*** EXPERIMENT NO: 04 ***

Aim : To execute different SQL join operations, sub-queries and correlated queries on a multi-relation database.

Problem Statement: Use the **SalesCo** database established in Experiment-02 with the below mentioned schemata to execute the listed queries involving join operations, sub-queries of different kinds and correlated queries.

CUSTOMER (C_CODE, LNAME , FNAME, C_AREA, C_PHONE, BALANCE)

INVOICE (INV_NUM, C_CODE, INV_DATE)

LINE (INV_NUM, L_NUM, P_CODE, L_UNITS, L_PRICE)

PRODUCT (P_CODE, DESCRIPT, P_DATE, QTY, P_MIN, P_PRICE, P_DISC, V_CODE)

VENDOR (V_CODE, V_NAME, V_CONTACT, V_AREA, V_PHO NE, V_STATE, V_ORDER)

Author : Bhavesh Kewalramani

Roll No : 025 [5A]

Date : 28-August-2021

****

QUERY-01: Write SQL code to create a table PART without any tuple from PRODUCT such that it includes product code-PT_CODE, product description- PT_DESC, the unit price-PT_PRICE and the supplier code. Now populate PART with the tuples fetching the contents from PRODUCT.
For the PART table created, compare its schema with PRODUCT for the common attributes. Observe all the constraints on PART table (use USER_CONSTRAINTS) and state your inferences.

****

```
CREATE TABLE PART AS (
SELECT P_CODE AS PT_CODE, DESCRIPT AS PT_DESC, P_PRICE AS PT_PRICE,V_CODE
FROM PRODUCT
WHERE 1=2);
```

```
Table created.
```

```
SELECT *
FROM PART;
```

```
no rows selected
```

```
      INSERT INTO PART
      (PT_CODE,PT_DESC,PT_PRICE,V_CODE)
      SELECT P_CODE,DESCRIPT,P_PRICE,V_CODE
      FROM PRODUCT;

17 rows created.

      SELECT *
      FROM PART;

PT_CODE PT_DESC                PT_PRICE     V_CODE
------- -------------------- ---------- ----------
AB112   Power Drill              109.99      25595
SB725   7.25in Saw Blade          14.99      21344
SB900   9.00 in Saw Blade         17.49      21344
JB012   Jigsaw 12in Blade        109.92      24288
JB008   Jigsaw 8in Blade          99.87      24288
CD00X   Cordless Drill            38.95      25595
CH10X   Claw Hammer                9.95      21225
SH100   Sledge Hammer              14.4
RF100   Rat Tail File              4.99      21344
HC100   Hicut Chain Saw          256.99      24288
PP101   PVC Pipe                   5.87
MC001   Metal Screw                6.99      21225
WC025   2.5in wide Screw           8.45      21231
SM48X   Steel Malting Mesh       119.95      25595
HW15X   Ball Pin Hammer            17.5      24992
AB111   Reversible Drill            435      24992
PP102   PVC Pipe                  15.25      24992

17 rows selected.

      DESC PART

 Name                                      Null?    Type
----------------------------------------- --------- ----------------------------
 PT_CODE                                   NOT NULL CHAR(5)
 PT_DESC                                   NOT NULL VARCHAR2(30)
 PT_PRICE                                  NOT NULL NUMBER(6,2)
 V_CODE                                             NUMBER(5)

      DESC PRODUCT
```

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
 P_CODE                                    NOT NULL CHAR(5)
 DESCRIPT                                  NOT NULL VARCHAR2(30)
 P_DATE                                    NOT NULL DATE
 QTY                                       NOT NULL NUMBER(4)
 P_MIN                                     NOT NULL NUMBER(3)
 P_PRICE                                   NOT NULL NUMBER(6,2)
 P_DISC                                    NOT NULL NUMBER(2,2)
 V_CODE                                             NUMBER(5)
```

The PT_CODE,PT_DESC,PT_PRICE,V_CODE Of table PART is similar to P_CODE,DESCRIPT,P_PRICE,V_CODE of table PRODUCT. Therefore the structure of both tables is also same for both containing common attributes with different names.

```
      SELECT TABLE_NAME, CONSTRAINT_NAME, CONSTRAINT_TYPE
      FROM USER_CONSTRAINTS
      WHERE UPPER(TABLE_NAME) IN ('PRODUCT','PART');
```

```
TABLE_NAME                     CONSTRAINT_NAME                C
------------------------------ ------------------------------ -
PART                           SYS_C007657                    C
PART                           SYS_C007658                    C
PART                           SYS_C007659                    C
PRODUCT                        SYS_C007410                    C
PRODUCT                        SYS_C007411                    C
PRODUCT                        SYS_C007412                    C
PRODUCT                        SYS_C007413                    C
PRODUCT                        SYS_C007414                    C
PRODUCT                        SYS_C007415                    C
PRODUCT                        SYS_C007416                    C
PRODUCT                        PRODUCT_CK_P_MIN               C

TABLE_NAME                     CONSTRAINT_NAME                C
------------------------------ ------------------------------ -
PRODUCT                        PRODUCT_PK_P_CODE              P
PRODUCT                        PRODUCT_VENDOR_FK_V_CODE       R

