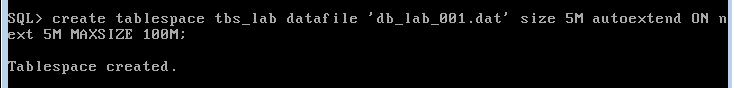
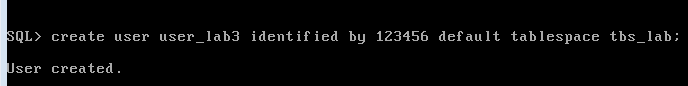
# Prerequisited Task:

Connect as system user and Create new tablespace tbs\_lab with new datafile db\_lab\_001.dat:   
  
create tablespace tbs\_lab datafile 'db\_lab\_001.dat' size 5M autoextend ON next 5M MAXSIZE 100M;



Create new user:

create user $username$ identified by 123456 default tablespace tbs\_lab;



Grant Connect Role and Resource Role:

grant connect to $username$;

grant resource to $username$;

grant select on scott.dept to $username$;

grant select on scott.emp to $username$;

# Heap Organized Tables

## Task 1 – Heap Understanding

A heap is a classic data structure studied in computer science. It is basically a big area of space, disk,

or memory (disk in the case of a database table, of course) that is managed in an apparently random fashion. Data will be placed where it fits best, rather than in any specific sort of order. Many people expect data to come back out of a table in the same order it was put into it, but with a heap, this is definitely not assured. In fact, rather the opposite is guaranteed: the rows will come out in a wholly unpredictable order.

Step 1:

create table t

( a int,

b varchar2(4000) default rpad('\*',4000,'\*'),

c varchar2(3000) default rpad('\*',3000,'\*')

)

/

Step 2:

insert into t (a) values ( 1);

insert into t (a) values ( 2);

insert into t (a) values ( 3);

commit;

delete from t where a = 2 ;

commit;

insert into t (a) values ( 4);

commit;

**Step 3:**

select a from t;

**Task Results:**

Expected:

select a from t;

A

----------

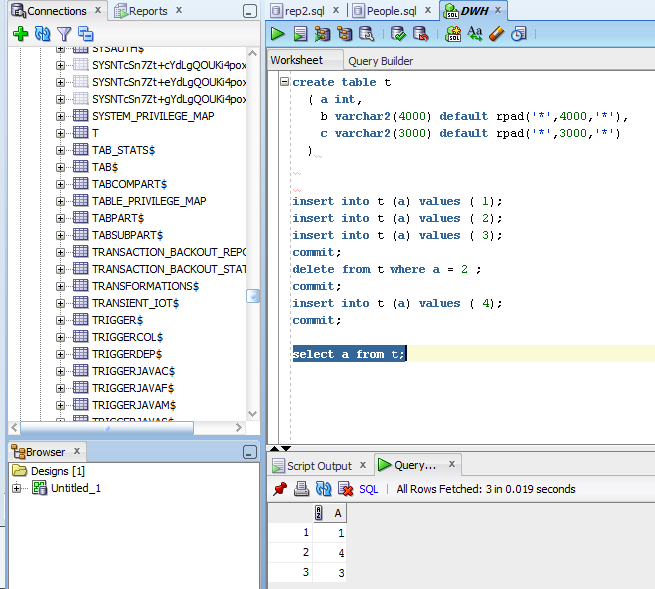
1

4

3

Screenshot of data results below:

Oracle SQL Developer:



**NOTE:**  Adjust columns B and C to be appropriate for your block size if you would like to reproduce this. For example, if you have a 2KB block size, you do not need column C, and column B should be a VARCHAR2(1500) with a default of 1,500 asterisks. Since data is managed in a heap in a table like this, as space becomes available, it will be reused.

