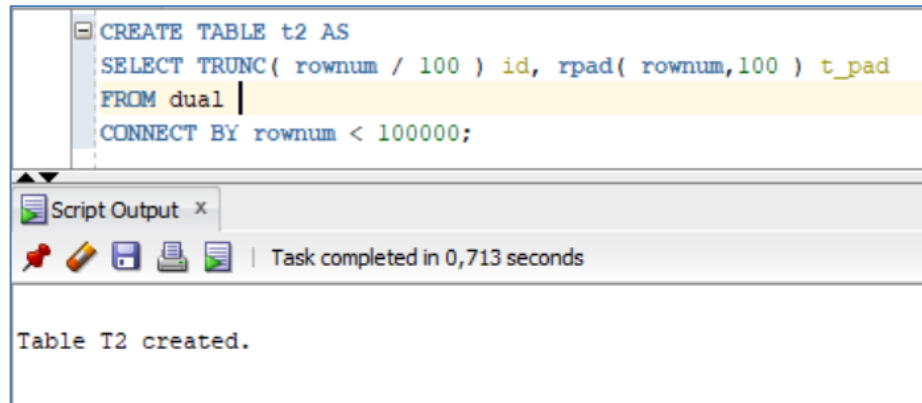


Lab Report #4

Sadovskaya Veronika

Task 1 – Full Scans and the High-water Mark and Block reading

Step 1:



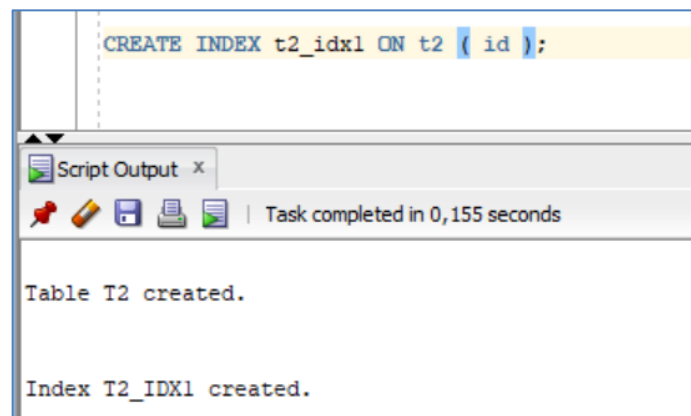
The screenshot shows the SQL Developer interface. The top pane contains the following SQL script:

```
CREATE TABLE t2 AS  
SELECT TRUNC( rownum / 100 ) id, rpad( rownum,100 ) t_pad  
FROM dual  
CONNECT BY rownum < 100000;
```

The bottom pane, titled "Script Output", shows the message "Table T2 created." and "Task completed in 0,713 seconds".

Figure 1.1

Step 2:



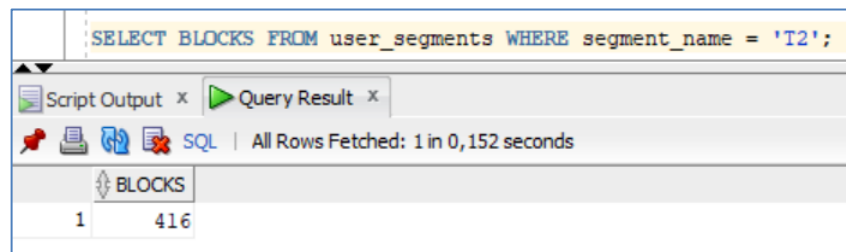
The screenshot shows the SQL Developer interface. The top pane contains the following SQL script:

```
CREATE INDEX t2_idx1 ON t2 ( id );
```

The bottom pane, titled "Script Output", shows the messages "Table T2 created." and "Index T2_IDX1 created." and "Task completed in 0,155 seconds".

Figure 1.2

Step 3:



The screenshot shows the SQL Developer interface. The top pane contains the following SQL query:

```
SELECT BLOCKS FROM user_segments WHERE segment_name = 'T2';
```

The bottom pane, titled "Query Result", shows the results of the query. The results are displayed in a table with two columns: "BLOCKS".

BLOCKS
416

The status bar indicates "All Rows Fetched: 1 in 0,152 seconds".

