# **Task 2 Report**

## 1. PREREQUISITE TASK

**Step 1: create new tablespace:**

*alter session set "\_oracle\_script"=true;*

*create tablespace tbs\_lab datafile 'db\_lab\_001.dat' size 5M autoextend ON next 5M*

*MAXSIZE 100M;*

**Step 2: create new user:**

*create user OR\_DEF identified by admin default tablespace tbs\_lab;*

**Step 3: gtant privileges:**

*grant connect to OR\_DEF;*

*grant resource to OR\_DEF;*

*grant select on scott.dept to OR\_DEF;*

*grant select on scott.emp to OR\_DEF;*

*ALTER USER OR\_DEF QUOTA 100M ON tbs\_lab;*

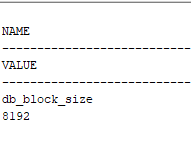
**2. HEAP ORGANIZED TABLES**

**Step 1:show block size**

*SELECT name,value*

*FROM v$parameter*

*WHERE name = 'db\_block\_size';*

**

**Step 2: create table**

*create table t (*

*a int,*

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

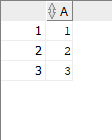
*c varchar2(3000) default rpad('\*',3000,'\*') )*

**Step 3: insert and delete values**

*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;*

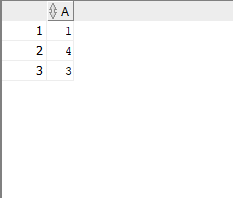
**

*insert into t (a) values ( 4);*

*commit;*

**Step 4: show result**

*select a from t;*



***We see an example of a Heap organized table.***

***For a record, it looks not for a sequence in a block, but for the first free block. So when writing the last value 4, it occupied the block, the freed block after deleting the value 2***

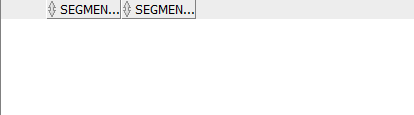
**TASK 2 – UNDERSTANDING HEAP TABLE SEGMENTS**

**Step 1: create table**

*create table t2 ( x int primary key, y clob, z blob );*

**Step 2: show segments**

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

**

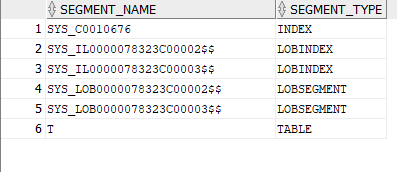
**Step 3: create table**

*Create table t ( x int primary key, y clob, z blob )*

*SEGMENT CREATION IMMEDIATE*

**Step 4: show segments**

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

**

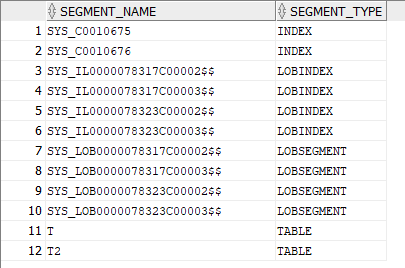
**Step 5: insert data to t2**

*insert into t2 (x) values ( 1);*

*commit;*

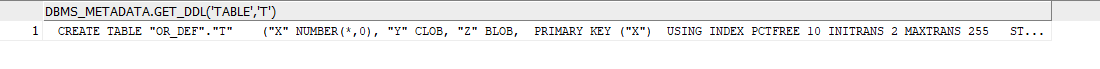
**Step 6: show segments**

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

**

**Step 5: show metadata**

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

**

***Oracle has a feature - creating segments on demand. This means that when we define any segment (table, index, etc.), none of the associated segments are created until rows are inserted.***

***By default, the DEFERRED\_SEGMENT\_CREATION system value is enabled. This behavior is shown in the table t2 as an example. If we call the user\_segments table, we can tell that none of the visible segments have been created. After inserting data into the table - segments are created.***

***By specifying SEGMENT CREATION IMMEDIATE as in table t, the segment will be created when the table is created.***

**3.INDEX ORGANIZED TABLES. COMPARE PERFORMANCE**

**Step 1: create table and index**

*CREATE TABLE emp*

*AS SELECT object\_id empno , object\_name ename , created hiredate , owner job*

*FROM all\_objects*

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

Calculate Statistic:

