**Air Cargo Analysis**

1. **Create an ER diagram for the given airlines database.**

Diagram, table

Description automatically generated

1. **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.**

**SQL code:**

CREATE TABLE route\_details (

route\_id INT NOT NULL UNIQUE,

flight\_num INT NOT NULL CHECK (flight\_num > 0),

origin\_airport CHAR (3) NOT NULL,

destination\_airport CHAR (3) NOT NULL,

aircraft\_id INT NOT NULL,

distance\_miles SMALLINT NOT NULL CHECK (distance\_miles > 0),

PRIMARY KEY (route\_id)

);

**Output:**



1. **Write a query to display all the passengers who have travelled in routes 01 to 25 from the passengers\_on\_flights table.**

**SQL code:**

SELECT \* FROM passengers\_on\_flights WHERE route\_id between 01 AND 25;

**Output:**

Table

Description automatically generated with medium confidence

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

**SQL code:**

SELECT COUNT (class\_id) Bussiness\_Class, SUM(price\_per\_ticket) revenue\_in\_bussiness FROM ticket\_details WHERE class\_id = 'Bussiness';

**Output:**



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

**SQL code:**

SELECT CONCAT(first\_name,' ',last\_name) FULL\_NAME FROM customer;

**Output:**

A picture containing application

Description automatically generated

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

**SQL code:**

SELECT customer.customer\_id,customer.first\_name,customer.last\_name,ticket\_details.p\_date

FROM customer

INNER JOIN ticket\_details

ON customer.customer\_id = ticket\_details.customer\_id;

**Output:**

Table

Description automatically generated

1. **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.**

**SQL code:**

SELECT first\_name, last\_name FROM customer

WHERE

customer\_id IN (SELECT customer\_id FROM ticket\_details WHERE brand = 'Emirates');

**Output:**

Table

Description automatically generated

1. **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.**

**SQL code:**

SELECT \* FROM passengers\_on\_flights GROUP BY customer\_id HAVING class\_id = 'Economy Plus';

**Output:**

A picture containing table

Description automatically generated

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

**SQL code:**

SELECT IF(SUM(Price\_per\_ticket)>10000,'Revenue is greater than 10000','Revenue is less than 10000') Check\_if\_revenu\_greater\_than\_10000 from ticket\_details

**Output:**

Graphical user interface, text

Description automatically generated

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

**SQL code: Create a new user**

CREATE USER junior@localhost

IDENTIFIED BY 'junior123';

**SQL code: Grant user access**

GRANT EXECUTE ON securitydb.\*

TO junior@localhost;

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

**SQL code:**

SELECT \*, MAX(Price\_per\_ticket) OVER ( PARTITION BY class\_id ) max\_price

FROM ticket\_details;

**Output:**

Table

Description automatically generated with medium confidence

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.**

**SQL code: Create index**

CREATE INDEX route\_id ON passengers\_on\_flights (route\_id);

SELECT customer\_id,depart,arrival,aircraft\_id FROM passengers\_on\_flights WHERE route\_id = '04';

**Output:**

Graphical user interface, application

Description automatically generated

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

**SQL code:**

SELECT \* FROM passengers\_on\_flights WHERE route\_id = '04';

**Output:**

Diagram

Description automatically generated

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

**SQL code:**

SELECT customer\_id, aircraft\_id, SUM(Price\_per\_ticket) as Total\_Price FROM ticket\_details GROUP BY customer\_id, aircraft\_id WITH ROLLUP;

**Output:**

Table

Description automatically generated

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

**SQL code:**

CREATE VIEW BussinessClassView AS SELECT customer\_id, class\_id, brand from cep\_ticket\_details where class\_id = "Bussiness";

select \* from BussinessClassView;

**Output:**

Table

Description automatically generated

1. **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.**

**SQL code:**

DROP PROCEDURE IF EXISTS getPassengersBetweenRoutes;

DELIMITER &&

CREATE PROCEDURE getPassengersBetweenRoutes(

IN route\_a INT,

IN route\_b INT)

