Report

On the

U1M3.LW.Database Types of Tables, Indexes

Sadovskaya Alina

1. Prerequisites:

Connect as system user and Create new tablespace  tbs\_lab with new datafile  db\_lab\_001.dat:

create tablespace tbs\_lab datafile '/oracle/u01/app/oracle/product/19.0.0/pdb\_asadovskaya

/dbs/db\_lab\_001.dat’

size 5M autoextend ON next 5M MAXSIZE 100M;

Create new user:

create user Alina identified by 10082001 default tablespace tbs\_lab;

Grant Connect Role and Resource Role:

grant connect to Alina;

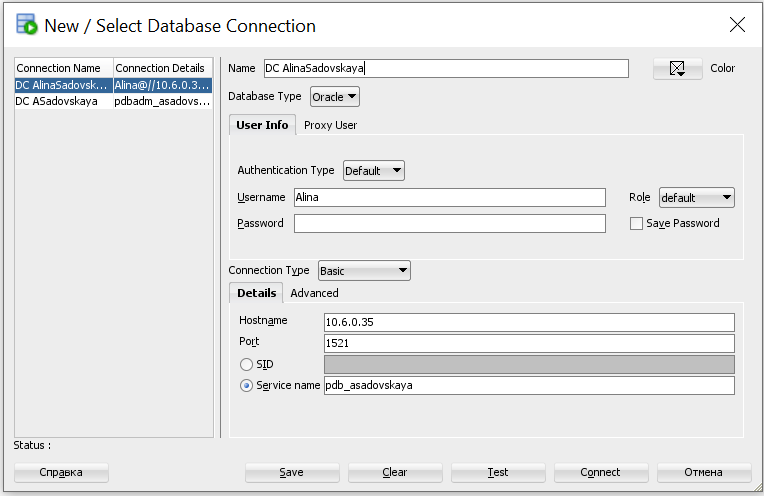
grant resource to Alina;

grant select on scott.dept to Alina;

grant select on scott.emp to Alina;

To run the last two commands to create tables scott schema you need to run a script copied from github.

After that we create a connection using the created user ALINA.



Now we can start completing tasks 1-5 of the laboratory work.

# 2. Heap Organized Tables

## 2.1. Task 1 – Heap Understanding

**Step 1:**

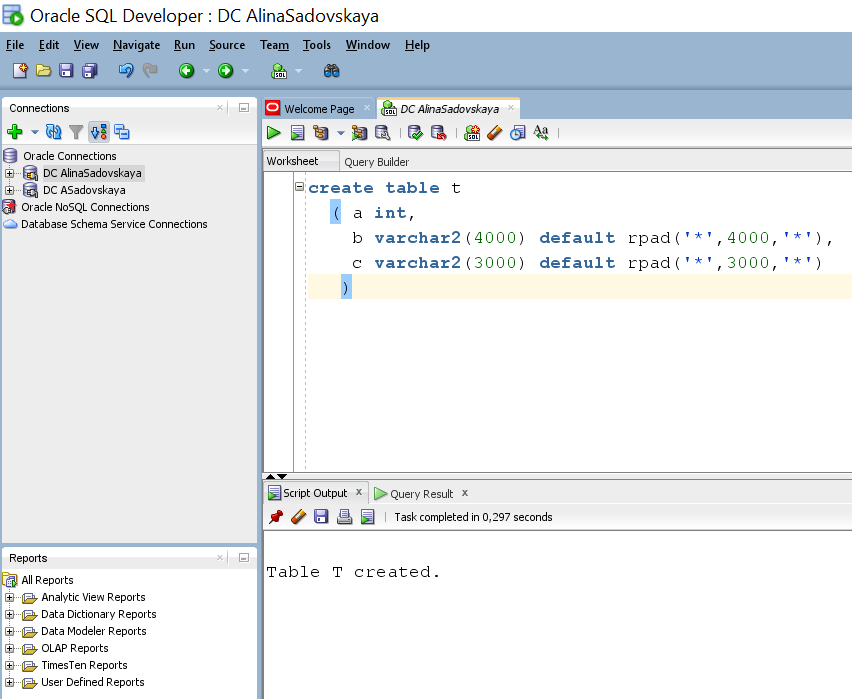
create table t

( a int,

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

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

)



**Step 2:**

**Changing the created table:**

insert into t (a) values ( 1);

insert into t (a) values ( 2);

insert into t (a) values ( 3);

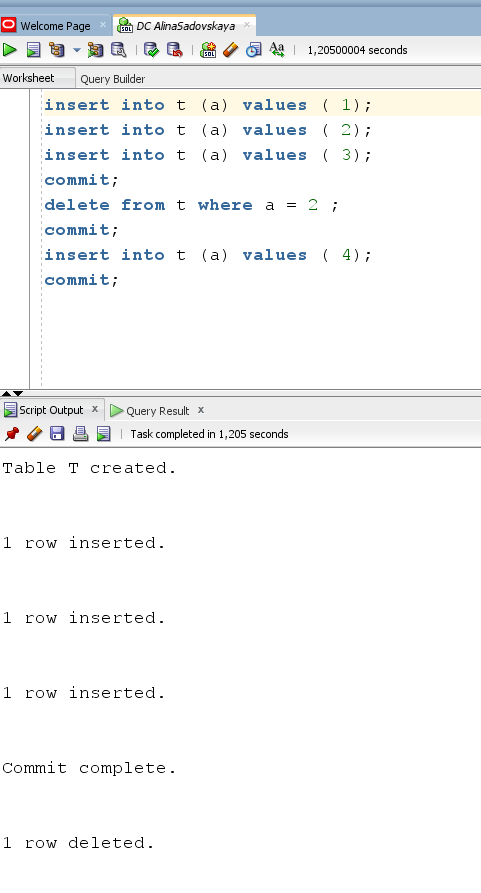
commit;

delete from t where a = 2 ;

commit;

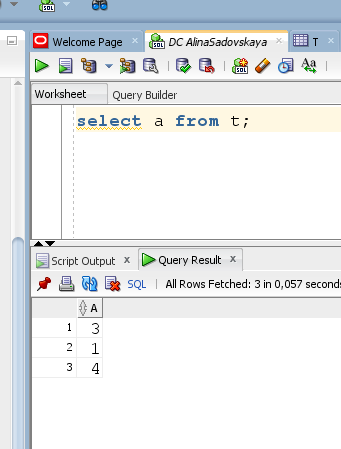
insert into t (a) values ( 4);

commit;

