INFS2200 ASSIGNMENT

Student Number: 4536071

Task 1 – Constraints

1.1 Find Existing Constraints

```
SELECT CONSTRAINT_NAME, TABLE_NAME, INDEX_NAME
FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'EMP';
```

CONSTRAINT_NAME	TABLE_NAME	INDEX_NAME
PK_EMPNO	EMP	PK_EMPNO

```
SELECT CONSTRAINT_NAME, TABLE_NAME, INDEX_NAME
FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'DEPT';
```

CONSTRAINT_NAME	TABLE_NAME	INDEX_NAME
PK_DEPTNO	DEPT	PK_DEPTNO

```
SELECT CONSTRAINT_NAME, TABLE_NAME, INDEX_NAME
FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'PURCHASE';
```

CONSTRAINT_NAME	TABLE_NAME	INDEX_NAME
PK_PURCHASENO	PURCHASE	PK_PURCHASENO

```
SELECT CONSTRAINT_NAME, TABLE_NAME, INDEX_NAME
FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'CLIENT';
```

CONSTRAINT_NAME	TABLE_NAME	INDEX_NAME
PK_CLIENTNO	CLIENT	PK_CLIENTNO

1.2 Create Missing Constraints

```
ALTER TABLE DEPT ADD CONSTRAINT UN_DNAME UNIQUE (DName);

ALTER TABLE PURCHASE MODIFY AMOUNT INT CONSTRAINT CK_AMOUNT NOT NULL;
ALTER TABLE EMP MODIFY ENAME VARCHAR(30) CONSTRAINT CK_ENAME NOT NULL;
ALTER TABLE DEPT MODIFY DNAME VARCHAR(30) CONSTRAINT CK_DNAME NOT NULL;
ALTER TABLE CLIENT MODIFY CNAME VARCHAR(30) CONSTRAINT CK_CNAME NOT NULL;
ALTER TABLE PURCHASE MODIFY RECEIPTNO INT CONSTRAINT CK_RECEIPTNO NOT NULL;

ALTER TABLE PURCHASE ADD CONSTRAINT CK_SERVICETYPE
CHECK (ServiceType IN ('Training', 'Data Recovery', 'Consultation',
'Software Installation', 'Software Repair'));

ALTER TABLE PURCHASE ADD CONSTRAINT CK_PAYMENTTYPE
CHECK (PaymentType IN ('Debit', 'Cash', 'Credit'));

ALTER TABLE PURCHASE ADD CONSTRAINT CK_GST
CHECK (GST IN ('Yes', 'No'));

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

ALTER TABLE PURCHASE ADD CONSTRAINT FK_EMPNO FOREIGN KEY (ServedBy)
REFERENCES EMP (EmpNo);

ALTER TABLE PURCHASE ADD CONSTRAINT FK_CLIENTNO FOREIGN KEY (ClientNo)
REFERENCES CLIENT (ClientNo);
```

CONSTRAINT_NAME	TABLE_NAME	INDEX_NAME
FK_DEPTNO	EMP	
CK_ENAME	EMP	
PK_EMPNO	EMP	PK_EMPNO

CONSTRAINT_NAME	TABLE_NAME	INDEX_NAME
CK_DNAME	DEPT	
UN_DNAME	DEPT	UN_DNAME
PK_DEPTNO	DEPT	PK_DEPTNO

CONSTRAINT_NAME	TABLE_NAME	INDEX_NAME
FK_EMPNO	PURCHASE	
FK_CLIENTNO	PURCHASE	
CK_AMOUNT	PURCHASE	
CK_RECEIPTNO	PURCHASE	
CK_SERVICETYPE	PURCHASE	
CK_PAYMENTTYPE	PURCHASE	
CK_GST	PURCHASE	
PK_PURCHASENO	PURCHASE	PK_PURCHASENO

CONSTRAINT_NAME	TABLE_NAME	INDEX_NAME
CK_CNAME	CLIENT	
PK_CLIENTNO	CLIENT	PK_CLIENTNO

Task 2 – Triggers

2.1 Top Client

```
CREATE TABLE TOPCOMPANYCLIENT AS
SELECT C.ClientNo AS ClientNumber, C.CName AS ClientName,
SUM(P.Amount) AS TotalAmount
FROM CLIENT C, PURCHASE P
WHERE C.ClientNo = P.ClientNo
GROUP BY C.ClientNo, C.CName
ORDER BY SUM(P.Amount) DESC
FETCH FIRST 1 ROWS ONLY;

SELECT * FROM TOPCOMPANYCLIENT;
```

```
CLIENTNUMBER CLIENTNAME                    TOTALAMOUNT
------------ ------------------------------ -----------
       24535 Sally Moon                           20100
```

