Report

Laboratory Work 3

Dmitry Ladutsko

July 19, 2022

### Prerequisites

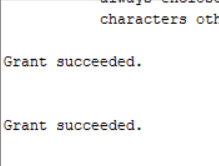
create tablespace tbs\_lab datafile 'db\_lab\_001.dat'

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

create user $DLadutko$ identified by 220025220025 default tablespace tbs\_lab;

grant connect to $DLadutko$;

grant resource to $DLadutko$;



*Picture 1 - Result*

*Note.* I found scott schema in the internet. It is attached in the lab script files named ‘scott schema’.

## 

### Task 1 – Heap Understanding

Step 1:

create table t

( a int,

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

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

)

/



*Picture 1.1 - Result*

Step 2:

insert into t (a) values ( 1);

insert into t (a) values ( 2);

insert into t (a) values ( 3);

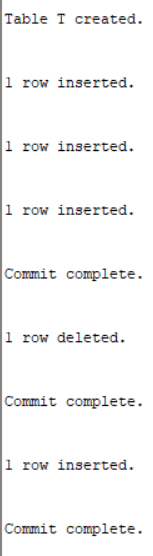
commit;

delete from t where a = 2 ;

commit;

insert into t (a) values ( 4);

commit;

**

*Picture 1.2 – Insert Result*

Step 3:

select a from t;



Picture 1.3 - Query Result

Clean up:

drop table T;



Picture 1.4 - Result

### Task 2 – Understanding Low level of data abstraction:

### Heap Table Segments

Step 1:

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



Picture 2.1 – Creation Result

Step 2:

PURGE RECYCLEBIN;

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



Picture 2.2 - Query Result

Step 3:

Create table t

( x int primary key,

y clob,

z blob )

SEGMENT CREATION IMMEDIATE

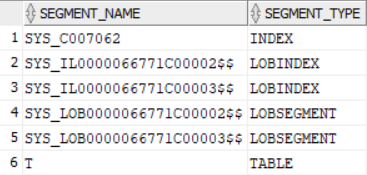
/



Picture 2.3 - Creation Result

Step 4:

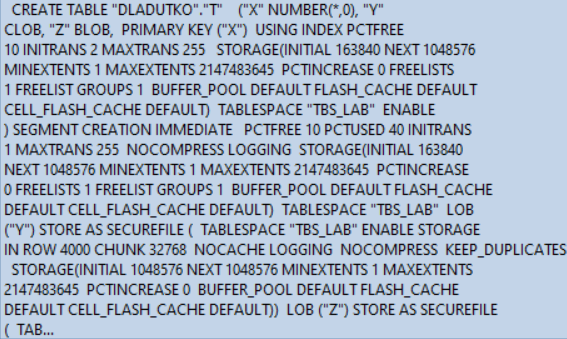
select segment\_name, segment\_type from user\_segments;



Picture 2.4 - Query Result

Step 5:

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



Picture 2.5 - Query Result

### Task 3: Index Organized Tables:

### 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

/



Picture 3.1 – Creation Result

Create Index:

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



Picture 3.2 – Altering Result

begin

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

end;



Picture 3.3- Procedure 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)

)

/



Picture 3.4 - Creation Result

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

/



Picture 3.5 - Creation Result

Step 4:

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;

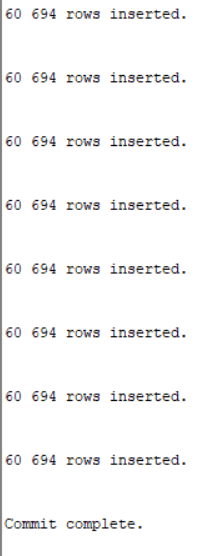
INSERT INTO heap\_addresses

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;

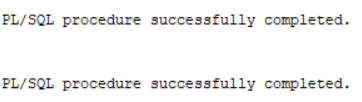


Picture 3.6 – Insertion Result

Step 5:

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

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



Picture 3.7- Procedure Result

Step 6:

