**Alexander Korytko**

Task 3

# 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 alexander IDENTIFIED BY 55555 DEFAULT TABLESPACE tbs\_lab;

Grant Connect Role and Resource Role:

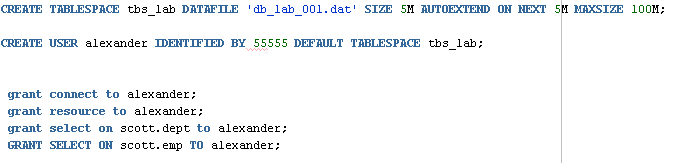
grant connect to alexander;

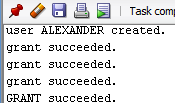
grant resource to alexander;

grant select on scott.dept to alexander;

GRANT SELECT ON scott.emp TO alexander;

**Results:**





# 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

----------

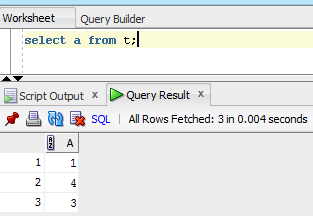
1

4

3

Screenshot of data results below:

**Results:**



**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

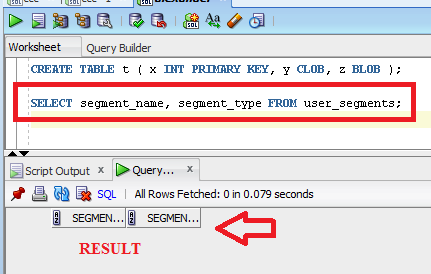
Step 1:

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

Step 2:

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

**RESULTS:**



**Step 3:**

# Create table t

( x int primary key,

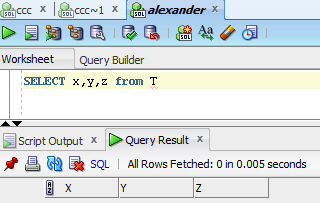
y clob,

z blob )

SEGMENT CREATION IMMEDIATE

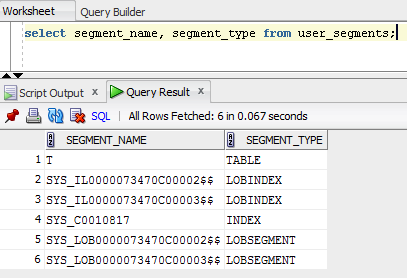
/

**Result:**



**Step 4:**

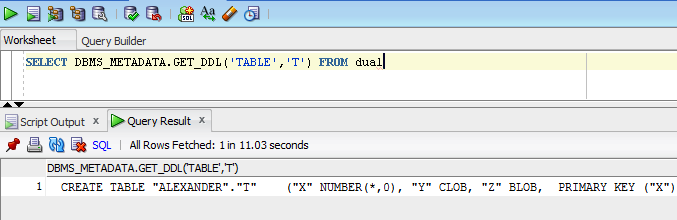
# select segment\_name, segment\_type from user\_segments;



**Step 5:**

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

**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

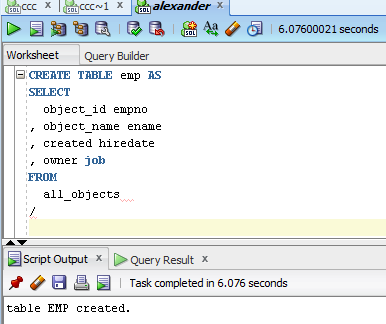
, owner job

FROM

all\_objects

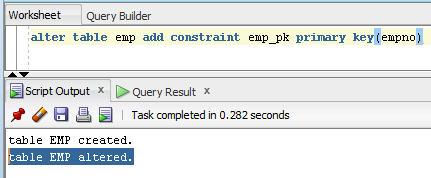
/

**RESULTS:**



Create Index:

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



Calculate Statistic:

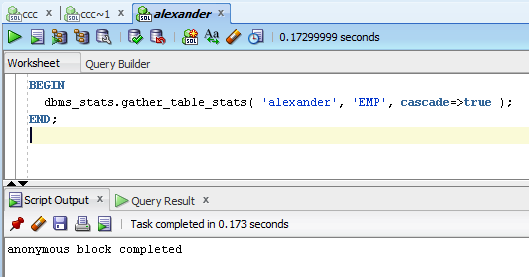
begin

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

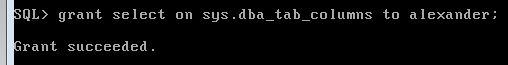
end;