Figure 1.3 – Block count

SELECT COUNT(DISTINCT (dbms_rowid.rowid_block_number(rowid))) block_ct FROM t2 ;	
Script Output x Query Result x	
All Rows Fetched: 1 in 0,296 seconds	
BLOCK_CT	
1	376

Figure 1.4 – Used block count

SET AUTOTRACE TRACEONLY; SELECT COUNT(*) FROM t2 ;	
Script Output x Query Result x	
Task completed in 0,511seconds	
Statistics	

1	DB time
32	Requests to/from client
33	SQL*Net roundtrips to/from client
2	buffer is not pinned count
509	bytes received via SQL*Net from client
60522	bytes sent via SQL*Net to client
2	calls to get snapshot scn: kcmgss
6	calls to kcmgcs
380	consistent gets
380	consistent gets from cache
380	consistent gets pin
380	consistent gets pin (fastpath)
2	execute count
12451840	logical read bytes from cache
376	no work - consistent read gets
33	non-idle wait count
2	opened cursors cumulative
2	opened cursors current

Figure 1.5 – Explain plan

Step 4:

DELETE FROM t2;	
Script Output x Query Result x	
Task completed in	
99 999 rows deleted.	

Figure 1.6 – Delete all rows from table

Step 5:

SELECT BLOCKS FROM user_segments WHERE segment_name = 'T2';	
Script Output x Query Result x Query Result 1 x	
All Rows Fetched: 1 in 0,009 seconds	
BLOCKS	
1	416

Figure 1.7 – Block count

<pre>SELECT COUNT(DISTINCT (dbmsz_rowid.rowid_block_number(rowid))) block_ct FROM t2 ;</pre>	
Script Output x Query Result x Query Result 1 x	
SQL All Rows Fetched: 1 in 0,005 seconds	
BLOCK_CT	
1	0

Figure 1.8 – Used block count

<pre>SELECT COUNT(DISTINCT (dbmsz_rowid.rowid_block_number(rowid))) block_ct FROM t2 ; SET AUTOTRACE TRACEONLY; SELECT COUNT(*) FROM t2 ; SET AUTOTRACE OFF;</pre>	
Script Output x	
Task completed in 0,421 seconds	
Statistics	
<pre>----- 1 CPU used by this session 1 CPU used when call started 6 Requests to/from client 380 consistent gets 380 consistent gets from cache 380 consistent gets pin 380 consistent gets pin (fastpath) 6 non-idle wait count 2 opened cursors cumulative 1 opened cursors current 380 session logical reads 8 user calls</pre>	

Figure 1.9 – Explain plan

Step 6:

<pre>INSERT INTO t2 (ID, T_PAD) VALUES (1, '1'); COMMIT;</pre>	
Script Output x	
Task completed in 0,029 seconds	
1 row inserted.	
Commit complete.	

Figure 1.10 –Insert 1 row in table

Step 7:

<pre>SELECT BLOCKS FROM user_segments WHERE segment_name = 'T2';</pre>	
Script Output x Query Result x	
SQL All Rows Fetched: 1 in 0,005 seconds	
BLOCKS	
1	416

Figure 1.11 –Block count

SELECT COUNT(DISTINCT (dbms_rowid.rowid_block_number(rowid))) block_ct FROM t2 ;	
Script Output x	Query Result x
All Rows Fetched: 1 in 0,008 seconds	
BLOCK_CT	
1	1

Figure 1.12 – Used block count

SET AUTOTRACE TRACEONLY;	
SELECT COUNT(*) FROM t2 ;	
Script Output x	Query Result x
Task completed in 0,403 seconds	
Statistics	

1	CPU used by this session
1	CPU used when call started
2	DB time
6	Requests to/from client
380	consistent gets
380	consistent gets from cache
380	consistent gets pin
380	consistent gets pin (fastpath)
6	non-idle wait count
2	opened cursors cumulative
1	opened cursors current
1	process last non-idle time
380	session logical reads
8	user calls

Figure 1.13 – Explain plan

Step 8:

TRUNCATE TABLE t2;	
Script Output x	
Task completed in 0,048 seconds	
Table T2 truncated.	

