|  |
| --- |
|  |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial status | Valeryia\_Lupanava | 02-OCT-2017 |  |  |

Contents

[1. Prerequisite Task 3](#_Toc497410702)

[2. Heap Organized Tables 5](#_Toc497410703)

[2.1. Task results of task 1 – Heap Understanding 5](#_Toc497410704)

[2.2. Task Results of task 2 – Understanding Heap Table Segments 10](#_Toc497410705)

[3. Index Organized Tables 14](#_Toc497410706)

[3.1. Task results of task 3: Compare performance 14](#_Toc497410707)

[4. Index Clustered Tables 21](#_Toc497410708)

[4.1. Task result of task 4: Cluster Storage by Blocks 21](#_Toc497410709)

[5. Hash Clustered Tables 26](#_Toc497410710)

[5.1. Task result of task 5: Analyses Cluster Storage by Blocks 26](#_Toc497410711)

[6. Row Migration\* 31](#_Toc497410712)

[6.1. Task results 31](#_Toc497410713)

# Prerequisite Task

Connect as system to pluggable database pdb. Create a new tablespace tbs\_pdb\_test with the new datafile pdb\_test\_001.dbf:

CREATE TABLESPACE TBS\_PDB\_TEST

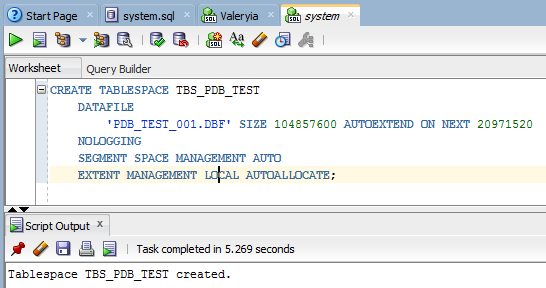
DATAFILE

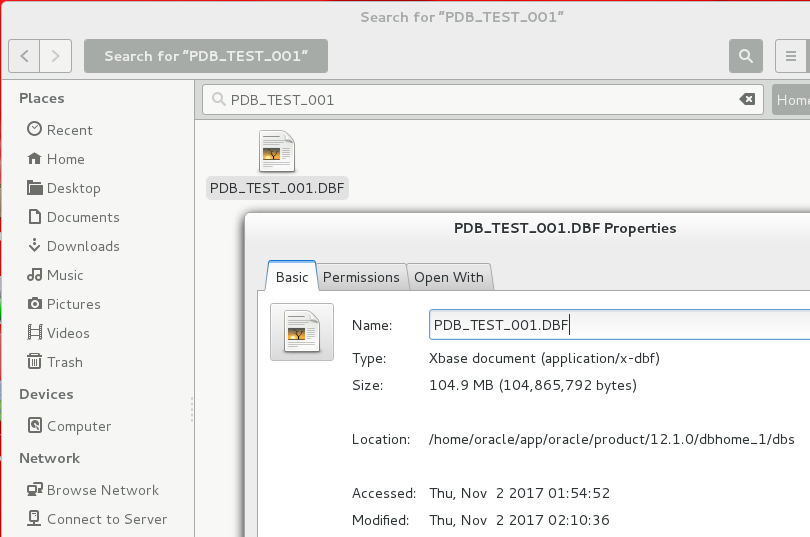
'PDB\_TEST\_001.DBF' SIZE 104857600 AUTOEXTEND ON NEXT 20971520

NOLOGGING

SEGMENT SPACE MANAGEMENT AUTO

EXTENT MANAGEMENT LOCAL AUTOALLOCATE;





Create a new user:

create user $username$ identified by $pwd$ default tablespace tbs\_pdb\_test;





Grant Connect Role and Resource Role:

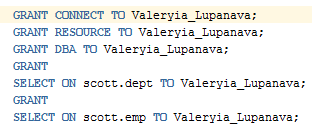
grant connect to $username$;

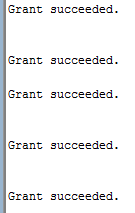
grant resource to $username$;

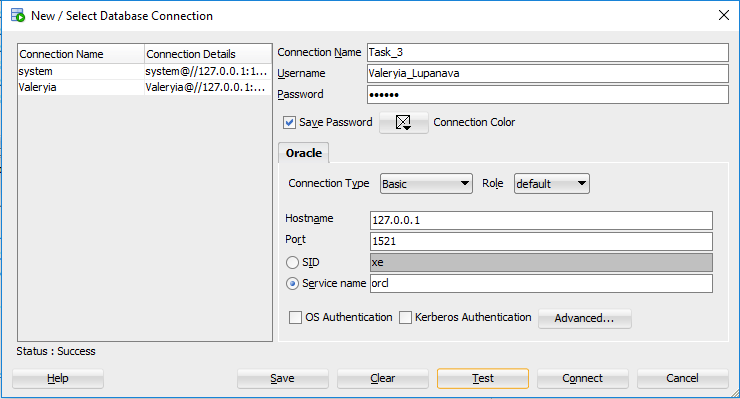
grant dba to $username$;

grant select on scott.dept to $username$;

grant select on scott.emp to $username$;







# Heap Organized Tables

## Task results of task 1 – Heap Understanding

Step 1:

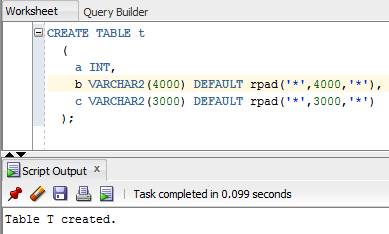
create table t

( a int,

b varchar2(4000) default rpad('\*',4000,'\*'),

c varchar2(3000) default rpad('\*',3000,'\*')

);



**Создали таблицу с тремя колонками. В первой ПК, в остальных символы «\*». По дефолту во вторую и третью колонки записываюся «\*» в количестве 4000 и 3000 тысячи символов соответственно.**

Step 2:

insert into t (a) values ( 1);

insert into t (a) values ( 2);

insert into t (a) values ( 3);

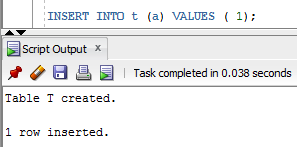
commit;

