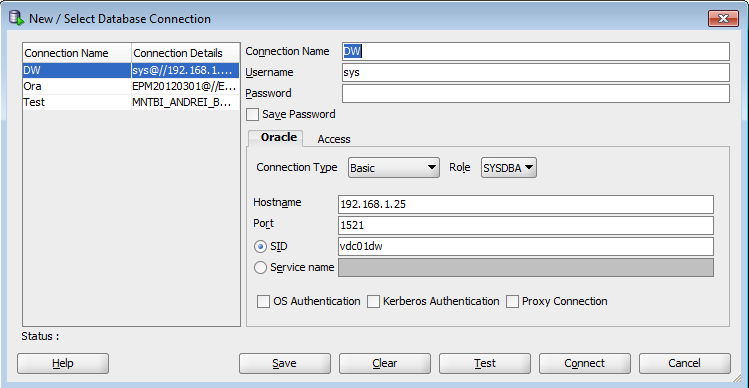
# Prerequisited Task:

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 oracle identified by oracle default tablespace tbs\_lab;

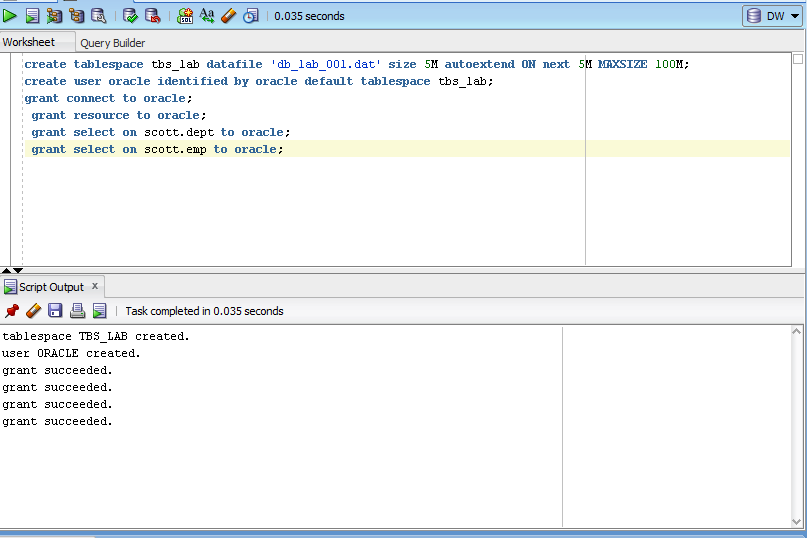
Grant Connect Role and Resource Role:

grant connect to oracle;

grant resource to oracle;

grant select on scott.dept to oracle;

grant select on scott.emp to oracle;



# 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);

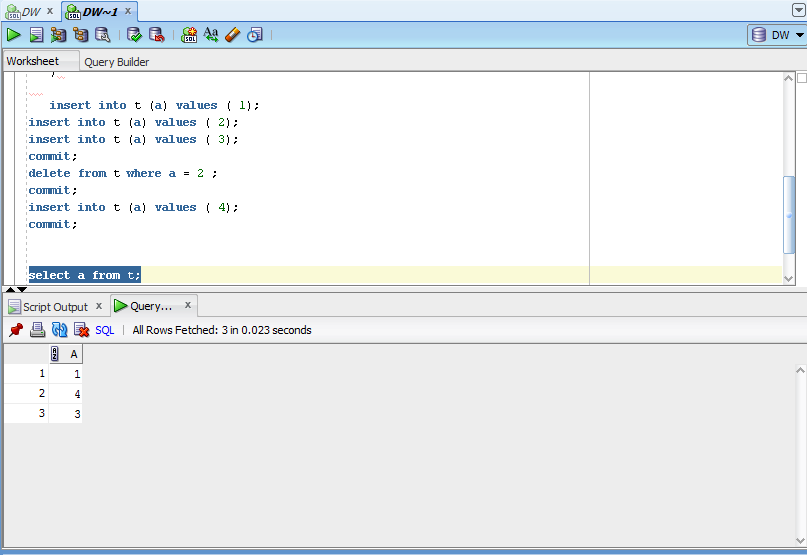
commit;

**Step 3:**

select a from t;

**Task Results:**

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;

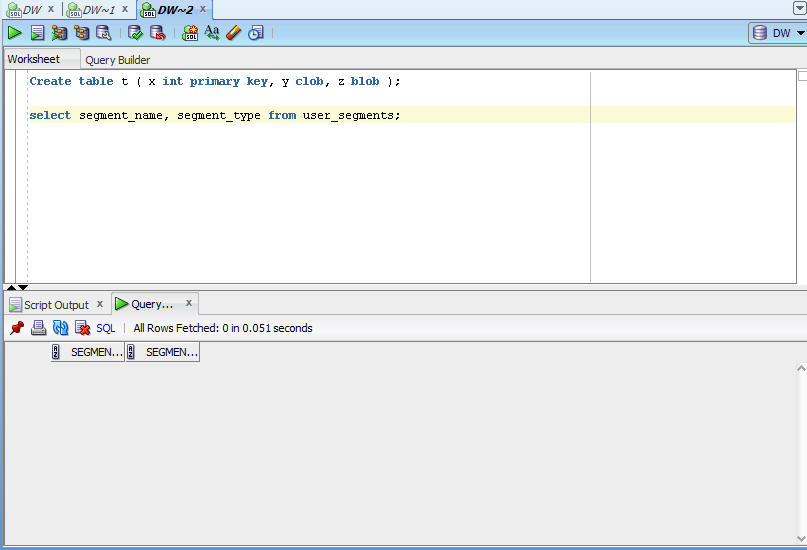
## Task 2 – Understanding Low level of data abstraction: Heap Table Segments

