Heap Understanding:

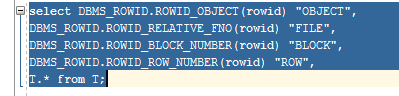
I create table, insert 3 values and then I checked size of my blozk\_size by

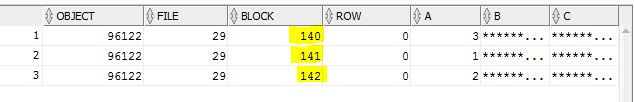


And here my result:

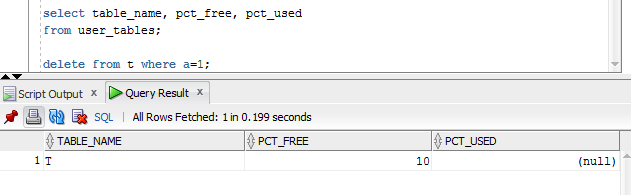


Then I checked how my rows are stored in blocks by :

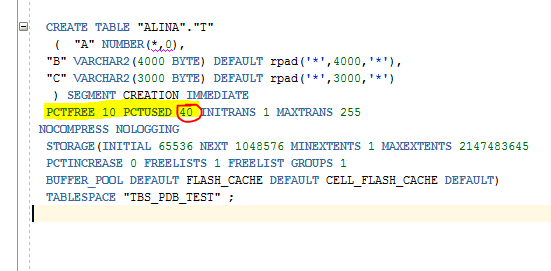




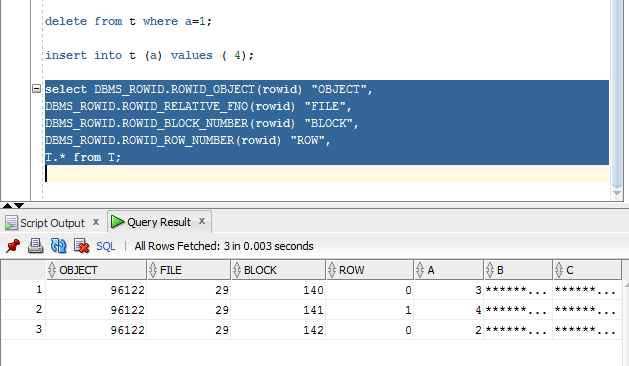
Rows 1,2,3 are stored in different block because size of rows ~7010 bytes and size of our block is 8192 bytes.

Then I checked pct\_free and pct\_used and I saw this result: 

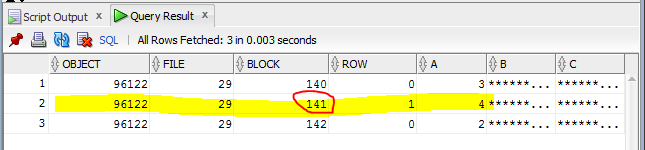
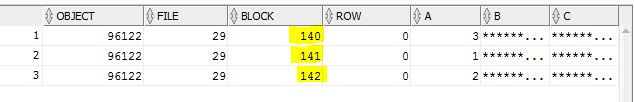
Pct\_used is null because in ASSM, PCTUSED is simply ignored. But I checked it by ddl script:



Then I delete rows where a=1 and insert rows with a=4 and check one more time blocks.



Here you can see that our 141th block was used by another row with value 4 for a and value row on the second screenshots is changed for 1:



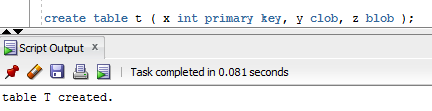
As data is added, the first free space found in the segment that can fit the data will be used. And in our example is 141 was full, then we deleted row (it means that now 141 block is empty) and when we insert new information Oracle used 141 block for new information.

As from courseware : As data is removed from the table, it allows space to become available for reuse by subsequent INSERTs and UPDATEs.

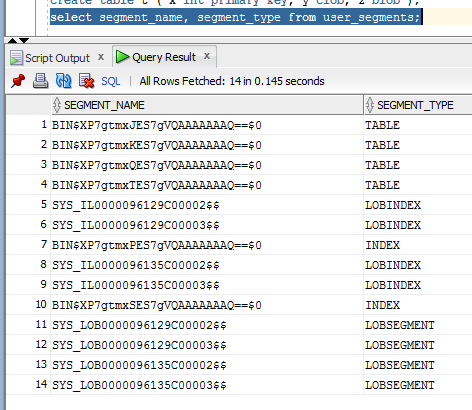
This task is show it to us.

