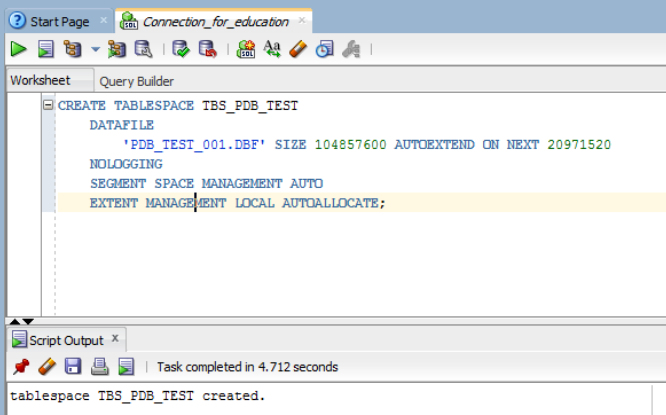
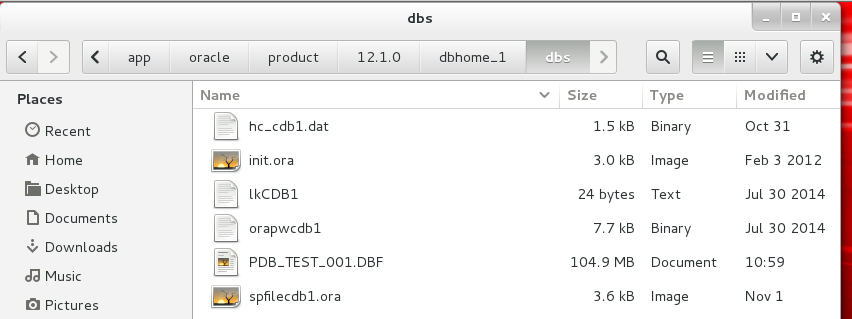
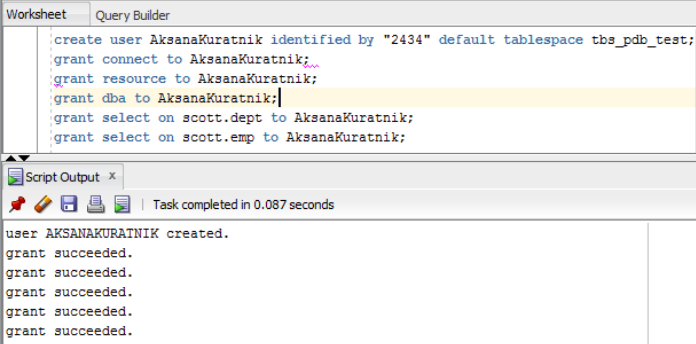
# Prerequisite Task

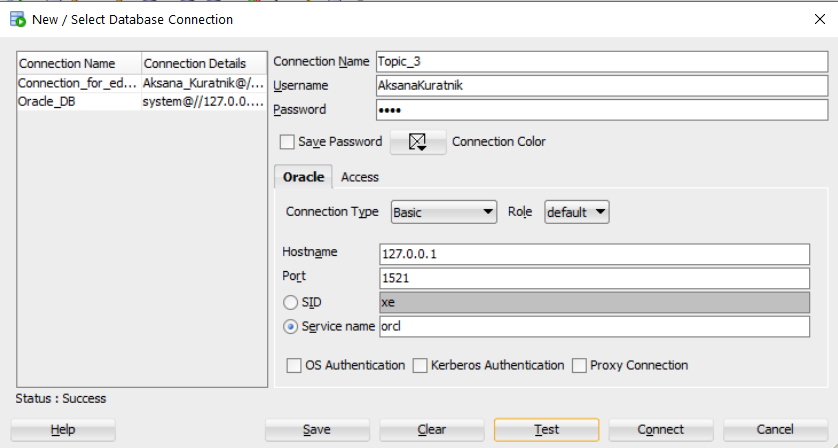
Createion of a new tablespace tbs\_pdb\_test with the new datafile pdb\_test\_001.dbf:





New user was created with all necessary grants.

