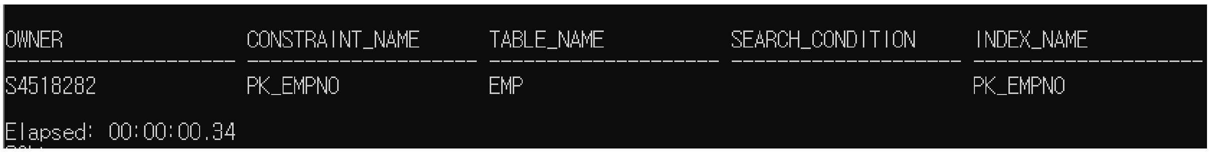
Task 1 – *Constraints*

1.1 Original Constraints

1.1.1 EMP

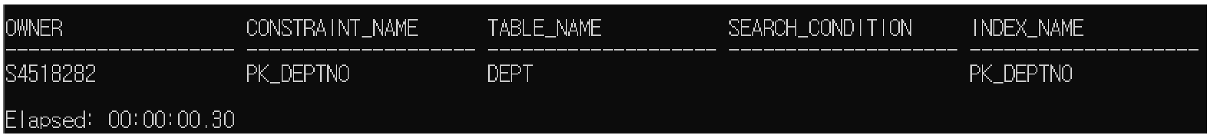
SELECT OWNER, CONSTRAINT_NAME, TALBE_NAME, SEARCH_CONDITION,
INDEX_NAME FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'EMP';



```
OWNER                CONSTRAINT_NAME      TABLE_NAME           SEARCH_CONDITION     INDEX_NAME
-------------------- -------------------- -------------------- -------------------- --------------------
S4518282             PK_EMPNO             EMP                                       PK_EMPNO

Elapsed: 00:00:00.34
```

1.1.2 DEPT

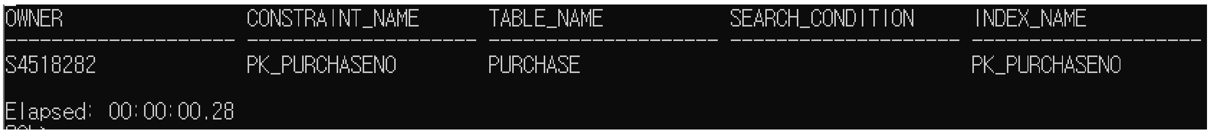
SELECT OWNER, CONSTRAINT_NAME, TALBE_NAME, SEARCH_CONDITION,
INDEX_NAME FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'DEPT'



```
OWNER                CONSTRAINT_NAME      TABLE_NAME           SEARCH_CONDITION     INDEX_NAME
-------------------- -------------------- -------------------- -------------------- --------------------
S4518282             PK_DEPTNO            DEPT                                      PK_DEPTNO

Elapsed: 00:00:00.30
```

1.1.3 PURCHASE

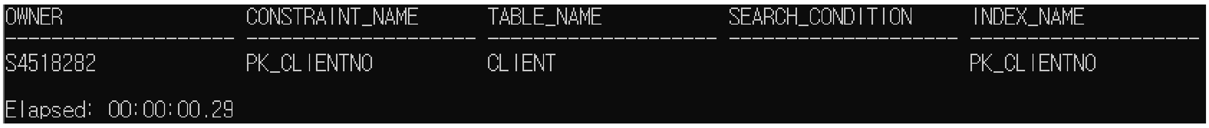
SELECT OWNER, CONSTRAINT_NAME, TALBE_NAME, SEARCH_CONDITION,
INDEX_NAME FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'PURCHASE';



```
OWNER                CONSTRAINT_NAME      TABLE_NAME           SEARCH_CONDITION     INDEX_NAME
-------------------- -------------------- -------------------- -------------------- --------------------
S4518282             PK_PURCHASENO        PURCHASE                                  PK_PURCHASENO

Elapsed: 00:00:00.28
```

1.1.4 CLIENT

SELECT OWNER, CONSTRAINT_NAME, TALBE_NAME, SEARCH_CONDITION,
INDEX_NAME FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'CLIENT'

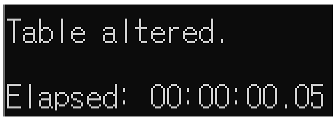


```
OWNER                CONSTRAINT_NAME      TABLE_NAME           SEARCH_CONDITION     INDEX_NAME
-------------------- -------------------- -------------------- -------------------- --------------------
S4518282             PK_CLIENTNO          CLIENT                                    PK_CLIENTNO

Elapsed: 00:00:00.29
```

1.2 Creation of missing constraints

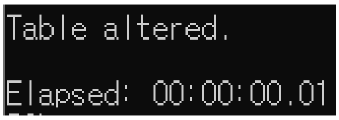
1.2.1 EMP CONSTRAINTS

ALTER TABLE EMP ADD CONSTRAINT **CK_ENAME**
CHECK (ENAME IS NOT NULL);



```
Table altered.

Elapsed: 00:00:00.05
```

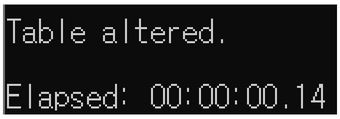
ALTER TABLE EMP ADD CONSTRAINT
FK_DEPTNO FOREIGN KEY (DEPTNO)
REFERENCES DEPT (DEPTNO);



```
Table altered.

Elapsed: 00:00:00.01
```

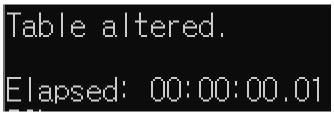
1.2.2 DEPT CONSTRAINTS

ALTER TABLE DEPT ADD CONSTRAINT **UN_DNAME** UNIQUE (DNAME);



```
Table altered.

Elapsed: 00:00:00.14
```

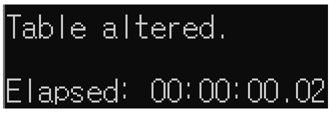
ALTER TABLE DEPT ADD CONSTRAINT **CK_DNAME** CHECK (DNAME IS NOT NULL);



```
Table altered.

Elapsed: 00:00:00.01
```

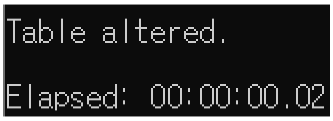
1.2.3 PURCHASE CONSTRAINTS

ALTER TABLE PURCHASE ADD CONSTRAINT **CK_RECEIPTNO** CHECK (RECEIPTNO IS NOT NULL);



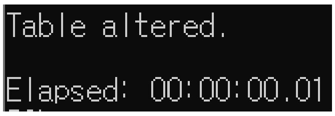
```
Table altered.

Elapsed: 00:00:00.02
```

ALTER TABLE PURCHASE ADD CONSTRAINT **CK_AMOUNT** CHECK (AMOUNT IS NOT NULL);



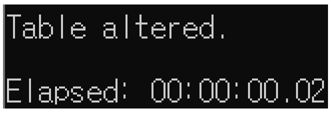
```
Table altered.

Elapsed: 00:00:00.02
```

ALTER TABLE PURCHASE ADD CONSTRAINT **CK_SERVICETYPE** CHECK (SERVICETYPE IN (‘Training’, ‘Data Recovery’, ‘Consultation’, ‘Software Installation’, ‘Software Repair’));



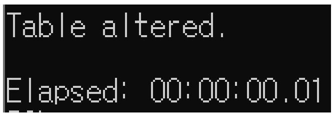
```
Table altered.

Elapsed: 00:00:00.01
```

ALTER TABLE PURCHASE ADD CONSTRAINT **CK_PAYMENTTYPE** CHECK (PAYMENTTPYE IN (‘Debit’, ‘Cash’, ‘Credit’));



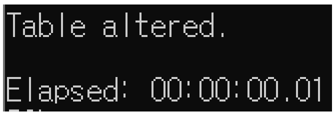
```
Table altered.

Elapsed: 00:00:00.02
```

ALTER TABLE PURCHASE ADD CONSTRAINT **CK_GST** CHECK (GST IN (‘Yes’, ‘No’));



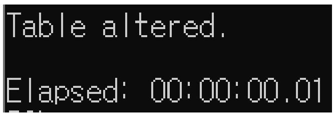
```
Table altered.

Elapsed: 00:00:00.01
```

ALTER TABLE PURCHASE ADD CONSTRAINT **FK_EMPNO** FOREIGN KEY (SERVEDBY) REFERENCES EMP (EMPNO);



```
Table altered.

Elapsed: 00:00:00.01
```

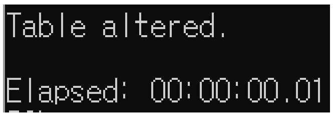
ALTER TABLE PURCHASE ADD CONSTRAINT **FK_CLIENTNO** FOREIGN KEY (CLIENTNO) REFERENCES CLIENT (CLIENTNO);



```
Table altered.

Elapsed: 00:00:00.01
```