13 rows selected.
```

The check constraints have been implemented automatically when the table PRODUCT was copied to table PART but the primary and foreign key constraints didn’t got implemented during copying of table PRODUCT to table PART.

****

QUERY-02: Write a SQL code that will list all vendors who have supplied a part (You must ensure that only unique V_CODE values are displayed). Also retrieve information on vendors referenced in PRODUCT who have supplied products with prices in excess of 10 units.

****

```
SELECT DISTINCT V.V_CODE,V.V_NAME

FROM VENDOR V

RIGHT JOIN PART P

ON V.V_CODE=P.V_CODE;
```

```
    V_CODE V_NAME

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

     24288 Justin Stores


     21344 Gomez Sons

     24992 Hindustan Forge

     25595 HighEnd Supplies

     21225 Bryson, Inc.

     21231 GnB Supply


7 rows selected.
```

```
SELECT V.V_CODE,V.V_NAME

FROM VENDOR V

RIGHT JOIN PRODUCT P

ON P.V_CODE=V.V_CODE

WHERE P.QTY>10;
```

```
    V_CODE V_NAME
---------- ------------------------------
     21225 Bryson, Inc.
     21225 Bryson, Inc.
     21231 GnB Supply
     21344 Gomez Sons
     21344 Gomez Sons
     21344 Gomez Sons
     24288 Justin Stores
     25595 HighEnd Supplies
     25595 HighEnd Supplies
     24992 Hindustan Forge
     24992 Hindustan Forge
     24992 Hindustan Forge


13 rows selected.
```

****

QUERY-03: Write SQL code that will retrieve the product particulars for the parts with the highest and the lowest price. Use this query to retrieve the product particulars for the parts with the highest and the lowest inventory value (In both outputs the highest price products should be listed first).

****

```
SELECT P1.*
FROM PART P1,PRODUCT P2
WHERE P1.PT_CODE=P2.P_CODE
AND P2.P_PRICE=(SELECT MAX(P_PRICE)
                FROM PRODUCT)
UNION
SELECT P1.*
FROM PART P1,PRODUCT P2
WHERE P1.PT_CODE=P2.P_CODE
AND P2.P_PRICE=(SELECT MIN(P_PRICE)
                FROM PRODUCT);
```

```
PT_CODE PT_DESC                          PT_PRICE     V_CODE
------- ------------------------------ ---------- ----------
AB111   Reversible Drill                      435      24992
RF100   Rat Tail File                        4.99      21344
```

```
SELECT P1.*
FROM PART P1,PRODUCT P2
WHERE P1.PT_CODE=P2.P_CODE
AND P2.P_PRICE*P2.QTY=(SELECT MAX(P_PRICE*QTY)
                       FROM PRODUCT)
UNION
SELECT P1.*
FROM PART P1,PRODUCT P2
WHERE P1.PT_CODE=P2.P_CODE
AND P2.P_PRICE*P2.QTY=(SELECT MIN(P_PRICE*QTY)
                       FROM PRODUCT);
```

```
PT_CODE PT_DESC                          PT_PRICE     V_CODE
------- ------------------------------ ---------- ----------
AB111   Reversible Drill                      435      24992
SH100   Sledge Hammer                        14.4
```

****
QUERY-04: Write SQL code that will retrieve the product particulars for all products whose prices (largest first) exceed the average product price of the inventory. Also list the number of products that are supplied by each vendor.
****

```
SELECT P_CODE,DESCRIPT,P_PRICE
FROM PRODUCT
WHERE P_PRICE > (SELECT AVG(P_PRICE)
                 FROM PRODUCT)
ORDER BY P_PRICE DESC;
```

```
P_CODE DESCRIPT                          P_PRICE
------ ----------------------------- ----------
AB111  Reversible Drill                      435
HC100  Hicut Chain Saw                    256.99
SM48X  Steel Malting Mesh                 119.95
AB112  Power Drill                        109.99
JB012  Jigsaw 12in Blade                  109.92
JB008  Jigsaw 8in Blade                    99.87

6 rows selected.
```

```
      SELECT V_CODE,COUNT(*)
      FROM PRODUCT
      GROUP BY V_CODE;
```

```
    V_CODE   COUNT(*)
---------- ----------
     25595          3
                    2
     24992          3
     21231          1
     21225          2
     24288          3
     21344          3

7 rows selected.
```

****

QUERY-05: Write SQL code to generate a listing of the number of products in the inventory supplied by each vendor that has prices average below 10.

Extend this query to generate a listing of the total cost of products for each vendor - TOT_COST, such that the total cost exceeds 400.00 and the high value vendor is placed last.

****

```
      SELECT V_CODE,COUNT(*),AVG(P_PRICE)
      FROM PRODUCT
      GROUP BY V_CODE
      HAVING AVG(P_PRICE)<10;
```

```
    V_CODE   COUNT(*) AVG(P_PRICE)
---------- ---------- ------------
     21231          1         8.45
     21225          2         8.47
```

```
SELECT V_CODE,COUNT(*),SUM(QTY*P_PRICE) AS TOT_COST
FROM PRODUCT
HAVING SUM(QTY*P_PRICE) > 400
GROUP BY V_CODE
ORDER BY TOT_COST;
```

```
    V_CODE   COUNT(*)   TOT_COST
---------- ---------- ----------
     21344          3    1009.07
                    2    1218.76
     21225          2    1431.13
     21231          1    2002.65
     25595          3    3506.42
     24288          3    4305.47
     24992          3     7987.5