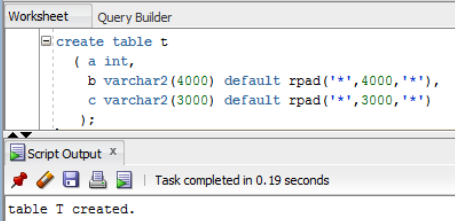




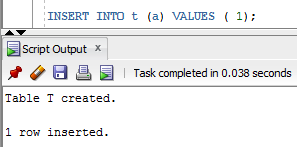
# Heap Organized Tables

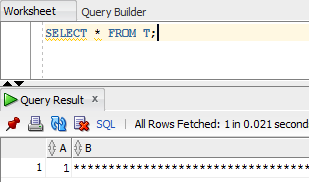
## Task results of task 1 – Heap Understanding

A new table was created.  
I created table with 3 columns. There are the Primary Key at the first column. The second and the third columns contain symbols “\*” by default. The second column contains 4000 symbols “\*”. The third column contains 3000 symbols “\*”.

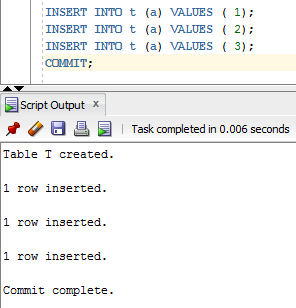


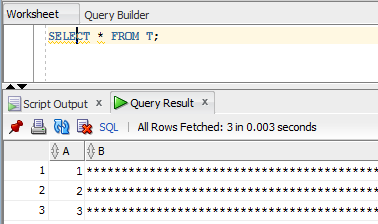
I inserted a row on the table. The PK of the row is “1”.



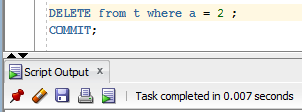


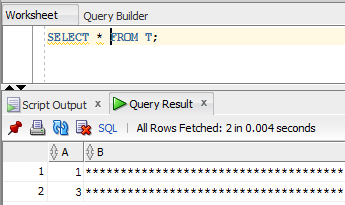
I inserted other 2 rows on the table. The PK of the rows are “2” and “3”. At the end I executed the commit command to save the changes.





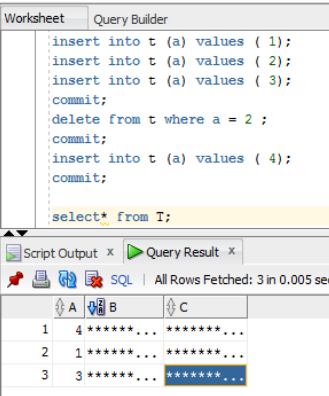
Then I dropped the row where PK was 2 and executed the commit command again.





After that I inserted the row with PK=4 and executed the commit command to save the changes again.

select a from t;

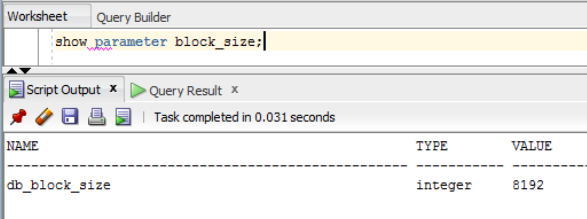


## Task Results:

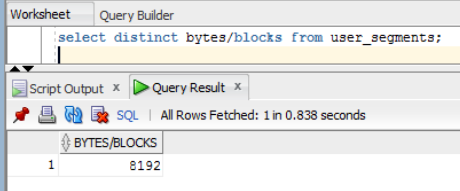
NOTE: Adjust columns B and C to be appropriate for your block size if you would like to reproduce this. For example, if you have a 2KB block size, you do not need column C, and column B should be a VARCHAR2(1500) with a default of 1,500 asterisks. Since data is managed in a heap in a table like this, as space becomes available, it will be reused.

I check the block size with three methods. The block size is 8192.

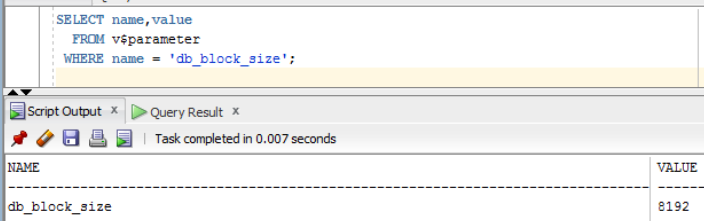
The first method:



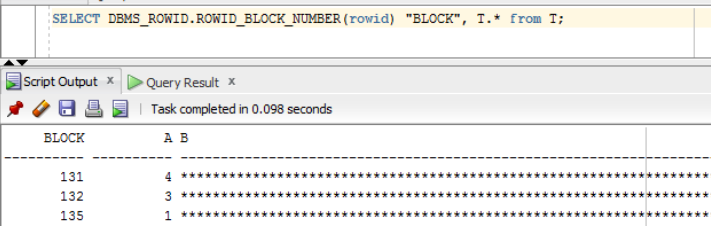
The second method:



The third method:

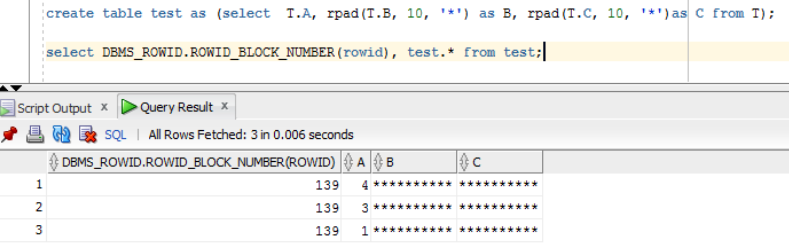


Then I checked in what blocks the rows were inserted.



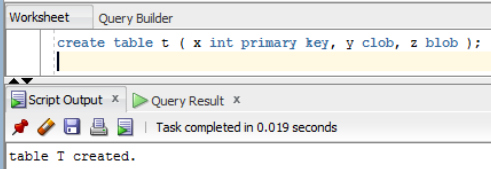
There is one row for a block. It happens because block size is 8192 and row size is 7004 (4(A)+4000(B)+3000(C))

I checked what would be changed if I shortened the length of the row. I created the table with the same structure but the length of rows B and C were 10 and 10 instead of 4000 and 3000 as it was before. Then I saw that the rows were on the same block(139). It happened because the block size were big enough for all rows. That means that we may shorten the block size or increase the length of the rows.

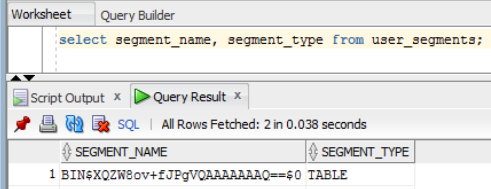


## Task Results of task 2 – Understanding Heap Table Segments

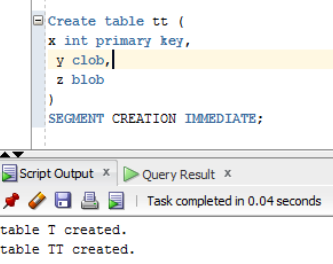
The table t was created.



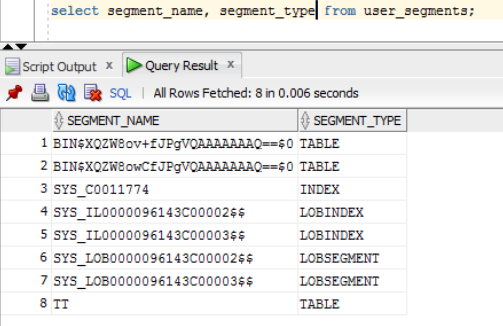
Then I saw the segments.



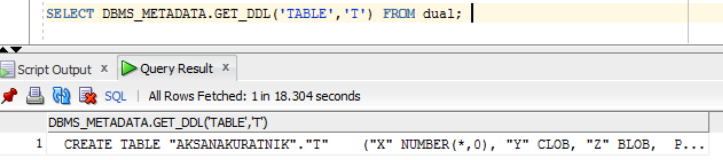
Then I created new table and added the command to make the segments automatically.

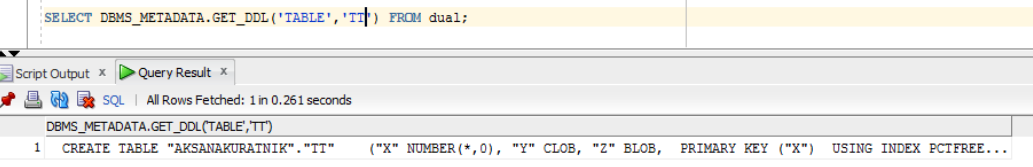


It was created the logical segment.



Then I saw metadata about two tables.

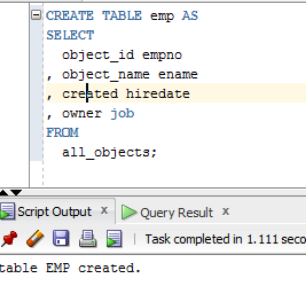




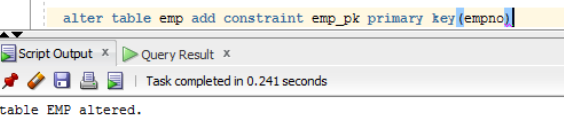
# Index Organized Tables

## Task results of task 3: Compare performance

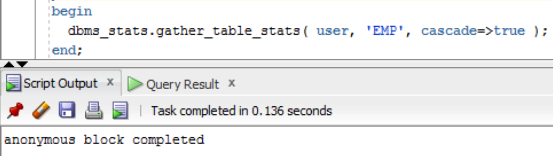
Table emp was created.



