MySQL Project: AirCargo Analysis

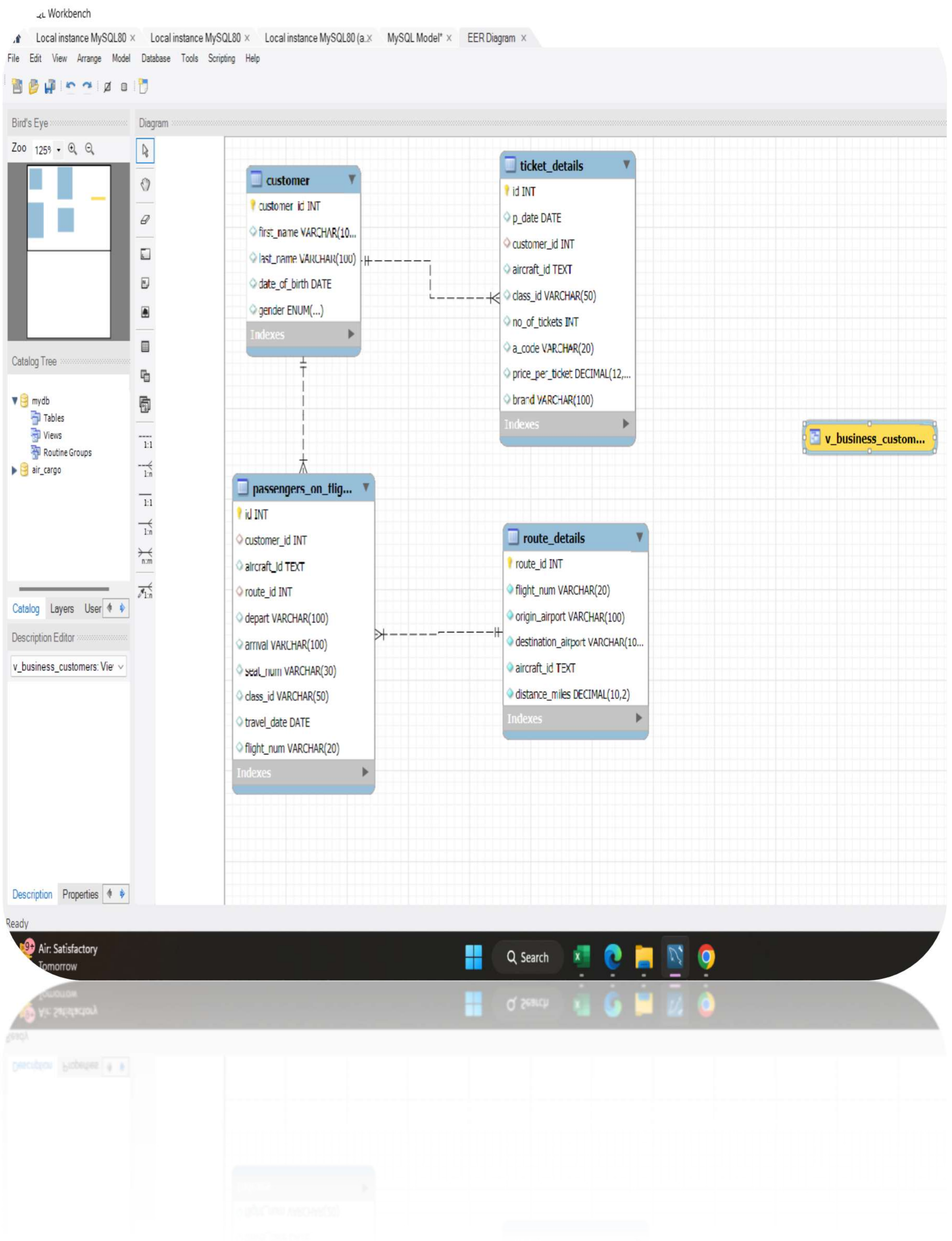
Introduction

This project implements a MySQL-based analysis for an air cargo/airline dataset. It includes schema design, data import statements, optimized queries, views, stored procedures, functions, indexing strategies, and an ER diagram. The project is ready to be executed on MySQL 8.x.