7 rows selected.
```

****

QUERY-06: Write SQL code to create a view - PRODUCT_STATS from PRODUCT that generate a report that shows a summary of total product cost - TOT_COST, and statistics on the quantity on hand [maximum - MX_QTY, minimum - MN_QTY, average -AV _QTY] for each vendor.

****

```
SELECT V_CODE,SUM(QTY*P_PRICE) AS TOT_COST,MAX(QTY) AS MX_QTY,MIN(QTY) AS
MN_QTY,AVG(QTY) AS AV_QTY
FROM PRODUCT
GROUP BY V_CODE;
```

```
    V_CODE   TOT_COST     MX_QTY     MN_QTY     AV_QTY
---------- ---------- ---------- ---------- ----------
     25595    3506.42         18          8 12.6666667
              1218.76        188          8         98
     24992     7987.5         50         15         35
     21231    2002.65        237        237        237
     21225    1431.13        172         23       97.5
     24288    4305.47         11          6 8.33333333
     21344    1009.07         43         18         31

7 rows selected.
```

****
QUERY-07: Write a SQL query that will list for each customer who has made purchases, the customer number, the customer balance and the aggregate purchase amount.
****

```
      SELECT I.C_CODE, C.BALANCE,SUM(L.L_UNITS*L.L_PRICE) AS PURCHASE_AMOUNT
      FROM INVOICE I
      JOIN CUSTOMER C
      ON I.C_CODE=C.C_CODE
      JOIN LINE L
      ON L.INV_NUM=I.INV_NUM
      GROUP BY I.C_CODE,C.BALANCE
      ORDER BY I.C_CODE;
```

```
    C_CODE    BALANCE PURCHASE_AMOUNT
---------- ---------- ---------------
     10011          0          479.57
     10012     345.86          153.85
     10014          0          422.77
     10015          0           34.97
     10018     216.55           34.87
     10020        700             350

6 rows selected.
```

****

QUERY-08: Modify Query-07 to include the number of individual product purchases made by each customer. (If the customer's invoice is based on three products, one per L_NUM, then count 3 product purchases. For example, customer 10011 generated 3 invoices, which contained a total of 5 lines, each representing a product purchase.)

****

```
       SELECT I.C_CODE, C.BALANCE,SUM(L.L_UNITS*L.L_PRICE) AS
                                        PURCHASE_AMOUNT,SUM(L.L_NUM) AS PURCHASES
       FROM INVOICE I
       JOIN CUSTOMER C
       ON I.C_CODE=C.C_CODE
       JOIN LINE L
       ON L.INV_NUM=I.INV_NUM
       GROUP BY I.C_CODE,C.BALANCE
       ORDER BY I.C_CODE;
```

```
    C_CODE    BALANCE PURCHASE_AMOUNT  PURCHASES
---------- ---------- --------------- ----------
     10011          0          479.57          8
     10012     345.86          153.85          6
     10014          0          422.77         13
     10015          0           34.97          3
     10018     216.55           34.87          3
     10020        700             350          1

6 rows selected.
```

****

QUERY-09: Write SQL query to produce the total purchase per invoice (The invoice total is the sum of the product purchases in the LINE that corresponds to the INVOICE). Further, produce a listing showing invoice numbers with corresponding invoice total identified to a customer (Use GROUP BY on C_CODE). Also generate a listing showing the number of invoices and the total purchase amounts by customer

****

```
       SELECT I.INV_NUM,SUM(L.L_UNITS*L.L_PRICE) AS INVOICE_TOTAL
       FROM INVOICE I
       JOIN LINE L
       ON I.INV_NUM=L.INV_NUM
       GROUP BY I.INV_NUM;
```

```
   INV_NUM INVOICE_TOTAL
---------- -------------
      1001         24.94
      1002          9.98
      1003        153.85
      1004         34.87
      1005         70.44
      1006        397.83
      1007         34.97
      1008        399.15
      1009           350

9 rows selected.
```

```
SELECT C.C_CODE,I.INV_NUM,SUM(L.L_UNITS*L.L_PRICE) AS INVOICE_TOTAL
FROM INVOICE I
JOIN CUSTOMER C
ON I.C_CODE=C.C_CODE
JOIN LINE L
ON L.INV_NUM=I.INV_NUM
GROUP BY C.C_CODE,I.INV_NUM
ORDER BY C.C_CODE;
```

```
    C_CODE    INV_NUM INVOICE_TOTAL
---------- ---------- -------------
     10011       1002          9.98
     10011       1005         70.44
     10011       1008        399.15
     10012       1003        153.85
     10014       1001         24.94
     10014       1006        397.83
     10015       1007         34.97
     10018       1004         34.87
     10020       1009           350

9 rows selected.
```

```
      SELECT C.C_CODE,COUNT(DISTINCT I.INV_NUM) AS TOTAL_NO_INVOICES,
                                 SUM(L.L_UNITS*L.L_PRICE) AS TOTAL_PURCHASE_AMOUNT
      FROM INVOICE I
      JOIN CUSTOMER C
      ON I.C_CODE=C.C_CODE
      JOIN LINE L
      ON L.INV_NUM=I.INV_NUM
      GROUP BY C.C_CODE
      ORDER BY C.C_CODE;
```

