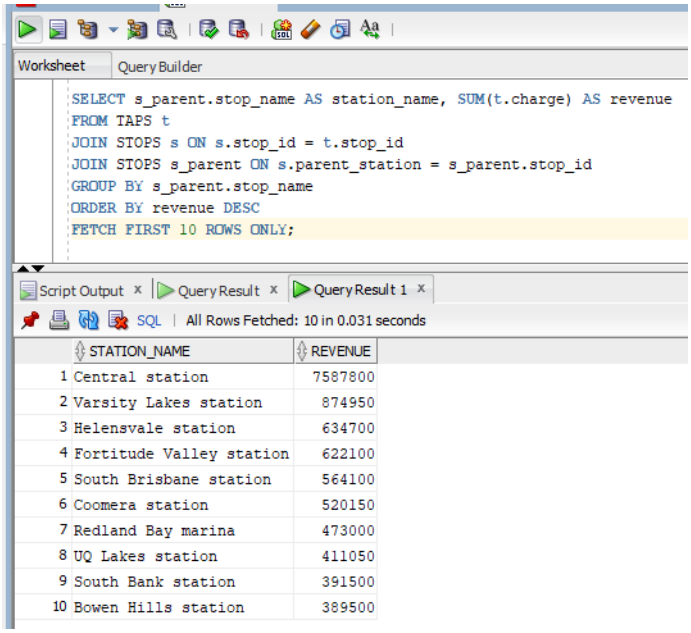
Name: Nguyen Van Quoc Chuong

Id: 47787810

INFS2200 Assignment 2 report

TASK 1: VIEWS

Task 1.1:



	STATION_NAME	REVENUE
1	Central station	7587800
2	Varsity Lakes station	874950
3	Helensvale station	634700
4	Fortitude Valley station	622100
5	South Brisbane station	564100
6	Coomera station	520150
7	Redland Bay marina	473000
8	UQ Lakes station	411050
9	South Bank station	391500
10	Bowen Hills station	389500

Task 1.2:

Worksheet | Query Builder

```
SELECT s_parent.stop_name AS station_name, SUM(t.charge) AS revenue
FROM TAPS t
JOIN STOPS s ON s.stop_id = t.stop_id
JOIN STOPS s_parent ON s.parent_station = s_parent.stop_id
JOIN CUSTOMERS c ON t.customer_id = c.customer_id
WHERE c."class" = 'Concession'
    AND (t.charge >= 350)
GROUP BY s_parent.stop_name
ORDER BY revenue DESC
FETCH FIRST 10 ROWS ONLY;
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 10 in 0.018 seconds

	STATION_NAME	REVENUE
1	Fortitude Valley station	546100
2	South Bank station	391500
3	Coomera station	295250
4	Varsity Lakes station	259350
5	Helensvale station	224100
6	Eumundi station	222200
7	Eight Mile Plains station	164150
8	UQ Lakes station	155050
9	King George Square bus station	80850
10	Upper Mt Gravatt station	58800

Task 1.3:

```
CREATE OR REPLACE VIEW V_STATION_BOARDINGS AS
SELECT
    s_parent.stop_name AS station_name,
    TO_CHAR(t."timestamp", 'Day') AS DOW,
    c."class" AS fare_class,
    COUNT(*) AS boardings
FROM TAPS t
JOIN STOPS s ON t.stop_id = s.stop_id
JOIN STOPS s_parent ON s.parent_station = s_parent.stop_id
JOIN CUSTOMERS c ON t.customer_id = c.customer_id
WHERE t.charge = 0
GROUP BY
    s_parent.stop_name,
    TO_CHAR(t."timestamp", 'D'),
    TO_CHAR("timestamp", 'Day'),
    c."class"
ORDER BY
    s_parent.stop_name,
    TO_NUMBER(TO_CHAR(t."timestamp", 'D')),
    c."class";
```

```
View V_STATION_BOARDINGS created.

