U1M4.LW.Access and Join Methods

Part 1

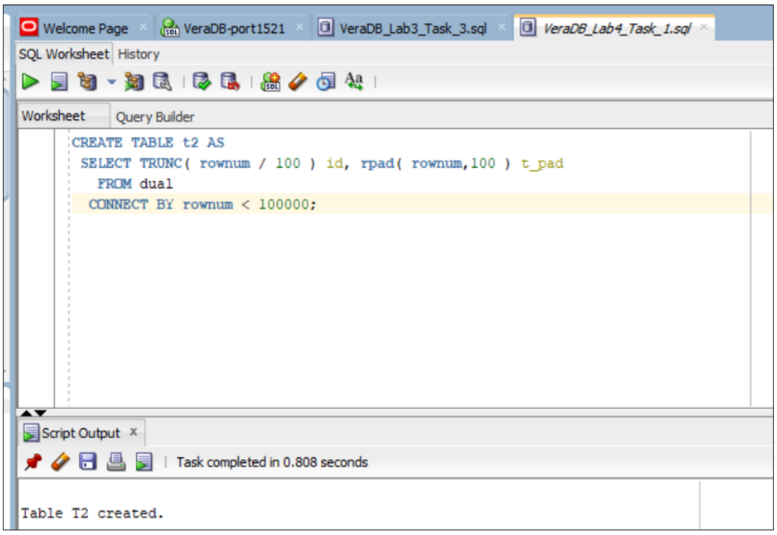
Shkrabatouskaya Vera

https://github.com/VeraShkrabatouskaya/DataMola_Data-Camping-2022

1. Table access full scan

1.1. Task 1: Full Scans and the High-water Mark and Block reading

Step 1: Create table t2

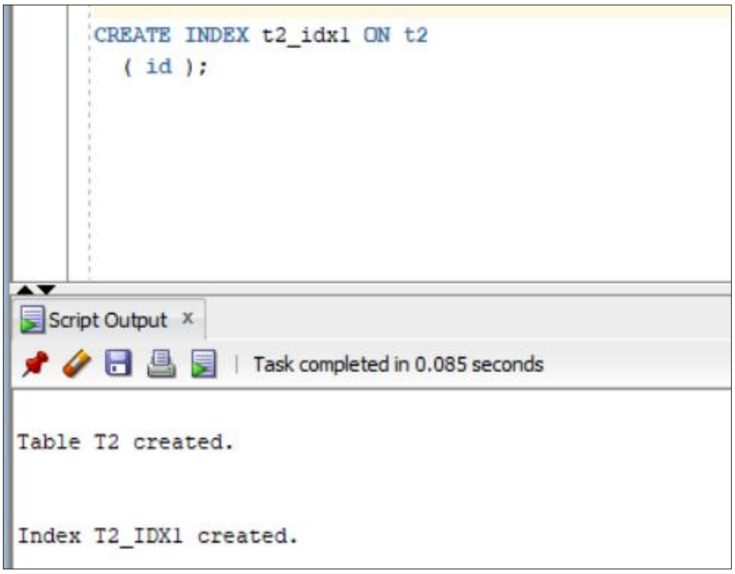


```
CREATE TABLE t2 AS
 SELECT TRUNC( rownum / 100 ) id, rpad( rownum,100 ) t_pad
   FROM dual
  CONNECT BY rownum < 100000;
```

Task completed in 0.808 seconds

```
Table T2 created.
```

Step 2: Create index



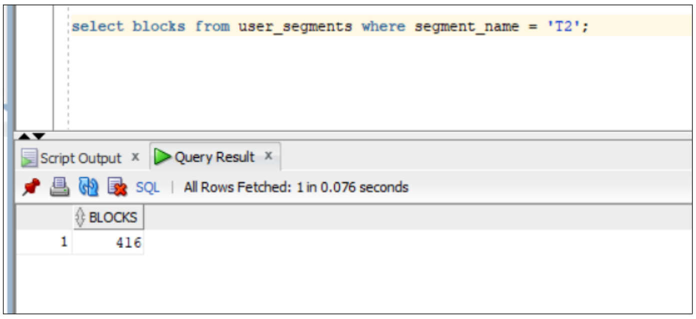
```
CREATE INDEX t2_idx1 ON t2
  ( id );
```

Task completed in 0.085 seconds

```
Table T2 created.


Index T2_IDX1 created.
```

Step 3:

Block count:

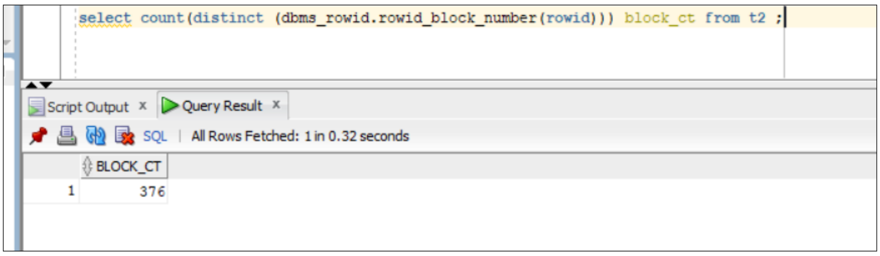


```
select blocks from user_segments where segment_name = 'T2';
```

All Rows Fetched: 1 in 0.076 seconds

	BLOCKS
1	416

Used Block Count:

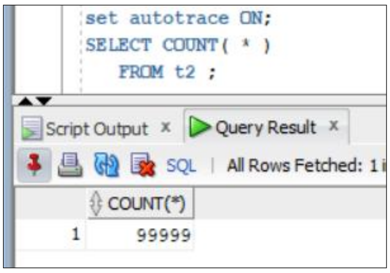


```
select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;
```

All Rows Fetched: 1 in 0.32 seconds

	BLOCK_CT
1	376

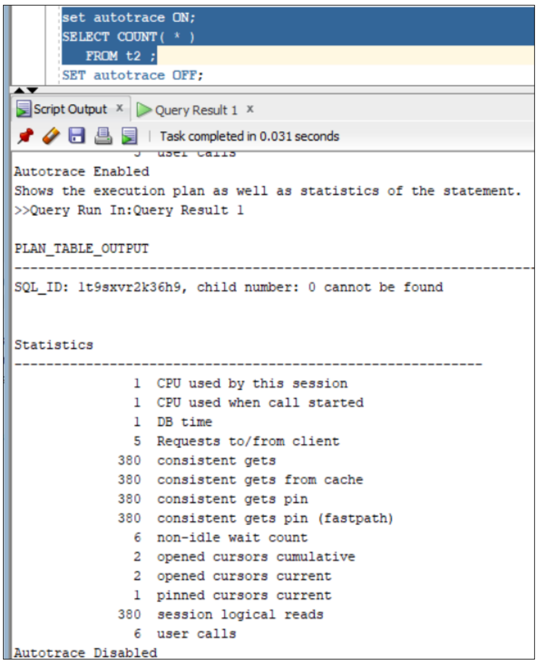
Explain Plan / Count rows:



```
set autotrace ON;
SELECT COUNT( * )
   FROM t2 ;
```

All Rows Fetched: 1

	COUNT(*)
1	99999

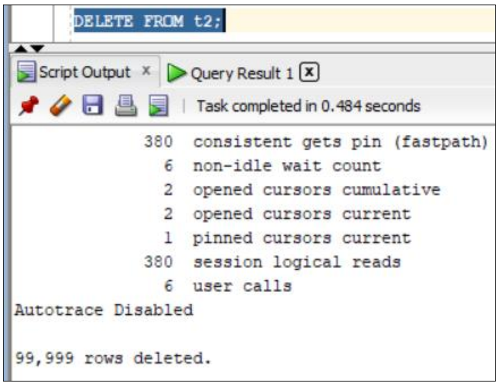


```
set autotrace ON;
SELECT COUNT( * )
   FROM t2 ;
SET autotrace OFF;
```

Task completed in 0.031 seconds

```
Autotrace Enabled
Shows the execution plan as well as statistics of the statement.
>>Query Run In:Query Result 1

PLAN_TABLE_OUTPUT
------------------------------------------------------------------
SQL_ID: 1t9sxvr2k36h9, child number: 0 cannot be found


Statistics
-----------------------------------------------------------
               1  CPU used by this session
               1  CPU used when call started
               1  DB time
               5  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
               2  opened cursors current
               1  pinned cursors current
             380  session logical reads
               6  user calls
Autotrace Disabled
```

Step 4: Delete All Rows from table

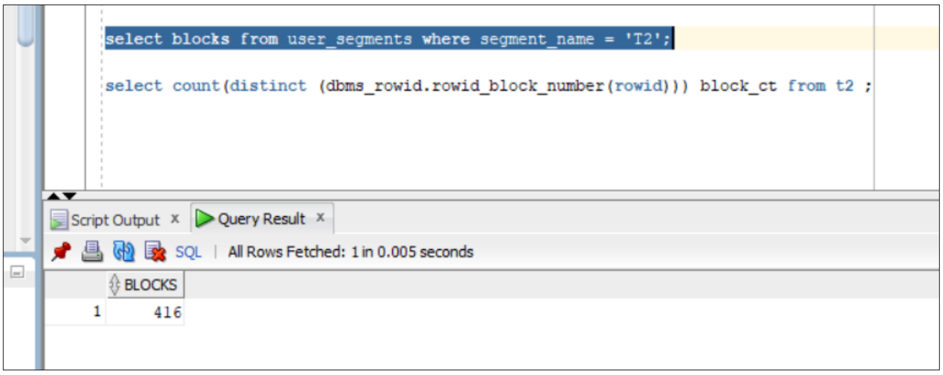


```
DELETE FROM t2;
```

```
             380  consistent gets pin (fastpath)
               6  non-idle wait count
               2  opened cursors cumulative
               2  opened cursors current
               1  pinned cursors current
             380  session logical reads
               6  user calls
Autotrace Disabled

99,999 rows deleted.
```

Step 5: Repeat Step 3 and collect results.

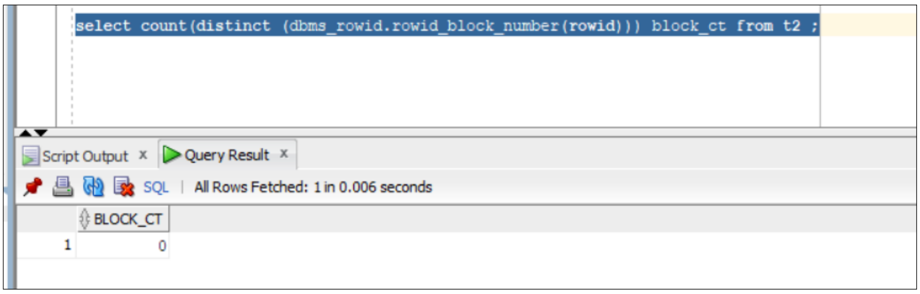
Block count:



```
select blocks from user_segments where segment_name = 'T2';

select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;
```

	BLOCKS
1	416

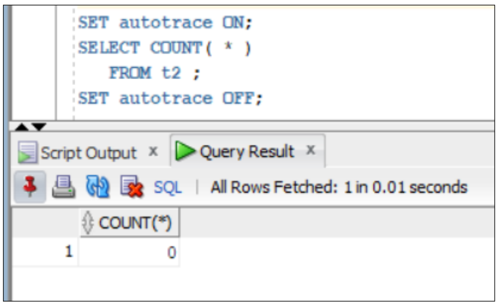
Used Block Count:



```
select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;
```