```
    C_CODE TOTAL_NO_INVOICES TOTAL_PURCHASE_AMOUNT
---------- ----------------- ---------------------
     10011                 3                479.57
     10012                 1                153.85
     10014                 2                422.77
     10015                 1                 34.97
     10018                 1                 34.87
     10020                 1                   350

6 rows selected.
```

****
QUERY-10: Write SQL code to find the customer balance summary for all customers who have not made purchases during the current invoicing period. Use this query to generate a summary of the customer balance characteristics (the output should include the minimum, maximum and average balances over all purchases).
****

```
      SELECT C.C_CODE,C.BALANCE
      FROM CUSTOMER C
      MINUS
      SELECT C.C_CODE,C.BALANCE
      FROM CUSTOMER C
      JOIN INVOICE I
      ON I.C_CODE=C.C_CODE;
```

```
    C_CODE    BALANCE
---------- ----------
     10010          0
     10013     536.75
     10016     221.19
     10017     768.93
     10019          0
```

```
SELECT MAX(BALANCE) AS MX_BALANCE ,MIN(BALANCE) AS MN_BALANCE,
                                              AVG(BALANCE) AS AVG_BALANCE
FROM (SELECT C_CODE,BALANCE
       FROM CUSTOMER
       WHERE C_CODE IN (
                           SELECT DISTINCT C_CODE
                           FROM INVOICE
                        )
       GROUP BY C_CODE,BALANCE
);
```

```
MX_BALANCE MN_BALANCE AVG_BALANCE
---------- ---------- -----------
       700          0  210.401667
```

****

QUERY-11: Write SQL code to create a table INV_CUSTOMER that includes INV_NUM as QUOTE_ID, INV_DATE as QUOTE_DT and C_NAME combining FNAME and LNAME with embedded space. Enforce the entity integrity constraint on QUOTE_ID. (You may use subquery to create the table structure. Ensure that the created table is empty).

Now, use SELECT subquery to populate INV_CUSTOMER using the information contained in INVOICE and CUSTOMER.

****

```
CREATE TABLE INV_CUSTOMER AS (
SELECT I.INV_NUM AS QUOTE_ID,I.INV_DATE AS QUOTE_DT,
                                        C.FNAME||' '||C.LNAME AS C_NAME
FROM CUSTOMER C NATURAL JOIN INVOICE I
WHERE 1=2);
```

```
Table created.


       SELECT *
       FROM INV_CUSTOMER;

no rows selected


       ALTER TABLE INV_CUSTOMER
       ADD
       CONSTRAINT INV_CUSTOMER_PK_QUOTE_ID PRIMARY KEY (QUOTE_ID);

Table altered.


       SELECT TABLE_NAME, CONSTRAINT_NAME, CONSTRAINT_TYPE
       FROM USER_CONSTRAINTS
       WHERE UPPER(TABLE_NAME)='INV_CUSTOMER';

TABLE_NAME                     CONSTRAINT_NAME                C
------------------------------ ------------------------------ -
INV_CUSTOMER                   SYS_C007664                    C
INV_CUSTOMER                   SYS_C007665                    C


       INSERT INTO INV_CUSTOMER
       (QUOTE_ID,QUOTE_DT,C_NAME)
       SELECT I.INV_NUM,I.INV_DATE,C.FNAME||’ ‘||C.LNAME
       FROM CUSTOMER C,INVOICE I
       WHERE C.C_CODE=I.C_CODE;

9 rows created.



       SELECT *
       FROM INV_CUSTOMER
       ORDER BY QUOTE_ID;
```

```
  QUOTE_ID QUOTE_DT  C_NAME
---------- --------- --------------------
      1001 16-JAN-20 Bill Johnson
      1002 16-JAN-20 Elena Johnson
      1003 16-JAN-20 Kathy Smith
      1004 17-JAN-20 Ming Lee
      1005 17-JAN-20 Elena Johnson
      1006 17-JAN-20 Bill Johnson
      1007 17-JAN-20 Julia Samuels
      1008 17-JAN-20 Elena Johnson
      1009 18-JUL-21 Bhavesh Kewalram

9 rows selected.
```

****

QUERY-12: Modify **Query-11** to create a view INV_CUTOMER_VW with the mentioned composition. Do not enforce entity integrity as in **Query-11.** Populate this view in similar manner .

State the problem(s) are encountered. Try populating taking alternative approach you knew. Does that work?

Now create the same view (use CREATE OR REPLACE VIEW) such that the view is populated at the creation time. Check the view contents. Now try inserting a record - 1011, Jagat Narayan, 12-Mar-2020, and observe the result.

Three non-discounted products - ZZ999 & AB212 (vendor 24992) and SH200 were added to the inventory. The details are as below...

SH200, Sledge Hammer, 05-Jul-2020, 10, 3, 25.8

ZZ999, Cordless Drill, 10-Jul-2020, 200, 40, 25.5

AB212, Power Drill, 03-Aug-2020, 15, 3, 275.0

****

```
CREATE OR REPLACE VIEW INV_CUSTOMER_VW
AS SELECT I.INV_NUM AS QUOTE_ID,I.INV_DATE AS QUOTE_DT,
                                       C.FNAME||' '||C.LNAME AS C_NAME
FROM INVOICE I,CUSTOMER C
WHERE I.C_CODE=C.C_CODE;
```

```
View created.
```

```
INSERT INTO INV_CUSTOMER_VW
(QUOTE_ID,QUOTE_DT,C_NAME)
(
      SELECT I.INV_NUM,I.INV_DATE,C.FNAME||' '||C.LNAME
      FROM CUSTOMER C JOIN INVOICE I
      ON I.C_CODE=C.C_CODE
);
```

```
(QUOTE_ID,QUOTE_DT,C_NAME)
                   *
ERROR at line 2:
ORA-01733: virtual column not allowed here
```

The insertion with the above query gives an error and similar error occur when we try to insert using different methods. The error is because Views do not work when ‘||’ like opertions are used and when joins are used with conditions.