Elapsed: 00:00:00.014
```

Top 10 stations

```
SELECT * FROM V_STATION_BOARDINGS
FETCH FIRST 10 ROWS ONLY;
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 10 in 0.039 seconds

	STATION_NAME	DOW	FARE_CLASS	BOARDINGS
1	Albion station	Monday	Adult	26
2	Albion station	Tuesday	Adult	43
3	Albion station	Wednesday	Adult	40
4	Albion station	Thursday	Adult	37
5	Albion station	Friday	Adult	34
6	Albion station	Saturday	Adult	8
7	Albion station	Sunday	Adult	3
8	Alderley station	Monday	Adult	27
9	Alderley station	Tuesday	Adult	40
10	Alderley station	Wednesday	Adult	41

Last 10 stations

```
SELECT * FROM V_STATION_BOARDINGS
ORDER BY ROWNUM DESC
FETCH FIRST 10 ROWS ONLY;
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 10 in 0.031 seconds

	STATION_NAME	DOW	FARE_CLASS	BOARDINGS
1	Zillmere station	Sunday	Adult	4
2	Zillmere station	Saturday	Adult	6
3	Zillmere station	Friday	Adult	29
4	Zillmere station	Thursday	Adult	44
5	Zillmere station	Wednesday	Adult	37
6	Zillmere station	Tuesday	Adult	43
7	Zillmere station	Monday	Adult	31
8	Wooloowin station	Sunday	Adult	2
9	Wooloowin station	Saturday	Adult	12
10	Wooloowin station	Friday	Adult	29

Task 1.4:

```sql
CREATE MATERIALIZED VIEW MV_STATION_BOARDINGS
AS
SELECT
    s_parent.stop_name AS station_name,
    TO_CHAR(t."timestamp", 'Day') AS DOW,
    c."class" AS fare_class,
    COUNT(*) AS boardings
FROM TAPS t
JOIN STOPS s ON t.stop_id = s.stop_id
JOIN STOPS s_parent ON s.parent_station = s_parent.stop_id
JOIN CUSTOMERS c ON t.customer_id = c.customer_id
WHERE t.charge = 0
GROUP BY
    s_parent.stop_name,
    TO_CHAR(t."timestamp", 'D'),
    TO_CHAR("timestamp", 'Day'),
    c."class"
ORDER BY
    s_parent.stop_name,
    TO_NUMBER(TO_CHAR(t."timestamp", 'D')),
    c."class";
```

```
Materialized view MV_STATION_BOARDINGS created.

Elapsed: 00:00:00.121
```

Task 1.5:

V_STATION_BOARDINGS:

```
EXPLAIN PLAN FOR
SELECT * FROM V_STATION_BOARDINGS;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Script Output | Query Result | Query Result 1 | SQL | All Rows Fetched: 29 in 0.092 seconds

PLAN_TABLE_OUTPUT

```
Plan hash value: 3376659461

-----------------------------------------------------------------------------------------------
| Id  | Operation              | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                    |   227 | 13166 |   158   (2)| 00:00:01 |
|   1 |  VIEW                  | V_STATION_BOARDINGS |   227 | 13166 |   158   (2)| 00:00:01 |
|   2 |   SORT ORDER BY        |                    |   227 | 21792 |   158   (2)| 00:00:01 |
|   3 |    HASH GROUP BY       |                    |   227 | 21792 |   158   (2)| 00:00:01 |
|*  4 |     HASH JOIN          |                    |   227 | 21792 |   156   (1)| 00:00:01 |
|   5 |      TABLE ACCESS FULL | CUSTOMERS          |   100 |  2000 |     3   (0)| 00:00:01 |
|*  6 |      HASH JOIN         |                    |   227 | 17252 |   153   (1)| 00:00:01 |
|*  7 |       HASH JOIN        |                    |   580 | 34220 |    50   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL| STOPS             | 13491 |  237K|    25   (0)| 00:00:01 |
|   9 |        TABLE ACCESS FULL| STOPS             | 13491 |  540K|    25   (0)| 00:00:01 |
|* 10 |       TABLE ACCESS FULL | TAPS              |  5285 | 89845 |   103   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   4 - access("T"."CUSTOMER_ID"="C"."CUSTOMER_ID")
   6 - access("T"."STOP_ID"="S"."STOP_ID")
   7 - access("S"."PARENT_STATION"="S_PARENT"."STOP_ID")
  10 - filter("T"."CHARGE"=0)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
Explained.

Elapsed: 00:00:00.023
```