1.2.4 CLIENT CONSTRAINTS

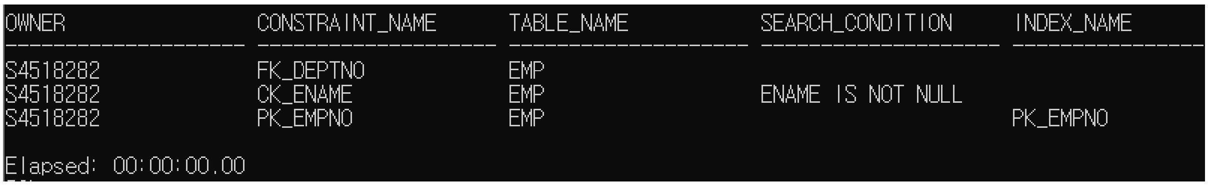
ALTER TABLE CLIENT ADD CONSTRAINT **CK_CNAME** CHECK (CNAME IS NOT NULL);



```
Table altered.

Elapsed: 00:00:00.01
```

1.2.5 EMP

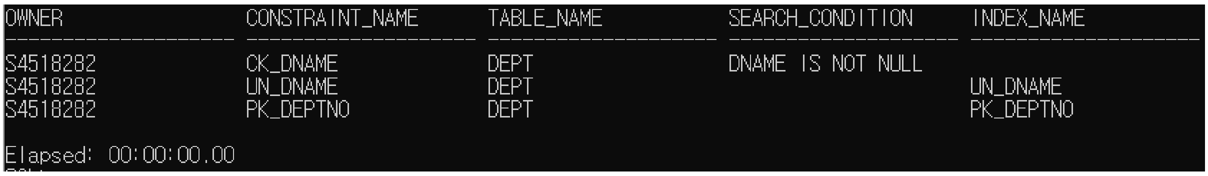
SELECT OWNER, CONSTRAINT_NAME, TALBE_NAME, SEARCH_CONDITION, INDEX_NAME FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'EMP';



```
OWNER                CONSTRAINT_NAME      TABLE_NAME           SEARCH_CONDITION     INDEX_NAME
-------------------- -------------------- -------------------- -------------------- ----------------
S4518282             FK_DEPTNO            EMP
S4518282             CK_ENAME             EMP                  ENAME IS NOT NULL
S4518282             PK_EMPNO             EMP                                       PK_EMPNO

Elapsed: 00:00:00.00
```

1.2.6 DEPT

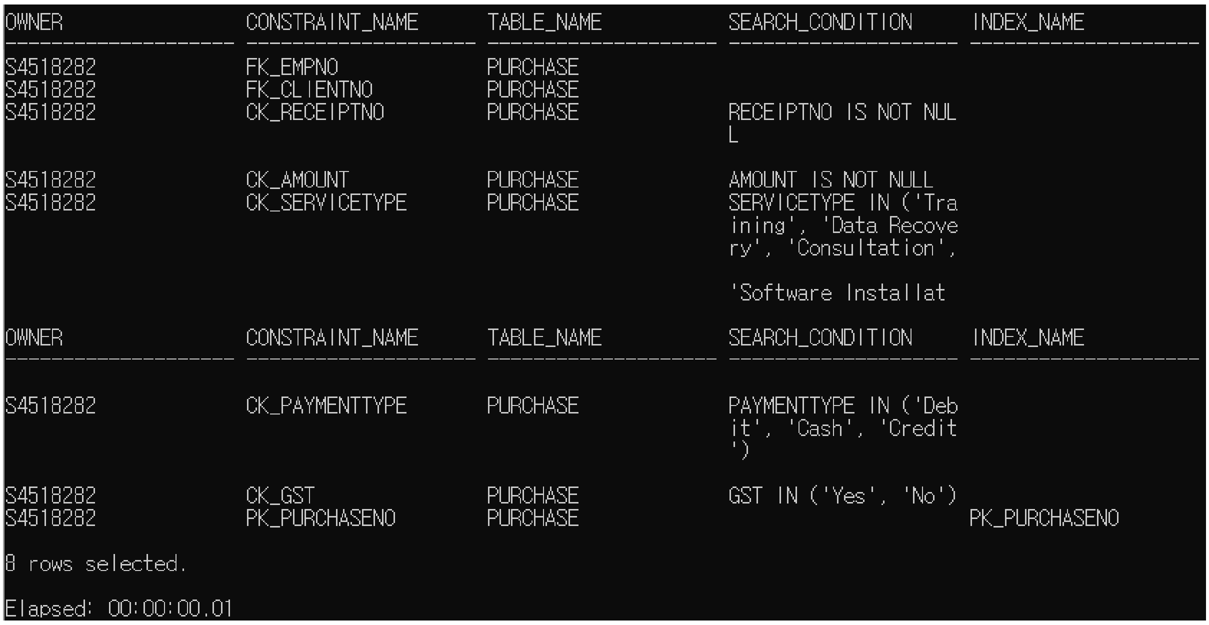
SELECT OWNER, CONSTRAINT_NAME, TALBE_NAME, SEARCH_CONDITION, INDEX_NAME FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'DEPT'



```
OWNER                CONSTRAINT_NAME      TABLE_NAME           SEARCH_CONDITION     INDEX_NAME
-------------------- -------------------- -------------------- -------------------- --------------------
S4518282             CK_DNAME             DEPT                 DNAME IS NOT NULL
S4518282             UN_DNAME             DEPT                                      UN_DNAME
S4518282             PK_DEPTNO            DEPT                                      PK_DEPTNO

Elapsed: 00:00:00.00
```

1.2.7 PURCHASE

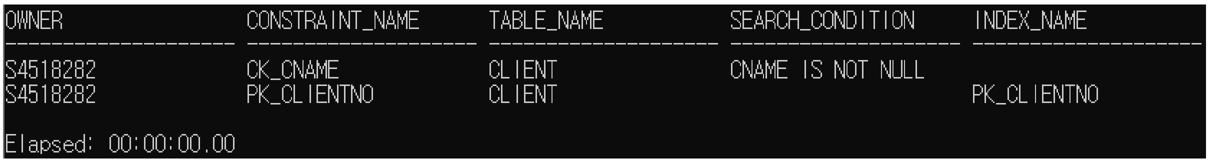
SELECT OWNER, CONSTRAINT_NAME, TALBE_NAME, SEARCH_CONDITION, INDEX_NAME FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'PURCHASE';



```
OWNER                CONSTRAINT_NAME      TABLE_NAME           SEARCH_CONDITION     INDEX_NAME
-------------------- -------------------- -------------------- -------------------- --------------------
S4518282             FK_EMPNO             PURCHASE
S4518282             FK_CLIENTNO          PURCHASE
S4518282             CK_RECEIPTNO         PURCHASE             RECEIPTNO IS NOT NUL
                                                               L

S4518282             CK_AMOUNT            PURCHASE             AMOUNT IS NOT NULL
S4518282             CK_SERVICETYPE       PURCHASE             SERVICETYPE IN ('Tra
                                                               ining', 'Data Recove
                                                               ry', 'Consultation',

                                                               'Software Installat

OWNER                CONSTRAINT_NAME      TABLE_NAME           SEARCH_CONDITION     INDEX_NAME
-------------------- -------------------- -------------------- -------------------- --------------------

S4518282             CK_PAYMENTTYPE       PURCHASE             PAYMENTTYPE IN ('Deb
                                                               it', 'Cash', 'Credit
                                                               ')

S4518282             CK_GST               PURCHASE             GST IN ('Yes', 'No')
S4518282             PK_PURCHASENO        PURCHASE                                  PK_PURCHASENO

8 rows selected.

Elapsed: 00:00:00.01
```

1.2.8 CLIENT

SELECT OWNER, CONSTRAINT_NAME, TALBE_NAME, SEARCH_CONDITION, INDEX_NAME FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'CLIENT'

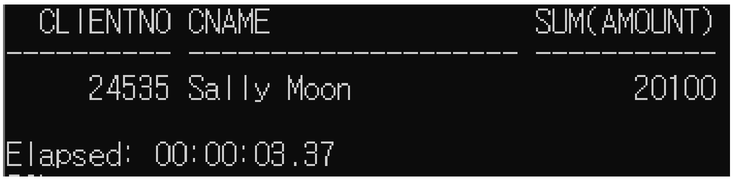


```
OWNER                CONSTRAINT_NAME      TABLE_NAME           SEARCH_CONDITION     INDEX_NAME
-------------------- -------------------- -------------------- -------------------- --------------------
S4518282             CK_CNAME             CLIENT               CNAME IS NOT NULL
S4518282             PK_CLIENTNO          CLIENT                                    PK_CLIENTNO

Elapsed: 00:00:00.00
```

Task 2 – *Triggers*

2.1 Top Client

```
SELECT C.CLIENTNO, CNAME, SUM(AMOUNT) FROM CLIENT C, PURCAHSE P
WHERE C.CLIENTNO = P.CLIENTNO
GROUP BY C.CLIENTNO, CNAME
HAVING SUM(AMOUNT) >= ALL(
        SELECT SUM(AMOUNT) FROM CLIENT C, PURCHASE P
        WHERE C.CLIENTNO = P.CLIENTNO
        GROUP BY C.CLIENTNO, CNAME);
```

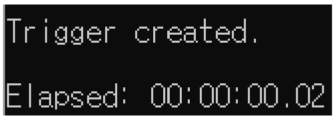


```
  CLIENTNO CNAME                SUM(AMOUNT)
---------- -------------------- -----------
     24535 Sally Moon                 20100

Elapsed: 00:00:03.37
```

