PRACTICAL NO. 4

NAME ARJUN DOYE
ROLL 48 7b

```
/*
Problem Statement:
Write and Execute SQL indexing queries for
data warehouse.
*/


/*Query 1:
Create individual b-tree indexes on the following columns of the table:
customers_copy_btree
(a) cust_gender
(b) cust_year_of_birth
(c) cust_last_name
(d) cust_street_address
How long does it take to create the indexes?
*/

/*btree indexes*/

create table customer_copy
as select * from sh.customers;

set timing on;

create index cust_gender_btree on customer_copy(cust_gender);

/*Index created.

Elapsed: 00:00:00.20
*/

create index cust_year_of_birth_btree on customer_copy(cust_year_of_birth);

/*Index created.

Elapsed: 00:00:00.03
*/

create index cust_last_name_btree on customer_copy(cust_last_name);

/*Index created.

Elapsed: 00:00:00.03
*/
```

```
create index cust_street_address_btree on customer_copy(cust_street_address);

/*Index created.

Elapsed: 00:00:00.07
*/

---------------------------------------------------------------------------------------------------------------------------

/*Query 2:
Create bitmap indexes on the above columns. How long does it take to create bitmap
indexes? Compare it with the results of btree index creation.
*/
/*bitmap indexes*/

create table customer_copy_bitmap
as select * from customer_copy;

/*
Table created.

Elapsed: 00:00:01.48
*/


create bitmap index cust_gender_bitmap on customer_copy_bitmap(cust_gender);

/*Index created.

Elapsed: 00:00:00.20
*/

create bitmap index cust_year_of_birth_bitmap on customer_copy_bitmap(cust_year_of_birth);

/*
Index created.

Elapsed: 00:00:00.04
*/

create bitmap index cust_last_name_bitmap on customer_copy_bitmap(cust_last_name);

/*
Index created.

Elapsed: 00:00:00.03
*/
```

```
create bitmap index cust_street_address_bitmap on customer_copy_bitmap(cust_street_address);

/*
Index created.

Elapsed: 00:00:00.54
*/

---------------------------------------------------------------------------------------------------------------------------

/*Query 3:
Do as directed:
(a) Find the size of each segment: customers_copy_bitmap and customers_copy_btree
(b) The b-tree index range for high and low cardinality address index.
(c) The bitmap index range for high and low cardinality address index.
*/


select segment_name, bytes/1024/1024 from user_segments
    where segment_name like '%BTREE'
    order by BYTES;

/*
SEGMENT_NAME                                                                    BYTES/1024/1024
------------------------------------------------------------------------------- ---------------
CUST_GENDER_BTREE                                                                          .875
CUST_YEAR_OF_BIRTH_BTREE                                                                      1
CUST_LAST_NAME_BTREE                                                                          2
CUST_STREET_ADDRESS_BTREE                                                                     3
SALES_COPY_BTREE                                                                             36

Elpsed: 00:00:00.03
*/

--OR--

select bytes/1024/1024 from user_segments
  where segment_name='CUSTOMER_COPY_BITMAP';

/*
BYTES/1024/1024
---------------
             12
*/

select segment_name, bytes/1024/1024 from user_segments
    where segment_name like '%BITMAP'
    order by BYTES;
```

```
/*
SEGMENT_NAME                                                                    BYTES/1024/1024
------------------------------------------------------------------------------- ---------------
CUST_GENDER_BITMAP                                                                        .0625
CUST_LAST_NAME_BITMAP                                                                      .125
CUST_YEAR_OF_BIRTH_BITMAP                                                                  .1875
CUST_STREET_ADDRESS_BITMAP                                                                    3
CUSTOMER_COPY_BITMAP                                                                         12

Elapsed: 00:00:00.01
*/

--OR--

select bytes/1024/1024 from user_segments
   where segment_name='CUSTOMER_COPY';

 /*
 BYTES/1024/1024
---------------
          12
 */

 ---------------------------------------------------------------------------------------------------------------------------------


/*Query 4:
 Use year of birth, which had 75 different values in our test data as filter column. Also show
the execution plan for both indexes- btree and bitmap. Compare the cost of the execution plan
for b-tree and bitmap indexes.
*/

 set lines 200
set autotrace traceonly
select * from   customer_copy where  cust_year_of_birth = 1967;

/*
956 rows selected.

Elapsed: 00:00:00.03