Clean up:

# drop table T;

## Task 2 – Understanding Low level of data abstraction: Heap Table Segments

Step 1:

# Create table t ( x int primary key, y clob, z blob );

Step 2:

# select segment\_name, segment\_type 2 from user\_segments;

Step 3:

# Create table t

( x int primary key,

y clob,

z blob )

SEGMENT CREATION IMMEDIATE

/

**Step 4:**

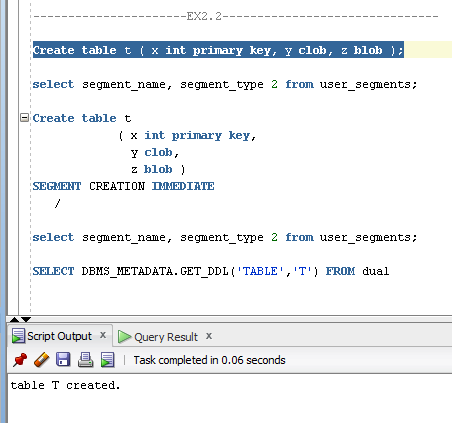
# select segment\_name, segment\_type 2 from user\_segments;

Step 5:

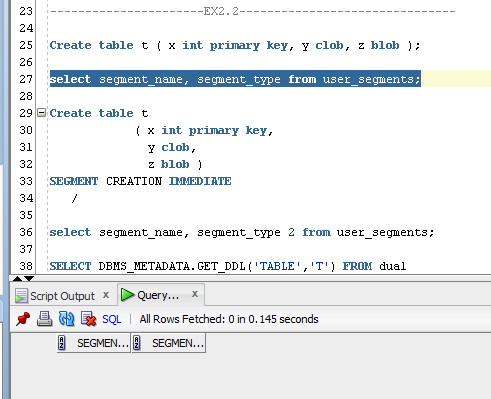
# SELECT DBMS\_METADATA.GET\_DDL('TABLE','T') FROM dual

**Task Results:**

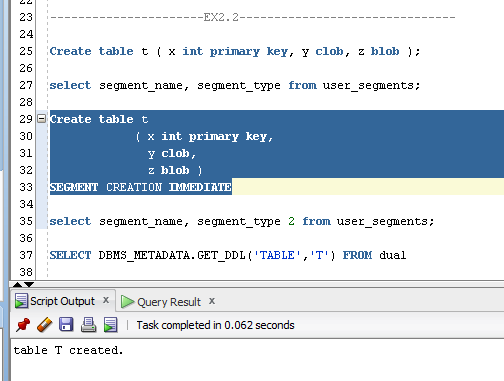
Step 1: Table Created;



Step 2: Empty selection



Step 3: Table Created (1 Tip);