2.2 TOP_DISCOUNT Trigger

If the tuple that is being inserted into the PURCAHSE table has the CLIENTNO = 24535 (Sally Moon as she is the top client from task 2.1) then we must make the AMOUNT be 85% of what it’s supposed to be.

```
CREATE OR REPLACE TRIGGER TOP_DISCOUNT
          BEFORE INSERT ON ”PURCHASE”
          FOR EACH ROW
BEGIN
          IF (:NEW.CLIENTNO = 24535)
          THEN :NEW.AMOUNT := 0.85*:NEW.AMOUNT;
          END IF;
END;
/
```

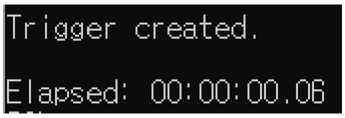


```
Trigger created.

Elapsed: 00:00:00.02
```

2.3 SUNCHINE_DEPT Trigger

Firstly, put all employee working in the department ‘SALES – Sunshine’ into a temporary table vEMPNO and check if the tuple that is being inserted into PURCHASE. If the client is being SERVEDBY an employee within vEMPNO then we must change the PAYMENTYPE = ‘Cash’. Also if the SERVICETYPE = ‘Data Recovery’ we must change the AMOUNT to 70% of what it’s supposed to be.

```
CREATE OR REPLACE TRIGGER TOP_DISCOUNT
        BEFORE INSERT ON ”PURCHASE”
        FOR EACH ROW
        DECLARE
        vEMPNO EMP.EMPNO%TYPE;
BEGIN
        SELECT EMPNO INTO vEMPNO FROM EMP WHERE DEPTNO = (
                SELECT DEPTNO FROM DEPT WHERE DNAME = ’SALES – Sunshine’);

        IF (:NEW.SERVEDBY IN (vEMPNO))
        THEN :NEW.PAYMENTTYPE := ‘Cash’;
        END IF;

        IF (:NEW.SERVICETYPE = ‘Data Recovery’)
        THEN :NEW.AMOUNT := 0.70*:NEW.AMOUNT;
        END IF;
END;
/
```

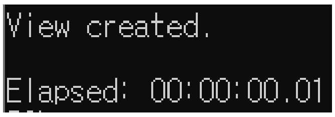


```
Trigger created.

Elapsed: 00:00:00.06
```

Task 3 – *Views*

3.1 V_DEPT_AMOUNT

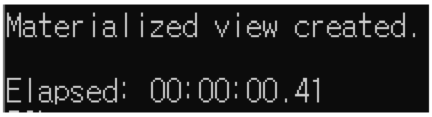
CREATE VIEW V_DEPT_AMOUNT **AS**
SELECT D.DEPTNO, D.DNAME, MAX(AMOUNT) MAXA, MIN(AMOUNT) MINA, AVG(AMOUNT) AVGA, SUM(AMOUNT) SUMA
FROM DEPT D, PURCHASE P, EMP E
WHERE P.SERVEDBY = E.EMPNO AND D.DEPTNO = E.DEPTNO
GROUP BY D.DEPTNO, D.DNAME;



```
View created.

Elapsed: 00:00:00.01
```

3.2 MV_DEPT_AMOUNT

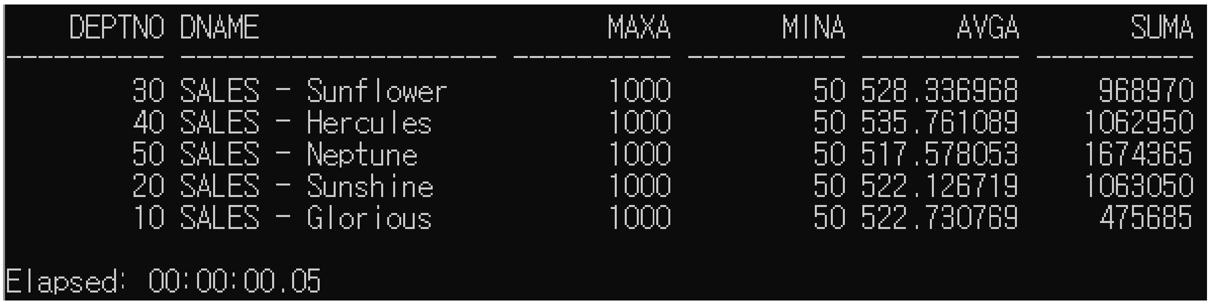
CREATE MATERIALIZED **VIEW** V_DEPT_AMOUNT AS BUILD **IMMEDIATE AS**
SELECT D.DEPTNO, D.DNAME, MAX(AMOUNT) MAXA, MIN(AMOUNT) MINA, AVG(AMOUNT) AVGA, SUM(AMOUNT) SUMA
FROM DEPT D, PURCHASE P, EMP E
WHERE P.SERVEDBY = E.EMPNO AND D.DEPTNO = E.DEPTNO
GROUP BY D.DEPTNO, D.DNAME;



```
Materialized view created.

Elapsed: 00:00:00.41
```

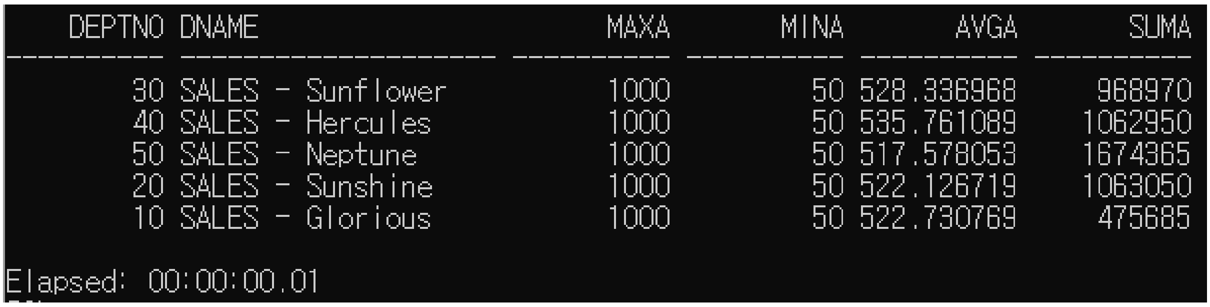
3.3 VIEW EXECUTION AND EXPLANATION

SELECT * **FROM** V_DEPT_AMOUNT;



```
    DEPTNO DNAME                                MAXA       MINA       AVGA       SUMA
---------- ------------------------------ ---------- ---------- ---------- ----------
        30 SALES - Sunflower                    1000         50 528.336968     968970
        40 SALES - Hercules                     1000         50 535.761089    1062950
        50 SALES - Neptune                      1000         50 517.578053    1674365
        20 SALES - Sunshine                     1000         50 522.126719    1063050
        10 SALES - Glorious                     1000         50 522.730769     475685

Elapsed: 00:00:00.05
```

SELECT * **FROM** MV_DEPT_AMOUNT;

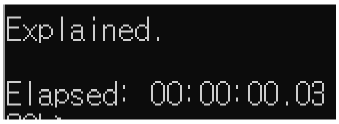


```
    DEPTNO DNAME                                MAXA       MINA       AVGA       SUMA
---------- ------------------------------ ---------- ---------- ---------- ----------
        30 SALES - Sunflower                    1000         50 528.336968     968970
        40 SALES - Hercules                     1000         50 535.761089    1062950
        50 SALES - Neptune                      1000         50 517.578053    1674365
        20 SALES - Sunshine                     1000         50 522.126719    1063050
        10 SALES - Glorious                     1000         50 522.730769     475685

Elapsed: 00:00:00.01
```

As shown above in the execution of both queries which retrieve the data from the virtual or materialized view, the query execution time of the materialized view is faster than the virtual view. Although small it is still noticeable (00.05 vs 00.01 for materialized and virtual views respectively).

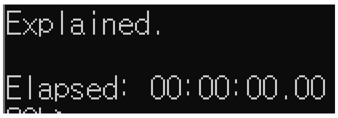
Query Execution Plan:

EXPLAIN PLAN FOR SELECT * FROM V_DEPT_AMOUNT;



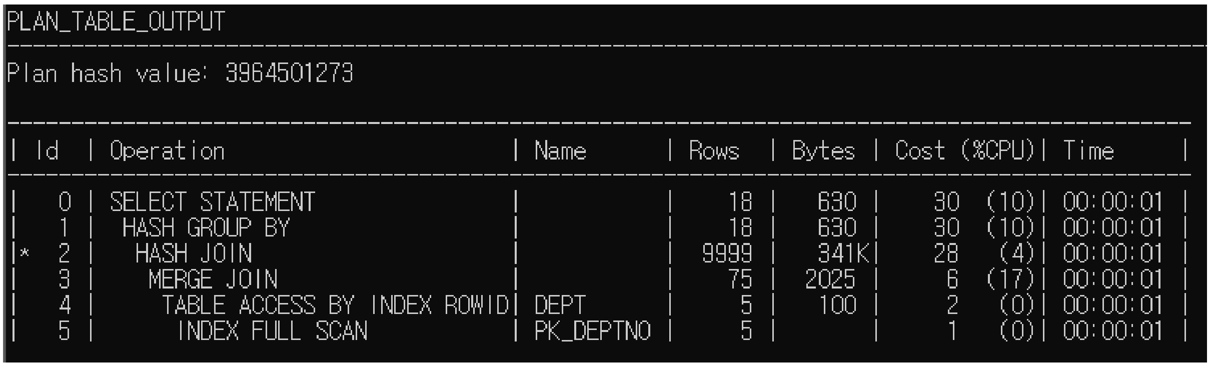
```
Explained.

Elapsed: 00:00:00.03
```

