NAME: Heet Dhanuka
ROLL NO.: B -34
BATCH: B2

DWDM Practical 4

Aim : Implementing indexes and IOT in Oracle.

Q1. Create a copy of customers table and name it customers_copy_btree_<rollno>. Create individual b-tree indexes on the following columns of the table customers_copy_btree_<rollno> :
(a) cust_gender
(b) cust_year_of_birth
(c) cust_last_name
(d) cust_street_address

```
SQL> CREATE TABLE customers_copy_btree_34 AS
  2  SELECT * FROM customers;

Table created.
```

```
SQL> CREATE INDEX idx_cust_gender_34
  2  ON customers_copy_btree_34(cust_gender);

Index created.

SQL>
SQL> CREATE INDEX idx_cust_year_of_birth_34
  2  ON customers_copy_btree_34(cust_year_of_birth);

Index created.

SQL>
SQL> CREATE INDEX idx_cust_last_name_34
  2  ON customers_copy_btree_34(cust_last_name);

Index created.

SQL>
SQL> CREATE INDEX idx_cust_street_address_34
  2  ON customers_copy_btree_34(cust_street_address);

Index created.
```

Q2. 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.

```
SQL> CREATE BITMAP INDEX bm_idx_cust_gender_34
  2  ON customers_copy_btree_34(cust_gender);

Index created.

SQL>
SQL> CREATE BITMAP INDEX bm_idx_cust_year_of_birth_34
  2  ON customers_copy_btree_34(cust_year_of_birth);

Index created.

SQL>
SQL> CREATE BITMAP INDEX bm_idx_cust_last_name_34
  2  ON customers_copy_btree_34(cust_last_name);

Index created.

SQL>
SQL> CREATE BITMAP INDEX bm_idx_cust_street_address_34
  2  ON customers_copy_btree_34(cust_street_address);

Index created.
```

1. B-Tree Index is typically **faster** to create than a **Bitmap Index**, especially if the table is large.
2. Bitmap Index takes **more time** but is **smaller in size** for columns with low cardinality

Aspect	B-Tree Index	Bitmap Index
Query	`CREATE INDEX`	`CREATE BITMAP INDEX`
Best for	High-cardinality columns	Low-cardinality columns
Creation Speed	Faster	Slower
Storage Size	Larger	Smaller
DML Impact	Minimal	Slows updates/deletes
Usage	OLTP (frequent updates)	OLAP (analytics, filters)

Q3. Find the size of each segment: customers_copy_bitmap and customers_copy_btree
(Hint : Use users_segment table)

```
SQL> SELECT segment_name, bytes / (1024 * 1024) AS size_mb
  2  FROM user_segments
  3  WHERE segment_name IN ('CUSTOMERS_COPY_BTREE_34', 'CUSTOMERS_COPY_BITMAP_34');

SEGMENT_NAME
--------------------------------------------------------------------------------
   SIZE_MB
----------
CUSTOMERS_COPY_BTREE_34
     .0625
```

Q4. Do as directed :

a. Create function based index on Employee table of HR schema. Function should be on salary attribute based on commission percentage

```
SQL> CREATE INDEX idx_emp_salary_comm ON hr.employees (salary * (1 + NVL(commission_pct, 0)));

Index created.
```

b. Find out list of employees having commission percentage less than 50000.

```
SQL> SELECT * FROM hr.employees WHERE salary * NVL(commission_pct, 0) < 50000;

EMPLOYEE_ID FIRST_NAME           LAST_NAME
----------- -------------------- -------------------------
EMAIL                     PHONE_NUMBER         HIRE_DATE JOB_ID         SALARY
------------------------- -------------------- --------- ---------- ----------
COMMISSION_PCT MANAGER_ID DEPARTMENT_ID
-------------- ---------- -------------
        100 Steven               King
SKING                     515.123.4567         17-JUN-03 AD_PRES         24000
                                       90

        101 Neena                Kochhar
NKOCHHAR                  515.123.4568         21-SEP-05 AD_VP           17000
                     100               90

EMPLOYEE_ID FIRST_NAME           LAST_NAME
----------- -------------------- -------------------------
EMAIL                     PHONE_NUMBER         HIRE_DATE JOB_ID         SALARY
------------------------- -------------------- --------- ---------- ----------
COMMISSION_PCT MANAGER_ID DEPARTMENT_ID
-------------- ---------- -------------

        102 Lex                  De Haan
LDEHAAN                   515.123.4569         13-JAN-01 AD_VP           17000
                     100               90

        103 Alexander            Hunold
AHUNOLD                   590.423.4567         03-JAN-06 IT_PROG          9000

EMPLOYEE_ID FIRST_NAME           LAST_NAME
----------- -------------------- -------------------------
EMAIL                     PHONE_NUMBER         HIRE_DATE JOB_ID         SALARY
------------------------- -------------------- --------- ---------- ----------
COMMISSION_PCT MANAGER_ID DEPARTMENT_ID
-------------- ---------- -------------
                     102               60

        104 Bruce                Ernst
BERNST                    590.423.4568         21-MAY-07 IT_PROG          6000
                     103               60

        105 David                Austin
```