MV_STATION_BOARDINGS:

```
EXPLAIN PLAN FOR
SELECT * FROM MV_STATION_BOARDINGS;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Script Output | Query Result | Query Result 1 | Query Result 2 | SQL | All Rows Fetched: 8 in 0.021 seconds

PLAN_TABLE_OUTPUT

```
Plan hash value: 1663832714

-----------------------------------------------------------------------------------------------
| Id  | Operation             | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                    |   590 | 23600 |     4   (0)| 00:00:01 |
|   1 |  MAT_VIEW ACCESS FULL| MV_STATION_BOARDINGS |   590 | 23600 |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

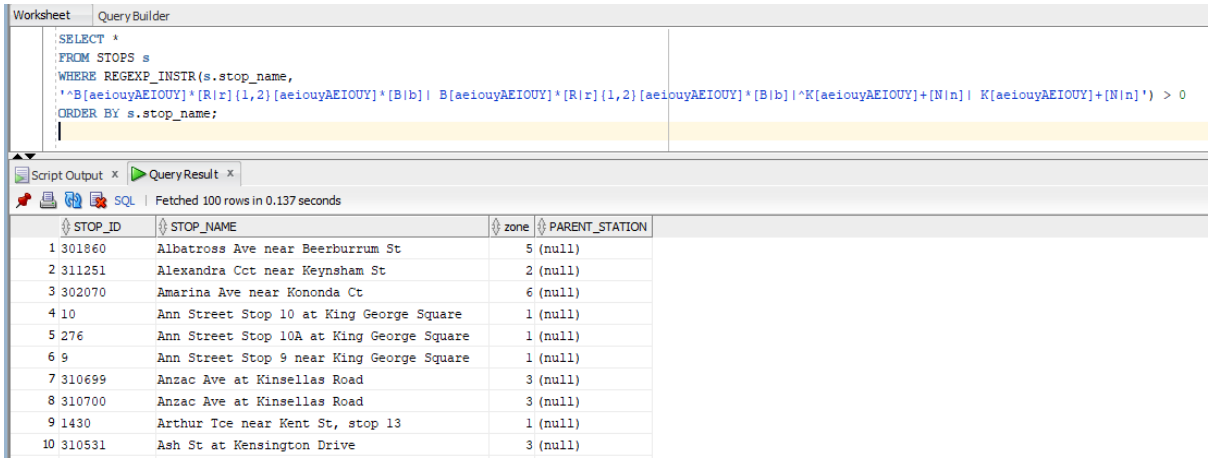
```
Explained.

Elapsed: 00:00:00.003
```

The cost for this query is significantly lower compared to the previous query you provided. In this plan, the cost is only 4, while in the previous plan, it was 158. This indicates that accessing the data

from the Materialized View is much more efficient in terms of resource usage and performance. The Elapsed time is also lower a bit.

TASK 2: Indexes

Task 2.1:



```
SELECT *
FROM STOPS s
WHERE REGEXP_INSTR(s.stop_name,
'^B[aeiouyAEIOUY]*[R|r]{1,2}[aeiouyAEIOUY]*[B|b]| B[aeiouyAEIOUY]*[R|r]{1,2}[aeiouyAEIOUY]*[B|b]|^K[aeiouyAEIOUY]+[N|n]| K[aeiouyAEIOUY]+[N|n]') > 0
ORDER BY s.stop_name;
```

Script Output | Query Result | SQL | Fetched 100 rows in 0.137 seconds

	STOP_ID	STOP_NAME	zone	PARENT_STATION
1	301860	Albatross Ave near Beerburrum St	5	(null)
2	311251	Alexandra Cct near Keynsham St	2	(null)
3	302070	Amarina Ave near Kononda Ct	6	(null)
4	10	Ann Street Stop 10 at King George Square	1	(null)
5	276	Ann Street Stop 10A at King George Square	1	(null)
6	9	Ann Street Stop 9 near King George Square	1	(null)
7	310699	Anzac Ave at Kinsellas Road	3	(null)
8	310700	Anzac Ave at Kinsellas Road	3	(null)
9	1430	Arthur Tce near Kent St, stop 13	1	(null)
10	310531	Ash St at Kensington Drive	3	(null)