*begin*

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

*end;*

**Step 2: create table**

*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) )*

**Step 3: create table**

*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*

**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' );*

**Step 6: Compare Trace and Performance:**

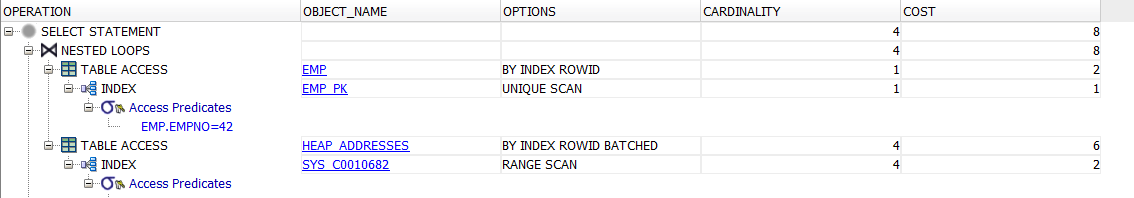
**Explain 1:**

*SELECT \**

*FROM emp , heap\_addresses*

*WHERE emp.empno = heap\_addresses.empno*

*AND emp.empno = 42;*

**

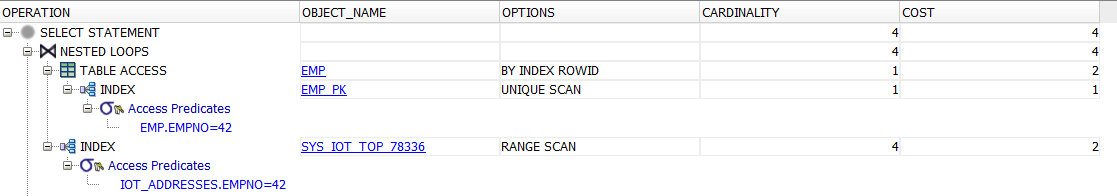
**Explain 2:**

*SELECT \**

*FROM emp , iot\_addresses*

*WHERE emp.empno = iot\_addresses.empno*

*AND emp.empno = 42;*

**

***The first query with no heap\_addresses index has a COST of 8. The second query with an iot\_addresses index has a COST of 4.***

***The only difference in the plans of these queries is that in the first case we use table access to read addresses (cost - 6), and in the second case we use an index to get information (cost - 2).***

**4. \*\*\* ROW MIGRATION**

**my block size:**

*SELECT name,value*

*FROM v$parameter*

*WHERE name = 'db\_block\_size';*

**

**Create example table:**

*CREATE TABLE row\_mig\_chain\_demo(*

*x int PRIMARY KEY,*

*a CHAR(2000),*

*b CHAR(2000),*

*c CHAR(2000),*

*d CHAR(2000),*

*e CHAR(2000)*

*);*

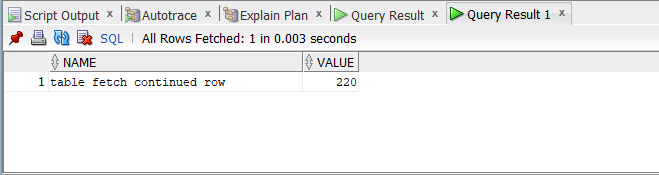
**Number of database chained rows:**

*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';*

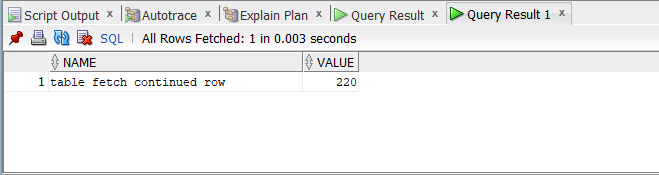
**

**Insert values and check again:**

*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);*

**

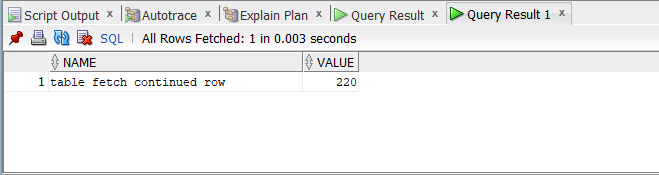
The number hasn’t changed. This data is so small right now, all three rows fit on a single block.