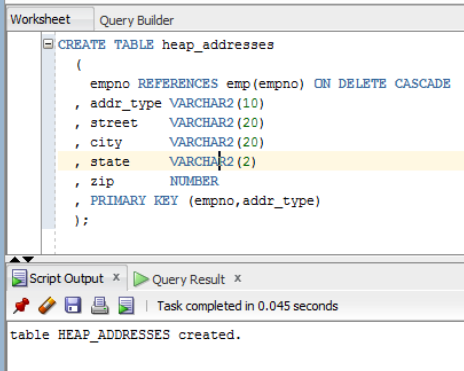
Index on PK was created.



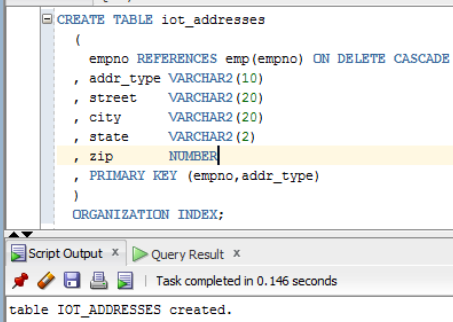
Statistics was calculate.



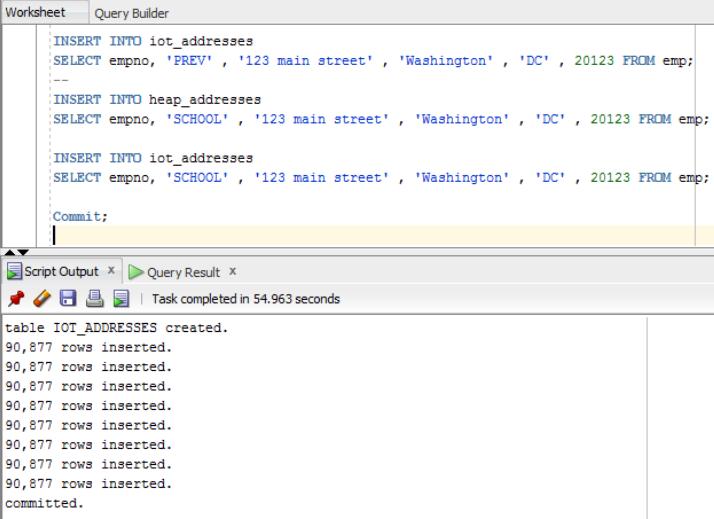
The table heap\_addresses was created.



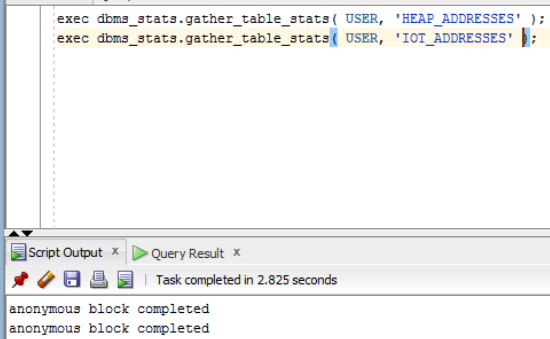
The table iot\_addresses was created.



Then I insert values into tables.



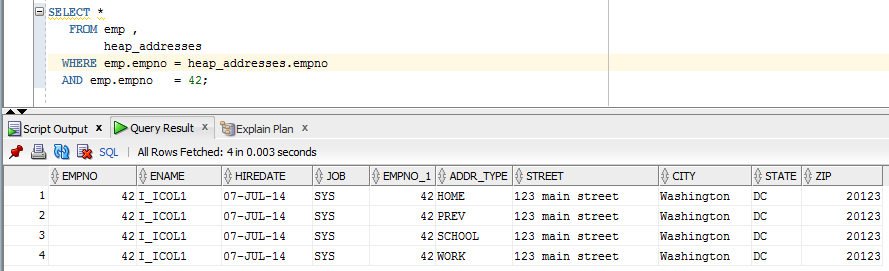
Statistics was calculated.

