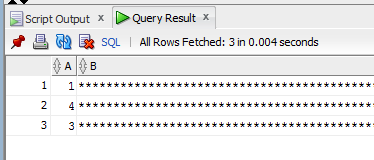
**Task1**

Select from t table:



Select with rowid\_object, rowed\_relative\_fno, rowid\_block\_number and rowid\_row\_number from t table:

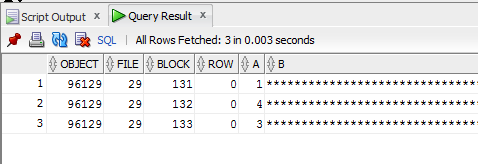
select DBMS\_ROWID.ROWID\_OBJECT(rowid) "OBJECT",

DBMS\_ROWID.ROWID\_RELATIVE\_FNO(rowid) "FILE",

DBMS\_ROWID.ROWID\_BLOCK\_NUMBER(rowid) "BLOCK",

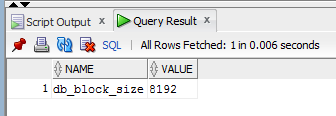
DBMS\_ROWID.ROWID\_ROW\_NUMBER(rowid) "ROW",

t.\* from t;

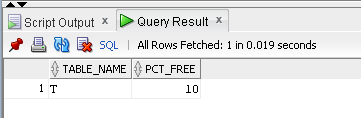


In the screenshot, we see that the rows are written in different blocks.

Size of block:



The pct\_free is:



Now we can try to calculate size of b and c in order to write three rows into one block:

1. Available size of blocks for writing rows

8192\*0,9=7372,8

1. Size for one row:

7372/3=2438,…

1. Size for b and c :

As lengthb(a)=1 for all our rows, so 2437-1=2436

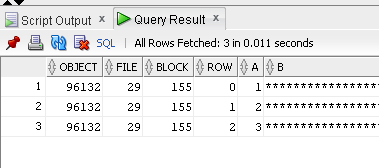
1. Let’s make the same size for b and c:

2436/2=1218

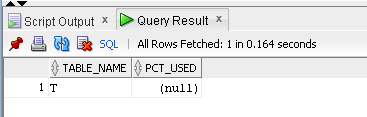
But this result is incorrect! The third row is written to the other block.

I suppose that the reason is about header of block. It takes some size of block and I didn’t take it into consideration. The size of header is difficult to calculate because it consists of mane parts with not fixed size.

Right size for b and c is 1203. When I set this size for b and c, rows are written into the same block:



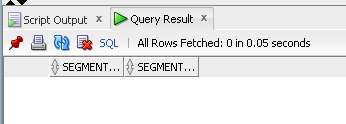
After this operations I wanted to look at pct\_used:



I found the following: « This parameter is ignored for objects created in locally managed [tablespaces](http://www.orafaq.com/wiki/Tablespace) with Segment Space Management ([ASSM](http://www.orafaq.com/wiki/ASSM)) specified as AUTO». We defined exactly this kind of tablespace for new user.

**Task2**

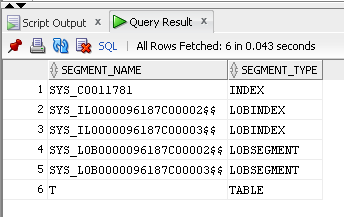
After creation of the first table, there is no segments:



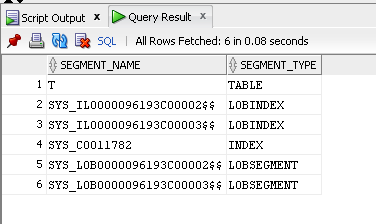
The reason is implicit using of default clause SEGMENT\_CREATION\_DEFERRED.

“This clause defers creation of the table segment — as well as segments for any LOB columns of the table, any indexes created implicitly as part of table creation, and any indexes subsequently explicitly created on the table — until the first row of data is inserted into the table.”