**Update rows:**

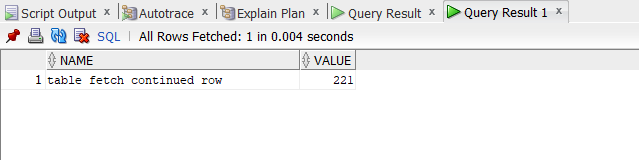
UPDATE row\_mig\_chain\_demo SET a = 'z1', b = 'z2', c = 'z3' WHERE x = 3;

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

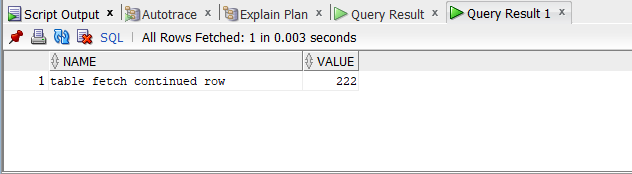
UPDATE row\_mig\_chain\_demo SET a = 'w1', b = 'w2', c = 'w3' WHERE x = 1;

**

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



SELECT \* FROM row\_mig\_chain\_demo WHERE x = 2;



This means rows with id 2 and 1 migrated.

**5. TYPES OF INDEXES**

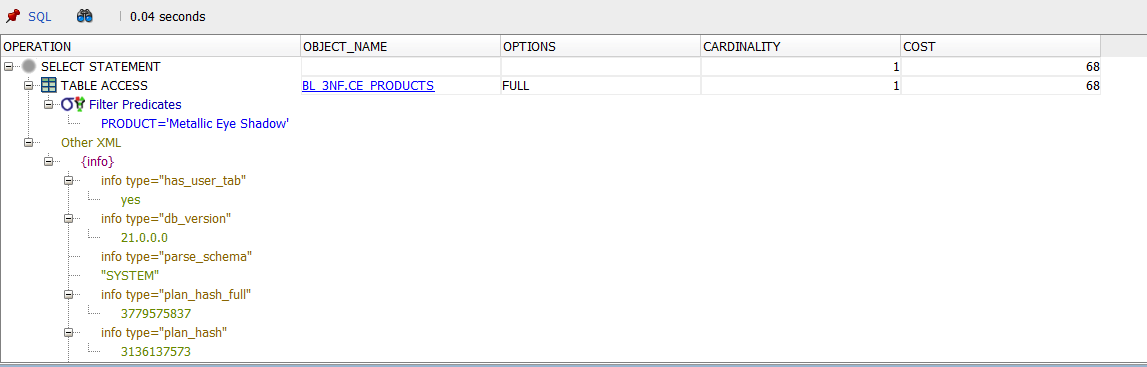
**Test query:**

*SELECT \**

*FROM bl\_3nf.ce\_products*

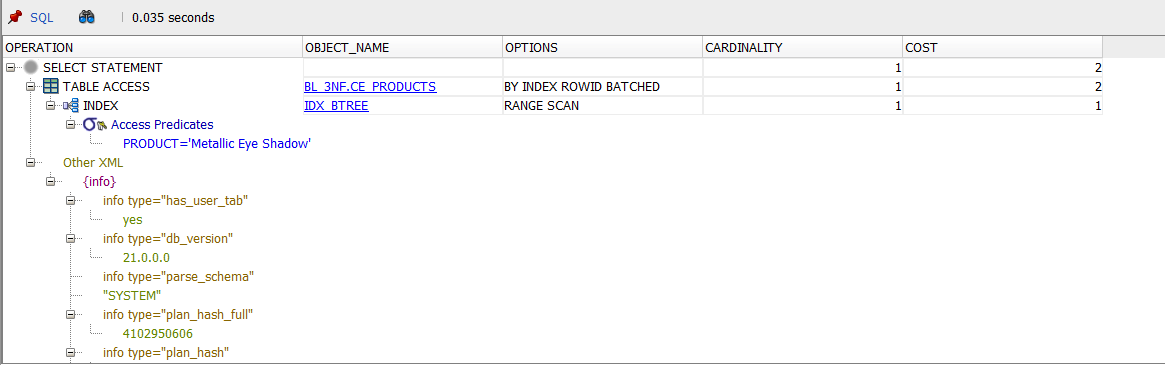
*WHERE unit\_price >400;*

Before creation of INDEX returns Explain plan:

