

# Air Cargo Analysis

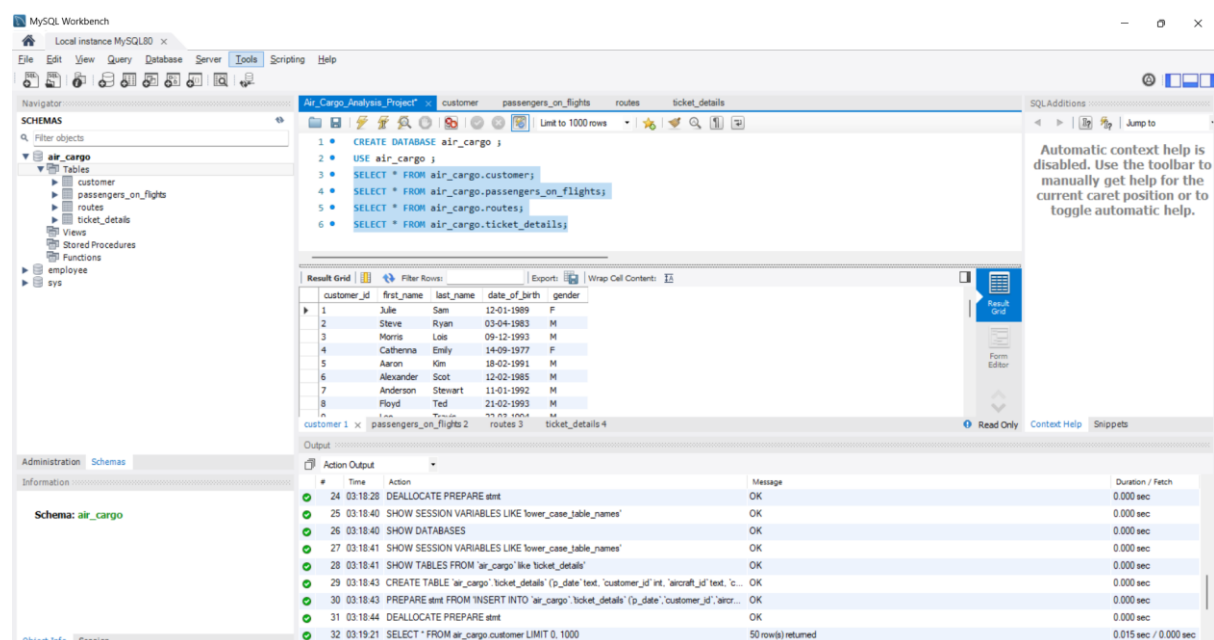
## Problem Statement Scenario:

Air Cargo is an aviation company that provides air transportation services for passengers and freight. Air Cargo uses its aircraft to provide different services with the help of partnerships or alliances with other airlines. The company wants to prepare reports on regular passengers, busiest routes, ticket sales details, and other scenarios to improve the ease of travel and booking for customers.

## Project Objective:

We, as a DBA expert, need to focus on identifying the regular customers to provide offers, analyse the busiest route which helps to increase the number of aircraft required and prepare an analysis to determine the ticket sales details. This will ensure that the company improves its operability and becomes more customer-centric and a favourable choice for air travel.