Understanding Heap Table Segments

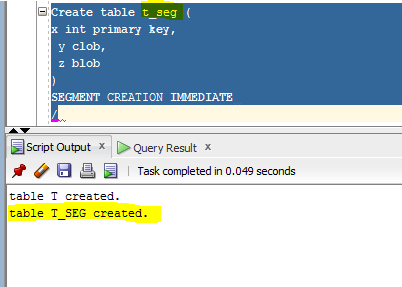
We create first table:



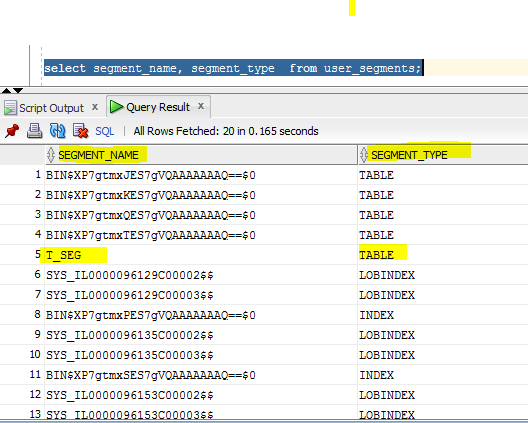
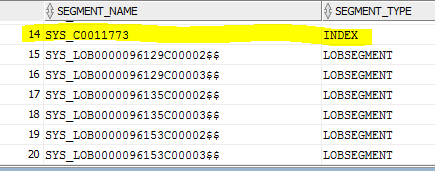
Then I checked user\_segments and I didn’t find out table and it’s ok, because it’s just table.



I create another table:



And also checked it from user\_segments:

The difference is for empty table t wasn’t create segment but for empty table t\_seg was create segment table type and indes SYS\_C0011773. This will approve from DDl script from 2 tables:

Table t:

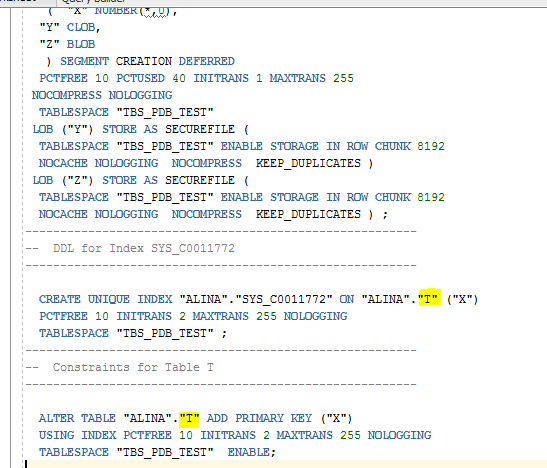
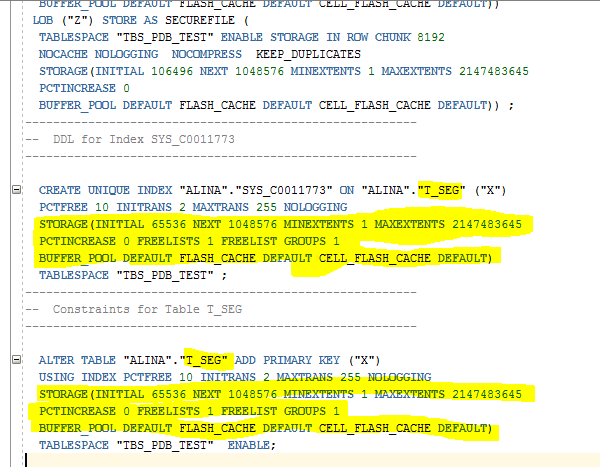


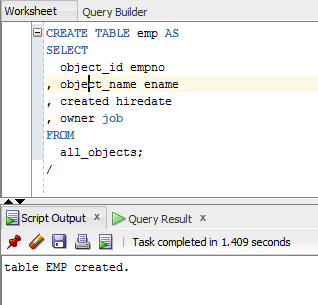
Table t\_seg:



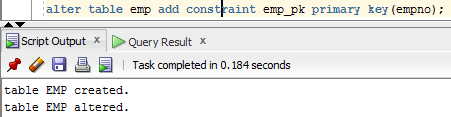
Conclusion is when non-partitioned tables are created, none of the associated segments (table, implicit index and LOB segments) are created until rows are inserted into the table. For systems with lots of empty tables, this can represent a large space saving.