```
CREATE OR REPLACE VIEW INV_CUSTOMER_VW
AS SELECT *
FROM INV_CUSTOMER;
```

```
View created.
```

```
INSERT INTO INV_CUSTOMER_VW
VALUES (1011,'12-Mar-2020','Jagat Narayan');
```

```
1 row created.
```

```
SELECT *
FROM INV_CUSTOMER_VW
ORDER BY QUOTE_ID;
```

```
  QUOTE_ID QUOTE_DT  C_NAME
---------- --------- ---------------------
      1001 16-JAN-20 Bill Johnson
      1002 16-JAN-20 Elena Johnson
      1003 16-JAN-20 Kathy Smith
      1004 17-JAN-20 Ming Lee
      1005 17-JAN-20 Elena Johnson
      1006 17-JAN-20 Bill Johnson
      1007 17-JAN-20 Julia Samuels
      1008 17-JAN-20 Elena Johnson
      1009 18-JUL-21 Bhavesh Kewalram
      1011 12-MAR-20 Jagat Narayan

10 rows selected.


       INSERT INTO PRODUCT
       VALUES ('SH200','Sledge Hammer','05-JUL-2020',10,3,25.8,0,NULL);

1 row created.


       INSERT INTO PRODUCT
       VALUES('ZZ999','Cordless Drill','10-JUL-2020',200,40,25.5,0,24992);

1 row created.


       INSERT INTO PRODUCT
       VALUES('AB212','Power Drill','03-AUG-2020',200,3,275.0,0,NULL);

1 row created.


       SELECT *
       FROM PRODUCT;

P_CODE DESCRIPT             P_DATE     QTY P_MIN    P_PRICE   P_DISC     V_CODE
------ -------------------- --------- ---- ----- ---------- -------- ----------
AB112  Power Drill          03-NOV-19    8     5     109.99     .000      25595
SB725  7.25in Saw Blade     13-DEC-19   32    15      14.99     .050      21344
SB900  9.00 in Saw Blade    13-NOV-19   18    12      17.49     .000      21344
SH200  Sledge Hammer        05-JUL-20   10     3       25.8     .000
ZZ999  Cordless Drill       10-JUL-20  200    40       25.5     .000      24992
JB012  Jigsaw 12in Blade    30-DEC-19    8     5     109.92     .050      24288
```

```
JB008  Jigsaw 8in Blade     24-DEC-19    6     5      99.87     .050      24288
CD00X  Cordless Drill       20-JAN-20   12     5      38.95     .050      25595
CH10X  Claw Hammer          20-JAN-20   23    10       9.95     .100      21225
SH100  Sledge Hammer        02-JAN-20    8     5       14.4     .050
RF100  Rat Tail File        15-DEC-19   43    20       4.99     .000      21344
HC100  Hicut Chain Saw      07-FEB-20   11     5     256.99     .050      24288
PP101  PVC Pipe             20-FEB-20  188    75       5.87     .000
MC001  Metal Screw          01-MAR-20  172    75       6.99     .000      21225
WC025  2.5in wide Screw     24-FEB-20  237   100       8.45     .000      21231
SM48X  Steel Malting Mesh   17-JAN-20   18     5     119.95     .100      25595
HW15X  Ball Pin Hammer      21-JAN-21   40    10       17.5     .000      24992
AB111  Reversible Drill     22-JUL-21   15     5        435     .100      24992
PP102  PVC Pipe             23-JUL-21   50    12      15.25     .020      24992
AB212  Power Drill          03-AUG-20  200     3        275     .000

20 rows selected.
```

****

QUERY-13: Write SQL code using subquery to list the supplier number and supplier name of only those suppliers who supply some products.

****

```
      SELECT DISTINCT P.V_CODE,V.V_NAME
      FROM PRODUCT P
      LEFT JOIN (SELECT V_CODE,V_NAME
                 FROM VENDOR
                ) V
      ON P.V_CODE=V.V_CODE;
```

```
    V_CODE V_NAME
---------- ------------------------------
     24288 Justin Stores

     21344 Gomez Sons
     24992 Hindustan Forge
     25595 HighEnd Supplies
     21225 Bryson, Inc.
     21231 GnB Supply

7 rows selected.
```

****

QUERY-14: Write SQL code using subquery that will compute the average price of all products. Modify the query to compute the average price of all products based on the product description.

****

```
      SELECT SUM(P_PRICE)/COUNT(P_CODE) as AVG_PRICE
      FROM (
            SELECT P_CODE,P_PRICE
            FROM PRODUCT
      );
```