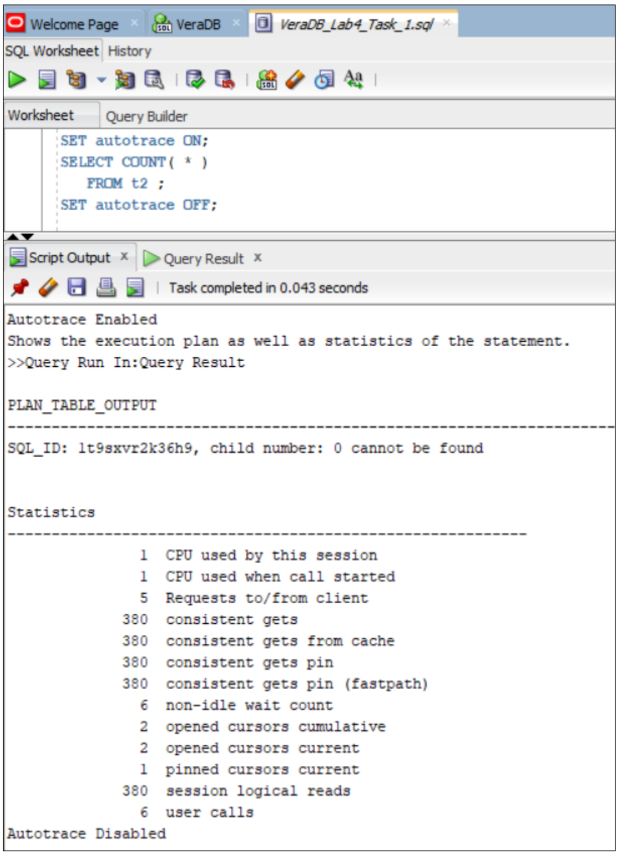
	BLOCK_CT
1	0

Explain Plan / Count rows:



```
SET autotrace ON;
SELECT COUNT( * )
   FROM t2 ;
SET autotrace OFF;
```

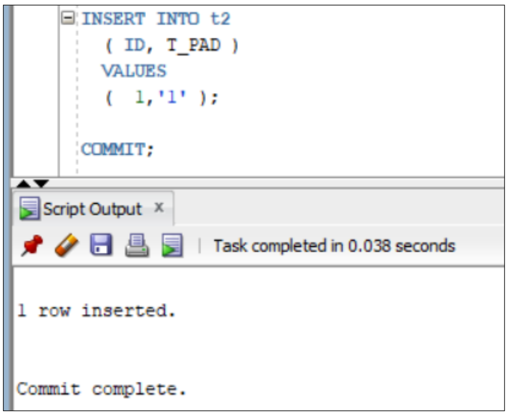
	COUNT(*)
1	0



```
SET autotrace ON;
SELECT COUNT( * )
   FROM t2 ;
SET autotrace OFF;
```

```
Autotrace Enabled
Shows the execution plan as well as statistics of the statement.
>>Query Run In:Query Result

PLAN_TABLE_OUTPUT
---------------------------------------------------------------------
SQL_ID: 1t9sxvr2k36h9, child number: 0 cannot be found


Statistics
-----------------------------------------------------------
               1  CPU used by this session
               1  CPU used when call started
               5  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
               2  opened cursors current
               1  pinned cursors current
             380  session logical reads
               6  user calls
Autotrace Disabled
```

<u>Step 6</u>: Insert 1 row

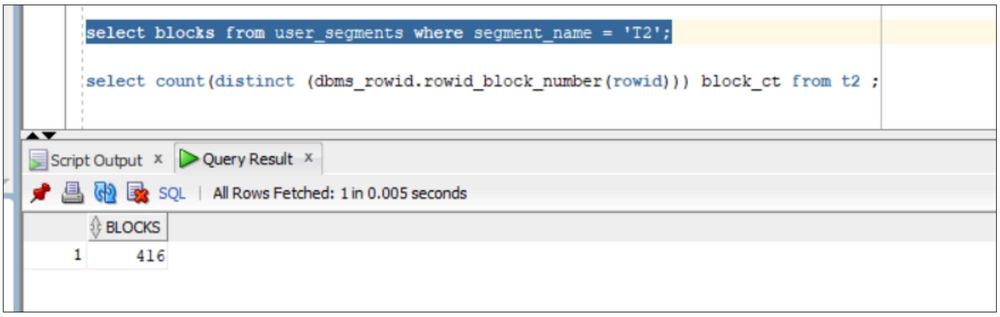


```
INSERT INTO t2
  ( ID, T_PAD )
  VALUES
  (  1,'1' );

COMMIT;
```

```
1 row inserted.


Commit complete.
```

Step 7: Repeat Step 3 and collect results.

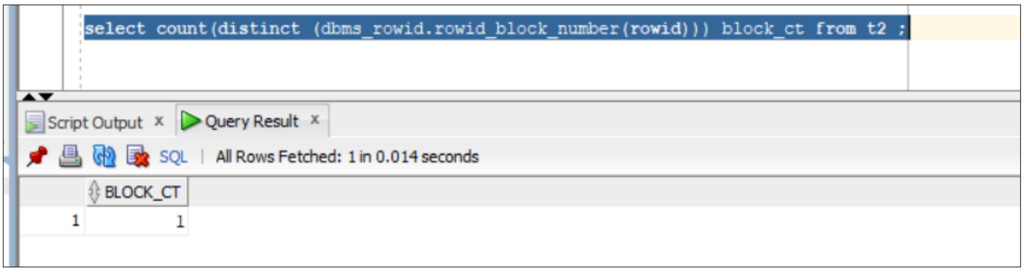
Block count:



```
select blocks from user_segments where segment_name = 'T2';

select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;
```

	BLOCKS
1	416

Used Block Count:

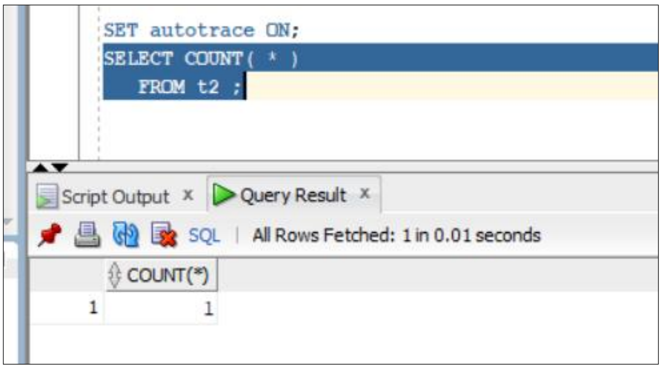


```
select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;
```

All Rows Fetched: 1 in 0.014 seconds

	BLOCK_CT
1	1

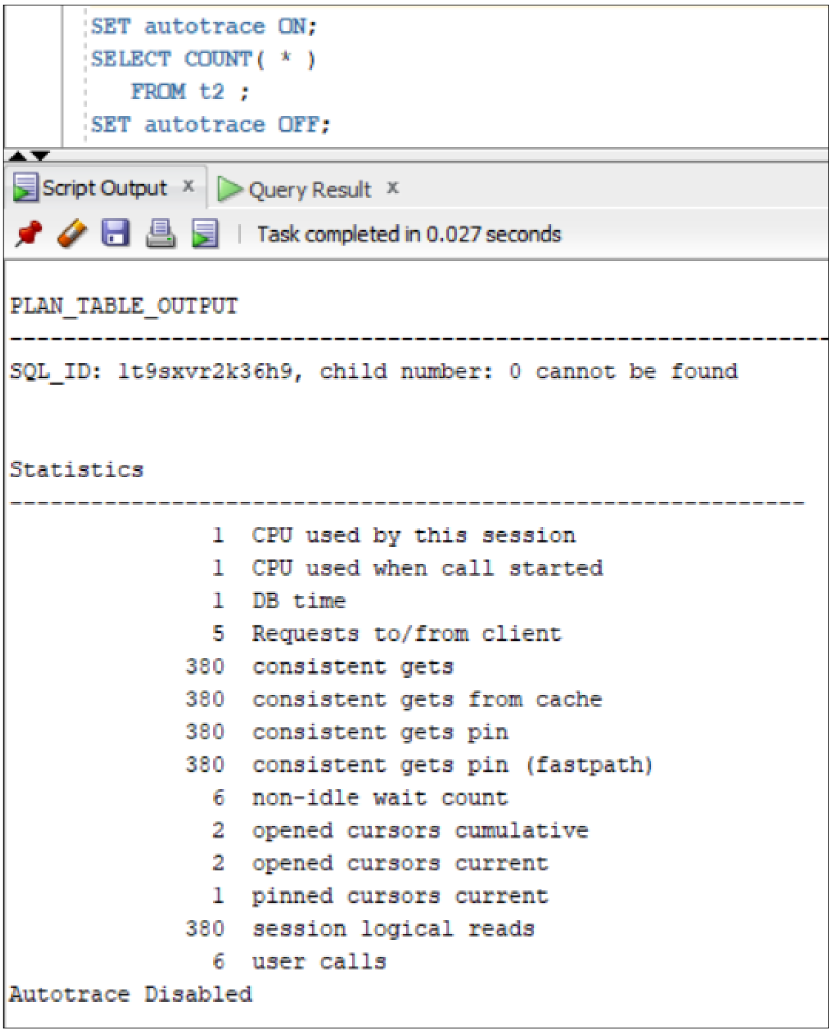
Explain Plan / Count rows:



```
SET autotrace ON;
SELECT COUNT( * )
  FROM t2 ;
```

All Rows Fetched: 1 in 0.01 seconds

	COUNT(*)
1	1

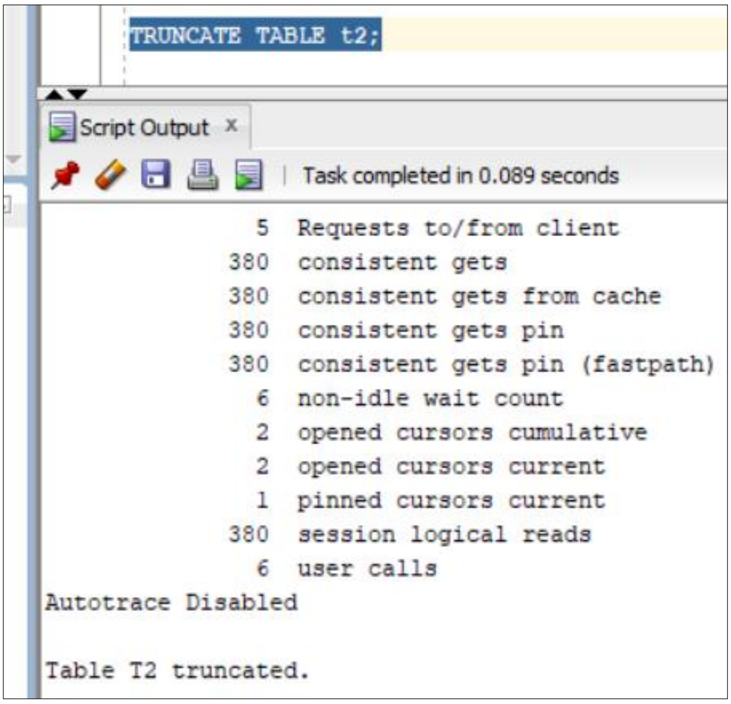


```
SET autotrace ON;
SELECT COUNT( * )
  FROM t2 ;
SET autotrace OFF;
```

Task completed in 0.027 seconds

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------
SQL_ID: 1t9sxvr2k36h9, child number: 0 cannot be found


