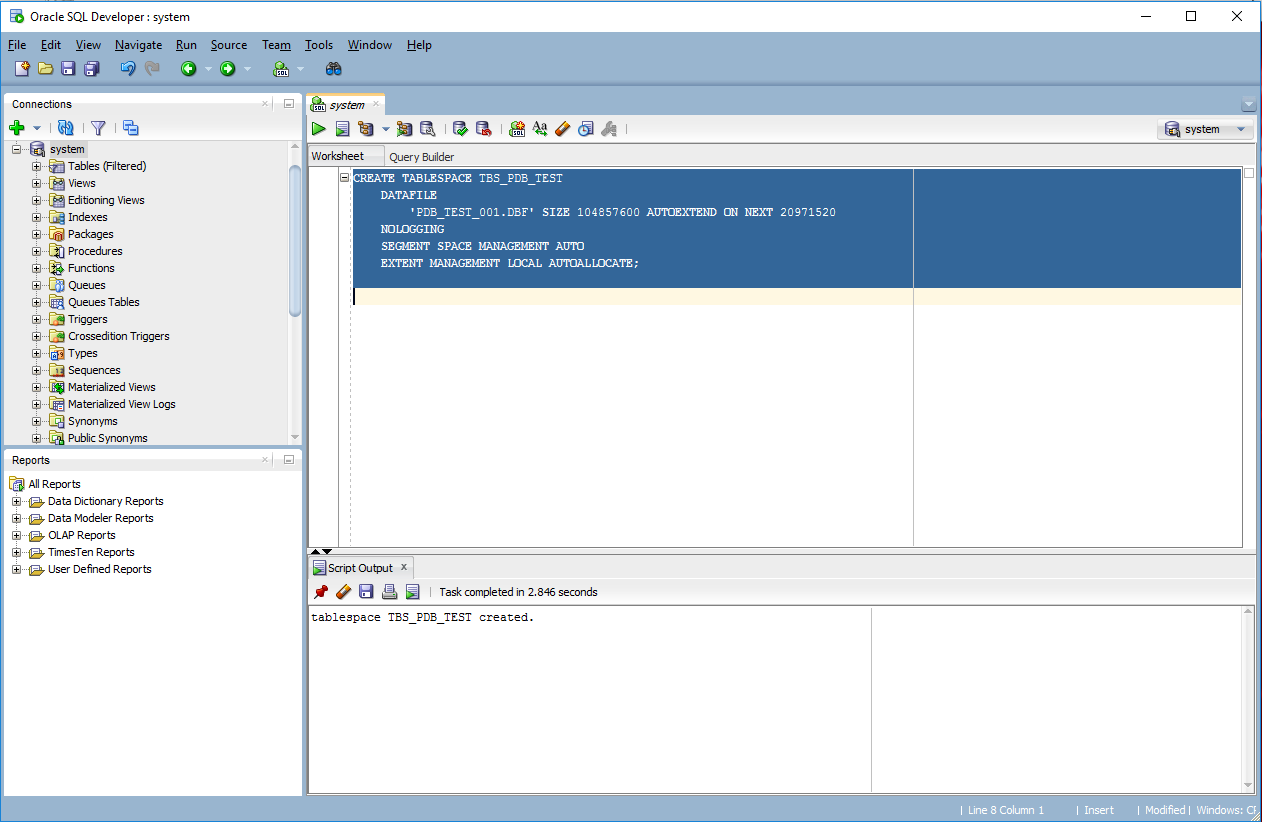
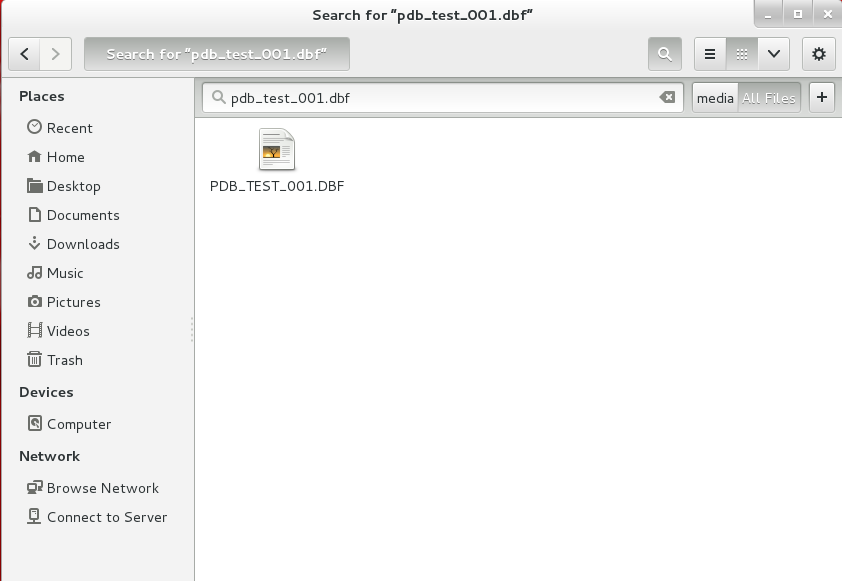