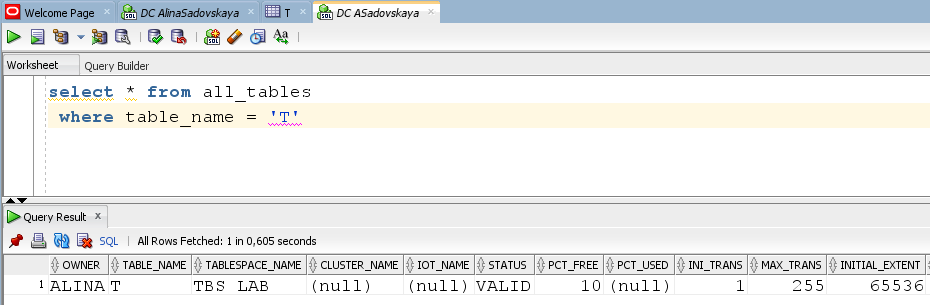


**Step 3:**

select a from t;

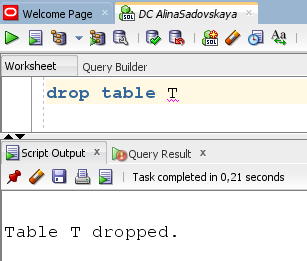


Let's check where our table T is located:



**Step 4:**

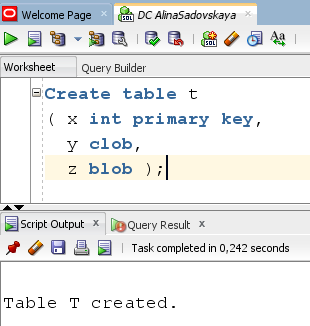
Clean up table T



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

Step 1:

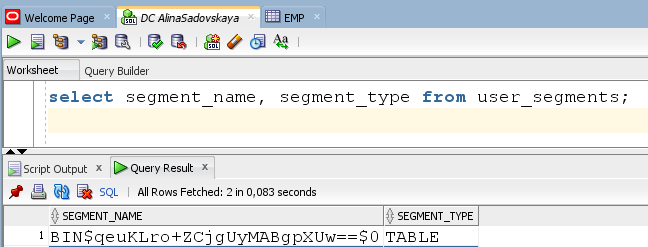
# Create table t ( x int primary key, y clob, z blob );



Step 2:

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

After executing this command, we will find an empty select



Step 3:

# Create table t

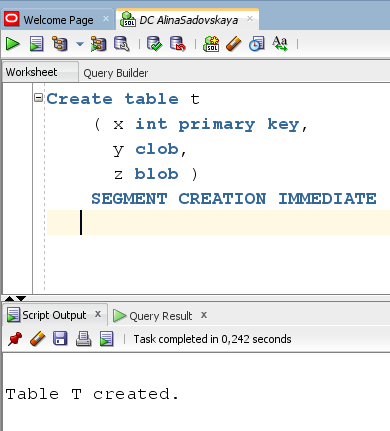
( x int primary key,

y clob,

z blob )

SEGMENT CREATION IMMEDIATE

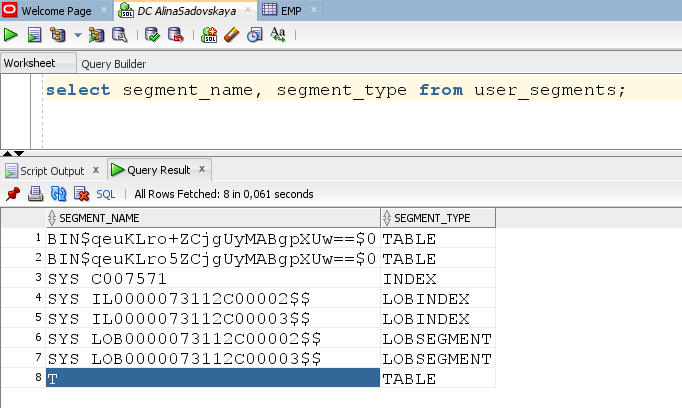
/



This will allocate storage irrespective of deferred\_segment\_creation settings.

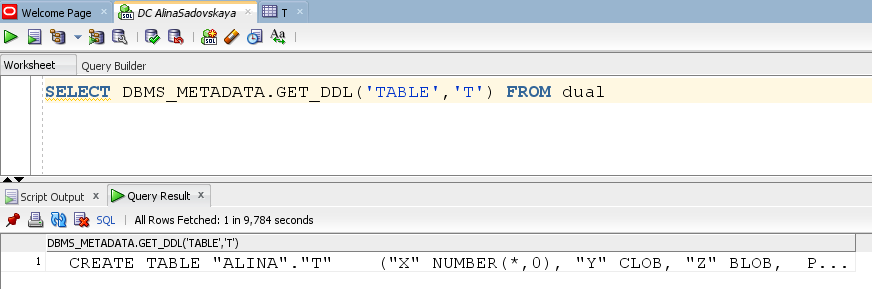
**Step 4:**

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



Step 5:

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



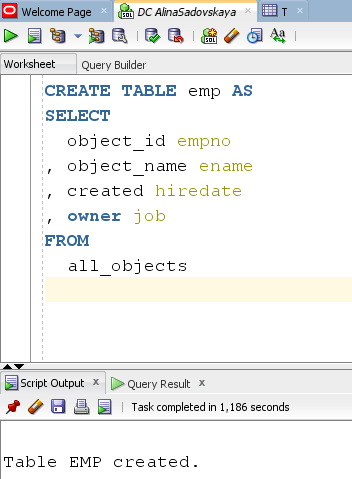
This package provides a way for you to retrieve metadata from the database dictionary as XML.

