```
-- Use the flight database for all operations
USE flight;

-- View the contents of the 'routes' table
SELECT * FROM routes;

-- Q2: Add constraints to the 'routes' table
ALTER TABLE routes
ADD CONSTRAINT flight_num CHECK(flight_num > 0);
-- Flight number must be positive

ALTER TABLE routes
ADD CONSTRAINT unq_route_id UNIQUE (route_id, flight_num);
-- Each route-flight combo must be unique

ALTER TABLE routes
ADD CONSTRAINT dist_miles CHECK(distance_miles > 0);
-- Distance must be greater than 0


-- Q3: Display all passengers who have travelled on route IDs between 1 and 25
SELECT * FROM passengers_on_flights
WHERE route_id BETWEEN 1 AND 25;



-- Q4: Get total number of business class passengers and total revenue
SELECT COUNT(class_id), SUM(price_per_ticket)
FROM ticket_details
WHERE class_id = 'Bussiness' -- Note: Should be 'Business'
GROUP BY class_id;


-- Q5: Display full name of all customers
SELECT CONCAT(first_name, "  ", last_name) AS full_name
FROM customer;


-- Q6: Create a new table showing ticket booking info for each customer
CREATE TABLE customer_ticket_details AS
SELECT
   c.customer_id,
   CONCAT(c.first_name, "  ", c.last_name) AS full_name,
   t.p_date AS booking_date,
   t.aircraft_id,
   t.class_id,
   t.brand AS airline
FROM
   customer c
JOIN
   ticket_details t ON c.customer_id = t.customer_id
ORDER BY
   c.customer_id;
```

-- Q7: Create a table showing which customers flew with which brand

```
CREATE TABLE customer_booking AS
SELECT
   c.customer_id,
   c.first_name,
   c.last_name,
   p.brand
FROM
   customer c
JOIN
   ticket_details p ON c.customer_id = p.customer_id
ORDER BY
   customer_id;

-- View the customer_booking table
SELECT * FROM customer_booking;
```

-- Q8: Show customers who travelled in Economy Plus at least once

```
SELECT customer_id, COUNT(*) AS total_flights
FROM passengers_on_flights
WHERE travel_class = 'Economy Plus'
GROUP BY customer_id
HAVING COUNT(*) >= 1;
```

-- Q9: Calculate revenue and check if it exceeds 10,000

```
CREATE TABLE revenue AS
SELECT
   COUNT(no_of_tickets),
   price_per_ticket,
   brand,
   class_id,
   (COUNT(no_of_tickets) * price_per_ticket) AS revenue
FROM ticket_details
GROUP BY brand, class_id, price_per_ticket;
```

-- View revenue table and check if revenue > 10,000

```
SELECT * FROM revenue;
SELECT SUM(revenue), IF(SUM(revenue) > 10000, "YES", "NO") FROM revenue;
```

-- Q10: Create a user and grant privileges (Run only once if not already created)

```
CREATE USER 'dev_user'@'localhost' IDENTIFIED BY 'securepass';
GRANT SELECT, INSERT, UPDATE ON company_db.* TO 'dev_user'@'localhost';
FLUSH PRIVILEGES;
```

-- Q11: Find max ticket price for each class

```
SELECT class_id, MAX(price_per_ticket)
FROM ticket_details
GROUP BY class_id;
```

-- Q12: Create an index on route_id for faster querying

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

-- Retrieve data for route ID 4

```
SELECT
   customer_id,
   aircraft_id,
   flight_num,
   travel_date,
   seat_num,
   class_id
FROM
   passengers_on_flights
WHERE
   route_id = 4;
```

-- Q13: View query execution plan to check index usage

```
EXPLAIN
SELECT
   customer_id,
   aircraft_id,
   flight_num,
   travel_date,
   seat_num,
   class_id
FROM
   passengers_on_flights
WHERE
   route_id = 4;
```

-- Q14: Use ROLLUP to compute ticket revenue grouped by customer and aircraft

```
SELECT
   customer_id,
   aircraft_id,
   SUM(no_of_tickets * price_per_ticket) AS total_price
FROM ticket_details
GROUP BY ROLLUP(customer_id, aircraft_id);
```

