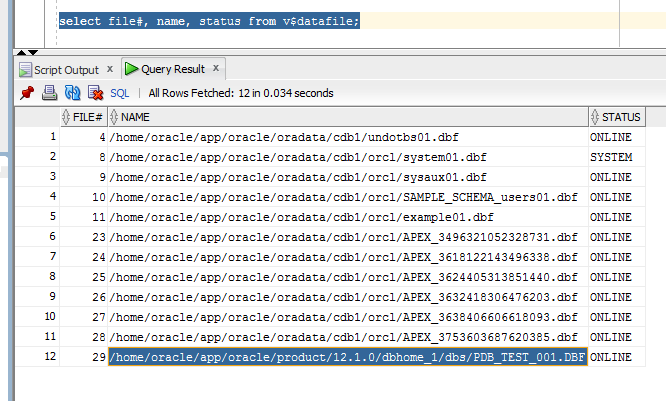
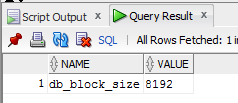
# Prerequisite Task

# 



# Heap Organized Tables

SELECT name,value FROM v$parameter WHERE name = 'db\_block\_size';



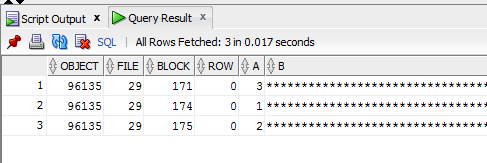
select dbms\_rowid.rowid\_object(rowid) "OBJECT",

dbms\_rowid.rowid\_relative\_fno(rowid) "FILE",

dbms\_rowid.rowid\_block\_number(rowid) "BLOCK",

dbms\_rowid.rowid\_row\_number(rowid) "ROW",

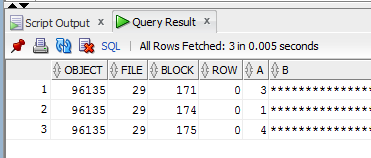
t.\* from t;



После вставки строки:

delete from t where a = 2 ;

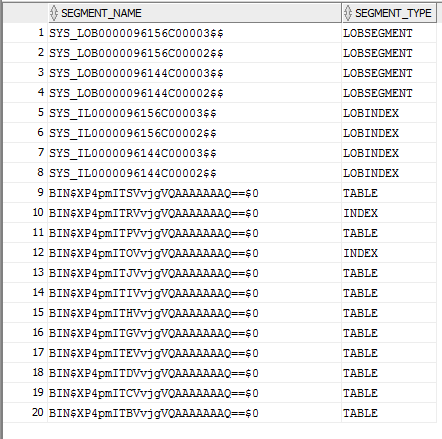
insert into t (a) values ( 4);



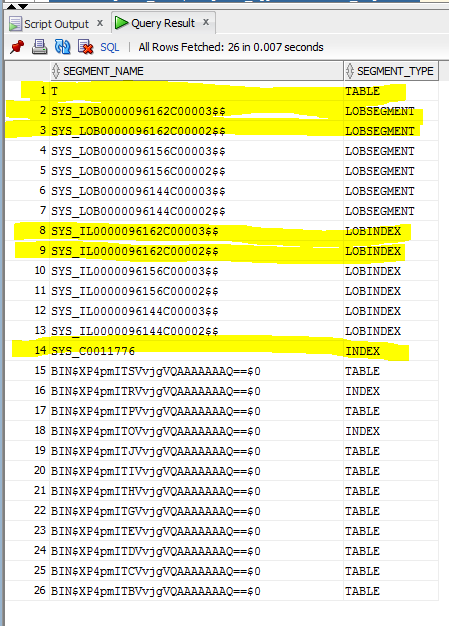
Выводы: В чем суть heap table - в скорости вставки данных, Oracle, где находит первое свободное место, которое может уместить вставляемые данные, туда и производит вставку. При удалении данных из heap таблицы пространство становится доступным для повторного применения.