EXPLAIN PLAN FOR SELECT * FROM MV_DEPT_AMOUNT;



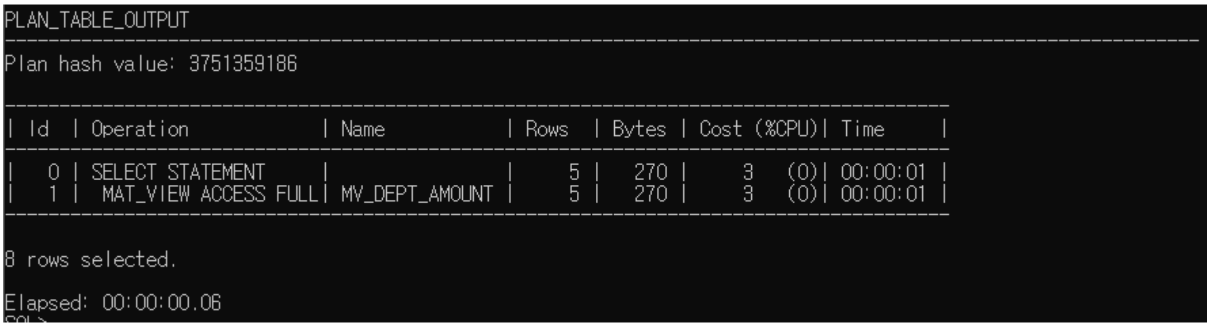
```
Explained.

Elapsed: 00:00:00.00
```

SELECT PLAN_TABLE_OUPUT FROM TABLE (DBMS_XPLAN.DISPLAY); -- VIRTUAL VIEW



```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 3964501273

-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |    18 |   630 |    30  (10)| 00:00:01 |
|   1 |  HASH GROUP BY               |           |    18 |   630 |    30  (10)| 00:00:01 |
|*  2 |   HASH JOIN                  |           |  9999 |   341K|    28   (4)| 00:00:01 |
|   3 |    MERGE JOIN                |           |    75 |  2025 |     6  (17)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| DEPT    |     5 |   100 |     2   (0)| 00:00:01 |
|   5 |      INDEX FULL SCAN         | PK_DEPTNO |     5 |       |     1   (0)| 00:00:01 |
```

SELECT PLAN_TABLE_OUPUT FROM TABLE (DBMS_XPLAN.DISPLAY); -- MATERIALIZED VIEW



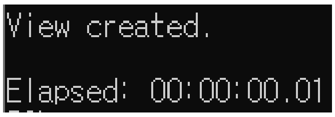
```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 3751359186

-----------------------------------------------------------------------------------
| Id  | Operation            | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                |     5 |   270 |     3   (0)| 00:00:01 |
|   1 |  MAT_VIEW ACCESS FULL| MV_DEPT_AMOUNT |     5 |   270 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------

8 rows selected.

Elapsed: 00:00:00.06
```

The reason can be seen in the execution plans above. As virtual views are not physically stored within the database the views/queries must be recomputed each time the query is ran. Thus oracle has to perform query optimization by choosing to use efficient algorithms such as 'HASH JOIN', 'MERGE JOIN' & Taking use of the index on the primary key (DEPTNO). However, because materialized views are stored physically oracle has gone ahead and accessed the view/table directly, which can be seen by the 'MAT_VIEW ACCESS FULL', thus no query modification is performed which will overall improve the speed of query processing. Ultimately, the 'Time' column in the execution plan immediately shows the query processing time differences.

3.4 DEPT_EMP_AMOUNT

3.4.1 V_DEPT_EMP_AMOUNT

```
CREATE VIEW V_DEPT_EMP_AMOUNT AS
SELECT D.DEPTNO, EMPNO, COUNT(*) CA, AVG(AMOUNT) AVGA, MAX(AMOUNT) MAXA, SUM(AMOUNT)
SUMA
FROM DEPT D, EMP E, PURCHASE P
WHERE D.DEPTNO = E.DEPTNO AND P.SERVEDBY = E.EMPNO
GROUP BY D.DEPTNO, EMPNO
ORDER BY D.DEPTNO ASC, SUMA DESC;
```



```
View created.

Elapsed: 00:00:00.01
```

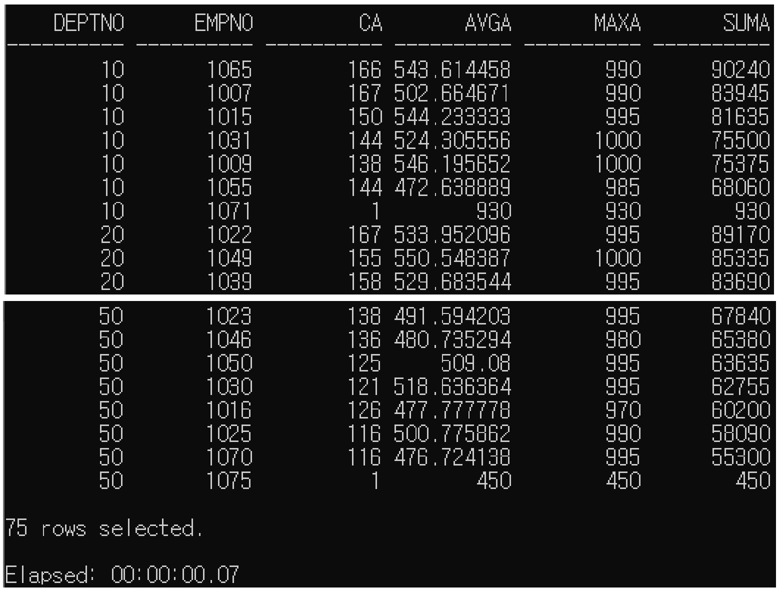
3.4.2 MV_DEPT_EMP_AMOUNT

```
CREATE MATERIALIZED VIEW MV_DEPT_EMP_AMOUNT BUILD IMMEDIATE AS
SELECT D.DEPTNO, EMPNO, COUNT(*) CA, AVG(AMOUNT) AVGA, MAX(AMOUNT) MAXA, SUM(AMOUNT)
SUMA
FROM DEPT D, EMP E, PURCHASE P
WHERE D.DEPTNO = E.DEPTNO AND P.SERVEDBY = E.EMPNO
GROUP BY D.DEPTNO, EMPNO
ORDER BY D.DEPTNO ASC, SUMA DESC;
```

```
Materialized view created.

Elapsed: 00:00:00.08
```

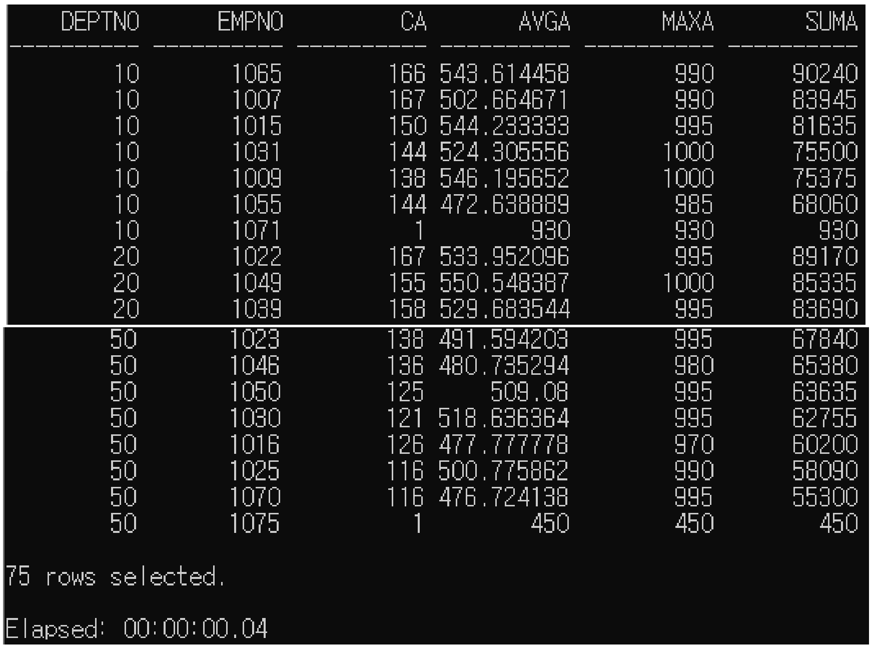
3.5 VIEW EXECUTION AND EXPLANATION

SELECT * **FROM** V_DEPT_EMP_AMOUNT;



```
    DEPTNO      EMPNO         CA       AVGA       MAXA       SUMA
---------- ---------- ---------- ---------- ---------- ----------
        10       1065        166 543.614458        990      90240
        10       1007        167 502.664671        990      83945
        10       1015        150 544.233333        995      81635
        10       1031        144 524.305556       1000      75500
        10       1009        138 546.195652       1000      75375
        10       1055        144 472.638889        985      68060
        10       1071          1        930        930        930
        20       1022        167 533.952096        995      89170
        20       1049        155 550.548387       1000      85335
        20       1039        158 529.683544        995      83690
        50       1023        138 491.594203        995      67840
        50       1046        136 480.735294        980      65380
        50       1050        125     509.08        995      63635
        50       1030        121 518.636364        995      62755
        50       1016        126 477.777778        970      60200
        50       1025        116 500.775862        990      58090
        50       1070        116 476.724138        995      55300
        50       1075          1        450        450        450

75 rows selected.

Elapsed: 00:00:00.07
```