2.2 Top Discount Trigger

```
CREATE OR REPLACE TRIGGER TOP_DISCOUNT
BEFORE INSERT ON PURCHASE
FOR EACH ROW
DECLARE
  ClientID INT;
BEGIN
SELECT T.ClientNumber INTO ClientID
FROM TOPCOMPANYCLIENT T;
IF :NEW.CLIENTNO = ClientID THEN
:NEW.AMOUNT :=  (0.85 * :NEW.AMOUNT);
END IF;
 END;
/
```

2.2 Trigger Check

```
SELECT *
FROM PURCHASE P
WHERE P.CLIENTNO = 24535
AND P.PAYMENTTYPE = 'Cash'
AND P.GST = 'No'
AND P.ServedBy = 1045;
```

```
PURCHASENO  RECEIPTNO SERVICETYPE              PAYMENTTYP GST     AMOUNT   SERVEDBY   CLIENTNO
---------- ---------- ------------------------ ---------- --- ---------- ---------- ----------
       346     546443 Software Repair          Cash       No         915       1045      24535
```

```
INSERT INTO PURCHASE VALUES (10000, 101101, 'Consultation', 'Cash', 'No', 1000, 1045, 24535);
```

```
PURCHASENO  RECEIPTNO SERVICETYPE              PAYMENTTYP GST     AMOUNT   SERVEDBY   CLIENTNO
---------- ---------- ------------------------ ---------- --- ---------- ---------- ----------
       346     546443 Software Repair          Cash       No         915       1045      24535
     10000     101101 Consultation             Cash       No         850       1045      24535
```

Above, a new purchase made by our top client was inserted into the purchase table with the amount 1000. The trigger works and we can see this by the updated screenshot of the table above with the amount at 850.

2.3 Sales - Sunshine Department

```
CREATE OR REPLACE TRIGGER SUNSHINE_DEPT
BEFORE INSERT ON PURCHASE
FOR EACH ROW
DECLARE
  DepartName VARCHAR(30);
BEGIN
  SELECT DISTINCT D.DName INTO DepartName
  FROM EMP E, DEPT D, PURCHASE P
  WHERE E.DEPTNO = D.DEPTNO
  AND E.EMPNO = P.SERVEDBY
  AND :NEW.SERVEDBY = P.SERVEDBY;

  IF DepartName = 'SALES – Sunshine'
  AND :NEW.PAYMENTTYPE != 'Cash' THEN
        :NEW.PAYMENTTYPE := 'Cash';
  END IF;

  IF DepartName = 'SALES – Sunshine'
  AND:NEW.SERVICETYPE = 'Data Recovery' THEN
           :NEW.AMOUNT :=  (:NEW.AMOUNT * 0.70);
  END IF;
END;
/
```

2.3 Trigger Check

Check 1: Update Columns to Cash:

```
INSERT INTO PURCHASE VALUES (90000, 522555, 'Consultation', 'Credit', 'Yes', 1000, 7777,
24542);

SELECT *
FROM PURCHASE P
WHERE P.PurchaseNo = 90000;
```

PURCHASENO	RECEIPTNO	SERVICETYPE	PAYMENTTYP	GST	AMOUNT	SERVEDBY	CLIENTNO
90000	522555	Consultation	Cash	Yes	1000	7777	24542

Check 2: Update to Cash and Amount for Data Recovery:

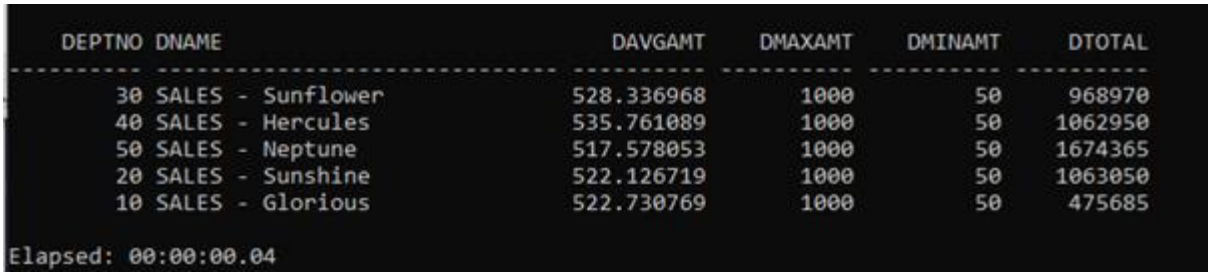
```
INSERT INTO PURCHASE VALUES (80000, 522555, 'Data Recovery', 'Debit', 'No', 1000, 7777, 24542);

SELECT *
FROM PURCHASE P
WHERE P.PurchaseNo = 80000;
```