Figure 1.14 – Truncate table

Step 9:

SELECT BLOCKS FROM user_segments WHERE segment_name = 'T2';	
Script Output x	Query Result x
All Rows Fetched: 1 in 0,008 seconds	
BLOCKS	
1	6

Figure 1.15 – Block count

SELECT COUNT(DISTINCT (dbms_rowid.rowid_block_number(rowid))) block_ct FROM t2 ;	
SET AUTOTRACE TRACEONLY;	
Script Output x	Query Result x
All Rows Fetched: 1 in 0,011 seconds	
BLOCK_CT	
1	0

Figure 1.16 – Used block count

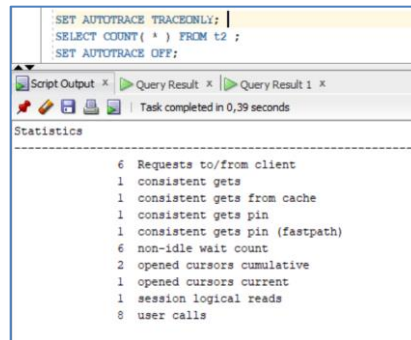


Figure 1.17 – Explain plan

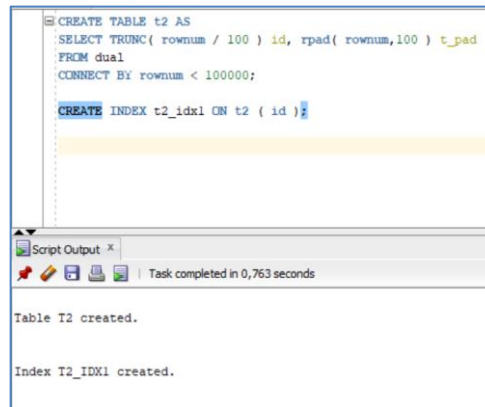
№	Count of Blocks	Count of Used Blocks	Count of Rows	Consistent gets	Description
1-3	416	376	99999	380	The field “consistent gets” specifies the number of requests to the block. At this stage, the table was filled in and HWM was installed (the maximum number of rows is observed).
4-5	416	0	0	380	'Consistent gets' has not changed, since the HWM remains the same(when performing the 'delete' operation, the object's statistics and allocated space are saved).
6-7	416	1	1	380	'Consistent gets' has not changed, because the HWM remains the same(when performing the 'insert' operation (row_count = 1), the maximum number of elements has not changed(max_row_count = 9999)).
8-9	6	0	0	1	'Consistent gets' has changed, since the HWM has also changed(when performing the TRUNCATE operation, all table data is freed, so TRUNCATE deletes all statistics, allocated space, and resets the HWM).

Данный метод доступа, как следует из названия, подразумевает перебор всех строк таблицы с исключением тех, которые не удовлетворяют предикату where (если таковой есть). Применяется он либо в случае, когда

условия предиката отсутствуют в индексе, либо когда индекса нет в принципе.

Task 2 – Index Clustering factor parameter

Step 1: Create table T2 and index on T2



```
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 );
```

Script Output x

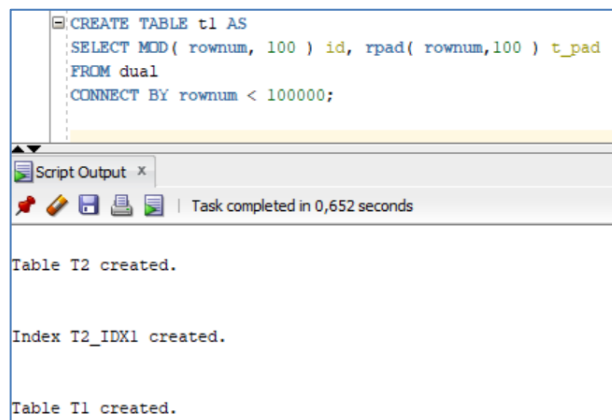
Task completed in 0,763 seconds

Table T2 created.

Index T2_IDX1 created.

Figure 2.1

Step 2: Create table T1



```
CREATE TABLE t1 AS
SELECT MOD( rownum, 100 ) id, rpad( rownum,100 ) t_pad
FROM dual
CONNECT BY rownum < 100000;
```

Script Output x

Task completed in 0,652 seconds

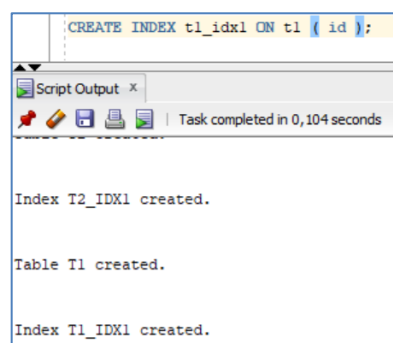
Table T2 created.

Index T2_IDX1 created.

Table T1 created.