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,

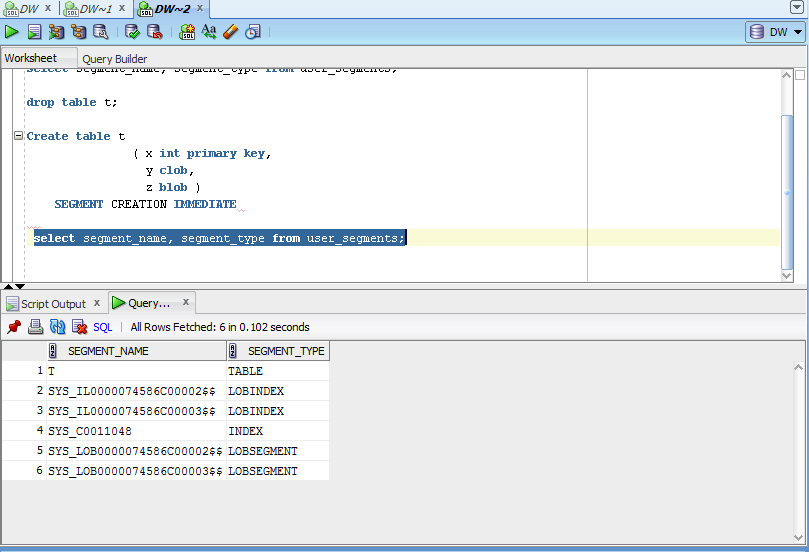
z blob )

SEGMENT CREATION IMMEDIATE

/

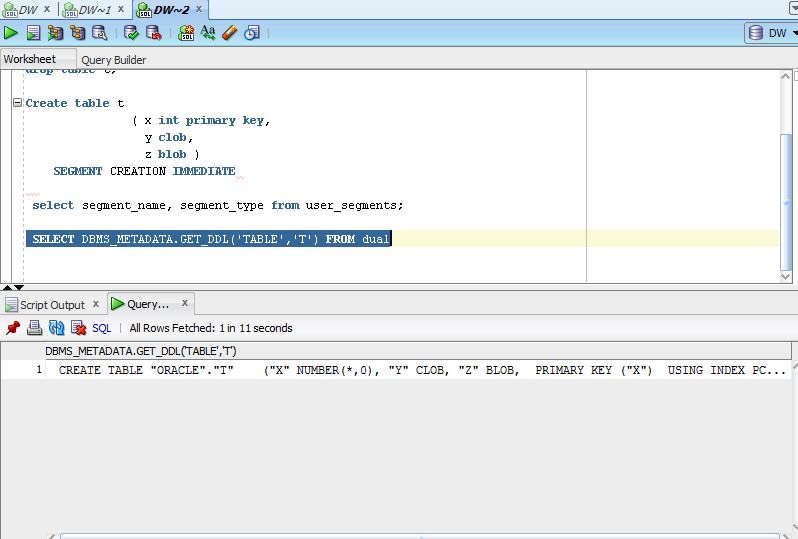
**Step 4:**

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



Step 5:

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



# 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:

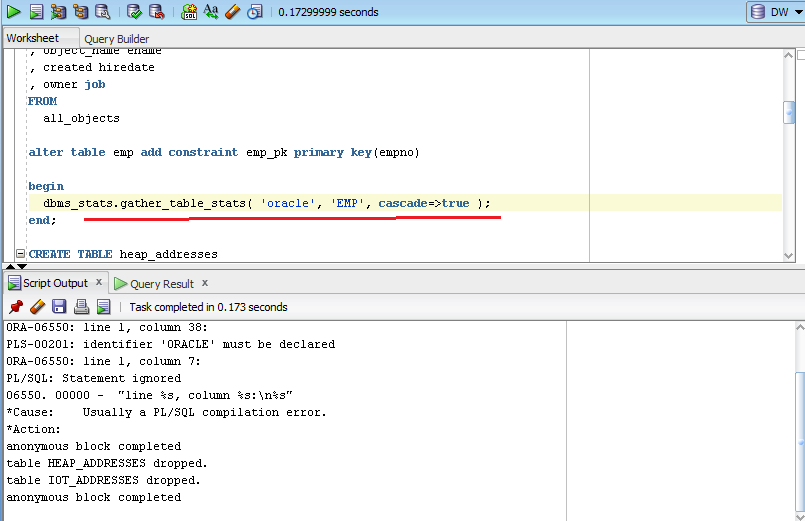
# alter table emp add constraint emp\_pk primary key(empno)