Statistics
-----------------------------------------------------------
               1  CPU used by this session
               1  CPU used when call started
               1  DB time
               5  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
               2  opened cursors current
               1  pinned cursors current
             380  session logical reads
               6  user calls
Autotrace Disabled
```

Step 8: Truncate Table



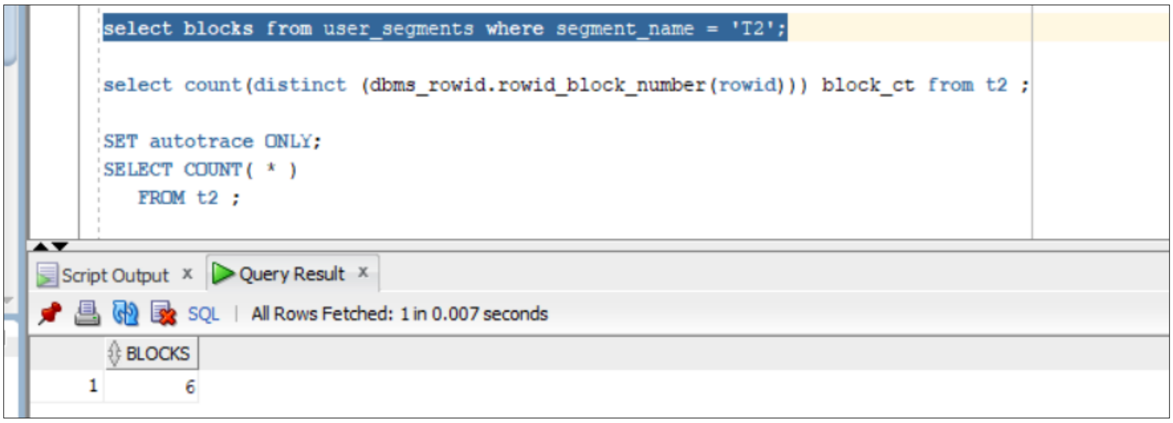
```
TRUNCATE TABLE t2;
```

Task completed in 0.089 seconds

```
                5  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
                2  opened cursors current
                1  pinned cursors current
              380  session logical reads
                6  user calls
Autotrace Disabled

Table T2 truncated.
```

Step 9: Repeat Step 3 and collect results.

Block count:

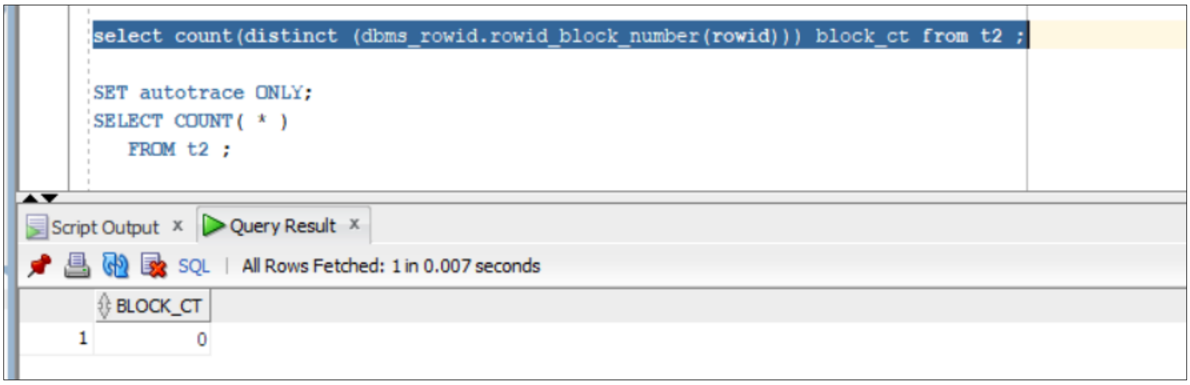


```
select blocks from user_segments where segment_name = 'T2';

select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;

SET autotrace ONLY;
SELECT COUNT( * )
   FROM t2 ;
```

All Rows Fetched: 1 in 0.007 seconds

	BLOCKS
1	6

Used Block Count:

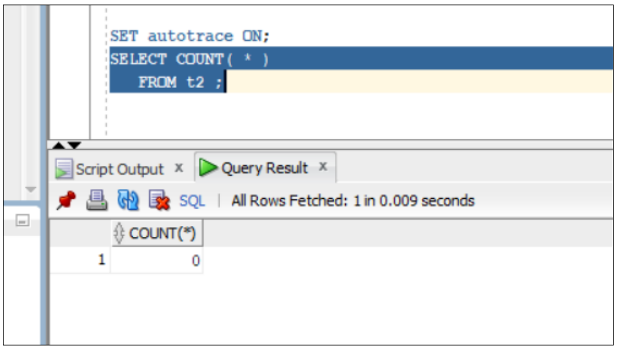


```
select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;

SET autotrace ONLY;
SELECT COUNT( * )
   FROM t2 ;
```

All Rows Fetched: 1 in 0.007 seconds

	BLOCK_CT
1	0

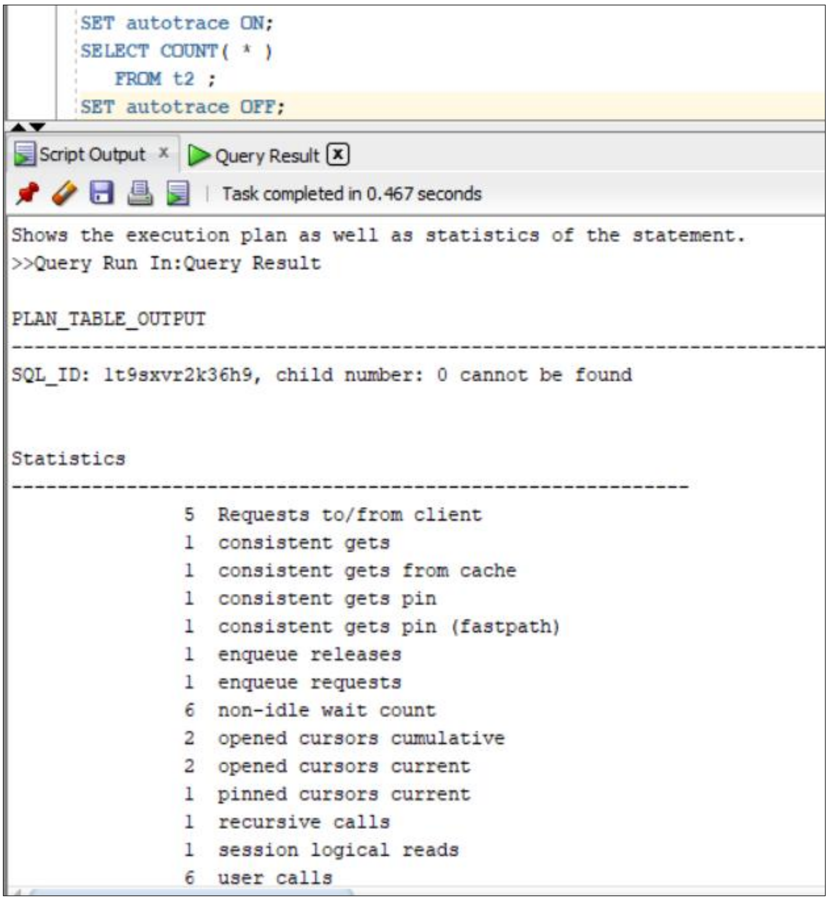
Explain Plan / Count rows:



```
SET autotrace ON;
SELECT COUNT( * )
   FROM t2 ;
```

All Rows Fetched: 1 in 0.009 seconds

	COUNT(*)
1	0

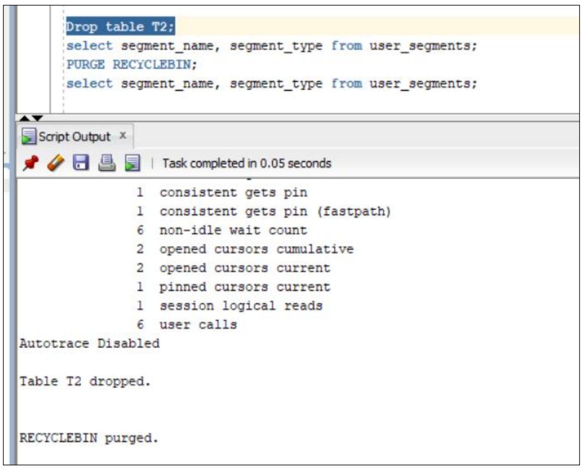


```
SET autotrace ON;
SELECT COUNT( * )
   FROM t2 ;
SET autotrace OFF;
```

Task completed in 0.467 seconds

```
Shows the execution plan as well as statistics of the statement.
>>Query Run In:Query Result

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------
SQL_ID: 1t9sxvr2k36h9, child number: 0 cannot be found


Statistics
-----------------------------------------------------------
               5  Requests to/from client
               1  consistent gets
               1  consistent gets from cache
               1  consistent gets pin
               1  consistent gets pin (fastpath)
               1  enqueue releases
               1  enqueue requests
               6  non-idle wait count
               2  opened cursors cumulative
               2  opened cursors current
               1  pinned cursors current
               1  recursive calls
               1  session logical reads
               6  user calls
```

Drop table and purge the recycle bin



```
Drop table T2;
select segment_name, segment_type from user_segments;
PURGE RECYCLEBIN;
select segment_name, segment_type from user_segments;
```

Task completed in 0.05 seconds

```
               1  consistent gets pin
               1  consistent gets pin (fastpath)
               6  non-idle wait count
               2  opened cursors cumulative
               2  opened cursors current
               1  pinned cursors current
               1  session logical reads
               6  user calls
Autotrace Disabled

Table T2 dropped.


RECYCLEBIN purged.
```

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	416	376	99999	380	Full Table
2	416	0	0	380	Empty Table
3	416	1	1	380	Table with 1 row
4	6	0	0	1	Truncate Table

When full scanning an object, all the blocks associated with that object must be retrieved and processed to determine if rows in a block match your query's needs. Oracle must read an entire block into memory in order to get to the row data stored in that block. So, when a full scan occurs, there are actually two things the optimizer needs to consider: how many blocks must be read and how much data in each block will be thrown away.

This example shows that regardless of whether the table is full, empty, or with 1 row, we would need 380 consistent gets from cache for a full table scan, which is inefficient from a cost and performance optimization perspective for tables with 1 row and empty. Even though almost all the rows have been deleted and some blocks have actually become totally unused, the highwater mark remains the same. When a full scan operation occurs, all blocks up to the highwater mark will be read in and scanned, even if they are empty. For tables that are frequently loaded and unloaded (using DELETE instead of TRUNCATE), we may discover that response time suffers.

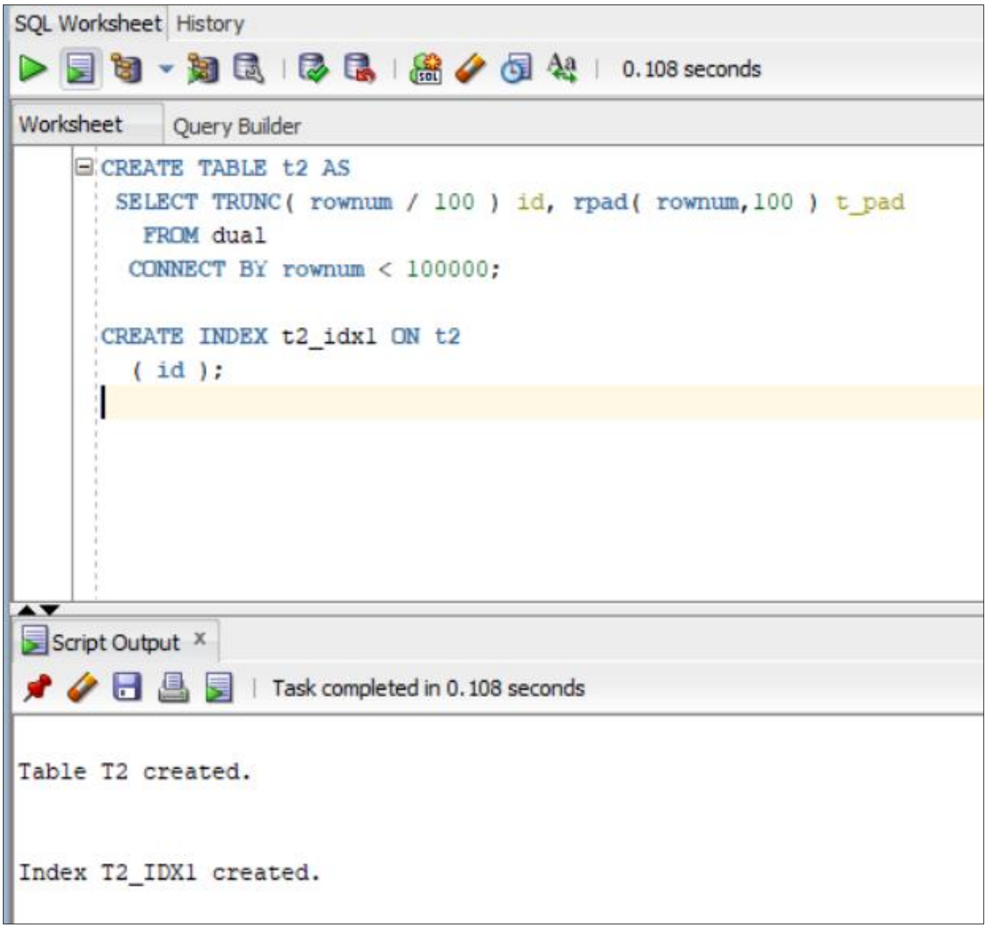
Code: Task 1