PURCHASENO	RECEIPTNO	SERVICETYPE	PAYMENTTYP	GST	AMOUNT	SERVEDBY	CLIENTNO
80000	522555	Data Recovery	Cash	No	700	7777	24542

Task 3 – Views

3.1 Virtual View

```
CREATE OR REPLACE VIEW V_DEPT_AMOUNT AS
SELECT D.DEPTNO, D.DNAME, AVG(P.AMOUNT) AS DAVGAMT, MAX(P.AMOUNT) AS DMAXAMT,
MIN(P.AMOUNT) AS DMINAMT, SUM(P.AMOUNT) AS DTOTAL
FROM DEPT D, PURCHASE P, EMP E
WHERE E.DEPTNO = D.DEPTNO
AND E.EMPNO = P.SERVEDBY
GROUP BY D.DEPTNO, D.DNAME;
```

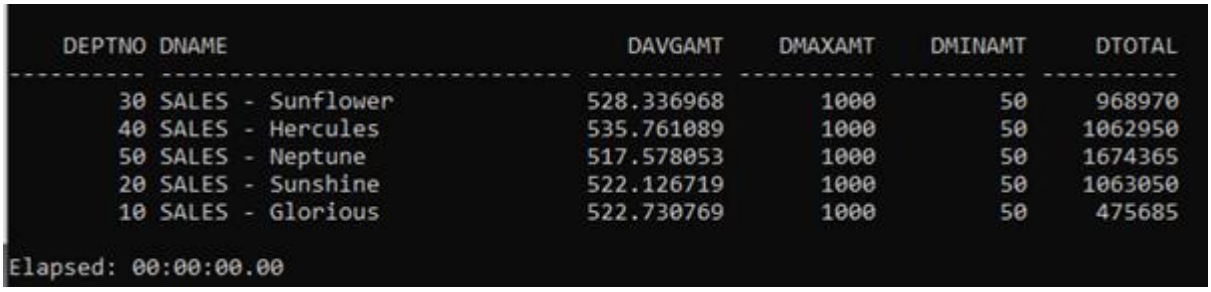


DEPTNO	DNAME	DAVGAMT	DMAXAMT	DMINAMT	DTOTAL
30	SALES - Sunflower	528.336968	1000	50	968970
40	SALES - Hercules	535.761089	1000	50	1062950
50	SALES - Neptune	517.578053	1000	50	1674365
20	SALES - Sunshine	522.126719	1000	50	1063050
10	SALES - Glorious	522.730769	1000	50	475685

Elapsed: 00:00:00.04

3.2 Materialised View

```
CREATE MATERIALIZED VIEW MV_DEPT_AMOUNT
BUILD IMMEDIATE
AS
SELECT D.DEPTNO, D.DNAME, AVG(P.AMOUNT) AS DAVGAMT, MAX(P.AMOUNT) AS DMAXAMT,
MIN(P.AMOUNT) AS DMINAMT, SUM(P.AMOUNT) AS DTOTAL
FROM DEPT D, PURCHASE P, EMP E
WHERE E.DEPTNO = D.DEPTNO
AND E.EMPNO = P.SERVEDBY
GROUP BY D.DEPTNO, D.DNAME;
```



DEPTNO	DNAME	DAVGAMT	DMAXAMT	DMINAMT	DTOTAL
30	SALES - Sunflower	528.336968	1000	50	968970
40	SALES - Hercules	535.761089	1000	50	1062950
50	SALES - Neptune	517.578053	1000	50	1674365
20	SALES - Sunshine	522.126719	1000	50	1063050
10	SALES - Glorious	522.730769	1000	50	475685

Elapsed: 00:00:00.00

3.3 Execution Time

The materialised view did speed up query processing, as can be viewed below:

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

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 58894710

-------------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |  9218 |   738K|    25  (12)| 00:00:01 |
|   1 |  HASH GROUP BY                |           |  9218 |   738K|    25  (12)| 00:00:01 |
|   2 |   NESTED LOOPS                |           |  9218 |   738K|    23   (5)| 00:00:01 |
|   3 |    NESTED LOOPS               |           |  9218 |   738K|    23   (5)| 00:00:01 |
|   4 |     NESTED LOOPS              |           |  9218 |   468K|    23   (5)| 00:00:01 |
|   5 |      TABLE ACCESS FULL        | PURCHASE  |  9218 |   234K|    22   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
|   6 |       TABLE ACCESS BY INDEX ROWID| EMP       |     1 |    26 |     0   (0)| 00:00:01 |
|*  7 |        INDEX UNIQUE SCAN         | PK_EMPNO  |     1 |       |     0   (0)| 00:00:01 |
|*  8 |      INDEX UNIQUE SCAN           | PK_DEPTNO |     1 |       |     0   (0)| 00:00:01 |
|   9 |     TABLE ACCESS BY INDEX ROWID  | DEPT      |     1 |    30 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

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

```
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.
```

3.3 Summary:

Clearly, the materialized view sped up the query processing by 00:00:00.04 according to the screenshots of elapsed time above.

Also, the cost is significantly less at 3 for the materialized view whereas the virtual view cost ranges from 25 to 22 depending on the operation. Also, number of bytes was lower at 270 for each materialized view operation compared to 738K for the select statements and nested loop operations for the virtual view.

The number of operations for the materialized view only has 2: one for the select statement and one to process the view. The virtual view has 1 select statement operations and then 9 number operations for the nested loops and other SQl logic. We can also see that the number of rows that were accessed is 9218 for just the select statement for a virtual view in comparison to 5 for the select statement of the materialized view.

3.4 Employee Views

3.4.1 Virtual View

```
CREATE OR REPLACE VIEW V_DEPT_EMP_AMOUNT AS
SELECT E.EMPNO, D.DEPTNO, COUNT(P.PurchaseNo) AS ETOTALPNUM,
AVG(P.AMOUNT) AS EAVGPURCH, MAX(P.AMOUNT) AS EMAXPURCH,
SUM(P.Amount) AS ETOTALSERVED
FROM DEPT D, PURCHASE P, EMP E
WHERE E.DEPTNO = D.DEPTNO
AND E.EMPNO = P.SERVEDBY
GROUP BY E.EMPNO, D.DEPTNO
ORDER BY D.DEPTNO ASC, SUM(P.Amount) DESC;
```

3.4.2 Materialized View

```
CREATE MATERIALIZED VIEW MV_DEPT_EMP_AMOUNT
BUILD IMMEDIATE
AS
SELECT E.EMPNO, D.DEPTNO, COUNT(P.PurchaseNo) AS ETOTALPNUM,
AVG(P.AMOUNT) AS EAVGPURCH, MAX(P.AMOUNT) AS EMAXPURCH,
SUM(P.Amount) AS ETOTALSERVED
FROM DEPT D, PURCHASE P, EMP E
WHERE E.DEPTNO = D.DEPTNO
AND E.EMPNO = P.SERVEDBY
GROUP BY E.EMPNO, D.DEPTNO
ORDER BY SUM(P.Amount) DESC, D.DEPTNO ASC;
```