Last 10 rows

443	311544	Welsby Pde near Kangaroo Ave	4	(null)
444	311549	Welsby Pde near Kangaroo Ave	4	(null)
445	11202	Whites St at Kenna Street, stop 46	2	(null)
446	319700	Whytecliffe Pde at King Street	3	(null)
447	4468	Wilgarning St at Kingaroy Street, stop 3	2	(null)
448	4469	Wilgarning St at Kingaroy Street, stop 3	2	(null)
449	312631	Wimbledon Dr near Keneally Ct	3	(null)
450	6212	Wynnum Rd at Kianawah Park, stop 41	2	(null)
451	6213	Wynnum Rd at Kianawah Park, stop 41	2	(null)
452	6039	Wynnum Rd at Kianawah Rd, stop 46/49	2	(null)
453	6038	Wynnum Rd at Kianawah Rd, stop 49/46	2	(null)

Task 2.2:

```
CREATE OR REPLACE FUNCTION MyFunctionBasedIndex(p_string IN VARCHAR2) RETURN NUMBER DETERMINISTIC
IS
BEGIN
  RETURN REGEXP_INSTR(p_string, '^B[aeiouyAEIOUY]*[R|r]{1,2}[aeiouyAEIOUY]*[B|b]| B[aeiouyAEIOUY]*[R|r]{1,2}[aeiouyAEIOUY]*[B|b]|^K[aeiouyAEIOUY]+[N|n]| K[aeiouyAEIOUY]+[N|n]');
END;
/
CREATE INDEX IDX_PINK ON STOPS(MyFunctionBasedIndex(stop_name));
```

```
Function MYFUNCTIONBASEDINDEX compiled

Elapsed: 00:00:00.006

Index IDX_PINK created.

Elapsed: 00:00:00.330
```

Task 2.3:

Execution plan WITHOUT index:

```
EXPLAIN PLAN FOR
SELECT *
FROM STOPS s
WHERE REGEXP_INSTR(s.stop_name,
'^B[aeiouyAEIOUY]*[R|r]{1,2}[aeiouyAEIOUY]*[B|b]| B[aeiouyAEIOUY]*[R|r]{1,2}[aeiouyAEIOUY]*[B|b]|^K[aeiouyAEIOUY]+[N|n]| K[aeiouyAEIOUY]+[N|n]') > 0
```

Script Output | Query Result

SQL | All Rows Fetched: 20 in 0.149 seconds

PLAN_TABLE_OUTPUT

```
Plan hash value: 52011983

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |   476 | 29988 |    26   (4)| 00:00:01 |
|   1 |  SORT ORDER BY     |       |   476 | 29988 |    26   (4)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| STOPS |   476 | 29988 |    25   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   2 - filter( REGEXP_INSTR ("S"."STOP_NAME",'^B[aeiouyAEIOUY]*[R|r]{1,2
              }[aeiouyAEIOUY]*[B|b]| B[aeiouyAEIOUY]*[R|r]{1,2}[aeiouyAEIOUY]*[B|b]|^K
              [aeiouyAEIOUY]+[N|n]| K[aeiouyAEIOUY]+[N|n]')>0)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
Explained.

Elapsed: 00:00:00.084
```

- The row estimate difference between the SELECT statement (453) and the explain plan (476) is attributed to Oracle's dynamic sampling, which refines cost estimates. The actual number of rows returned by the query (453) is the accurate figure to consider. The estimate in the explain plan (476) is an intermediate step and may not precisely match the actual result.