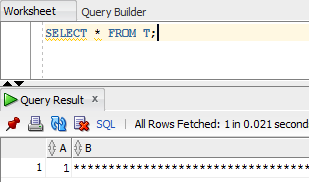
delete from t where a = 2 ;

commit;

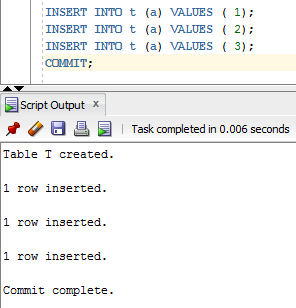
insert into t (a) values ( 4);

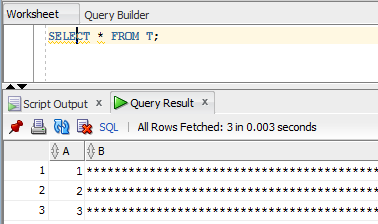
commit;



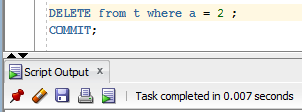


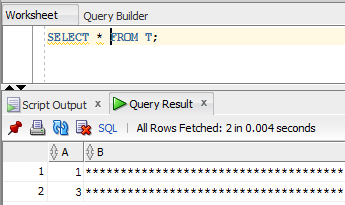
**Затем вставили в таблицу первую строку с ПК, равным 1 и проверили значения.**



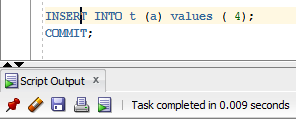


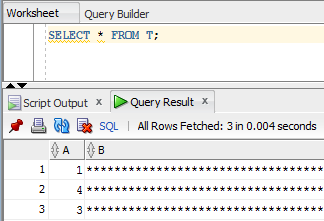
**Затем вставили в таблицу еще две строки с ПК, равными 2 и 3 соответственно и проверили значения. Выполнили коммит, чтобы изменения сохранились.**





**Затем удалили в таблице строку с ПК, равным 2 и проверили значения. Выполнили коммит, чтобы изменения сохранились.**

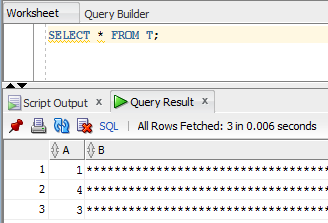




**Затем вставили в таблице строку с ПК, равным 4 и проверили значения.**

Step 3:

select a from t;

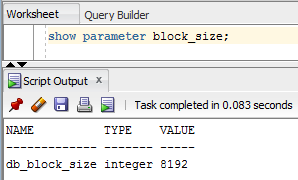


**После проверки правильности результата выполнили коммит, чтобы изменения сохранились.**

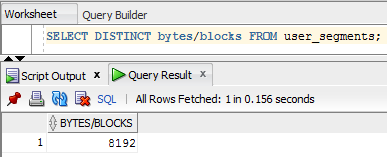
**NOTE:** Adjust columns B and C to be appropriate for your block size if you would like to reproduce this. For example, if you have a 2KB block size, you do not need column C, and column B should be a VARCHAR2(1500) with a default of 1,500 asterisks. Since data is managed in a heap in a table like this,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

**Проверим размер блока тремя способами. Как видим ниже, все показывают одинаковый результат. Размер блока составляет 8192.**

show parameter block\_size;



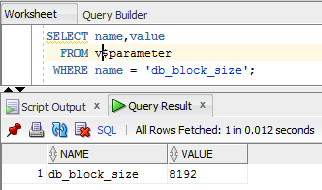
select distinct bytes/blocks from user\_segments;



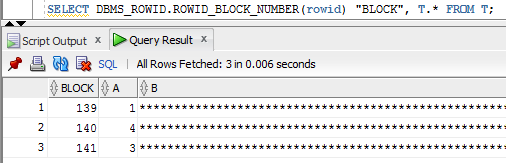
SELECT name,value

FROM v$parameter

WHERE name = 'db\_block\_size';

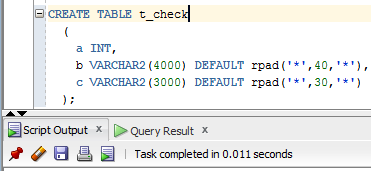


**Проверим, в какие блоки записались наши строки.**

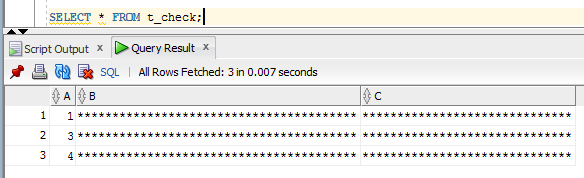


**Можно увидеть, что одна строка на один блок. Поскольку у нас размер блока 8192, а размер строки 7004, что составляет 85%, то в один блок может войти только одна строка. Остальная память остается на PCTFree.**

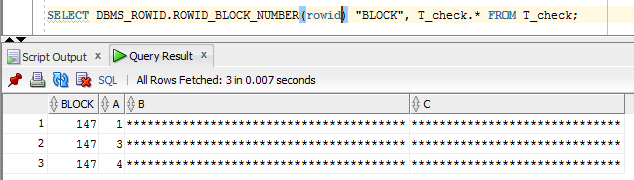
**Теперь проверим, что будет, если уменьшить размер строк.**



**Сделаем аналогичные вставки данных и проверем результат.**



**Проверим результат записи по блокам.**

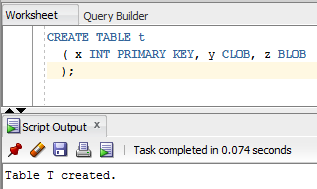


**Видно, что все записалось в один блок, поскольку размер блока достаточно велик для одной строки. Поэтому можно уменьшить размер блока либо увеличить количество символов в строках таблицы.**

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

Step 1:

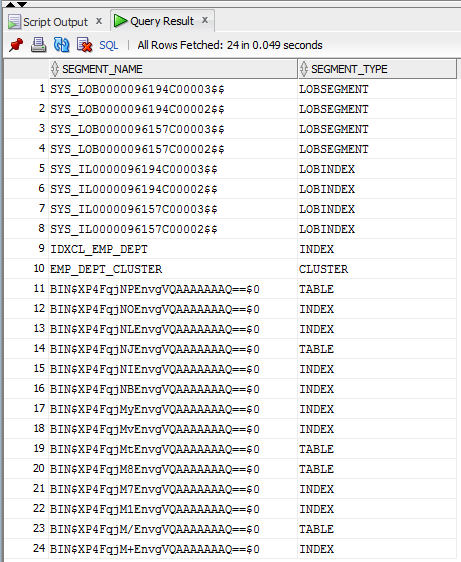
create table t ( x int primary key, y clob, z blob );