# Index Organized Tables:

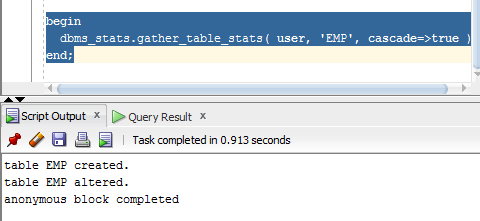
I create table:



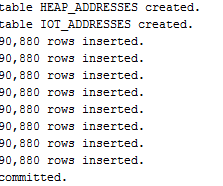
Then I add constraint for PK:



And Calculation:



Than I create 2 tables and insert some values:



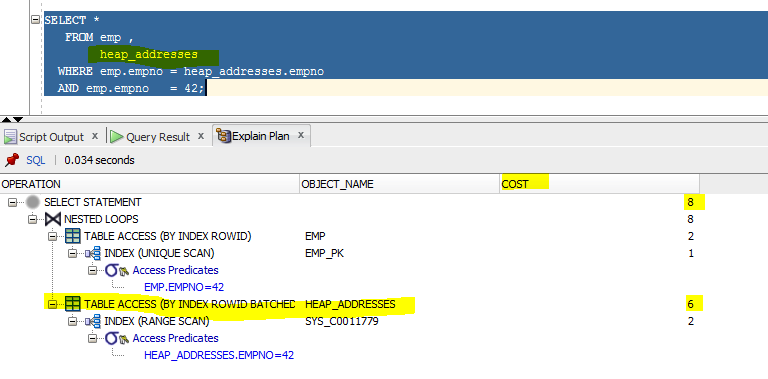
I create table IOT\_ADDRESSES and this table is index-organized. Index creates after creating table. When some values are inserting in just table ( not index-organised), they will be placed in first free block without any sorted. Index-organised table consist from values that we inserted and they are sort by PK that we used for creating.

I calculated statistic:

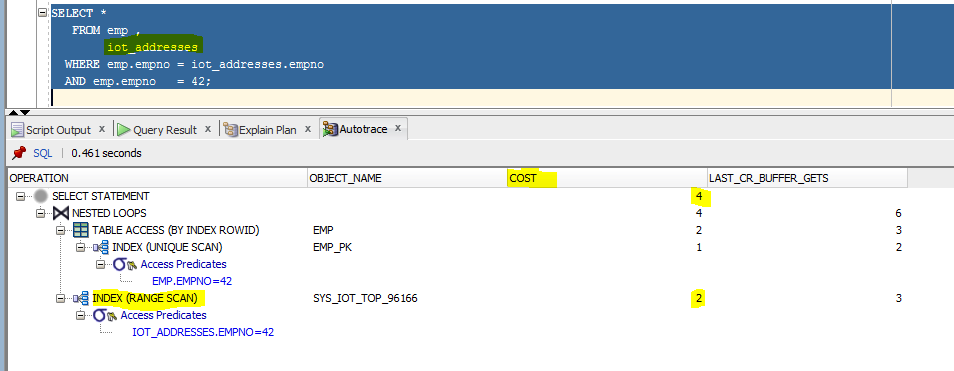


And now I will compare 2 execution plan for heap table and index-organized(IO) table:

Heap table:



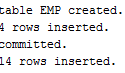
IOT:



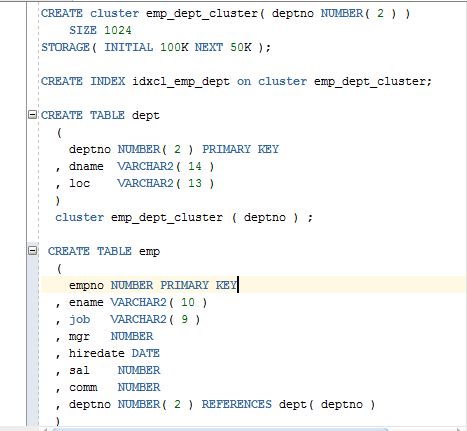
This result show to us heap table with index will using TABLE ACCESS by index rowed ( like pointer/cursor for data), but when we are using IOT we can see that ORCALE use just Index (range scan) without any TABLE ACCESS. It because IOT index consist from data based on PK. Index IOT consist from usual data and data from PK. It means ORACLE shouldn’t remember all data from table, IOT has PK+all data from table. And IOT has better execution plan than heap tables.