Execution plan WITH index:

```
EXPLAIN PLAN FOR
SELECT /*+ INDEX(s IDX_PINK) */ *
FROM STOPS s
WHERE MyFunctionBasedIndex(s.stop_name) > 0
ORDER BY s.stop_name;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Script Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3

SQL | All Rows Fetched: 19 in 0.034 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 2974718955

-----------------------------------------------------------------------------------------------
| Id  | Operation                           | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |           |   453 | 34428 |    12   (9)| 00:00:01 |
|   1 |  SORT ORDER BY                      |           |   453 | 34428 |    12   (9)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| STOPS     |   453 | 34428 |    11   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                 | IDX_PINK  |   453 |       |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   3 - access("S4778781"."MYFUNCTIONBASEDINDEX"("STOP_NAME")>0)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
Explained.

Elapsed: 00:00:00.265
```

- We can see that, with index, the cost was reduced to 12, but the CPU% was 9%. The Elapsed time and Bytes with index are higher. Generally, the first query without the function-based index perform a full table scan, while the second query, which uses a function-based index, is much more efficient in terms of cost, as it utilizes the index to perform the regular expression matching. This can significantly improve query performance when dealing with large datasets, as it avoids scanning the entire table.

Task 2.4:

Worksheet | Query Builder

```
SELECT COUNT(*)
FROM TAPS t1
WHERE EXISTS (
    SELECT charge
    FROM TAPS t2
    WHERE t1.customer_id = t2.customer_id
    AND t1.stop_id = t2.stop_id
    AND t1.charge = t2.charge
    AND t2.charge > 0
    GROUP BY t2.customer_id, t2.stop_id, t2.charge
    HAVING COUNT(*) >= 201
);
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 1 in 10.319 seconds

	COUNT(*)
1	15969

Task 2.5:

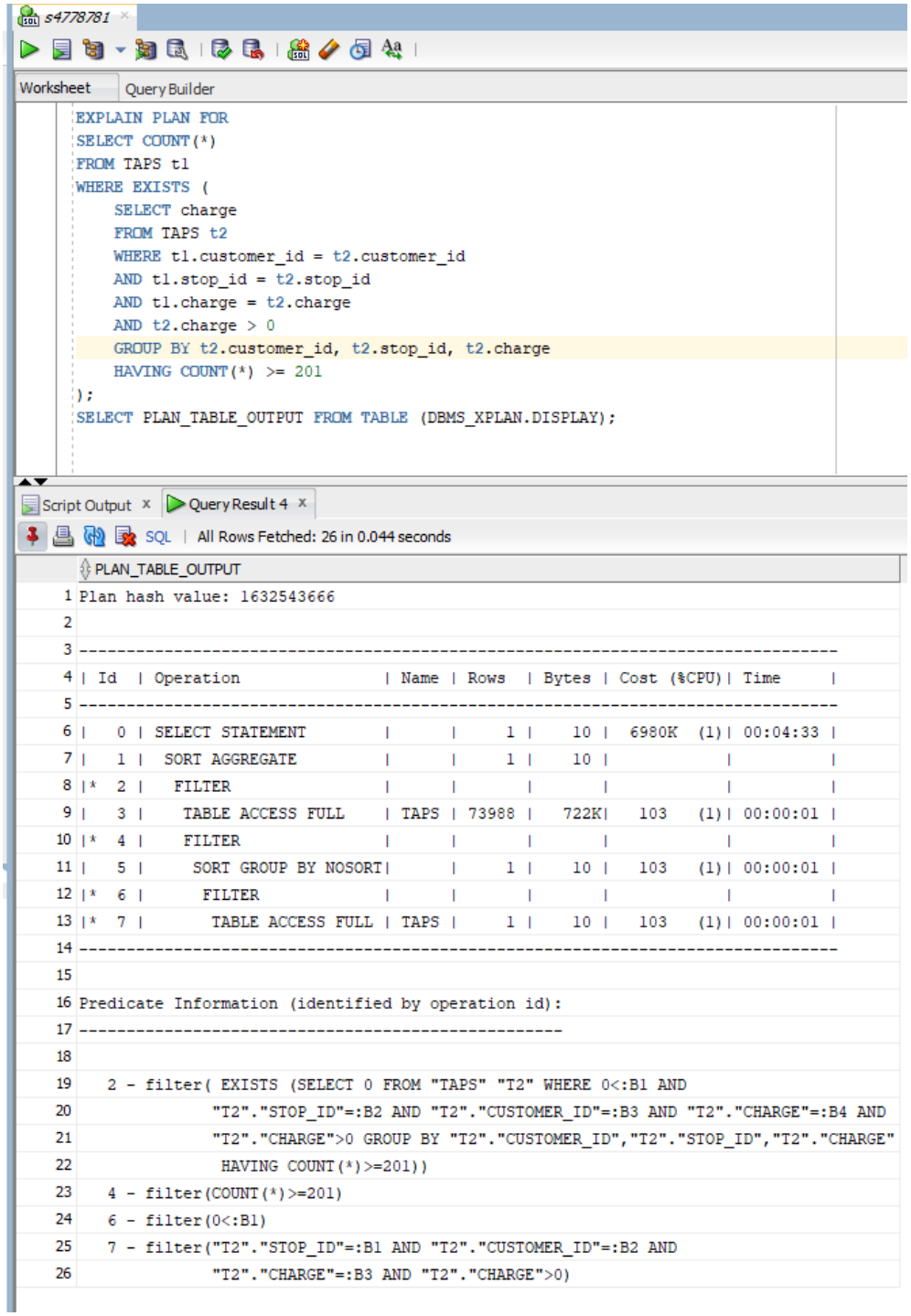
```
CREATE BITMAP INDEX BIDX_CUST_ID
ON TAPS (customer_id);