****

**Step 4**: Expected Result:

SEGMENT\_NAME SEGMENT\_TYPE

------------------------------ ------------------

T TABLE

SYS\_IL0000093076C00002$$ LOBINDEX

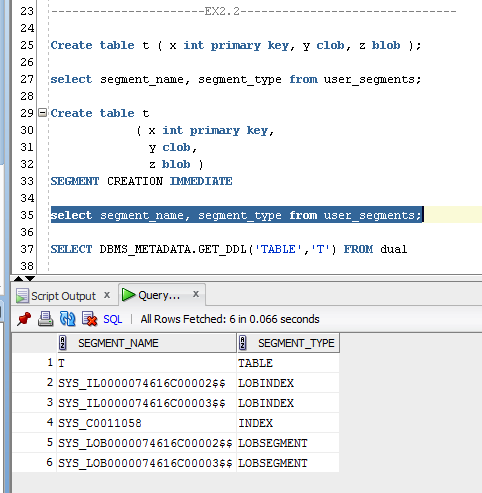
SYS\_IL0000093076C00003$$ LOBINDEX

SYS\_C0019048 INDEX

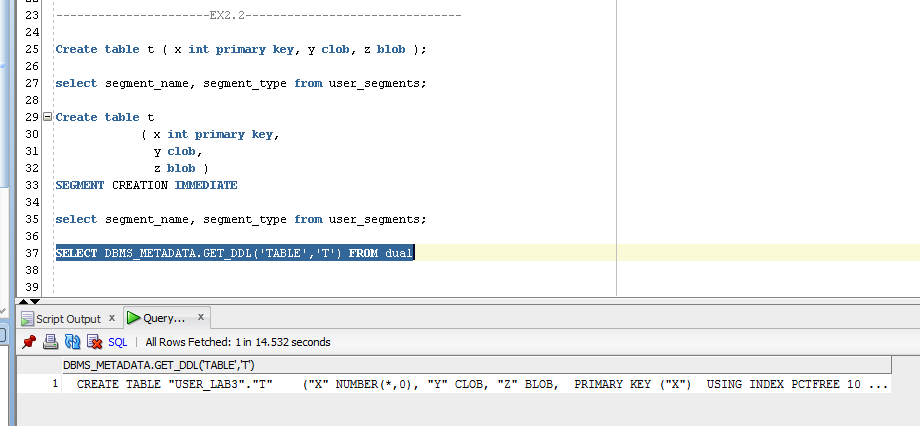
SYS\_LOB0000093076C00002$$ LOBSEGMENT

SYS\_LOB0000093076C00003$$ LOBSEGMENT

6 rows selected.



**Step 5 results**



# Index Organized Tables

## Task 3: Compare performance of using IOT tables

Step 1:

CREATE TABLE emp AS

SELECT

object\_id empno

, object\_name ename

, created hiredate

, owner job

FROM

all\_objects

/

Create Index:

# alter table emp add constraint emp\_pk primary key(empno)

Calculate Statistic:

begin

dbms\_stats.gather\_table\_stats( user, 'EMP', cascade=>true );

end;

Step 2:

CREATE TABLE heap\_addresses

(

empno REFERENCES emp(empno) ON DELETE CASCADE

, addr\_type VARCHAR2(10)

, street VARCHAR2(20)

, city VARCHAR2(20)

, state VARCHAR2(2)

, zip NUMBER

, PRIMARY KEY (empno,addr\_type)

)

/

Step 3:

CREATE TABLE iot\_addresses

(

empno REFERENCES emp(empno) ON DELETE CASCADE

, addr\_type VARCHAR2(10)

, street VARCHAR2(20)

, city VARCHAR2(20)

, state VARCHAR2(2)

, zip NUMBER

, PRIMARY KEY (empno,addr\_type)

)

ORGANIZATION INDEX

/

Step 4: Initial inserts:

INSERT INTO heap\_addresses

SELECT empno, 'WORK' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

INSERT INTO iot\_addresses

SELECT empno , 'WORK' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

--

INSERT INTO heap\_addresses

SELECT empno, 'HOME' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

INSERT INTO iot\_addresses

SELECT empno, 'HOME' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

--

INSERT INTO heap\_addresses

SELECT empno, 'PREV' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

INSERT INTO iot\_addresses

SELECT empno, 'PREV' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

--

INSERT INTO heap\_addresses

SELECT empno, 'SCHOOL' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

INSERT INTO iot\_addresses

SELECT empno, 'SCHOOL' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

Commit;

Step 5: Calculate statistic:

exec dbms\_stats.gather\_table\_stats( $username$, 'HEAP\_ADDRESSES' );

exec dbms\_stats.gather\_table\_stats( $username$, 'IOT\_ADDRESSES' );

**Step 6:** Compare Trace and Performance:

Explain 1:

SELECT \*

FROM emp ,

heap\_addresses

WHERE emp.empno = heap\_addresses.empno

AND emp.empno = 42;

Explain 2:

SELECT \*

FROM emp ,

iot\_addresses

WHERE emp.empno = iot\_addresses.empno

AND emp.empno = 42;

Compare results and explain Cost value calculation and difference on execution plan.