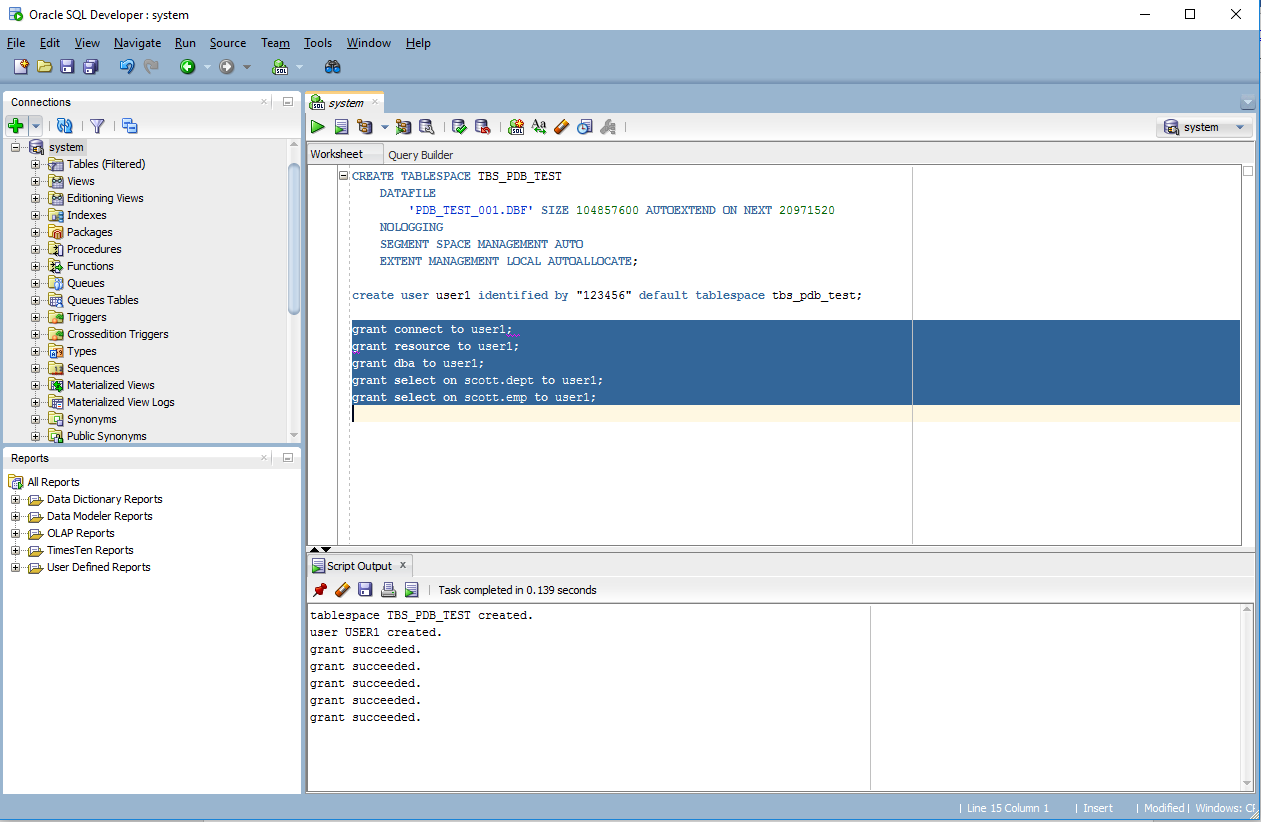
**Task 1**