## Task 2 – Understanding Heap Table Segments

select segment\_name, segment\_type from user\_segments;

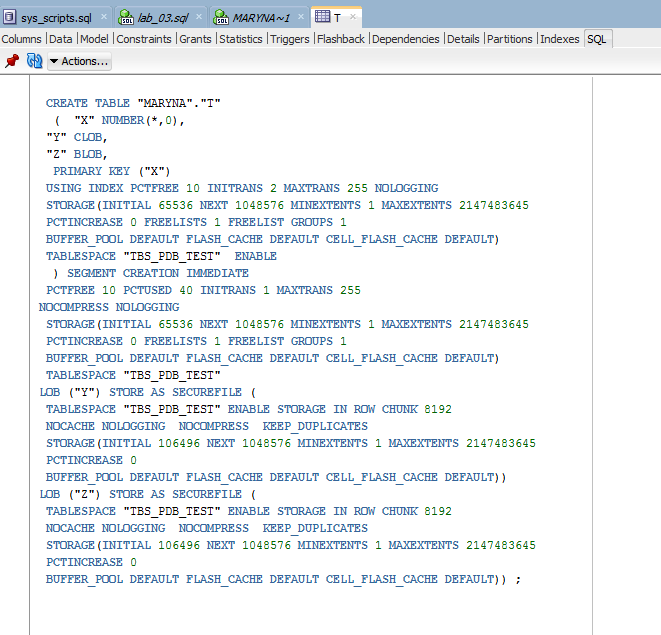


create table t ( x int primary key, y clob, z blob) segment creation immediate;



select dbms\_metadata.get\_ddl('TABLE','T') from dual;

Результат равен коду ddl, который также можно посмотреть в свойствах таблицы:



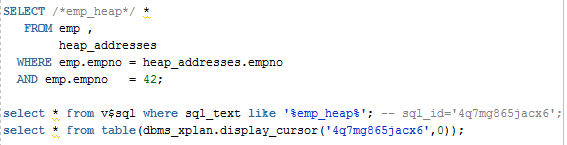
Выводы:

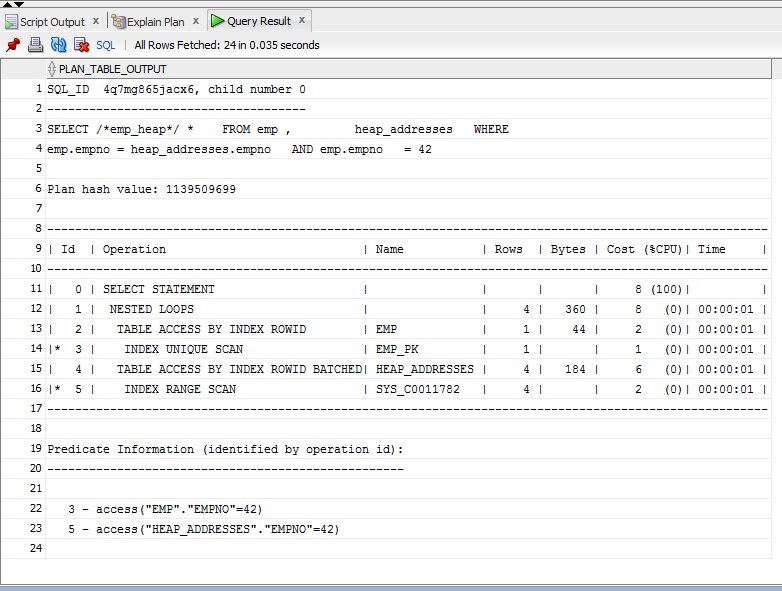
Таблицам и индексам выделяются отдельные сегменты. При создании таблицы со столбцом clob/blob для каждого создается LOBSEGMENT и LOBINDEX. Информация LOB-столбцов (а обычно это очень большие объемы данных) хранятся порциями в LOB-сегменте, а LOB-индекс применяется для отслеживания их местоположения и упорядочения доступа в ним. В Oracle 12С при сreate table по умолчанию создание сегментов откладывается до вставки первой строки, для того чтобы немедленно создать сегменты нужно применить конструкцию – segment creation immediate .