**Первым шагом мы создали таблицу.**

Step 2:

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



**Вторым шагом мы посмотрели, выделенные сегменты. Сегмента на индекс и на таблицу отдельно нет.**

Step 3:

Create table t (

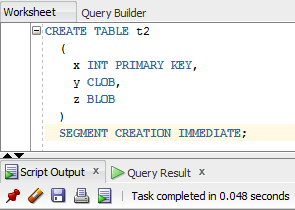
x int primary key,

y clob,

z blob

)

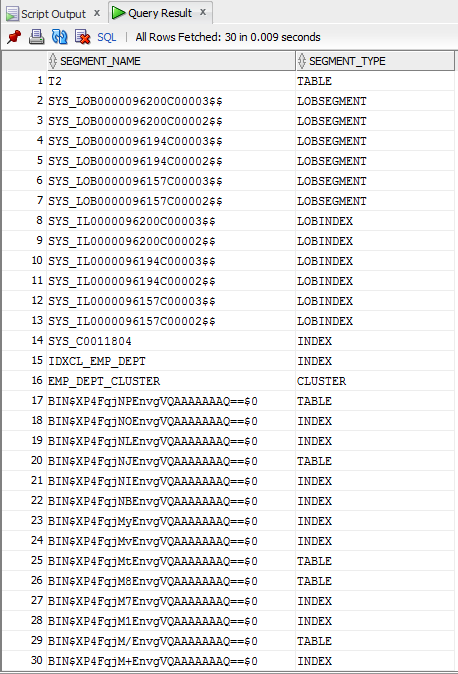
SEGMENT CREATION IMMEDIATE;



**Третьим шагом при создании таблицы пропишем команду по автоматическому выделению сегментов нашей таблице.**

Step 4:

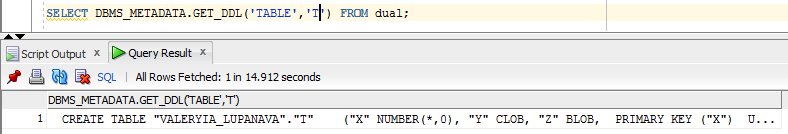
# select segment\_name, segment\_type 2 from user\_segments;

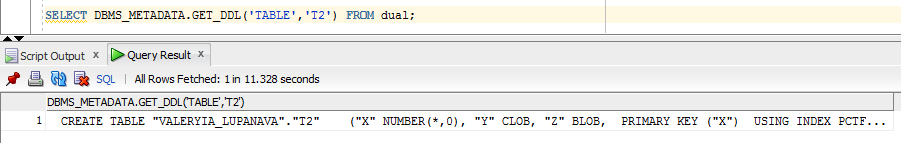


**Теперь можно увидеть, что отдельно на каждый логический объект таблицы выделился сегмент. А именно, на индекс SYS\_C0011804, на таблицу T2.**

Step 5:

# SELECT DBMS\_METADATA.GET\_DDL('TABLE','T') FROM dual





**С помощью данной команды мы можем посмотреть метаданные о необходимом объекте DDL. В данном случае мы посмотрели информацию о только что созданной таблице.**

# Index Organized Tables

## Task results of task 3: Compare performance

Step 1:

CREATE TABLE emp AS

SELECT

object\_id empno

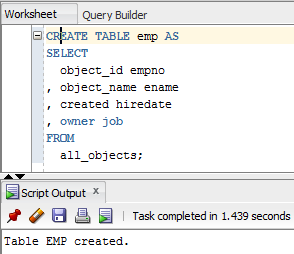
, object\_name ename

, created hiredate

, owner job

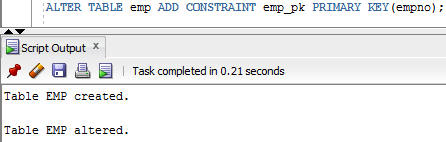
FROM

all\_objects;



Create Index:

alter table emp add constraint emp\_pk primary key(empno)

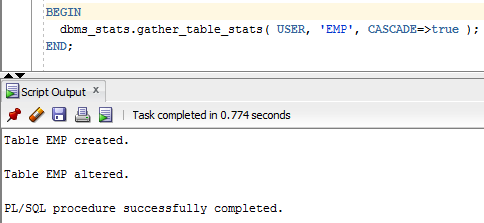


Calculate Statistic:

begin

dbms\_stats.gather\_table\_stats( user, 'EMP', cascade=>true );

end;



**Первым шагом создали таблицу и индекс по ПК. Собрали статистику.**

Step 2:

CREATE TABLE heap\_addresses

(

empno REFERENCES emp(empno) ON DELETE CASCADE

, addr\_type VARCHAR2(10)

, street VARCHAR2(20)

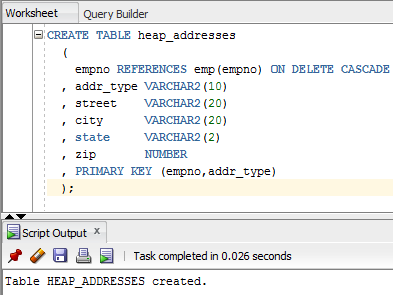
, city VARCHAR2(20)

, state VARCHAR2(2)

, zip NUMBER

, PRIMARY KEY (empno,addr\_type)

);



**Вторым шагом создали таблицу с составным ПК.**

Step 3:

CREATE TABLE iot\_addresses

(

empno REFERENCES emp(empno) ON DELETE CASCADE

, addr\_type VARCHAR2(10)

, street VARCHAR2(20)

, city VARCHAR2(20)

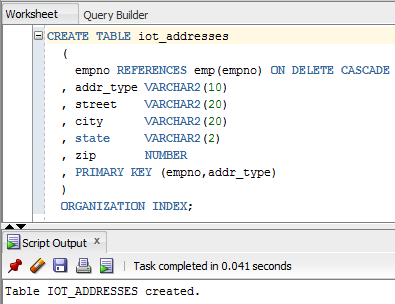
, state VARCHAR2(2)