BEGIN

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

GET DIAGNOSTICS CONDITION 1

@sqlstate = RETURNED\_SQLSTATE,

@errno = MYSQL\_ERRNO,

@text = MESSAGE\_TEXT;

SET @full\_error = CONCAT("SQLEXCEPTION Handler - ERROR ", @errno, " (", @sqlstate, "): ", @text);

SELECT @full\_error AS msg;

END;

-- MySQL Query

SELECT \* FROM cep\_passengers\_on\_flight WHERE route\_id BETWEEN route\_a AND route\_b;

END &&

CALL getPassengersBetweenRoutes(1,25);

**Output:**

Table

Description automatically generated with medium confidence

1. **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.**

**SQL code:**

DELIMITER &&

CREATE PROCEDURE Get\_Customers\_With\_2K()

BEGIN

SELECT \* FROM cep\_routes where distance\_miles>2000;

END &&

CALL Get\_Customers\_With\_2K();

**Output:**

Table

Description automatically generated

1. **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.**

**SQL code:**

DROP PROCEDURE IF EXISTS getTravelType;

DELIMITER $$

CREATE PROCEDURE getTravelType()

BEGIN

DECLARE dist INT DEFAULT 0;

DECLARE done BIT DEFAULT FALSE;

DECLARE route\_cur CURSOR FOR SELECT distance\_miles FROM cep\_routes;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN route\_cur;

CREATE TEMPORARY TABLE IF NOT EXISTS cep\_routes\_2 (

travel\_type VARCHAR(3)

) AS ( SELECT \* FROM cep\_routes);

while\_label: WHILE done = FALSE DO

FETCH route\_cur INTO dist;

IF dist >=0 AND dist <= 2000 THEN

UPDATE cep\_routes\_2

SET travel\_type = "SDT"

WHERE distance\_miles = dist;

ELSEIF dist >2000 AND dist <=6500 THEN

UPDATE cep\_routes\_2

SET travel\_type = "IDT"

WHERE distance\_miles = dist;

ELSEIF dist >6500 THEN

UPDATE cep\_routes\_2

SET travel\_type = "LDT"

WHERE distance\_miles = dist;

ELSE

ITERATE while\_label;

END IF;

END WHILE while\_label;

SELECT \* FROM cep\_routes\_2 ORDER BY distance\_miles;

END$$

CALL getTravelType();

**Output:**

Table

Description automatically generated

1. **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***

**SQL code: Create stored function**

DELIMITER $$

CREATE FUNCTION complimentary\_servicess(class\_id VARCHAR(225))

RETURNS VARCHAR(2255) DETERMINISTIC

BEGIN DECLARE complimentary\_servicess VARCHAR(2255);

IF class\_id = 'Bussiness' THEN SET complimentary\_servicess = 'Yes';

ELSEIF class\_id = 'Economy Plus' THEN SET complimentary\_servicess = 'Yes';

ELSE SET complimentary\_servicess = 'No' ;

END IF; RETURN (complimentary\_servicess); END$$

DELIMITER $$;

**SQL code: Create stored procedure to call stored function**

DELIMITER $$

CREATE PROCEDURE Extra\_complimentary\_services() BEGIN

SELECT p\_date,customer\_id,class\_id,complimentary\_servicess(class\_id) as Extra\_complimentary\_services FROM ticket\_details ORDER BY class\_id;

END$$ DELIMITER ; call Extra\_complimentary\_services();

**Output:**

Table

Description automatically generated

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

**SQL code:**

DELIMITER $$

CREATE PROCEDURE first\_last\_name\_is\_scott()

BEGIN DECLARE a VARCHAR(255);

DECLARE b VARCHAR(255);

DECLARE cursor\_1 CURSOR FOR SELECT first\_name,last\_name FROM customer

WHERE last\_name = 'Scott';

OPEN cursor\_1;

REPEAT FETCH cursor\_1 INTO a,b;

UNTIL b = 0 END REPEAT;

SELECT a as first\_name, b as last\_name;

CLOSE cursor\_1;

END;

$$ DELIMITER ;

call first\_last\_name\_is\_scott();

**Output:**

