by this session | 1 |
| CPU used when call started | 2 |
| logical read bytes from cache | 24576 |
| no work - consistent read gets | 1536 |
| Number of read IOs issued | 53 |
| opened cursors cumulative | 2 |
| parse count (total) | 2 |
| physical read bytes | 12582912 |
| physical read IO requests | 53 |
| physical read total bytes | 12582912 |
| physical read total IO requests | 53 |
| physical read total multi block requests | 45 |
| physical reads | 1536 |
| physical reads direct | 1536 |
| Requests to/from client | 25 |
| session logical reads | 1539 |
| session pga memory | -393216 |
| sorts (memory) | 2 |
| sorts (rows) | 2356 |
| SQL\*Net roundtrips to/from client | 25 |
| table scan blocks gotten | 1536 |
| table scan disk non-IMC rows gotten | 99999 |
| table scan rows gotten | 99999 |
| table scans (direct read) | 1 |
| table scans (short tables) | 1 |
| user calls | 27 |
| workarea memory allocated | 261 |

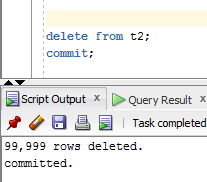
Мои результаты:

|  |  |
| --- | --- |
| bytes received via SQL\*Net from client | 505 |
| bytes sent via SQL\*Net to client | 47708 |
| calls to get snapshot scn: kcmgss | 3 |
| calls to kcmgcs | 5 |
| cell physical IO interconnect bytes | 12582912 |
| consistent gets | 1539 |
| consistent gets direct | 1536 |
| consistent gets from cache | 3 |
| consistent gets pin | 3 |
| consistent gets pin (fastpath) | 3 |
| CPU used by this session | 3 |
| CPU used when call started | 3 |
| DB time | 4 |
| enqueue releases | 1 |
| enqueue requests | 1 |
| execute count | 2 |
| file io wait time | 3476 |
| logical read bytes from cache | 24576 |
| no work - consistent read gets | 1536 |
| non-idle wait count | 56 |
| Number of read IOs issued | 31 |
| opened cursors cumulative | 2 |
| parse count (hard) | 1 |
| parse count (total) | 2 |
| physical read bytes | 12582912 |
| physical read IO requests | 31 |
| physical read total bytes | 12582912 |
| physical read total IO requests | 31 |
| physical read total multi block requests | 23 |
| physical reads | 1536 |
| physical reads direct | 1536 |
| recursive calls | 1 |
| Requests to/from client | 25 |
| session cursor cache hits | 1 |
| session logical reads | 1539 |
| session pga memory | 917504 |
| sorts (memory) | 2 |
| sorts (rows) | 2356 |
| SQL\*Net roundtrips to/from client | 25 |
| table scan blocks gotten | 1536 |
| table scan disk non-IMC rows gotten | 99999 |
| table scan rows gotten | 99999 |
| table scans (direct read) | 1 |
| table scans (short tables) | 1 |
| user calls | 27 |
| workarea executions - optimal | 5 |

### Step 4

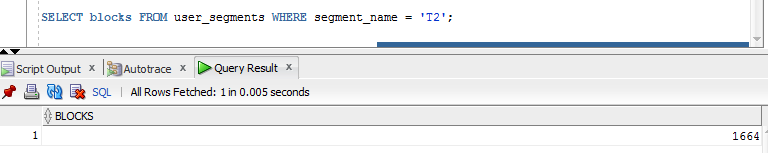
Delete all rows from the table:

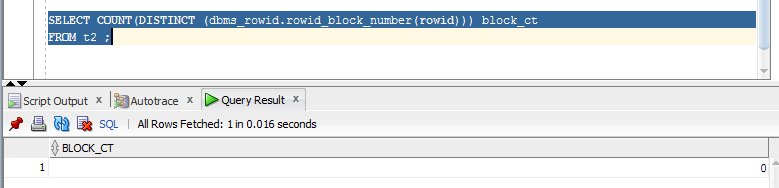
DELETE FROM t2;



### Step 5

Repeat Step 3 and collect results.