```
/*1.1. Task 1: Full Scans and the High-water Mark and Block reading*/
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 );

select blocks from user_segments where segment_name = 'T2';
select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;

set autotrace ON;
SELECT COUNT( * )
   FROM t2 ;
SET autotrace OFF;

DELETE FROM t2;

select blocks from user_segments where segment_name = 'T2';
select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;

SET autotrace ON;
SELECT COUNT( * )
   FROM t2 ;
SET autotrace OFF;

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 ;

SET autotrace ON;
SELECT COUNT( * )
   FROM t2 ;
SET autotrace OFF;

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 ;

SET autotrace ON;
SELECT COUNT( * )
   FROM t2 ;
SET autotrace OFF;

Drop table T2;
select segment_name, segment_type from user_segments;
PURGE RECYCLEBIN;
select segment_name, segment_type from user_segments;
```

2. Index Scan types

2.1. Task 2: Index Clustering factor parameter

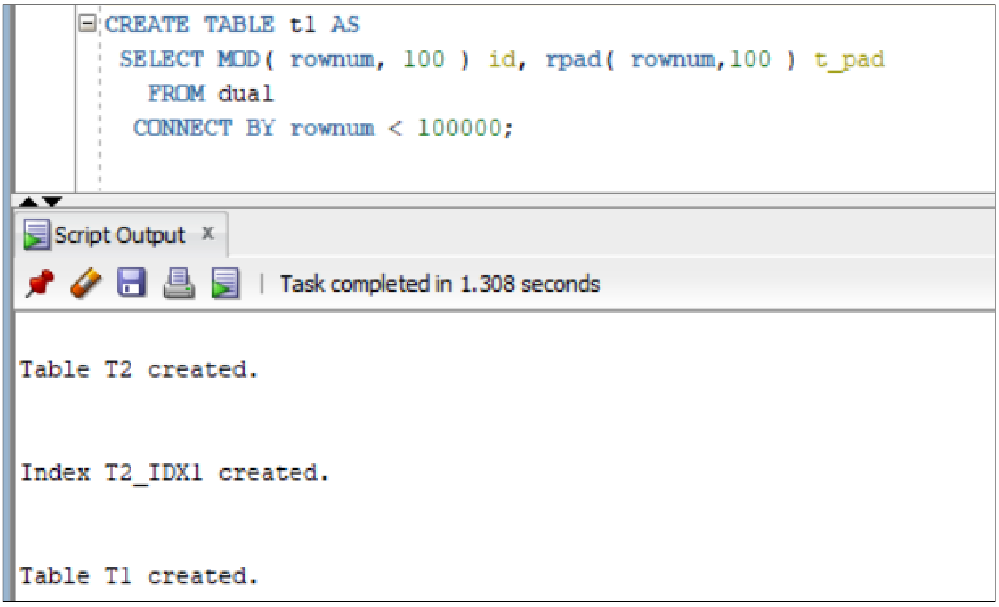
Step 1: Create table t2 as on task 1 step 1-2



```
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 );
```

```
Table T2 created.


Index T2_IDX1 created.
```

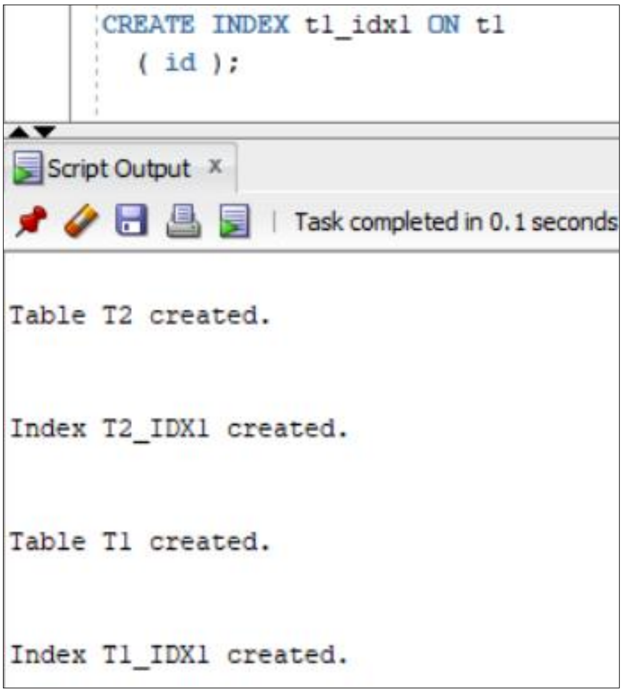
Step 2: Create table t1



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

```
Table T2 created.


Index T2_IDX1 created.


Table T1 created.
```

Step 3: Create index

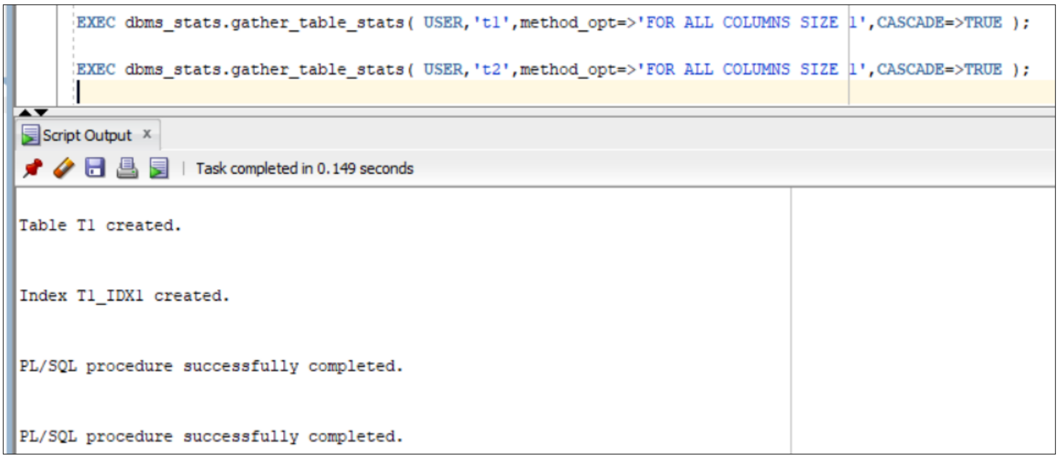


```
CREATE INDEX t1_idx1 ON t1
  ( id );
```

Task completed in 0.1 seconds

```
Table T2 created.


Index T2_IDX1 created.


Table T1 created.


Index T1_IDX1 created.
```

Step 4: Calculate statistic for both tables:

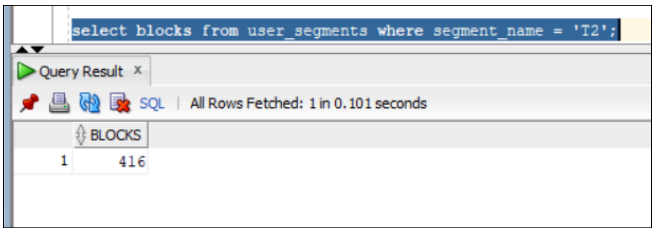


```
EXEC dbms_stats.gather_table_stats( USER,'t1',method_opt=>'FOR ALL COLUMNS SIZE 1',CASCADE=>TRUE );

EXEC dbms_stats.gather_table_stats( USER,'t2',method_opt=>'FOR ALL COLUMNS SIZE 1',CASCADE=>TRUE );
```

Task completed in 0.149 seconds

```
Table T1 created.


Index T1_IDX1 created.


PL/SQL procedure successfully completed.


PL/SQL procedure successfully completed.
```

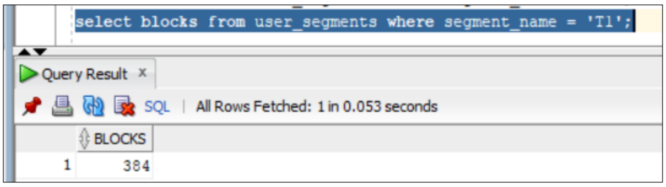
Block count:



```
select blocks from user_segments where segment_name = 'T2';
```

All Rows Fetched: 1 in 0.101 seconds

```
    BLOCKS
1      416
```

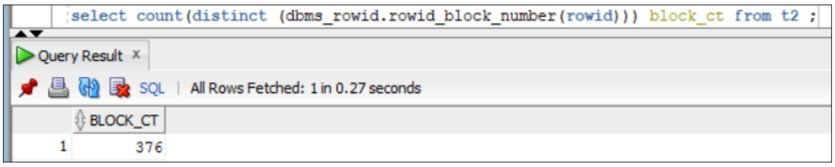


```
select blocks from user_segments where segment_name = 'T1';
```

All Rows Fetched: 1 in 0.053 seconds

```
    BLOCKS
1      384
```

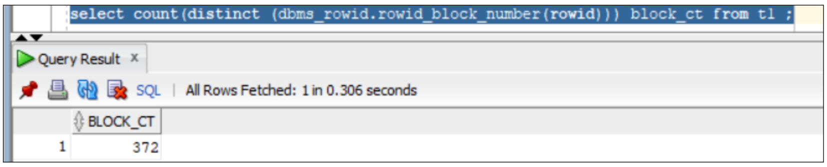
Used Block Count:



```
select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t2 ;
```

All Rows Fetched: 1 in 0.27 seconds

	BLOCK_CT
1	376

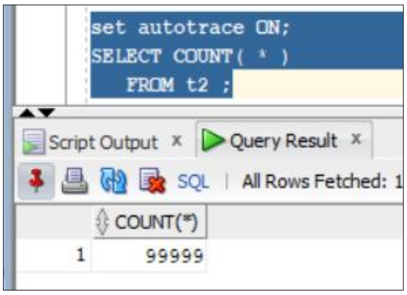