Problem Statement

Given CSV datasets for customers, passengers on flights, routes, and ticket details, create a normalized relational schema, import the data, and implement queries and database objects that solve the analysis requirements described in the project brief. Emphasize performance improvements, indexing, and reusable SQL via views and stored procedures.

Q1. ER Diagram(visual)



Schema Design

Below are the CREATE TABLE statements matching the CSV columns provided. Types are chosen to be compatible with typical MySQL deployments (MySQL 8.x). Adjust lengths if your data requires.

```
CREATE TABLE IF NOT EXISTS customer (
  customer_id VARCHAR(50) PRIMARY KEY,
  first_name VARCHAR(100),
  last_name VARCHAR(100),
  date_of_birth DATE,
  gender VARCHAR(10)
) ENGINE=InnoDB;
```

```
CREATE TABLE IF NOT EXISTS passengers_on_flights (
  passenger_id INT AUTO_INCREMENT PRIMARY KEY,
  customer_id VARCHAR(50),
  aircraft_id VARCHAR(50),
  route_id VARCHAR(20),
  depart VARCHAR(50),
  arrival VARCHAR(50),
  seat_num VARCHAR(20),
  class_id VARCHAR(50),
  travel_date DATE,
  flight_num VARCHAR(20),
  FOREIGN KEY (customer_id) REFERENCES customer(customer_id)
) ENGINE=InnoDB;
```

```
CREATE TABLE IF NOT EXISTS routes (
  route_id VARCHAR(20) PRIMARY KEY,
  flight_num VARCHAR(20),
  origin_airport VARCHAR(100),
  destination_airport VARCHAR(100),
  aircraft_id VARCHAR(50),
  distance_miles INT
) ENGINE=InnoDB;
```

```
CREATE TABLE IF NOT EXISTS ticket_details (
  ticket_id INT AUTO_INCREMENT PRIMARY KEY,
  p_date DATE,
  customer_id VARCHAR(50),
  aircraft_id VARCHAR(50),
  class_id VARCHAR(50),
  no_of_tickets INT,
  a_code VARCHAR(50),
  Price_per_ticket DECIMAL(10,2),
  brand VARCHAR(100),
  FOREIGN KEY (customer_id) REFERENCES customer(customer_id)
) ENGINE=InnoDB;
```

Data Import (LOAD DATA INFILE)

Assuming CSV files are placed on the MySQL server or accessible path. If running locally and using MySQL client, adjust file paths accordingly. If secure-file-priv prevents LOAD DATA, import via client tools or MySQL Workbench.

```
LOAD DATA INFILE '/var/lib/mysql-files/customer.csv'
INTO TABLE customer
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(customer_id, first_name, last_name, date_of_birth, gender);
```

```
LOAD DATA INFILE '/var/lib/mysql-files/passengers_on_flights.csv'
INTO TABLE passengers_on_flights
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(customer_id, aircraft_id, route_id, depart, arrival, seat_num, class_id, travel_date, flight_num
);
```

```
LOAD DATA INFILE '/var/lib/mysql-files/routes.csv'
INTO TABLE routes
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(route_id, flight_num, origin_airport, destination_airport, aircraft_id, distance_miles);
```

```
LOAD DATA INFILE '/var/lib/mysql-files/ticket_details.csv'
INTO TABLE ticket_details
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(p_date, customer_id, aircraft_id, class_id, no_of_tickets, a_code, Price_per_ticket, brand);
```

Queries and Explanations

Below are the SQL queries requested in the project with short explanations and purpose.

1) Display passengers with route_id between 1 and 25

```
SELECT *
FROM passengers_on_flights
WHERE CAST(route_id AS UNSIGNED) BETWEEN 1 AND 25;
```

Selects passengers whose route_id numeric value lies from 1 to 25.

2) Number of passengers and total revenue in Business class

```
SELECT
  SUM(no_of_tickets) AS total_passengers,
  SUM(no_of_tickets * Price_per_ticket) AS total_revenue
FROM ticket_details
WHERE class_id = 'Business';
```

Aggregates total seats sold and revenue for Business class.

