Following operations should be performed:

Create an ER diagram for the given airlines database.

-- Tables:-

create table customer(

customer\_id INTEGER,

first\_name VARCHAR(100),

last\_name VARCHAR(100),

date\_of\_birth DATE,

gender VARCHAR(1)

);

Create table passengers\_on\_flights (

aircraft\_id varchar(50),

route\_id INTEGER,

customer\_id INTEGER,

depart VARCHAR(100),

arrival VARCHAR(100),

seat\_num varchar(50),

class\_id varchar(50),

travel\_date date,

flight\_num INTEGER

);

Create table ticket\_details(

p\_date date,

customer\_id Integer,

aircraft\_id varchar(50),

class\_id varchar(50),

no\_of\_tickets Integer,

a\_code varchar(50),

price\_per\_ticket Integer,

brand varchar(100)

);

Create table routes

( Route\_id Integer,

Flight\_num Integer,

Origin\_airport varchar(50),

Destination\_airport varchar(50),

Aircraft\_id varchar(100),

Distance\_miles Integer

);

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

select count(\*) as count , 'Customer' as table\_name from customer

union all

select count(\*) as count , 'passengers\_on\_flights' as table\_name from passengers\_on\_flights

union all

select count(\*) as count , 'routes' as table\_name from routes

union all

select count(\*) as count , 'ticket\_details' table\_name from ticket\_details

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

Write a query to create route\_details table using suitable data types for the fields, such as route\_id, flight\_num, origin\_airport, destination\_airport, aircraft\_id, and distance\_miles. Implement the check constraint for the flight number and unique constraint for the route\_id fields. Also, make sure that the distance miles field is greater than 0.

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

Write a query to display all the passengers (customers) who have travelled in routes 01 to 25. Take data from the passengers\_on\_flights table.

select c.\* from customer c left join passengers\_on\_flights pof

on c.customer\_id = pof.customer\_id

where pof.route\_id between 1 and 25;

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

Write a query to identify the number of passengers and total revenue in business class from the ticket\_details table.

select count(\*) as number\_of\_passengers , sum(price\_per\_ticket) as revenue\_from\_bussiness\_class

from ticket\_details

where class\_id = 'Bussiness';

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

Write a query to display the full name of the customer by extracting the first name and last name from the customer table.

select first\_name ||' '||last\_name as full\_name, first\_name, last\_name

from customer;

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

Write a query to extract the customers who have registered and booked a ticket. Use data from the customer and ticket\_details tables.

select c.customer\_id, first\_name ||' '||last\_name as cutomer\_name,

td.class\_id,td.price\_per\_ticket,td.no\_of\_tickets,td.brand,td.a\_code

from customer c left join ticket\_details td

on c.customer\_id=td.customer\_id

order by c.customer\_id

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

Write a query to identify the customer’s first name and last name based on their customer ID and brand (Emirates) from the ticket\_details table.

select c.customer\_id, first\_name ||' '||last\_name as cutomer\_name,

td.class\_id,td.price\_per\_ticket,td.no\_of\_tickets,td.brand,td.a\_code

from customer c left join ticket\_details td

on c.customer\_id=td.customer\_id

where td.brand = 'Emirates'

order by c.customer\_id

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

Write a query to identify the customers who have travelled by Economy Plus class using Group By and Having clause on the passengers\_on\_flights table.

select c.customer\_id, first\_name ||' '||last\_name as cutomer\_name,

td.class\_id,td.price\_per\_ticket,td.no\_of\_tickets,td.brand,td.a\_code

from customer c left join ticket\_details td

on c.customer\_id=td.customer\_id

where

td.brand = 'Economy Plus'

order by c.customer\_id ;

select c.customer\_id,count(\*)

from customer c left join ticket\_details td

on c.customer\_id=td.customer\_id

where

td.class\_id = 'Economy Plus'

group by c.customer\_id;

select td.class\_id,count(\*)

from customer c left join ticket\_details td

on c.customer\_id=td.customer\_id

where

td.class\_id = 'Economy Plus'

group by td.class\_id;

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

Write a query to identify whether the revenue has crossed 10000 using the IF clause on the ticket\_details table.

select sum(price\_per\_ticket) as total\_revenue

from ticket\_details

having sum(price\_per\_ticket) > 10000;

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

Write a query to create and grant access to a new user to perform operations on a database.