*Explain 1:*

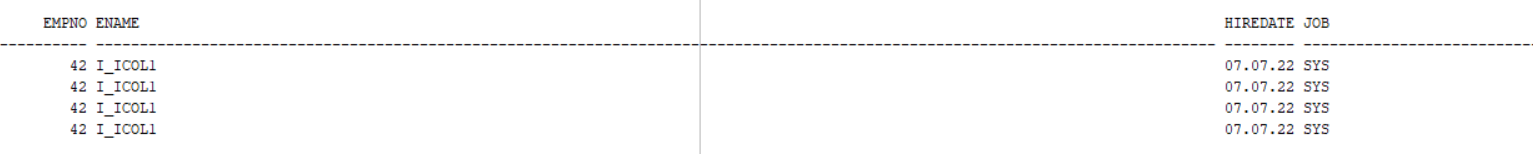
SELECT \*

FROM emp ,

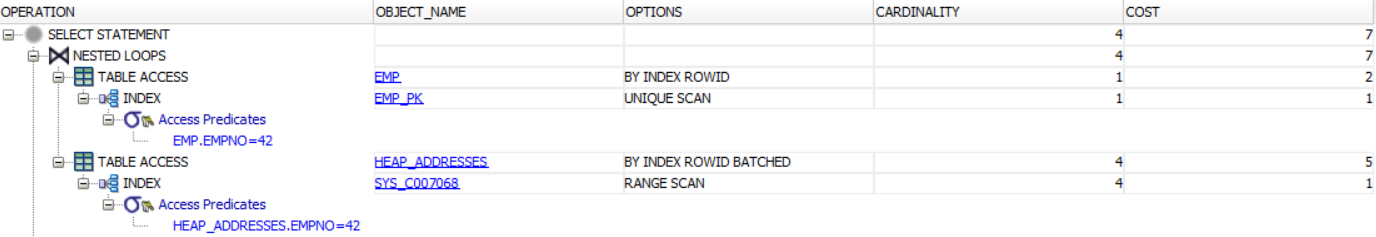
heap\_addresses

WHERE emp.empno = heap\_addresses.empno

AND emp.empno = 42;



Picture 3.8 - Select Result



Picture 3.9 – Explain Plan

*Explain 2:*

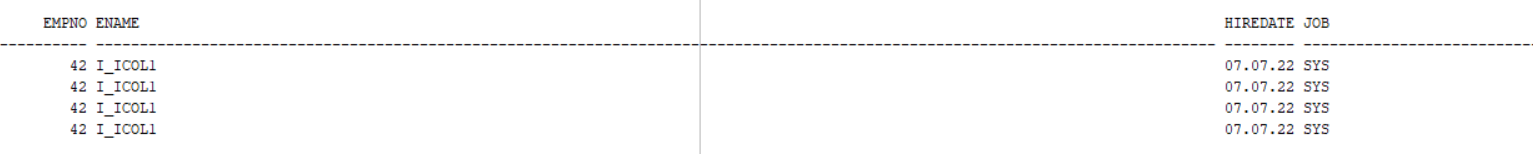
SELECT \*

FROM emp ,

iot\_addresses

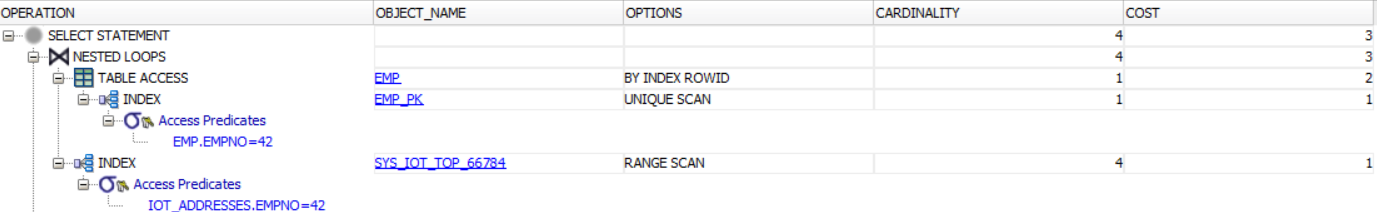
WHERE emp.empno = iot\_addresses.empno

AND emp.empno = 42;



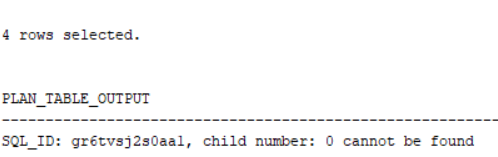
Picture 3.10 - Query Result

*Note*. Same SELECTS!