```
  AVG_PRICE
-----------
    80.6425
```

```
      SELECT DISTINCT DESCRIPT,SUM(P_PRICE)/COUNT(DESCRIPT) AS AVG_PRICE
      FROM (SELECT DESCRIPT,P_PRICE
           FROM PRODUCT
      )
      GROUP BY DESCRIPT;
```

```
DESCRIPT              AVG_PRICE
-------------------- ----------
Hicut Chain Saw          256.99
Ball Pin Hammer            17.5
7.25in Saw Blade          14.99
9.00 in Saw Blade         17.49
Jigsaw 12in Blade        109.92
Jigsaw 8in Blade          99.87
Metal Screw                6.99
2.5in wide Screw           8.45
Rat Tail File              4.99
Sledge Hammer              20.1
PVC Pipe                  10.56
Reversible Drill            435
Power Drill             192.495
```

```
Claw Hammer                 9.95
Steel Malting Mesh        119.95
Cordless Drill            32.225

16 rows selected.
```

****

QUERY-15: Write SQL code using subquery that will list product code, product description and unit product price for all products having the unit price higher than or equal to the average product price.

****

```
      SELECT P_CODE,DESCRIPT,P_PRICE
      FROM PRODUCT
      WHERE P_PRICE>=(
                        SELECT AVG(P_PRICE)
                        FROM PRODUCT
      );
```

```
P_CODE DESCRIPT                P_PRICE
------ -------------------- ----------
AB112  Power Drill              109.99
JB012  Jigsaw 12in Blade        109.92
JB008  Jigsaw 8in Blade          99.87
HC100  Hicut Chain Saw          256.99
SM48X  Steel Malting Mesh       119.95
AB111  Reversible Drill            435
AB212  Power Drill                 275

7 rows selected.
```

****

QUERY-16: Write SQL code that will list supplier number, name and contact person for suppliers who do not supply any product in current season.

****

```
    SELECT DISTINCT P.V_CODE,V.V_NAME,V.V_CONTACT
    FROM PRODUCT P,VENDOR V
    WHERE P.V_CODE=V.V_CODE AND TO_CHAR(P.P_DATE,'MON-YY')<>(
                                        SELECT TO_CHAR(MAX(P_DATE),'MON-YY')
                                        FROM PRODUCT
    );
```

```
    V_CODE V_NAME                         V_CONTACT
---------- ------------------------------ --------------------
     21344 Gomez Sons                     Mark Welder
     21231 GnB Supply                     Ted Jones
     24288 Justin Stores                  Gracy Yu
     21225 Bryson, Inc.                   Bella Ford
     25595 HighEnd Supplies               Smith Rust
     24992 Hindustan Forge                Nancy Kewalramani

6 rows selected.
```

****

QUERY-17: Write SQL code using subquery to update the product price to the average product price, but only for the products that are supplied by vendors not belonging to the state 'TN' and 'KY'.

Add a line for invoice number 1003 to include a 10 items of the product named ZZ999

-1003, 4, ZZ999, 10, 25.5

****

```
    UPDATE PRODUCT P
    SET P.P_PRICE=(SELECT AVG(P_PRICE) FROM PRODUCT)
    WHERE EXISTS(
                 SELECT P.V_CODE,V.V_CODE,V.V_STATE FROM VENDOR V
                 WHERE P.V_CODE=V.V_CODE
                 AND V.V_STATE IN (
                                   SELECT DISTINCT V_STATE
                                   FROM VENDOR
                                   WHERE UPPER(V_STATE) NOT IN ('TN','KY')
                 )
    );
```

```
3 rows updated.
```

```
       INSERT INTO LINE
       VALUES (1003,4,'ZZ999',10,25.5);

1 row created.
```

****
QUERY-18: Write SQL code using subquery to find all the customers (include customer numbers, first name and last name) who have ordered some kind of a blade. Now find the customers who have ordered the part "Power Drill".
****

```
       SELECT DISTINCT C.C_CODE,C.FNAME,C.LNAME
       FROM CUSTOMER C
       JOIN INVOICE I
       ON C.C_CODE=I.C_CODE
       JOIN LINE L
       ON I.INV_NUM=L.INV_NUM
       AND L.P_CODE IN (
                        SELECT P_CODE
                        FROM PRODUCT
                        WHERE UPPER(DESCRIPT) LIKE '%BLADE'
       );

    C_CODE FNAME      LNAME
---------- ---------- ----------
     10014 Bill       Johnson
     10012 Kathy      Smith
     10015 Julia      Samuels


       SELECT C.C_CODE,C.FNAME,C.LNAME
       FROM CUSTOMER C
       JOIN INVOICE I
       ON C.C_CODE=I.C_CODE
       JOIN LINE L
       ON I.INV_NUM=L.INV_NUM
       AND L.P_CODE IN (
                          SELECT PT_CODE
                          FROM PART
                          WHERE UPPER(PT_DESC)='POWER DRILL'
       );
```

```
no rows selected
```

```
**********************************************************************************
QUERY-19:   Write SQL code using subquery to find all the customers who have
purchased  a drill or a hammer or a saw.
**********************************************************************************
```