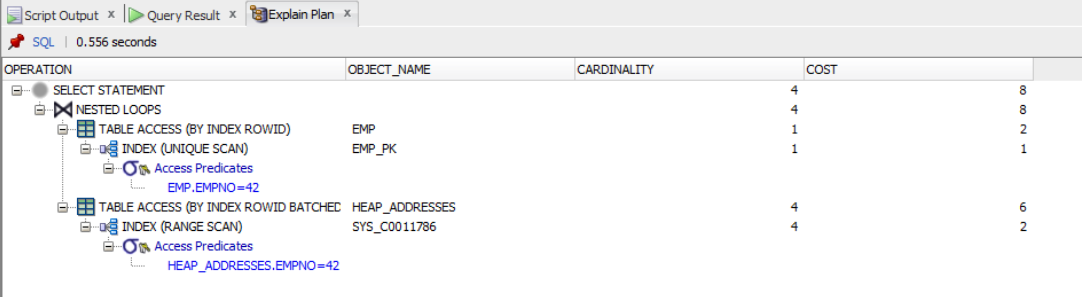


Compare Trace and Performance:

Explain Plan 1:

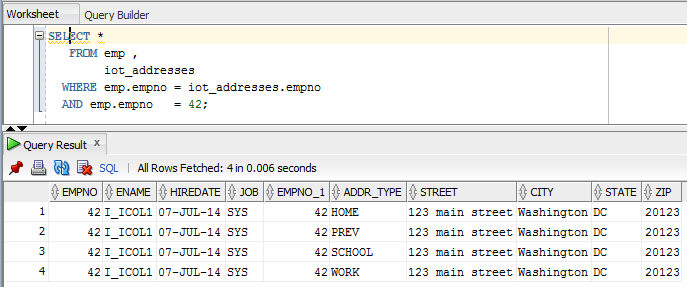
When we use a normal (HEAP) table and create an index on it, that index uses the physical (physical) rowid as a pointer to the address of the record itself.

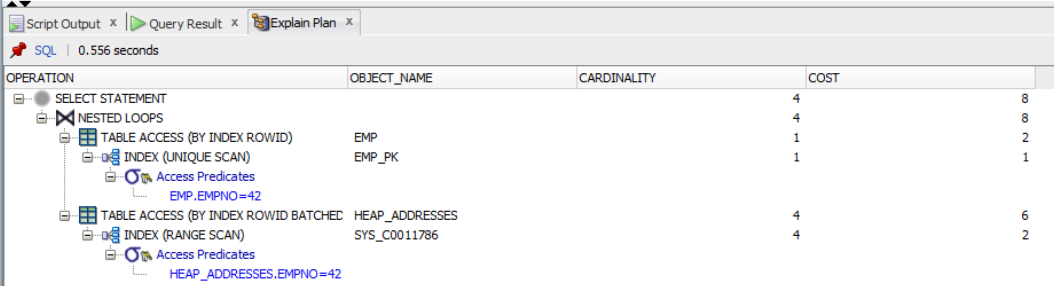




Explain Plan 2:

For indexed tables, a slightly different approach is used: an index is immediately created based on the primary key and includes other columns of the record.



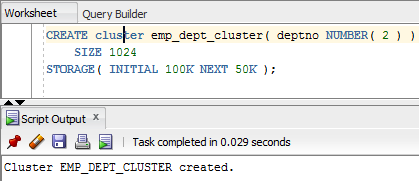


The cost of searching for an ISP is less (Explain Plan 2) than the cost of a heap (Explain Plan 1).

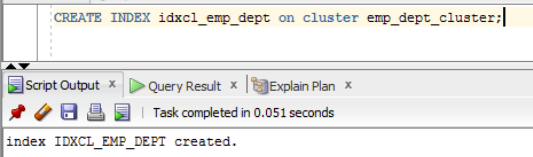
# Index Clustered Tables

## Task result of task 4: Cluster Storage by Blocks

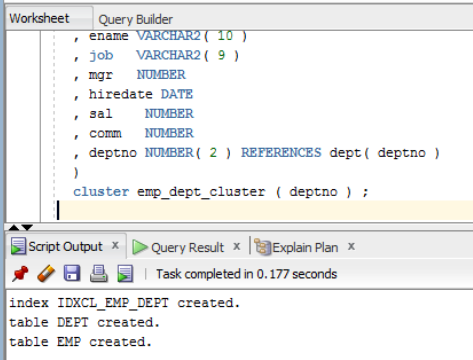
The cluster was created.