Picture 3.11 – Explain Plan

*Note*. Even though selects are same, the number of operations using IOT is less. We can see it using Explain Plan Oracle functionality, HOT select costs 7 points, but IOT Select cost 3 points.

*Table

Description automatically generated with medium confidence*

Picture 3.12 - Query Result and Statistics HOT

*A picture containing graphical user interface

Description automatically generatedTable

Description automatically generated*

Picture 3.13 - Query Result and Statistics for IOT

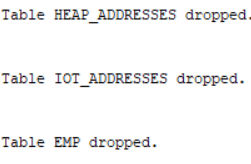
*Note.* If we compare separate statistic point, we can see that in most of cases the amount of individually compared operations in ION are less then HOT.

Step 7:

drop table emp;

drop table heap\_addresses;

drop table iot\_addresses;



Picture 3.14 – Clean Up Schema

### 

### Task 4: Analyses Cluster Storage by Blocks

Step 1:

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

SIZE 1024

STORAGE( INITIAL 100K NEXT 50K );



Picture 4.1 – Cluster Creation

Step 2:

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



Picture 4.2 - Index Creation

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

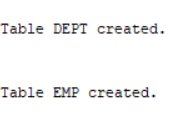
, sal NUMBER

, comm NUMBER

, deptno NUMBER( 2 ) REFERENCES dept( deptno )

)

cluster emp\_dept\_cluster ( deptno ) ;



Picture 4.3 - Result

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.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

Table

Description automatically generated

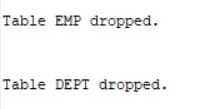
Picture 4.4 - Query Result

Step 5:

drop table dept;

drop table emp;

drop cluster emp\_dept\_cluster;





Picture 4.5 – Table and Cluster Clean Up

Step 6:

Scheme is empty.

### 

### Task 5: Analyses Cluster Storage by Blocks

CREATE CLUSTER emp\_dept\_clusterr (

deptno NUMBER( 2 ) )

HASHKEYS 10000

HASH IS deptno

SIZE 256;



Picture 5.1 – Cluster Creation

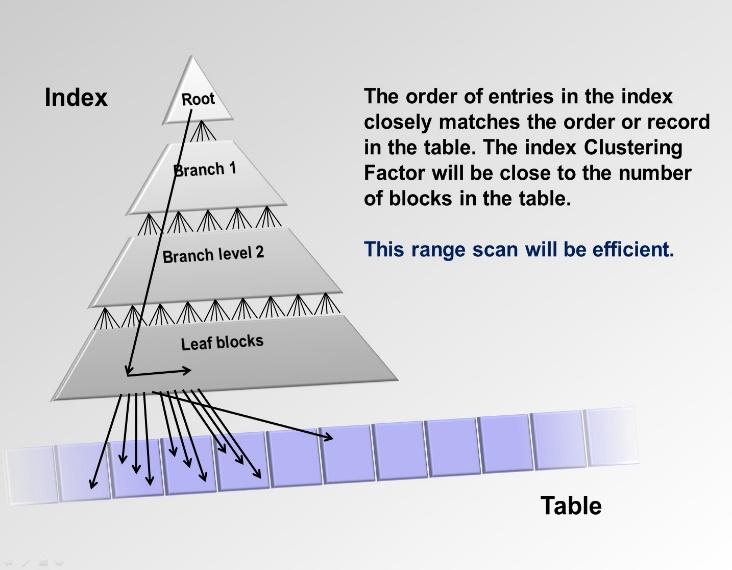
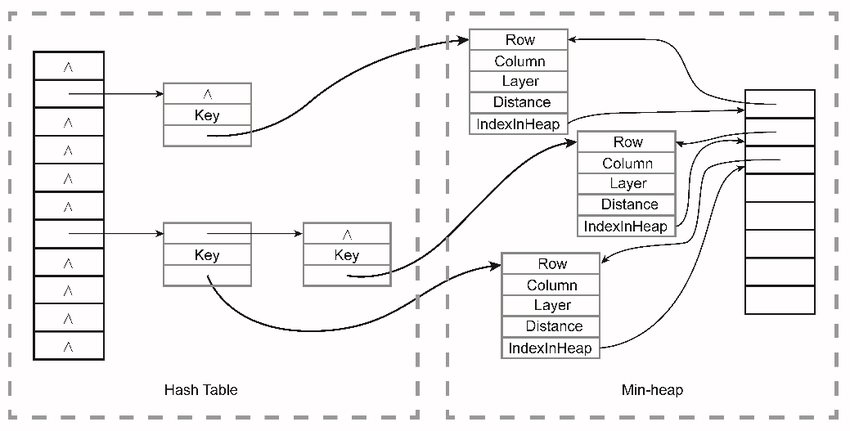
### Review