CREATE BITMAP INDEX BIDX_STOP_ID
ON TAPS (stop_id);

CREATE BITMAP INDEX BIDX_CHARGE
ON TAPS (charge);
```

```
INDEX BIDX_CUST_ID created.

Elapsed: 00:00:00.026

INDEX BIDX_STOP_ID created.

Elapsed: 00:00:00.022

INDEX BIDX_CHARGE created.

Elapsed: 00:00:00.022
```

Task 2.6:

Before using Bitmap Index:

```
Explained.

Elapsed: 00:00:00.013
```



```
EXPLAIN PLAN FOR
SELECT COUNT(*)
FROM TAPS t1
WHERE EXISTS (
    SELECT charge
    FROM TAPS t2
    WHERE t1.customer_id = t2.customer_id
    AND t1.stop_id = t2.stop_id
    AND t1.charge = t2.charge
    AND t2.charge > 0
    GROUP BY t2.customer_id, t2.stop_id, t2.charge
    HAVING COUNT(*) >= 201
);
SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);
```

```
Plan hash value: 1632543666

-------------------------------------------------------------------------------------
| Id  | Operation                | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |      |     1 |    10 |  6980K  (1)| 00:04:33 |
|   1 |  SORT AGGREGATE          |      |     1 |    10 |            |          |
|*  2 |   FILTER                 |      |       |       |            |          |
|   3 |    TABLE ACCESS FULL     | TAPS | 73988 |   722K|   103   (1)| 00:00:01 |
|*  4 |    FILTER                |      |       |       |            |          |
|   5 |     SORT GROUP BY NOSORT |      |     1 |    10 |   103   (1)| 00:00:01 |
|*  6 |      FILTER              |      |       |       |            |          |
|*  7 |       TABLE ACCESS FULL  | TAPS |     1 |    10 |   103   (1)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   2 - filter( EXISTS (SELECT 0 FROM "TAPS" "T2" WHERE 0<:B1 AND
              "T2"."STOP_ID"=:B2 AND "T2"."CUSTOMER_ID"=:B3 AND "T2"."CHARGE"=:B4 AND
              "T2"."CHARGE">0 GROUP BY "T2"."CUSTOMER_ID","T2"."STOP_ID","T2"."CHARGE"
               HAVING COUNT(*)>=201))
   4 - filter(COUNT(*)>=201)
   6 - filter(0<:B1)
   7 - filter("T2"."STOP_ID"=:B1 AND "T2"."CUSTOMER_ID"=:B2 AND
              "T2"."CHARGE"=:B3 AND "T2"."CHARGE">0)
```

After using Bitmap Index:

```
Explained.