c. Create function based index on employee name for Upper and lower function.

```
SQL> CREATE INDEX idx_emp_name_upper ON hr.employees (UPPER(last_name));

Index created.

SQL> CREATE INDEX idx_emp_name_lower ON hr.employees (LOWER(last_name));

Index created.
```

d. Create user table with attributes (UserId, UserName, Gender)

```
SQL> CREATE TABLE user_table (    UserId NUMBER PRIMARY KEY,    UserName VARCHAR2(100),
 Gender CHAR(1));

Table created.
```

e. Insert 10000 records in user table

```
SQL> BEGIN
  2      FOR i IN 1..10000 LOOP
  3          INSERT INTO USERS (UserId, UserName, Gender)
  4          VALUES (i, 'User' || i, CASE WHEN MOD(i,2) = 0 THEN 'M' ELSE 'F' END);
  5      END LOOP;
  6      COMMIT;
  7  END;
  8
  9  /

PL/SQL procedure successfully completed.
```

f. Build regular index on Username

```
SQL> CREATE INDEX idx_user_name
  2  ON USERS(UserName);

Index created.
```

g. Build function based index on user name based on Upper function

```
SQL> CREATE INDEX idx_user_name_upper
  2  ON USERS(UPPER(UserName));

Index created.
```

h. Compare the response time and comment.

```
SQL> SET TIMING ON;
SQL> SELECT * FROM USERS WHERE UserName = 'User5000';

    USERID
----------
USERNAME
--------------------------------------------------------------------------------
G
-
      5000
User5000
M


Elapsed: 00:00:00.00
SQL> SET TIMING OFF;
SQL>
```

1.Regular Index works well when searching exactly as stored.
2.Function-Based Index is helpful when using `UPPER()`, `LOWER()`, or calculations in `WHERE` clauses.
3.Without Function-Based Index, `UPPER(UserName) = 'USER5000'` results in a full table scan, slowing down performance.

Q5. Do as directed :
a. Create an IOT look_ups with the attributes (lookup_code, lookup_value, lookup_description).
b. Constraint: lookup_code should be primary key
c. lookup_description should be in overflow area.

```
SQL> CREATE TABLE LOOK_UPS (
  2       LOOKUP_CODE NUMBER PRIMARY KEY,
  3       LOOKUP_VALUE VARCHAR2(100),
  4       LOOKUP_DESCRIPTION VARCHAR2(500)
  5  ) ORGANIZATION INDEX
  6  PCTTHRESHOLD 20
  7  OVERFLOW;

Table created.

SQL>
```

Q6. Do as directed :
a. Create a Index Organized Table(IOT) emp_iot based on hr.employees

```
SQL> CREATE TABLE EMP_IOT (
  2       EMPLOYEE_ID NUMBER PRIMARY KEY,
  3       FIRST_NAME VARCHAR2(50),
  4       LAST_NAME VARCHAR2(50),
  5       EMAIL VARCHAR2(100),
  6       PHONE_NUMBER VARCHAR2(20),
  7       JOB_ID VARCHAR2(10),
  8       SALARY NUMBER,
  9       COMMISSION_PCT NUMBER,
 10       MANAGER_ID NUMBER,
 11       DEPARTMENT_ID NUMBER
 12   ) ORGANIZATION INDEX;

Table created.
```

b. Create a Index Organized Table(IOT) emp101_emp based on hr.employees. Place the column hiredate in overflow area.

```
SQL> CREATE TABLE EMP101_IOT (
  2      EMPLOYEE_ID NUMBER PRIMARY KEY,
  3      FIRST_NAME VARCHAR2(50),
  4      LAST_NAME VARCHAR2(50),
  5      EMAIL VARCHAR2(100),
  6      PHONE_NUMBER VARCHAR2(20),
  7      JOB_ID VARCHAR2(10),
  8      SALARY NUMBER,
  9      COMMISSION_PCT NUMBER,
 10      MANAGER_ID NUMBER,
 11      DEPARTMENT_ID NUMBER,
 12      HIRE_DATE DATE
 13  ) ORGANIZATION INDEX
 14  PCTTHRESHOLD 20
 15  OVERFLOW;

Table created.
```

c. Compare the timings of executing select all from employees,emp_iot, and emp101_iot. Comment on your observations.

`HR.EMPLOYEES` (Heap Table) → Slowest, full table scan increases disk I/O.
`EMP_IOT` (IOT Table) → Faster, rows indexed by primary key for efficient lookups.
`EMP101_IOT` (IOT with Overflow) → Fastest if `HIRE_DATE` is rarely accessed; slower if `HIRE_DATE` is frequently queried due to overflow access.