Figure 2.2

Step 3: Create index on T1



```
CREATE INDEX t1_idx1 ON t1 ( id );
```

Script Output x

Task completed in 0,104 seconds

Index T2_IDX1 created.

Table T1 created.

Index T1_IDX1 created.

Figure 2.3

Step 4: Calculate statistics for T1 and T2

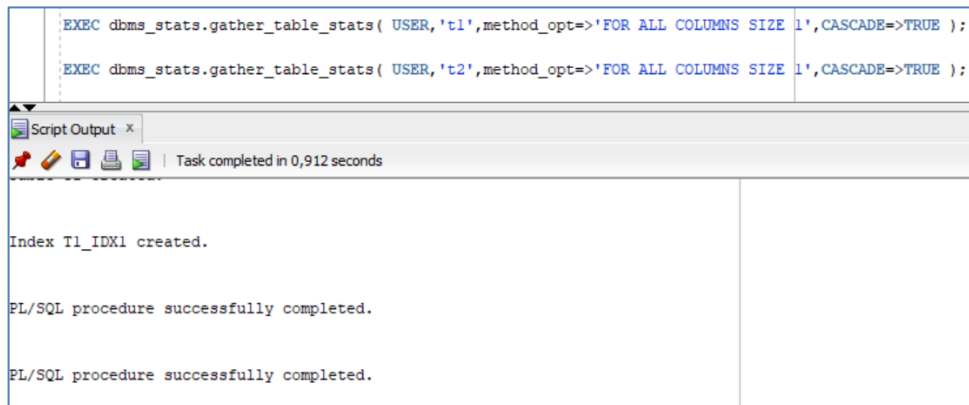


Figure 2.4

Step 5: Select clustering Factor

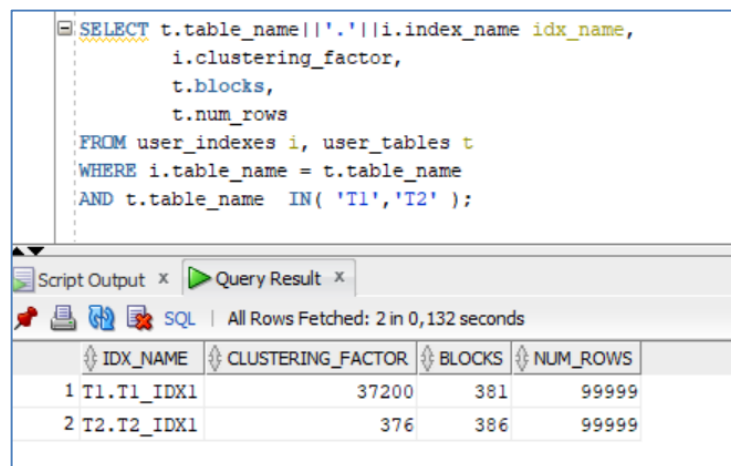


Figure 2.5

The Oracle Database Reference guide describes the purpose of the CLUSTERING _ FACTOR column as follows. Specifies the order of rows in the table based on index values.

- If the value is close to the number of blocks, the table is very well ordered. In this case, index entries in a single leaf block tend to point to rows in the same data blocks.

- If the value is close to the number of rows, the table is ordered very randomly. In this case, it is unlikely that index entries from the same leaf block will point to rows in the same data blocks. The clustering factor can also be considered as a number representing the number of logical I / o operations in a table that must be performed to read the entire table through the index.

In other words, CLUSTERING _ FACTOR is a sign of how the table is ordered in relation to the index itself.

As you can see from the following image, table T2 is well ordered, and table T1 is slightly worse (CLUSTERING _ FACTOR_T1 > CLUSTERING _ FACTOR_T2).

We can conclude that Bitmap Index has best selective performance in execution Select clause filtered by IN (list of values). Range Scan Unique will also give good results (if the indexes are unique).

Данный метод доступа просматривает все листовые блоки индекса для поиска соответствий условиям предиката. Для того чтобы Oracle мог применить этот метод доступа, хотя бы одно из полей ключа должно иметь ограничение NOT NULL, т.к. только в этом случае соответствующая строка таблицы попадет в индекс. Этот метод обычно быстрее чем TABLE FULL SCAN.