This lab was firstly about selection the amount of memory for blocks. We saw how we could change column separate cell size to ‘replace’ one.

We used data from sys user\_segments table to describe the storage allocated for segments by a current user’s object. Using SEGMENT CREATION IMMEDIATE, we saw that we can create a segment manually for new empty table, otherwise new segment will not be create until the first row is inserted into a partition (screenshots).

We also saw differences using both (IOT and HOT) on same SELECT’s by dint of Explain Plan Oracle functionality. Index-organized tables are useful when related pieces of data must be stored together or data must be physically stored in a specific order. In addition, we closely worked with heap organized and index organized tables. Here are some differences of these types of tables:

|  |  |
| --- | --- |
| Heap-Organized Table | Index-Organized Table |
| The [rowid](http://download.oracle.com/docs/cd/E11882_01/server.112/e16508/glossary.htm" \l "CHDEFIHG) uniquely identifies a row. Primary key constraint may optionally be defined. | Primary key uniquely identifies a row. Primary key constraint must be defined. |
| Physical rowid in ROWID [pseudocolumn](http://download.oracle.com/docs/cd/E11882_01/server.112/e16508/glossary.htm" \l "CHDJADFD) allows building secondary indexes. | Logical rowid in ROWID pseudocolumn allows building secondary indexes. |
| Individual rows may be accessed directly by rowid. | Access to individual rows may be achieved indirectly by primary key. |
| Sequential [full table scan](http://download.oracle.com/docs/cd/E11882_01/server.112/e16508/glossary.htm#CHDCGIFF) returns all rows in some order. | A [full index scan](http://download.oracle.com/docs/cd/E11882_01/server.112/e16508/glossary.htm#CHDDAHCB) or fast full index scan returns all rows in some order. |
| Can be stored in a [table cluster](http://download.oracle.com/docs/cd/E11882_01/server.112/e16508/glossary.htm#CHDJGGGF) with other tables. | Cannot be stored in a table cluster. |
| Can contain a column of the LONG data type and columns of [LOB](http://download.oracle.com/docs/cd/E11882_01/server.112/e16508/glossary.htm#CHDHHDDI) data types. | Can contain LOB columns but not LONG columns. |
| Can contain [virtual columns](http://download.oracle.com/docs/cd/E11882_01/server.112/e16508/glossary.htm#CHDDFACC) (only relational heap tables are supported). | Cannot contain virtual columns. |



An **index-organized table** allows you to store its entire data in an index. A normal index only stores the indexed columns; an index-organized table stores all its columns in the index.

A **heap-organized table** is a table with rows stored in no particular order. This is a standard Oracle table; the term "heap" is mostly used to differentiate it from an index-organized table.