Результаты:

|  |  |
| --- | --- |
| bytes received via SQL\*Net from client | 505 |
| bytes sent via SQL\*Net to client | 47709 |
| calls to get snapshot scn: kcmgss | 3 |
| calls to kcmgcs | 7 |
| consistent gets | 1541 |
| consistent gets from cache | 1541 |
| consistent gets pin | 1541 |
| consistent gets pin (fastpath) | 1541 |
| CPU used by this session | 4 |
| CPU used when call started | 4 |
| DB time | 3 |
| enqueue releases | 1 |
| enqueue requests | 1 |
| execute count | 2 |
| logical read bytes from cache | 12623872 |
| no work - consistent read gets | 1536 |
| non-idle wait count | 25 |
| opened cursors cumulative | 2 |
| parse count (hard) | 1 |
| parse count (total) | 2 |
| recursive calls | 1 |
| Requests to/from client | 25 |
| session cursor cache hits | 1 |
| session logical reads | 1541 |
| sorts (memory) | 2 |
| sorts (rows) | 2356 |
| SQL\*Net roundtrips to/from client | 25 |
| table scan blocks gotten | 1536 |
| table scan disk non-IMC rows gotten | 99999 |
| table scan rows gotten | 99999 |
| table scans (short tables) | 1 |
| user calls | 27 |
| workarea executions - optimal | 5 |

### Step 6

Insert 1 row:

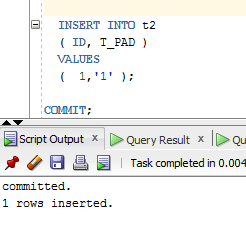
INSERT INTO t2

( ID, T\_PAD )

VALUES

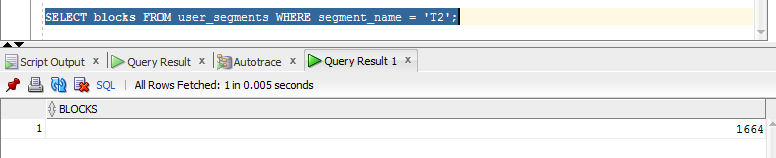
( 1,'1' );

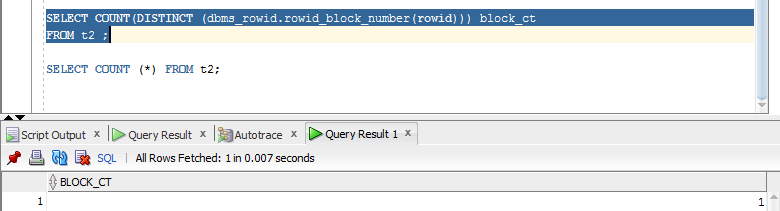
COMMIT;



### Step 7

Repeat Step 3 and collect results.





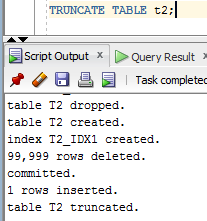
Результаты:

|  |  |
| --- | --- |
| bytes received via SQL\*Net from client | 505 |
| bytes sent via SQL\*Net to client | 47711 |
| calls to get snapshot scn: kcmgss | 3 |
| calls to kcmgcs | 7 |
| consistent gets | 1541 |
| consistent gets from cache | 1541 |
| consistent gets pin | 1541 |
| consistent gets pin (fastpath) | 1541 |
| CPU used by this session | 4 |
| CPU used when call started | 4 |
| DB time | 4 |
| enqueue releases | 1 |
| enqueue requests | 1 |
| execute count | 2 |
| logical read bytes from cache | 12623872 |
| no work - consistent read gets | 1536 |
| non-idle wait count | 25 |
| opened cursors cumulative | 2 |
| parse count (hard) | 1 |
| parse count (total) | 2 |
| recursive calls | 1 |
| Requests to/from client | 25 |
| session cursor cache hits | 1 |
| session logical reads | 1541 |
| sorts (memory) | 2 |
| sorts (rows) | 2356 |
| SQL\*Net roundtrips to/from client | 25 |
| table scan blocks gotten | 1536 |
| table scan disk non-IMC rows gotten | 100000 |
| table scan rows gotten | 100000 |
| table scans (short tables) | 1 |
| user calls | 27 |