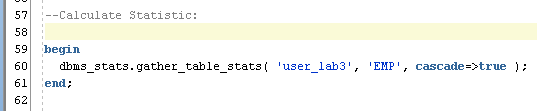
Step 7: Drop all tables;

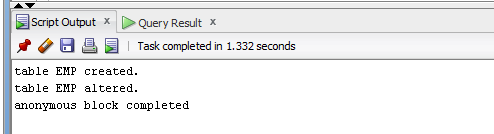
**Task Results:**

Step 1-3: Tables Created;

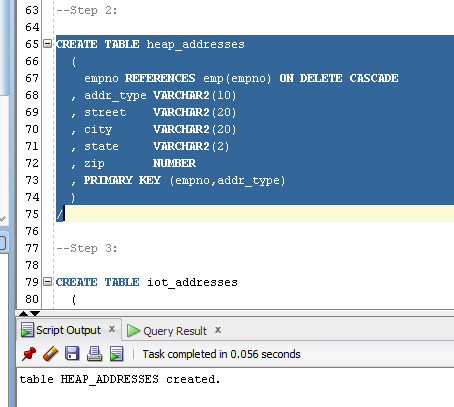
Step1



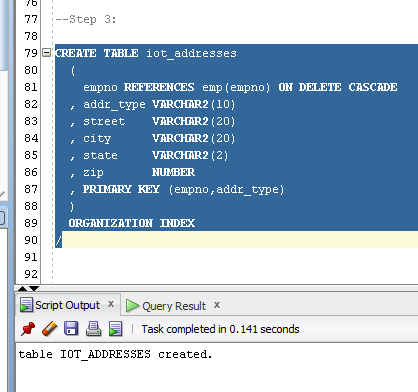




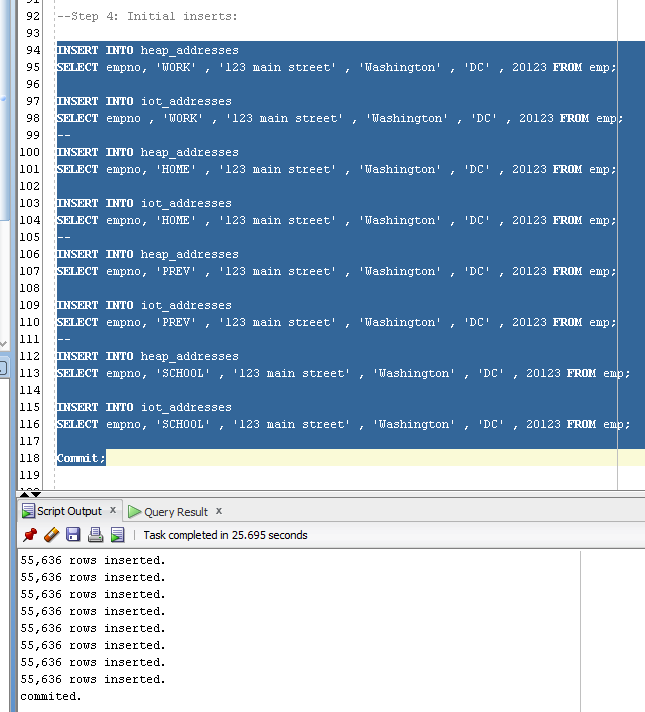
Step2



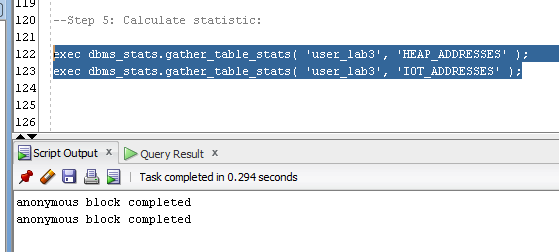