- **Importing Datasets:**
  - customer



## **Conclusion:**

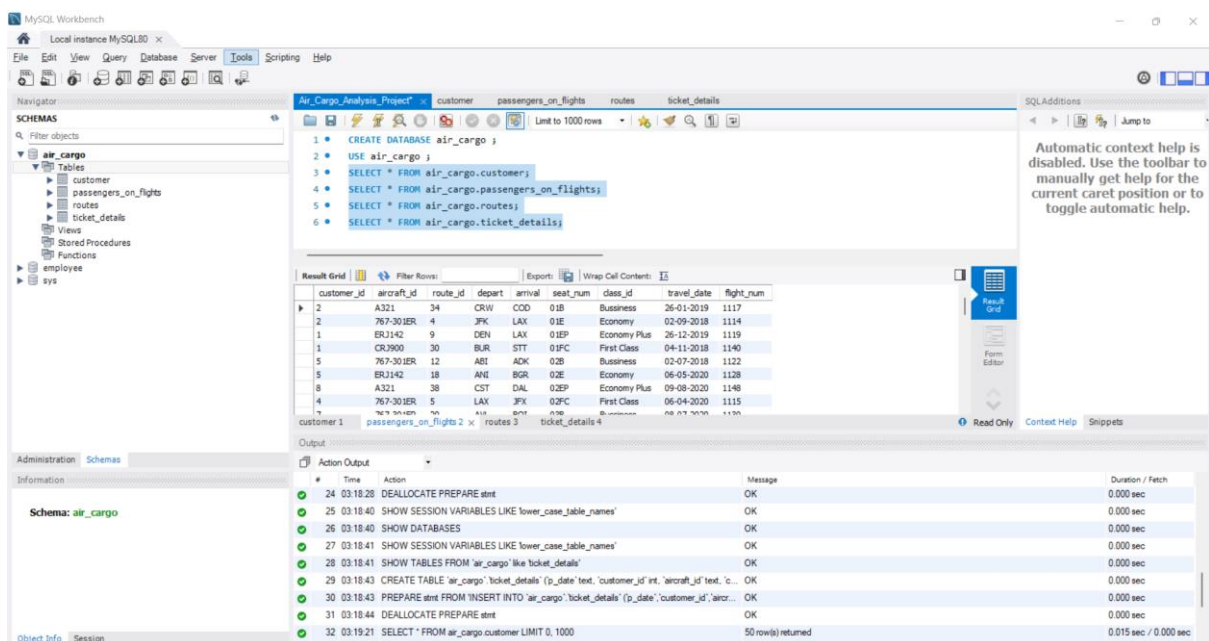
In this project, the *Air Cargo* database was created and analysed through a series of SQL operations designed to enhance the company's route and customer management. The tasks included:

- Designing an ER diagram and defining primary and foreign key relationships.

- Writing queries to extract passenger, route, and ticket details.
- Applying conditional logic using IF, CASE, and aggregation functions.
- Creating views, stored procedures, and stored functions for automated and reusable operations.
- Using indexing and the EXPLAIN plan to improve query performance.
- Implementing advanced SQL concepts such as ROLLUP, WINDOW FUNCTIONS, and runtime parameterised procedures.

Through these implementations, the project demonstrates practical applications of SQL in database design, performance optimisation, and analytical reporting — enabling more efficient decision-making for route planning, customer offers, and revenue tracking in the aviation domain.

- passengers\_on\_flights



The screenshot displays the MySQL Workbench interface for a project named 'Air\_Cargo\_Analysis\_Project'. The left sidebar shows the 'SCHEMAS' pane with a tree view of the database structure, including tables like 'customer', 'passengers\_on\_flights', 'routes', and 'ticket\_details'. The central SQL editor contains the following queries:

```

1 CREATE DATABASE air_cargo ;
2 USE air_cargo ;
3 SELECT * FROM air_cargo.customer;
4 SELECT * FROM air_cargo.passengers_on_flights;
5 SELECT * FROM air_cargo.routes;
6 SELECT * FROM air_cargo.ticket_details;

```

The 'Result Grid' shows the output of the queries, with columns for 'customer\_id', 'aircraft\_id', 'route\_id', 'depart', 'arrival', 'seat\_num', 'class\_id', 'travel\_date', and 'flight\_num'. The output window at the bottom shows the execution of the queries, including the creation of the database and tables, and the execution of the SELECT query.

#	Time	Action	Message	Duration / Fetch
24	03:18:28	DEALLOCATE PREPARE stmt	OK	0.000 sec
25	03:18:40	SHOW SESSION VARIABLES LIKE 'lower_case_table_names'	OK	0.000 sec
26	03:18:40	SHOW DATABASES	OK	0.000 sec
27	03:18:41	SHOW SESSION VARIABLES LIKE 'lower_case_table_names'	OK	0.000 sec
28	03:18:41	SHOW TABLES FROM 'air_cargo' like 'ticket_details'	OK	0.000 sec
29	03:18:43	CREATE TABLE 'air_cargo'.'ticket_details' ('p_date' text, 'customer_id' int, 'aircraft_id' text, 'c...	OK	0.000 sec
30	03:18:43	PREPARE stmt FROM 'INSERT INTO 'air_cargo'.'ticket_details' ('p_date','customer_id','airc...	OK	0.000 sec
31	03:18:44	DEALLOCATE PREPARE stmt	OK	0.000 sec
32	03:19:21	SELECT * FROM air_cargo.customer LIMIT 0, 1000	50 row(s) returned	0.015 sec / 0.000 sec