, zip NUMBER

, PRIMARY KEY (empno,addr\_type)

)

ORGANIZATION INDEX;



**Третьим шагом создали индексно-организованных таблицу. После создания таблицы создается индекс. Когда записи вставляются, то они помещаются в первый доступный блок, который найдет оракл, не придерживаясь никакого определенного порядка. В индексно-организованных таблицах записи размещаются в порядке определенном первичным ключом, который мы установили.**

Step 4: Initial inserts:

INSERT INTO heap\_addresses

SELECT empno, 'WORK' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

INSERT INTO iot\_addresses

SELECT empno , 'WORK' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

--

INSERT INTO heap\_addresses

SELECT empno, 'HOME' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

INSERT INTO iot\_addresses

SELECT empno, 'HOME' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

--

INSERT INTO heap\_addresses

SELECT empno, 'PREV' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

INSERT INTO iot\_addresses

SELECT empno, 'PREV' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

--

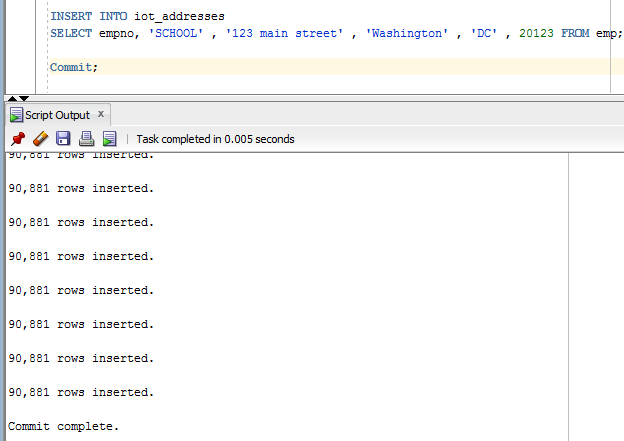
INSERT INTO heap\_addresses

SELECT empno, 'SCHOOL' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

INSERT INTO iot\_addresses

SELECT empno, 'SCHOOL' , '123 main street' , 'Washington' , 'DC' , 20123 FROM emp;

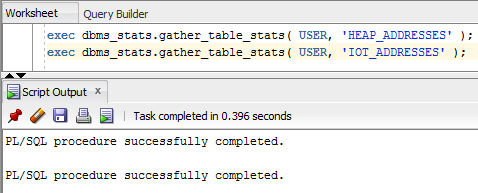
Commit;



Step 5: Calculate statistic:

exec dbms\_stats.gather\_table\_stats( $username$, 'HEAP\_ADDRESSES' );

exec dbms\_stats.gather\_table\_stats( $username$, 'IOT\_ADDRESSES' );



**Вставили данные в таблицы и собрали статистику.**

Step 6: Compare Trace and Performance:

Explain 1:

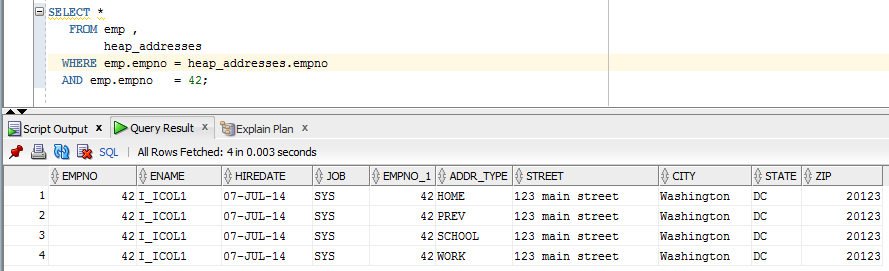
SELECT \*

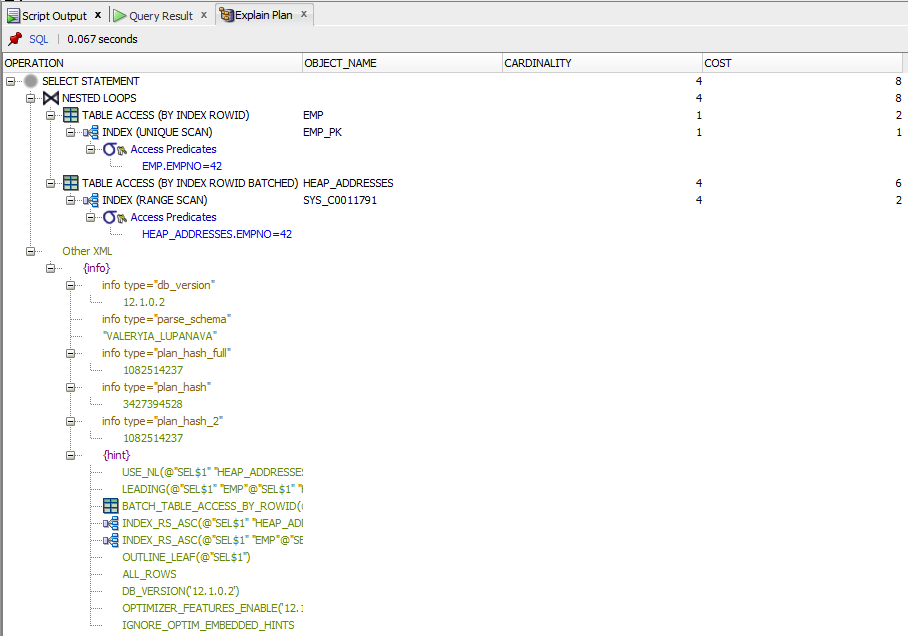
FROM emp ,

heap\_addresses

WHERE emp.empno = heap\_addresses.empno

AND emp.empno = 42;





Explain 2:

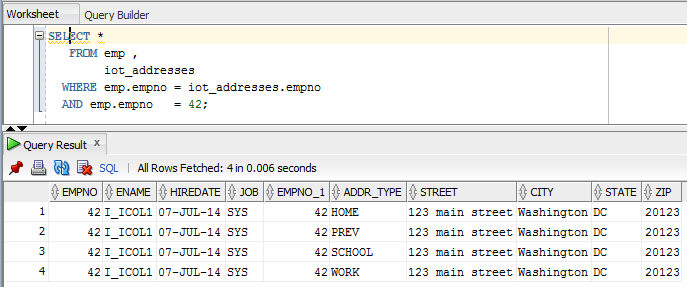
SELECT \*

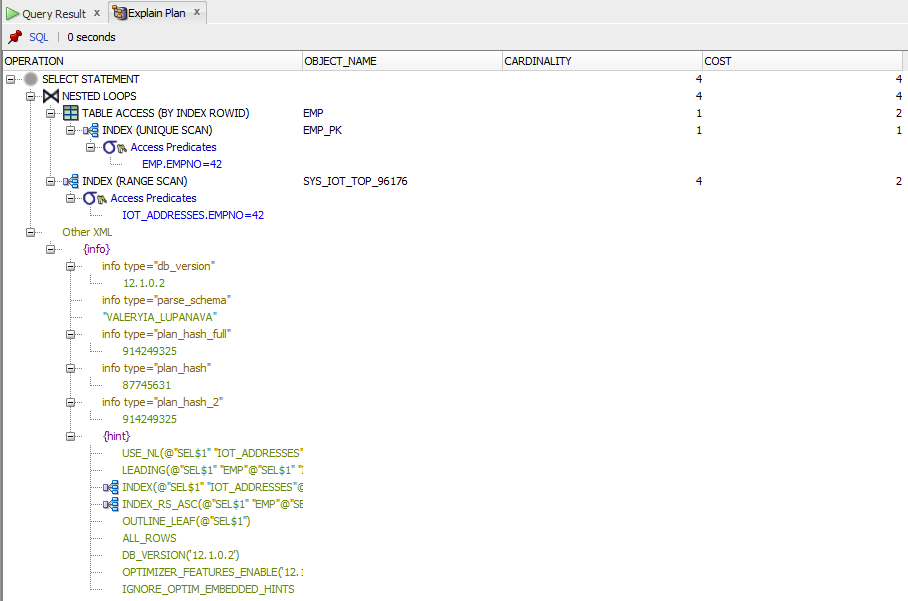
FROM emp ,

iot\_addresses

WHERE emp.empno = iot\_addresses.empno

AND emp.empno = 42;





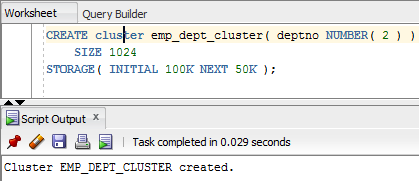
**Вывод: когда мы используете обычную (HEAP) таблицу и создаем индекс на нее, тот индекс использует физический (physical) rowid как указатель на адрес самой записи. Для индексно- организованных таблиц используется несколько иной подход: сразу создается индекс, основанный на первичном ключе и включающий прочие колонки записи. Фактически есть только индекс, а таблицы в виде кучи нет. Поэтому для такого типа индекса оракл не использует физический ROWID. Предназначение ИОТ - взять ключи и сами данные одновременно, избегая индексно-табличного поиска. И минимизировать операции ввода вывода в ходе этого процесса. Что мы и наблюдаем по плану выполнения. Стоимость поиска по ИОТ меньше (EXPLAIN PLAN 2), чем стоимость по куче (EXPLAIN PLAN 1).**

# Index Clustered Tables

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

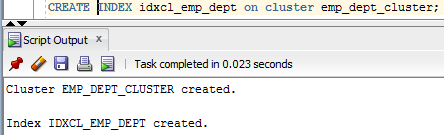
Step 1:

CREATE cluster emp\_dept\_cluster( deptno NUMBER( 2 ) )  
 SIZE 1024   
STORAGE( INITIAL 100K NEXT 50K );



Step 2:

CREATE INDEX idxcl\_emp\_dept on cluster emp\_dept\_cluster;



Step 3:

CREATE TABLE dept

(

deptno NUMBER( 2 ) PRIMARY KEY

, dname VARCHAR2( 14 )

, loc VARCHAR2( 13 )

)

cluster emp\_dept\_cluster ( deptno ) ;

CREATE TABLE emp

(

empno NUMBER PRIMARY KEY

, ename VARCHAR2( 10 )

, job VARCHAR2( 9 )

, mgr NUMBER

, hiredate DATE

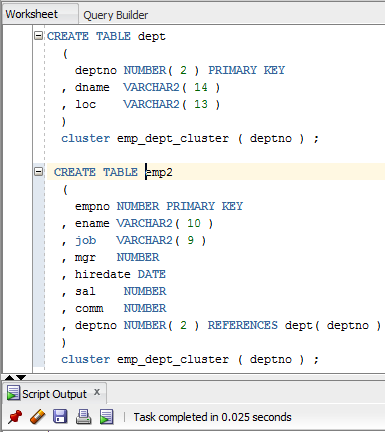
, sal NUMBER

, comm NUMBER

, deptno NUMBER( 2 ) REFERENCES dept( deptno )

)

cluster emp\_dept\_cluster ( deptno ) ;



Step 4:

INSERT INTO dept( deptno , dname , loc)

SELECT deptno , dname , loc

FROM scott.dept;

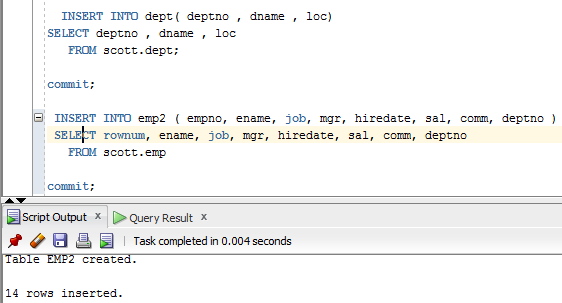
commit;

INSERT INTO emp ( empno, ename, job, mgr, hiredate, sal, comm, deptno )

SELECT rownum, ename, job, mgr, hiredate, sal, comm, deptno

FROM scott.emp

commit;



**Изначально мы создали кластер и потом кластерный индекс. Параметр SIZE в кластере указывается для того, чтобы определить, какое количество байт может включать один ключ. В данном случае 1024 байта. Отсюда можно понять сколько кластерный ключей сможет вместить наш кластер. Можно обратить внимание, что при создании индекса столбцы не указывались, потому что они наследуются из самого кластера. Создали две кластеризованные таблицы и заполнили их данными.**

Step 5:

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

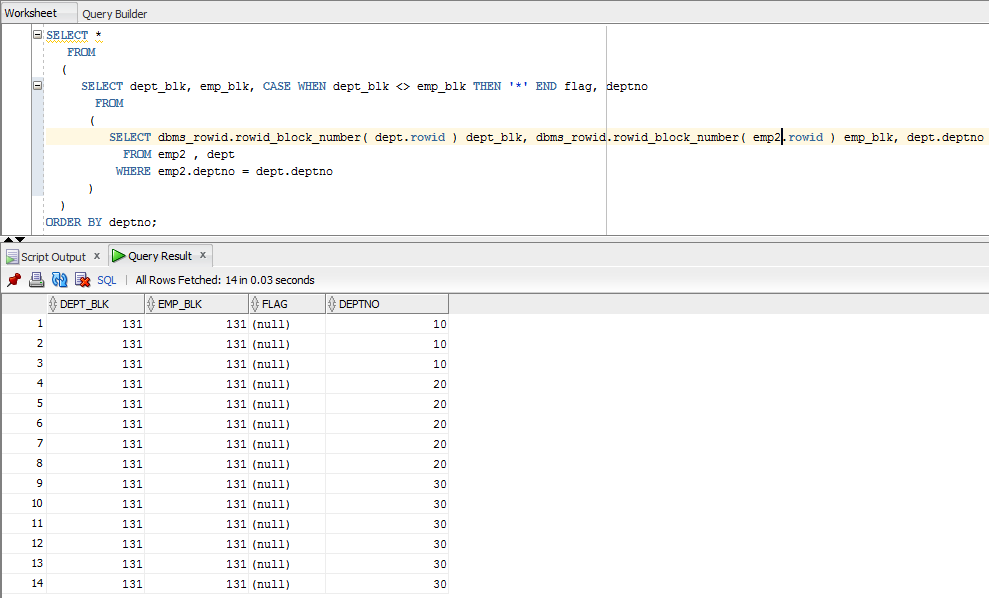
FROM emp , dept

WHERE emp.deptno = dept.deptno

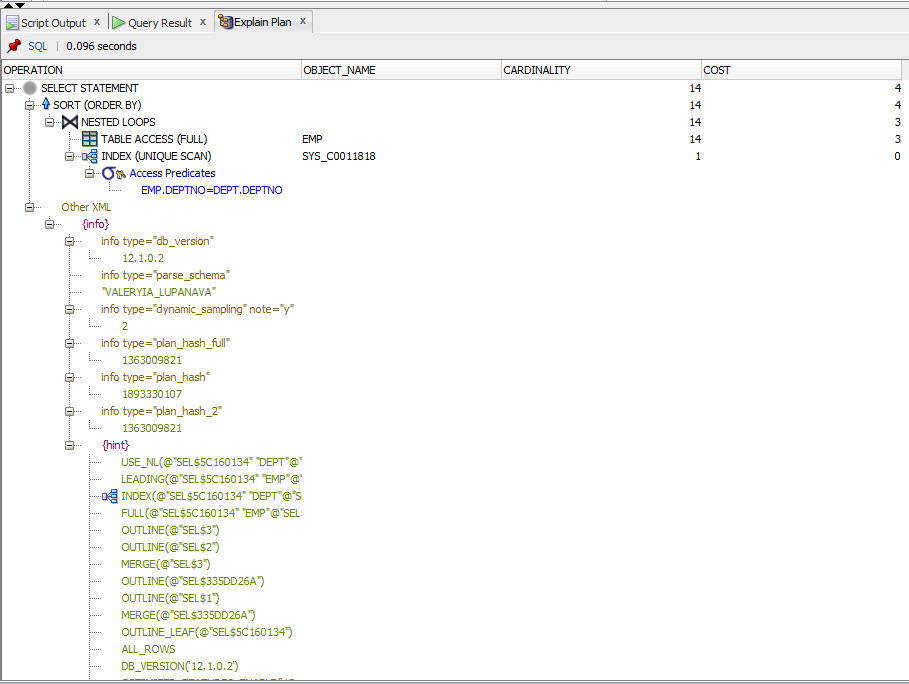
)

)

ORDER BY deptno;



Explanation plan:



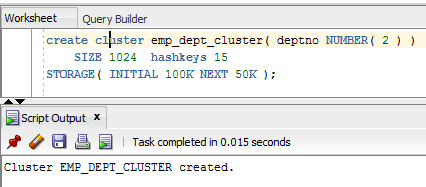
**Вывод: можно увидеть, что поскольку таблицы принадлежать одному кластеру, то в они хранятся в одном блоке. Следовательно при обрщении к таблицам происходит обращение не к каждой по отдельности, а сразу ко всему блоку, что существенно ускоряет процесс и уменьшает стоимость. В EXPLAIN PLAN можно это и наблюдать. Однако, если наши таблицы будут часто изменяться или нужен будет постоянно поиск по какой-то определенной таблице, то помещать их все в кластер нецелесообразно.**

# Hash Clustered Tables

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

Step 1:

CREATE cluster emp\_dept\_cluster( deptno NUMBER( 2 ) )  
 SIZE 1024 hashkeys 15  
STORAGE( INITIAL 100K NEXT 50K );



**Можно увидеть, что в кластер мы добавили параметр HASHKEYS, который который отвечает за размер хэш таблицы. Это значение умножается на SIZE и в итоге получается размер, выделяемый под кластер.**

Step 2:

CREATE TABLE dept

(

deptno NUMBER( 2 ) PRIMARY KEY

, dname VARCHAR2( 14 )

, loc VARCHAR2( 13 )

)

cluster emp\_dept\_cluster ( deptno ) ;

CREATE TABLE emp

(

empno NUMBER PRIMARY KEY

, ename VARCHAR2( 10 )

, job VARCHAR2( 9 )

, mgr NUMBER

, hiredate DATE

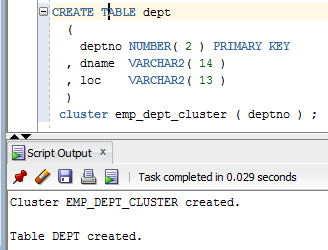
, sal NUMBER

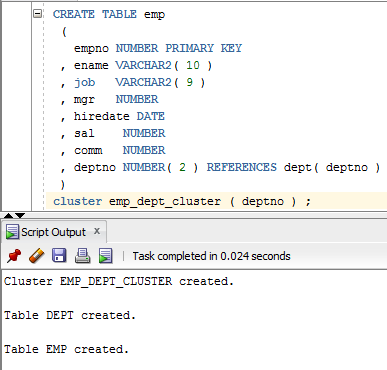
, comm NUMBER

, deptno NUMBER( 2 ) REFERENCES dept( deptno )