-- Q15: Create a view of business class customers and their flight details

```
SELECT
   p.customer_id,
   c.first_name,
   c.last_name,
   p.aircraft_id,
   p.travel_date,
   p.flight_num
FROM customer c
INNER JOIN passengers_on_flights p ON c.customer_id = p.customer_id
WHERE p.class_id = "Bussiness"; -- Note: Should be 'Business'
```

-- Q16: Stored procedure to get passengers between route range

```
DELIMITER $$
CREATE PROCEDURE GetPassengersByRouteRange(
   IN start_route INT,
   IN end_route INT
)
BEGIN
   DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
   BEGIN
      SELECT 'Error: Table passengers_on_flights does not exist.' AS error_message;
   END;

   SELECT
      customer_id,
      route_id,
      aircraft_id,
      flight_num,
      seat_num,
      class_id,
      travel_date
   FROM
      passengers_on_flights
   WHERE
      route_id BETWEEN start_route AND end_route;
END $$
DELIMITER ;
CALL GetPassengersByRouteRange(2, 6);
```

-- Q17: Stored procedure to return flights with distance > 2000 miles

```
DELIMITER $$
CREATE PROCEDURE travel_dist(IN dist INT)
BEGIN
   SELECT * FROM routes r
   WHERE r.distance_miles > dist;
END $$
DELIMITER ;

CALL travel_dist(2000);
```

-- Q18: Function to classify distance into SDT, IDT, or LDT

```
DELIMITER $$
CREATE FUNCTION dist_type(dist INT)
RETURNS VARCHAR(150)
DETERMINISTIC
BEGIN
   DECLARE dist_type VARCHAR(150);
   IF dist > 0 AND dist <= 2000 THEN
      SET dist_type = "Short Distance Travel(SDT)";
   ELSEIF dist > 2000 AND dist <= 6500 THEN
      SET dist_type = "Intermediate Distance Travel(IDT)";
   ELSE
      SET dist_type = "Long Distance Travel(LDT)";
   END IF;
   RETURN dist_type;
END $$
DELIMITER ;
```

-- Use the function to classify routes

```
SELECT
   aircraft_id,
   origin_airport,
   destination_airport,
   flight_num,
   route_id,
   distance_miles,
   dist_type(distance_miles) AS type_of_distance
FROM routes;
```

-- Q19: Function to define if complimentary services are available

```
DELIMITER $$
CREATE FUNCTION comp_serve(class VARCHAR(100))
RETURNS VARCHAR(300)
DETERMINISTIC
BEGIN
   DECLARE comp_service VARCHAR(150);
   IF class = "Bussiness" OR class = "Economy Plus" THEN
      SET comp_service = "available";
   ELSE
      SET comp_service = "not available";
   END IF;
   RETURN comp_service;
END $$
DELIMITER ;
```

-- Use the function

```
SELECT
   aircraft_id,
   class_id,
   brand,
   comp_serve(class_id) AS complimentary_services
FROM ticket_details;
```

-- Q20: Cursor to fetch first customer whose last name ends in "Scott"

```
DELIMITER $$
CREATE PROCEDURE GetFirstScottCustomer()
BEGIN
   DECLARE done INT DEFAULT FALSE;
   DECLARE v_id INT;
   DECLARE v_first_name VARCHAR(100);
   DECLARE v_last_name VARCHAR(100);

   DECLARE scott_cursor CURSOR FOR
      SELECT customer_id, first_name, last_name
      FROM customer
      WHERE last_name LIKE '%Scott';

   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

   OPEN scott_cursor;
   FETCH scott_cursor INTO v_id, v_first_name, v_last_name;
   IF NOT done THEN
      SELECT v_id AS customer_id, v_first_name AS first_name, v_last_name AS last_name;
   ELSE
      SELECT 'No customer found with last name ending in Scott' AS message;
   END IF;
   CLOSE scott_cursor;
END $$
DELIMITER ;
```