Execution Plan
----------------------------------------------------------
Plan hash value: 718019990

------------------------------------------------------------------------------
| Id  | Operation         | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |              |   956 |   278K|   406   (1)| 00:00:05 |
```

```
|*  1 |  TABLE ACCESS FULL| CUSTOMER_COPY |   956 |   278K|   406   (1)| 00:00:05 |
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("CUST_YEAR_OF_BIRTH"=1967)

Note
-----
   - dynamic sampling used for this statement (level=2)


Statistics
----------------------------------------------------------
          9  recursive calls
          0  db block gets
       1580  consistent gets
          3  physical reads
          0  redo size
     152373  bytes sent via SQL*Net to client
       1212  bytes received via SQL*Net from client
         65  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        956  rows processed
*/

set lines 200
set autotrace traceonly
select * from   customer_copy where  cust_year_of_birth = 1967;

/*
956 rows selected.

Elapsed: 00:00:00.01

Execution Plan
----------------------------------------------------------
Plan hash value: 718019990

--------------------------------------------------------------------------------
| Id  | Operation         | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |               |   956 |   278K|   406   (1)| 00:00:05 |
|*  1 |  TABLE ACCESS FULL| CUSTOMER_COPY |   956 |   278K|   406   (1)| 00:00:05 |
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):
```

```
------------------------------------------------

   1 - filter("CUST_YEAR_OF_BIRTH"=1967)

Note
-----
   - dynamic sampling used for this statement (level=2)


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
       1519  consistent gets
          0  physical reads
          0  redo size
     152373  bytes sent via SQL*Net to client
       1212  bytes received via SQL*Net from client
         65  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        956  rows processed
*/

 ---------------------------------------------------------------------------------------------------------------------------

/*Query 5:
 Show that update to the bitmap indexed column takes a bit longer than the b-tree indexed
column.
(a) Create new indexes on cust_id column of btree and bitmap table.
(b) Set the timing on
(c) Write a PL/SQL procedure for each table as directed:
       i. Create new columns- upd_cust_id and cust_yob_value with number format.
       ii. In loop of 500 allot random values to both columns
       iii. set cust_year_of_birth = cust_yob_value and consider cust_id = upd_cust_id
(d) What is the size of the indexes compared to the size as before the updates took place.
*/

create index cust_id_1_idx on customer_copy (cust_id);

create index cust_id_2_idx on customer_copy_bitmap (cust_id);

declare
       upd_customer_id number(5);
       customer_yob_value number(4);
       begin
             for i in 1 .. 5000 loop
                   upd_customer_id := dbms_random.value(1,55000);
                   customer_yob_value := dbms_random.value(1900,2000);
```

```
                update customer_copy
                set cust_year_of_birth = customer_yob_value
                where cust_id = upd_customer_id;
                commit;
            end loop;
        end;
/

/*
PL/SQL procedure successfully completed.

Elapsed: 00:00:00.28
*/

-----------------------------------

declare
        upd_customer_id number(5);
        customer_yob_value number(4);
        begin
            for i in 1 .. 5000 loop
                upd_customer_id := dbms_random.value(1,55000);
                customer_yob_value := dbms_random.value(1900,2000);
                update customer_copy_bitmap
                set cust_year_of_birth = customer_yob_value
                where cust_id = upd_customer_id;
                commit;
            end loop;
        end;
 /

/*
PL/SQL procedure successfully completed.

Elapsed: 00:00:00.82
*/

-----------------------------------------------------------------------------------------------------------------------------------

/*Query 6:
 Comparison of time for index creation for normal bitmap index and join bitmap index.
Do as directed:
a) Create table customers_bijx_test_bitmap from customers & sales_bijx_test_bitmap as
from sales
b) create bitmap index sales_bijx_test_bitmap_bix1 on sales_bijx_test_bitmap table and
cust_id column, and bitmap index cust_bijx_test_bitmap_bix1 on
customers_bijx_test_bitmap table and cust_last_name column.
What is the elapsed time for each index creation?
c) Create table customers_bijx_test_bitjoin from customers and Create table
```