Connected to system user and created new user – user1

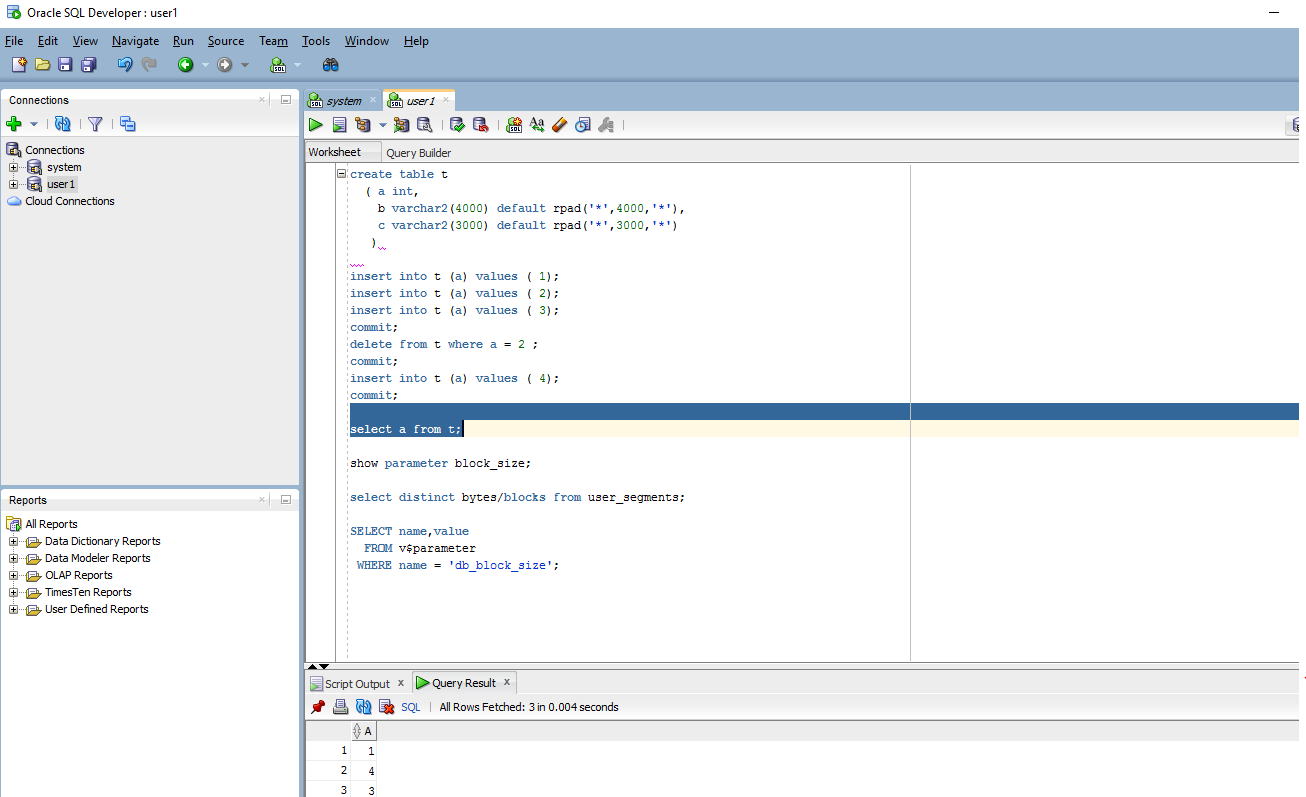


File created and we see it here:

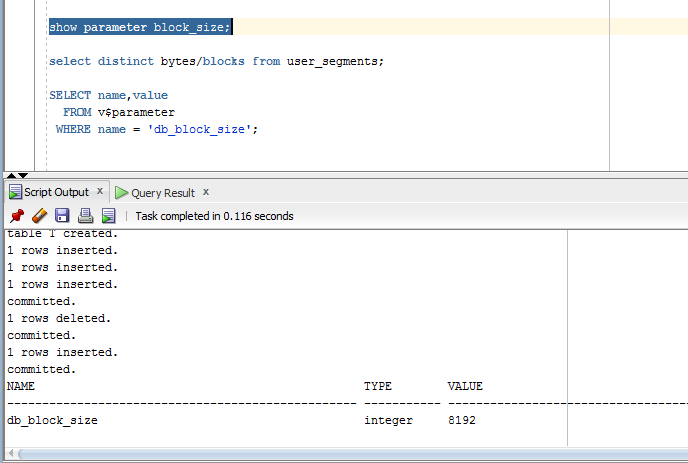


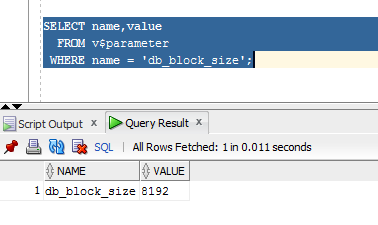


Connected to user1:

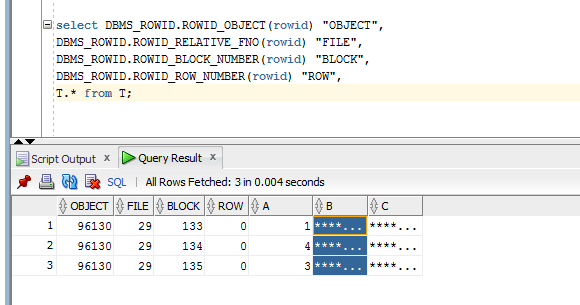


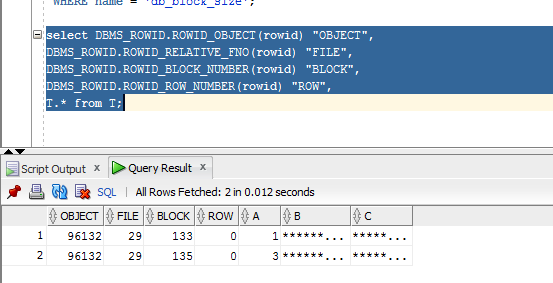
Created table and checked db\_block\_size:

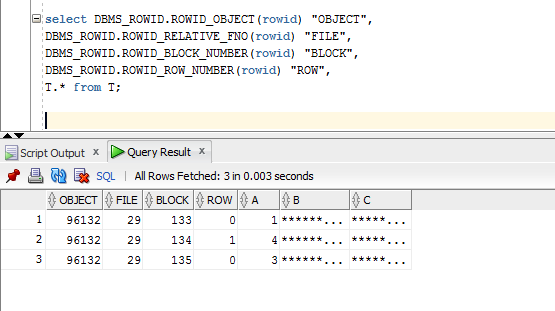


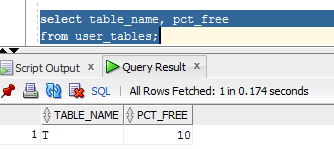


These 3 screens show how heap table works with inserting into different blocks:

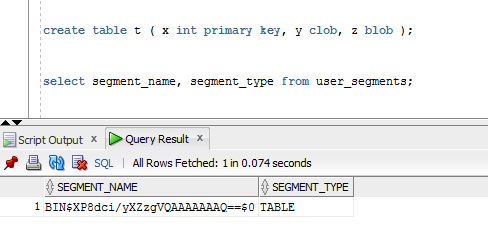


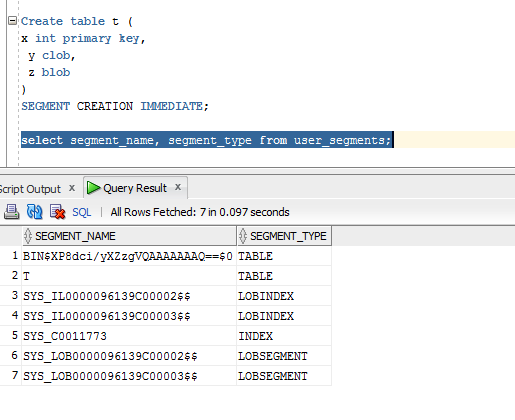




Since our db\_block\_size is 8192 and our columns are 1+3000+4000=7001 and it’s less than 90% (10% used for updates of existing rows by default ), that’s why 1 row will be inserted into one block. And after deleting it will use free block (134 block again).