# Index Organized Tables

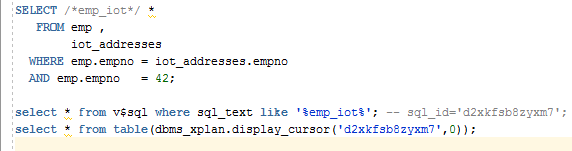
## Task 3: Compare performance

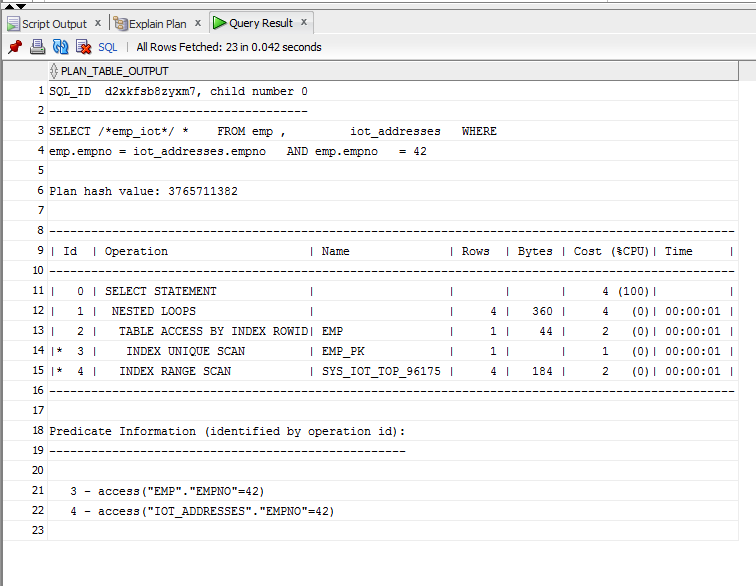
1. Таблица в виде кучи:





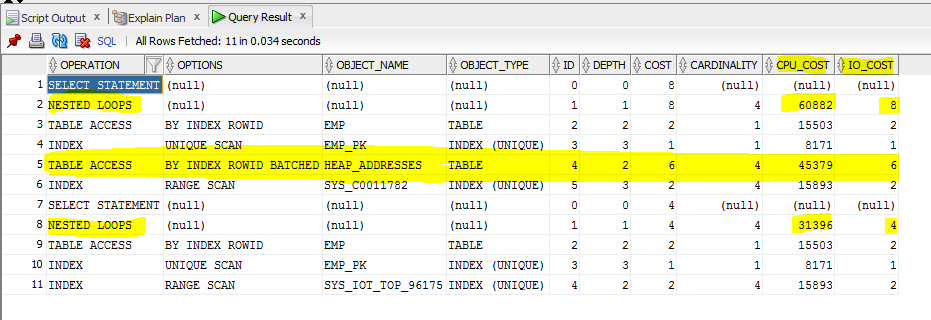
1. IOT – таблица:





Сравнение:





В случае IOT таблицы отсутствует обращение по rowid к данным таблицы, потому что IOT таблица сама содержит эти данные.

Выводы:

IOT таблицы хранятся в индексной структуре, т.е. данные сохраняются в сортированном порядке согласно первичному ключу. И поскольку Oracle не требуется идти из индекса в таблицу, то при использовании IOT таблицы значительно ускоряется доступ к данным. По сути индекс-таблицы ведут себя как “обычные” таблицы, для доступа к ним применяется язык SQL. Получается это гибрид индекса и таблицы, эквивалентен логике работы покрывающего индекса.