sales_bijx_test_bitjoin from Sales and add constraint of primary key to cust_id column of cust_bijx_test_bitjoin table.
d) (a) create bitmap index named sales_bijx_test_bitjoin_bjx1 using sales_bijx_test_bitjoin & customers_bijx_test_bitjoin.cust_id tables.
(b) create bitmap index named sales_bijx_test_bitjoin_bjx2 using tables sales_bijx_test_bitjoin and customers_bijx_test_bitjoin.cust_last_name
Conclude which index creation takes more time.
*/

```
create table customers_bijx_test_bitmap as (select * from customers);

/*Table created.

Elapsed: 00:00:00.81
*/

create table sales_bijx_test_bitmap as (select * from sales);

/*Table created.

Elapsed: 00:00:02.93*/

select count(*) from customers_bijx_test_bitmap;
/*
  COUNT(*)
----------
     55500

Elapsed: 00:00:00.20
*/

select count(*) from sales_bijx_test_bitmap;
/*
  COUNT(*)
----------
    918843

Elapsed: 00:00:00.57
*/

-----------------------------


create bitmap index sales_bijx_test_bitmap_bix1 on sales_bijx_test_bitmap(cust_id);

/*
Index created.

Elapsed: 00:00:00.16
```

```
*/

create bitmap index cust_bijx_test_bitmap_bix1 on customers_bijx_test_bitmap(cust_last_name);

/*
Index created.

Elapsed: 00:00:00.01
*/

---------------------------------

create table customers_bijx_test_bitjoin as (select * from customers);

/*Table created.

Elapsed: 00:00:00.68
*/
create table sales_bijx_test_bitjoin as (select * from customers);
/*
Table created.

Elapsed: 00:00:01.15
*/
alter table customers_bijx_test_bitjoin add constraint cust_bijx_test_bitjoin_pk primary key (cust_id);
/*
Table altered.

Elapsed: 00:00:00.24
*/

create bitmap index sales_bijx_test_bitjoin_bjx1 on sales_bijx_test_bitjoin(customers_bijx_test_bitjoin.cust_id)
from sales_bijx_test_bitjoin, customers_bijx_test_bitjoin
  where sales_bijx_test_bitjoin.cust_id = customers_bijx_test_bitjoin.cust_id;

/*
Index created.

Elapsed: 00:00:00.45
*/

create bitmap index sales_bijx_test_bitjoin_bjx2 on
sales_bijx_test_bitjoin(customers_bijx_test_bitjoin.cust_last_name)
from sales_bijx_test_bitjoin, customers_bijx_test_bitjoin
where sales_bijx_test_bitjoin.cust_id = customers_bijx_test_bitjoin.cust_id;

/*
Index created.
```

Elapsed: 00:00:00.21
*/

---COMPRESSED INDEX --
/*
1. Create table Student(StudId, StudName)
2. Add 10 Rows
3. Define Index on StudName(First Name and Last Name)
4. Get the Statistics of Index
5. Now add about 10000 rows that will have same last name
6. Get the Statistics of Index
7. Drop Index
8. Create Compressed Index
9. Get the Statistics of Index
10. Compare statics and give your comments*/

--1.
```
create table student(
studid int,
studname varchar2(30));
```

--2.
```
	insert into student values(1,'A');
	insert into student values(2,'B');
	insert into student values(3,'C');
	insert into student values(4,'D');
	insert into student values(5,'E');
	insert into student values(6,'F');
	insert into student values(7,'G');
	insert into student values(8,'H');
	insert into student values(9,'I');
```

--3.
```
CREATE INDEX STUDENT_BTREE_INDEX ON STUDENT(studname);
```
Index created.
Elapsed: 00:00:00.02

