

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 $DLadutko$;
```

Grant succeeded.

Grant succeeded.

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,'*')  
)  
/
```

Table T created.

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;

Table T created.

1 row inserted.

1 row inserted.

1 row inserted.

Commit complete.

1 row deleted.

Commit complete.

1 row inserted.

Commit complete.

Picture 1.2 – Insert Result

Step 3:

select a from t;

	A
1	1
2	4
3	3

Picture 1.3 - Query Result

Clean up:

drop table T;

Table T dropped.

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);

Table T created.

Picture 2.1 – Creation Result

Step 2:

PURGE RECYCLEBIN;

select segment_name, segment_type from user_segments;

no rows selected

Picture 2.2 - Query Result

Step 3:

Create table t

```
( x int primary key,  
  y clob,  
  z blob )
```

SEGMENT CREATION IMMEDIATE

/

Table T created.

Picture 2.3 - Creation Result

Step 4:

```
select segment_name, segment_type from user_segments;
```

	SEGMENT_NAME	SEGMENT_TYPE
1	SYS_C007062	INDEX
2	SYS_IL0000066771C00002\$\$	LOBINDEX
3	SYS_IL0000066771C00003\$\$	LOBINDEX
4	SYS_LOB0000066771C00002\$\$	LOBSEGMENT
5	SYS_LOB0000066771C00003\$\$	LOBSEGMENT
6	T	TABLE

Picture 2.4 - Query Result

Step 5:

```
SELECT DBMS_METADATA.GET_DDL('TABLE','T') FROM dual
```

```
CREATE TABLE "DLADUTKO"."T" ("X" NUMBER(*,0), "Y"  
CLOB, "Z" BLOB, PRIMARY KEY ("X") USING INDEX PCTFREE  
10 INITRANS 2 MAXTRANS 255 STORAGE(INITIAL 163840 NEXT 1048576  
MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0 FREELISTS  
1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT  
CELL_FLASH_CACHE DEFAULT) TABLESPACE "TBS_LAB" ENABLE  
) SEGMENT CREATION IMMEDIATE PCTFREE 10 PCTUSED 40 INITRANS  
1 MAXTRANS 255 NOCOMPRESSION LOGGING STORAGE(INITIAL 163840  
NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE  
0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE  
DEFAULT CELL_FLASH_CACHE DEFAULT) TABLESPACE "TBS_LAB" LOB  
("Y") STORE AS SECUREFILE ( TABLESPACE "TBS_LAB" ENABLE STORAGE  
IN ROW 4000 CHUNK 32768 NOCACHE LOGGING NOCOMPRESSION KEEP_DUPLICATES  
STORAGE(INITIAL 1048576 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS  
2147483645 PCTINCREASE 0 BUFFER_POOL DEFAULT FLASH_CACHE  
DEFAULT CELL_FLASH_CACHE DEFAULT)) LOB ("Z") STORE AS SECUREFILE  
( TAB...
```

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  
/
```



Table EMP created.

Picture 3.1 – Creation Result

Create Index:

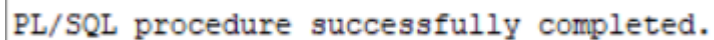
```
alter table emp add constraint emp_pk primary key(empno)
```



Table EMP altered.

Picture 3.2 – Altering Result

```
begin  
  dbms_stats.gather_table_stats( user, 'EMP', cascade=>true );  
end;
```



PL/SQL procedure successfully completed.

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)  
)  
/
```

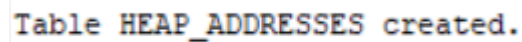


Table HEAP_ADDRESSES created.

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  
/
```

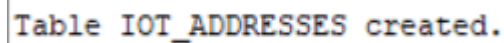


Table IOT_ADDRESSES created.

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;

```

```

60 694 rows inserted.

60 694 rows inserted.

60 694 rows inserted.

60 694 rows inserted.

60 694 rows inserted.

60 694 rows inserted.

60 694 rows inserted.

60 694 rows inserted.

Commit complete.

```

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' );

```


PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

Picture 3.7- Procedure Result

Step 6:

Explain 1:

```
SELECT *  
FROM emp ,  
      heap_addresses  
WHERE emp.empno = heap_addresses.empno  
AND emp.empno = 42;
```

EMPNO	ENAME	HIREDATE	JOB
42	I_ICOLL	07.07.22	SYS
42	I_ICOLL	07.07.22	SYS
42	I_ICOLL	07.07.22	SYS
42	I_ICOLL	07.07.22	SYS

Picture 3.8 - Select Result

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			4	7
NESTED LOOPS			4	7
TABLE ACCESS	EMP	BY INDEX ROWID	1	2
INDEX	EMP_PK	UNIQUE SCAN	1	1
Access Predicates				
EMP.EMPNO=42				
TABLE ACCESS	HEAP_ADDRESSES	BY INDEX ROWID BATCHED	4	5
INDEX	SYS_C007068	RANGE SCAN	4	1
Access Predicates				
HEAP_ADDRESSES.EMPNO=42				

Picture 3.9 – Explain Plan

Explain 2:

```
SELECT *  
FROM emp ,  
      iot_addresses  
WHERE emp.empno = iot_addresses.empno  
AND emp.empno = 42;
```

EMPNO	ENAME	HIREDATE	JOB
42	I_ICOLL	07.07.22	SYS
42	I_ICOLL	07.07.22	SYS
42	I_ICOLL	07.07.22	SYS
42	I_ICOLL	07.07.22	SYS

Picture 3.10 - Query Result

Note. Same SELECTS!

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				4
NESTED LOOPS				3
TABLE ACCESS	EMP	BY INDEX ROWID	4	3
INDEX	EMP_PK	UNIQUE SCAN	1	2
Access Predicates				1
EMP.EMPNO=42				
INDEX	SYS_IOT_TOP_66784	RANGE SCAN	4	1
Access Predicates				
IOT_ADDRESSES.EMPNO=42				

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.

Statistics	
1 CPU used by this session	
1 CPU used when call started	
1 DB time	
6 Requests to/from client	
18 consistent gets	
13 consistent gets examination	
13 consistent gets examination (fastpath)	
18 consistent gets from cache	
5 consistent gets pin	
5 consistent gets pin (fastpath)	
46 db block gets	
46 db block gets from cache	
46 db block gets from cache (fastpath)	
19 enqueue releases	
19 enqueue requests	
9 messages sent	
6 non-idle wait count	
20 opened cursors cumulative	
1 opened cursors current	
1 pinned cursors current	
25 recursive calls	
64 session logical reads	
7 user calls	

4 rows selected.

PLAN_TABLE_OUTPUT

SQL_ID: gr6tvsj2s0aal, child number: 0 cannot be found

Picture 3.12 - Query Result and Statistics HOT

Statistics	
1 CPU used by this session	
1 CPU used when call started	
1 DB time	
6 Requests to/from client	
5 consistent gets	
4 consistent gets examination	
4 consistent gets examination (fastpath)	
5 consistent gets from cache	
1 consistent gets pin	
1 consistent gets pin (fastpath)	
1 enqueue releases	
1 enqueue requests	
6 non-idle wait count	
2 opened cursors cumulative	
1 opened cursors current	
1 pinned cursors current	
1 recursive calls	
5 session logical reads	
7 user calls	

4 rows selected.

PLAN_TABLE_OUTPUT

SQL_ID: cjyysj5t08qkr, child number: 0 cannot be found

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;
```

```
Table HEAP_ADDRESSES dropped.
```

```
Table IOT_ADDRESSES dropped.
```

```
Table EMP dropped.
```

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 );
```

```
Cluster EMP_DEPT_CLUSTER created.
```

Picture 4.1 – Cluster Creation

Step 2:

```
CREATE INDEX idxcl_emp_dept on cluster emp_dept_cluster;
```

```
Index IDXCL_EMP_DEPT created.
```

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 );

```

Table DEPT created.

Table EMP created.

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

```

DEPT_BLK	EMP_BLK F	DEPTNO
323	323	20
323	323	20
323	323	20
323	323	30
323	323	30
323	323	30
323	323	30
323	323	30
323	323	30

9 rows selected.

Picture 4.4 - Query Result

Step 5:

```
drop table dept;
drop table emp;
drop cluster emp_dept_cluster;
```

```
Table EMP dropped.

Table DEPT dropped.

Cluster EMP_DEPT_CLUSTER dropped.
```

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;
```

```
Cluster EMP_DEPT_CLUSTER created.
```

Picture 5.1 – Cluster Creation

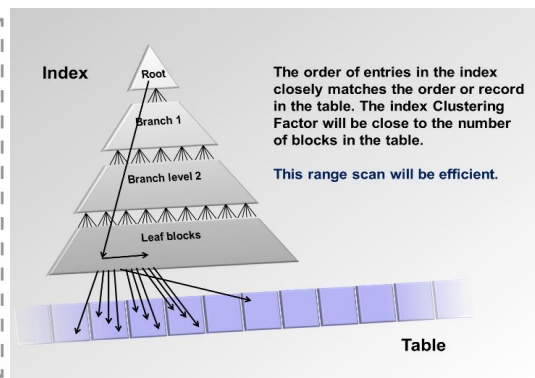
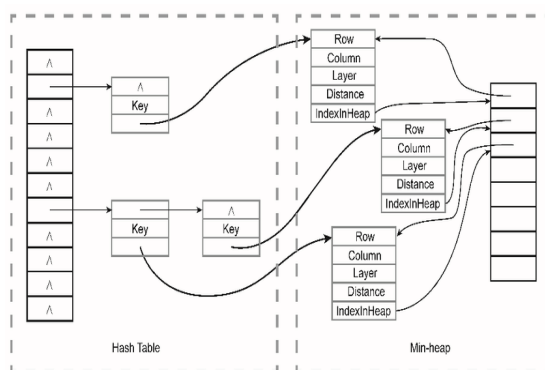
Review

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 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 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 returns all rows in some order.	A full index scan or fast full index scan returns all rows in some order.
Can be stored in a table cluster with other tables.	Cannot be stored in a table cluster.
Can contain a column of the LONG data type and columns of LOB data types.	Can contain LOB columns but not LONG columns.
Can contain virtual columns (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.