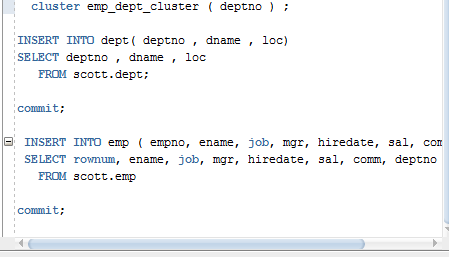
# Index Clustered Tables:



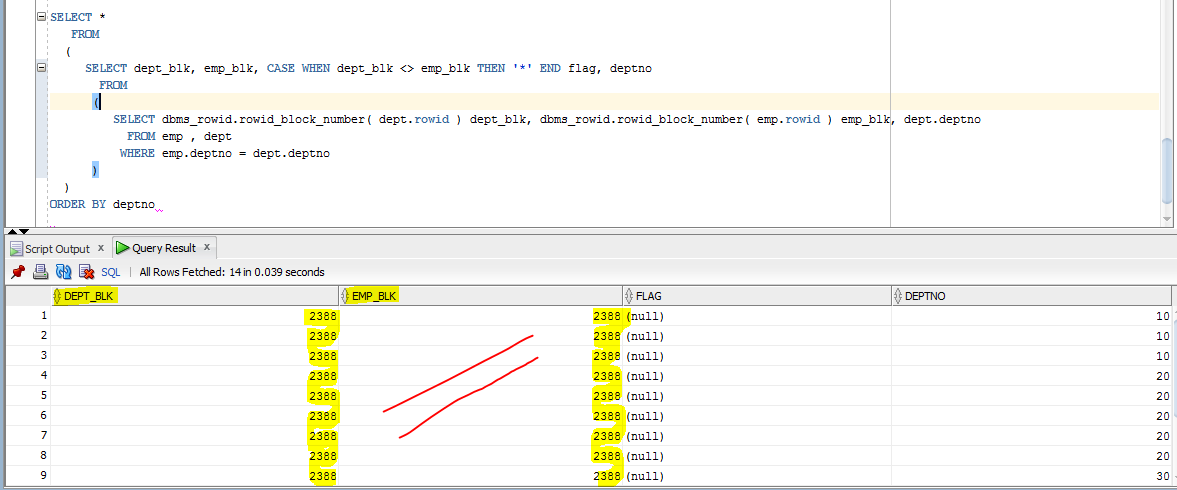


First of all I create cluster, than I create index for cluster and 2 tables and insert some values.

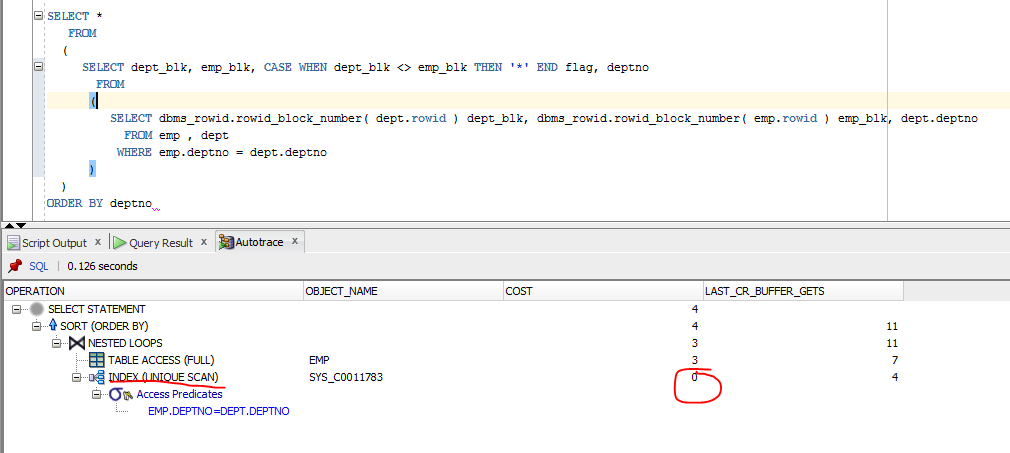




Now we have 2 tables with one shared cluster column. From physical point of view this is one column (maybe, I’m not sure).