- routes

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHMAS

Filter objects

air\_cargo

Tables

customer

passengers\_on\_flights

routes

ticket\_details

Views

Stored Procedures

Functions

employee

sys

air\_cargo\_Analysis\_Project x

customer passengers\_on\_flights routes ticket\_details

Limit to 1000 rows

1 \* CREATE DATABASE air\_cargo ;

2 \* USE air\_cargo ;

3 \* SELECT \* FROM air\_cargo.customer;

4 \* SELECT \* FROM air\_cargo.passengers\_on\_flights;

5 \* SELECT \* FROM air\_cargo.routes;

6 \* SELECT \* FROM air\_cargo.ticket\_details;

Result Grid

route_id	flight_num	origin_airport	destination_airport	aircraft_id	distance_miles
1	1111	EWK	HNL	767-301ER	4962
2	1112	HNL	EWK	767-301ER	4962
3	1113	EWK	LHR	A321	3466
4	1114	JFK	LAX	767-301ER	2475
5	1115	LAX	JFK	767-301ER	2475
6	1116	HNL	LAX	767-301ER	2556
7	1117	LAX	ORD	A321	1745
8	1118	ORD	EWK	A321	719

customer 1 passengers\_on\_flights 2 routes 3 x ticket\_details 4

Output

Action Output

#	Time	Action	Message	Duration / Fetch
24	03:18:28	DEALLOCATE PREPARE stmt	OK	0.000 sec
25	03:18:40	SHOW SESSION VARIABLES LIKE 'lower_case_table_names'	OK	0.000 sec
26	03:18:40	SHOW DATABASES	OK	0.000 sec
27	03:18:41	SHOW SESSION VARIABLES LIKE 'lower_case_table_names'	OK	0.000 sec
28	03:18:41	SHOW TABLES FROM 'air_cargo' like 'ticket_details'	OK	0.000 sec
29	03:18:43	CREATE TABLE 'air_cargo'.'ticket_details' (p_date text, customer_id int, aircraft_id text, c...	OK	0.000 sec
30	03:18:43	PREPARE stmt FROM INSERT INTO 'air_cargo'.'ticket_details' (p_date, customer_id, aircor...	OK	0.000 sec
31	03:18:44	DEALLOCATE PREPARE stmt	OK	0.000 sec
32	03:19:21	SELECT * FROM air_cargo.customer LIMIT 0, 1000	50 row(s) returned	0.015 sec / 0.000 sec

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

- ticket\_details

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHMAS

Filter objects

air\_cargo

Tables

customer

passengers\_on\_flights

routes

ticket\_details

Views

Stored Procedures

Functions

employee

sys

air\_cargo\_Analysis\_Project x

customer passengers\_on\_flights routes ticket\_details

Limit to 1000 rows

1 \* CREATE DATABASE air\_cargo ;

2 \* USE air\_cargo ;

3 \* SELECT \* FROM air\_cargo.customer;

4 \* SELECT \* FROM air\_cargo.passengers\_on\_flights;

5 \* SELECT \* FROM air\_cargo.routes;

6 \* SELECT \* FROM air\_cargo.ticket\_details;

Result Grid

p_date	customer_id	aircraft_id	class_id	no_of_tickets	a_code	price_per_ticket	brand
26-12-2018	27	767-301ER	Economy	1	DAL	130	Emirates
02-02-2020	22	BRJ142	Economy Plus	1	AGB	220	Jet Airways
03-03-2020	21	CRJ900	Business	1	BOH	490	British Airways
04-04-2020	4	767-301ER	First Class	1	AGB	390	Emirates
05-05-2020	5	BRJ142	Economy	1	CTM	120	Jet Airways
07-07-2020	7	767-301ER	Business	1	BFS	430	Emirates
08-08-2020	8	A321	Economy Plus	1	DAL	275	Qatar Airways
09-09-2020	9	767-301ER	First Class	1	BOH	380	Emirates
10-10-2020	10	A321	Economy	1	MPC	175	Emirates