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)

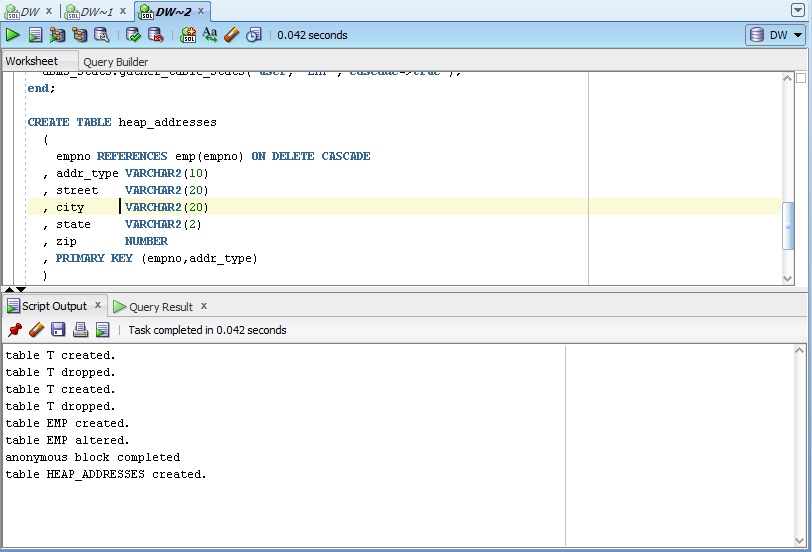
, state VARCHAR2(2)

, 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)

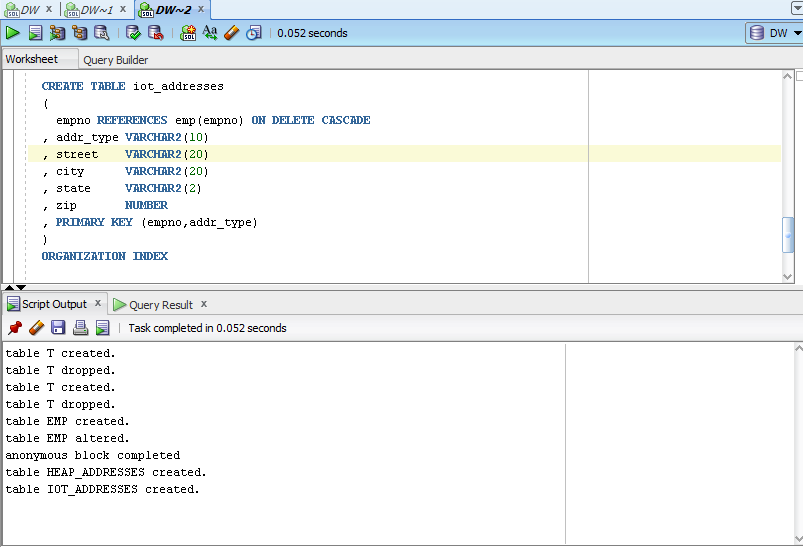
, zip NUMBER

, 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;

--

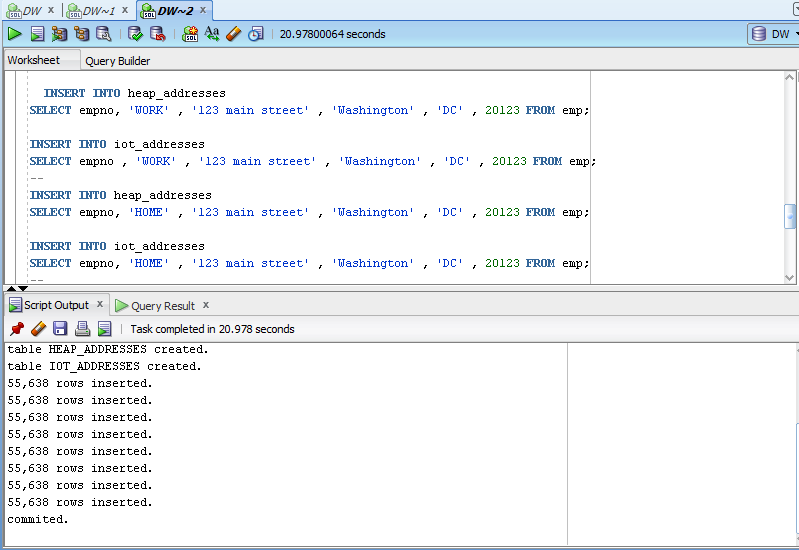
INSERT INTO heap\_addresses

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;