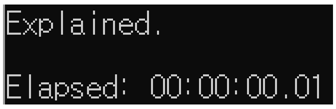
SELECT * **FROM** MV_DEPT_EMP_AMOUNT;



```
    DEPTNO      EMPNO         CA       AVGA       MAXA       SUMA
---------- ---------- ---------- ---------- ---------- ----------
        10       1065        166 543.614458        990      90240
        10       1007        167 502.664671        990      83945
        10       1015        150 544.233333        995      81635
        10       1031        144 524.305556       1000      75500
        10       1009        138 546.195652       1000      75375
        10       1055        144 472.638889        985      68060
        10       1071          1        930        930        930
        20       1022        167 533.952096        995      89170
        20       1049        155 550.548387       1000      85335
        20       1039        158 529.683544        995      83690
        50       1023        138 491.594203        995      67840
        50       1046        136 480.735294        980      65380
        50       1050        125     509.08        995      63635
        50       1030        121 518.636364        995      62755
        50       1016        126 477.777778        970      60200
        50       1025        116 500.775862        990      58090
        50       1070        116 476.724138        995      55300
        50       1075          1        450        450        450

75 rows selected.

Elapsed: 00:00:00.04
```

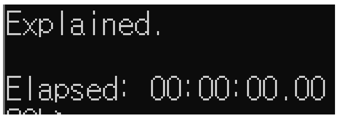
Query Execution Plan:

EXPLAIN PLAN FOR SELECT * FROM V_DEPT_EMP_AMOUNT;



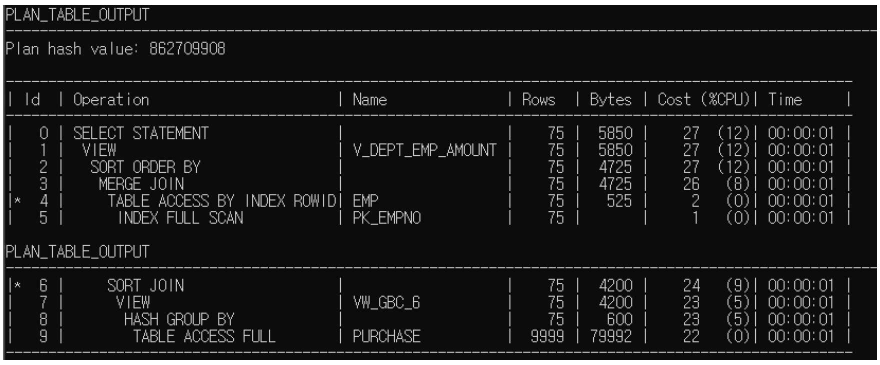
```
Explained.

Elapsed: 00:00:00.01
```

EXPLAIN PLAN FOR SELECT * FROM MV_DEPT_EMP_AMOUNT;



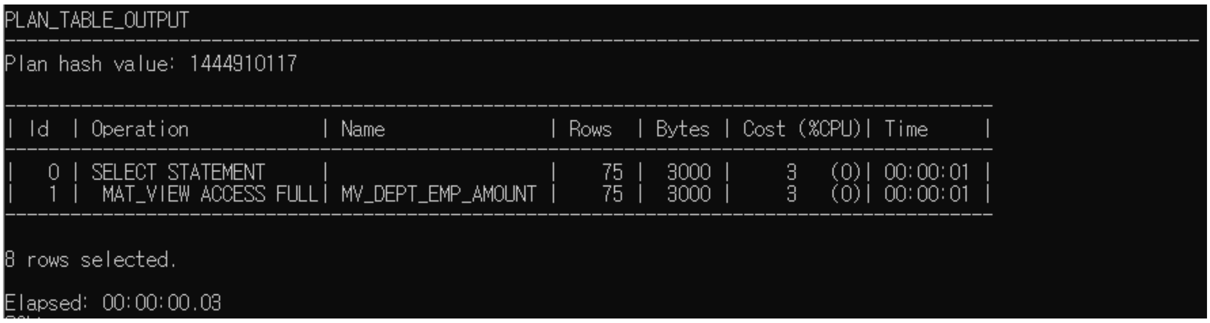
```
Explained.

Elapsed: 00:00:00.00
```

SELECT PLAN_TABLE_OUPUT FROM TABLE (DBMS_XPLAN.DISPLAY); -- VIRTUAL VIEW



```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
Plan hash value: 862709908

------------------------------------------------------------------------------------------
| Id  | Operation                      | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                   |    75 |  5850 |    27  (12)| 00:00:01 |
|   1 |  VIEW                          | V_DEPT_EMP_AMOUNT |    75 |  5850 |    27  (12)| 00:00:01 |
|   2 |   SORT ORDER BY                |                   |    75 |  4725 |    27  (12)| 00:00:01 |
|   3 |    MERGE JOIN                  |                   |    75 |  4725 |    26   (8)| 00:00:01 |
|*  4 |     TABLE ACCESS BY INDEX ROWID| EMP               |    75 |   525 |     2   (0)| 00:00:01 |
|   5 |      INDEX FULL SCAN           | PK_EMPNO          |    75 |       |     1   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
|*  6 |     SORT JOIN                  |                   |    75 |  4200 |    24   (9)| 00:00:01 |
|   7 |      VIEW                      | VW_GBC_6          |    75 |  4200 |    23   (5)| 00:00:01 |
|   8 |       HASH GROUP BY            |                   |    75 |   600 |    23   (5)| 00:00:01 |
|   9 |        TABLE ACCESS FULL       | PURCHASE          |  9999 | 79992 |    22   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
```

SELECT PLAN_TABLE_OUPUT FROM TABLE (DBMS_XPLAN.DISPLAY); -- MATERIALIZED VIEW



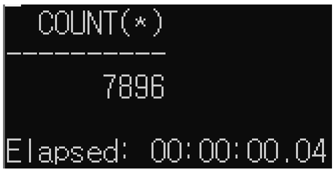
```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
Plan hash value: 1444910117

-------------------------------------------------------------------------------------------
| Id  | Operation            | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                    |    75 |  3000 |     3   (0)| 00:00:01 |
|   1 |  MAT_VIEW ACCESS FULL| MV_DEPT_EMP_AMOUNT |    75 |  3000 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

8 rows selected.

Elapsed: 00:00:00.03
```

Similarly to task 3.1~3.3 the query execution time of the materialized view was faster than the virtual view (00.04 vs 00.07 respectively). This is again expected as materialized views are stored physically while the virtual views are not. Thus as shown in the execution plan for virtual view above, oracle uses query optimization for the query processing of the virtual view by using efficient algorithms such as 'HASH JOIN', 'MERGE JOIN' & even taking advantage of the index on the primary key (EMPNO). However when executing the query for the materialized view, oracle just needs to access the table stored physically thus making the execution of materialized views much faster.

Task 4 – *Indexes*

4.1 Receipt book

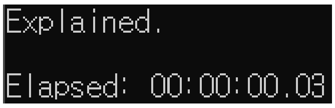
```
SELECT COUNT(*) FROM PURCHASE
WHERE SUBSTR(RECEIPTNO, 0, 3) IN (
          SELECT SUBSTR(RECEIPTNO, 0, 3) FROM PURCHASE
          GROUP BY SUBSTR(RECEIPTNO, 0 ,3)
          HAVING COUNT(*) >= 10);
```



```
  COUNT(*)
----------
      7896

Elapsed: 00:00:00.04
```