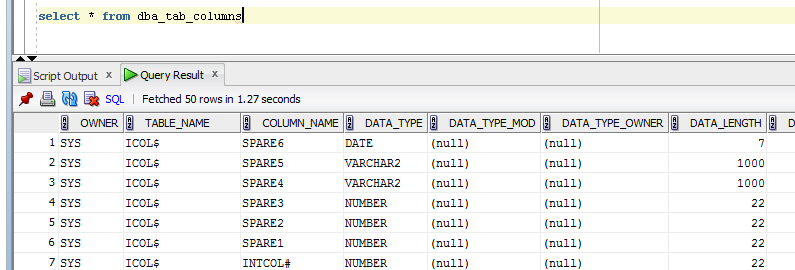
exec DBMS\_STATS.GATHER\_SCHEMA\_STATS(ownname,estimate\_percent, block\_sample , method\_opt,degree,granularity,cascade,stattab, statid,options,statown ,no\_invalidate, gather\_temp,gather\_fixed);

**Result:**

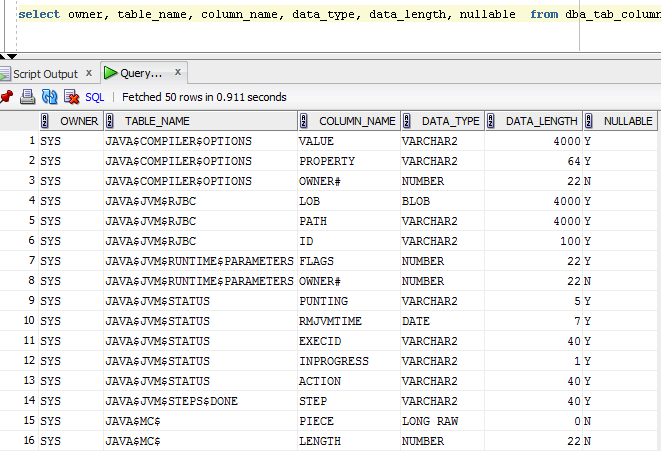


Add grants to our user to select data from statistic Table





**FINAL RESULT:**



**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)

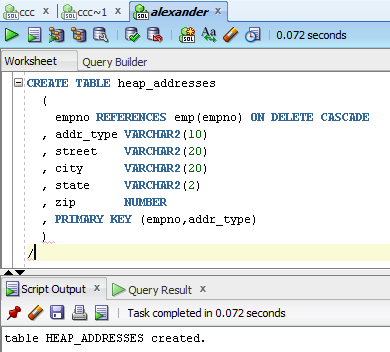
, zip NUMBER

, PRIMARY KEY (empno,addr\_type)

)

/

**RESULTS:**



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

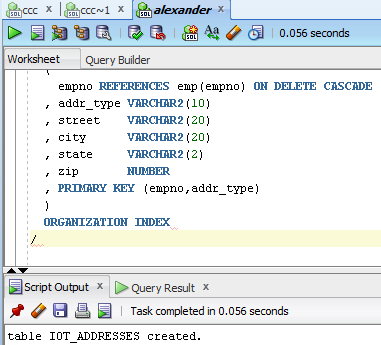
, PRIMARY KEY (empno,addr\_type)

)

ORGANIZATION INDEX

/

**Results:**



Step 4: Initial inserts:

INSERT INTO heap\_addresses

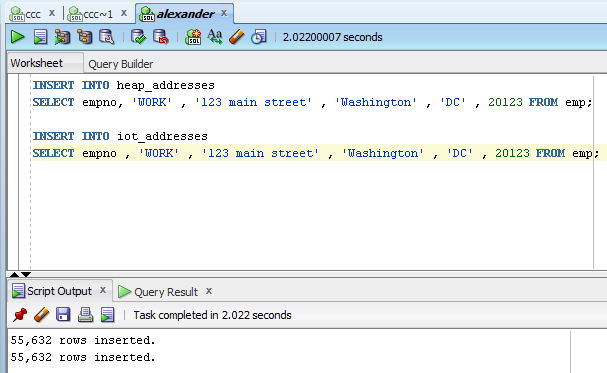
SELECT empno, 'WORK' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