```
select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from t1 ;
```

All Rows Fetched: 1 in 0.306 seconds

	BLOCK_CT
1	372

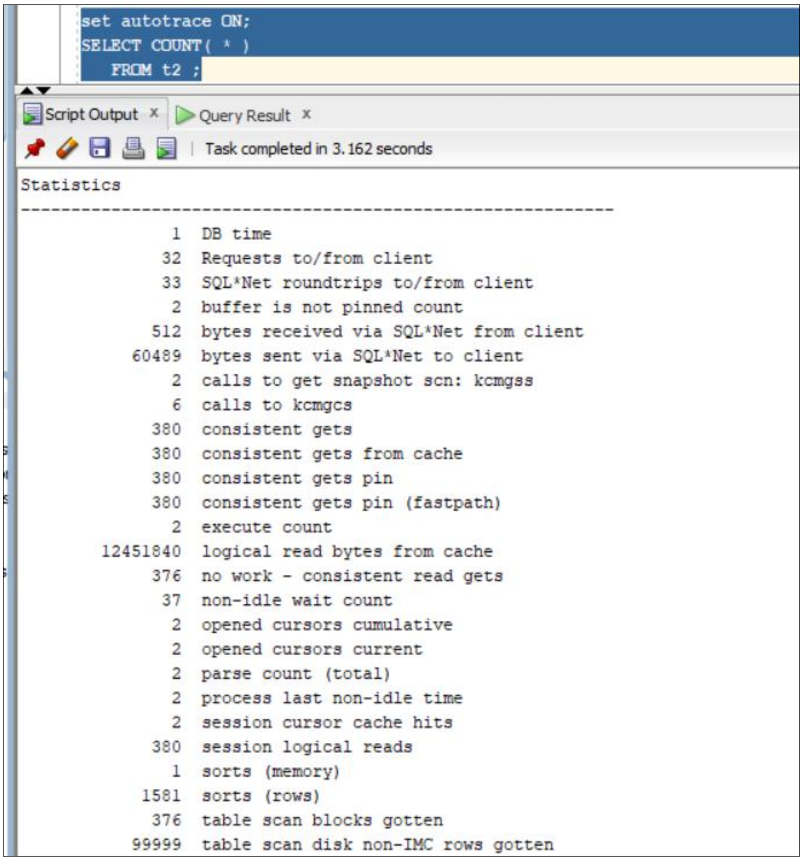
Explain Plan / Count rows:



```
set autotrace ON;
SELECT COUNT( * )
   FROM t2 ;
```

All Rows Fetched: 1

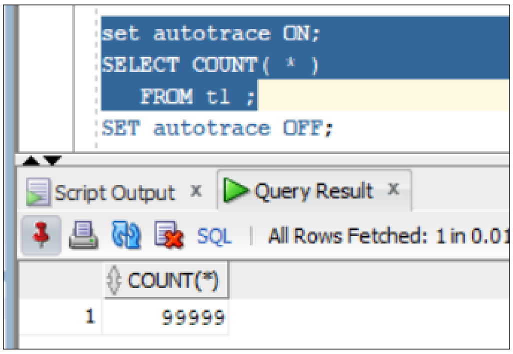
	COUNT(*)
1	99999



```
set autotrace ON;
SELECT COUNT( * )
   FROM t2 ;
```

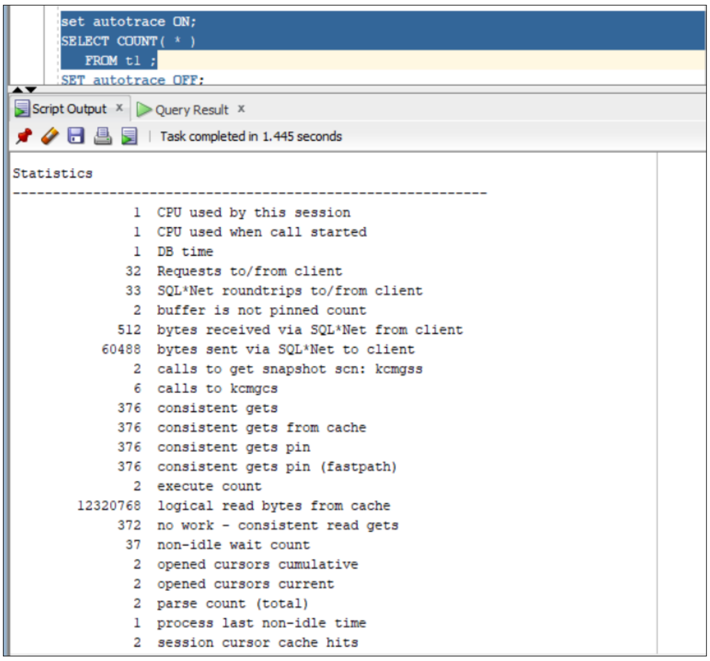
Task completed in 3.162 seconds

```
Statistics
-----------------------------------------------------------
               1  DB time
              32  Requests to/from client
              33  SQL*Net roundtrips to/from client
               2  buffer is not pinned count
             512  bytes received via SQL*Net from client
           60489  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
              37  non-idle wait count
               2  opened cursors cumulative
               2  opened cursors current
               2  parse count (total)
               2  process last non-idle time
               2  session cursor cache hits
             380  session logical reads
               1  sorts (memory)
            1581  sorts (rows)
             376  table scan blocks gotten
           99999  table scan disk non-IMC rows gotten
```



```
set autotrace ON;
SELECT COUNT( * )
   FROM t1 ;
SET autotrace OFF;
```

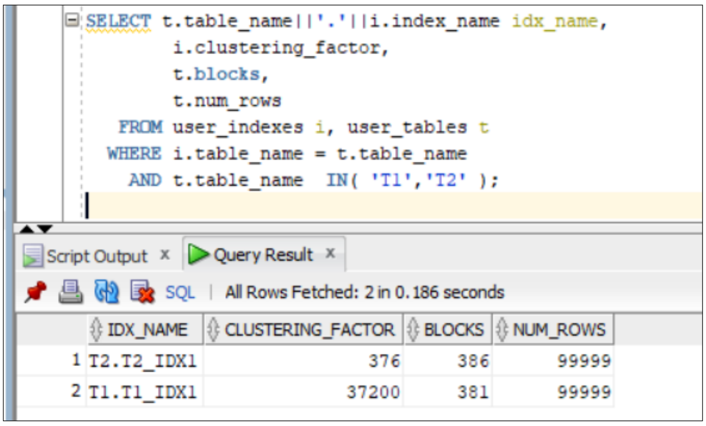
	COUNT(*)
1	99999



```
set autotrace ON;
SELECT COUNT( * )
   FROM t1 ;
SET autotrace OFF;
```

```
Statistics
-----------------------------------------------------------
               1  CPU used by this session
               1  CPU used when call started
               1  DB time
              32  Requests to/from client
              33  SQL*Net roundtrips to/from client
               2  buffer is not pinned count
             512  bytes received via SQL*Net from client
           60488  bytes sent via SQL*Net to client
               2  calls to get snapshot scn: kcmgss
               6  calls to kcmgcs
             376  consistent gets
             376  consistent gets from cache
             376  consistent gets pin
             376  consistent gets pin (fastpath)
               2  execute count
        12320768  logical read bytes from cache
             372  no work - consistent read gets
              37  non-idle wait count
               2  opened cursors cumulative
               2  opened cursors current
               2  parse count (total)
               1  process last non-idle time
               2  session cursor cache hits
```

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' );
```

	IDX_NAME	CLUSTERING_FACTOR	BLOCKS	NUM_ROWS
1	T2.T2_IDX1	376	386	99999
2	T1.T1_IDX1	37200	381	99999

Summary:

- The clustering factor is a measure of how well ordered the table data is as related to the indexed values. It is used to check the cost of a table lookup following an index access (multiplying the clustering factor by index's selectivity gives the cost of the operation). The clustering factor records the number of blocks that will be read when scanning the index. If the index being used has a large clustering factor, then more table data blocks have to be visited to get the rows in each index block (because adjacent rows are in different blocks). If the clustering factor is close to the number of blocks in the table, then the index is well ordered, but if the clustering factor is close to the number of rows in the table, then the index is not well ordered.

The clustering factor is computed by the following:

- The index is scanned in order.
- The block portion of the ROWID pointed at by the current indexed valued is compared to the previous indexed value (comparing adjacent rows in the index).
- If the ROWIDs point to different TABLE blocks, the clustering factor is incremented (this is done for the entire index).

When the clustering factor is higher, the cost of index range scan is higher.

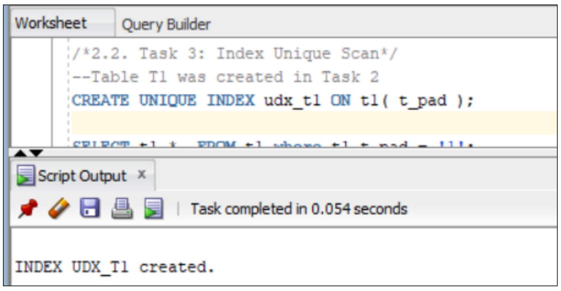
- We have different values for indexes t1_idx1 and t2_idx1. Recall that index entries are stored in sorted order while table data is stored in random order. The clustering factor of an index indicates to the optimizer if data rows containing the same indexed values will be located in the same or a small set of contiguous blocks, or if rows will be scattered across numerous table blocks. We can notice that an index for table T2 would have a lower clustering factor. Lower numbers that are closer to the number of table blocks are used to indicate highly ordered, or clustered, rows of data based on the indexed value. The clustering factor for table T1, however, would be higher and typically closer to the number of rows in the table.
- We can see that index t2_idx1 has best selective performance in execution Select clause filtered by IN (, list of values,).

Code: Task 2

```
/*2.1. Task 2: Index Clustering factor parameter*/
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 );

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

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' );
```

2.2. Task 3: Index Unique Scan

Step 1: Create Unique Index

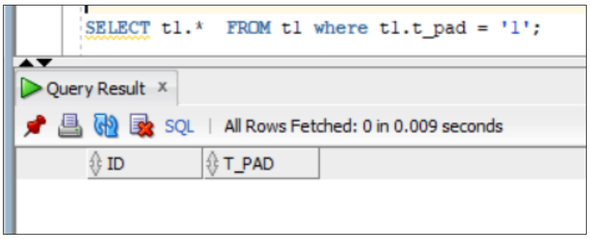


```
/*2.2. Task 3: Index Unique Scan*/
--Table T1 was created in Task 2
CREATE UNIQUE INDEX udx_t1 ON t1( t_pad );

SELECT t1.*  FROM t1 where t1.t_pad = '1';
```

Task completed in 0.054 seconds

```
INDEX UDX_T1 created.
```

Step 2: Data results



```
SELECT t1.*  FROM t1 where t1.t_pad = '1';
```

All Rows Fetched: 0 in 0.009 seconds

ID	T_PAD

Drop table and purge the recycle bin

Summary:

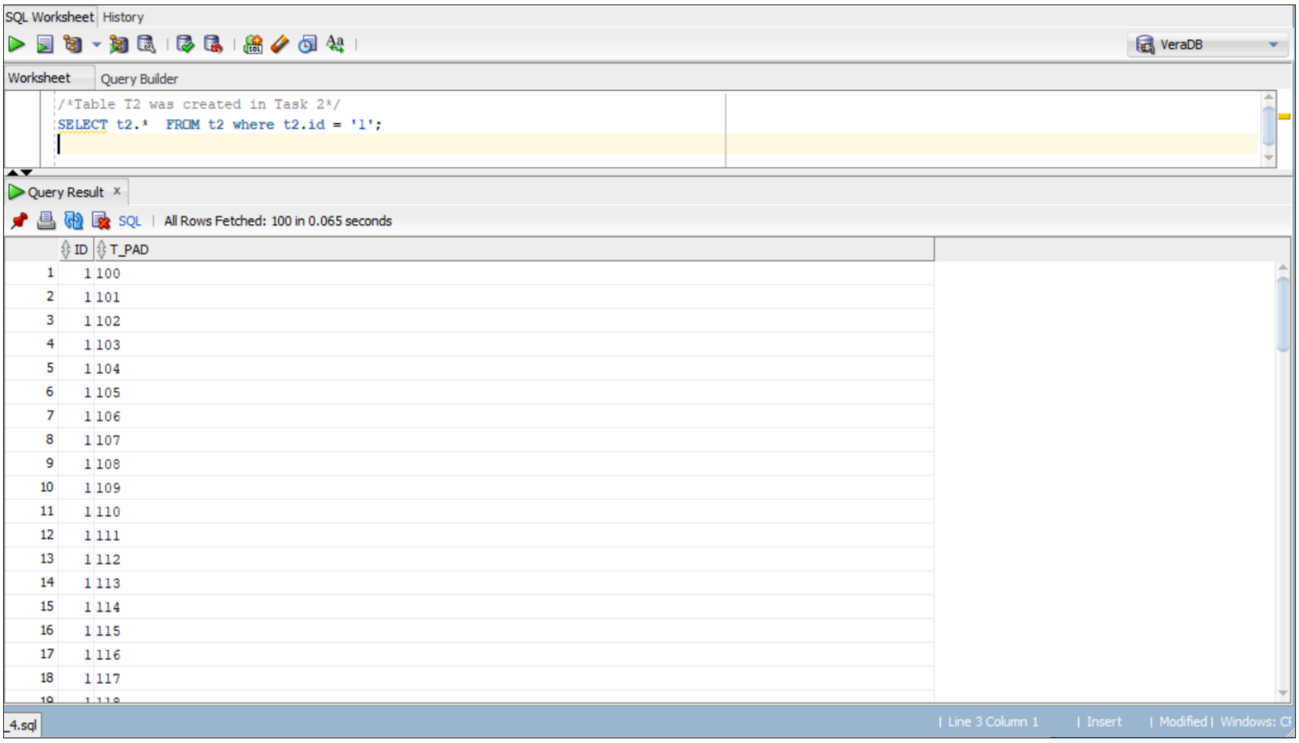
An index unique scan is chosen when a predicate contains a condition using a column defined with a UNIQUE or PRIMARY KEY index. These types of indexes guarantee that only one row will ever be returned for a specified value. In this cases, the index structure will be traversed from root to leaf block to a single entry, retrieve the rowid, and use it to access the table data block containing the one row.
Column t1.t_pad has a string value created by the RPAD function, which returns a string value of 100 padded_length.
Oracle read t1.t_pad in step 2 as a string with a value of 1 character, but not of 100 characters, so we saw an empty block.

Code: Task 3

```
/*2.2. Task 3: Index Unique Scan*/
--Table T1 was created in Task 2
CREATE UNIQUE INDEX udx_t1 ON t1( t_pad );

SELECT t1.*  FROM t1 where t1.t_pad = '1';

Drop table T1;
select segment_name, segment_type from user_segments;
PURGE RECYCLEBIN;
select segment_name, segment_type from user_segments;
```

2.3. Task 4: Index Range Scan

Step 1: Data results



```
/*Table T2 was created in Task 2*/
SELECT t2.*  FROM t2 where t2.id = '1';
```

All Rows Fetched: 100 in 0.065 seconds

	ID	T_PAD
1	1	100
2	1	101
3	1	102
4	1	103
5	1	104
6	1	105
7	1	106
8	1	107
9	1	108
10	1	109
11	1	110
12	1	111
13	1	112
14	1	113
15	1	114
16	1	115
17	1	116
18	1	117
19	1	118

Drop table and purge the recycle bin

Summary:

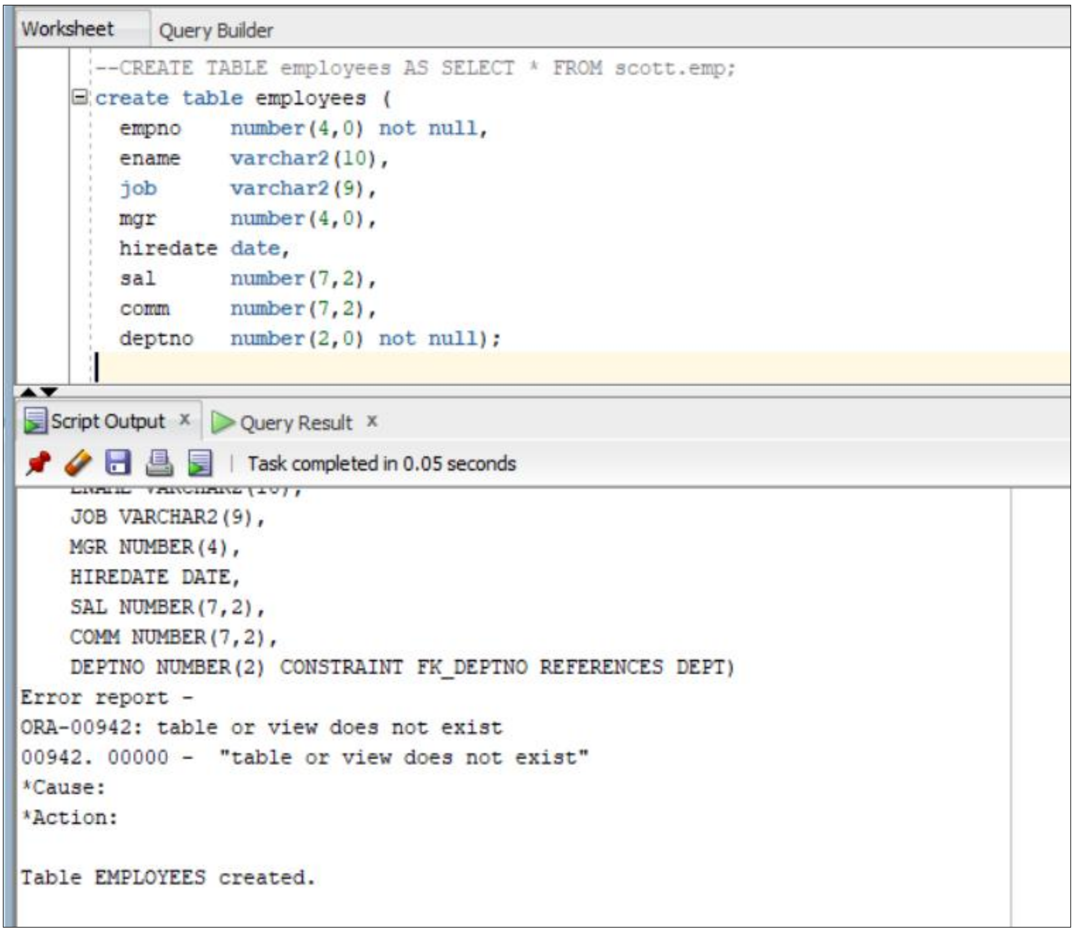
An index range scan is chosen when a predicate contains a condition that will return a range of data.
The index can be unique or non-unique as it is the condition that determines whether or not multiple rows will be returned or not.
Oracle read all t2.t_pad values (from 100 to 199) for t2.id = 1 in step 1.

Code: Task 4

```
/*2.3. Task 4: Index Range Scan*/
--Table T2 was created in Task 2
SELECT t2.*  FROM t2 where t2.id = '1';

Drop table T2;
select segment_name, segment_type from user_segments;
PURGE RECYCLEBIN;
select segment_name, segment_type from user_segments;
```

2.4. Task 5: Index Skip Scan

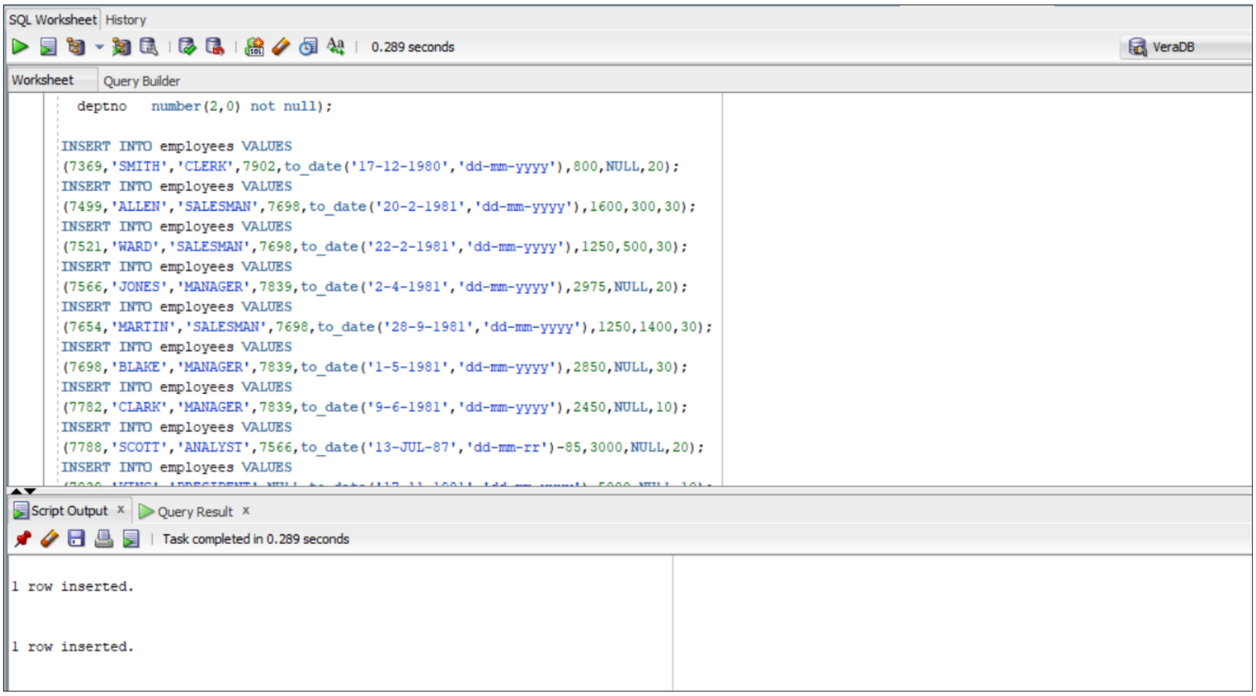
Step 1: Create table



```
--CREATE TABLE employees AS SELECT * FROM scott.emp;
create table employees (
  empno     number(4,0) not null,
  ename     varchar2(10),
  job       varchar2(9),
  mgr       number(4,0),
  hiredate date,
  sal       number(7,2),
  comm      number(7,2),
  deptno    number(2,0) not null);
```

```
    JOB VARCHAR2(9),
    MGR NUMBER(4),
    HIREDATE DATE,
    SAL NUMBER(7,2),
    COMM NUMBER(7,2),
    DEPTNO NUMBER(2) CONSTRAINT FK_DEPTNO REFERENCES DEPT)
Error report -
ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:

Table EMPLOYEES created.
```