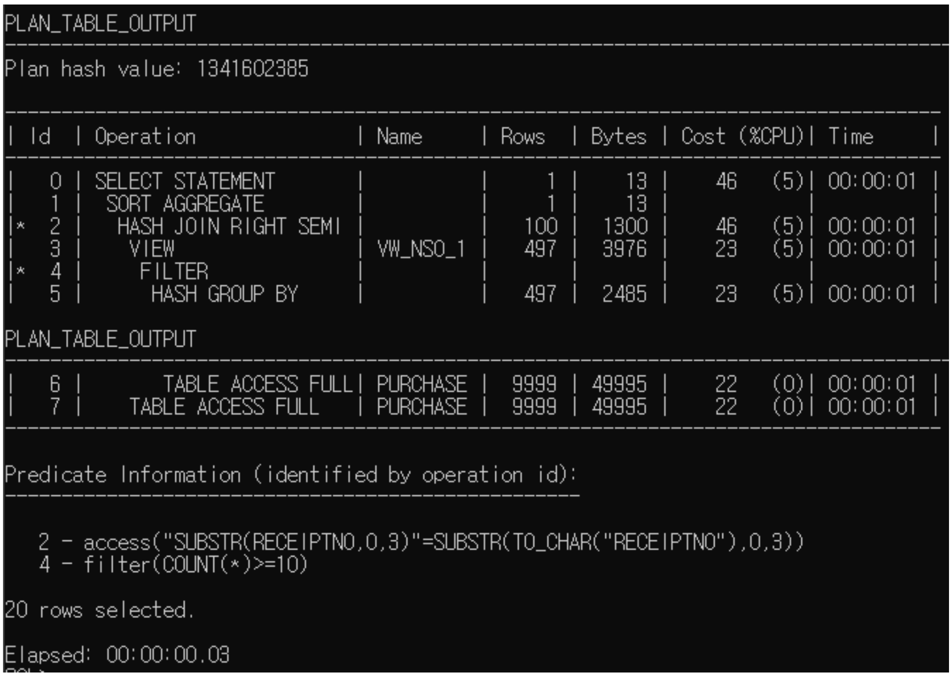
4.2 Receipt book query execution plan/time and creation of index

```
EXPLAIN PLAN FOR SELECT COUNT(*) FROM PURCHASE
WHERE SUBSTR(RECEIPTNO, 0, 3) IN (
          SELECT SUBSTR(RECEIPTNO, 0, 3) FROM PURCHASE
          GROUP BY SUBSTR(RECEIPTNO, 0 ,3)
          HAVING COUNT(*) >= 10);
```



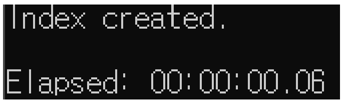
```
Explained.

Elapsed: 00:00:00.03
```

```
SELECT PLAN_TABLE_OUPUT FROM TABLE (DBMS_XPLAN.DISPLAY); --Before Index
```



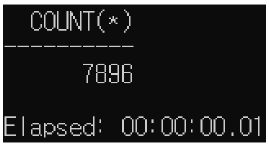
```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------
Plan hash value: 1341602385

--------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |    13 |    46   (5)| 00:00:01 |
|   1 |  SORT AGGREGATE        |          |     1 |    13 |            |          |
|*  2 |   HASH JOIN RIGHT SEMI |          |   100 |  1300 |    46   (5)| 00:00:01 |
|   3 |    VIEW                | VW_NSO_1 |   497 |  3976 |    23   (5)| 00:00:01 |
|*  4 |     FILTER             |          |       |       |            |          |
|   5 |      HASH GROUP BY     |          |   497 |  2485 |    23   (5)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------
|   6 |       TABLE ACCESS FULL| PURCHASE |  9999 | 49995 |    22   (0)| 00:00:01 |
|   7 |    TABLE ACCESS FULL   | PURCHASE |  9999 | 49995 |    22   (0)| 00:00:01 |
--------------------------------------------------------------------------------------

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

   2 - access("SUBSTR(RECEIPTNO,0,3)"=SUBSTR(TO_CHAR("RECEIPTNO"),0,3))
   4 - filter(COUNT(*)>=10)

20 rows selected.

Elapsed: 00:00:00.03
```

CREATE INDEX BOOK_INDEX **ON** PURCHASE **SUBSTR(**RECEIPTNO, 0, 3);



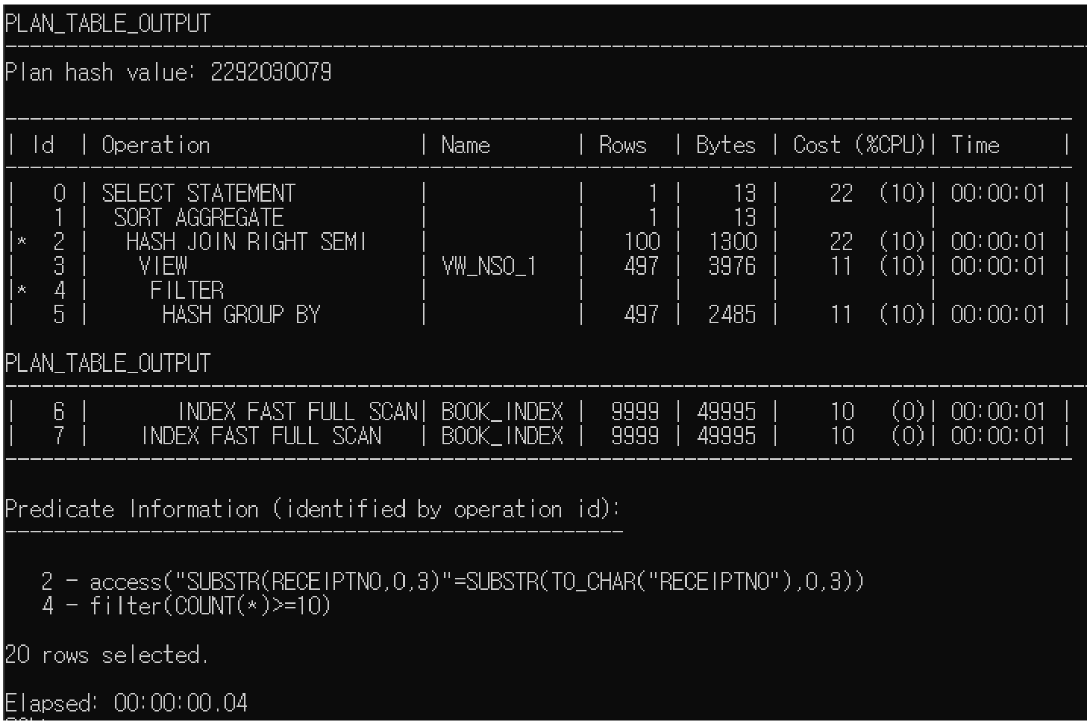
```
Index created.

Elapsed: 00:00:00.06
```

Result of Query After index creation.



```
  COUNT(*)
----------
      7896

Elapsed: 00:00:00.01
```

SELECT PLAN_TABLE_OUPUT FROM TABLE (DBMS_XPLAN.DISPLAY); --After Index



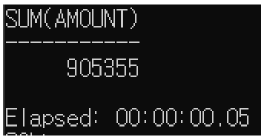
```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 2292030079

-------------------------------------------------------------------------------
| Id  | Operation               | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |            |     1 |    13 |    22  (10)| 00:00:01 |
|   1 |  SORT AGGREGATE         |            |     1 |    13 |            |          |
|*  2 |   HASH JOIN RIGHT SEMI  |            |   100 |  1300 |    22  (10)| 00:00:01 |
|   3 |    VIEW                 | VW_NSO_1   |   497 |  3976 |    11  (10)| 00:00:01 |
|*  4 |     FILTER              |            |       |       |            |          |
|   5 |      HASH GROUP BY      |            |   497 |  2485 |    11  (10)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|   6 |       INDEX FAST FULL SCAN| BOOK_INDEX |  9999 | 49995 |    10   (0)| 00:00:01 |
|   7 |    INDEX FAST FULL SCAN   | BOOK_INDEX |  9999 | 49995 |    10   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   2 - access("SUBSTR(RECEIPTNO,0,3)"=SUBSTR(TO_CHAR("RECEIPTNO"),0,3))
   4 - filter(COUNT(*)>=10)

20 rows selected.

Elapsed: 00:00:00.04
```

For the query in task 4.1 a function-based index on SUBSTR(RECEIPTNO, 0, 3) was chosen as it is part of the WHERE clause which can yield lots of performance gain.

As seen above, the query execution time improved after making the function-based index (00.04 & 00.01 before and after the index creation respectively). This is evident within the query execution plan where before BOOK_INDEX is created the query optimizer uses the 'TABLE ACCESS FULL' while after the creation of the index the query optimizer has gone ahead and chosen to do an 'INDEX FAST FULL SCAN' on the BOOK_INDEX. This means that oracle will excess the index created on SUBSTR(RECEIPTNO, 0, 3) rather than having to process the SUBSTR(). Thus improving the overall performance of the query.