# 3. Index Organized Tables

# Task 3: Compare performance of using IOT tables

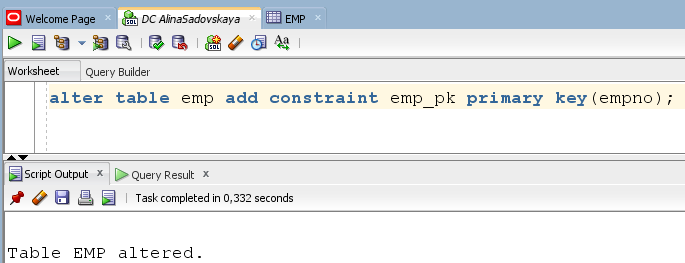
**Step 1:**

**Create table emp**



**Create Index:**

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



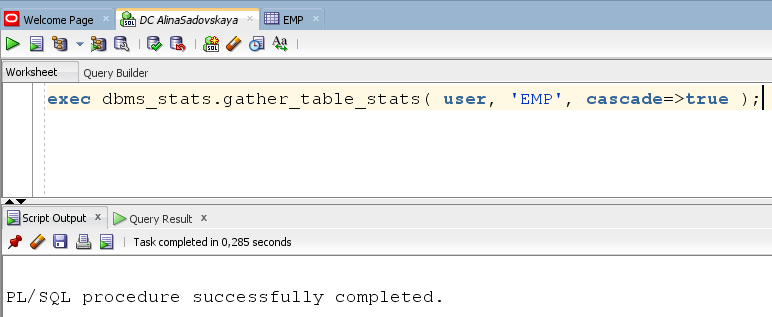
We have created a primary key on the field “empno”.

**Calculate Statistic:**

begin

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

end;



With the DBMS\_STATS package you can view and modify optimizer statistics gathered for database objects.

**Step 2:**

CREATE TABLE heap\_addresses

(

empno REFERENCES emp(empno) ON DELETE CASCADE

, addr\_type VARCHAR2(10)

, street VARCHAR2(20)

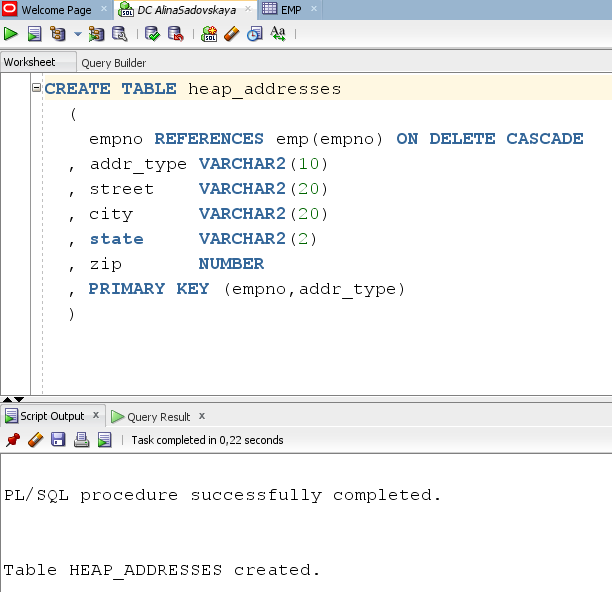
, city VARCHAR2(20)

, 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)

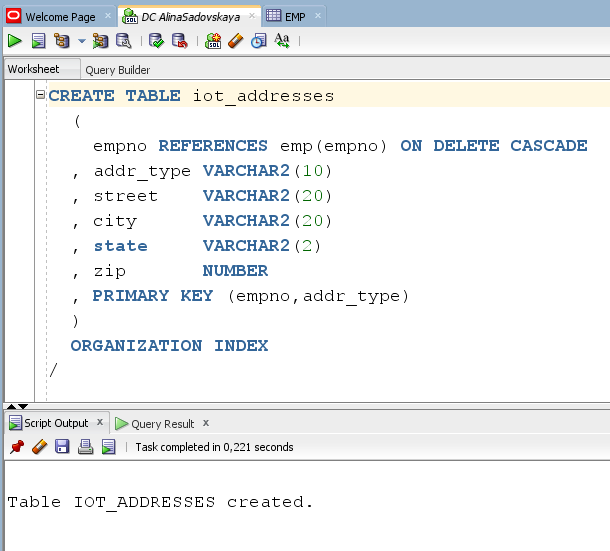
, zip NUMBER

, PRIMARY KEY (empno,addr\_type)

)

ORGANIZATION INDEX

/



**Step 4:** Initial inserts:

Filling in the table with data.

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;