INSERT INTO iot\_addresses

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

--

**Results:**



INSERT INTO heap\_addresses

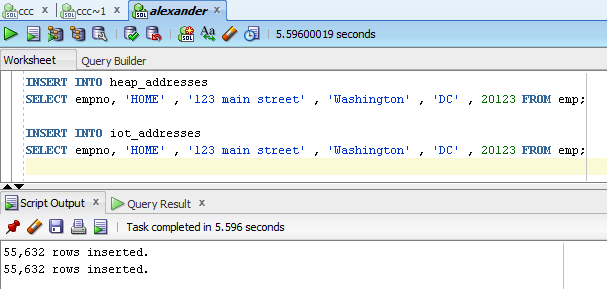
SELECT empno, 'HOME' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

INSERT INTO iot\_addresses

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

--

**Results:**



INSERT INTO heap\_addresses

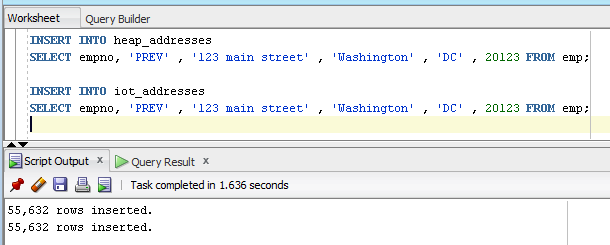
SELECT empno, 'PREV' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

INSERT INTO iot\_addresses

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

--

**RESULTS:**



INSERT INTO heap\_addresses

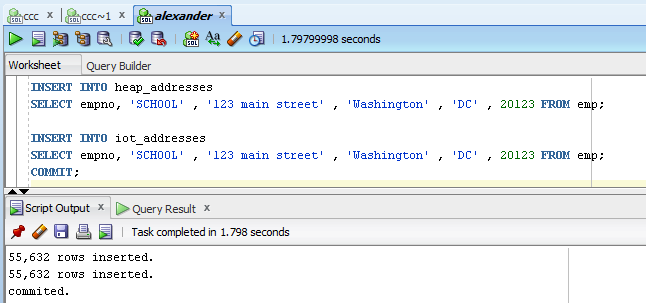
SELECT empno, 'SCHOOL' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

INSERT INTO iot\_addresses

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

Commit;

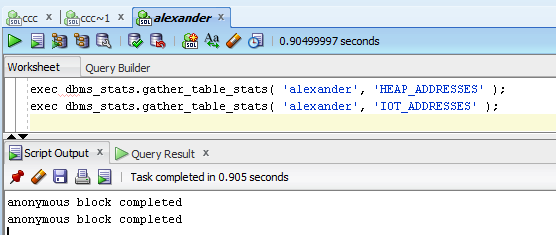
**RESULTS:**



**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:

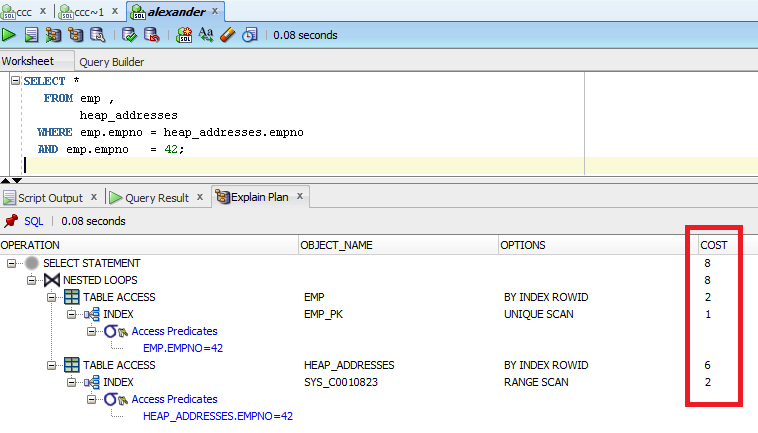
SELECT \*

FROM emp ,

heap\_addresses

WHERE emp.empno = heap\_addresses.empno

AND emp.empno = 42;



Explain 2:

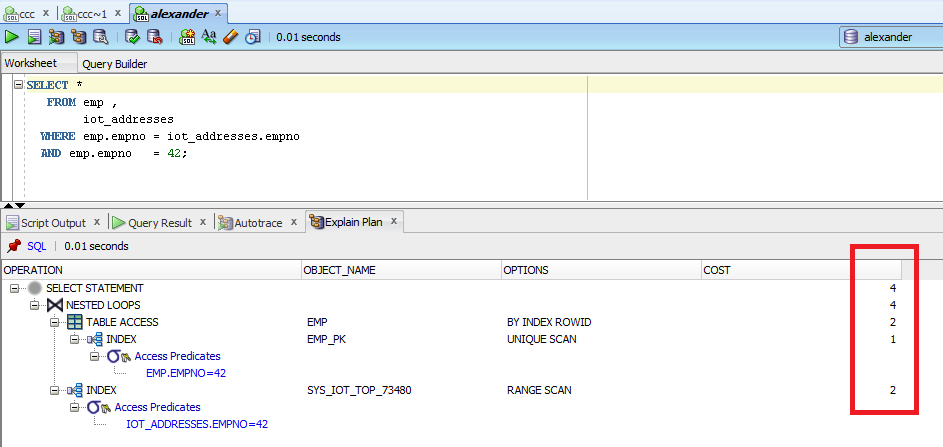
SELECT \*

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.

**RESULTS**:

In a **Heap Table**, data comes wherever there is place, at the end of the table. Getting to the data means getting to the index -> extracting the row id’s -> going to the table.

In an **Index Organized Table**, the data is stored in a “group by” fashion, based on the primary key. Getting to the data means getting to the index and that’s it. There is no need for an extra I/O on the table, all data is in the index.

That’s why Heap table cost > IOT table cost

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:

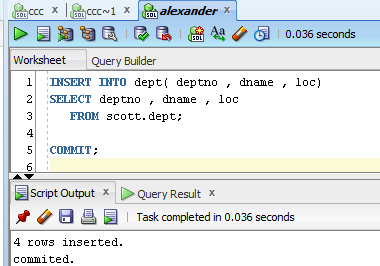
INSERT INTO dept( deptno , dname , loc)

SELECT deptno , dname , loc

FROM scott.dept;

commit;

**Results:**



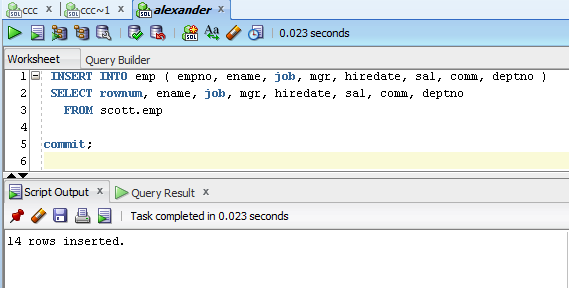
INSERT INTO emp ( empno, ename, job, mgr, hiredate, sal, comm, deptno )

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

FROM scott.emp

commit;

**RESULTS:**



**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

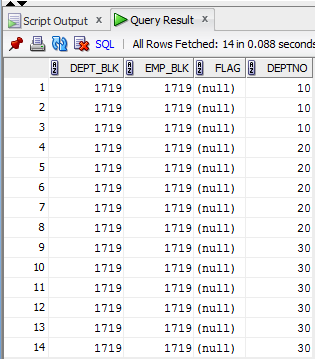
WHERE emp.deptno = dept.deptno

)

)

ORDER BY deptno

**RESULTS:**



Clusters are groups of one or more tables physically stored together because they share common columns (**deptno NUMBER( 2 ) in our case**) and are often used together. This improves disk access time.

The related columns of the tables in a cluster are called the **cluster key**. The cluster key is indexed so that rows of the cluster can be retrieved with a minimum amount of I/O.

No matter how many tables within the cluster contain the **cluster key** value, it is stored only once each in the cluster and the cluster index. Therefore, less storage is required.

Whether or not a table is part of a cluster is transparent to users and to applications. Data stored in a clustered table is accessed by SQL in the same way as data stored in a nonclustered table.

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).

**Results**:

To find or store a row in a hash cluster, Oracle applies a hash function to the row's cluster key value. The resulting hash value corresponds to a data block in the cluster.  
All rows with the same key value are stored together on disk.

The resulting hash key value points directly to the area on disk that stores the rows.

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

SIZE 1024

STORAGE (INITIAL 100K NEXT 50K

MINEXTENTS 1 MAXEXTENTS 3

PCTINCREASE 0)

HASH IS deptno HASHKEYS 150;