Elapsed: 00:00:00.016
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 178317411

-------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |              |     1 |    10 |   189K  (1)| 00:00:08 |
|   1 |  SORT AGGREGATE                    |              |     1 |    10 |            |          |
|*  2 |   FILTER                           |              |       |       |            |          |
|   3 |    TABLE ACCESS FULL               | TAPS         | 73988 |  722K|   103   (1)| 00:00:01 |
|*  4 |    FILTER                          |              |       |       |            |          |
|   5 |     SORT GROUP BY NOSORT           |              |     1 |    10 |     3   (0)| 00:00:01 |
|*  6 |      FILTER                        |              |       |       |            |          |
|*  7 |       TABLE ACCESS BY INDEX ROWID  | TAPS         |     1 |    10 |     3   (0)| 00:00:01 |
|   8 |        BITMAP CONVERSION TO ROWIDS|              |       |       |            |          |
|   9 |         BITMAP AND                 |              |       |       |            |          |
|* 10 |          BITMAP INDEX SINGLE VALUE| BIDX_STOP_ID |       |       |            |          |
|* 11 |          BITMAP INDEX SINGLE VALUE| BIDX_CUST_ID |       |       |            |          |
-------------------------------------------------------------------------------------------------

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

   2 - filter( EXISTS (SELECT 0 FROM "TAPS" "T2" WHERE 0<:B1 AND "T2"."STOP_ID"=:B2 AND
              "T2"."CUSTOMER_ID"=:B3 AND "T2"."CHARGE"=:B4 AND "T2"."CHARGE">0 GROUP BY
              "T2"."CUSTOMER_ID","T2"."STOP_ID","T2"."CHARGE" HAVING COUNT(*)>=201))
   4 - filter(COUNT(*)>=201)
   6 - filter(0<:B1)
   7 - filter("T2"."CHARGE"=:B1 AND "T2"."CHARGE">0)
  10 - access("T2"."STOP_ID"=:B1)
  11 - access("T2"."CUSTOMER_ID"=:B1)
```

The bitmap index greatly enhances efficiency. In the two scenarios, the SELECT statement results in a cost of 6980K without the bitmap index and only 189K with it. Since the elapsed time varies, I rely solely on the consistent cost metric to evaluate its effectiveness.

TASK 3: EXECUTION PLANS

Task 3.1.a:

```
ANALYZE INDEX PK_TAPID VALIDATE STRUCTURE;
SELECT Height
FROM INDEX_STATS;
```

	HEIGHT
1	2

The height of B+ tree is 2.

Task 3.1.b:

```
SELECT LF_BLKS
FROM INDEX_STATS;
```

	LF_BLKS
1	138

The number of leaf blocks is 138.

Task 3.1.c:

```
SELECT BLOCKS
FROM USER_TABLES
WHERE TABLE_NAME = 'TAPS';
```

	BLOCKS
1	370

Task 3.2:

```
ALTER SESSION SET OPTIMIZER_MODE = RULE;
EXPLAIN PLAN FOR
SELECT * FROM TAPS
WHERE tap_id > 100;
SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 18 in 0.044 seconds

```
PLAN_TABLE_OUTPUT
1  Plan hash value: 2967462872
2
3  ------------------------------------------------
4  | Id  | Operation                    | Name     |
5  ------------------------------------------------
6  |   0 | SELECT STATEMENT             |          |
7  |   1 |  TABLE ACCESS BY INDEX ROWID| TAPS     |
8  |*  2 |   INDEX RANGE SCAN           | PK_TAPID |
9  ------------------------------------------------
10
11 Predicate Information (identified by operation id):
12 ---------------------------------------------------
13
14    2 - access("TAP_ID">100)
15
16 Note
17 -----
18    - rule based optimizer used (consider using cbo)
```

Elapsed: 00:00:00.444

Analyzing the query execution:

- INDEX (RANGE SCAN): The PK_TAPID index is employed to scan through a range of values, specifically addressing the conditions specified in the WHERE clause.
- TABLE ACCESS BY INDEX ROWID: Rows are retrieved using the index, and it's important to note that a full table scan, as seen in the previous example, is avoided.
- The SELECT statement processes and returns rows that meet the criteria outlined in the WHERE clause.