--

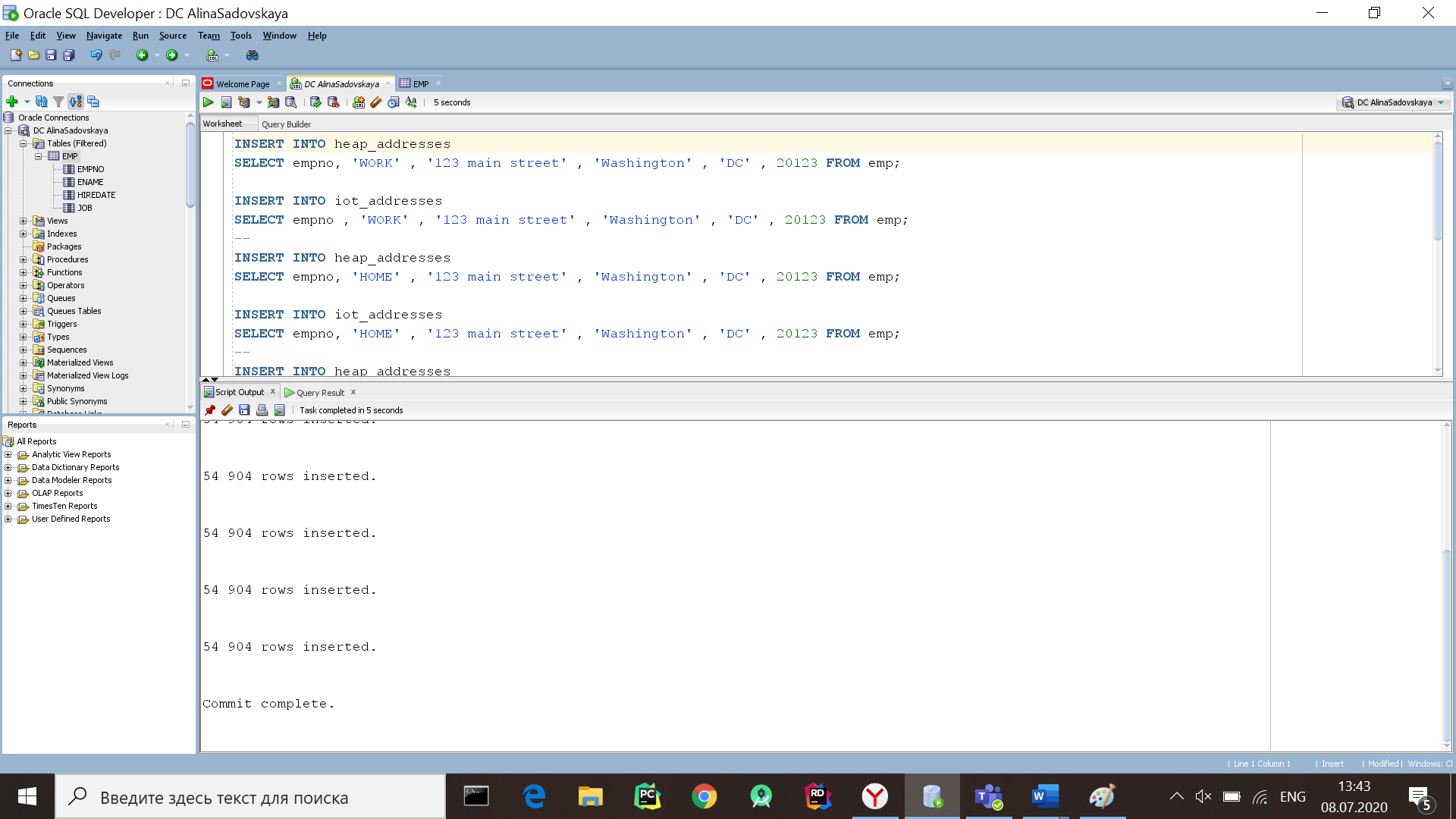
INSERT INTO heap\_addresses

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;

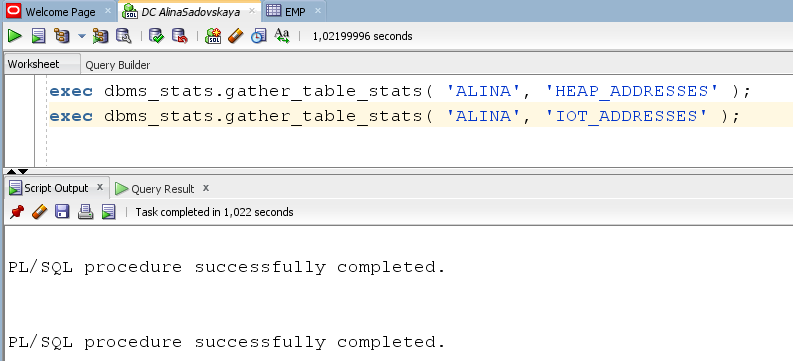
Commit;



**Step 5: Calculate statistic:**

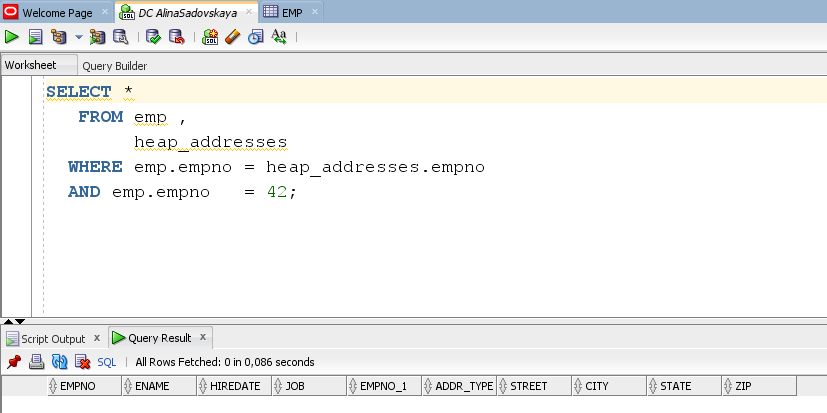
exec dbms\_stats.gather\_table\_stats( $username$, 'HEAP\_ADDRESSES' );

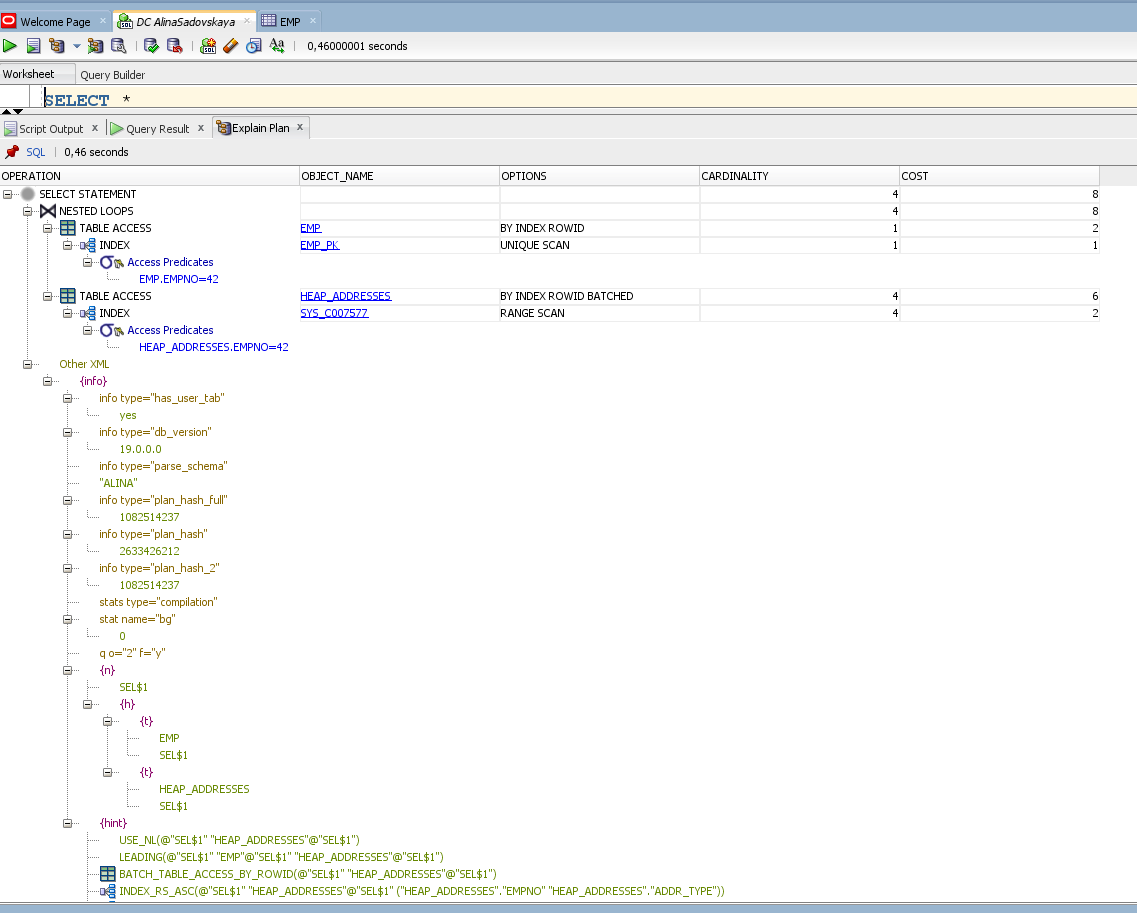
exec dbms\_stats.gather\_table\_stats( $username$, 'IOT\_ADDRESSES' );