### Step 8

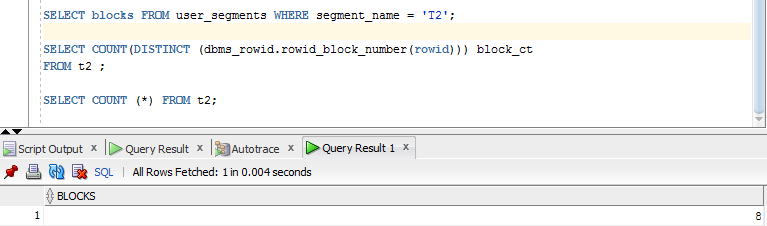
Truncate table:

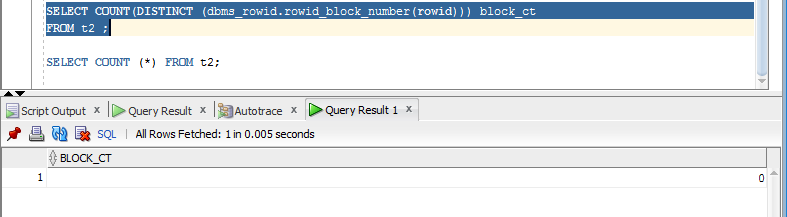
TRUNCATE TABLE t2;



### Step 9

Repeat Step 3 and collect results.





Результаты:

|  |  |
| --- | --- |
| bytes received via SQL\*Net from client | 505 |
| bytes sent via SQL\*Net to client | 47702 |
| calls to get snapshot scn: kcmgss | 3 |
| calls to kcmgcs | 5 |
| consistent gets | 3 |
| consistent gets from cache | 3 |
| consistent gets pin | 3 |
| consistent gets pin (fastpath) | 3 |
| CPU used by this session | 3 |
| CPU used when call started | 3 |
| DB time | 4 |
| enqueue releases | 1 |
| enqueue requests | 1 |
| execute count | 2 |
| logical read bytes from cache | 24576 |
| non-idle wait count | 25 |
| opened cursors cumulative | 2 |
| parse count (hard) | 1 |
| parse count (total) | 2 |
| recursive calls | 1 |
| Requests to/from client | 25 |
| session cursor cache hits | 1 |
| session logical reads | 3 |
| sorts (memory) | 2 |
| sorts (rows) | 2356 |
| SQL\*Net roundtrips to/from client | 25 |
| table scans (short tables) | 1 |
| user calls | 27 |
| workarea executions - optimal | 5 |
| workarea memory allocated | 24 |

### Task Result

Expecting:

* Tasks’ results and your analysis of these results in the description column.

Consistent gets показывает количество операций ввода-вывода.

Count of blocks показывает выделенное количество блоков под данный сегмент

Count of Used Blocks показывает сколько блоков содержат данныею

Count of Rows показывает сколько строк содержиться.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 1664 | 1536 | 99999 | 1539 | Создание таблицы и вставки в нее 99999 строк, здесь же устанавливается значение HWM на уровне последненго использованного блока. Т.е. при выборке Oracle придется читать всю таблицу до HWM. |
| 2 | 1664 | 0 | 0 | 1541 | Удаляем строки посредством Delete, мы думаем, что наши блоки уже не содержат информации, но это не так. Блоки, как бы по-прежнему заполнены информацией, до уровня HWM и они будут, как бы заполнены, до тех пор, пока объект не будет rebuilt, truncated or shrunk. Поэтому число Consistent gets будет прежним и как видим даже больше, скорее всего из-за чтения из cash. Это влияет на производительность, т.к. full scan table по прежнему будет долгим |
| 3 | 1664 | 1 | 1 | 1541 | При вставке одной строки Insert 1 row, мы видим, что количество использованных блоков увеличилось на один (по сравнению с предыдущим случаем), как, собественно, и количество строк. Но количество consistent gets осталось таким же, т.к. Oracle, как бы, затирает старые данные новыми, поэтому и не видим улучшение performance. |
| 4 | 8 | 0 | 0 | 3 | При использовании операции Truncate, во-первых сбросится HWM до 0 и удалит все индексы данной таблицы. Естественно, это повлияет на performance. По default на пустую таблицу выделяется один сегмент, который содержит 8 блоков, но видно, что ни один из них не задействован, т.е. строк нет. Consistent gets тоже становится в разы меньшим, т.к. фактически нечего читать. |