Step3

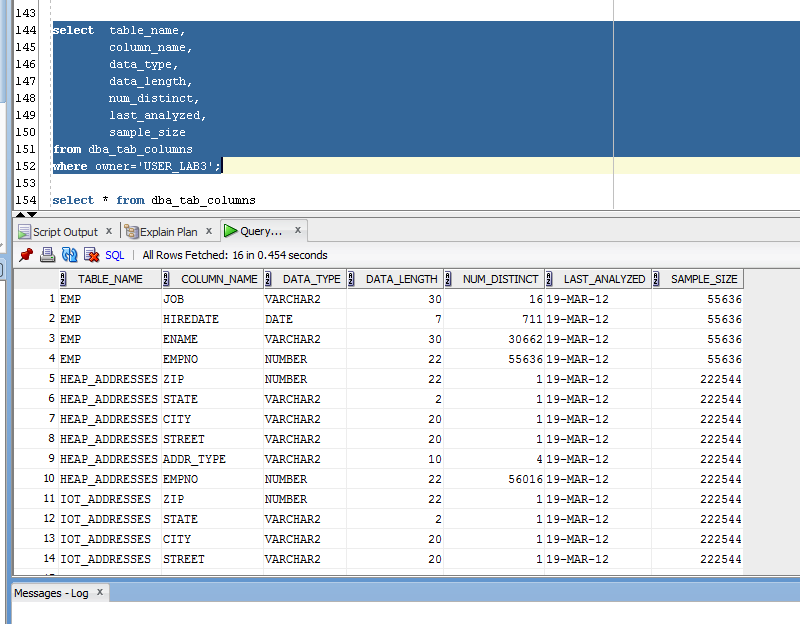


Step 4: Inserting rows;



Step 5: Calculate statistic;

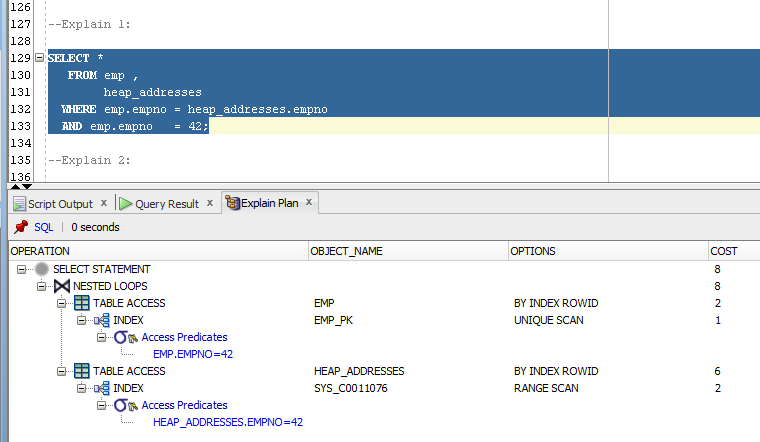


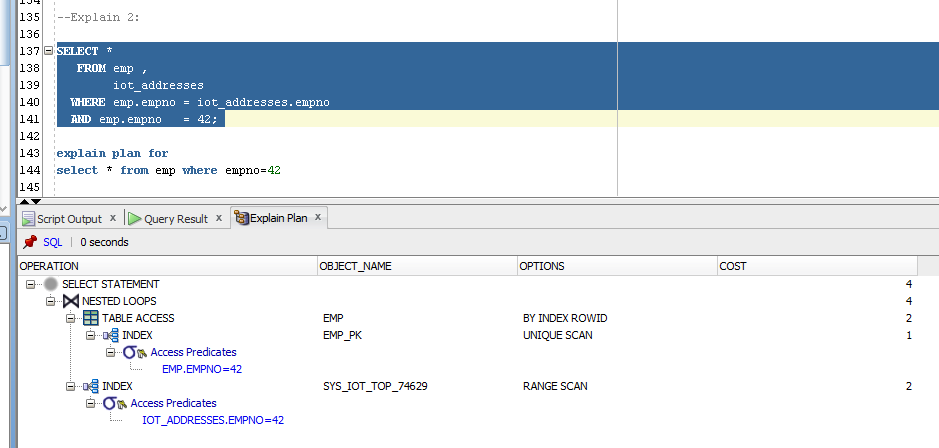


**Step 6:**

Expected Heap table cost > IOT table cost

Prepare screenshots and write explanation why cost is different.