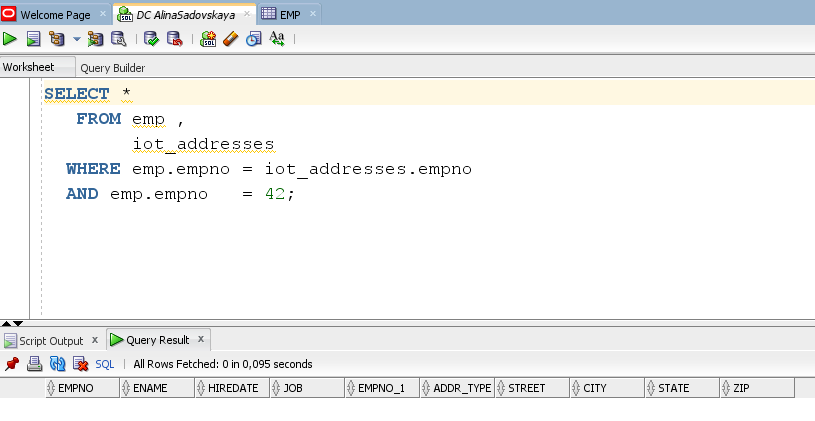
**Step 6:** Compare Trace and Performance:

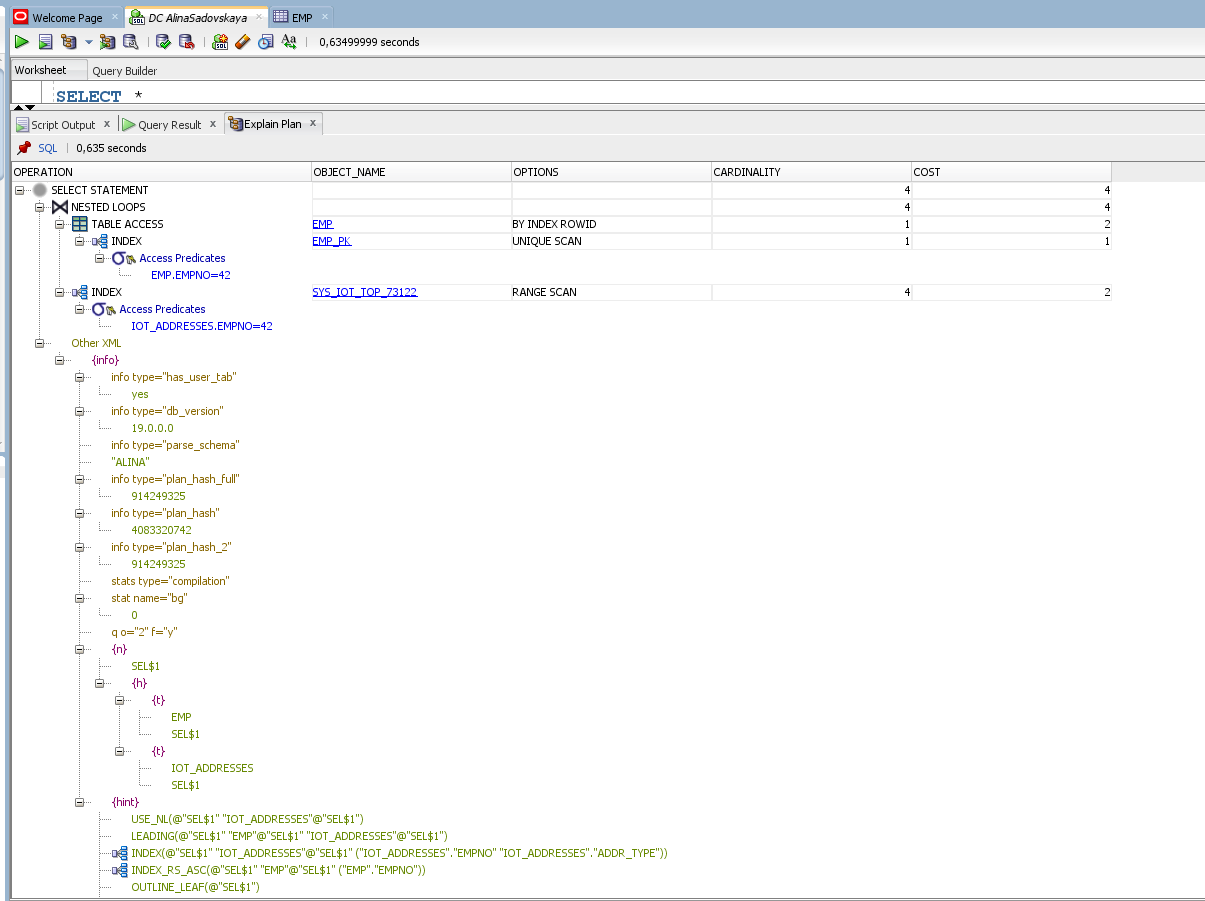
For Heap Table:





For IOT:





After comparing the prices of queries from IOT and Heap table(COST\_IOT < COST\_HEAP), we came to the obvious result that queries from IOT tables are processed faster, because we know this advantages of an IOT:

* As an IOT has the structure of an index and stores all the columns of the row, accesses via primary key conditions are faster as they don't need to access the table to get additional column values.
* As an IOT has the structure of an index and is thus sorted in the order of the primary key, accesses of a range of primary key values are also faster.
* As the index and the table are in the same segment, less storage space is needed.