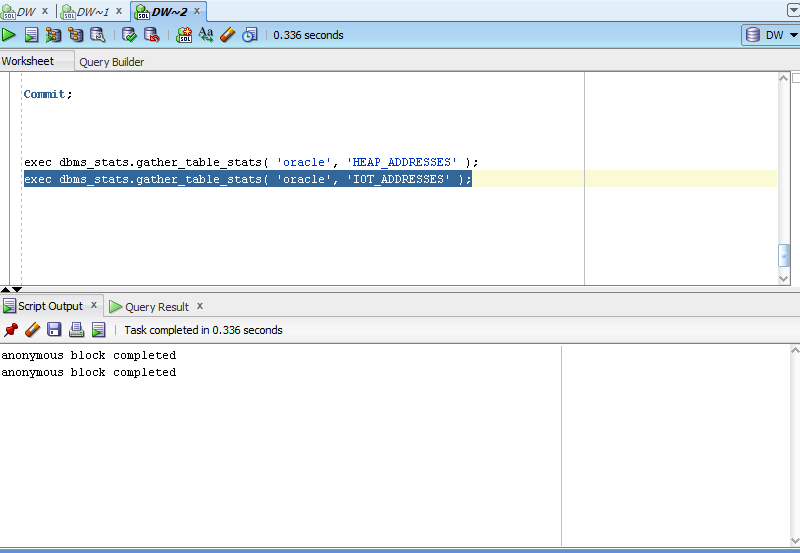
Commit;



Step 5: Calculate statistic:

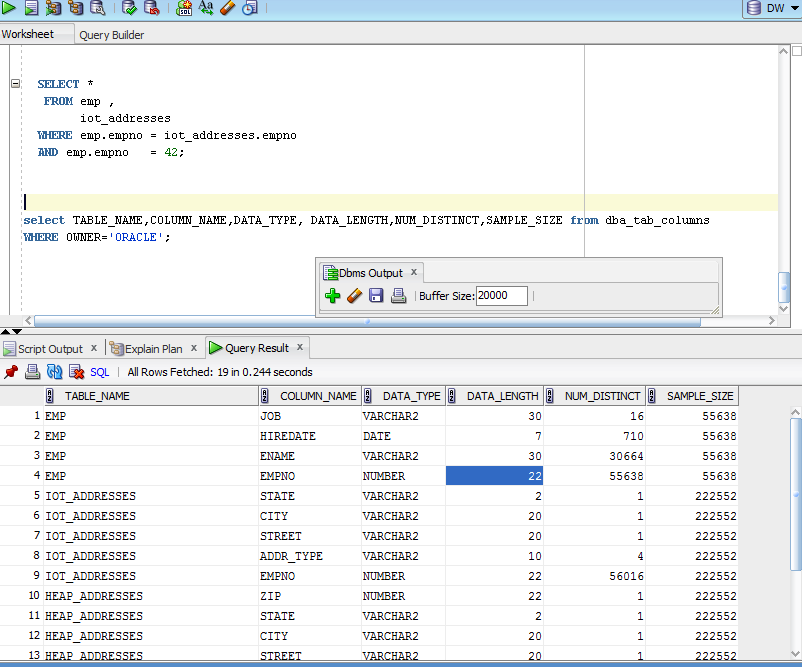
exec dbms\_stats.gather\_table\_stats( $username$, 'HEAP\_ADDRESSES' );

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



select TABLE\_NAME,COLUMN\_NAME,DATA\_TYPE, DATA\_LENGTH,NUM\_DISTINCT,SAMPLE\_SIZE from dba\_tab\_columns

WHERE OWNER='ORACLE';



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

Explain 1:

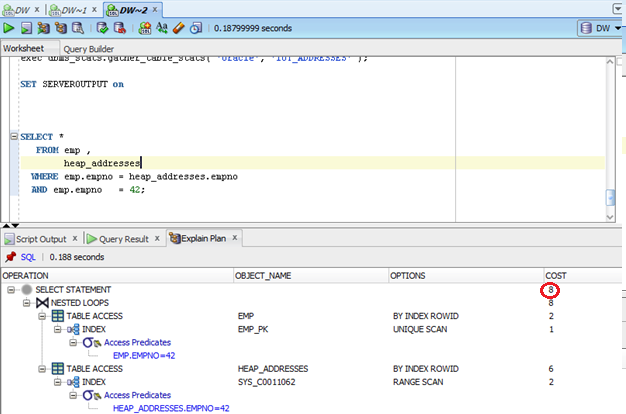
SELECT \*

FROM emp ,

heap\_addresses

WHERE emp.empno = heap\_addresses.empno

AND emp.empno = 42;



Explain 2:

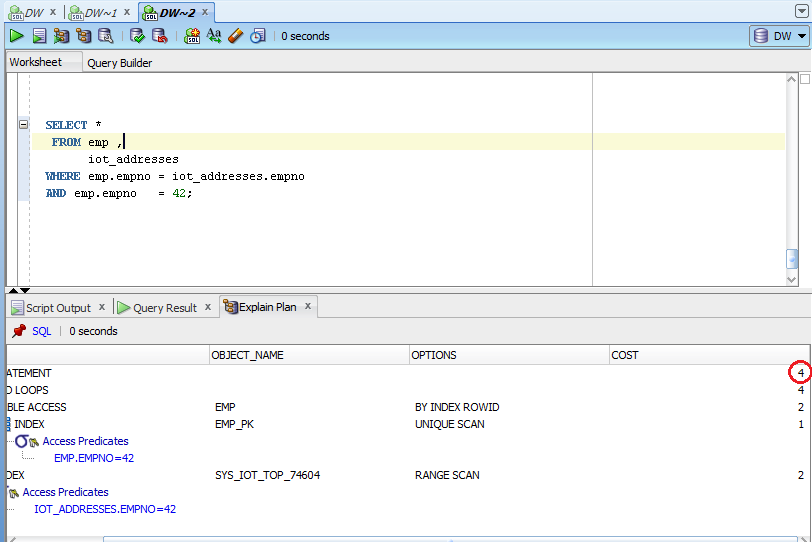
SELECT \*

FROM emp ,

iot\_addresses

WHERE emp.empno = iot\_addresses.empno

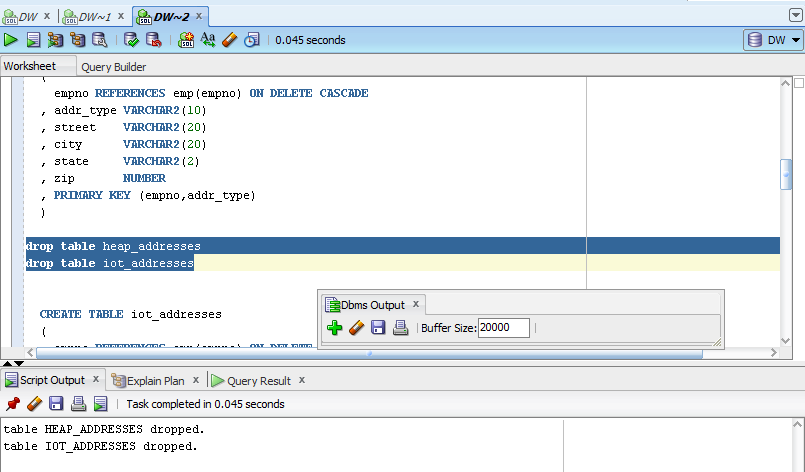
AND emp.empno = 42;



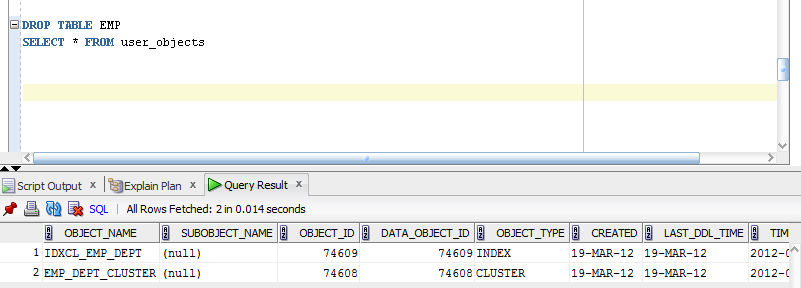
Compare results and explain Cost value calculation and difference on execution plan.

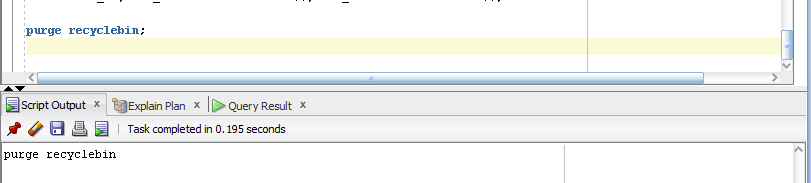
Select from IOT table costs two times fewer, because of it’s organization of storage data. We don’t need to reed first index and when table, we reed all at once from index organized table.

Step 7: Drop all tables;



Step 7: clean up scheme;





# Index Clustered Tables

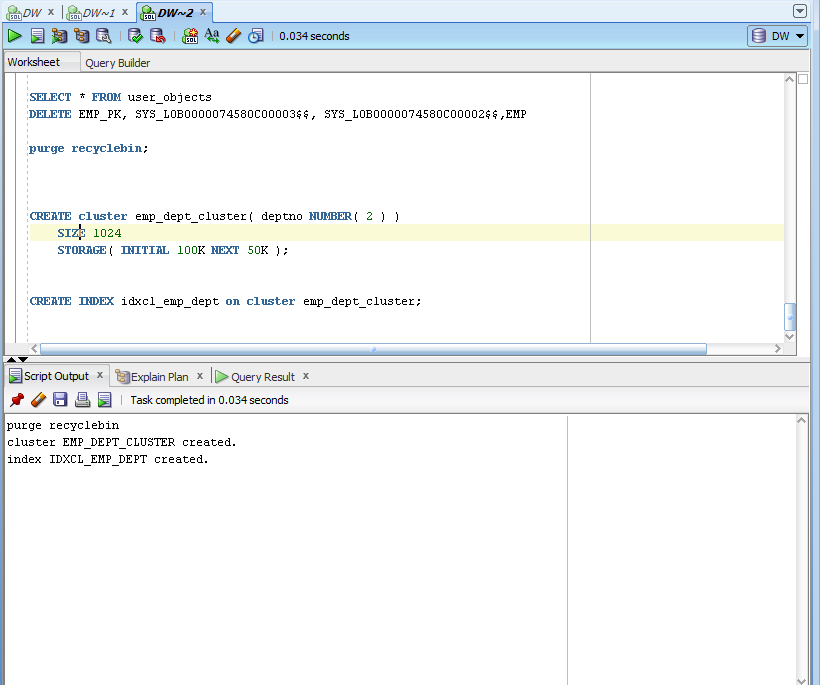
## Task 4: Analyses Cluster Storage by Blocks