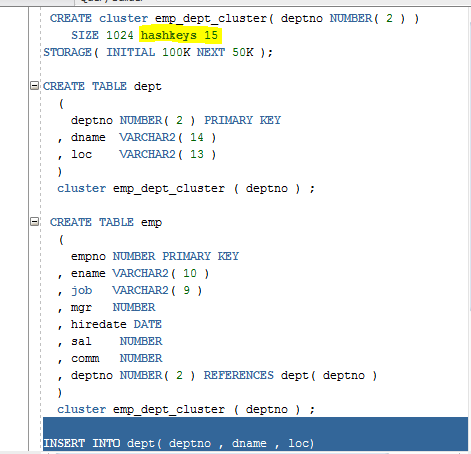


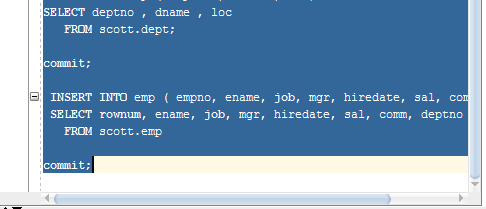
We execute one query and as you can see values from dept\_bul the same from emp\_bulk=> dept.rowid=emp.rowid. it means that this data holding in one block. Access to this information is fasting, because we don’t need to read 2 another column from different tables because this column consist in our cluster.

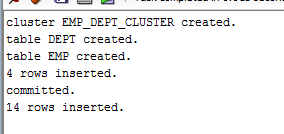


As we can see cost from index = 0. For reading it so fast and cluster for reading is good. But if we know that our dates will be rewrite or update many times we shouldn’t use ICT.