**Step 7:**

Drop table iot\_addresses;

Drop table heap\_addresses;

Drop table emp;

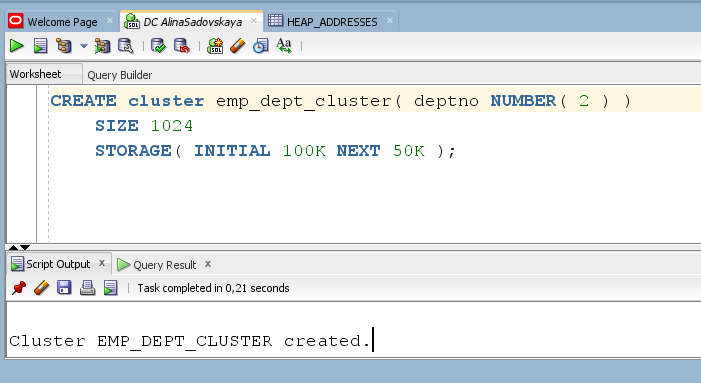
# 4. Index Clustered Tables

## Task 4: Analyses Cluster Storage by Blocks

**Step 1:**

**Create cluster:**

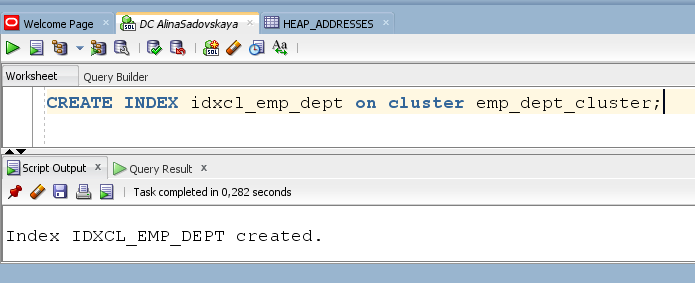
CREATE cluster emp\_dept\_cluster( deptno NUMBER( 2 ) )  
 SIZE 1024   
 STORAGE( INITIAL 100K NEXT 50K );



**Step 2:**

**CREATE INDEX:**

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

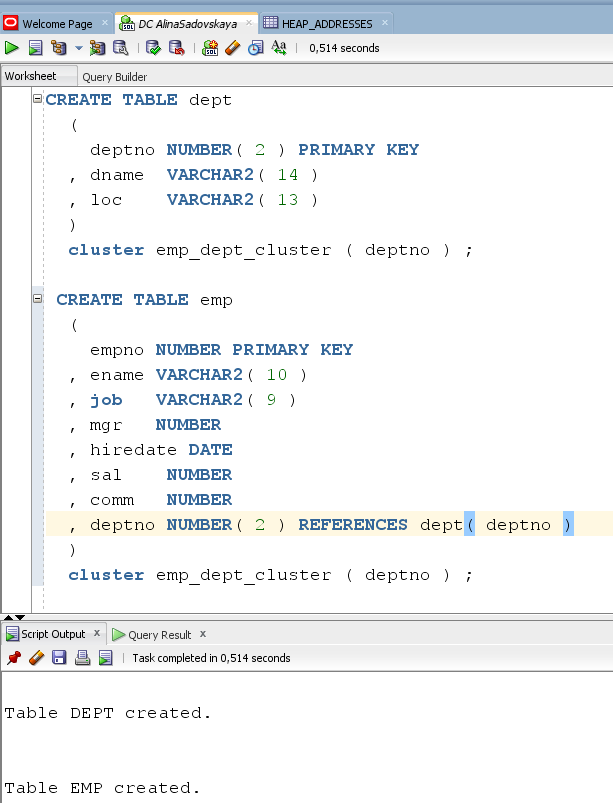
, sal NUMBER

, 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;

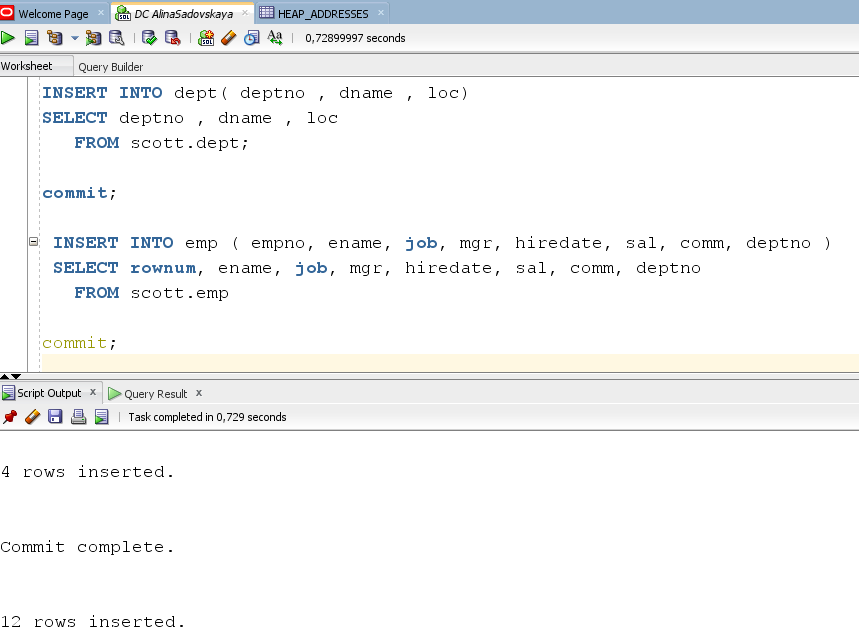
commit;