**Task 2**

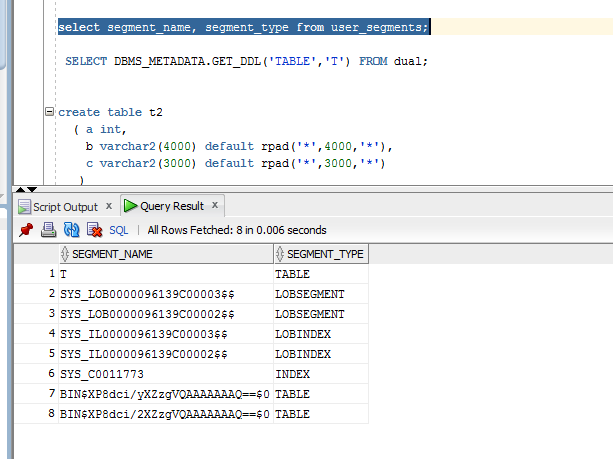


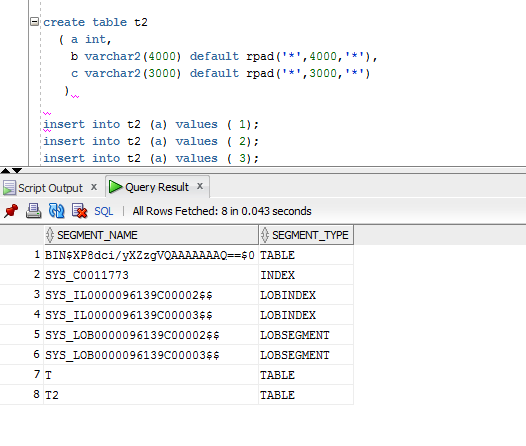


SEGMENT CREATION IMMEDIATE: Cегменты появляются во время создания таблицы. Это - поведение по умолчанию в базах данных Oracle.

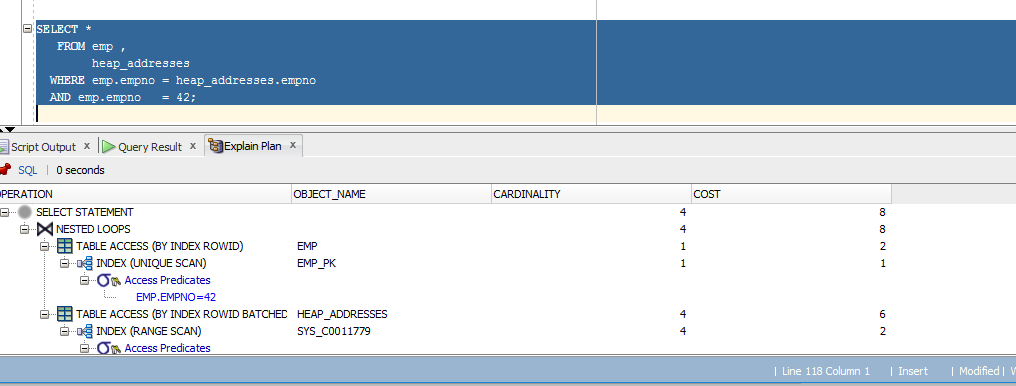
In other way until rows will be inserted the segment will not be created.

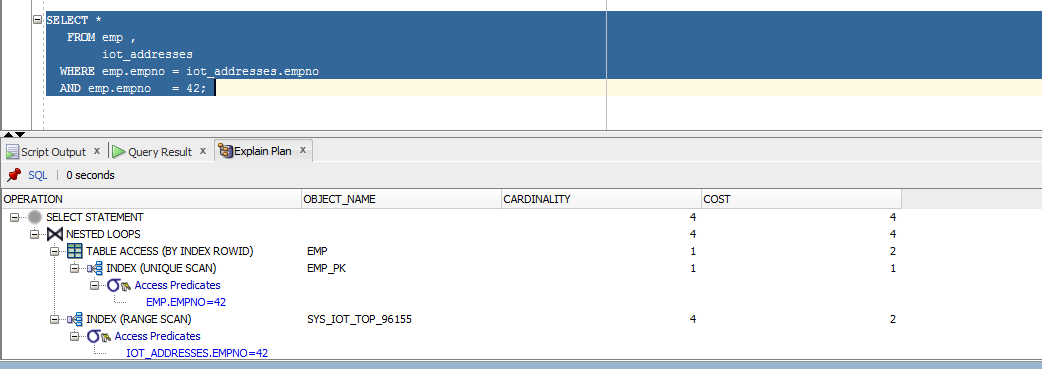
To check this we should try to insert rows in the table:





**Task 3**

This is a comparison of 2 tables. First one is without index and the second one uses (ORGANIZATION INDEX). We see that it faster in 2 times than without index:

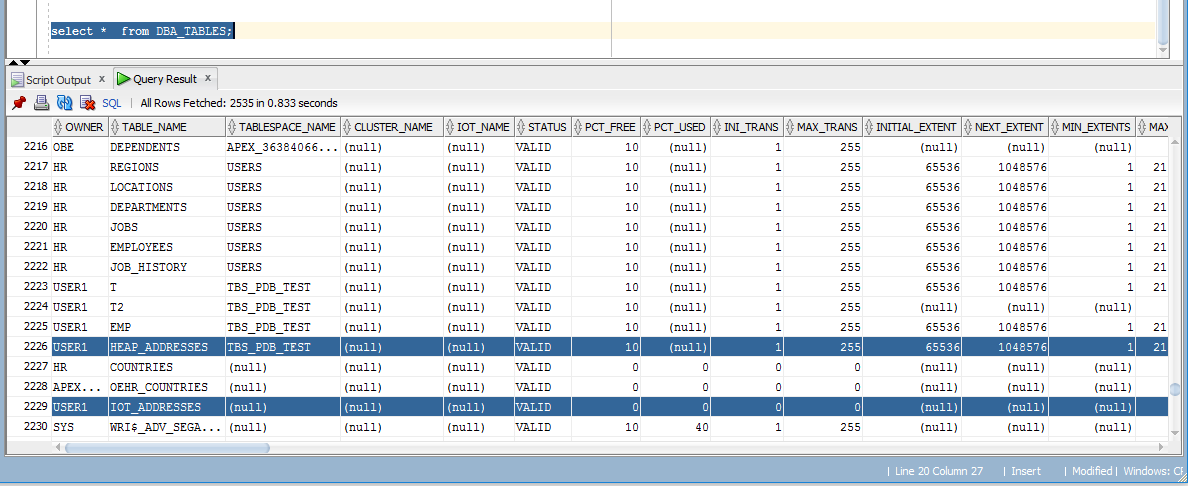


An **index-organized table** has a storage organization that is a variant of a primary B-tree. Unlike an ordinary (heap-organized) table whose data is stored as an unordered collection (heap), data for an index-organized table is stored in a B-tree index structure in a primary key sorted manner. Each leaf block in the index structure stores both the key and nonkey columns.

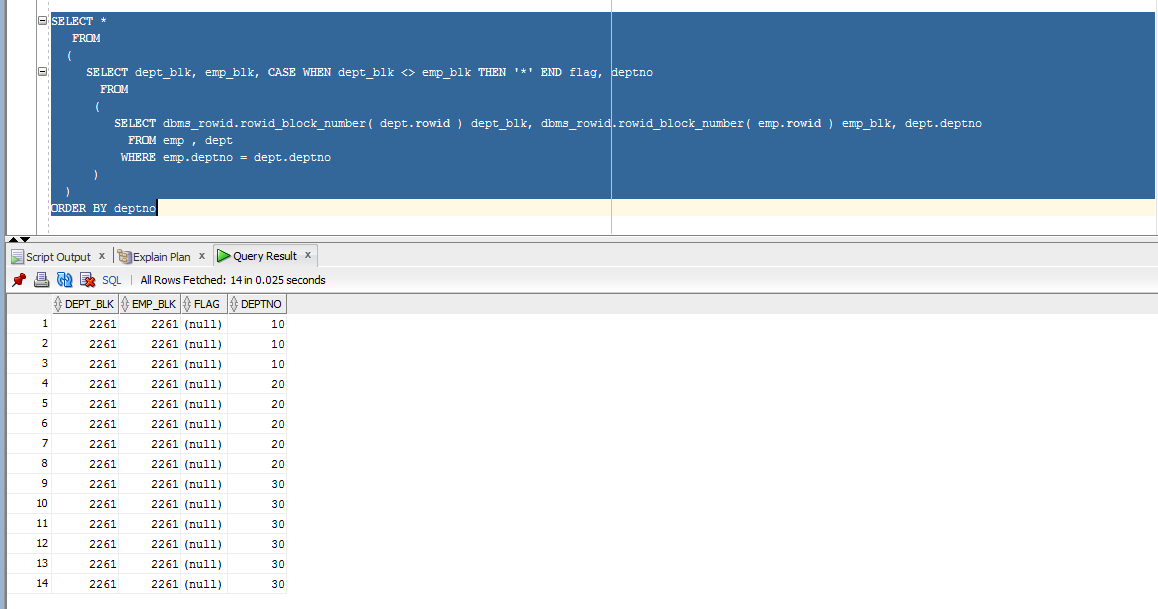
Index-organized tables have full table functionality.