--4
```
SELECT
  COMPRESSION,
  LEAF_BLOCKS,
  Round(NUM_ROWS/Decode(LEAF_BLOCKS,0,1,LEAF_BLOCKS)) "ROWS PER BLOCK",   DISTINCT_KEYS,
  NUM_ROWS,NUM_ROWS-DISTINCT_KEYS DUP_ROWS
FROM
  USER_INDEXES
WHERE
```

```
  INDEX_NAME = 'STUDENT_BTREE_INDEX';


COMPRESS LEAF_BLOCKS ROWS PER BLOCK DISTINCT_KEYS   NUM_ROWS   DUP_ROWS
-------- ----------- -------------- ------------- ---------- ----------har
DISABLED         1            9           9        9        0

SELECT t.blocks, t.num_rows, i.clustering_factor
FROM user_tables t, user_indexes i
 WHERE t.table_name = i.table_name AND i.index_name='STUDENT_BTREE_INDEX';

    BLOCKS   NUM_ROWS CLUSTERING_FACTOR
---------- ---------- -----------------
                              1
--5
DECLARE v_a NUMBER;
BEGIN
v_a := 11;
WHILE v_a < 10000
LOOP
INSERT INTO STUDENT VALUES(v_a,'Smith');
v_a := v_a + 1;
END LOOP;
COMMIT;
END;
/

PL/SQL procedure successfully completed.

--6
EXEC DBMS_STATS.gather_table_stats('HASHIR', 'STUDENT');

SELECT
  COMPRESSION,
  LEAF_BLOCKS,
  Round(NUM_ROWS/Decode(LEAF_BLOCKS,0,1,LEAF_BLOCKS)) "ROWS PER BLOCK",   DISTINCT_KEYS,
  NUM_ROWS,NUM_ROWS-DISTINCT_KEYS DUP_ROWS
FROM
  USER_INDEXES
WHERE
   INDEX_NAME = 'STUDENT_BTREE_INDEX';

COMPRESS LEAF_BLOCKS ROWS PER BLOCK DISTINCT_KEYS   NUM_ROWS   DUP_ROWS
-------- ----------- -------------- ------------- ---------- ----------
DISABLED        36          278          10      9998      9988


SQL> SELECT t.blocks, t.num_rows, i.clustering_factor
  2  FROM user_tables t, user_indexes i
```

```
  3   WHERE t.table_name = i.table_name AND i.index_name='STUDENT_BTREE_INDEX';

    BLOCKS   NUM_ROWS CLUSTERING_FACTOR
---------- ---------- -----------------
        28       9998              21

--7
DROP INDEX STUDENT_BTREE_INDEX;

Index dropped.

--8
CREATE INDEX EMP_EMPNAME_IDX ON STUDENT(studname)COMPRESS TABLESPACE USERS;

Index Created.

--9
SELECT
  COMPRESSION,
  LEAF_BLOCKS,
  Round(NUM_ROWS/Decode(LEAF_BLOCKS,0,1,LEAF_BLOCKS)) "ROWS PER BLOCK",   DISTINCT_KEYS,
  NUM_ROWS,NUM_ROWS-DISTINCT_KEYS DUP_ROWS
FROM
  USER_INDEXES
WHERE
   INDEX_NAME = 'STUDNAME_IDX';

COMPRESS LEAF_BLOCKS ROWS PER BLOCK DISTINCT_KEYS   NUM_ROWS   DUP_ROWS
-------- ----------- -------------- ------------- ---------- ----------
ENABLED          16          625           10       9998       9988

SELECT t.blocks, t.num_rows, i.clustering_factor
FROM user_tables t, user_indexes i
 WHERE t.table_name = i.table_name AND i.index_name='STUDNAME_IDX';

    BLOCKS   NUM_ROWS CLUSTERING_FACTOR
---------- ---------- -----------------
        28       9998              21

--10

When compression is enabled, less leaf blocks are created and
rows per block gets increased.

--FUNCTION BASED INDEX
/*Function Based Indexes:
1. Create function based index on Employee table of HR schema. Function should be on
salary attribute based on commission percentage.
Find out list of employees having commission percentage less than 50000.
```

2. Create function based index on employee name for Upper and lower function.
3. Create user table with attributes (UserId, UserName, Gender)
4. Insert 10000 records in user table
5. Build regular index on Username
6. Build function based index on user name based on Upper function
7. Compare the response time and comment.
*/

--1
CREATE TABLE HR AS (SELECT * FROM HR.EMPLOYEES);
SQL> CREATE TABLE HR AS (SELECT * FROM HR.EMPLOYEES);

Table created.

