```
-- Question 2 query is not required as I have made the tables and uploaded
the data using Table Data Import Wizard.alter

-- Fetch customers who booked a trip on routes 1 to 25, sorted by
customer_id
select *
from customer
where customer_id in
(
select distinct customer_id
from passengers where route_id between 1 and 25
)
order by customer_id;

-- Count unique passengers and calculate total revenue for Business class
tickets
select count(distinct customer_id) as num_passengers, sum(no_of_tickets *
price_per_ticket) as total_revenue
from ticket
where class_id = 'Bussiness';

-- Combine first and last names into a full name for each customer
select concat(first_name, ' ', last_name) as full_name
from customer;

-- Fetch the first and last names of customers who have made ticket
bookings
select first_name, last_name
from customer
where customer_id in
(
select distinct b.customer_id
from customer a, ticket b
);

-- Alternative query for the same question
select c.first_name, c.last_name
from customer c
join ticket t on c.customer_id = t.customer_id;

-- Fetch the first and last names of customers who have booked tickets
with Emirates
select first_name, last_name
from customer
where customer_id in
(
select distinct customer_id
from ticket
where brand = 'Emirates'
);

-- -- Retrieve all customer details for those who have booked tickets in
Economy Plus class
select *
```

```
from customer a
inner join
(
select distinct customer_id
from passengers
where class_id = 'Economy Plus') b
on a.customer_id = b.customer_id;

-- Check if total revenue from ticket sales exceeds 10,000 and return a
status message
select if (
(select sum(no_of_tickets * price_per_ticket) as total_revenue
from ticket)> 10000,
'Crossed 10k', 'Not crossed 10k'
) as revenue_check;

-- Step 1: Create a new user with a secure password
create user if not exists 'newuser'@'localhost' identified by
'strongpassword123';
grant all privileges on scienceqtech.* to 'newuser'@'localhost';

-- Retrieve the highest ticket price for each class and sort by max price
select distinct class_id, max(price_per_ticket) over (partition by
class_id) as max_price
from ticket
order by max_price;

-- Retrieve all passenger details where the route ID is 4
explain select *
from passengers
where route_id = 4;

create index route_id on passengers (route_id);
explain select *
from passengers
where route_id = 4;

-- Calculate total ticket price per customer and aircraft, including
subtotals and grand total
select customer_id, aircraft_id, sum(price_per_ticket) as total_price
from ticket
group by customer_id, aircraft_id with rollup
order by customer_id, aircraft_id;

-- Create a view to retrieve all customer details along with the brand of
tickets booked in Business class
create view business_class_customers as
select a.*, b.brand
from customer a
inner join
(
select distinct customer_id , brand
from ticket
where class_id = 'Bussiness'
```

```
order by customer_id) b
on a.customer_id = b.customer_id;

select *
from business_class_customers;

-- Retrieve all customers who have booked tickets for routes 1 or 5
select *
from customer
where customer_id in (
select distinct customer_id
from passengers
where route_id in (1,5)
);

-- Create a stored procedure that dynamically retrieves customers based on
input route IDs
delimiter //
create procedure check_route (in rid varchar(255))
begin
      declare exit handler for SQLEXCEPTION
                  select 'please check if table customer/route id are
created - one/both are missing' Message;
      set @query = concat(
    'select * from customer where customer_id in (
    select distinct customer_id from passengers
    where find_in_set(route_id,"', rid,'"));'
    );

    prepare sql_query from @query;
    execute sql_query;
    deallocate prepare sql_query;
    end //
    delimiter ;

    -- Execute the stored procedure to fetch customers who traveled on
routes 1 and 5
    call check_route('1,5');

-- Create a stored procedure to retrieve routes with a distance greater
than 2000 miles
delimiter //
create procedure check_dist()
begin
      select *
    from routes
    where distance_miles > 2000;
end //
delimiter ;

-- Execute the stored procedure to fetch all routes with a distance over
2000 miles
call check_dist;
```

```
-- Categorize flights based on distance into Short-Distance (SDT),
Intermediate-Distance (IDT), or Long-Distance (LDT)
select flight_num, distance_miles,
      case
                  when distance_miles between 0 and 2000 then 'SDT'
                  when distance_miles between 2001 and 6500 then  'IDT'
                  else 'LDT'
    end as distance_category
    from routes;

-- Remove the existing group_dist function if it already exists.
DROP FUNCTION IF EXISTS group_dist;

-- Create a function to classify distances into SDT, IDT, or LDT.
delimiter //
create function group_dist(dist int)
returns varchar(10)
deterministic
   begin
                  declare dist_cat char(3);
                  if dist between 0 and 2000 then
                              set dist_cat = 'SDT';
                  elseif dist between 2001 and 6500 then
                              set dist_cat = 'IDT';
                  elseif dist > 6500 then
                              set dist_cat = 'LDT';
      end if;
    return (dist_cat);
end //
delimiter ;

-- Remove the existing group_dist_proc procedure if it already exists
DROP PROCEDURE IF EXISTS group_dist_proc;

-- Create a procedure to retrieve flight details with categorized distance
delimiter //
create procedure group_dist_proc()
begin
      select flight_num, distance_miles, group_dist(distance_miles) as
distance_category
    from routes;
end //
delimiter ;

-- Execute the stored procedure to get categorized flight distances
call group_dist_proc();

-- Categorize ticket holders for complimentary services based on class
type
select p_date, customer_id, class_id,
      case
                  when class_id in ('Bussiness', 'Economy Plus') then 'Yes'
                  else 'No'
    end as complrmrntary_service
```

```
from ticket;

-- Create a function to check if a ticket class qualifies for
complimentary service
delimiter //
create function check_comp_serv (cls varchar(15))
returns char(3)
deterministic
begin
      declare comp_ser char(3);
    if cls in ('Bussiness', 'Economy Plus') then
                set comp_ser = 'Yes';
      else
                set comp_ser = 'No';
      end if;
    return(comp_ser);
end //
delimiter ;

-- Create a procedure to retrieve ticket details along with complimentary
service eligibility
delimiter //
create procedure check_comp_serv_proc()
begin
      select p_date, customer_id, class_id, check_comp_serv(class_id) as
complementary_service
    from ticket;
end //
delimiter ;

-- Execute the procedure to check which tickets qualify for complimentary
service
call check_comp_serv_proc();

-- Retrieve one customer record with the last name 'Scott'
select *
from customer
where last_name = 'Scott'
limit 1;

-- Remove the existing stored procedure if it already exists
DROP PROCEDURE IF EXISTS cust_lname_scott;

-- Create a procedure to fetch customer records with last name 'Scott' and
store them in a table
delimiter //
create procedure cust_lname_scott()
begin
      declare c_id int;
    declare f_name varchar(20);
    declare l_name varchar(20);
    declare dob date;
    declare gen char(1);
```

```
    declare cust_rec cursor
    for
    select *
    from customer
    where last_name = 'Scott';

    create table if not exists cursor_table(

                            c_id int,

                            f_name varchar(20),

                            l_name varchar(20),

                            dob date,

                            gen char(1)
                                        );
      open cust_rec;
    fetch cust_rec into c_id, f_name, l_name, dob, gen;
    insert into cursor_table(c_id, f_name, l_name, dob, gen) values (c_id,
f_name, l_name, dob, gen);
    close cust_rec;

    select *
    from cursor_table;
end //
delimiter ;

-- Execute the procedure to fetch and insert customer records into
`cursor_table`
call cust_lname_scott();
```