customer 1 passengers\_on\_flights 2 routes 3 x ticket\_details 4 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
24	03:18:28	DEALLOCATE PREPARE stmt	OK	0.000 sec
25	03:18:40	SHOW SESSION VARIABLES LIKE 'lower_case_table_names'	OK	0.000 sec
26	03:18:40	SHOW DATABASES	OK	0.000 sec
27	03:18:41	SHOW SESSION VARIABLES LIKE 'lower_case_table_names'	OK	0.000 sec
28	03:18:41	SHOW TABLES FROM 'air_cargo' like 'ticket_details'	OK	0.000 sec
29	03:18:43	CREATE TABLE 'air_cargo'.'ticket_details' (p_date text, customer_id int, aircraft_id text, c...	OK	0.000 sec
30	03:18:43	PREPARE stmt FROM INSERT INTO 'air_cargo'.'ticket_details' (p_date, customer_id, aircor...	OK	0.000 sec
31	03:18:44	DEALLOCATE PREPARE stmt	OK	0.000 sec
32	03:19:21	SELECT * FROM air_cargo.customer LIMIT 0, 1000	50 row(s) returned	0.015 sec / 0.000 sec

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

## Following operations should be performed:

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

Changed the data types of columns and defined primary keys during the ER diagram creation phase (these can also be modified later directly in SQL if needed) :

customer - Table									
Table Name: customer		Schema: air_cargo							
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
customer_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
first_name	VARCHAR(50)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
last_name	VARCHAR(50)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
date_of_birth	DATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

routes - Table									
Table Name: routes		Schema: air_cargo							
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
route_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
flight_num	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
origin_airport	VARCHAR(20)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
destination_airport	DECIMAL(20)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

passengers_on_flights - Table									
Table Name: passengers_on_flights		Schema: air_cargo							
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
travel_date	DATE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
flight_num	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
passengers_on_flightscol	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

ticket_details - Table									
Table Name: ticket_details		Schema: air_cargo							
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
p_date	DATE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
customer_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
aircraft_id	VARCHAR(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
class_id	VARCHAR(20)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

## Defined Foreign keys:

passengers\_on\_flights - Table

Table Name: passengers\_on\_flights

Schema: air\_cargo

Foreign Key Name	Referenced Table
fk_passenger_customer	'air_cargo'.customer
fk_passenger_route	'air_cargo'.routes

Column	Referenced Column
<input type="checkbox"/> customer_id	
<input type="checkbox"/> aircraft_id	
<input checked="" type="checkbox"/> route_id	route_id
<input type="checkbox"/> depart	
<input type="checkbox"/> arrival	
<input type="checkbox"/> seat_num	
<input type="checkbox"/> class_id	
<input type="checkbox"/> travel_date	
<input type="checkbox"/> flight_num	
<input type="checkbox"/> passengers_on_...	