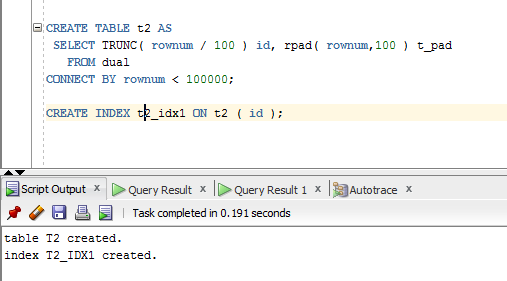
Вывод: если вы хотите удалить строки из таблицы, то лучше использовать truncate, но не забывать, что при этом восстановить предыдущие данные будет крайне тяжело.

# Index Scan types

## Task 2: Index Clustering Factor

### Step 1

Create table t2 as in task 1 steps 1-2



### Step 2

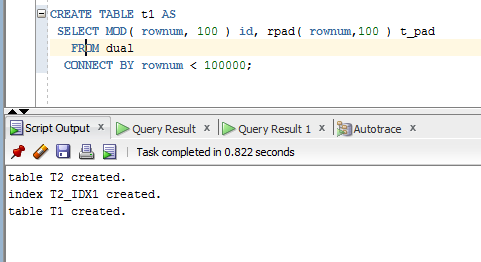
Create table t1 as listed below:

CREATE TABLE t1 AS

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

FROM dual

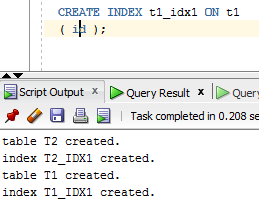
CONNECT BY rownum < 100000;



### Step 3

CREATE INDEX t1\_idx1 ON t1

( id );



### Step 4

Don’t forget to gather statistics 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

View index clustering factor:

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

i.clustering\_factor,

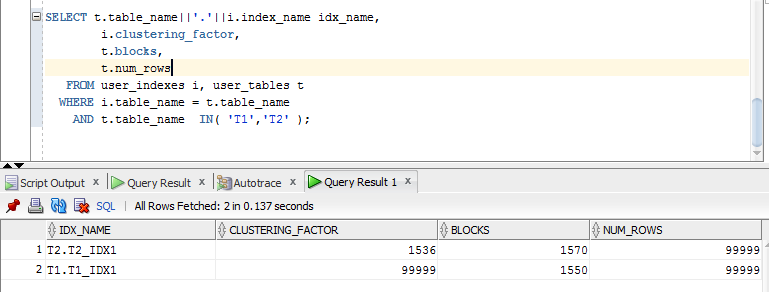
t.blocks,

t.num\_rows

FROM user\_indexes i, user\_tables t

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

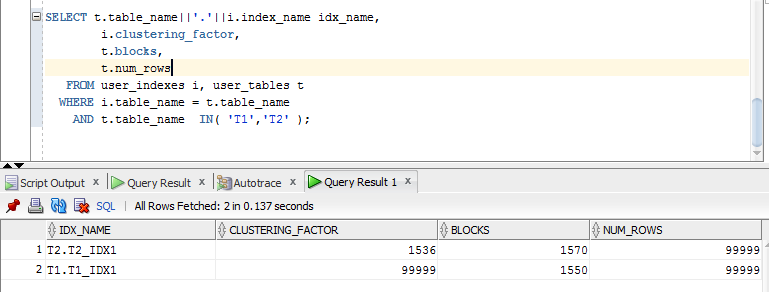
AND t.table\_name IN( 'T1','T2' );



### Task Result

Expecting:

* Screenshot of step 5.