Heap table cost > IOT table cost.

This is due to the fact, that when you add data to the Heap table, space for them may be added in any place of the segment, and, when referring to elements of the Heap table, it is necessary to find the desired segment index at which we get necessary row\_id. And only with this row\_id we can obtain the necessary data from the table.

With the IOT table is not the case, because these tables are stored in an index structure. Thus, when referring to items in the table, reading the index, and we immediately obtain the necessary data.

Step 7: clean up scheme;

# Index Clustered Tables

## Task 4: Analyses Cluster Storage by Blocks

Step 1:

CREATE cluster emp\_dept\_cluster( deptno NUMBER( 2 ) )  
 SIZE 1024   
 STORAGE( INITIAL 100K NEXT 50K );

Step 2:

CREATE INDEX idxcl\_emp\_dept on cluster emp\_dept\_cluster;

Step 3:

CREATE TABLE dept

(

deptno NUMBER( 2 ) PRIMARY KEY

, dname VARCHAR2( 14 )

, loc VARCHAR2( 13 )

)

cluster emp\_dept\_cluster ( deptno ) ;

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 )

)

cluster emp\_dept\_cluster ( deptno ) ;

Step 4:

INSERT INTO dept( deptno , dname , loc)

SELECT deptno , dname , loc

FROM scott.dept;

commit;

INSERT INTO emp ( empno, ename, job, mgr, hiredate, sal, comm, deptno )

SELECT rownum, ename, job, mgr, hiredate, sal, comm, deptno

FROM scott.emp

commit;

**Step 5:**

SELECT \*

FROM

(

SELECT dept\_blk, emp\_blk, CASE WHEN dept\_blk <> emp\_blk THEN '\*' END flag, deptno

FROM

(

SELECT dbms\_rowid.rowid\_block\_number( dept.rowid ) dept\_blk, dbms\_rowid.rowid\_block\_number( emp.rowid ) emp\_blk, dept.deptno

FROM emp , dept

WHERE emp.deptno = dept.deptno

)

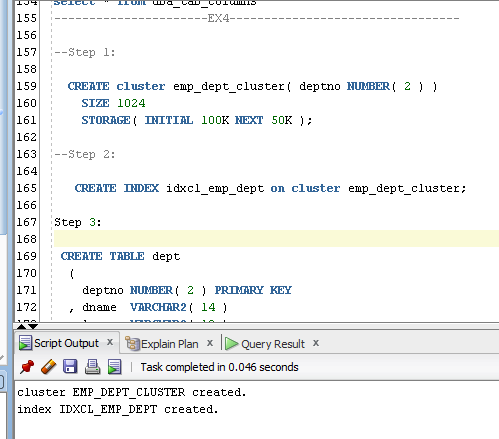
)

ORDER BY deptno

Step 6: Drop all tables;

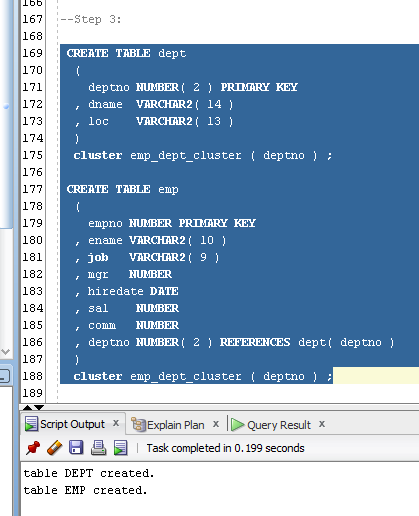
**Task Results:**

Step 1-2: Cluster Created and Index Created;