* Fast random access on the primary key because an index-only scan is sufficient. And, because there is no separate table storage area, changes to the table data (such as adding new rows, updating rows, or deleting rows) result only in updating the index structure.
* Fast range access on the primary key because the rows are clustered in primary key order.

Here is some statistics that gathered thought our work:



**Task 4**



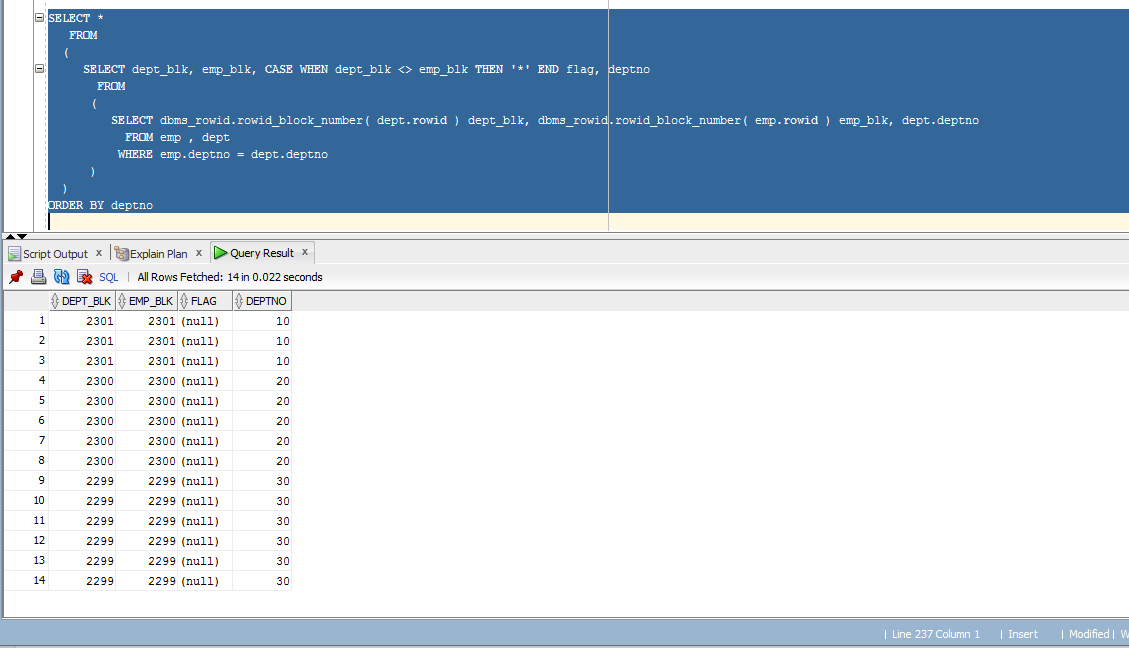
**Here we store data grouped *by index that we created on a cluster***

With clustered tables, data from many tables may be stored on the same block. All data that contains the same cluster key value, such as DEPTNO = 10, will be physically stored together. The data is clustered around the cluster key value.

Clusters are groups of one or more tables, physically stored on the same database blocks, with all rows that share a common cluster key value being stored physically near each other.