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

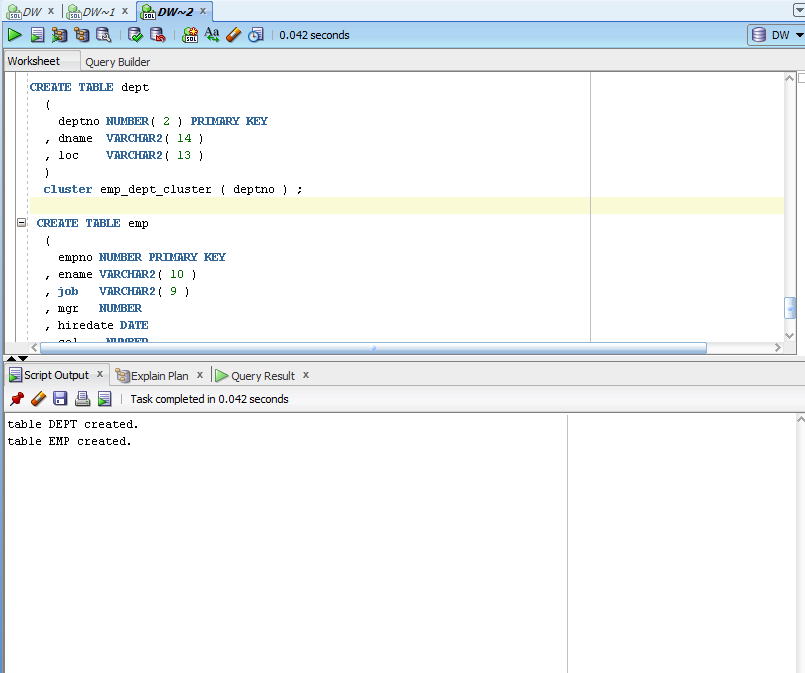
, sal NUMBER

, 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;

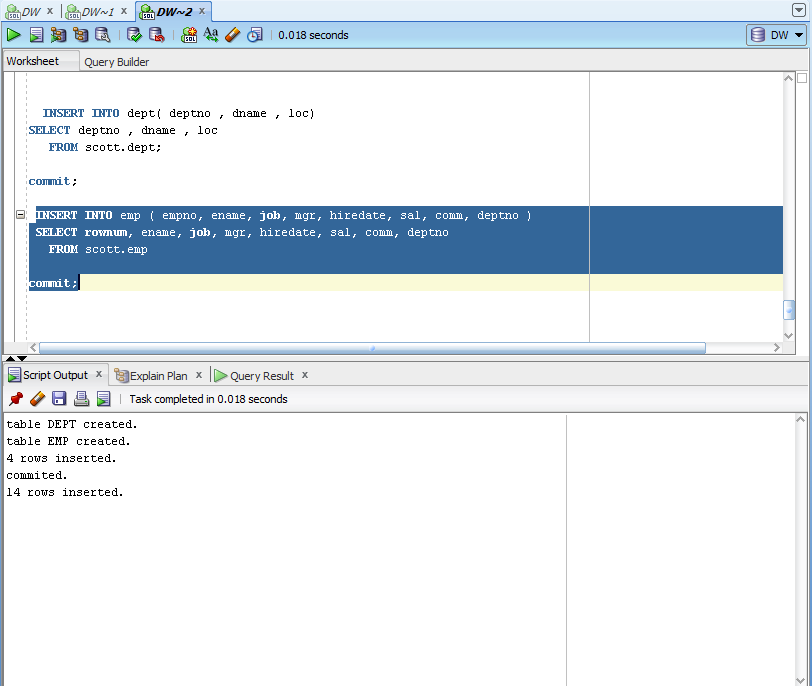
commit;

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

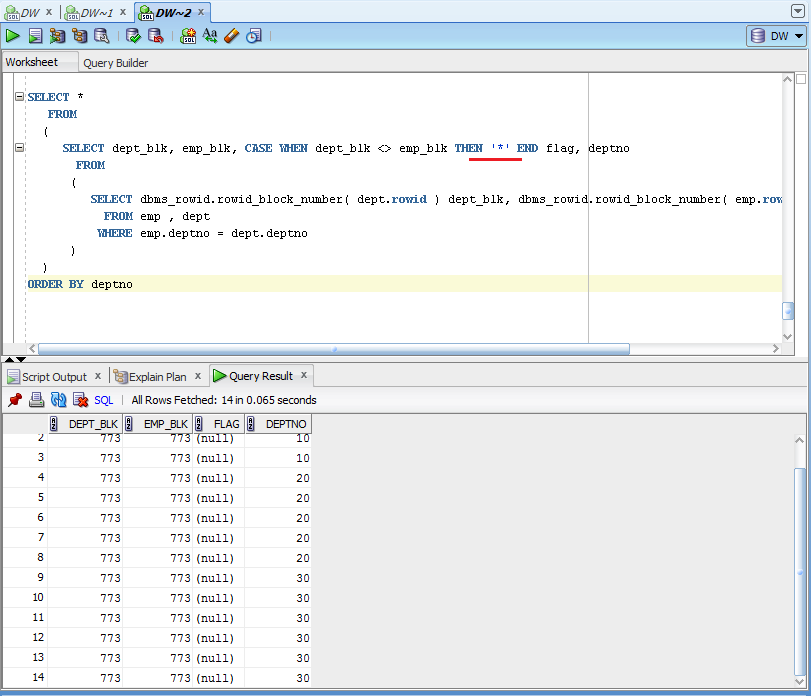
FROM emp , dept

WHERE emp.deptno = dept.deptno

)