Virtual View:

```
SELECT * FROM V_DEPT_EMP_AMOUNT;
```

Materialized View:

```
SELECT * FROM MV_DEPT_EMP_AMOUNT;
```

View output:

```
SQL> SELECT * FROM V_DEPT_EMP_AMOUNT;

     EMPNO     DEPTNO ETOTALPNUM  EAVGPURCH  EMAXPURCH ETOTALSERVED
---------- ---------- ---------- ---------- ---------- ------------
      1065         10        166 543.614458        990        90240
      1007         10        167 502.664671        990        83945
      1015         10        150 544.233333        995        81635
      1031         10        144 524.305556       1000        75500
      1009         10        138 546.195652       1000        75375
      1055         10        144 472.638889        985        68060
      1071         10          1        930        930          930
      1022         20        167 533.952096        995        89170
      1049         20        155 550.548387       1000        85335
      1039         20        158 529.683544        995        83690
      1017         20        146 543.390411       1000        79335

     EMPNO     DEPTNO ETOTALPNUM  EAVGPURCH  EMAXPURCH ETOTALSERVED
---------- ---------- ---------- ---------- ---------- ------------
      1028         20        145 544.827586        995        79000
      1036         20        142 545.387324        995        77445
      1002         20        144 525.243056       1000        75635
      1006         20        156 480.737179        975        74995
      1037         20        141 529.893617        985        74715
      1010         20        146 503.184932       1000        73465
      1013         20        139 502.769784       1000        69885
      1058         20        142 491.795775        995        69835
      1040         20        137 508.029197        995        69600
      1060         20        117  519.82906        975        60820
      1072         20          1        125        125          125

     EMPNO     DEPTNO ETOTALPNUM  EAVGPURCH  EMAXPURCH ETOTALSERVED
---------- ---------- ---------- ---------- ---------- ------------
      1020         30        167 543.323353        995        90735
      1067         30        168  529.22619       1000        88910
      1048         30        150 571.433333        990        85715
      1061         30        141 560.496454       1000        79030
      1044         30        147 534.931973        995        78635
      1001         30        152 504.638158        995        76705
      1069         30        147 517.823129       1000        76120
      1027         30        126 545.793651       1000        68770
      1053         30        141 486.737589        995        68630
      1052         30        131 521.030534        990        68255
      1064         30        128  527.03125        995        67460

     EMPNO     DEPTNO ETOTALPNUM  EAVGPURCH  EMAXPURCH ETOTALSERVED
---------- ---------- ---------- ---------- ---------- ------------
      1066         30        123 505.731707        995        62205
      1062         30        112 511.651786        990        57305
      1073         30          1        495        495          495
      1019         40        168 533.928571       1000        89700
      1021         40        155 575.064516       1000        89135
      1059         40        152 571.546053        995        86875
      1012         40        150 545.466667       1000        81820
      1003         40        150 531.766667        985        79765
      1004         40        135 584.481481        995        78905
      1005         40        146 506.335616        990        73925
      1042         40        148 480.641892        995        71135
```

3.5 Execution Time:

Virtual View:

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

```
75 rows selected.

Elapsed: 00:00:00.05
```

```
SQL> SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 740344440

-------------------------------------------------------------------------------
| Id  | Operation                     | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                   |  9218 |   702K|    25  (12)| 00:00:01 |
|   1 |  VIEW                         | V_DEPT_EMP_AMOUNT |  9218 |   702K|    25  (12)| 00:00:01 |
|   2 |   SORT ORDER BY               |                   |  9218 |   819K|    25  (12)| 00:00:01 |
|   3 |    NESTED LOOPS               |                   |  9218 |   819K|    24   (9)| 00:00:01 |
|   4 |     NESTED LOOPS              |                   |  9218 |   819K|    24   (9)| 00:00:01 |
|   5 |      VIEW                     | VW_GBC_6          |  9218 |   585K|    23   (5)| 00:00:01 |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
|   6 |       HASH GROUP BY           |                   |  9218 |   234K|    23   (5)| 00:00:01 |
|   7 |        TABLE ACCESS FULL      | PURCHASE          |  9218 |   234K|    22   (0)| 00:00:01 |
|*  8 |      INDEX UNIQUE SCAN        | PK_EMPNO          |     1 |       |     0   (0)| 00:00:01 |
|*  9 |     TABLE ACCESS BY INDEX ROWID| EMP              |     1 |    26 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Materialized View:

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

```
75 rows selected.

Elapsed: 00:00:00.02
```

```
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 |
--------------------------------------------------------------------------------
```

3.5 Summary:

Clearly, the materialized view sped up the query processing by 00:00:00.03 according to the screenshots of elapsed time above. Also, the cost is significantly less at 3 for the materialized view whereas the virtual view cost ranges from 25 to 22 depending on the operation. We can also see that the number of bytes was lower at 300 for each materialized operation compared to 702K for the select statement for the virtual view. The number of operations for the materialized view only has 2: one for the select statement and one to process the view. The virtual view has 1 select statement operations and then 9 number operations for the nested loops and other SQl logic.

Task 4 – Indexes

4.1 Number of purchases

```
SELECT COUNT(*) AS RECEIPT_BOOKS
FROM (SELECT TRUNC(SUBSTR(P2.ReceiptNo, 1, 3)) AS Books,
                (SELECT COUNT(1)
                FROM PURCHASE P
                WHERE TRUNC(SUBSTR(P.ReceiptNo, 1, 3)) =
                TRUNC(SUBSTR(P2.ReceiptNo, 1, 3))) AS NumOfBooks
FROM PURCHASE P2) Specified
WHERE Specified.NumOfBooks >= 10;
```

```
RECEIPT_BOOKS
-------------
         7896

Elapsed: 00:00:41.85
```

4.2 Function-based Index

Setting up a plan for query:

```
EXPLAIN PLAN FOR SELECT COUNT(*) AS RECEIPT_BOOKS
FROM (SELECT TRUNC(SUBSTR(P2.ReceiptNo, 1, 3)) AS Books,
                (SELECT COUNT(1)
                FROM PURCHASE P
                WHERE TRUNC(SUBSTR(P.ReceiptNo, 1, 3)) =
                TRUNC(SUBSTR(P2.ReceiptNo, 1, 3))) AS NumOfBooks
FROM PURCHASE P2) Specified
WHERE Specified.NumOfBooks >= 10;
```

Execution Plan without an Index:

```
SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 2870517031

-------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |     1 |    13 |  6498   (3)| 00:00:01 |
|   1 |  SORT AGGREGATE     |          |     1 |    13 |            |          |
|*  2 |   FILTER            |          |       |       |            |          |
|   3 |    TABLE ACCESS FULL| PURCHASE |  9218 |   117K|    22   (0)| 00:00:01 |
|   4 |    SORT AGGREGATE   |          |     1 |    13 |            |          |
|*  5 |     TABLE ACCESS FULL| PURCHASE |    92 |  1196 |    22   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
```

Now by creating an index on this table:

```
CREATE INDEX BOOK_INDEX ON PURCHASE (TRUNC(SUBSTR(ReceiptNo, 1, 3)));
```

```
RECEIPT_BOOKS
-------------
         7896

Elapsed: 00:00:00.04
SQL>
```

Execution Plan with Index in Place:

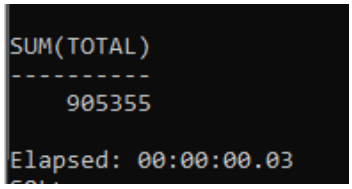
```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 589993641

-------------------------------------------------------------------------------
| Id  | Operation           | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |            |     1 |    13 |   310   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE     |            |     1 |    13 |            |          |
|*  2 |   FILTER            |            |       |       |            |          |
|   3 |    TABLE ACCESS FULL| PURCHASE   |  9218 |   117K|    22   (0)| 00:00:01 |
|   4 |    SORT AGGREGATE   |            |     1 |    13 |            |          |
|*  5 |     INDEX RANGE SCAN| BOOK_INDEX |    92 |  1196 |     1   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
```

4.2 Summary:

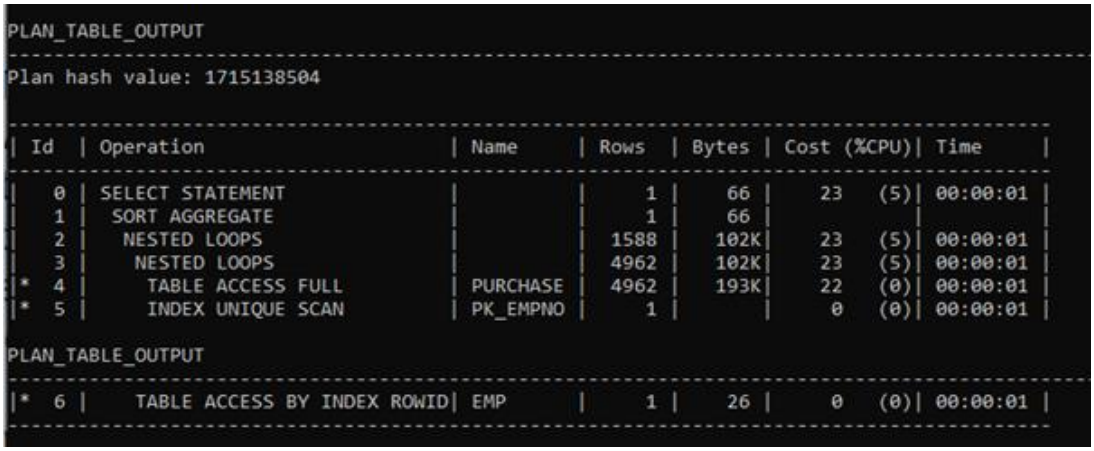
With the index in place, the query speed up incredibly. As we can see it took the query 41.85 seconds without the index in place whereas when an index was created on the table, it only took 00.04 seconds to process. This is a significant difference in execution time. The cost is also less at 310 + 22 + 1 = 333 total with an index and 6498 + 22 + 22 = 6542 without an index. Therefore a cost difference of 6209. Therefore using an index is a great decision for this query.

4.3 Total amount from sales for Department 50 without software