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

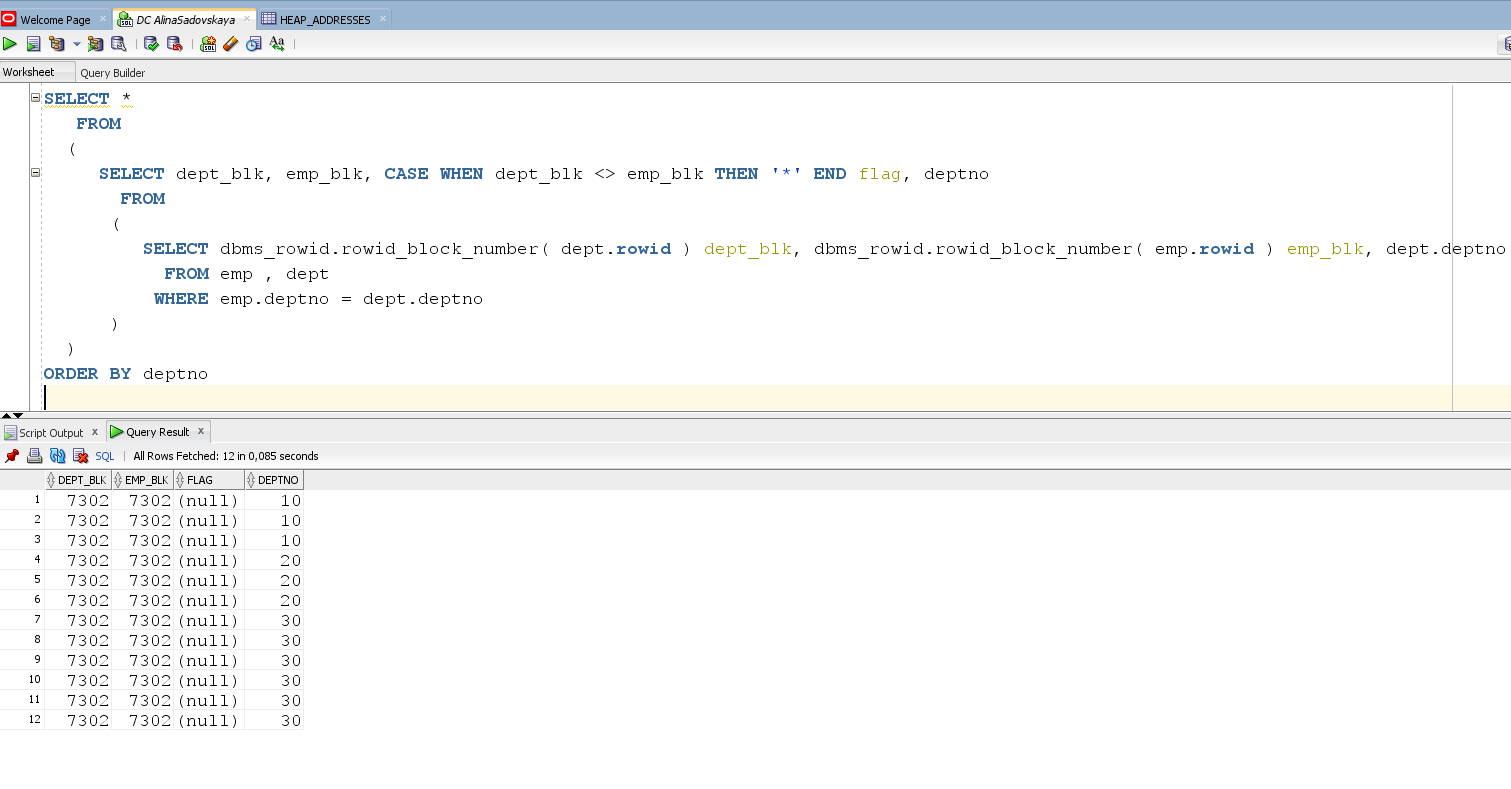
FROM emp , dept

WHERE emp.deptno = dept.deptno

)

)

ORDER BY deptno



As we can see all data have to be stored on the same block.

Each row in the EMR table is stored in a single block with the corresponding DEPT row. But what happens if our estimate is wrong and 1024 is not enough? What if some departments ' data reaches a value close to 1024, while others exceed it? Then, obviously, the data will not fit in one block, and some EMR records will have to be placed in a separate block from the corresponding DEPT record. You can notice that the 12 rows of the EMR table are located in a block that differs from the block that contains the DEPT table rows that correspond to them by the DEPTNO value. Given that the cluster size is underestimated (the SIZE parameter was too small for real data), we could recreate the cluster with a larger SIZE and then get the necessary results.

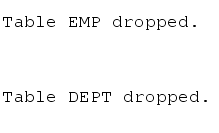
So, a cluster is an appropriate option if there is data that is mostly readable (this does not mean that it is never overwritten; cluster tables are quite capable of modification), read through indexes (either an index on the cluster key, or other indexes that you provide on tables in the cluster), and often join together.

Clusters are used for storing related data from many tables in the same database block. Clusters can accelerate the implementation of intensive in terms of read operations that always join data together or access related sets of data. Clustered index tables reduce the number of blocks that Oracle has to cache. Instead of allocating ten blocks for ten employees from a single Department, Oracle will put them in a single block and thus increase the efficiency of the buffer cache.

As for the disadvantages: if the value for the SIZE parameter is calculated incorrectly, clusters may be inefficient in terms of space utilization and slow down the execution of DML operations.

**Step 6:**

Drop all tables;

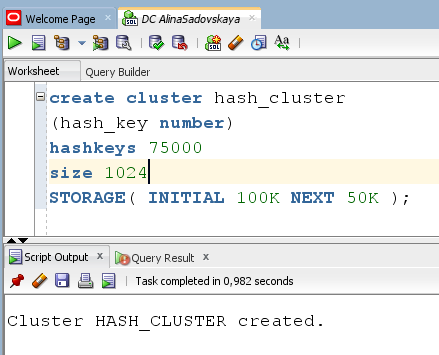


# 5. Hash Clustered Tables

## Task 5: Analyses Cluster Storage by Blocks

**Step 1:**

**Create cluster:**



**Step 2:**

**CREATE tables:**

CREATE TABLE dept\_hash

(

deptno NUMBER( 2 ) PRIMARY KEY

, dname VARCHAR2( 14 )

, loc VARCHAR2( 13 )

)

cluster emp\_dept\_cluster\_HASH ( deptno ) ;

CREATE TABLE emp\_hash

(

empno NUMBER PRIMARY KEY

, ename VARCHAR2( 10 )

, job VARCHAR2( 9 )