)

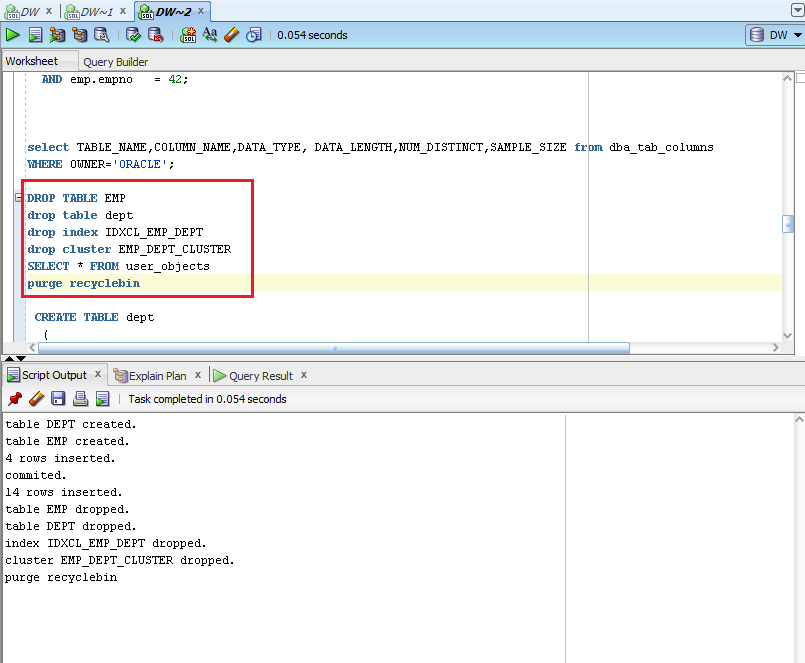
ORDER BY deptno



As we see there are no rows in flag columns with \*, all dept\_blk=emp\_blk, so all information for each id is stored in one block. The idea is to store indexes one near another. In our case our 1 index is stored in one block. According to this the speed of reading from hash table will be fast.

Step 6: Drop all tables;

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).

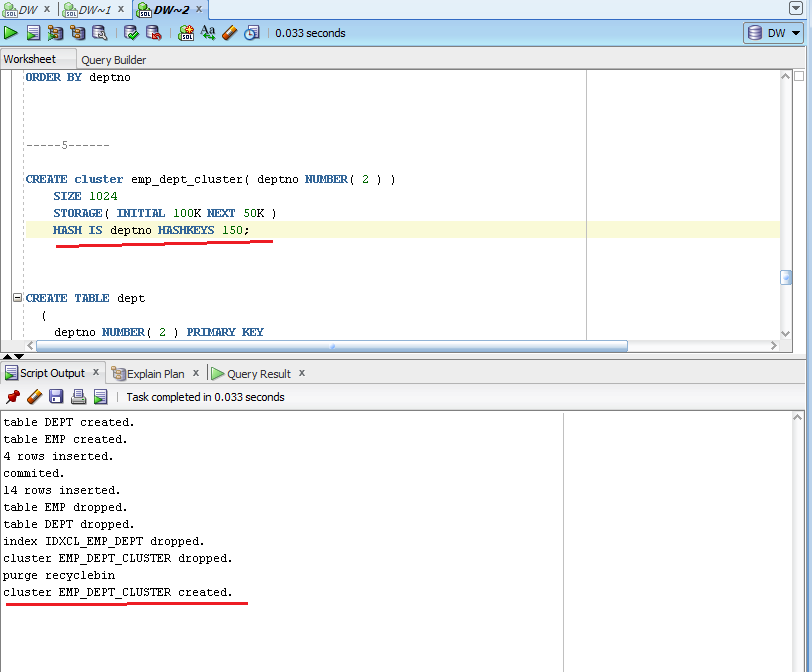
Creation of the cluster

CREATE cluster emp\_dept\_cluster( deptno NUMBER( 2 ) )

SIZE 1024

STORAGE( INITIAL 100K NEXT 50K )

HASH IS deptno HASHKEYS 150;