Task 3 – Index Unique Scan

Step 1:

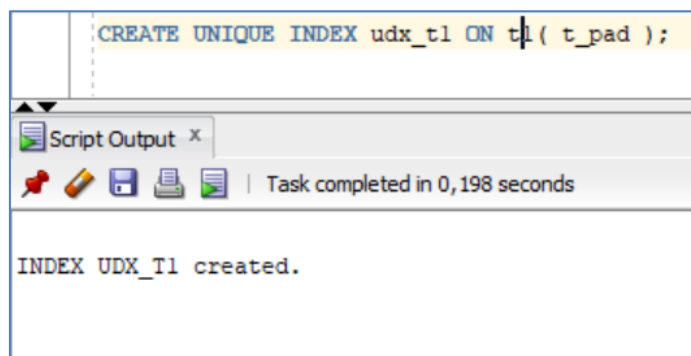


Figure 3.1

Step 2:

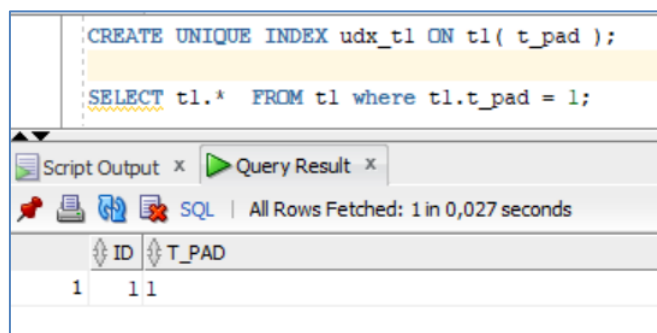


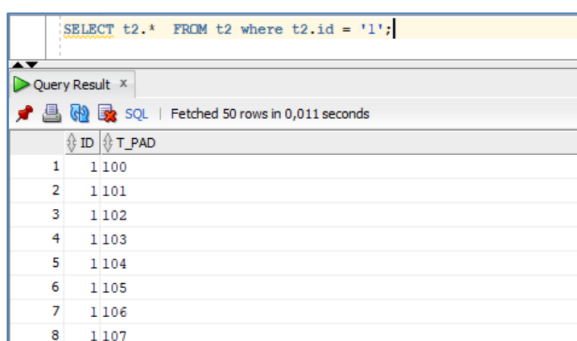
Figure 3.2

The index structure goes from the root to leaf block to a single entry, gets a rowid, which is used to access the table data block containing one row. The TABLE ACCESS BY INDEX ROWID step in the plan specifies access to a table data block.

Уникальное сканирование индекса выбирается, когда предикат содержит условие, использующее столбец, определенный с помощью UNIQUE или PRIMARY KEY индексов. Эти типы индексов гарантируют, что будет возвращено ноль или одна строка для указанного значения. В этом случае структура индекса будет проходить от корневого блока к конечному блоку, получит идентификатор строки и использовать его для доступа к блоку данных таблицы, содержащему одну строку. Шаг TABLE ACCESS BY INDEX ROWID в плане указывает доступ к блоку данных таблицы. Количество требуемых блоков доступа всегда будет равно высоте индекса плюс один, если строка не связана или не содержит большой объект, который хранится в другом месте.

Task 4 – Index Range Scan

Step 1:



The screenshot shows a SQL query window with the query: `SELECT t2.* FROM t2 where t2.id = '1';`. Below the query, a 'Query Result' window displays the results. It indicates 'Fetched 50 rows in 0,011 seconds'. The result table has two columns: 'ID' and 'T_PAD'. The data rows are as follows:

ID	T_PAD
1	1100
2	1101
3	1102
4	1103
5	1104
6	1105
7	1106
8	1107

Figure 4.1

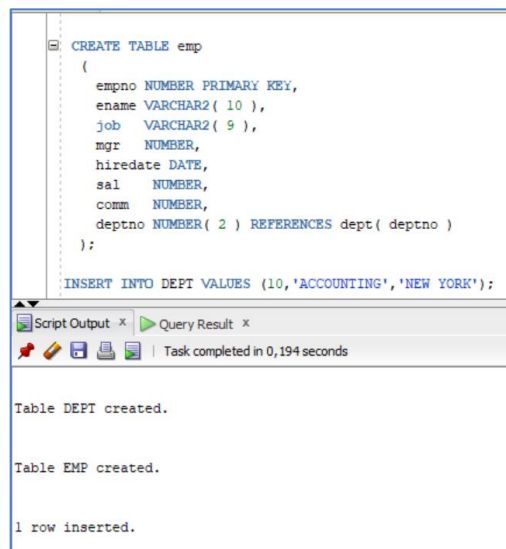
A range scan will traverse the index structure from the root block to the first leaf block containing an entry matching the specified condition. From that starting point, a rowid will be retrieved from the index entry and the table data block will be retrieved (TABLE ACCESS BY INDEX ROWID). After the first row is retrieved, the index leaf block will be accessed again and the next entry will be read to retrieve the next rowid.