* Description of the index clustering factor.

Clustering factor – фактор кластеризации указывает степень порядоченности строк в таблице на основе значений индекса.

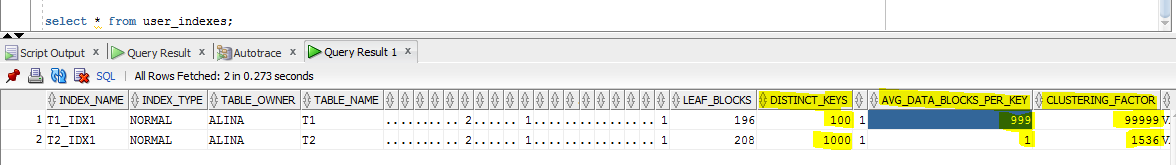
* + Если значение близко к количеству блоков (Blocks), то таблица очень хорошо упорядочена. В данном случае записи индекса в единственном листовом блоке имеют тенденцию указывать на строки в тех же самых блоках данных
  + Если значение близко к количеству строк, то таблица упорядочена весьма случайно. В этом случае маловероятно, что записи из одного листового блока будут указывать на строки в тех же самых блоках данных.

Данный фактор можно также рассматривать как число логических операций ввода-вывода в таблице для чтения всей таблицы через индекс.

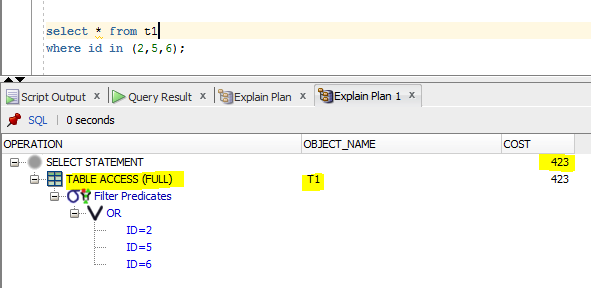
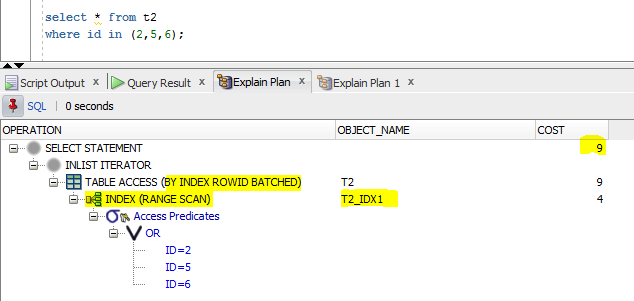
* Explanation: why do we have different factors for t1\_idx1 and t2\_idx.

Если бы мы читали каждую строку в таблице t2 через индекс t2\_idx1 от начала до конца, то было бы совершенно 1536 операций ввода-вывода.

Если бы мы делали тоже самое с таблицей t1, то мы бы выполнили 99999 операций ввода-вывода.

Причина следующая: Oracle производит сканирование данных по структуре индекса, и если следующая строка индекса находится в том же блоке БД, что и предыдущая, то операция ввода-вывода для получения данного блока таблицы не выполняется, т.е. он не делает операцию ввода-вывода из буферного кэша. Но, если же данной ситуации не наблюдается, то Oracle понадобится выполнять операцию ввода-вывода в буферном кэше, чтобы извлячь следующий блок для обработки. 

* Which Index has better performance while executing **select** clause filtered by **IN ( list of values ).**



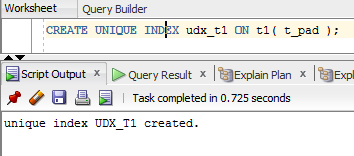
Данный пример показывает, что оптимизатор выбрал план доступа через индекс для таблицы t2 и план доступа с полным сканированием для таблицы t1.

Как можно видеть, сравнивая cost, performance лучше у таблицы t2. Из этого следуает вывод, что на использование индекса все-таки влияет физическая организация данных.