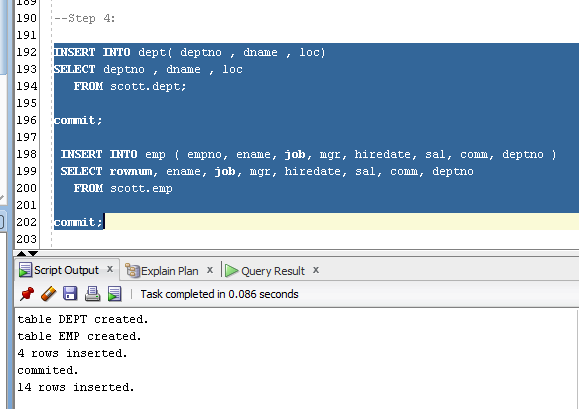


Step 3-4: Table Created and Initialize Inserting rows Finished;

Step3



Step4

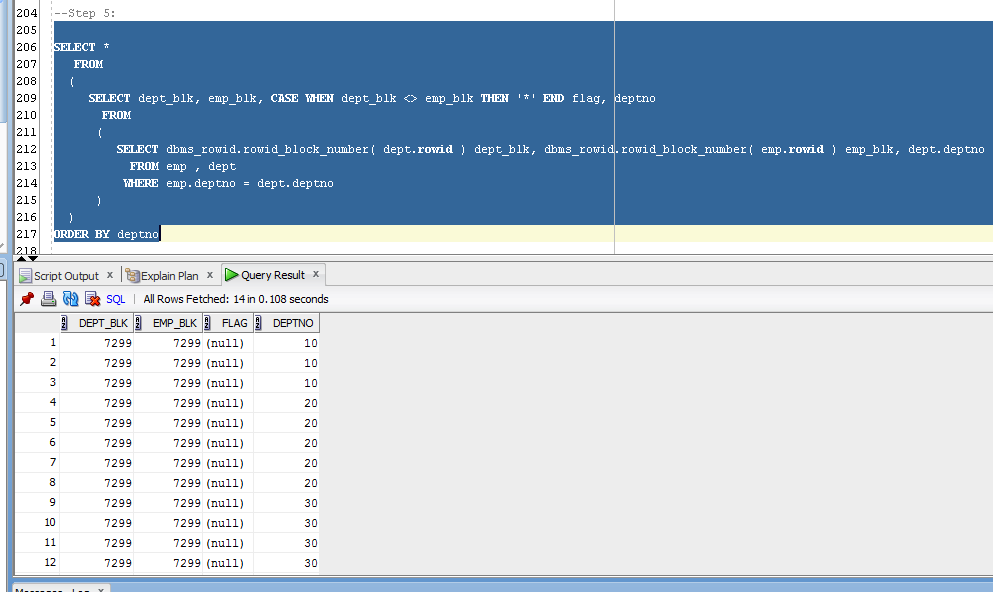


**Step 5:**

Expected All data have to be stored on the same block.

Prepare screenshots and write explanation why data storage look like on select.

Describe advantages of this type of storage.



The main feature is that by using the cluster structure, we achieve that table are physically located on the same database blocks, and records that correspond to the value of the cluster key, located nearby. Thus, we obtain winnings when joining tables, and looking for sample values ​​that are appropriate to different values ​​of the index, since they are allocated close physically. That’s why rows with the same deptno are settled nearby.