****

**Create index:**

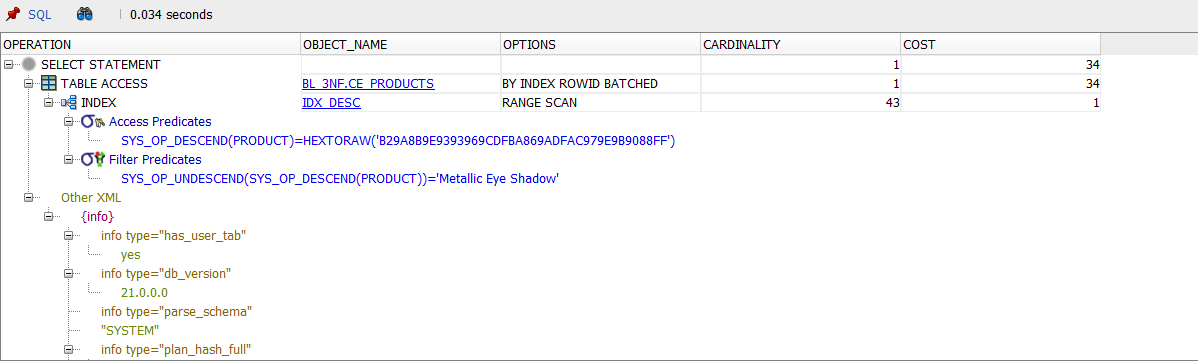
*CREATE INDEX idx\_btree ON bl\_3nf.ce\_products (product);*

****

We should use b-tree index When columns are unique or near-unique.

Create index:

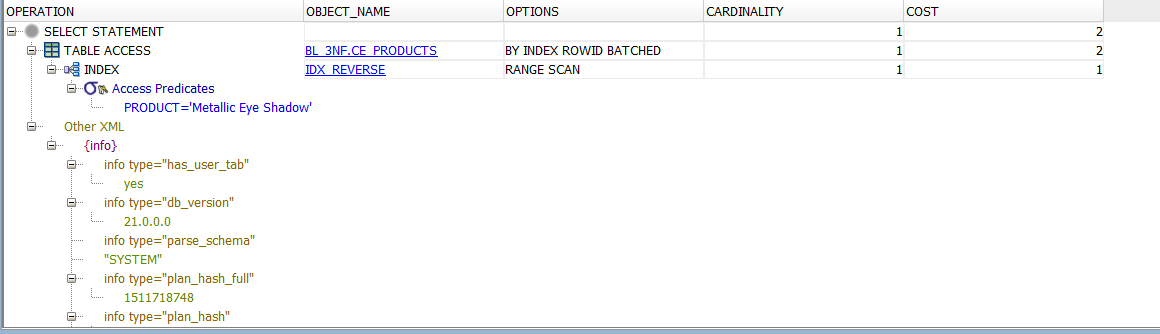
*CREATE INDEX idx\_desc ON bl\_3nf.ce\_products (product desc);*



We should use descending indexes in cases, when our B-tree can be unbalanced by an ever-increasing value for an index.

Create index:

*CREATE INDEX idx\_reverse ON bl\_3nf.ce\_products (product) REVERSE;*



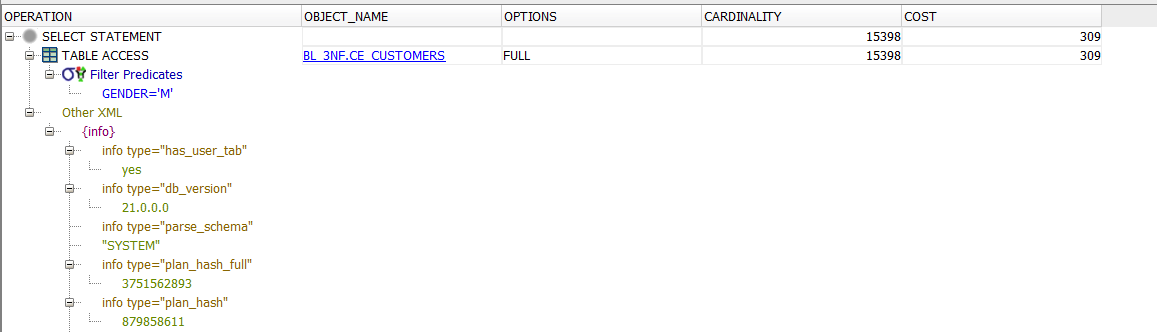
We should use reverse index for monotonically increasing values ​​(e.g. auto-increment identifier)