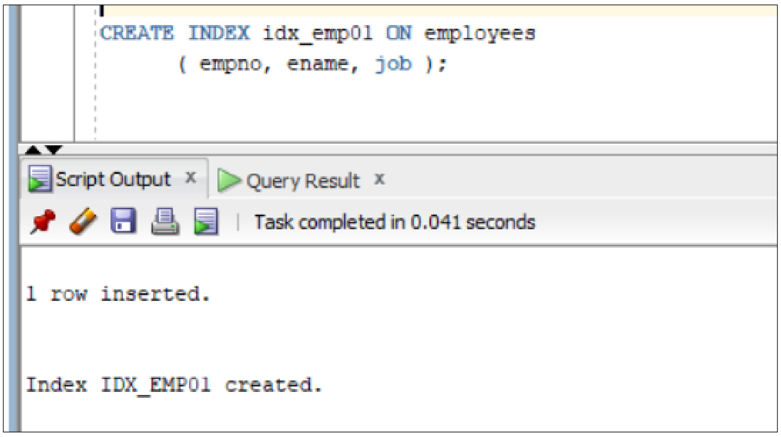
Insert values into table



```
  deptno   number(2,0) not null);

INSERT INTO employees VALUES
(7369,'SMITH','CLERK',7902,to_date('17-12-1980','dd-mm-yyyy'),800,NULL,20);
INSERT INTO employees VALUES
(7499,'ALLEN','SALESMAN',7698,to_date('20-2-1981','dd-mm-yyyy'),1600,300,30);
INSERT INTO employees VALUES
(7521,'WARD','SALESMAN',7698,to_date('22-2-1981','dd-mm-yyyy'),1250,500,30);
INSERT INTO employees VALUES
(7566,'JONES','MANAGER',7839,to_date('2-4-1981','dd-mm-yyyy'),2975,NULL,20);
INSERT INTO employees VALUES
(7654,'MARTIN','SALESMAN',7698,to_date('28-9-1981','dd-mm-yyyy'),1250,1400,30);
INSERT INTO employees VALUES
(7698,'BLAKE','MANAGER',7839,to_date('1-5-1981','dd-mm-yyyy'),2850,NULL,30);
INSERT INTO employees VALUES
(7782,'CLARK','MANAGER',7839,to_date('9-6-1981','dd-mm-yyyy'),2450,NULL,10);
INSERT INTO employees VALUES
(7788,'SCOTT','ANALYST',7566,to_date('13-JUL-87','dd-mm-rr')-85,3000,NULL,20);
INSERT INTO employees VALUES
```

```
1 row inserted.


1 row inserted.
```

Step 2: Create Index



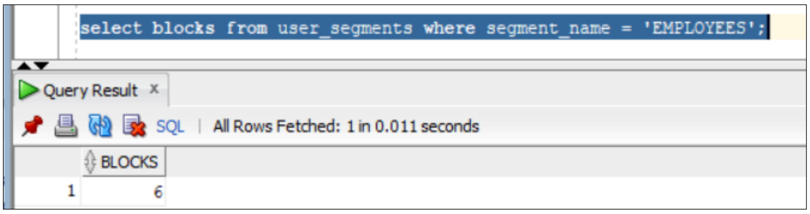
```
CREATE INDEX idx_emp01 ON employees
        ( empno, ename, job );
```

Task completed in 0.041 seconds

```
1 row inserted.


Index IDX_EMP01 created.
```

Step 3: Get trace and statistic of explain plan

Block count:

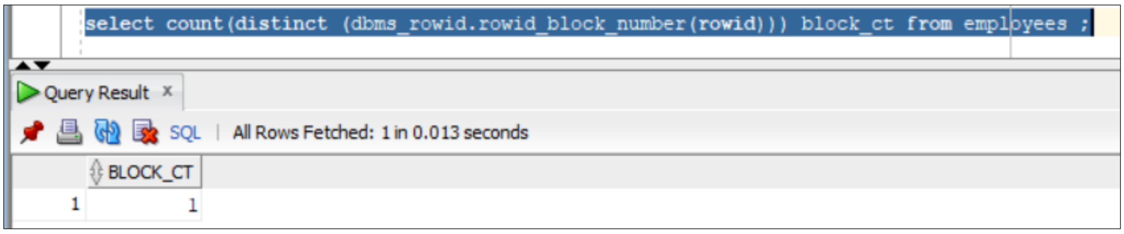


```
select blocks from user_segments where segment_name = 'EMPLOYEES';
```

All Rows Fetched: 1 in 0.011 seconds

```
   BLOCKS
1       6
```

Used Block Count:

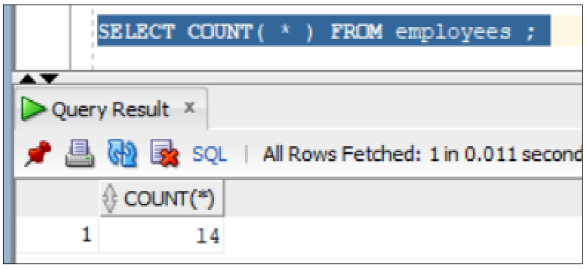


```
select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from employees ;
```

All Rows Fetched: 1 in 0.013 seconds

```
   BLOCK_CT
1         1
```

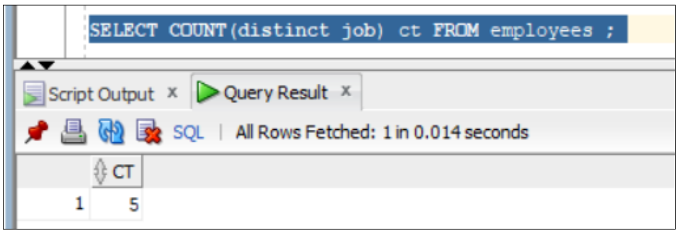
Rows count:



```
SELECT COUNT( * ) FROM employees ;
```

All Rows Fetched: 1 in 0.011 second

```
   COUNT(*)
1        14
```

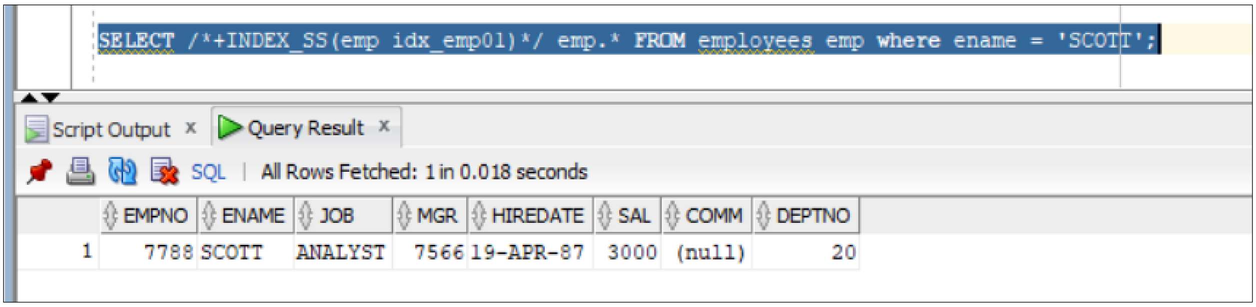


```
SELECT COUNT(distinct job) ct FROM employees ;
```

All Rows Fetched: 1 in 0.014 seconds

```
   CT
1   5
```

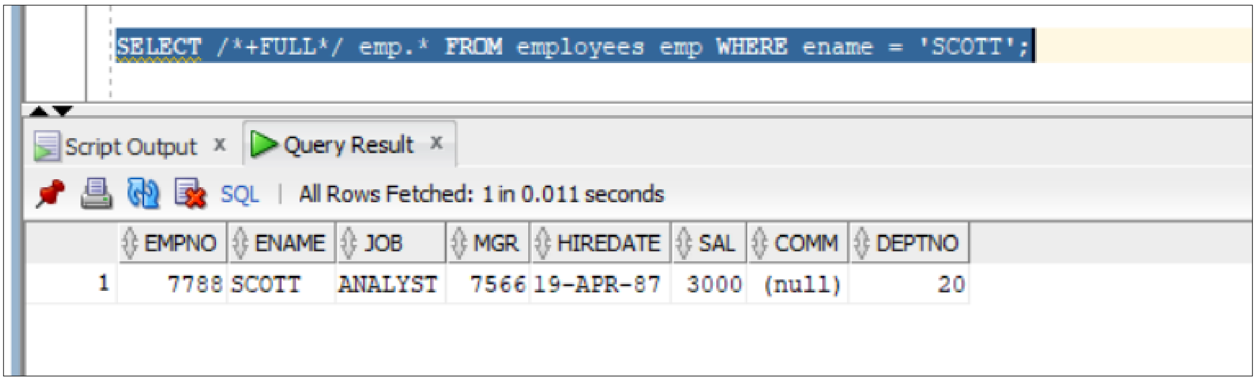
Explain Plan:



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

All Rows Fetched: 1 in 0.018 seconds

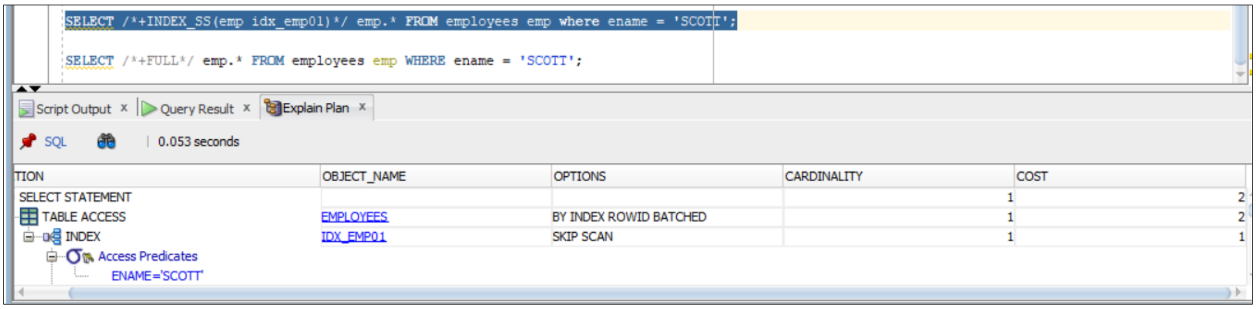
	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7788	SCOTT	ANALYST	7566	19-APR-87	3000	(null)	20



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

All Rows Fetched: 1 in 0.011 seconds

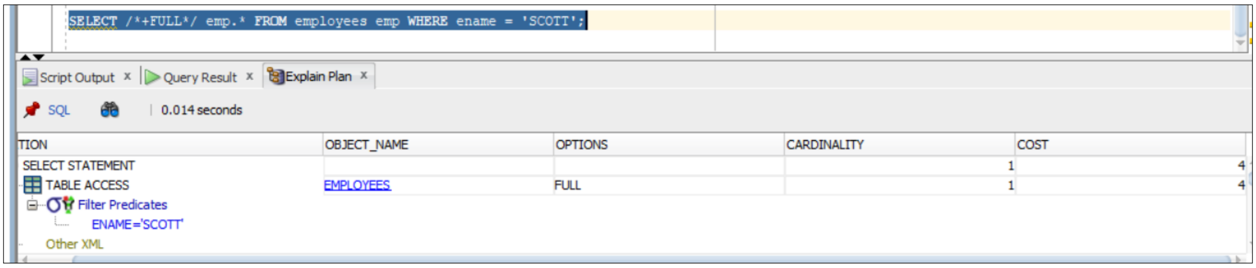
	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7788	SCOTT	ANALYST	7566	19-APR-87	3000	(null)	20