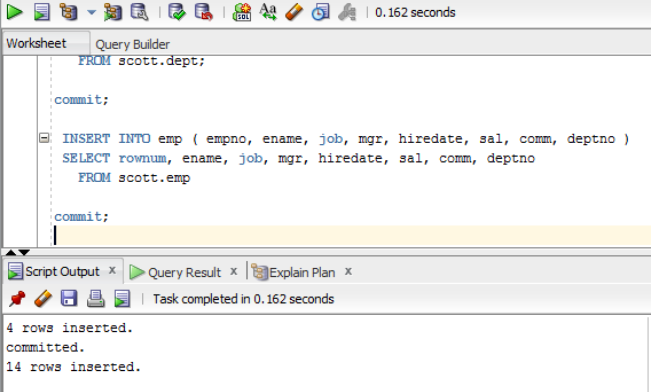
The clustered index was created.

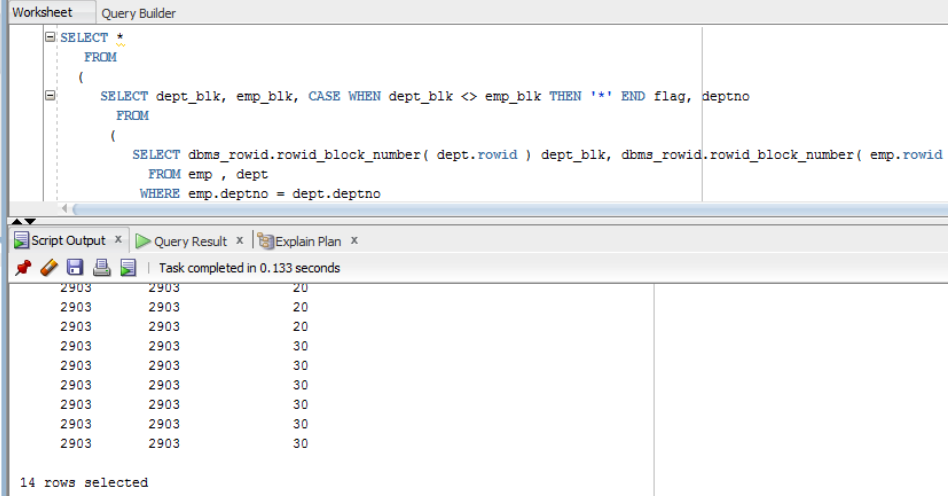


Clusterized tables emp and dept where created.

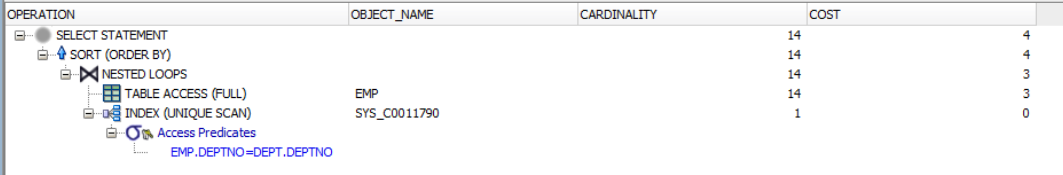


Rows were inserted into tables emp and dept.





Explian plan:

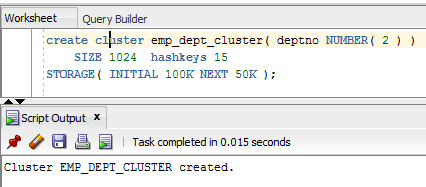


The tables are stored in the same block and belong to the same cluster. Therefore, when referring to the tables, there is a conversion not to each of them individually, but to the entire block, which significantly speeds up the process and reduces the cost.

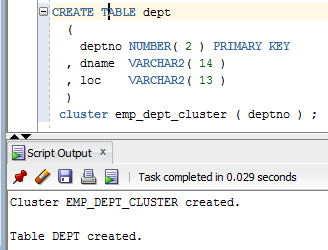
# Hash Clustered Tables

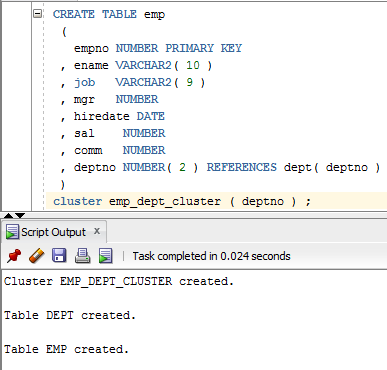
## Task result of task 5: Analyses Cluster Storage by Blocks

I added parameter to the cluster that gives an opportunity to calculate the size of the cache table.