Step 7: clean up scheme;

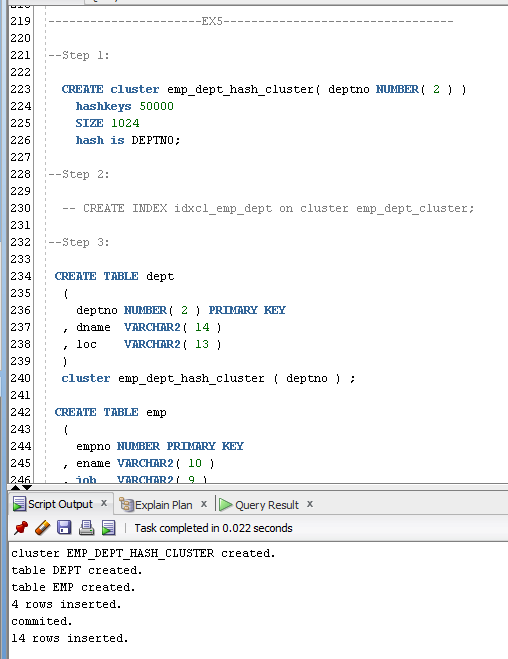
# Hash Clustered Tables

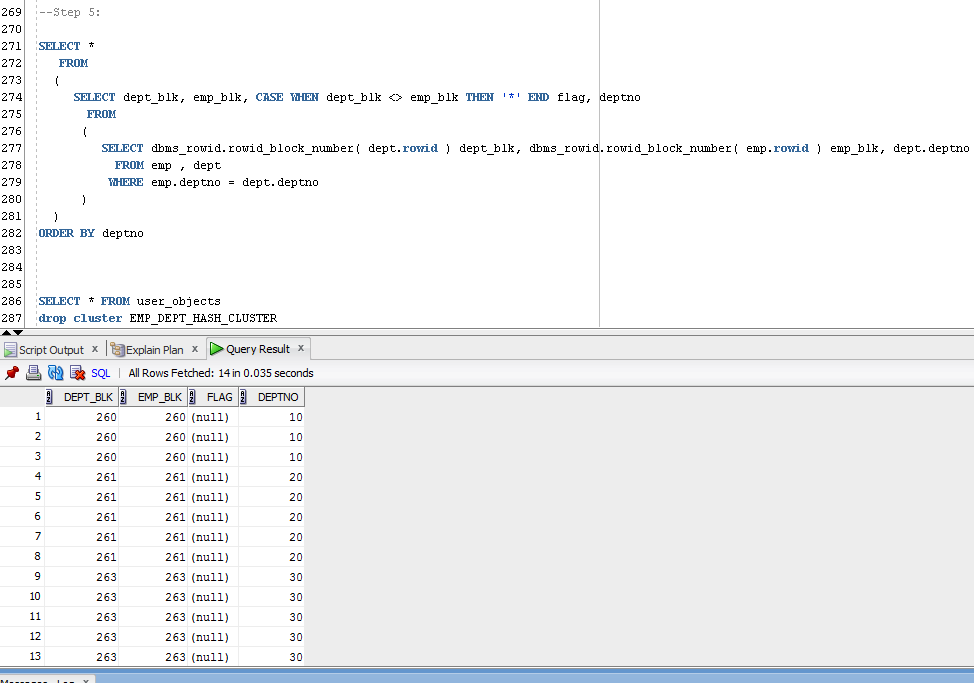
## Task 5: Analyses Cluster Storage by Blocks

Make the same steps as on task 4.1 with one difference create Hash Cluster.

NOTE: To find more information about hash cluster create statements use: Hash Clustered Tables

1. Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming; Techniques and Solutions, Second Edition; Thomas Kyte ; 2010 (Chapter 10: Database Tables - Hash Clustered Tables).





The results differs from the previous task, because we used hashed key to the cluster to get data with the same deptno allocated on the separate database block.

A hash cluster provides an alternative to a nonclustered table with an index or an index cluster. With an indexed table or index cluster, Oracle locates the rows in a table using key values that Oracle stores in a separate index. To use hashing, you create a hash cluster and load tables into it. Oracle physically stores the rows of a table in a hash cluster and retrieves them according to the results of a **hash function**.