## Task 3: Index Unique Scan

### Step 1

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



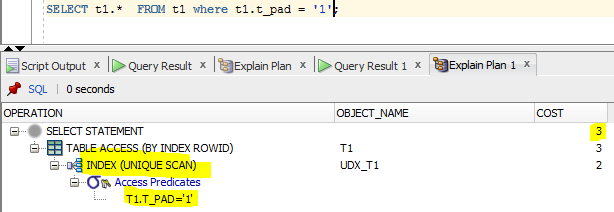
### Step 2

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

### Task Result

Expecting:

* Screenshot of step 2.
* Process description: How does oracle read the block on step#2?



При создании индекса, мы указали свойтсво уникальности. Уникальный индекс обязательно соответсвует одному значения, т.е. когда Oracle находит строгое равенство (в данном случае строка сравнивается со строкой) он дальше не продолжает поиск.

## Task 4: Index Range Scan

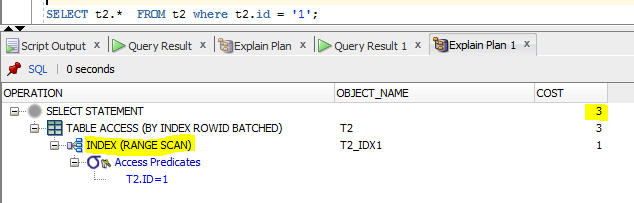
### Step 1

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

### Task Result

Expecting:

* Screenshot of the step 1.
* Description of process: How does oracle read the block on step#1?



При слздании данного индекса мы не указывали тип уникальности, поэтому оптимизатор будет использовать сканирование диапазона по индексу (index range scan). В нашем случае диапапзон представлен одним значением. Т.к. данный индекс не уникальный, то Oracle сохраняет идентификатор строки (rowid) . Данные будут отсортированы по значениям ключа индекса, а затем по возрастанию идентификаторов строк. Данный тип поиск начинается с листьев, где начинается значение листка равно началу диапазона, а заканчивается там, где крайнее значение диапазона равно значению сканированного листка.

## Task 5: Index Skip Scan

### Step 1

CREATE TABLE employees AS

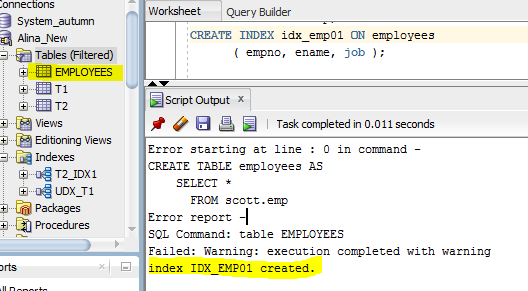
SELECT \*

FROM scott.emp;

### Step 2:

CREATE INDEX idx\_emp01 ON employees

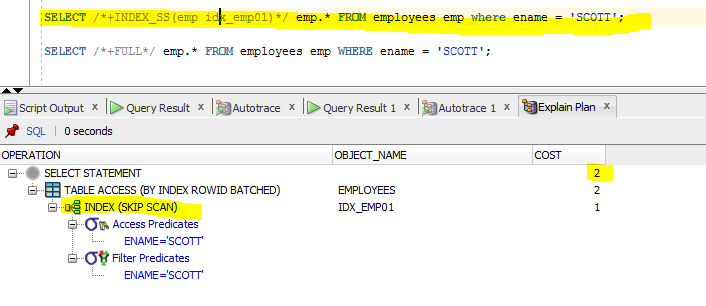
( empno, ename, job );



### Step 3

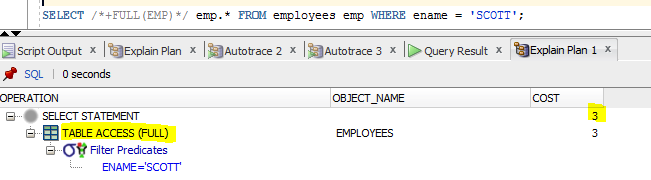
Collect trace and statistic for the queries below:

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



|  |  |
| --- | --- |
| buffer is not pinned count | 2 |
| bytes received via SQL\*Net from client | 563 |
| bytes sent via SQL\*Net to client | 47958 |
| calls to get snapshot scn: kcmgss | 5 |
| calls to kcmgcs | 2 |
| consistent gets | 2 |
| consistent gets from cache | 2 |
| consistent gets pin | 2 |
| consistent gets pin (fastpath) | 2 |
| CPU used by this session | 3 |
| CPU used when call started | 3 |
| DB time | 3 |
| enqueue releases | 1 |
| enqueue requests | 1 |
| execute count | 2 |
| logical read bytes from cache | 16384 |
| no work - consistent read gets | 2 |
| non-idle wait count | 25 |
| opened cursors cumulative | 2 |
| parse count (hard) | 1 |
| parse count (total) | 2 |
| recursive calls | 1 |
| Requests to/from client | 25 |
| session cursor cache hits | 1 |
| session logical reads | 2 |
| sorts (memory) | 2 |
| sorts (rows) | 2356 |
| SQL\*Net roundtrips to/from client | 25 |
| table fetch by rowid | 1 |
| user calls | 27 |
| workarea executions - optimal | 5 |
| workarea memory allocated | 32 |

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



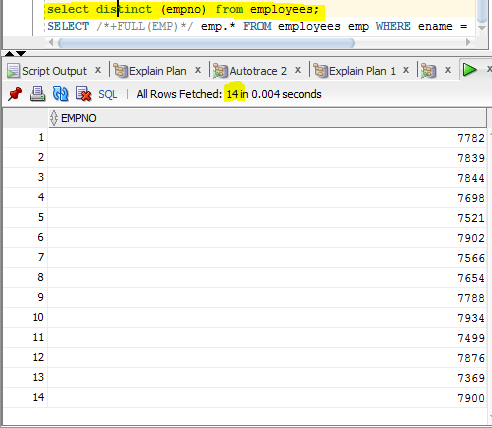
|  |  |
| --- | --- |
| bytes received via SQL\*Net from client | 549 |
| bytes sent via SQL\*Net to client | 47960 |
| calls to get snapshot scn: kcmgss | 3 |
| calls to kcmgcs | 4 |
| consistent gets | 3 |
| consistent gets from cache | 3 |
| consistent gets pin | 3 |
| consistent gets pin (fastpath) | 3 |
| CPU used by this session | 3 |
| CPU used when call started | 3 |
| DB time | 4 |
| enqueue releases | 1 |
| enqueue requests | 1 |
| execute count | 2 |
| logical read bytes from cache | 24576 |
| no work - consistent read gets | 1 |
| non-idle wait count | 25 |
| opened cursors cumulative | 2 |
| parse count (hard) | 1 |
| parse count (total) | 2 |
| recursive calls | 1 |
| Requests to/from client | 25 |
| session cursor cache hits | 1 |
| session logical reads | 3 |
| session pga memory | -131072 |
| sorts (memory) | 2 |
| sorts (rows) | 2356 |
| SQL\*Net roundtrips to/from client | 25 |
| table scan blocks gotten | 1 |
| table scan disk non-IMC rows gotten | 14 |
| table scan rows gotten | 14 |
| table scans (short tables) | 1 |
| user calls | 27 |
| workarea executions - optimal | 5 |
| workarea memory allocated | 32 |

### Task Result

Expecting:

* Step 3 screens.
* Process description: How does oracle uses the index created on the step#2?;

Index skip scan хорошо работает тогда и только тогда, когда головная часть индекса (первая колонка) имеет немного отличающиеся друг от друга значений и их немного. В подтверждение этому к нашему случаю представлена следующая картинка:



Это означает, что Oracle бегло просматривает индекс в поисках места, где empno меняет значение, и читает вниз по дереву с этого места в поисках выполнения условия ename=’SCOTT’. Нашел выполнение условия, и обращается к таблице по значению rowid, тем самым доставая необходимые данные для представления.

* Tasks’ results and your analysis of these results in the description column.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 8 | 1 | 1 | 2 | INDEX (Skip scan) |
| 2 | 8 | 1 | 1 | 3 | FULL SCAN |