Creation of tables

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

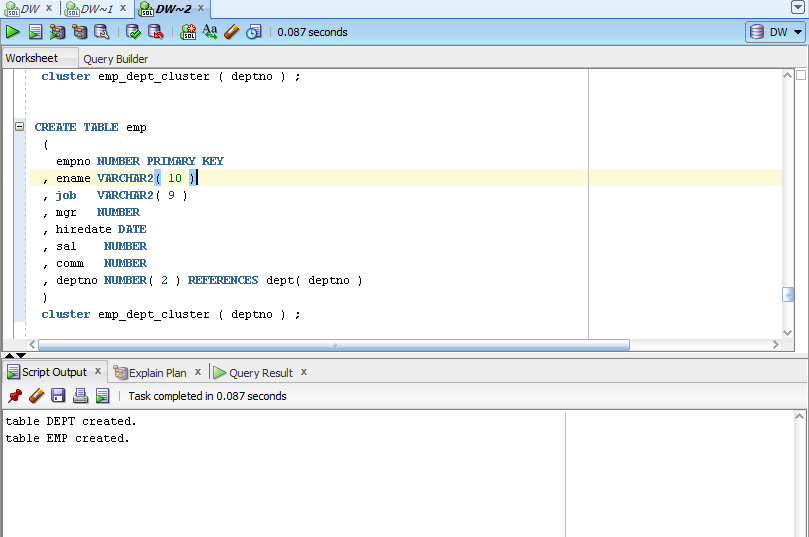
, sal NUMBER

, comm NUMBER

, deptno NUMBER( 2 ) REFERENCES dept( deptno )

)

cluster emp\_dept\_cluster ( deptno ) ;



Inserting of data

INSERT INTO dept( deptno , dname , loc)

SELECT deptno , dname , loc

FROM scott.dept;

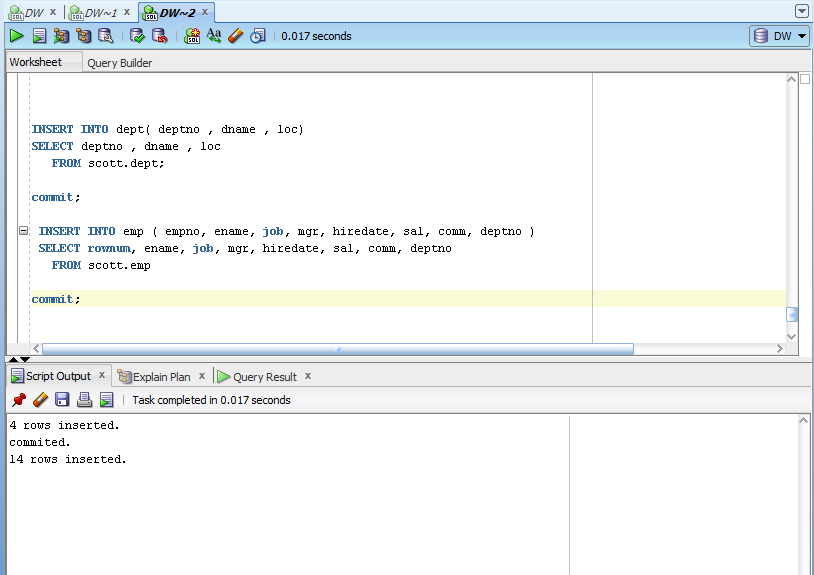
commit;

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

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

FROM scott.emp

commit;



Select statement

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

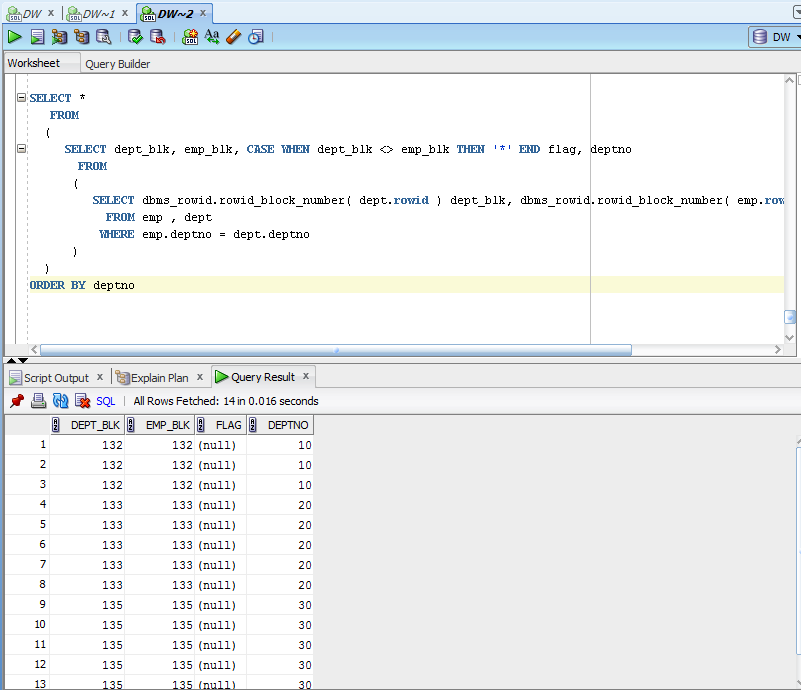
FROM emp , dept

WHERE emp.deptno = dept.deptno

)

)

ORDER BY deptno



In this case we have hash cluster. All information with one row\_id is stored together in one block. As we see deptno=10 is stored in block 132. This help to get the information for specified rowid faster.