```
       SELECT DISTINCT C.C_CODE,C.FNAME,C.LNAME
       FROM CUSTOMER C
       JOIN INVOICE I
       ON C.C_CODE=I.C_CODE
       JOIN LINE L
       ON I.INV_NUM=L.INV_NUM
       AND L.P_CODE IN (
                          SELECT PT_CODE
                          FROM PART
                          WHERE UPPER(PT_DESC) LIKE '%HAMMER'
       )
       OR L.P_CODE IN (
                          SELECT PT_CODE
                          FROM PART
                          WHERE UPPER(PT_DESC) LIKE '%DRILL'
       ) OR L.P_CODE IN (
                          SELECT PT_CODE
                          FROM PART
                          WHERE UPPER(PT_DESC) LIKE '%SAW'
       );
```

```
    C_CODE FNAME      LNAME
---------- ---------- ----------
     10018 Ming       Lee
     10014 Bill       Johnson
     10012 Kathy      Smith
     10020 Bhavesh    Kewalram
     10011 Elena      Johnson
     10015 Julia      Samuels

6 rows selected.
```

****

QUERY-20: Write SQL code using subquery to list all products with the total quantity sold greater than the average quantity sold.

****

```
SELECT P.P_CODE,P.DESCRIPT,SUM(L.L_UNITS) AS TOTAL_QUANTITY_SOLD
FROM PRODUCT P
JOIN LINE L
ON L.P_CODE=P.P_CODE
HAVING SUM(L.L_UNITS) > (
                          SELECT AVG(L_UNITS)
                          FROM LINE
)
GROUP BY P.P_CODE,P.DESCRIPT;
```

```
P_CODE DESCRIPT             TOTAL_QUANTITY_SOLD
------ -------------------- -------------------
SB725  7.25in Saw Blade                       8
ZZ999  Cordless Drill                        10
HW15X  Ball Pin Hammer                       20
RF100  Rat Tail File                          6
PP101  PVC Pipe                              17
CH10X  Claw Hammer                            5

6 rows selected.
```

****

QUERY-21: Write SQL code using subquery to list all customers who have purchased products HC100 and JB012.

****

```
SELECT DISTINCT C.*
FROM (
      SELECT C.*
      FROM CUSTOMER C,INVOICE I,LINE L
     WHERE C.C_CODE=I.C_CODE
      AND I.INV_NUM=L.INV_NUM
     AND L.P_CODE IN ('HC100','JB012')
) C;
```

```
    C_CODE LNAME      FNAME           C_AREA    C_PHONE    BALANCE
---------- ---------- ---------- ---------- ---------- ----------
     10014 Johnson    Bill              615    2455533          0
```

****

QUERY-22: Write SQL code using subquery that will for all products list the product price and the difference between each product's price and the average product price. Ensure that the average product price is also displayed.

****

```
SELECT P_CODE,P_PRICE,(
                          SELECT AVG(P_PRICE) AS AVG_PRODUCT_PRICE
                          FROM PRODUCT
) AS AVERAGE_PRODUCT_PRICE,P_PRICE-(
                                       SELECT AVG(P_PRICE)
                                       FROM PRODUCT
) AS DIFFERENCE
FROM PRODUCT
GROUP BY P_CODE,P_PRICE;
```

```
P_CODE    P_PRICE AVERAGE_PRODUCT_PRICE DIFFERENCE
------ ---------- --------------------- ----------
SH200        25.8                79.294    -53.494
ZZ999        25.5                79.294    -53.794
CH10X        9.95                79.294    -69.344
HW15X        17.5                79.294    -61.794
SB900       17.49                79.294    -61.804
JB012      109.92                79.294     30.626
PP102       15.25                79.294    -64.044
SH100        14.4                79.294    -64.894
WC025        8.45                79.294    -70.844
AB212         275                79.294    195.706
SB725       14.99                79.294    -64.304
JB008       99.87                79.294     20.576
PP101        5.87                79.294    -73.424
HC100      256.99                79.294    177.696
AB111         435                79.294    355.706
AB112       80.64                79.294      1.346
RF100        4.99                79.294    -74.304
MC001        6.99                79.294    -72.304
```

```
CD00X       80.64                79.294      1.346
SM48X       80.64                79.294      1.346

20 rows selected.
```

****

QUERY-23: Write SQL code using correlated query to list all product sales in which the units sold value is greater than the average units sold value for that product (as opposed to the average for all products).

****

```
      SELECT L.P_CODE,P.DESCRIPT,L.L_UNITS
      FROM PRODUCT P JOIN LINE L
      ON P.P_CODE=L.P_CODE
      JOIN (
            SELECT P_CODE,AVG(L_UNITS) AS AVG_UNITS
            FROM LINE
            GROUP BY P_CODE
      ) A
      ON A.P_CODE=L.P_CODE
      AND L.L_UNITS > A.AVG_UNITS;
```

```
P_CODE DESCRIPT                L_UNITS
------ -------------------- ----------
SB725  7.25in Saw Blade              5
RF100  Rat Tail File                 3
CH10X  Claw Hammer                   2
PP101  PVC Pipe                     12
```

****

QUERY-24: Write SQL code using correlated query to list all customers who have placed an order. (Use EXISTS clause in SELECT statement).

****

```
SELECT C.*
FROM CUSTOMER C
WHERE EXISTS (
                    SELECT 1
                    FROM INVOICE I
                    WHERE C.C_CODE=I.C_CODE
);
```