Elapsed: 00:00:00.13

CREATE INDEX INDEX_FBI_HR ON HR(COMMISSION_PCT*SALARY);
SQL> CREATE INDEX INDEX_FBI_HR ON HR(COMMISSION_PCT*SALARY);

Index created.

Elapsed: 00:00:00.03

SELECT * FROM HR
WHERE SALARY*COMMISSION_PCT < 50000;

SQL> SELECT * FROM HR WHERE (COMMISSION_PCT*SALARY) <50000;

EMPLOYEE_ID FIRST_NAME LAST_NAME EMAIL PHONE_NUMBER HIRE_DATE JOB_ID
SALARY COMMISSION_PCT MANAGER_ID DEPARTMENT_ID
---------- -------------------- ------------------------- ------------------------- -------------------- --------- ---------- ---------- --------------
---------- -------------
173 Sundita Kumar SKUMAR 011.44.1343.329268 21-APR-08 SA_REP 6100
.1 148
80
167 Amit Banda ABANDA 011.44.1346.729268 21-APR-08 SA_REP 6200
.1 147
80
179 Charles Johnson CJOHNSON 011.44.1644.429262 04-JAN-08 SA_REP
6200 .1 149
80
166 Sundar Ande SANDE 011.44.1346.629268 24-MAR-08 SA_REP 6400
.1 147
80
165 David Lee DLEE 011.44.1346.529268 23-FEB-08 SA_REP 6800
.1 147
80

164 Mattea Marvins MMARVINS 011.44.1346.329268 24-JAN-08 SA_REP 7200 .1 147 80
155 Oliver Tuvault OTUVAULT 011.44.1344.486508 23-NOV-07 SA_REP 7000 .15 145 80
178 Kimberely Grant KGRANT 011.44.1644.429263 24-MAY-07 SA_REP 7000 .15 149
172 Elizabeth Bates EBATES 011.44.1343.529268 24-MAR-07 SA_REP 7300 .15 148 80
171 William Smith WSMITH 011.44.1343.629268 23-FEB-07 SA_REP 7400 .15 148 80
163 Danielle Greene DGREENE 011.44.1346.229268 19-MAR-07 SA_REP 9500 .15 147 80
154 Nanette Cambrault NCAMBRAU 011.44.1344.987668 09-DEC-06 SA_REP 7500 .2 145 80
153 Christopher Olsen COLSEN 011.44.1344.498718 30-MAR-06 SA_REP 8000 .2 145 80
177 Jack Livingston JLIVINGS 011.44.1644.429264 23-APR-06 SA_REP 8400 .2 149 80
176 Jonathon Taylor JTAYLOR 011.44.1644.429265 24-MAR-06 SA_REP 8600 .2 149 80
161 Sarath Sewall SSEWALL 011.44.1345.529268 03-NOV-06 SA_REP 7000 .25 146 80
170 Tayler Fox TFOX 011.44.1343.729268 24-JAN-06 SA_REP 9600 .2 148 80
169 Harrison Bloom HBLOOM 011.44.1343.829268 23-MAR-06 SA_REP 10000 .2 148 80
149 Eleni Zlotkey EZLOTKEY 011.44.1344.429018 29-JAN-08 SA_MAN 10500 .2 100 80
175 Alyssa Hutton AHUTTON 011.44.1644.429266 19-MAR-05 SA_REP 8800 .25 149 80
152 Peter Hall PHALL 011.44.1344.478968 20-AUG-05 SA_REP 9000 .25 145 80
160 Louise Doran LDORAN 011.44.1345.629268 15-DEC-05 SA_REP 7500 .3 146