**Task 5**

**Here we store data grouped by *hash number of our cluster  
\*also we cannot create index on it because -*** In a hash cluster, the data is the index (metaphorically speaking). (from courseware)

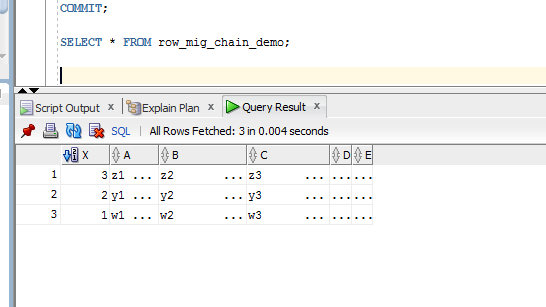
The hash cluster hashes the key to the cluster to arrive at the database block the data should be on. In a hash cluster, the data is the index (metaphorically speaking). These tables are appropriate for data that is read frequently via an equality comparison on the key. With an indexed table or indexed cluster, Oracle Database locates table rows using key values stored in a separate index.

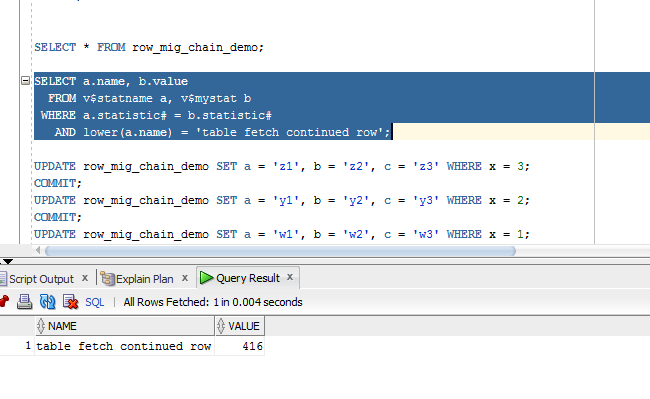
**Task 6**

* *Chained rows* - a chained rows is a LOB row (usually a BLOB, CLOB, RAW or LONG RAW) where the row length is large than the data block size.  Many shops will create a 32k blocksize to store large columns without row chaining.
* *Migrated rows* - Migrated rows occur when an UPDATE DML causes the rows to expand onto another data block.  This can be avoided by setting PCTFREE to a large enough value to accommodate row expansion, and existing migrated rows can be fixed by reorganizing the tables with the dbms\_redefinition utility.

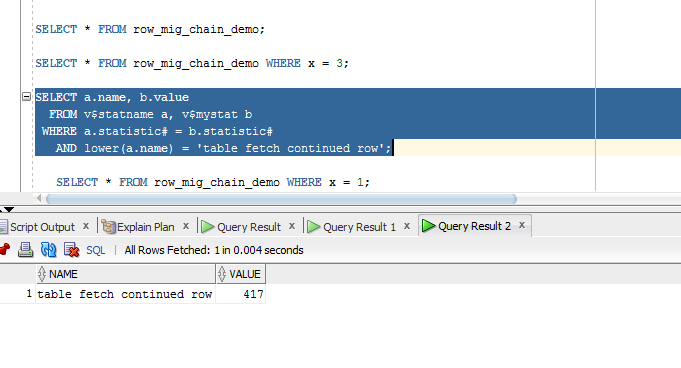
*Row migration* is when a row is forced to leave the block it was created on because it grew too large to fit on that block with the rest of the rows.

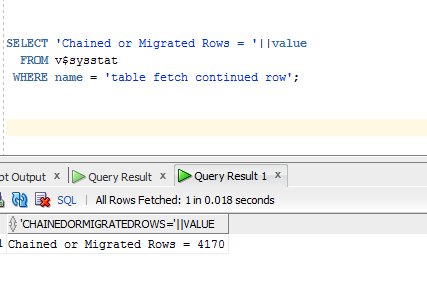
When Oracle migrates the row, it will leave behind a pointer to where the row really is.Here I tried to process row migration and row chaining:

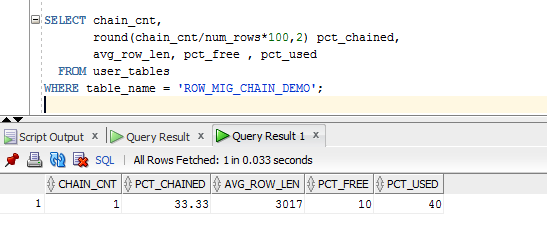




Row is migrated, using the primary key index, we forced a «table fetch continued row» +1.







I’m not sure that I have a good understanding of the whole process that I’ve done about migration, but at least I’ve tried. It’s better to cover it on a lecture again.