USE DATABASE DEMODATABASE;

CREATE OR REPLACE TABLE "CONSUMER\_COMPLAINTS"

(

"DATE\_RECEIVED" STRING ,

"PRODUCT NAME" VARCHAR2(50) ,

"SUB\_PRODUCT" VARCHAR2(40) ,

"ISSUE" VARCHAR2(100),

"SUB\_ISSUE" VARCHAR2(100),

"CONSUMER COMPLAINT NARRATIVE" VARCHAR2(100),

"Company Public Response" STRING,

"Company" VARCHAR(30),

"State Name" CHAR(4),

"Zip Code" INTEGER,

"Tags" VARCHAR(10),

"Consumer Consent Provided" CHAR(6),

"Submitted via" STRING,

"Date Sent to Company" STRING,

"Company Response to Consumer" VARCHAR(50),

"Timely Response" CHAR(4),

"CONSUMER DISPUTED" CHAR(4),

"COMPLAINT\_ID" NUMBER(12,0) NOT NULL PRIMARY KEY

);

SELECT \* FROM "DEMODATABASE"."PUBLIC"."CONSUMER\_COMPLAINTS";

select \* from "DEMODATABASE"."CLONE\_SCHEMA"."DEMO\_INFO\_TABLE";

CREATE OR REPLACE TABLE OWNER

(

OwnerID INTEGER NOT NULL PRIMARY KEY ,

Name VARCHAR2(20),

Surname STRING,

StreetAddress VARCHAR2(50),

City STRING,

State CHAR(4),

StateFull STRING,

ZipCode INTEGER

);

CREATE OR REPLACE TABLE PETS

(

PetID VARCHAR(10) NOT NULL PRIMARY KEY,

Name VARCHAR(20),

Kind STRING,

Gender CHAR(7),

Age INTEGER,

OwnerID INTEGER NOT NULL REFERENCES OWNER

);

SELECT \* FROM OWNER; -- 89 ROWS

SELECT \* FROM PETS;-- 100 ROWS

--SUBQUERY

SELECT \* FROM OWNER WHERE OWNERID IN (SELECT DISTINCT OWNERID FROM PETS);

SELECT NAME,SURNAME FROM OWNER WHERE OWNERID NOT IN (SELECT DISTINCT OWNERID FROM PETS WHERE KIND = 'Dog');

-- UNION ALL

SELECT OWNERID,NAME FROM OWNER

UNION ALL

SELECT OWNERID,NAME FROM PETS;

--EXCEPT CLAUSE

SELECT OWNERID,NAME FROM OWNER

EXCEPT

SELECT OWNERID,NAME FROM PETS;

--INTERSECT

SELECT OWNERID,NAME FROM OWNER

INTERSECT

SELECT OWNERID,NAME FROM PETS;

SELECT COUNT(DISTINCT OwnerID) from OWNER;

SELECT COUNT(DISTINCT PetID) from PETS;

-- NEED THE NAME OF OWNER & THEIR DOGS NAME ALONG WITH THEIR AGE ---- INNER JOIN

SELECT O.Name AS OWNER\_NAME,p.NAME AS PET\_NAME,p.age AS PET\_AGE

FROM OWNER o

INNER JOIN PETS p ON o.OwnerID = p.OwnerID;

--NEED THE NAME OF ALL THE OWNERS IRRESPECTIVE WETHER OR NOT THEY ARE HAVING PETS

SELECT O.Name AS OWNER\_NAME,p.NAME AS PET\_NAME,p.age AS PET\_AGE

FROM OWNER o

LEFT OUTER JOIN PETS p ON o.OwnerID = p.OwnerID;

--- COUNT OF PETS EACH OWNER HAS

SELECT O.Name AS OWNER\_NAME,COUNT(DISTINCT p.PETID)

FROM OWNER o

INNER JOIN PETS p ON o.OwnerID = p.OwnerID

GROUP BY 1

ORDER BY 2 DESC;

---RIGHT JOIN

SELECT O.Name AS OWNER\_NAME,p.NAME AS PET\_NAME,p.age AS PET\_AGE

FROM OWNER o

RIGHT JOIN PETS p ON o.OwnerID = p.OwnerID;

--FULL OUTER JOIN

SELECT O.\*,P.\*

FROM OWNER O

FULL OUTER JOIN PETS p ON o.OwnerID = p.OwnerID;

-- INFO OF ALL THE PETS HOLD BY THEIR OWNER

SELECT DISTINCT KIND FROM PETS;

SELECT KIND,COUNT(\*) FROM PETS

GROUP BY 1;

--- CROSS JOIN

SELECT O.\*,P.\*

FROM OWNER O

CROSS JOIN PETS p ;

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

CREATE OR REPLACE TABLE EMPLOYEE

(

EMPID INTEGER NOT NULL PRIMARY KEY,

EMP\_NAME VARCHAR2(20),

JOB\_ROLE STRING,

SALARY NUMBER(10,2)

);