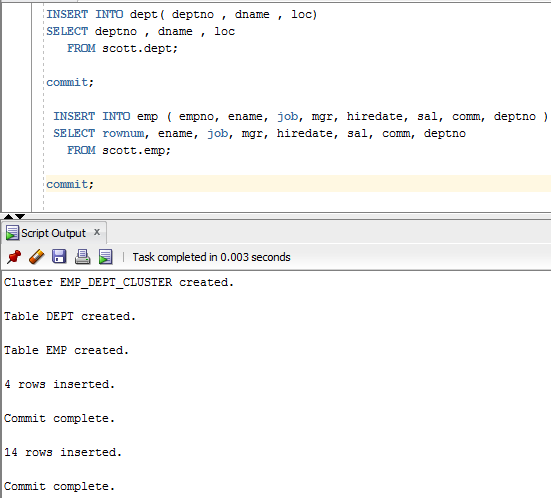


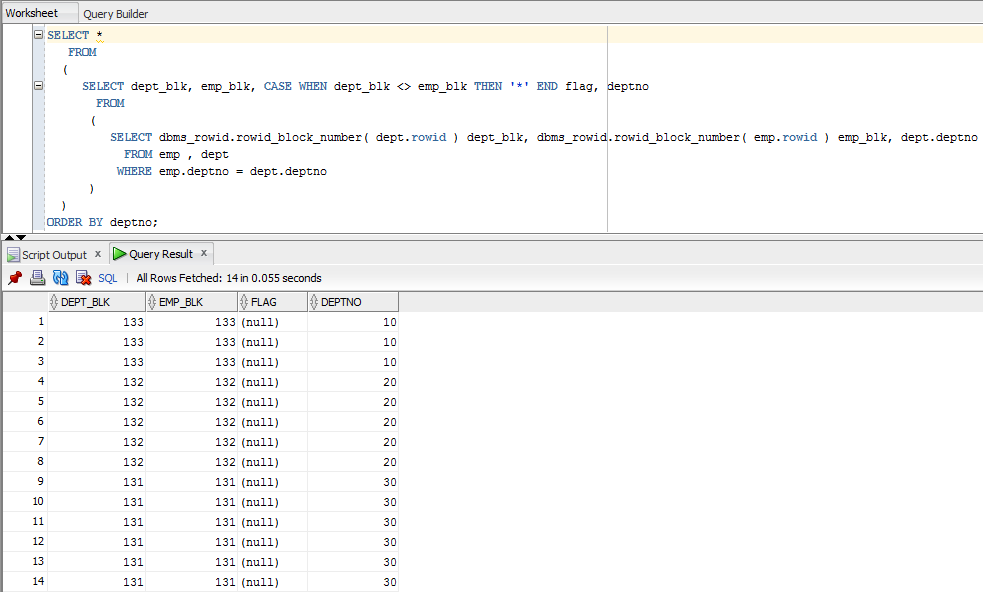
Tables dept and emp where created.



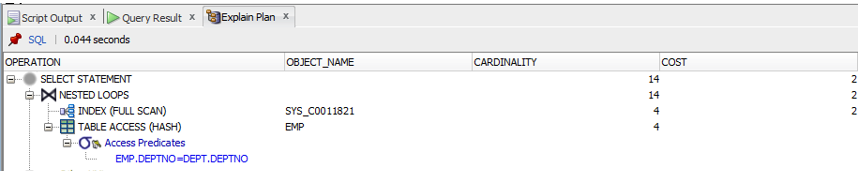


The values were inserted to the tables emp and dept.





Explain plan:



# Row Migration\*

Table row\_mig\_chain\_demo was created.

CREATE TABLE row\_mig\_chain\_demo (

x int PRIMARY KEY,

a CHAR(1000),

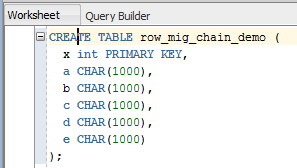
b CHAR(1000),

c CHAR(1000),

d CHAR(1000),

e CHAR(1000)

);

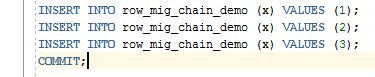


INSERT INTO row\_mig\_chain\_demo (x) VALUES (1);

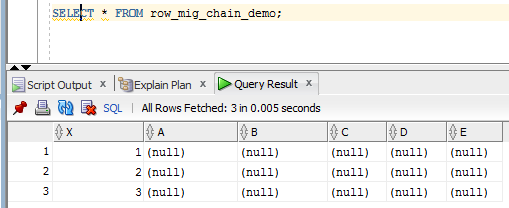
INSERT INTO row\_mig\_chain\_demo (x) VALUES (2);

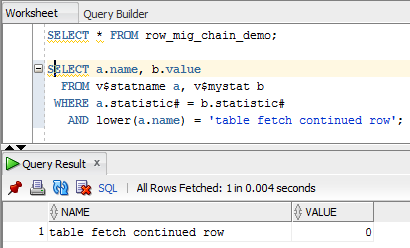
INSERT INTO row\_mig\_chain\_demo (x) VALUES (3);

COMMIT;



SELECT \* FROM row\_mig\_chain\_demo;





“0” means that all value is located on one data block.