# Index Clustered Tables

select \* from

( select dept\_blk, emp\_blk, case when dept\_blk <> emp\_blk then '\*' end flag, deptno

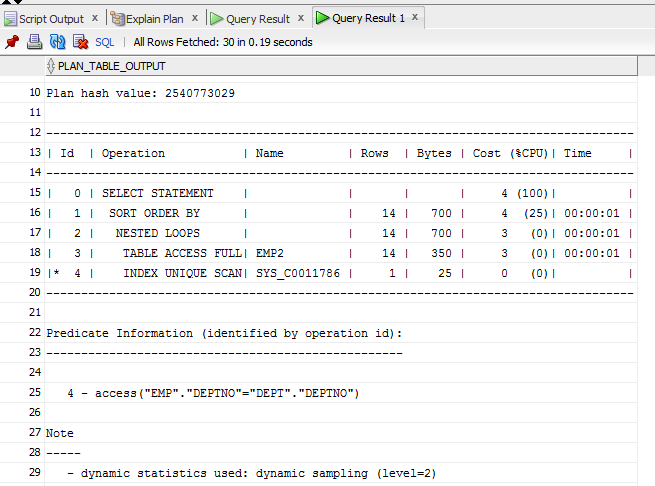
from

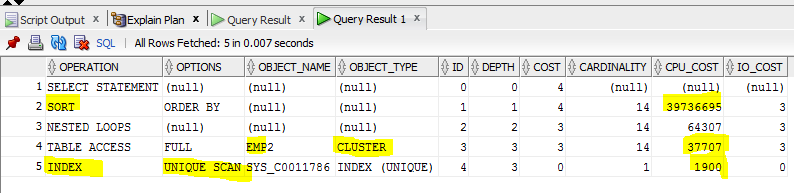
( select dbms\_rowid.rowid\_block\_number( dept.rowid ) dept\_blk, dbms\_rowid.rowid\_block\_number( emp.rowid ) emp\_blk, dept.deptno

from emp2 emp, dept

where emp.deptno = dept.deptno))

order by deptno;