For bitmap index I used different query:

Test query:

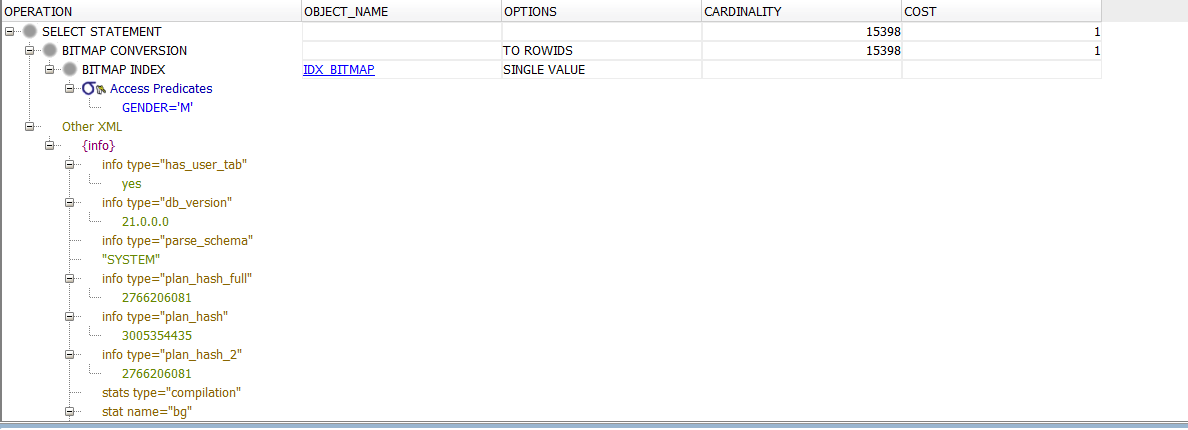
*Select gender from bl\_3nf.ce\_customers Where gender='M';*

Before creation of INDEX returns Explain plan:



Create index:

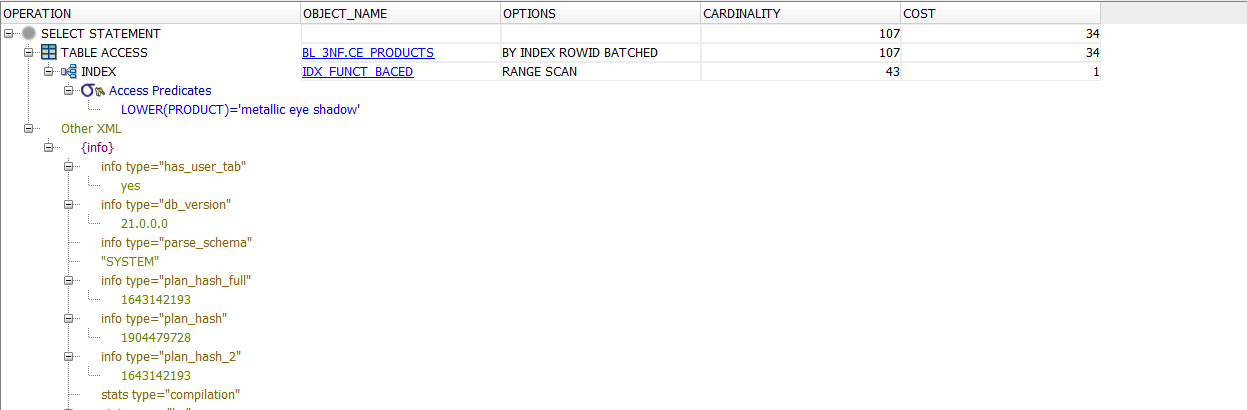
*CREATE BITMAP INDEX idx\_bitmap ON bl\_3nf.ce\_customers (gender);*



We should use bitmap index for columns having low distinct values

Create index:

*CREATE INDEX idx\_funct\_baced ON bl\_3nf.ce\_products (lower(product));*



We should use function based index If we have a query that consists of expression and use this query many times