CREATE USER new\_user WITH PASSWORD 'Abcd@1010';

GRANT USAGE ON SCHEMA public TO new\_user;

GRANT SELECT ON ALL TABLES IN SCHEMA public TO new\_user;

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

Write a query to find the maximum ticket price for each class using window functions on the ticket\_details table.

with cte as (select class\_id,

price\_per\_ticket,

row\_number() over (partition by class\_id order by price\_per\_ticket desc) as rn

from ticket\_details )

select \* from cte where rn=1;

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

Write a query to extract the passengers whose route ID is 4 by improving the speed and performance of the passengers\_on\_flights table.

select c.\* from customer c left join passengers\_on\_flights pof

on c.customer\_id = pof.customer\_id

where route\_id = 4;

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

For the route ID 4, write a query to view the execution plan of the passengers\_on\_flights table.

EXPLAIN select c.\* from customer c left join passengers\_on\_flights pof

on c.customer\_id = pof.customer\_id

where route\_id = 4;

XN Hash Join DS\_DIST\_ALL\_NONE (cost=0.62..1.36 rows=4 width=31)

" Hash Cond: (""outer"".customer\_id = ""inner"".customer\_id)"

-> XN Seq Scan on passengers\_on\_flights pof (cost=0.00..0.62 rows=3 width=4)

Filter: (route\_id = 4)

-> XN Hash (cost=0.50..0.50 rows=50 width=31)

-> XN Seq Scan on customer c (cost=0.00..0.50 rows=50 width=31)

XN Hash Join DS\_DIST\_ALL\_NONE:

This is a hash join operation between the two tables, "passengers\_on\_flights" and "customer," based on the join condition specified in the Hash Cond section. The join type "DS\_DIST\_ALL\_NONE" indicates that no distribution is required for the inner table during the join.

Hash Cond: ("outer".customer\_id = "inner".customer\_id):

This line specifies the join condition used in the hash join operation. It indicates that the join is performed based on the "customer\_id" column, which is present in both the outer and inner tables.

XN Seq Scan on passengers\_on\_flights pof:

This is a sequential scan operation on the "passengers\_on\_flights" table (aliased as "pof"). The cost of this scan is 0.00 and estimates that 3 rows will be scanned with a width of 4 bytes each. It also includes a filter condition (Filter: route\_id = 4) that is applied to the table.

XN Hash:

This is a hash operation, where the hash function is applied to the "customer" table before joining it with the "passengers\_on\_flights" table.

XN Seq Scan on customer c:

This is a sequential scan operation on the "customer" table (aliased as "c"). The cost of this scan is 0.00 and estimates that 50 rows will be scanned with a width of 31 bytes each.

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

Write a query to calculate the total price of all tickets booked by a customer across different aircraft IDs using rollup function.

SELECT country, SUM(salary) AS "Total Salary"

FROM EMPLOYEE

GROUP BY ROLLUP (country);

SELECT customer\_id, SUM(pric\_per\_ticket) AS "Total\_Revenue"

FROM ticket\_details

GROUP BY ROLLUP (customer\_id);

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

Write a query to create a view with only business class customers along with the brand of airlines.

create view bussiness\_cust as

select c.customer\_id,

first\_name || ' ' || last\_name as cutomer\_name,

td.class\_id,

td.price\_per\_ticket,

td.no\_of\_tickets,

td.brand,

td.a\_code

from customer c

left join ticket\_details td

on c.customer\_id = td.customer\_id

where td.class\_id = 'Bussiness'

order by c.customer\_id

;