```
SELECT SUM(TOTAL)
FROM (SELECT P.AMOUNT AS TOTAL
      FROM PURCHASE P, DEPT D, EMP E
      WHERE E.EMPNO = P.SERVEDBY
      AND D.DEPTNO = E.DEPTNO
      AND D.DEPTNO = 50
      AND P.PURCHASENO NOT IN (SELECT P2.PURCHASENO
                               FROM PURCHASE P2
                               WHERE INSTR(P2.ServiceType, 'Software') > 0));
```




```
EXPLAIN PLAN FOR SELECT SUM(TOTAL)
FROM (SELECT P.AMOUNT AS TOTAL
      FROM PURCHASE P, DEPT D, EMP E
      WHERE E.EMPNO = P.SERVEDBY
      AND D.DEPTNO = E.DEPTNO
      AND D.DEPTNO = 50
      AND P.PURCHASENO NOT IN (SELECT P2.PURCHASENO
                               FROM PURCHASE P2
                               WHERE INSTR(P2.ServiceType, 'Software') > 0));

SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);
```



4.4 Service Index

```
CREATE INDEX SERVICE_INDEX ON PURCHASE(INSTR(ServiceType, 'Software'));
```

```
SUM(TOTAL)
----------
    905355

Elapsed: 00:00:00.02
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 3392947326

------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |     1 |    91 |    23   (5)| 00:00:01 |
|   1 |  SORT AGGREGATE              |               |     1 |    91 |            |          |
|   2 |   NESTED LOOPS ANTI          |               |  2950 |   262K|    23   (5)| 00:00:01 |
|   3 |    NESTED LOOPS              |               |  2950 |   187K|    23   (5)| 00:00:01 |
|   4 |     TABLE ACCESS FULL        | PURCHASE      |  9218 |   351K|    22   (0)| 00:00:01 |
|*  5 |     TABLE ACCESS BY INDEX ROWID| EMP         |     1 |    26 |     0   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
|*  6 |      INDEX UNIQUE SCAN       | PK_EMPNO      |     1 |       |     0   (0)| 00:00:01 |
|*  7 |    TABLE ACCESS BY INDEX ROWID | PURCHASE    |     1 |    26 |     0   (0)| 00:00:01 |
|*  8 |     INDEX UNIQUE SCAN        | PK_PURCHASENO |     1 |       |     0   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
```

4.4 Summary:

This service index only just speeds up the query. The cost is the same for both with and without the index implemented. Therefore, implementing an index does not really help for this particular query.

4.5 Number of purchases with same PType, SType and GST

```
SELECT SUM(NUMPURCH) AS Number_of_Purchases
FROM (SELECT COUNT(P.PurchaseNo) AS NUMPURCH
      FROM PURCHASE P
      GROUP BY P.ServiceType, P.PaymentType, P.GST
      HAVING COUNT(P.PurchaseNo) >= 1000);
```

```
NUMBER_OF_PURCHASES
-------------------
               2202
```

4.6 Index on 4.5 Query

A non-clustered index would be the best choice for this query. Non-clustered indexes do not sort the physical data inside the table. A non-clustered index locates the table content in one place and the index is located in another. This is what allows more than one index to be applied to a table.

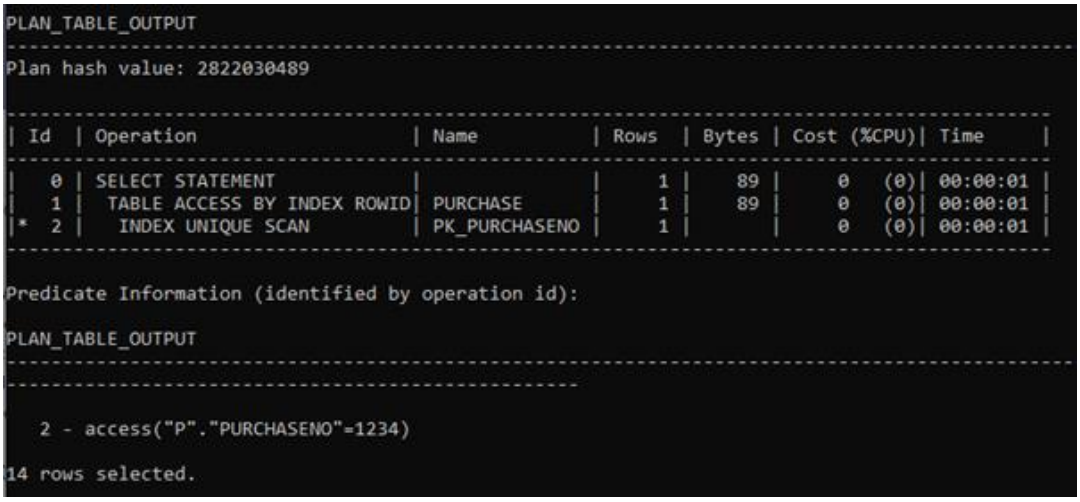
In particular, bitmap Indexes would be the best option because they are better on finite fields. They work best for columns that have a low cardinality. As GST contains Boolean data: “Yes” or “No” options and the other columns, ServiceType and PaymentType, also only have a few options, these tables have a low cardinality.