# Hash Clustered Tables:

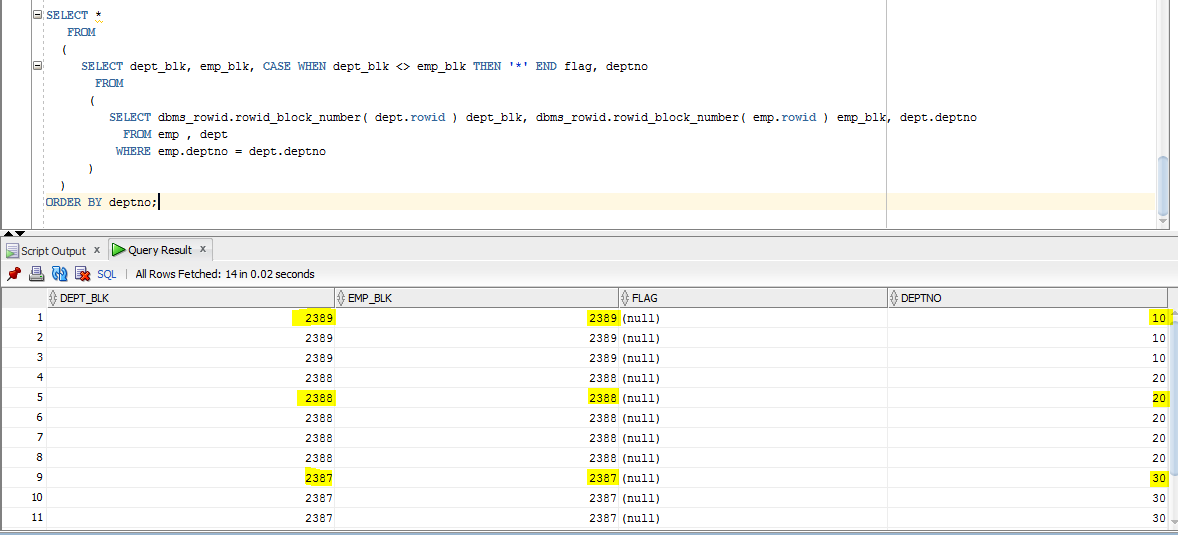






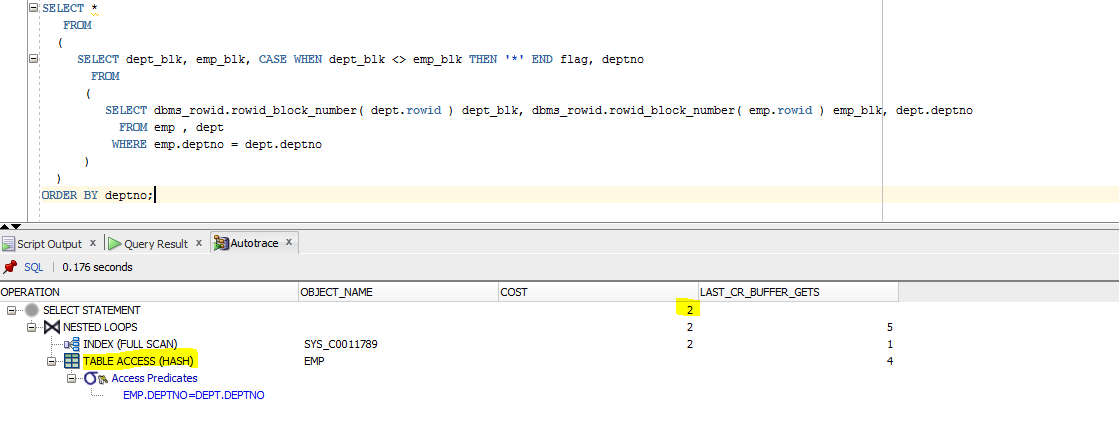
I create cluster but with HASHKEYS 15. Hashkey control size of cash for our cluster. 15\*size\_for\_cluster and this size will be available for our cash.

And I didn’t create index for cluster, all my doing was repeated from previous task.



We have the same values dept\_bulk and emp\_bulk but for every deptno. It means that hash cluster stored all rows for one deptno, for anouther deptno hash cluster stored another rows because they have another hash key value generated by hash function.

Execution plan for this query:

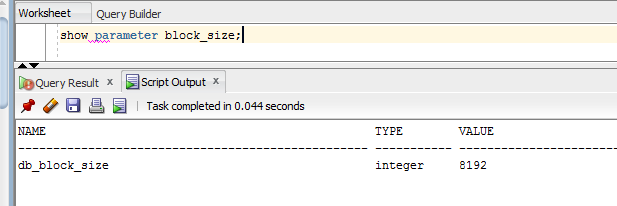


Execution plan shows to us that our data are stored together and we don’t need to join 2 tables, it made performance faster.

# Row Migration:

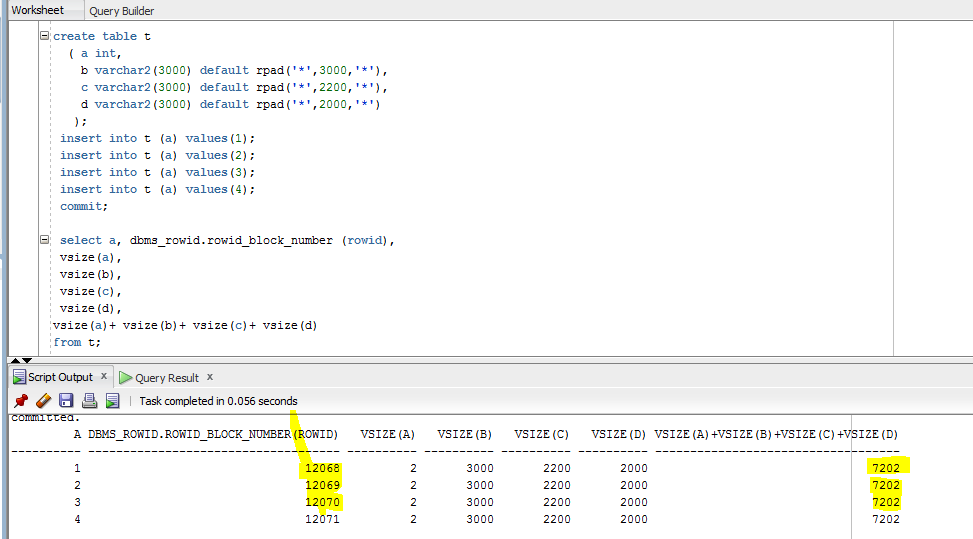
Row migration:

One more time I checked size of block:

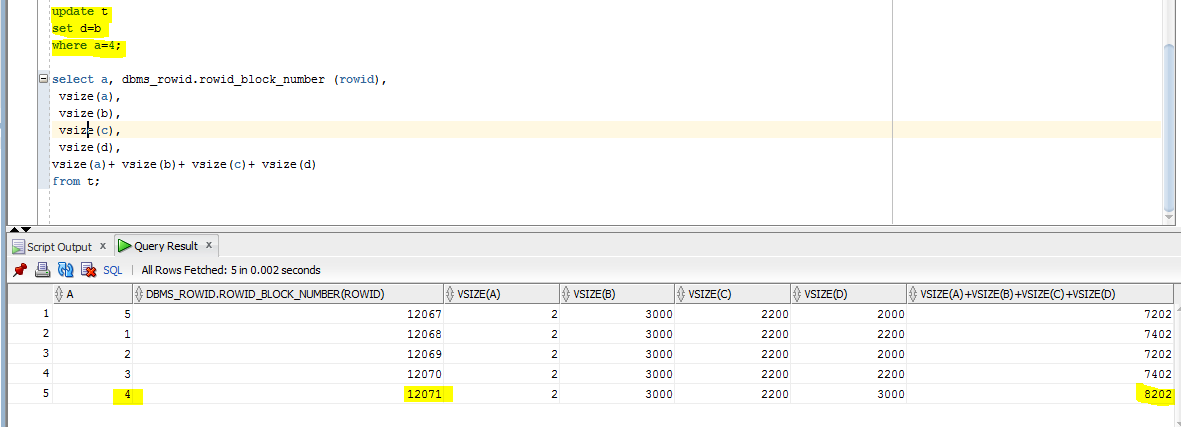


Size of our block is 8192 and pct\_free is 10%. It means that ~ 7300 we can use.

I create table and insert some values. Also I checked size of blocks and number of blocks.



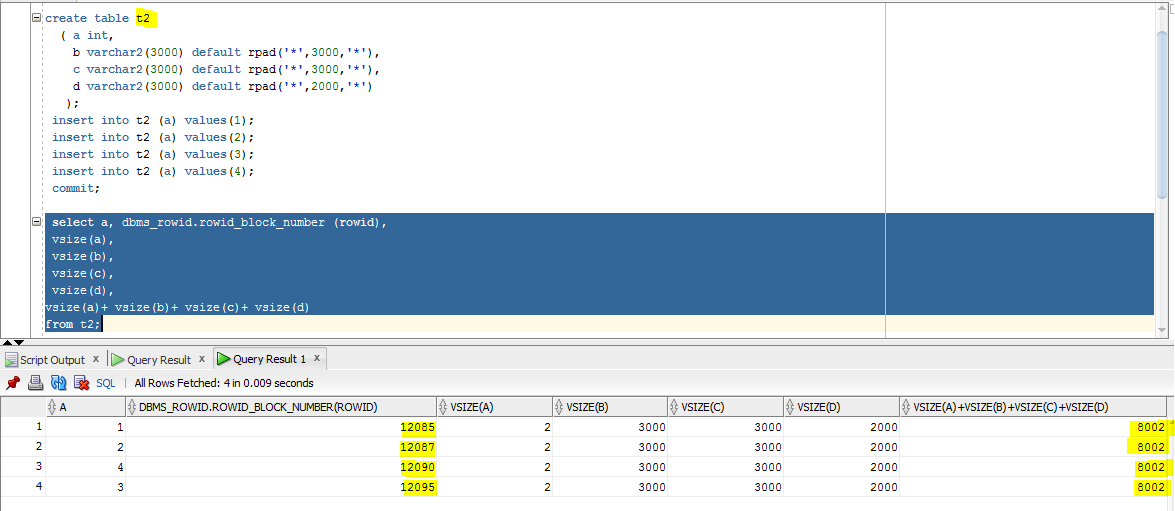
Then I update forth row:



As you can see row with a=4 still stay in the same block but its size is changed. Now our rows store in 12071 block and also this block have cursor/pointer for another block where end of our row is stored.

Rows migration is typically caused by UPDATE statement.

Row chaining:

I also create table but I put information >0.9 \*block\_size. And as result you can see every row occupies 2 blocks.

Row chaining is typically caused by INSERT statement.