So when I added a row, segments appeared:



After recreating my user (to delete rows from user\_segments), the second table was created. There appeared segments:

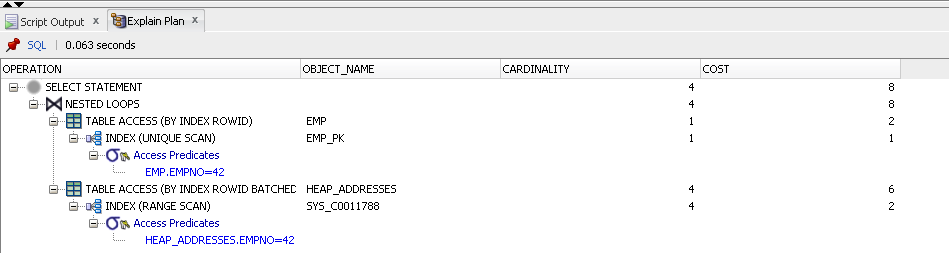


This SEGMENT CREATION IMMEDIATE clause overrides the setting of the DEFERRED\_SEGMENT\_CREATION initialization parameter. It can be seen during comparing ddl of two tables.

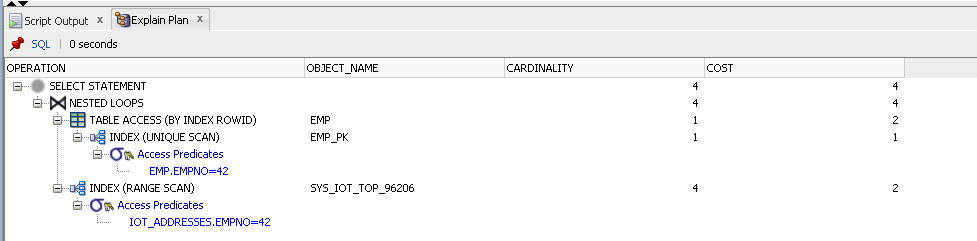
The table segment is created as part of the CREATE TABLE statement.

**Task3**

Execution plan for heap table



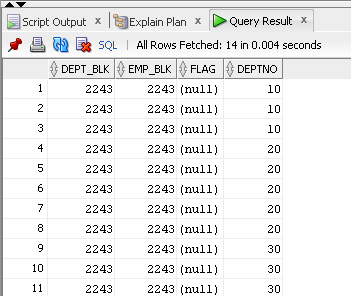
Execution plan for index organized table



Oracle Index-organized tables are stored in a B-tree index structure. Each index entry in the B-tree also stores the non-key column values. Oracle Indexed-organized tables provide faster access to table rows by the primary key or any key that is a valid prefix of the primary key. Because the non-key columns of a row are present in the B-tree leaf block itself, there is no additional block access for index blocks.

**Task4**

Result of query:



Data that contains the same cluster key value is physically stored together (on the same block). So rows from tables dept and emp with equal values of deptno (cluster key) are stored in the same blocks.

**Task5**

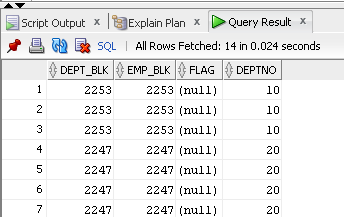
Query to create hash cluster:

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

SIZE 1024 hashkeys 100

STORAGE( INITIAL 100K NEXT 50K );

Result:



**Task6**

create table mig\_chain(

x int primary key,

a char(2000),

b char(2000),

c char(2000),

d char(2000),

e char(2000)

);

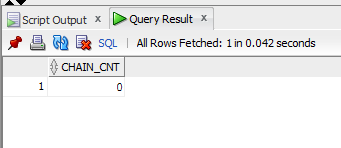
INSERT INTO mig\_chain (x) VALUES (1);

INSERT INTO mig\_chain (x) VALUES (2);

INSERT INTO mig\_chain (x) VALUES (3);

COMMIT;

No here is no fetched rows (size of rows aren’t bigger than size of block):



UPDATE mig\_chain SET a = 'z1', b = 'z2', c = 'z3' WHERE x = 3;

COMMIT;

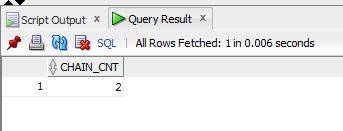
UPDATE mig\_chain SET a = 'y1', b = 'y2', c = 'y3' WHERE x = 2;

COMMIT;

UPDATE mig\_chain SET a = 'w1', b = 'w2', c = 'w3' WHERE x = 1;

COMMIT;

Now here is migrated rows (this rows have too large size for one block):



UPDATE mig\_chain SET d = 'z4', e = 'z5' WHERE x = 3;

COMMIT;

Now I have two migrated rows and one chained