80
151 David Bernstein DBERNSTE 011.44.1344.345268 24-MAR-05 SA_REP
9500 .25 145
80
159 Lindsey Smith LSMITH 011.44.1345.729268 10-MAR-05 SA_REP 8000
.3 146
80
162 Clara Vishney CVISHNEY 011.44.1346.129268 11-NOV-05 SA_REP 10500
.25 147
80
168 Lisa Ozer LOZER 011.44.1343.929268 11-MAR-05 SA_REP 11500
.25 148
80
150 Peter Tucker PTUCKER 011.44.1344.129268 30-JAN-05 SA_REP 10000
.3 145
80
158 Allan McEwen AMCEWEN 011.44.1345.829268 01-AUG-04 SA_REP
9000 .35 146
80
148 Gerald Cambrault GCAMBRAU 011.44.1344.619268 15-OCT-07 SA_MAN
11000 .3 100
80
174 Ellen Abel EABEL 011.44.1644.429267 11-MAY-04 SA_REP 11000
.3 149
80
157 Patrick Sully PSULLY 011.44.1345.929268 04-MAR-04 SA_REP 9500
.35 146
80
156 Janette King JKING 011.44.1345.429268 30-JAN-04 SA_REP 10000
.35 146
80
147 Alberto Errazuriz AERRAZUR 011.44.1344.429278 10-MAR-05 SA_MAN
12000 .3 100
80
146 Karen Partners KPARTNER 011.44.1344.467268 05-JAN-05 SA_MAN
13500 .3 100
80
145 John Russell JRUSSEL 011.44.1344.429268 01-OCT-04 SA_MAN 14000
.4 100
80

35 rows selected.

Elapsed: 00:00:00.05

--2

CREATE INDEX EMPNAME_INDEX ON HR(UPPER(FIRST_NAME) || LOWER(LAST_NAME));
SQL> CREATE INDEX EMPNAME_INDEX ON HR(UPPER(FIRST_NAME) || LOWER(LAST_NAME));

Index created.

Elapsed: 00:00:00.09

--3

```
CREATE TABLE user_data (
 userid         NUMBER(10)    NOT NULL,
 username  VARCHAR2(40)  NOT NULL,
 gender      VARCHAR2(1)
);
```

Table created.

Elapsed: 00:00:00.31

--4

```
BEGIN
  FOR userid IN 1 .. 100000 LOOP
    IF MOD(userid, 2) = 0 THEN
      INSERT INTO user_data
      VALUES (userid, 'John', 'M');
    ELSE
      INSERT INTO user_data
      VALUES (userid, 'Jayne', 'F');
    END IF;
    COMMIT;
  END LOOP;
END;
/
```

PL/SQL procedure successfully completed.

Elapsed: 00:00:19.29

--5

```
SQL> CREATE INDEX INDEX_REGULAR ON USER_DATA(USERNAME);
```

Index created.

****
Elapsed: 00:00:00.67
****
SELECT COUNT(*) FROM USER_DATA;
Elapsed: 00:00:00.67
SQL>

--6
CREATE INDEX INDEX_FBI_USERNAME ON USER_DATA(UPPER(USERNAME));

SQL> CREATE INDEX INDEX_FBI_USERNAME ON USER_DATA(UPPER(USERNAME));

Index created.

SQL> SELECT COUNT(*) FROM USER_DATA;

  COUNT(*)

    100000

Elapsed: 00:00:00.10

SQL>

--7
Function based index gives a faster retrieval than normal BTree index.