```
    C_CODE LNAME      FNAME          C_AREA    C_PHONE    BALANCE
---------- ---------- ---------- ---------- ---------- ----------
     10014 Johnson    Bill              615    2455533          0
     10011 Johnson    Elena             713    2753455          0
     10012 Smith      Kathy             615    2873453     345.86
     10018 Lee        Ming              713    2323234     216.55
     10015 Samuels    Julia             713    2345432          0
     10020 Kewalram   Bhavesh           904    3562098        700

6 rows selected.
```

Conclusion:

One of the most fundamental aspects of how joins operate is that we create a condition that compares a value from the first table (typically a primary key) with a value from the second table (usually a foreign key). If the condition involving these two values evaluates to true, the row containing the first value is linked to the row containing the second value.

When the join condition is fulfilled, the INNER JOIN joins rows from two tables.

LEFT JOIN is similar to an inner join, with the exception that rows from the first table are added to the join table regardless of the join condition's assessment.

The RIGHT JOIN is similar to an inner join, with the exception that rows from the second table are added to the join table regardless of the join condition's assessment.

A FULL JOIN is the result of combining left and right joins.

CROSS JOIN CROSS JOIN CROSS JOIN There isn't a join condition. The cross product of all rows across both tables yields the join table, which is the result of matching every row from the first table with the second table.

When utilizing joins, our queries may get cumbersome at times, especially when dealing with two or more JOINS. To make things easier, we may alias table and column names to make our query shorter. We may also utilize aliasing to provide more context for query results.

Finally, the result of a join query may be retrieved using a variety of techniques. Subqueries provide us another way to query the database and get the same or comparable results as we would if we utilized a JOIN clause.

A correlated subquery is one in which certain clauses refer to the outer query's column expressions.

Viva Questions:

1. **What is a correlated query?**

   The correlated subquery is linked to a conventional subquery by using an internal query to fill the outer query with result data. A corresponding subquery performs the external query several times, once every row that the internal query returns; it is handled by adding a column in the subquery to a parent query.

2. **What are the three types of results that a subquery can return?**

   Subqueries can return different types of information:

   - Scalar subquery will return a single value.
   - Column subquery will return a single column of one or more values.
   - Row subquery will return a single row of one or more values.
   - Table subquery will return a table of one or more rows of one or more columns.

3. **What do you understand by an inline subquery? Give example.**

   The subquery is a component of the FROM clause in inline subquery, and it replaces the table name. This subquery must be uncorrelated, which means it cannot reference any fields from the enclosing query.

   An inline subquery in a FROM clause may look like this:

   ```
   SELECT MAX(AVG_P_PRICE) AS MAX_AVG_P_PRICE
   FROM (
   SELECT AVG(P_PRICE) AS AVG_P_PRICE
   FROM PRODUCT
   GROUP BY P_CODE);
   ```

   ```
   MAX_AVG_P_PRICE
   ---------------
               435
   ```

4. What do you understand by Theta Join and Self-Join?

The equality indication is not required when comparing join columns. A theta join is a join operation that uses a generic join condition.

EXAMPLE:

```
   SELECT FNAME,LNAME,C_AREA
   FROM CUSTOMER JOIN INVOICE
   ON CUSTOMER.C_CODE=INVOICE.C_CODE;
```

```
FNAME      LNAME          C_AREA
---------- ---------- ----------
Elena      Johnson           713
Elena      Johnson           713
Elena      Johnson           713
Kathy      Smith             615
Bill       Johnson           615
Bill       Johnson           615
Julia      Samuels           713
Ming       Lee               713
Bhavesh    Kewalram          904

9 rows selected.
```

A natural join operation can be used to join two or more tables, but it can also be used to join a single table. The table is linked to itself through a Self Join, where a single column of the table is compared to itself. The table name appears twice in the FROM clause of a SELECT query when a column is compared to itself.

As a result, you must be able to refer to the same table's name again. At least one alias name can be used to do this. The same is true for the column names in a SELECT statement's join condition. The qualified names are used to identify both column names.

EXAMPLE:

```
   SELECT DISTINCT C1.FNAME,C1.LNAME,C1.C_AREA
   FROM CUSTOMER C1 JOIN CUSTOMER C2
   ON C1.C_AREA=C2.C_AREA
   AND C1.BALANCE<>C2.BALANCE;
```

```
FNAME      LNAME          C_AREA
---------- ---------- ----------
```

```
Anne       Harris            615
Gustav     Ford              615
Chris      Paul              615
James      Anderson          615
Kathy      Smith             615
Bill       Johnson           615
Ming       Lee               713
Elena      Johnson           713
Walter     Green             615
Julia      Samuels           713

10 rows selected.
```

5. List the execution differences while including an USING clause and an ON clause with JOIN query.

If you don't want to connect using all of the common columns, you may utilize the USING clause. The WHERE clause, as well as the columns mentioned in the USING clause, cannot include any qualifiers. USING clause allows you to specify the join key by name.

When two tables' column names don't match, the ON clause is used to connect them. In the WHERE clause, the join criteria are removed from the filter conditions. ON clause allows you to specify the column names for join keys in both tables.

The following is the distinction between using clause and on clause:

When using the USING clause to connect two or more tables, the column names of both tables must be the same, regardless of which table is being linked, but the ON clause allows column names to change.