Foreign Key Options

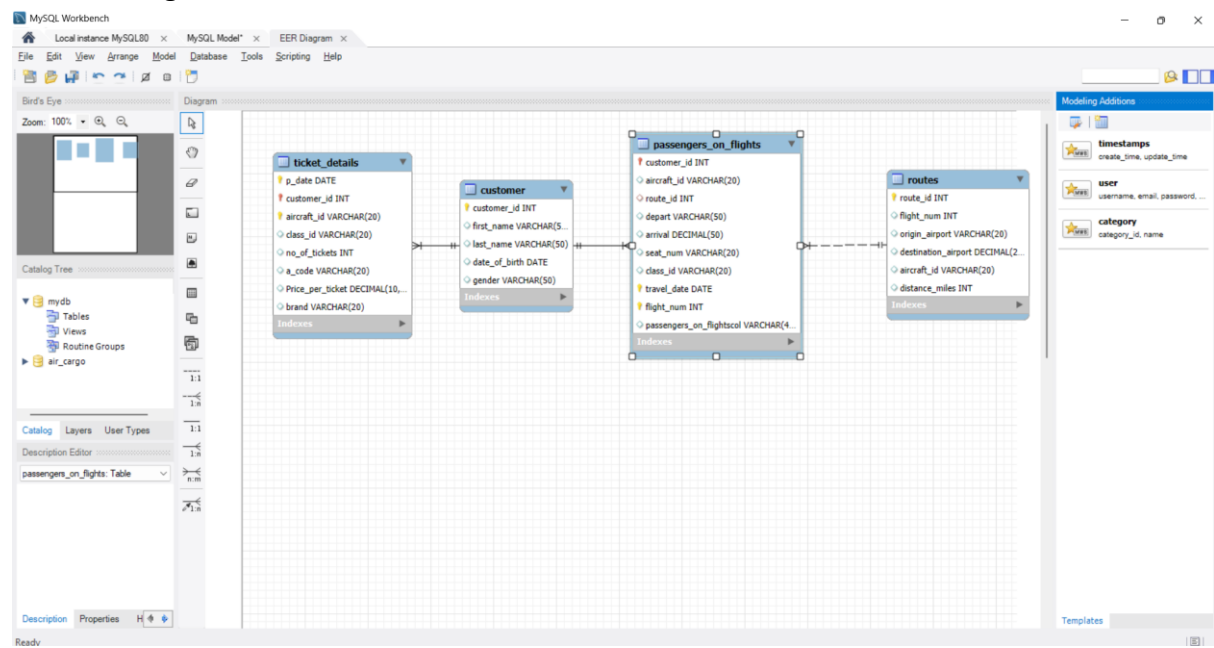
On Update: NO ACTION

On Delete: NO ACTION

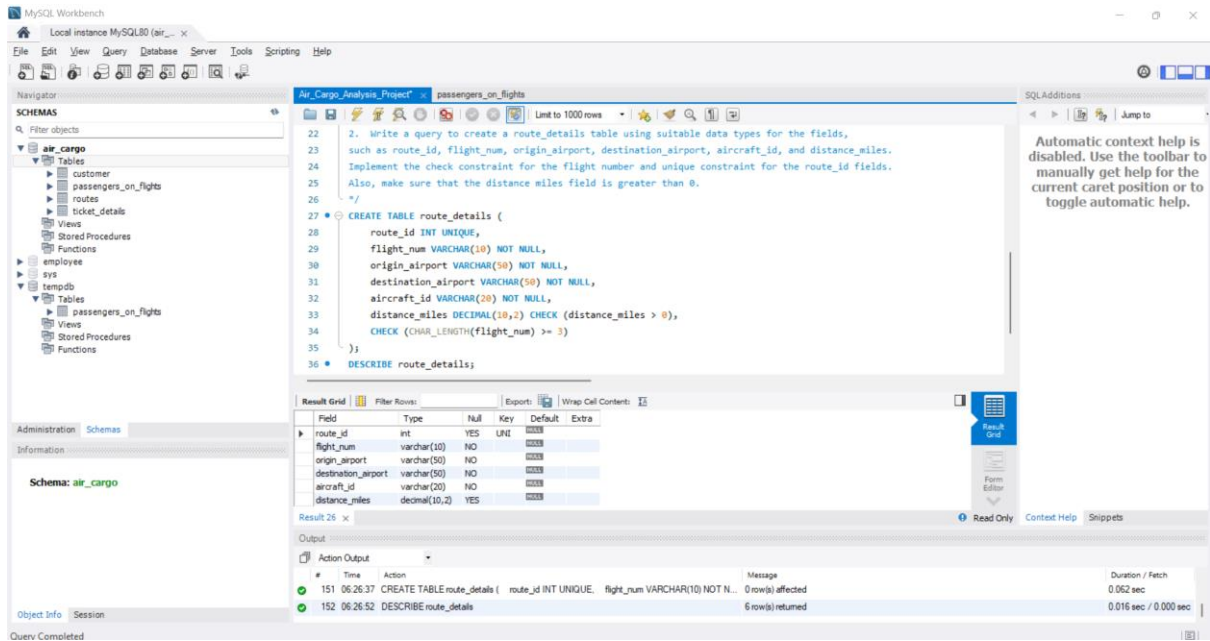
☐ Skip in SQL generation

Foreign Key Comment

## Final ER Diagram:



- Write a query to create a **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.