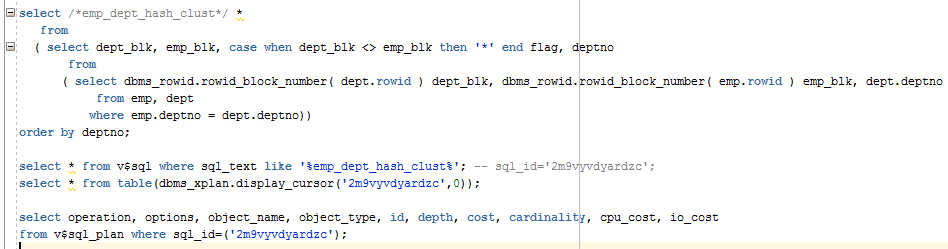


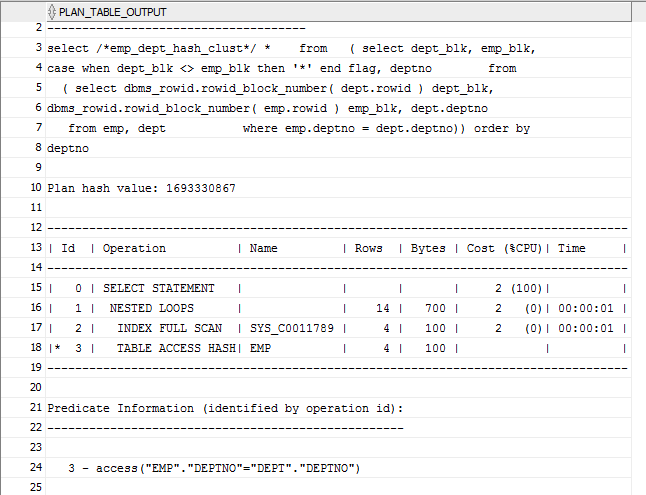


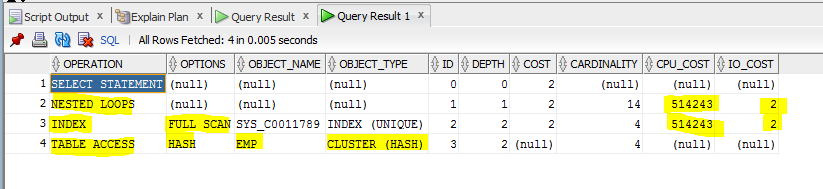
Выводы: кластеризованные индекс-таблицы одного кластера физически хранятся в тех же самых блоках БД, при этом строки этих таблиц имеют общее значение кластерного ключа и физически находятся рядом с друг другом. Т.е. все данные, содержащие одно и то же значение кластерного ключа, одинаковое deptno в нашем случае, будут храниться физически вместе. Именно поэтому план отобразил, что нашему запросу потребовалось прочитать только блоки emp типа кластер. Из этого можно сделать вывод, что кластеризованные индекс-таблицы сокращают количество блоков, считываемых с жесткого диска, тем самым ускоряя производительность.

# Hash Clustered Tables







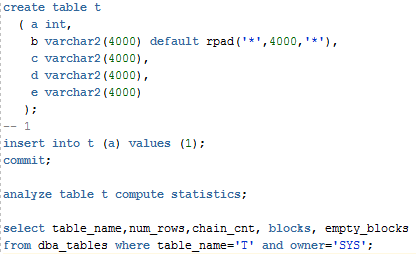


Выводы: Кластеризованные хеш-таблицы – это братья-близнецы с кластеризованными индекс-таблицами. Отличие лишь в том, но вместо использования индекса со структурой B-дерева для определения местонахождения данных по кластерному ключу хеш-кластер хеширует кластерный ключ, чтобы попасть в блок БД, где должны располагаться данные.

# Row Migration\*

Row chaining – когда вставляемые строки заведомо больше размера блока.

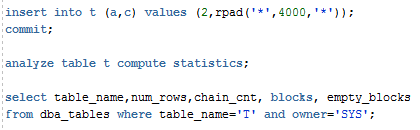
Для этого я создаю таблицу с необходимыми мне для анализа размерами столбцов, по дефолту устанавливаю только b, по мере необходимости буду создавать сама.



Вставляю строку, которая меньше размера блока, собираю статистику и получаю результат. При моей таблице есть 1 строка, содержащаяся в 1 блоке.



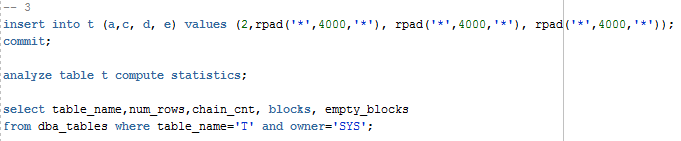
Далее я вставляю строку, которая заведомо не вмещается в один блок (Напомню, что поле b – заполняется дефолтно).



После сбора статистики вижу, что у меня 2 строки, одна из которых перемещенная, и они занимают уже 3 блока.



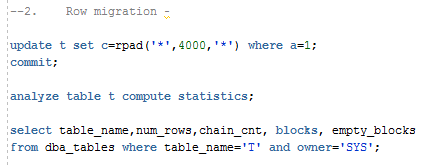
Идем дальше, вставляю строку, которая по размеру не вмещается уже в 2 блока.



После сбора статистики вижу, что у меня 3 строки, две из которых перемещенные, и они занимают уже 7 блоков.



Дальше приведу пример row migration. Я сделала апдейт первой строки, которая была вставлена самой первой, и хранится в 1 блоке. Сделала ее размер больше размера блока и собрала статистику.



Видно, что строк осталось 3, но количество перемещенных увеличилось на 1, и количество блоков увеличилось до 12.



Выводы:

В Oracle существует перемещение строк (row migration) - это ситуации, либо когда строка вынуждена покинуть блок, в котором создавалась, поскольку она стала слишком большой, либо когда строка была изначально слишком большой. При чтении перемещенной строки индекс будет указывать на исходный блок, а тот в свою очередь на новый, т.е. придется выполнить дополнительную операцию ввода-вывода. Поэтому это не есть хорошо.