--INDEX ORGANIZED TABLE

/*1. Create an IOT look_ups with the attributes (lookup_code, lookup_value,
lookup_description) in tablespace ts_lookup.
Constraint: lookup_code should be primary key
PctThreshold is 20 and and lookup_description should be in overflow area.
Overflow should be in ts_overflow tablespace.
2. Create a Index Organized Table(IOT) emp_iot based on hr.employees
3. Create a Index Organized Table(IOT) emp101_emp based on hr.employees. Place the
column hiredate in overflow area.
4. Compare the timings of executing select all from employees,emp_iot, and emp101_iot.
Comment on your observations.*/

--1
CREATE TABLESPACE LOOKUPS DATAFILE 'C:\Users\admin\Desktop\DWM' SIZE 10M;
SQL> CREATE TABLESPACE LOOKUPTEST DATAFILE 'C:\Users\admin\Desktop\DWM\lookuptest.dbf' SIZE 10M;

Tablespace created.

Elapsed: 00:00:01.51
SQL>

CREATE TABLE IOT_LOOKUPS(
lookup_code NUMBER(10),
lookup_value NUMBER(10),
lookup_description VARCHAR2(40),
CONSTRAINT LOOKUP_PK PRIMARY KEY(LOOKUP_CODE))

```
ORGANIZATION INDEX
PCTTHRESHOLD 20
INCLUDING LOOKUP_DESCRIPTION
OVERFLOW TABLESPACE LOOKUPTEST;
```

Table created.

```
BEGIN
  FOR lookup_code IN 1 .. 100000 LOOP
    IF MOD(lookup_code, 2) = 0 THEN
      INSERT INTO IOT_LOOKUPS
      VALUES (lookup_code, lookup_code+1, 'String');
    ELSE
      INSERT INTO IOT_LOOKUPS
      VALUES (lookup_code, lookup_code+2, 'String_2');
    END IF;
    COMMIT;
  END LOOP;
END;
/
```

PL/SQL procedure successfully completed.

Elapsed: 00:00:06.98

```
--2

CREATE TABLE EMP_IOT
(EMP_NO NUMBER,
EMP_NAME VARCHAR2(20),
EMP_DEPT NUMBER,
EMP_ADDRESS VARCHAR2(500),
EMP_HIST VARCHAR2(1000),
CONSTRAINT EMP_PK PRIMARY KEY(EMP_NO))
ORGANIZATION INDEX
INCLUDING EMP_NAME
TABLESPACE LOOKUP_DESCRIPTION
OVERFLOW TABLESPACE LOOKUPTEST;
```

Tablespace Created.

```
BEGIN
  FOR EMP_NO IN 1 .. 100000 LOOP
    IF MOD(EMP_NO, 2) = 0 THEN
      INSERT INTO EMP_IOT
      VALUES (EMP_NO,'SAM', 101, 'MYADDRESS', 'OLDTEXT');
    ELSE
      INSERT INTO EMP_IOT
      VALUES (EMP_NO,'JAM', 501, 'MYADDRESS', 'OLDTEXT');
```

```
   END IF;
   COMMIT;
  END LOOP;
END;
/

PL/SQL procedure successfully completed.

Elapsed: 00:00:13.96
--3

CREATE TABLE EMP_IOT101
(EMP_NO NUMBER,
EMP_NAME VARCHAR2(20),
EMP_DEPT NUMBER,
EMP_ADDRESS VARCHAR2(500),
EMP_HIST VARCHAR2(1000),
HIREDATE DATE,
CONSTRAINT EMP_PUK PRIMARY KEY(EMP_NO))
ORGANIZATION INDEX
INCLUDING HIREDATE
TABLESPACE LOOKUPTEST
OVERFLOW TABLESPACE LOOKUPTEST;

Table created.

BEGIN
  FOR EMP_NO IN 1 .. 100000 LOOP
   IF MOD(EMP_NO, 2) = 0 THEN
    INSERT INTO EMP_IOT101
    VALUES (EMP_NO,'SAM', 101, 'MYADDRESS', 'OLDTEXT', '1-4-2005');
   ELSE
    INSERT INTO EMP_IOT101
    VALUES (EMP_NO,'JAM', 501, 'MYADDRESS', 'OLDTEXT','1-4-2005');
   END IF;
   COMMIT;
  END LOOP;
END;
/

PL/SQL procedure successfully completed.

Elapsed: 00:00:20.16

--------------------------------------------------------------------------------** EOF
**--------------------------------------------------------------------------------
```