4.3 Department 50

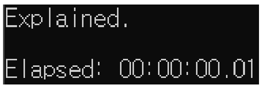
SELECT SUM(AMOUNT) **FROM** PURCHASE P
WHERE SERVEDBY IN (**SELECT** EMPNO **FROM** EMP
WHERE DEPTNO = 50)
AND INSTR(SERVICETYPE, 'Software') = 0;



```
SUM(AMOUNT)
-----------
     905355

Elapsed: 00:00:00.05
```

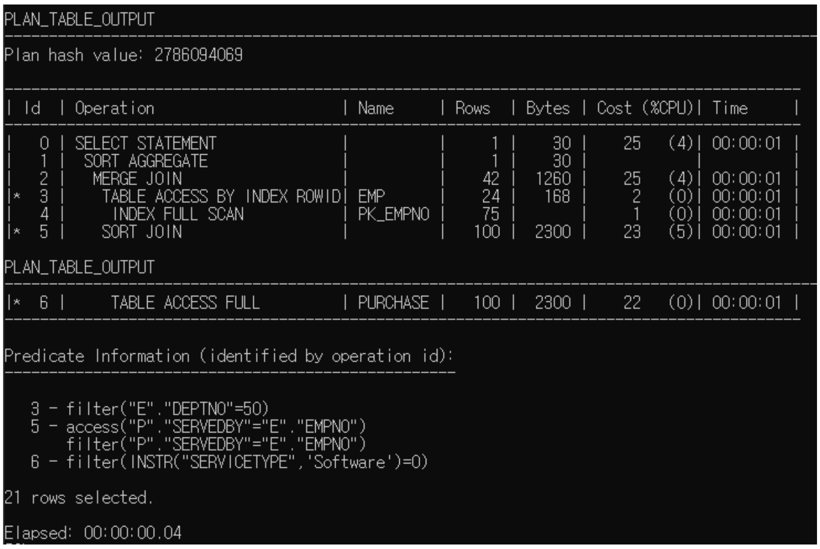
4.4 Department 50 query execution plan/time and creation of index

EXPLAIN PLAN FOR SELECT SUM(AMOUNT) **FROM** PURCHASE P
WHERE SERVEDBY IN (**SELECT** EMPNO **FROM** EMP
WHERE DEPTNO = 50)
AND INSTR(SERVICETYPE, 'Software') = 0;



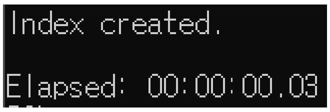
```
Explained.

Elapsed: 00:00:00.01
```

SELECT PLAN_TABLE_OUPUT FROM TABLE (DBMS_XPLAN.DISPLAY); --Before Index



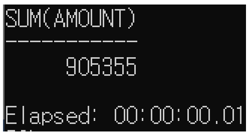
```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
Plan hash value: 2786094069

------------------------------------------------------------------------------------------
| Id  | Operation                     | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |     1 |    30 |    25   (4)| 00:00:01 |
|   1 |  SORT AGGREGATE               |          |     1 |    30 |            |          |
|   2 |   MERGE JOIN                  |          |    42 |  1260 |    25   (4)| 00:00:01 |
|*  3 |    TABLE ACCESS BY INDEX ROWID| EMP      |    24 |   168 |     2   (0)| 00:00:01 |
|   4 |     INDEX FULL SCAN           | PK_EMPNO |    75 |       |     1   (0)| 00:00:01 |
|*  5 |    SORT JOIN                  |          |   100 |  2300 |    23   (5)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
|*  6 |     TABLE ACCESS FULL         | PURCHASE |   100 |  2300 |    22   (0)| 00:00:01 |
------------------------------------------------------------------------------------------

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

   3 - filter("E"."DEPTNO"=50)
   5 - access("P"."SERVEDBY"="E"."EMPNO")
       filter("P"."SERVEDBY"="E"."EMPNO")
   6 - filter(INSTR("SERVICETYPE",'Software')=0)

21 rows selected.

Elapsed: 00:00:00.04
```

CREATE INDEX SERVICE_INDEX **ON** PURCHASE **INSTR(**SERVICETYPE, 'Software');



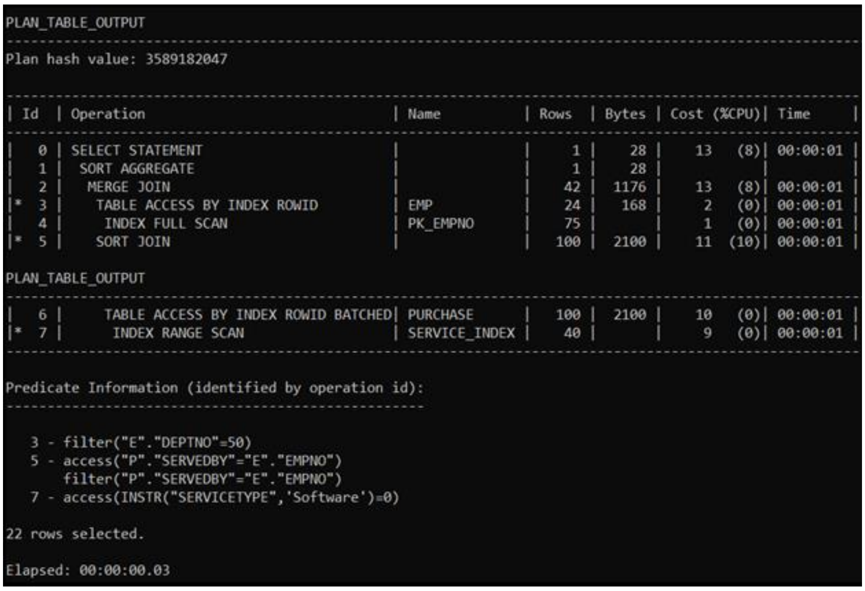
```
Index created.

Elapsed: 00:00:00.03
```

Result of Query After index creation.



```
SUM(AMOUNT)
-----------
     905355

Elapsed: 00:00:00.01
```

SELECT PLAN_TABLE_OUPUT FROM TABLE (DBMS_XPLAN.DISPLAY); --After Index



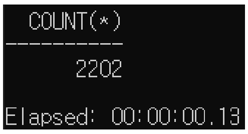
```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
Plan hash value: 3589182047

--------------------------------------------------------------------------------------------
| Id  | Operation                            | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |               |     1 |    28 |    13   (8)| 00:00:01 |
|   1 |  SORT AGGREGATE                      |               |     1 |    28 |            |          |
|   2 |   MERGE JOIN                         |               |    42 |  1176 |    13   (8)| 00:00:01 |
|*  3 |    TABLE ACCESS BY INDEX ROWID       | EMP           |    24 |   168 |     2   (0)| 00:00:01 |
|   4 |     INDEX FULL SCAN                  | PK_EMPNO      |    75 |       |     1   (0)| 00:00:01 |
|*  5 |    SORT JOIN                         |               |   100 |  2100 |    11  (10)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
|   6 |     TABLE ACCESS BY INDEX ROWID BATCHED| PURCHASE      |   100 |  2100 |    10   (0)| 00:00:01 |
|*  7 |      INDEX RANGE SCAN                | SERVICE_INDEX |    40 |       |     9   (0)| 00:00:01 |
--------------------------------------------------------------------------------------------

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

   3 - filter("E"."DEPTNO"=50)
   5 - access("P"."SERVEDBY"="E"."EMPNO")
       filter("P"."SERVEDBY"="E"."EMPNO")
   7 - access(INSTR("SERVICETYPE",'Software')=0)

22 rows selected.

Elapsed: 00:00:00.03
```

Similarly to Task 4.2 a function-based index was chosen on the INSTR(SERVICETYPE, 'Software') as it is part of the WHERE clause which can give us lots of performance gain.

Although there isn't much of a difference in the elapsed time of the query execution after the index creation (00.05 vs 00.01 before & after respectively) it is still an improvement. This is further highlighted in the query execution plan. Looking at the query execution of both before and after the index creation, the overall cost before is 98 and after is 59. This is due to operation id 7 where the oracle query optimizer uses the SERVICE_INDEX to do an 'INDEX RANGE SCAN' rather than processing the INSTR(…). Hence, improving the overall performance of the query.

4.5 ServiceType, PaymentType & GST

```
SELECT COUNT(*)
FROM PURCHASE
WHERE SERVICETYPE IN (
        SELECT SERVICETYPE FROM PURCHASE
        GROUP BY SERVICETYPE, PAYMENTTYPE, GST
        HAVING COUNT(*) >= 1000)
AND PAYMENTTYPE IN (
        SELECT PAYMENTTYPE FROM PURCHASE
        GROUP BY SERVICETYPE, PAYMENTTYPE, GST
        HAVING COUNT(*) >= 1000)
AND GST IN (
        SELECT GST FROM PURCHASE
        GROUP BY SERVICETYPE, PAYMENTTYPE, GST
        HAVING COUNT(*) >= 1000);
```



```
  COUNT(*)
----------
      2202

Elapsed: 00:00:00.13
```

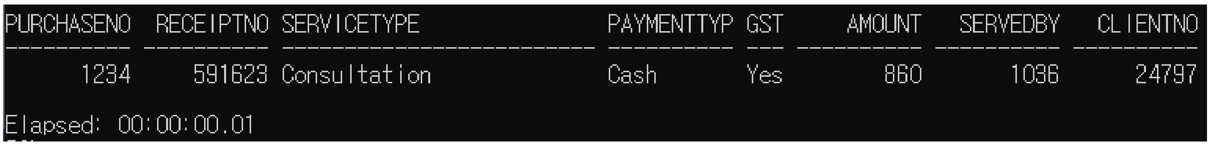
4.6 Indexing on ServiceType, PaymentType & GST

Firstly, unlike Task 4.1~4.4 there are no function-based index that can be created within the query written in Task 4.5 thus we must choose a different type of index. Overall the most suitable type of index to be created on all three of these columns are a bitmap index. This is because there's a low number of distinct values that each column/attribute can take. SERVICETYPE, PAYMENTTYPE & GST have 5, 3 & 2 distinct values respectively as defined in the constraints earlier in Task 1.2. Additionally, DB optimizers such as the one used in Oracle, can combine several bitmap indexes very easily which will allow for efficient execution of complex filters and queries *(Another thing to mention is that although there were constraints defined on these columns to prevent null values, the bitmap index also includes null values unlike the b-tree index, thus allowing the DB optimizer to use the index on 'null' queries).* Hence, due to these reasons the bitmap indexes will be more efficient compared to other indexes such as the b-tree index.