)

cluster emp\_dept\_cluster ( deptno ) ;





Step 3:

INSERT INTO dept( deptno , dname , loc)

SELECT deptno , dname , loc

FROM scott.dept;

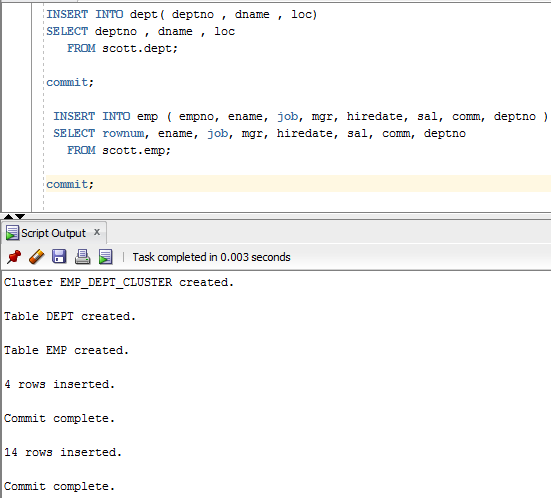
commit;

INSERT INTO emp ( empno, ename, job, mgr, hiredate, sal, comm, deptno )

SELECT rownum, ename, job, mgr, hiredate, sal, comm, deptno

FROM scott.emp

commit;



Step 4:

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

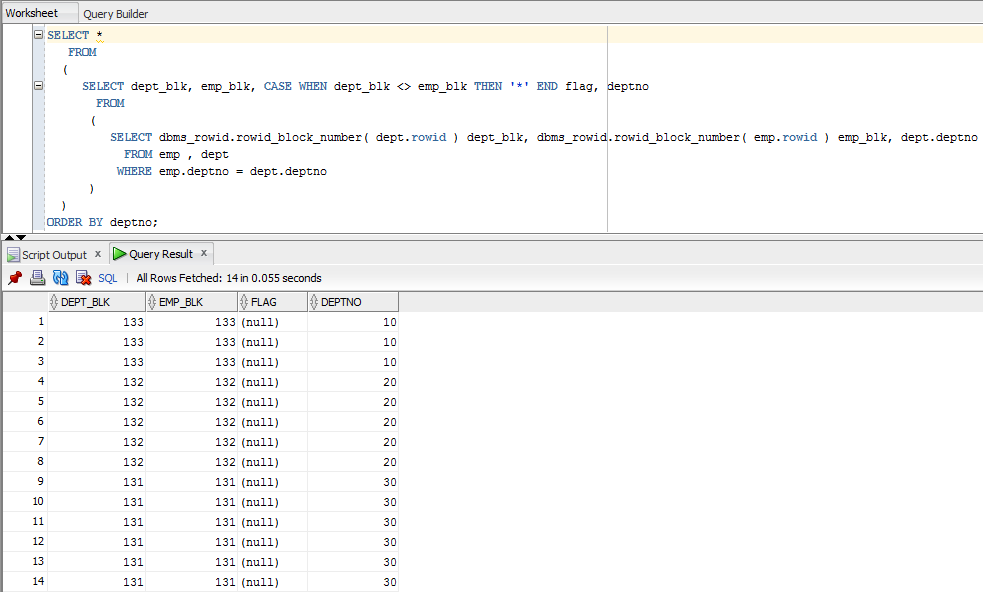
FROM emp , dept

WHERE emp.deptno = dept.deptno

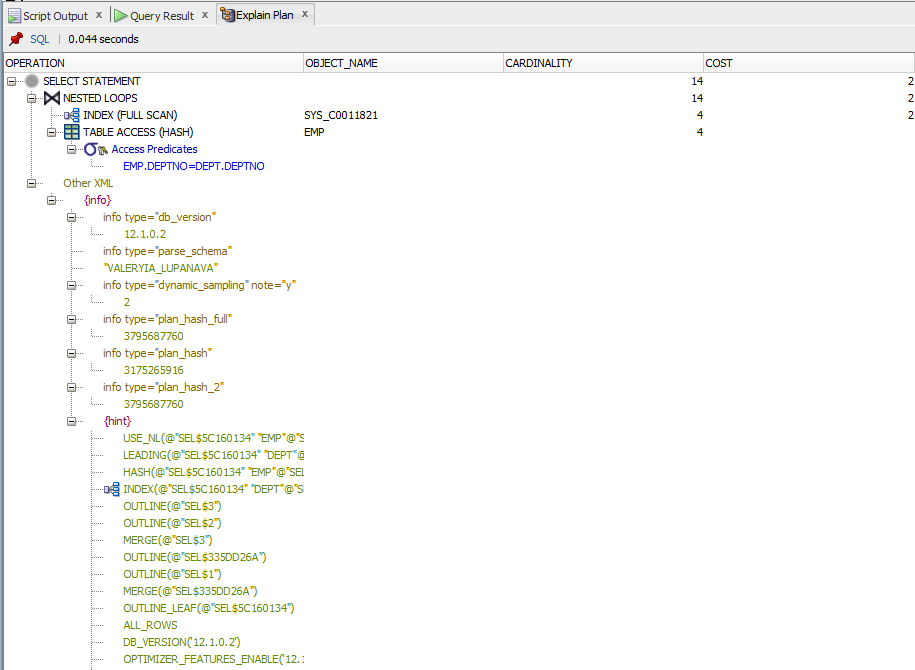
)

)

ORDER BY deptno;



Explanation plan:



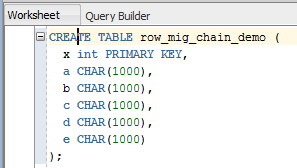
**Вывод: можно увидеть, что в этом случае строки извлекаются в соответствии с результатом хэш-функции. Т.е. таблицы объе диняются в кластере, затем каждой строке данных присвается хэш-функция, по которой впоследствии и будет осуществляться поиск. Для нахождения любого значения строки все, что потребуется сделать – это получить хэш-значение для ключа кластера - строки, так что единственная операция ввода-вывода выдаст данные нужной строки и обеспечит более эффективную производительность.**