Task 3.3:

```
ALTER SESSION SET OPTIMIZER_MODE = CHOOSE;
EXPLAIN PLAN FOR
SELECT * FROM TAPS
WHERE tap_id > 100;
SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);
```

```
Elapsed: 00:00:00.373
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 1703577251

-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      | 73889 |  1515K|   103   (1)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| TAPS | 73889 |  1515K|   103   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - filter("TAP_ID">100)
```

Query processing steps:

- The query scans the entire "TAPS" table.
- For each row, it checks the value in the "TAP_ID" column to see if it's greater than 100.
- Rows that meet this condition are included in the result set.

Key Difference: The main difference is in the access method. The 3.2 plan used an index range scan, while the 3.3 plan uses a full table scan. The choice of access method can significantly impact query performance, where an index scan is more selective and efficient than a full table scan.

Task 3.4:

```
Elapsed: 00:00:00.381
```

```
ALTER SESSION SET OPTIMIZER_MODE = CHOOSE;
EXPLAIN PLAN FOR
SELECT * FROM TAPS
WHERE tap_id > 73900;
SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);
```

```
cript Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4
SQL | All Rows Fetched: 14 in 0.02 seconds

PLAN_TABLE_OUTPUT
Plan hash value: 2404732744

-----------------------------------------------------------------------------------------------
| Id  | Operation                           | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |          |    88 |  1848 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| TAPS     |    88 |  1848 |     3   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | PK_TAPID |    88 |       |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   2 - access("TAP_ID">73900)
```

Query Processing Steps:

- Initially, an INDEX (UNIQUE SCAN) is performed on the "PK_TAPID" index, uniquely retrieving specific rows that meet the WHERE clause conditions.

- Subsequently, a TABLE ACCESS BY INDEX ROWID BATCHED operation locates rows using the index efficiently.
- Finally, the SELECT statement delivers the rows that satisfy the WHERE clause criteria.

Key Difference:

- Task 3.3, where "tap_id" needs to be greater than 100, the condition is not very selective. Oracle's query optimizer opts for a Table Full Scan.
- Conversely, in Task 3.4, the condition is tap_id > 73900, which is highly selective. Oracle's query optimizer recognizes this and chooses the more efficient Index Range Scan. It starts at the specified range point (73900) and scans through the index, retrieving rows that match the condition within the range.

Task 3.5:

```
Elapsed: 00:00:00.370
```

```
ALTER SESSION SET OPTIMIZER_MODE = CHOOSE;
EXPLAIN PLAN FOR
SELECT * FROM TAPS
WHERE tap_id = 10000;
SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);
```

ript Output | Query Result 4 | Query Result 5

SQL | All Rows Fetched: 14 in 0.02 seconds

```
PLAN_TABLE_OUTPUT
1  Plan hash value: 571498902
2
3  ------------------------------------------------------------------------------------------
4  | Id  | Operation                    | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
5  ------------------------------------------------------------------------------------------
6  |   0 | SELECT STATEMENT             |          |     1 |    21 |     2   (0)| 00:00:01 |
7  |   1 |  TABLE ACCESS BY INDEX ROWID| TAPS     |     1 |    21 |     2   (0)| 00:00:01 |
8  |*  2 |   INDEX UNIQUE SCAN          | PK_TAPID |     1 |       |     1   (0)| 00:00:01 |
9  ------------------------------------------------------------------------------------------
10
11 Predicate Information (identified by operation id):
12 ---------------------------------------------------
13
14   2 - access("TAP_ID"=10000)
```

Query Processing Steps:

- Firstly, an INDEX (UNIQUE SCAN) is executed, utilizing the "PK_TAPID" index for evaluating the WHERE clause condition. This results in the retrieval of a single row identifier from the index.
- Subsequently, a TABLE ACCESS BY INDEX ROWID operation is performed to locate rows using the index (Hashing), avoiding a full table scan.
- Finally, the SELECT statement returns rows that meet the WHERE clause conditions.

Key difference:

- 3.3 plan scans full table while 3.5 using unique scan (or hashing index) for executing operation “=”. That’s why the number of bytes and rows are much smaller than 3.3.