```
-- Air Cargo Analysis --

create database airlines;

use airlines;


select * from customer;

select * from passengers_on_flights;

select * from routes;

select * from ticket_details;


ALTER TABLE `airlines`.`customer`

MODIFY COLUMN `customer_id` VARCHAR(10) NOT NULL,

ADD PRIMARY KEY (`customer_id`);


ALTER TABLE `airlines`.`routes`

MODIFY COLUMN `route_id` VARCHAR(10) NOT NULL,

ADD PRIMARY KEY (`route_id`);


-- 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 --


CREATE TABLE route_details (

   route_id INT PRIMARY KEY,

   flight_num VARCHAR(4) CHECK (flight_num >= '1111' AND flight_num <= '1157'),

   origin_airport VARCHAR(255),

   destination_airport VARCHAR(255),

   aircraft_id INT,

   distance_miles DECIMAL(10, 2) CHECK (distance_miles > 0),

   UNIQUE (route_id)

);

select * from route_details;


-- 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 DISTINCT c.customer_id, c.first_name, c.last_name

FROM passengers_on_flights p

JOIN customer c ON p.customer_id = c.customer_id

WHERE p.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 class_id, COUNT(*) AS number_of_passengers, SUM(price_per_ticket * no_of_tickets) AS total_revenue
FROM ticket_details
WHERE class_id = 'Business'
GROUP BY class_id;
```

-- 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 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, c.first_name, c.last_name
FROM customer c
JOIN ticket_details t ON c.customer_id = t.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.first_name, c.last_name
FROM ticket_details t
JOIN customer c ON t.customer_id = c.customer_id
WHERE t.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 c.customer_id, c.first_name, c.last_name
FROM passengers_on_flights p
JOIN customer c ON p.customer_id = c.customer_id
WHERE p.class_id = 'Economy Plus'
GROUP BY c.customer_id, c.first_name, c.last_name
HAVING COUNT(*) > 0;
```

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

```
SELECT
   IF(SUM(price_per_ticket * no_of_tickets) > 10000, 'Yes', 'No') 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 'newuser'@'localhost' IDENTIFIED BY 'password';
GRANT SELECT, INSERT, UPDATE, DELETE ON dbname.* TO 'newuser'@'localhost';

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

SELECT DISTINCT
   class_id,
   MAX(price_per_ticket) OVER (PARTITION BY class_id) AS max_ticket_price
FROM ticket_details;

-- 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);

--  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(price_per_ticket * no_of_tickets) 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 c.customer_id,    c.first_name,    c.last_name,    t.brand AS airline_brand
FROM customer c
JOIN ticket_details t ON c.customer_id = t.customer_id
WHERE t.class_id = 'Business';

select * from business_class_customers;
```

```
-- 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 GetPassengersByRouteRange(IN start_route_id INT, IN end_route_id INT)
BEGIN
   DECLARE table_exists INT;

   -- Check if the table exists
   SELECT COUNT(*) INTO table_exists
   FROM information_schema.tables
   WHERE table_schema = DATABASE() AND table_name = 'passengers_on_flights';

   IF table_exists = 0 THEN
      SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'Table passengers_on_flights does not exist';
   ELSE
      SELECT *
      FROM passengers_on_flights
      WHERE route_id BETWEEN start_route_id AND end_route_id;
   END IF;
END //

DELIMITER ;

CALL GetPassengersByRouteRange(1, 25);

--          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 --

DELIMITER //

CREATE PROCEDURE GetLongDistanceRoutes()
BEGIN
   SELECT *
   FROM routes
   WHERE Distance_miles > 2000;
END //
```

```
DELIMITER ;
CALL GetLongDistanceRoutes();

--              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 GroupDistanceCategories()
BEGIN
   SELECT
      *,
      CASE
         WHEN Distance_miles >= 0 AND Distance_miles <= 2000 THEN 'SDT'
         WHEN Distance_miles > 2000 AND Distance_miles <= 6500 THEN 'IDT'
         WHEN Distance_miles > 6500 THEN 'LDT'
         ELSE 'Unknown'
      END AS Distance_Category
   FROM routes;
END //

DELIMITER ;
CALL GroupDistanceCategories();

-- 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.

DELIMITER //
CREATE FUNCTION GetComplimentaryServices(class_id VARCHAR(20)) RETURNS VARCHAR(3)
DETERMINISTIC
NO SQL
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 ExtractTicketDetailsWithComplimentaryServices()
BEGIN
    SELECT p_date, customer_id, class_id, GetComplimentaryServices(class_id) AS ComplimentaryServices
    FROM ticket_details;
END //
DELIMITER ;

CALL ExtractTicketDetailsWithComplimentaryServices();

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

DELIMITER //

CREATE PROCEDURE GetCustomerByLastName()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE last_name VARCHAR(50);
    DECLARE cur CURSOR FOR
        SELECT last_name FROM customer WHERE last_name LIKE '%Scott';
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur;
    FETCH cur INTO last_name;

    IF NOT done THEN
        SELECT * FROM customer WHERE last_name = last_name LIMIT 1;
    END IF;

    CLOSE cur;
END //

DELIMITER ;
```

```
CALL GetCustomerByLastName();
```