```
CREATE BITMAP INDEX PURCHASENO_INDEX ON PURCHASE(ServiceType);

CREATE BITMAP INDEX PURCHASENO_INDEX ON PURCHASE(PaymentType);

CREATE BITMAP INDEX PURCHASENO_INDEX ON PURCHASE(GST);
```

Task 5 – Execution Plan

5.1 Purchase number Oracle Optimisation:

```
EXPLAIN PLAN FOR SELECT *
FROM PURCHASE P
WHERE P.PurchaseNo = 1234;
SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);
```

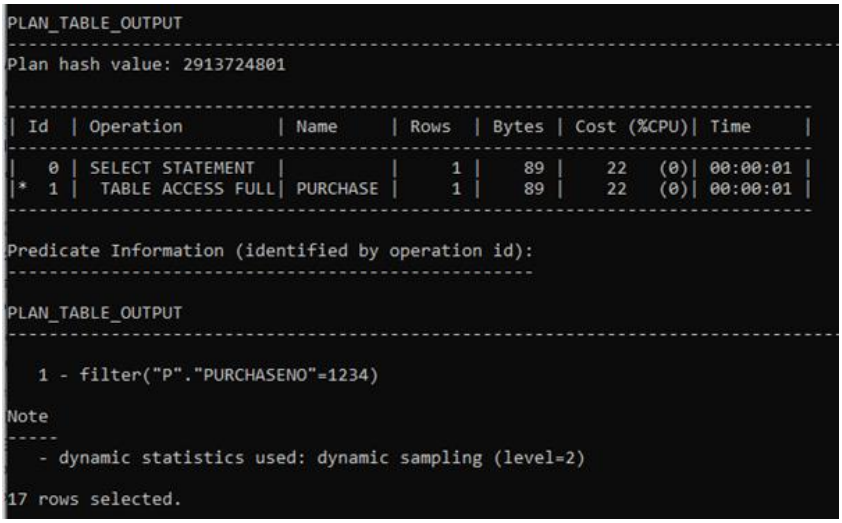


```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 2822030489

-----------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |     1 |    89 |     0   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| PURCHASE      |     1 |    89 |     0   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | PK_PURCHASENO |     1 |       |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------

Predicate Information (identified by operation id):

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

   2 - access("P"."PURCHASENO"=1234)

14 rows selected.
```

In this execution plan, the index on PK_PURCHASENO is used in a unique scan operation to evalue the WHERE clause. It then returns a rowID from the index. So, then rows are located by index (TABLE ACCESS BY INDEX ROWID) which means that the full table does not have to be accessed. The SELECT only returns rows satisfying the WHERE clause. From the plan, we can see that 89 bytes were accessed for both the SELECT and TABLE ACESS BY INDEX ROWID operations.

5.2 Dropping Constraint affects Execution Plan

```
Drop Primary Key Constraint:
ALTER TABLE PURCHASE
DROP CONSTRAINT PK_PURCHASENO;

EXPLAIN PLAN FOR SELECT *
FROM PURCHASE P
WHERE P.PurchaseNo = 1234;
SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY);
```



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

-----------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |     1 |    89 |    22   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| PURCHASE |     1 |    89 |    22   (0)| 00:00:01 |
-----------------------------------------------------------------------------

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

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

   1 - filter("P"."PURCHASENO"=1234)

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

17 rows selected.
```

In this execution plan, every row in the PURCHASE table is accessed and the WHERE clause condition is checked and dynamic sampling was used. One difference between this plan and the 5.1 plan is that this query selects 17 rows rather than 14. Also, the Cost for this execution plan is 22 for both operations whereas with the constraint in place, there is no cost for any operations.