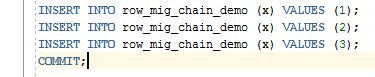
# Row Migration\*

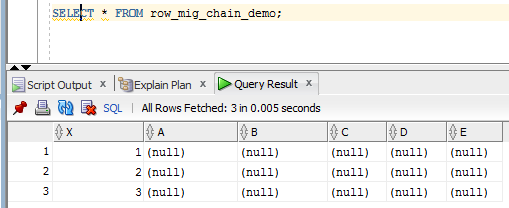
## Task results

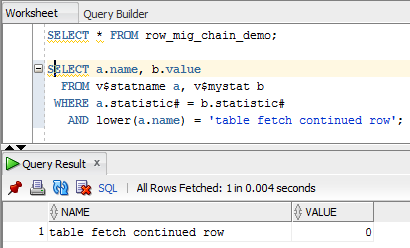
Row migration.

**Миграция означает, что вся строка будет двигаться, и мы просто оставим rowid.**

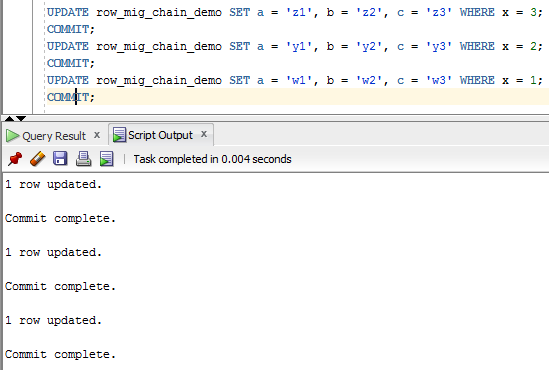


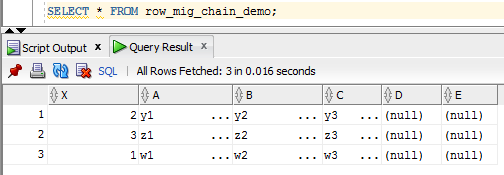


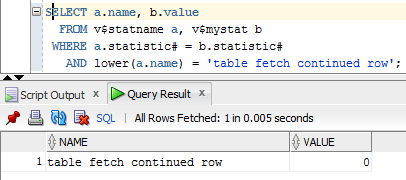




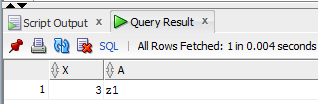
**Ноль показывает, что все значения SELECT находятся в одном DATA BLOCK и ему не пришлось перемещаться.**

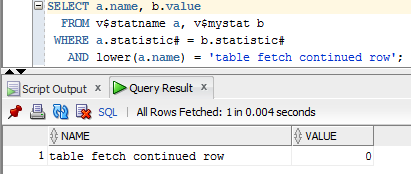




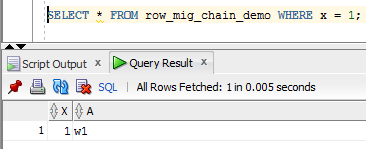


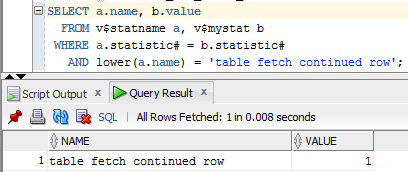
**Когда мы обновили данные, то добавили значения в а, b и c. Мы сначала обновили строку 3. Он не должен был мигрировать, но он заполнил блок 1. Затем мы обновили строку 2. Он переместился в блок 2 со строкой 3. Затем мы обновили строку 1, она перенесена в блок 3. Мы перенесли строки 2 и 1, оставив 3, где она была изначально. Но из-за того, что поиск осуществлен по диапазону индексон, он проигнорировал указатель строки и не увидел миграцию.**





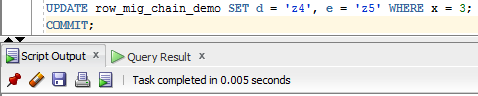
**Если мы проверим перемещение только третьей строки, то миграции снова не увидим, поскольку строка осталась в первом блоке.**



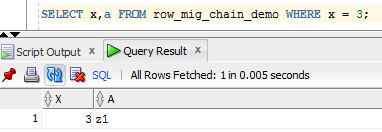


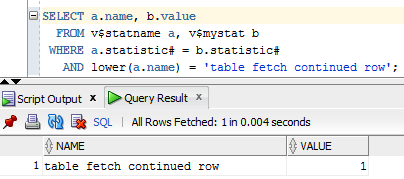
**Если мы проверим перемещение только первой строки, то миграцию теперь увидим, поскольку строка переместилась в третий блок и поиск прошел по конкретному ROWID.**

Row chaining.

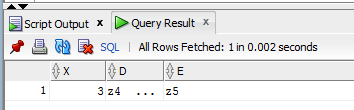


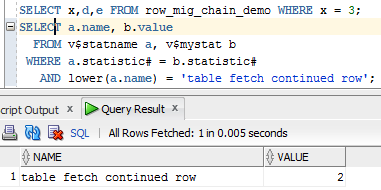
**Теперь третья строка не должна помещаться в первый блок, поскольку добавилось еще два значения для ранее нулевых ячеек.**



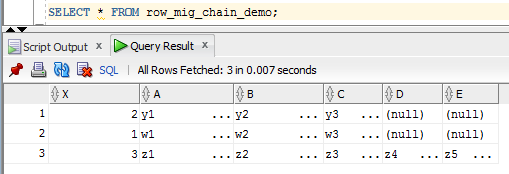


**Теперь мы достали значения x и а из третьей строки. Можно увидеть, что перемещения не произшло, поскольку эти два значения остались в первом блоке.**





**Теперь мы достали значения x и а, е из третьей строки. Можно увидеть, что перемещение произшло, поскольку е уже не поместилось в первый блок, а переместилось в четвертый.**



**То есть третья строка слишком велика для первого блока из-за ячеек d и e, поэтому она разбилась на цепочки по разным блокам, а не мигрировала полностью.**