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

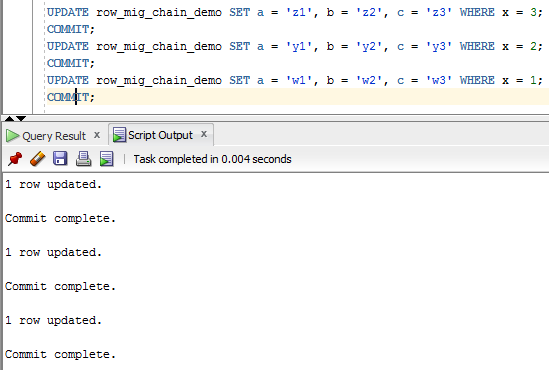
COMMIT;

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

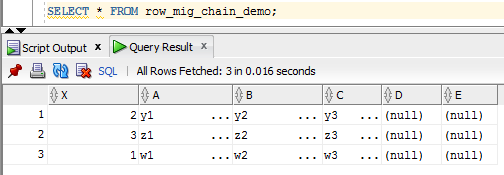
COMMIT;

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

COMMIT;



SELECT \* FROM row\_mig\_chain\_demo;

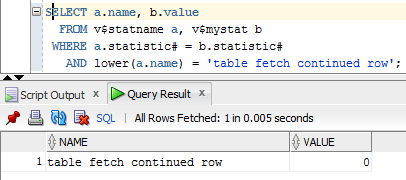


SELECT a.name, b.value

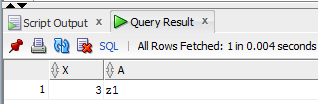
FROM v$statname a, v$mystat b

WHERE a.statistic# = b.statistic#

AND lower(a.name) = 'table fetch continued row';



SELECT \* FROM row\_mig\_chain\_demo WHERE x = 3;

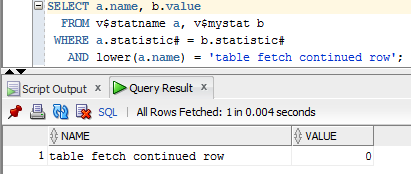


SELECT a.name, b.value

FROM v$statname a, v$mystat b

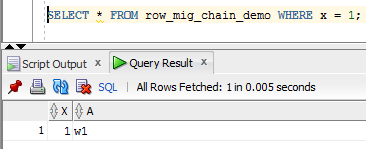
WHERE a.statistic# = b.statistic#

AND lower(a.name) = 'table fetch continued row';



We can’t see the migration because the row stayed at the first block.

SELECT \* FROM row\_mig\_chain\_demo WHERE x = 1;

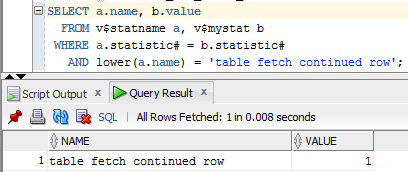


SELECT a.name, b.value

FROM v$statname a, v$mystat b

WHERE a.statistic# = b.statistic#

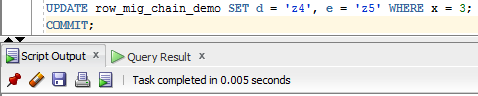
AND lower(a.name) = 'table fetch continued row';

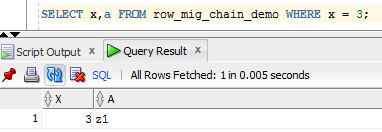


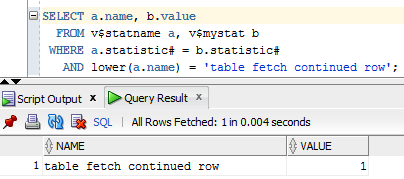
The row relocated from first block to the third.

Row chaining.

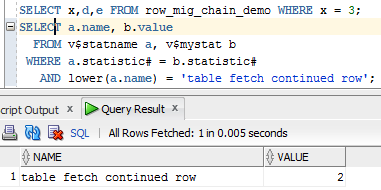
The table row\_mig\_chain\_demo were updated.







There is no migration because these two values is located on the first block.



There is the migration because e relocated to the forth block.