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

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

0.053 seconds

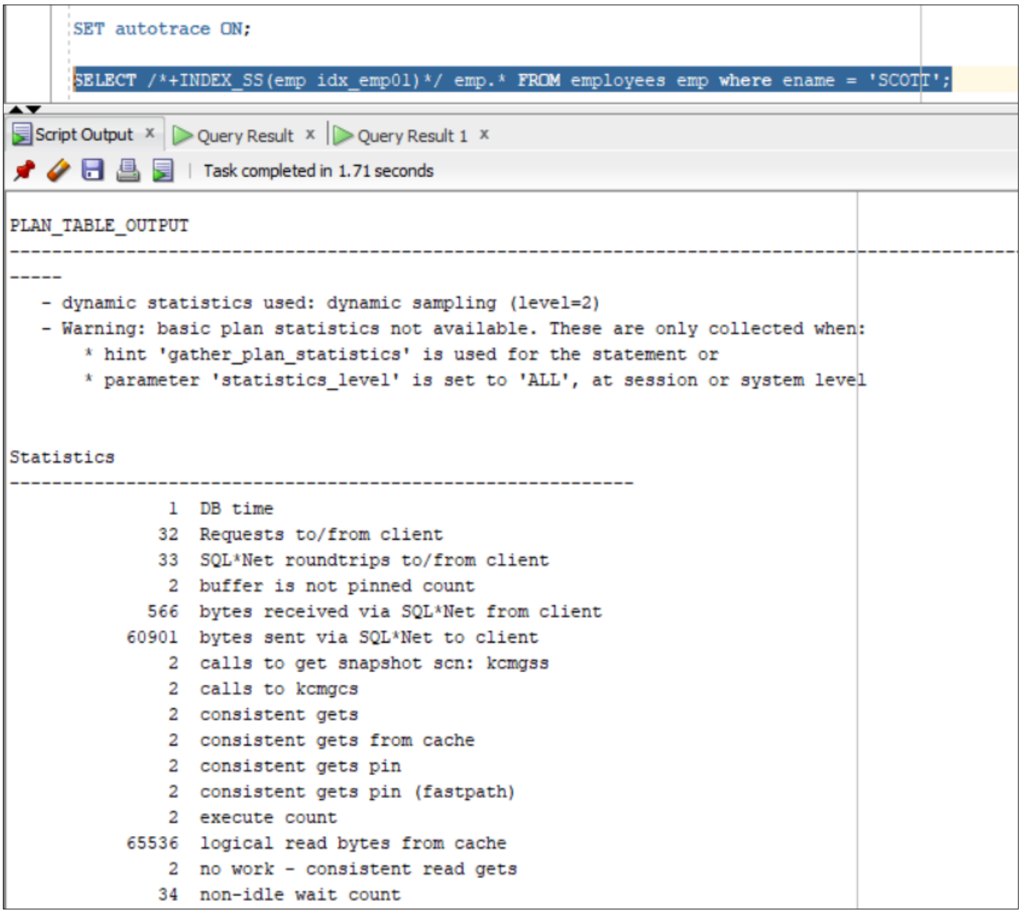
TION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	2
TABLE ACCESS	EMPLOYEES	BY INDEX ROWID BATCHED	1	2
INDEX	IDX_EMP01	SKIP SCAN	1	1
Access Predicates				
ENAME='SCOTT'				



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

0.014 seconds

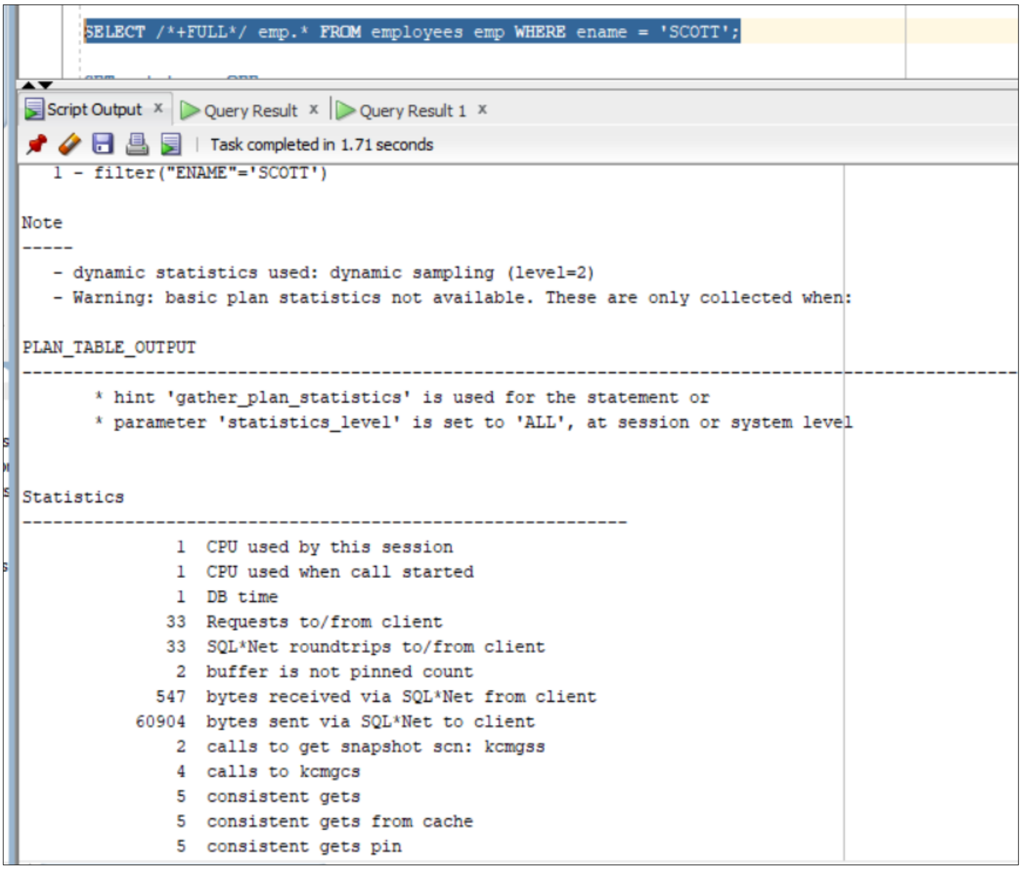
TION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	4
TABLE ACCESS	EMPLOYEES	FULL	1	4
Filter Predicates				
ENAME='SCOTT'				
Other XML				



```
SET autotrace ON;

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

Task completed in 1.71 seconds

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - Warning: basic plan statistics not available. These are only collected when:
       * hint 'gather_plan_statistics' is used for the statement or
       * parameter 'statistics_level' is set to 'ALL', at session or system level


Statistics
-----------------------------------------------------------
               1  DB time
              32  Requests to/from client
              33  SQL*Net roundtrips to/from client
               2  buffer is not pinned count
             566  bytes received via SQL*Net from client
           60901  bytes sent via SQL*Net to client
               2  calls to get snapshot scn: kcmgss
               2  calls to kcmgcs
               2  consistent gets
               2  consistent gets from cache
               2  consistent gets pin
               2  consistent gets pin (fastpath)
               2  execute count
           65536  logical read bytes from cache
               2  no work - consistent read gets
              34  non-idle wait count
```



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

Task completed in 1.71 seconds

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

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - Warning: basic plan statistics not available. These are only collected when:

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------
       * hint 'gather_plan_statistics' is used for the statement or
       * parameter 'statistics_level' is set to 'ALL', at session or system level


Statistics
-----------------------------------------------------------
               1  CPU used by this session
               1  CPU used when call started
               1  DB time
              33  Requests to/from client
              33  SQL*Net roundtrips to/from client
               2  buffer is not pinned count
             547  bytes received via SQL*Net from client
           60904  bytes sent via SQL*Net to client
               2  calls to get snapshot scn: kcmgss
               4  calls to kcmgcs
               5  consistent gets
               5  consistent gets from cache
               5  consistent gets pin
```

Drop table and purge the recycle bin

Summary:

- An index skip scan is chosen when the predicate contains a condition on a non-leading column in an index and the leading columns are fairly distinct. A skip scan works by logically splitting a multi-column index into smaller subindexes. The number of logical subindexes is determined by the number of distinct values in the leading columns of the index. Therefore, the more distinct the leading columns are, the more logical subindexes would need to be created. If too many subindexes would be required, the operation won't be as efficient as simply doing a full scan. However, in the cases where the number of subindexes needed would be smaller, the operation can be many times more efficient than a full scan as scanning smaller index blocks can be more efficient than scanning larger table blocks.
- Summary table with all result and text description of analyses this results.

№	Count of Blocks	Count of Used Blocks	Count of Rows	Consistent gets	Cost	Description
1	6	1	14	5	4	Full Scan: /*+FULL*/
2	6	1	14	2	2	Index Skip Scan: /*+INDEX_SS(emp idx_emp01)*/

In this example, a full table scan is more expensive than an Index Skip Scan, including the fact that a full scan requires 5 consistent gets from cache, versus 2 consistent gets from cache for an Index Skip Scan.
As you can see, an Index Skip Scan is much more efficient. What happened was that the index was logically divided into 5 subindexes (by job) and each subindex was scanned for a match for ename = 'SCOTT'.
For this index scan type, the fewer distinct values the leading column (or columns) have, the fewer logical subindexes will be needed and therefore the fewer total block accesses required.

Code: Task 5

```
/*2.4. Task 5: Index Skip Scan*/
--CREATE TABLE employees AS SELECT * FROM scott.emp;
create table employees (
  empno    number(4,0) not null,
  ename    varchar2(10),
  job      varchar2(9),
  mgr      number(4,0),
  hiredate date,
  sal      number(7,2),
  comm     number(7,2),
  deptno   number(2,0) not null);

INSERT INTO employees VALUES
(7369,'SMITH','CLERK',7902,to_date('17-12-1980','dd-mm-yyyy'),800,NULL,20);
INSERT INTO employees VALUES
(7499,'ALLEN','SALESMAN',7698,to_date('20-2-1981','dd-mm-yyyy'),1600,300,30);
INSERT INTO employees VALUES
(7521,'WARD','SALESMAN',7698,to_date('22-2-1981','dd-mm-yyyy'),1250,500,30);
INSERT INTO employees VALUES
(7566,'JONES','MANAGER',7839,to_date('2-4-1981','dd-mm-yyyy'),2975,NULL,20);
INSERT INTO employees VALUES
(7654,'MARTIN','SALESMAN',7698,to_date('28-9-1981','dd-mm-yyyy'),1250,1400,30);
INSERT INTO employees VALUES
(7698,'BLAKE','MANAGER',7839,to_date('1-5-1981','dd-mm-yyyy'),2850,NULL,30);
INSERT INTO employees VALUES
(7782,'CLARK','MANAGER',7839,to_date('9-6-1981','dd-mm-yyyy'),2450,NULL,10);
INSERT INTO employees VALUES
(7788,'SCOTT','ANALYST',7566,to_date('13-JUL-87','dd-mm-rr')-85,3000,NULL,20);
INSERT INTO employees VALUES
(7839,'KING','PRESIDENT',NULL,to_date('17-11-1981','dd-mm-yyyy'),5000,NULL,10);
INSERT INTO employees VALUES
(7844,'TURNER','SALESMAN',7698,to_date('8-9-1981','dd-mm-yyyy'),1500,0,30);
INSERT INTO employees VALUES
(7876,'ADAMS','CLERK',7788,to_date('13-JUL-87', 'dd-mm-rr')-51,1100,NULL,20);
INSERT INTO employees VALUES
(7900,'JAMES','CLERK',7698,to_date('3-12-1981','dd-mm-yyyy'),950,NULL,30);
INSERT INTO employees VALUES
(7902,'FORD','ANALYST',7566,to_date('3-12-1981','dd-mm-yyyy'),3000,NULL,20);
INSERT INTO employees VALUES
(7934,'MILLER','CLERK',7782,to_date('23-1-1982','dd-mm-yyyy'),1300,NULL,10);

CREATE INDEX idx_emp01 ON employees
    ( empno, ename, job );

select blocks from user_segments where segment_name = 'EMPLOYEES';

select count(distinct (dbms_rowid.rowid_block_number(rowid))) block_ct from employees ;

SELECT COUNT( * ) FROM employees ;
SELECT COUNT(distinct job) ct FROM employees ;

SET autotrace ON;
SELECT /*+INDEX_SS(emp idx_emp01)*/ emp.* FROM employees emp where ename = 'SCOTT';
SET autotrace OFF;

SET autotrace ON;
SELECT /*+FULL*/ emp.* FROM employees emp WHERE ename = 'SCOTT';
SET autotrace OFF;

Drop table employees;
select segment_name, segment_type from user_segments;
PURGE RECYCLEBIN;
select segment_name, segment_type from user_segments;
```