Данное сканирование выбирается, когда предикат содержит условие, возвращающее диапазон данных. Индекс может быть уникальным или неуникальным, так как это условие, которое определяет, будет ли несколько строк возвращено или нет. Указанные условия могут использовать такие операторы, как <, >, LIKE, BETWEEN и даже =. Для выбора диапазона сканирования диапазон должен быть достаточно избирательным. Чем больше

диапазоне, тем более вероятно, что вместо этого будет выбрана операция полного сканирования.

Task 5 – Index Skip Scan

Step 1:



```
CREATE TABLE emp
(
  empno NUMBER PRIMARY KEY,
  ename VARCHAR2( 10 ),
  job VARCHAR2( 9 ),
  mgr NUMBER,
  hiredate DATE,
  sal NUMBER,
  comm NUMBER,
  deptno NUMBER( 2 ) REFERENCES dept( deptno )
);

INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW YORK');
```

Script Output x Query Result x

Task completed in 0,194 seconds

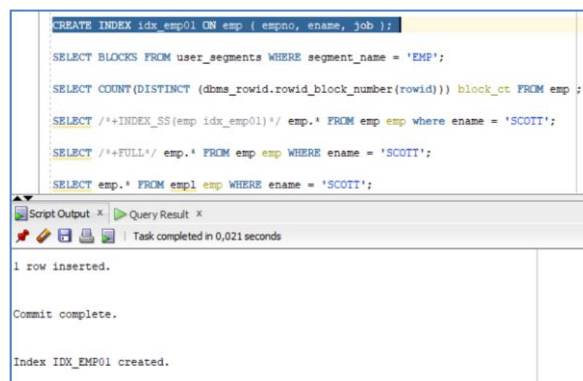
Table DEPT created.

Table EMP created.

1 row inserted.

Figure 5.1 – Create table emp

Step 2:



```
CREATE INDEX idx_emp01 ON emp ( empno, ename, job );

SELECT BLOCKS FROM user_segments WHERE segment_name = 'EMP';

SELECT COUNT(DISTINCT (dbms_rowid.rowid_block_number(rowid))) block_ct FROM emp ;

SELECT /*+INDEX_SS(emp idx_emp01)*/ emp.* FROM emp emp where ename = 'SCOTT';

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

SELECT emp.* FROM emp emp WHERE ename = 'SCOTT';
```

Script Output x Query Result x

Task completed in 0,021 seconds

1 row inserted.

Commit complete.

Index IDX_EMP01 created.

Figure 5.2 – Create index

Step 3:

<pre>SELECT /*+INDEX_SS(emp idx_emp01)*/ emp.* FROM employees emp where ename = 'SCOTT';</pre> <pre>SELECT /*+FULL*/ emp.* FROM employees emp WHERE ename = 'SCOTT';</pre>	
Script Output x Query Result x Query Result 1 x Query Result 2 x Query Result 3 x	
SQL All Rows Fetched: 1 in 0,008 seconds	
EMPNO	ENAME
1	7844 SCOTT
JOB	SALESMAN
MGR	7698 08.09.81
SAL	1500
COMM	0
DEPTNO	30

Figure 5.3 – Select with index skip scan

<pre>SELECT /*+INDEX_SS(emp idx_emp01)*/ emp.* FROM employees emp where ename = 'SCOTT';</pre> <pre>SELECT /*+FULL*/ emp.* FROM employees emp WHERE ename = 'SCOTT';</pre>	
Script Output x Query Result x Query Result 1 x Query Result 2 x Query Result 3 x	
SQL All Rows Fetched: 1 in 0,004 seconds	
EMPNO	ENAME
1	7844 SCOTT
JOB	SALESMAN
MGR	7698 08.09.81
SAL	1500
COMM	0
DEPTNO	30

Figure 5.4 – Select with full scan

<pre>SELECT emp.* FROM employees emp WHERE ename = 'SCOTT';</pre>	
Script Output x Query Result x Query Result 1 x Query Result 2 x Query	
SQL All Rows Fetched: 1 in 0,009 seconds	
EMPNO	ENAME
1	7844 SCOTT
JOB	SALESMAN
MGR	7698 08.09.81
SAL	1500
COMM	0
DEPTNO	30