, mgr NUMBER

, hiredate DATE

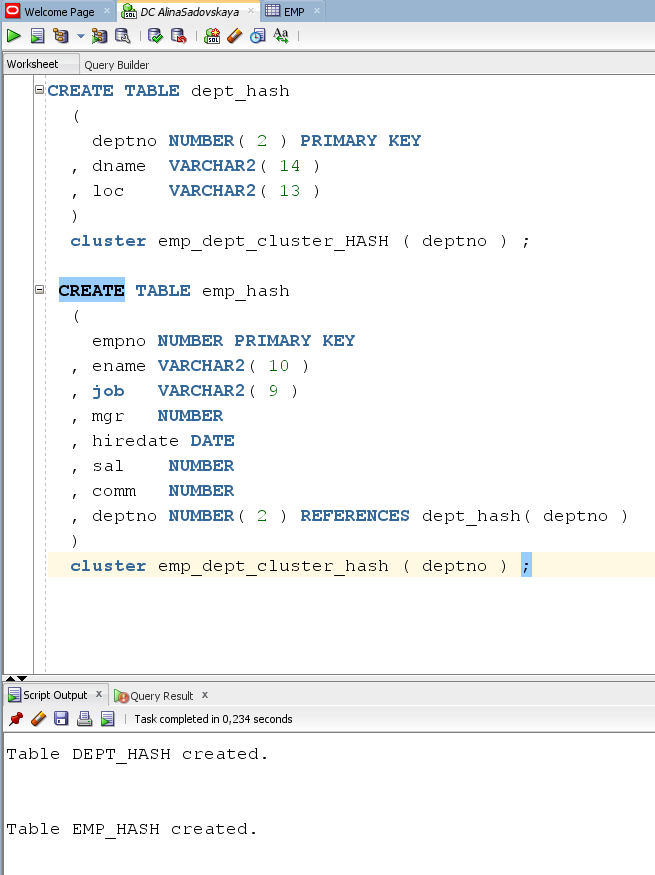
, sal NUMBER

, comm NUMBER

, deptno NUMBER( 2 ) REFERENCES dept\_hash( deptno )

)

cluster emp\_dept\_cluster\_hash ( deptno ) ;



**Step 3:**

INSERT INTO dept\_hash( deptno , dname , loc)

SELECT deptno , dname , loc

FROM scott.dept;

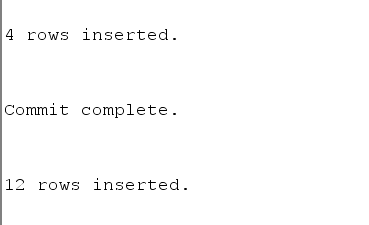
commit;

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

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

FROM scott.emp

commit;



**Step 4:**

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

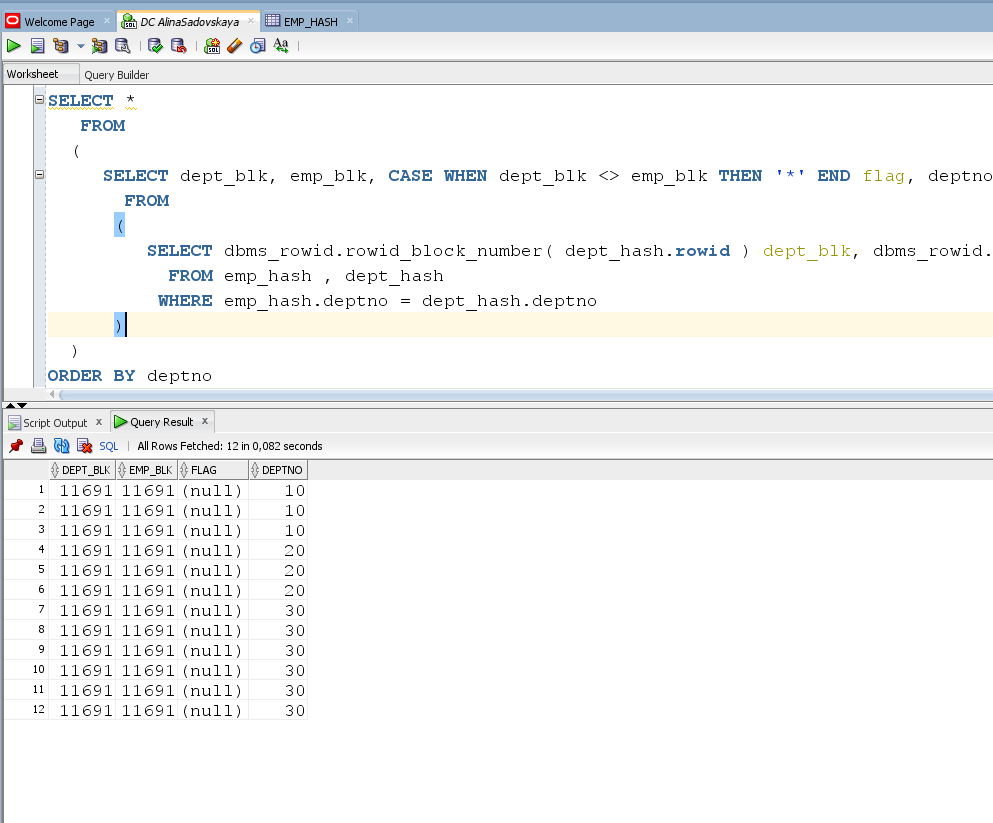
FROM emp\_hash , dept\_hash

WHERE emp\_hash.deptno = dept\_hash.deptno

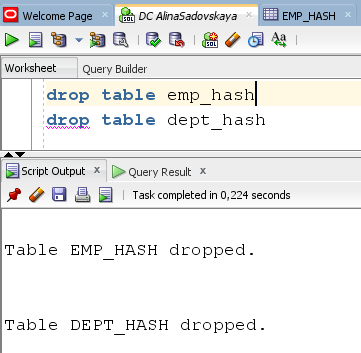
)

)

ORDER BY deptno



**Step 5:**

****

Conceptually, hash clusters are similar to index clusters, except that the cluster index is not used in them. In this case, the index is the data itself. The cluster key is hashed to the address of the block where the data should be located.

Hash clusters are suitable in the following situations.

• When it is fairly certain how many rows the table will have over its lifetime, or at least some reasonable upper limit.

• When DML operations, especially inserts, are easier compared to extraction operations. This means that there is a balance between the amount of data being extracted and the amount of data being added. One user can have 100,000 insertion operations per unit of time, and another user can have 100; it all depends on what scheme they use to extract data. Update operations do not introduce significant overhead unless you are updating to the NAS, which is not a good idea, because it can cause rows to move.

• When you constantly get access to data on the value of the HASHKEY. For example, it can be a table of parts, and these parts can be accessed by the part number. Reference tables are particularly well suited for use in hash clusters.