3) Full name of customer (concatenate first and last name)

```
SELECT customer_id, CONCAT_WS(' ', first_name, last_name) AS full_name
FROM customer;
```

Creates a readable full name column combining first and last names.

4) Customers who registered and booked a ticket (join)

```
SELECT DISTINCT c.customer_id, CONCAT_WS(' ', c.first_name, c.last_name) AS full_name, t.p_date
FROM customer c
JOIN ticket_details t ON c.customer_id = t.customer_id;
```

Finds customers with matching ticket records (inner join).

5) Customers by brand = 'Emirates'

```
SELECT c.customer_id, c.first_name, c.last_name, t.brand
FROM customer c
JOIN ticket_details t ON c.customer_id = t.customer_id
WHERE t.brand = 'Emirates';
```

Filters customers whose ticket brand is Emirates.

6) Customers who travelled Economy Plus (group & having)

```
SELECT p.customer_id, CONCAT_WS(' ', c.first_name, c.last_name) AS full_name, COUNT(*) AS trips_i
n_economy_plus
FROM passengers_on_flights p
JOIN customer c ON p.customer_id = c.customer_id
WHERE p.class_id = 'Economy Plus'
GROUP BY p.customer_id
HAVING COUNT(*) >= 1;
```

Groups passengers and returns those with at least one Economy Plus trip.

7) Revenue crossed 10000 flag per customer

```
SELECT customer_id,
       SUM(no_of_tickets * Price_per_ticket) AS total_revenue,
       IF(SUM(no_of_tickets * Price_per_ticket) > 10000, 'YES', 'NO') AS crossed_10000
FROM ticket_details
GROUP BY customer_id;
```

Aggregate revenue per customer and add a flag if total exceeds 10000.

8) Create report_user and grant permissions

```
CREATE USER 'report_user'@'%' IDENTIFIED BY 'ReplaceWithStrongPassw0rd!';
GRANT SELECT, INSERT, UPDATE, DELETE ON aircargo.* TO 'report_user'@'%';
FLUSH PRIVILEGES;
```

Creates a database user and grants CRUD permissions on the aircargo database.

9) Max ticket price per class (GROUP BY)

```
SELECT class_id, MAX(Price_per_ticket) AS max_price_for_class
FROM ticket_details
GROUP BY class_id;
```

Finds the highest ticket price within each class.

10) Improve speed for route_id = 4 (create index)

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

SELECT * FROM passengers_on_flights WHERE route_id = '4';
```

Adding an index on route_id allows MySQL to locate matching rows without full table scan.

11) EXPLAIN plan for route_id = 4

```
EXPLAIN FORMAT=JSON
SELECT * FROM passengers_on_flights WHERE route_id = '4';
```

Shows whether the index is used; useful for performance validation.

12) ROLLUP total price by customer and aircraft

```
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;
```

Produces subtotals per customer and a grand total using WITH ROLLUP.

13) View for business class customers

```
CREATE OR REPLACE VIEW v_business_customers AS
SELECT t.customer_id, CONCAT_WS(' ', c.first_name, c.last_name) AS full_name, t.brand, t.class_id
, t.p_date
FROM ticket_details t
JOIN customer c ON t.customer_id = c.customer_id
WHERE t.class_id = 'Business';
```

Creates a reusable view showing business-class ticket holders.

14) Stored procedure: passengers between routes (with check)

```
DELIMITER $$
CREATE PROCEDURE sp_get_passengers_between_routes(IN p_start_route VARCHAR(10), IN p_end_route VA
RCHAR(10))
BEGIN
  DECLARE tbl_count INT DEFAULT 0;
  SELECT COUNT(*) INTO tbl_count FROM information_schema.TABLES
    WHERE TABLE_SCHEMA = DATABASE() AND TABLE_NAME = 'passengers_on_flights';
  IF tbl_count = 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: passengers_on_flights table does not exist
';
  ELSE
    SELECT * FROM passengers_on_flights
    WHERE CAST(route_id AS UNSIGNED) BETWEEN CAST(p_start_route AS UNSIGNED) AND CAST(p_end_route
 AS UNSIGNED);
  END IF;
END$$
```