Figure 5.5 – Select without hint

<pre>SELECT BLOCKS FROM user_segments WHERE segment_name = 'EMP';</pre>	
Script Output x Query Result x	
SQL All Rows Fetched: 1 in 0,025 seconds	
BLOCKS	
1	6

Figure 5.6 – Count of blocks

<pre>SELECT COUNT(DISTINCT (dbms_rowid.rowid_block_number(rowid))) block_ct FROM emp ;</pre>	
Script Output x Query Result x	
SQL All Rows Fetched: 1 in 0,037 seconds	
BLOCK_CT	
1	1

Figure 5.7 – Count of used blocks

```

SET AUTOTRACE TRACEONLY;
SELECT emp.* FROM emp emp WHERE ename = 'SCOTT';

```

Script Output x Query Result 1 x

Task completed in 1,077 seconds

```

SELECT emp.* FROM emp emp WHERE ename = 'SCOTT'

Plan hash value: 3956160932

-----
| Id | Operation          | Name | E-Rows |
-----
| 0  | SELECT STATEMENT   |      |        |
|* 1  | TABLE ACCESS FULL| EMP  |    1   |
-----
PLAN_TABLE_OUTPUT

-----
Predicate Information (identified by operation id):
-----

1 - filter("ENAME"='SCOTT')

```

Note

Figure 5.8 – Trace and statistic of explain plan for select without hint

```

SET AUTOTRACE TRACEONLY;
SELECT emp.* FROM emp emp WHERE ename = 'SCOTT';

```

Script Output x Query Result 1 x

Task completed in 1,077 seconds

Statistics

```

-----
1  DB time
33  Requests to/from client
33  SQL*Net roundtrips to/from client
2  buffer is not pinned count
543 bytes received via SQL*Net from client
60946 bytes sent via SQL*Net to client
2  calls to get snapshot scn: kcmgcs
4  calls to kcmgcs
5  consistent gets
5  consistent gets from cache
5  consistent gets pin
5  consistent gets pin (fastpath)
2  execute count
163840 logical read bytes from cache
3  no work - consistent read gets
33  non-idle wait count
2  opened cursors cumulative
-1  opened cursors current
2  parse count (total)

