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 * FROM passengers_on_flights WHERE 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 NO_OF_PASSENGERS, SUM(NO_OF_TICKETS * PRICE_PER_TICKET) AS TOTAL_REVENUE FROM ticket_details WHERE CLASS_ID = 'Business';

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

SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME) AS FULL_NAME FROM customers;

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

SELECT customers.CUSTOMER_ID, customers.FIRST_NAME, customers.LAST_NAME FROM customers INNER JOIN ticket_details ON customers.CUSTOMER_ID = ticket_details.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 customers.FIRST_NAME, customers.LAST_NAME, ticket_details.BRAND FROM customers INNER JOIN ticket_details ON customers.CUSTOMER_ID = ticket_details.CUSTOMER_ID WHERE ticket_details.BRAND = 'Emirates';

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 customers.FIRST_NAME, customers.LAST_NAME FROM customers INNER JOIN passengers_on_flights ON customers.CUSTOMER_ID =

passengers_on_flights.CUSTOMER_ID WHERE passengers_on_flights.CLASS_ID = ‘Economy Plus’ GROUP BY customers.FIRST_NAME, customers.LAST_NAME;

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

```
SELECT
   CASE WHEN SUM(PRICE_PER_TICKET* NO_OF_TICKETS) > 10000 THEN 'Yes'
ELSE 'No' END AS revenue_crossed_10000
FROM
   ticket_details;
```

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

```
CREATE USER Areesha IDENTIFIED BY ‘Ary123’;

GRANT SELECT, INSERT, UPDATE, DELETE ON airlines TO Areesha IDENTIFIED BY
‘Ary123’;
```

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

```
SELECT CLASS_ID, PRICE_PER_TICKET AS MAX_TICKET_PRICE
FROM(
SELECT CLASS_ID, PRICE_PER_TICKET,
ROW_NUMBER() OVER (PARTITION BY CLASS_ID ORDER BY PRICE_PER_TICKET
DESC )AS ROW_NUM FROM ticket_details) WHERE ROW_NUM = 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.

```
CREATE INDEX idx_route_id ON passengers_on_flights (ROUTE_ID);

SELECT * FROM passengers_on_flights 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 * FROM passengers_on_flights WHERE ROUTE_ID = 4;
```

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

```
SELECT CUSTOMER_ID, AIRCRAFT_ID, SUM(NO_OF_TICKETS * PRICE_PER_TICKET)
AS TOTAL_PRICE FROM ticket_details GROUP BY CUSTOMER_ID, AIRCRAFT_ID
WITH ROLLUP;
```

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

```
CREATE VIEW business_class_customers AS (SELECT customers.FIRST_NAME,
customers.LAST_NAME, ticket_details.BRAND FROM customers INNER JOIN
ticket_details ON customers.CUSTOMER_ID = ticket_details.CUSTOMER_ID
WHERE CLASS_ID = ‘Business’);
```

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.

```
DELIMITER ‘//’;
CREATE PROCEDURE passengers_details_in_range(IN MIN_ROUTE_ID int, IN
MAX_ROUTE_ID int)
BEGIN
DECLARE table_exists int;
SELECT COUNT(*) INTO table_exists FROM information_schema.tables WHERE
table_name = ‘passengers_on_flights’;
IF table_exists = 0 THEN
SELECT ‘Error: Table passengers_on_flights does not exist’ AS ErrorMessage;
ELSE
SELECT * FROM passengers_on_flights WHERE ROUTE_ID BETWEEN
MIN_ROUTE_ID AND MAX_ROUTE_ID;
END IF;
```

```
END//

DELIMITER ;

CALL passengers_details_in_range(4, 10);
```

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.

```
DELIMTER ‘//’;

CREATE PROCEDURE routes_greater_than_2000()
BEGIN
SELECT * FROM route_details WHERE DISTANCE_MILES > 2000;
END//

DELIMITER ;

CALL routes_greater_than_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.

```
DELIMITER //

CREATE PROCEDURE group_by_distance()
BEGIN
    SELECT
        ROUTE_ID,
        FLIGHT_NUM,
        ORIGIN_AIRPORT,
        DESTINATION_AIRPORT,
        AIRCRAFT_ID,
        DISTANCE_MILES,
        CASE
            WHEN DISTANCE_MILES BETWEEN 0 AND 2000 THEN 'SDT'
            WHEN DISTANCE_MILES > 2000 AND DISTANCE_MILES <= 6500 THEN 'IDT'
            WHEN DISTANCE_MILES > 6500 THEN 'LDT'
        END AS DISTANCE_CATEGORY
    FROM
        routes;
END //

DELIMITER ;
```

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***

```
CREATE FUNCTION determine_complimentarty_services(CLASS_ID varchar(50))
RETURNS varchar(3)
DETERMINISTIC
BEGIN
DECLARE complimentary varchar(3);
IF CLASS_ID IN (‘Business’, ‘Economy Plus’) THEN
SET complimentary = ‘Yes’
ELSE
SET complimentary = ‘No’
END IF;
RETURN complimentary
END//

CREATE PROCEDURE get_ticket_details_with_complimenatry_services()
BEGIN
SELECT
P_DATE, CUSTOMER_ID, CLASS_ID, determine_complimentary_services(CLASS_ID)
AS complimentary_services
```

```
FROM ticket_details;
END//

DELIMITER ;

CALL get_ticket_details_with_complimentary_services();
```

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

```
SELECT CUSTOMER_ID, FIRST_NAME, LAST_NAME FROM customers WHERE
LAST_NAME LIKE ‘%Scott’;
```