INSERT INTO EMPLOYEE

VALUES('101','ANAND JHA','Analyst',50000);

INSERT INTO EMPLOYEE

VALUES(102,'ALex', 'Data Enginner',60000);

INSERT INTO EMPLOYEE

VALUES(103,'Ravi', 'Data Scientist',48000);

INSERT INTO EMPLOYEE

VALUES(104,'Peter', 'Analyst',98000);

INSERT INTO EMPLOYEE

VALUES(105,'Pulkit', 'Data Scientist',72000);

INSERT INTO EMPLOYEE

VALUES(106,'Robert','Analyst',100000);

INSERT INTO EMPLOYEE

VALUES(107,'Rishabh','Data Engineer',67000);

INSERT INTO EMPLOYEE

VALUES(108,'Subhash','Manager',148000);

INSERT INTO EMPLOYEE

VALUES(109,'Michaeal','Manager',213000);

INSERT INTO EMPLOYEE

VALUES(110,'Dhruv','Data Scientist',89000);

INSERT INTO EMPLOYEE

VALUES(111,'Amit Sharma','Analyst',72000);

DELETE FROM EMPLOYEE WHERE EMPID = 110;

SELECT \* FROM EMPLOYEE;

update employee set job\_role='Data Engineer'

where empid=102;

update employee set salary= 50000

where empid=104;

SELECT @temp\_var = EMP\_NAME WHERE EMPID = 104;

DECLARE

profit number(38,2);

revenue number(38,2);

cost number(38,2);

BEGIN

profit := revenue - cost;

return profit;

set my\_variable=10;

set my\_variable='example';

-------------------------------------------------------------WINDOW FUNCTIONS------------------------------------------------------------

-- SYNTAX : window\_function\_name(<exprsn>) OVER (<partition\_by\_clause> <order\_clause>)

--- display total salary based on job profile

SELECT JOB\_ROLE,SUM(SALARY) FROM EMPLOYEE

GROUP BY JOB\_ROLE;

-- display total salary along with all the records ()every row value

SELECT \* , SUM(SALARY) OVER() AS TOT\_SALARY

FROM EMPLOYEE;

-- display the total salary per job category for all the rows

SELECT \*,MAX(SALARY) OVER(PARTITION BY JOB\_ROLE) AS MAX\_JOB\_SAL

FROM EMPLOYEE;

select \*,max(salary) over(partition by JOB\_ROLE) as MAX\_SAL ,

min(salary) over(partition by JOB\_ROLE) as MIN\_SAL,

SUM(salary) over(partition by JOB\_ROLE) as TOT\_SAL

from Employee;

--ARRANGING ROWS WITHIN EACH PARTITION BASED ON SALARY IN DESC ORDDER

select \*,max(salary) over(partition by JOB\_ROLE ORDER BY SALARY DESC) as MAX\_SAL ,

min(salary) over(partition by JOB\_ROLE ORDER BY SALARY DESC) as MIN\_SAL,

SUM(salary) over(partition by JOB\_ROLE ORDER BY SALARY DESC) as TOT\_SAL

from Employee;

-- ROW\_NUMBER() It assigns a unique sequential number to each row of the table ...

SELECT \* FROM

(

SELECT \*,ROW\_NUMBER() OVER(PARTITION BY JOB\_ROLE ORDER BY SALARY DESC) AS PART\_ROW\_NUM

FROM EMPLOYEE

)

WHERE PART\_ROW\_NUM <=2;

/\* The RANK() window function, as the name suggests, ranks the rows within their partition based on the given condition.

In the case of ROW\_NUMBER(), we have a sequential number.

On the other hand, in the case of RANK(), we have the same rank for rows with the same value.

But there is a problem here. Although rows with the same value are assigned the same rank, the subsequent rank skips the missing rank.

This wouldn’t give us the desired results if we had to return “top N distinct” values from a table.

Therefore we have a different function to resolve this issue. \*/

SELECT \*,ROW\_NUMBER() OVER(PARTITION BY JOB\_ROLE ORDER BY SALARY DESC) AS ROW\_NUM ,

RANK() OVER(PARTITION BY JOB\_ROLE ORDER BY SALARY DESC) AS RANK\_ROW

FROM EMPLOYEE;

/\* The DENSE\_RANK() function is similar to the RANK() except for one difference, it doesn’t skip any ranks when ranking rows

Here, all the ranks are distinct and sequentially increasing within each partition.

As compared to the RANK() function, it has not skipped any rank within a partition. \*/

SELECT \* FROM

(

SELECT \*,ROW\_NUMBER() OVER(PARTITION BY JOB\_ROLE ORDER BY SALARY) AS ROW\_NUM ,

RANK() OVER(PARTITION BY JOB\_ROLE ORDER BY SALARY) AS RANK\_ROW,

DENSE\_RANK() OVER(PARTITION BY JOB\_ROLE ORDER BY SALARY) AS DENSE\_RANK\_ROW

FROM EMPLOYEE

)

WHERE DENSE\_RANK\_ROW <=2;