```

Figure 5.9 - Trace and statistic of explain plan for select without hint

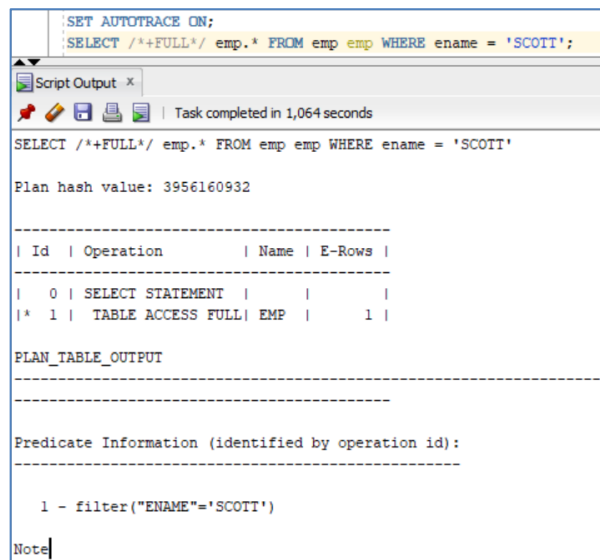


Figure 5.10 - Trace and statistic of explain plan for full scan

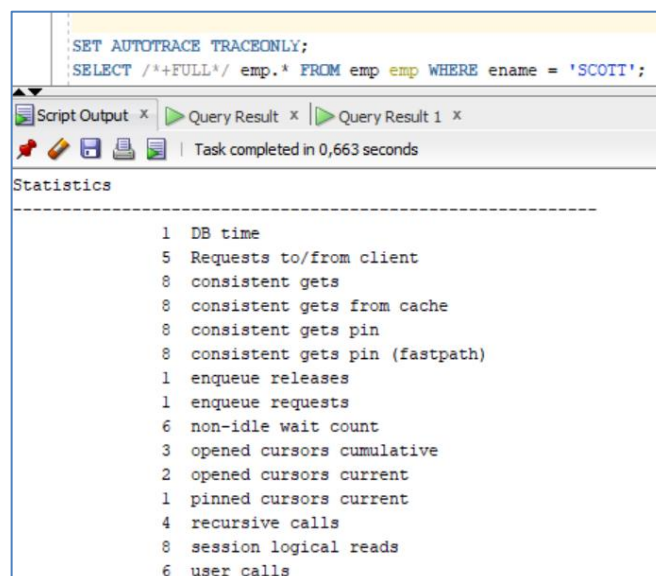


Figure 5.11 - Trace and statistic of explain plan for full scan

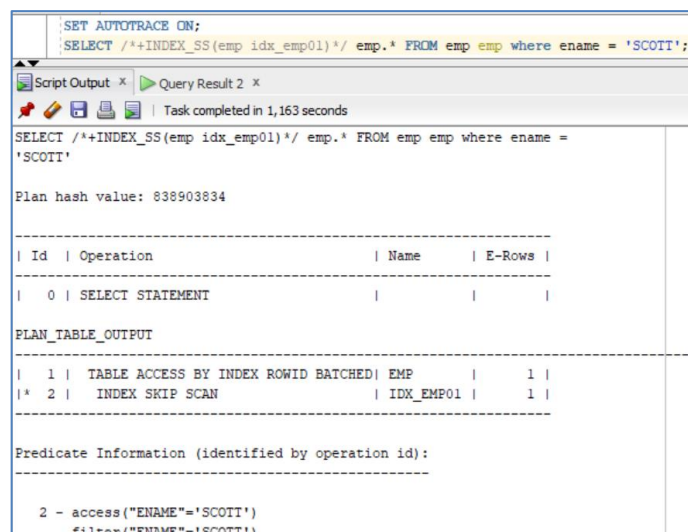


Figure 5.12 - Trace and statistic of explain plan for index skip scan

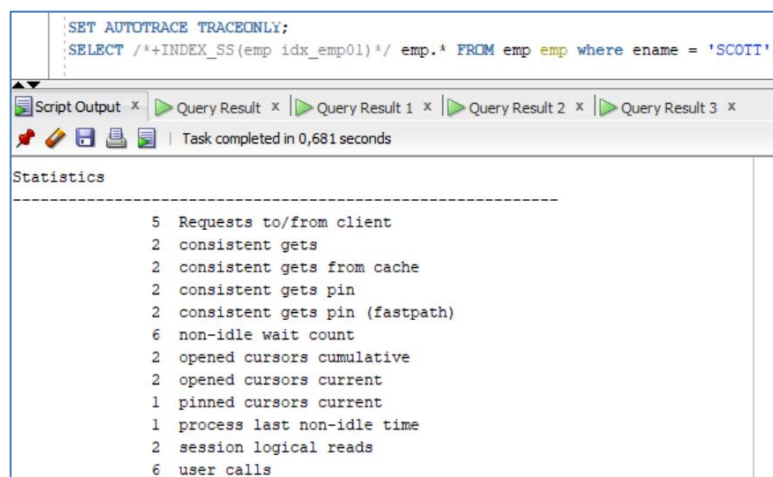


Figure 5.13 - Trace and statistic of explain plan for index skip scan

№	Count of Blocks	Count of Used Blocks	Count of Rows	Consistent gets	Description
Full scan	6	1	12	8	Oracle performed SELECT (using the hint /*+FULL(emp)*/) and used the Full Scan algorithm
Index skip scan	6	1	12	2	Oracle performed SELECT (using the hint /*+INDEX_SS(emp idx_emp01)*/) and used the Index Skip Scan algorithm
Without hint	6	1	12	5	Oracle performed SELECT (without using hints) and used the most appropriate Full Scan

Данное сканирование выбирается, когда предикат содержит условие для не ведущего столбца в индекс и ведущие столбцы довольно различны. Сканирование с пропуском работает логически разбивая многостолбцовые индексы на более мелкие подиндексы. Количество логических подиндексов равно определяемому количеством различных значений в ведущих столбцах индекса. Следовательно, чем больше различных значений содержит ведущий столбец, тем больше логических подиндексов необходимо создать. Если будет создано слишком много подиндексов, то данное сканирование будет не так эффективно, как полное сканирование. Однако в тех случаях, когда количество необходимых подиндексов будет небольшим, данное сканирование может быть более эффективным, чем полное сканирование, поскольку сканирование небольших индексированных блоков может быть более эффективным, чем сканирование больших блоков таблицы.