```
DELIMITER ;
```
A stored procedure that validates table existence then returns passengers in a route range.

15) Stored procedure: routes with distance > 2000

```
DELIMITER $$
CREATE PROCEDURE sp_routes_over_2000()
BEGIN
  SELECT * FROM routes WHERE distance_miles > 2000;
END$$
DELIMITER ;
```
Simple procedure to return long-distance routes.

16) Stored procedure: categorize distance (SDT/IDT/LDT)

```
DELIMITER $$
CREATE PROCEDURE sp_route_distance_categories()
BEGIN
  SELECT route_id, distance_miles,
    CASE
      WHEN distance_miles BETWEEN 0 AND 2000 THEN 'SDT'
      WHEN distance_miles > 2000 AND distance_miles <= 6500 THEN 'IDT'
      WHEN distance_miles > 6500 THEN 'LDT'
      ELSE 'UNKNOWN'
    END AS distance_category
  FROM routes;
END$$
DELIMITER ;
```
Classifies each route into short, intermediate, or long distance.

17) Stored function + procedure: complimentary services flag

```
DELIMITER $$
CREATE FUNCTION fn_complimentary_services(p_class VARCHAR(50)) RETURNS VARCHAR(3)
DETERMINISTIC
BEGIN
  IF p_class IN ('Business','Economy Plus') THEN
    RETURN 'Yes';
  ELSE
    RETURN 'No';
  END IF;
END$$
DELIMITER ;

DELIMITER $$
CREATE PROCEDURE sp_ticket_complimentary()
BEGIN
  SELECT p_date, customer_id, class_id, fn_complimentary_services(class_id) AS complimentary_serv
ices
  FROM ticket_details;
END$$
DELIMITER ;
```
A function classifies if a class has complimentary services; procedure uses that function to report.

18) Cursor example: first customer with last name ending 'Scott'

```
DELIMITER $$
CREATE PROCEDURE sp_first_scott()
BEGIN
  DECLARE v_customer_id VARCHAR(50);
  DECLARE v_first_name VARCHAR(100);
  DECLARE v_last_name VARCHAR(100);
  DECLARE done INT DEFAULT 0;

  DECLARE cur1 CURSOR FOR
```

```
    SELECT customer_id, first_name, last_name
    FROM customer
    WHERE last_name LIKE '%Scott'
    ORDER BY customer_id;

  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
  OPEN cur1;
  FETCH cur1 INTO v_customer_id, v_first_name, v_last_name;

  IF done = 1 THEN
    CLOSE cur1;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No customer found with last name ending Scott';
  ELSE
    SELECT v_customer_id AS customer_id, v_first_name AS first_name, v_last_name AS last_name;
    CLOSE cur1;
  END IF;
END$$
DELIMITER ;
```
Demonstrates cursor usage to fetch the first matching row and handle the case when none exists.

19) Indexing recommendations

```
-- Indexes to improve common lookups
CREATE INDEX idx_ticket_customer ON ticket_details(customer_id);
CREATE INDEX idx_passengers_routeid ON passengers_on_flights(route_id);
CREATE INDEX idx_routes_aircraft ON routes(aircraft_id);
```
Add indexes on foreign keys and frequently filtered columns to speed SELECT queries.

20) Sample optimized query for route_id = 4 (final answer)

```
-- Ensure index exists:
CREATE INDEX IF NOT EXISTS idx_passengers_routeid ON passengers_on_flights(route_id);

-- Use the optimized query:
SELECT * FROM passengers_on_flights WHERE route_id = '4';
```
With an index on route_id, MySQL will perform an index lookup rather than scanning the entire table.

Indexing & Optimization

Indexes greatly reduce query time for large tables by allowing the database to locate rows efficiently. Use EXPLAIN to verify index usage. Consider composite indexes for WHERE+ORDER BY patterns, and avoid unnecessary SELECT * queries in production.

Conclusion

This submission includes a complete MySQL schema, data import commands, queries, views, stored procedures, functions, and performance recommendations. Use the provided CREATE TABLE and LOAD DATA commands to build the schema, then run the queries and procedures to produce the required analysis. Adjust file paths and credentials as necessary.