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

Write a query to create a stored procedure to get the details of all passengers flying between a range of routes defined in run time. Also, return an error message if the table doesn't exist.

create or replace procedure public.passenger (in route\_num int)

LANGUAGE plpgsql

As $$

DECLARE

p1 varchar(50):= 'customer';

p2 varchar(50):= 'passenger\_on\_flight';

table\_cust\_exists boolean;

ticket\_details\_exists boolean;

BEGIN

-- Check if the customer table exists

SELECT EXISTS (

SELECT 1

FROM pg\_table\_def

WHERE

schemaname='public' and

tablename in ('customer')

) INTO table\_cust\_exists ;-- write another statement

-- Check if the ticket\_details table exists

SELECT EXISTS (

SELECT 1

FROM pg\_table\_def

WHERE

schemaname='public' and

tablename in ('ticket\_details')

) INTO ticket\_details\_exists;

IF table\_cust\_exists and ticket\_details\_exists THEN

RAISE INFO 'Both The Tables % exists.', p1 ;

select c.\* from customer c left join passengers\_on\_flights pof

on c.customer\_id = pof.customer\_id

where pof.route\_id = route\_num;

ELSE

if table\_cust\_exists then

RAISE INFO 'passengers\_on\_flights Table % does not exist.',p1;

else

RAISE INFO 'customer Table % does not exist.',p2;

END IF;

END IF;

END;

$$

;

call public.passenger (4);

-- To store the result use temp tables

-- Language

-- Raise info

--Table exists command

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

Write a query to create a stored procedure that extracts all the details from the routes table where the travelled distance is more than 2000 miles.

create or replace procedure public.routes (in miles int)

LANGUAGE plpgsql

As $$

DECLARE

p1 varchar(50):= 'customer';

p2 varchar(50):= 'passenger\_on\_flight';

table\_cust\_exists boolean;

ticket\_details\_exists boolean;

BEGIN

DROP TABLE IF EXISTS routes\_temp;

create temp table routes\_temp as select \* from routes where 1=2;

insert into routes\_temp (route\_id, flight\_num, origin\_airport, destination\_airport, aircraft\_id,

distance\_miles) select route\_id, flight\_num, origin\_airport, destination\_airport, aircraft\_id,

distance\_miles from routes where distance\_miles > miles ;

END;

$$

;

select \* from routes\_temp

call public.routes(2000);

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

Write a query to create a stored procedure that groups the distance travelled by each flight into three categories. The categories are, short distance travel (SDT) for >=0 AND <= 2000 miles, intermediate distance travel (IDT) for >2000 AND <=6500, and long-distance travel (LDT) for >6500.

create or replace procedure public.routes ()

LANGUAGE plpgsql

As $$

DECLARE

p1 varchar(50):= 'customer';

p2 varchar(50):= 'passenger\_on\_flight';

table\_cust\_exists boolean;

ticket\_details\_exists boolean;

BEGIN

DROP TABLE IF EXISTS routes\_distance\_miles;

create temp table routes\_distance\_miles as select \* from routes where 1=2;

insert into routes\_distance\_miles (route\_id, flight\_num, origin\_airport, destination\_airport, aircraft\_id,

distance\_miles) values select \* ,

case

when distance\_miles >=0 and distance\_miles <= 2000 then 'SDT'

when distance\_miles > 2000 and distance\_miles <= 6500 then 'INTEREDIATE'

when distance\_miles > 6500 then 'long\_distance'

END as distance\_miles

from routes; ;

END;

$$

;

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

Write a query to extract ticket purchase date, customer ID, class ID and specify if the complimentary services are provided for the specific class using a stored function in stored procedure on the ticket\_details table.

Condition:

If the class is Business and Economy Plus, then complimentary services are given as Yes, else it is No

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

20. Write a query to extract the first record of the customer whose last name ends with Scott using a cursor from the customer table.

with cte as (select \*, row\_number() over (partition by last\_name order by customer\_id ) as rn

from customer c

where c.last\_name like '%Scott')

select \* from cte where rn=1;