Task 5 – *Execution Plan*

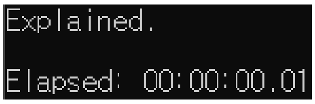
5.1 PURCHASENO = 1234

SELECT * FROM PURCHASE **WHERE** PURCHASENO = 1234; --Before



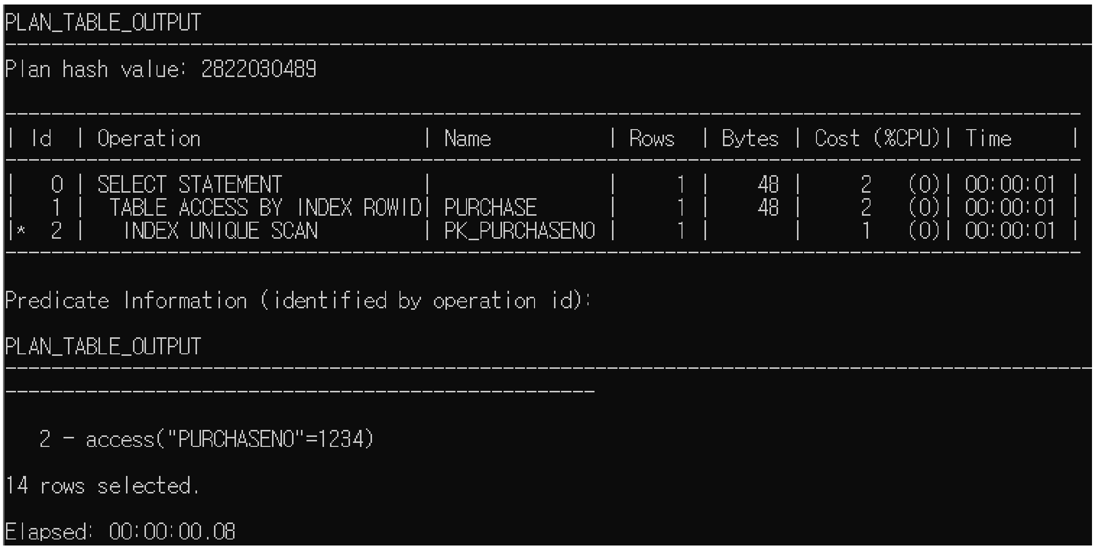
```
PURCHASENO  RECEIPTNO SERVICETYPE                PAYMENTTYP GST     AMOUNT   SERVEDBY   CLIENTNO
---------- ---------- ------------------------- ---------- --- ---------- ---------- ----------
      1234     591623 Consultation              Cash       Yes        860       1036      24797

Elapsed: 00:00:00.01
```

EXPLAIN PLAN FOR **SELECT * FROM** PURCHASE **WHERE** PURCHASENO = 1234;



```
Explained.

Elapsed: 00:00:00.01
```

SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY); -- Before



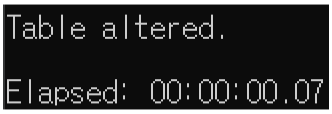
```
PLAN_TABLE_OUTPUT
----------------------------------------------------------------------------------------------------
Plan hash value: 2822030489

---------------------------------------------------------------------------------------------
| Id  | Operation                   | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |               |     1 |    48 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| PURCHASE      |     1 |    48 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN         | PK_PURCHASENO |     1 |       |     1   (0)| 00:00:01 |
---------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
----------------------------------------------------------------------------------------------------
---------------------------------------------------

   2 - access("PURCHASENO"=1234)

14 rows selected.

Elapsed: 00:00:00.08
```

Execution Plan Explanation:

1. As 'INDEX UNIQUE SCAN' is the most indented to the right, this process is run first. This operation uses the index PK_PURCHASENO to search for the value within the WHERE clause which in this case is '1234' and returns just 1 row-id from the index.
2. However, all the information needed isn't in the index. Thus, the oracle kernel goes through the actual table with the pointer it got from searching through the index to fetch further information requested in the query.
3. Finally after Oracle has found the correct row, the 'SELECT STATEMENT' operation returns the row.

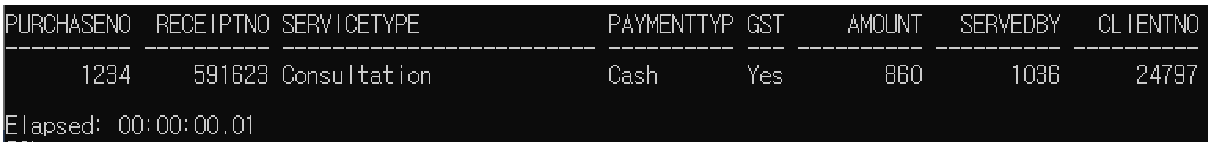
5.2 DROP CONSTRAINT AND RE-EXECUTE

ALTER TABLE PURCHASE **DROP CONSTRAINT** PK_PURCHASENO;



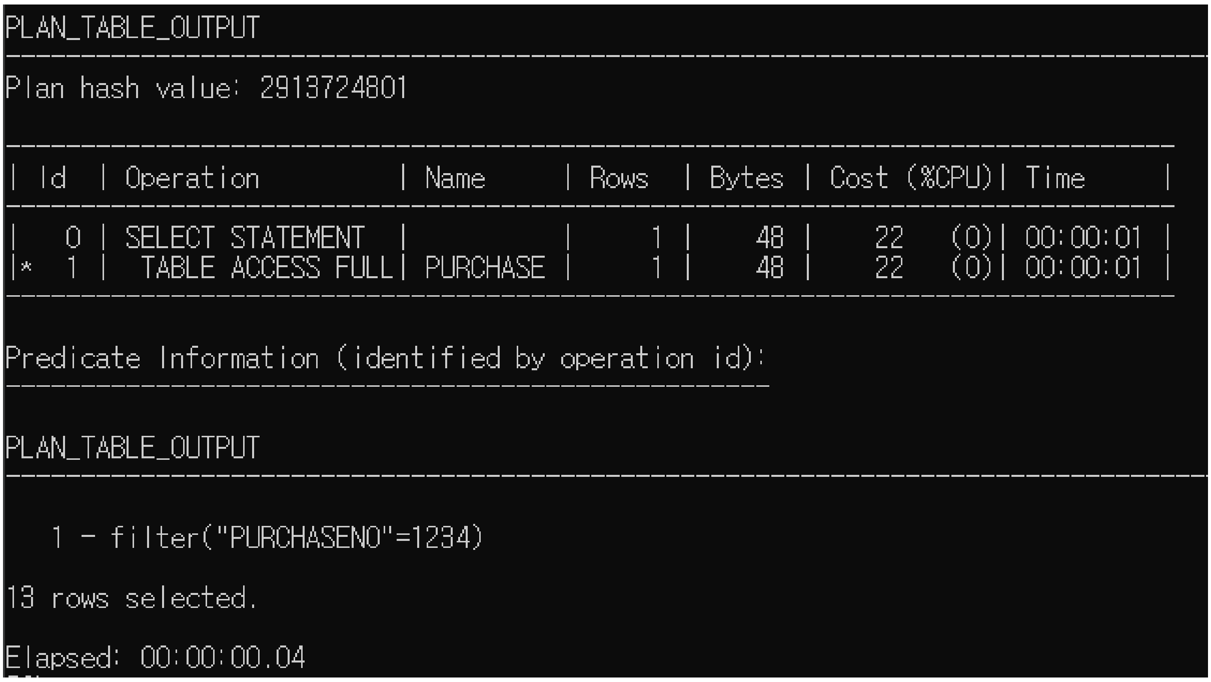
```
Table altered.

Elapsed: 00:00:00.07
```

SELECT * FROM PURCHASE **WHERE** PURCHASENO = 1234; -- After



```
PURCHASENO  RECEIPTNO SERVICETYPE               PAYMENTTYP GST     AMOUNT   SERVEDBY   CLIENTNO
---------- ---------- ------------------------- ---------- --- ---------- ---------- ----------
      1234     591623 Consultation              Cash       Yes        860       1036      24797

Elapsed: 00:00:00.01
```

SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY); -- After



```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 2913724801

------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |     1 |    48 |    22   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| PURCHASE |     1 |    48 |    22   (0)| 00:00:01 |
------------------------------------------------------------------------------

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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

   1 - filter("PURCHASENO"=1234)

13 rows selected.

Elapsed: 00:00:00.04
```

1. This time the most indented operation is 'TABLE ACCESS FULL'. This means that every row within the table PURCHASE has been accessed to do an equality search for PURCAHSENO = 1234.
2. Then the 'SELECT STATEMENT' operation returns the correct row.

Although there aren't any differences within the query elapsed time, there are differences within the execution plan that Oracle has chosen to evaluate the query. In general the main difference between the execution plan in 5.1 and 5.2 is that 5.2 uses the index thus accessing a B-tree index to search for the correct value unlike the execution plan in 5.1 which accesses the whole table and searches for the correct value by brute force. Another thing to mention is the Cost (%CPU). It shows that the overall cost for the query in 5.1 (44 overall) is much more expensive than the overall cost for the query in 5.2 (5 overall).