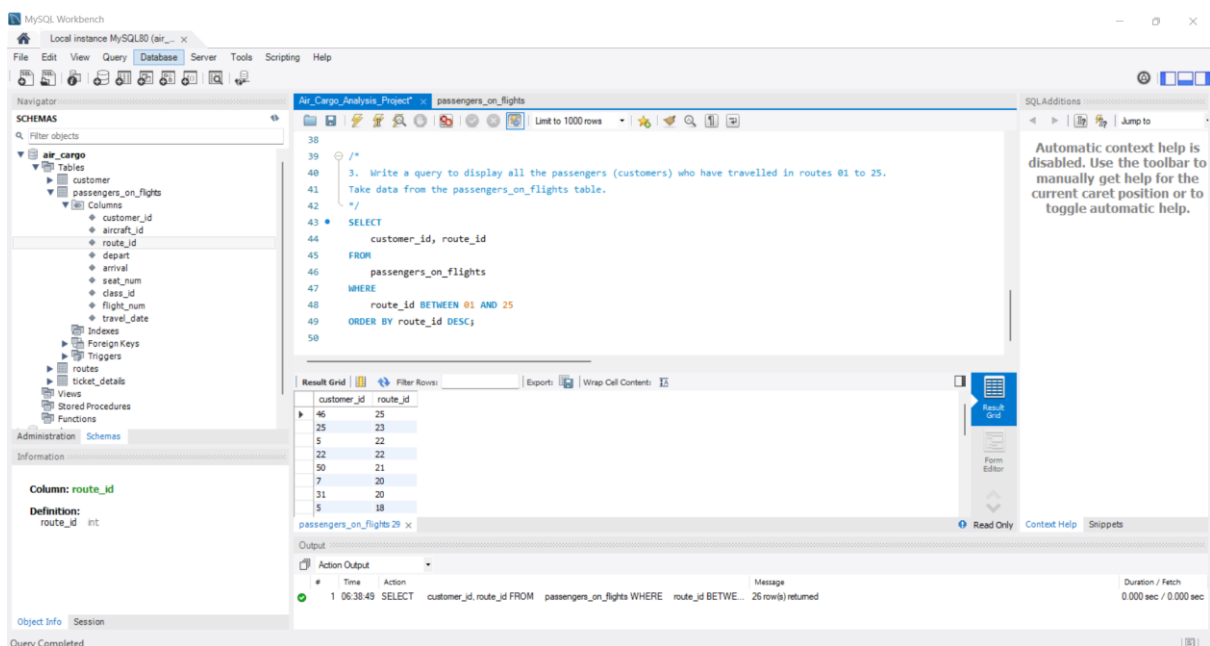


### Note:

The route\_id field has been defined with a **UNIQUE constraint** as per the problem statement.

Alternatively, it could also be defined as a **PRIMARY KEY**, which would inherently enforce uniqueness and not allow NULL values at the same time.

3. 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.



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

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'air\_cargo' selected. The 'ticket\_details' table is highlighted. The main editor contains the following SQL query:

```
4. Write a query to identify the number of passengers and total revenue
in business class from the ticket_details table.

SELECT
    COUNT(customer_id) AS number_of_passengers,
    SUM(price_per_ticket * no_of_tickets) AS total_revenue
FROM ticket_details
WHERE class_id = 'business';
```

The 'Result Grid' shows the output of the query:

number_of_passengers	total_revenue
13	6034

The bottom status bar indicates the query was completed successfully, returning 1 row(s).

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

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'air\_cargo' selected. The 'customer' table is highlighted. The main editor contains the following SQL query:

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

SELECT
    customer_id,
    CONCAT_WS(' ', TRIM(first_name), TRIM(last_name)) AS 'full_name',
    gender
FROM
    customer
ORDER BY full_name;
-- Using CONCAT_WS instead of CONCAT allows handling nulls gracefully
```

The 'Result Grid' shows the output of the query:

customer_id	full_name	gender
5	Alron Kim	M
40	Adam Paul	M
6	Alexander Scott	M
38	Alexis Scott	M
7	Anderson Stewart	M
44	Bly Brian	M
26	Bryan Colin	M
24	Calev Wilfr	M
14	Carol Vernon	F
4	Cathenna Emily	F

The bottom status bar indicates the query was completed successfully, returning 50 row(s).



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

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with the 'ticket\_details' table selected. The main editor contains the following SQL query:

```
6. 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,
  CONCAT(TRIM(c.first_name), ' ', TRIM(c.last_name)) AS full_name,
  c.gender,
  td.no_of_tickets
FROM
  customer c
  INNER JOIN
  ticket_details td ON c.customer_id = td.customer_id
ORDER BY full_name;
```

The 'Result Grid' shows the following data:

customer_id	full_name	gender	no_of_tickets
5	Aaron Kim	M	1
5	Aaron Kim	M	1
5	Aaron Kim	M	1
7	Anderson Stewart	M	1
44	Bly Brian	M	1
24	Calvin Willis	M	1
14	Carol Vernon	F	1

The 'Output' pane shows the execution details: 1 07:44:06 SELECT c.customer\_id, CONCAT(TRIM(c.first\_name), ' ', TRIM(c.last\_name)) AS full\_name, c.gender, td.no\_of\_tickets FROM customer c INNER JOIN ticket\_details td ON c.customer\_id = td.customer\_id ORDER BY full\_name; 50 row(s) returned. Duration / Fetch: 0.000 sec / 0.000 sec.

7. 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.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with the 'ticket\_details' table selected. The main editor contains the following SQL query:

```
7. 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
  td.customer_id,
  CONCAT_WS(' ', TRIM(c.first_name), TRIM(c.last_name)) AS full_name,
  td.brand
FROM
  ticket_details td
  INNER JOIN
  customer c ON c.customer_id = td.customer_id
WHERE
  td.brand = 'Emirates'
ORDER BY full_name;
```

The 'Result Grid' shows the following data:

customer_id	full_name	brand
5	Aaron Kim	Emirates
7	Anderson Stewart	Emirates
44	Bly Brian	Emirates
14	Carol Vernon	Emirates
4	Cathenna Emily	Emirates
4	Cathenna Emily	Emirates
27	Cheryl Vernon	Emirates

The 'Output' pane shows the execution details: 1 10:24:19 SELECT td.customer\_id, CONCAT\_WS(' ', TRIM(c.first\_name), TRIM(c.last\_name)) AS full\_name, td.brand FROM ticket\_details td INNER JOIN customer c ON c.customer\_id = td.customer\_id WHERE td.brand = 'Emirates' ORDER BY full\_name; 18 row(s) returned. Duration / Fetch: 0.015 sec / 0.000 sec.



- 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.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' panel with a tree view of the database structure, including tables like 'customer', 'passengers\_on\_flights', and 'ticket\_details'. The main editor window contains a SQL query:

```

106  /*
107  8. Write a query to identify the customers who have travelled by Economy Plus class
108  using Group By and Having clause on the passengers_on_flights table.
109  */
110  SELECT
111    pf.customer_id,
112    CONCAT(TRIM(c.first_name), ' ', TRIM(c.last_name)) AS full_name,
113    COUNT(*) AS total_flights
114  FROM
115    passengers_on_flights pf
116  INNER JOIN
117    customer c ON pf.customer_id = c.customer_id
118  GROUP BY pf.customer_id, full_name
119  HAVING SUM(pf.class_id = 'economy plus');

```

The 'Result Grid' shows the output of the query:

customer_id	full_name	total_flights
1	Julie Sam	2
8	Poyd Ted	2
11	Roger Walton	3
17	Catherine Shad	1
19	Joyce Paul	3
22	Pheny Eri	1

The 'Output' panel shows the execution details:

#	Time	Action	Message	Duration / Fetch
1	10:43:14	SELECT	pf.customer_id, CONCAT(TRIM(c.first_name), ' ', TRIM(c.last_name)) AS full_name, COUNT(*) AS total_flights FROM passengers_on_flights pf INNER JOIN customer c ON pf.customer_id = c.customer_id GROUP BY pf.customer_id, full_name HAVING SUM(pf.class_id = 'economy plus');	0.000 sec / 0.000 sec

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

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' panel with a tree view of the database structure, including tables like 'customer', 'passengers\_on\_flights', and 'ticket\_details'. The main editor window contains a SQL query:

```

120  /*
121  9. Write a query to identify whether the revenue has crossed 10000
122  using the IF clause on the ticket_details table.
123  */
124  SELECT
125    SUM(price_per_ticket * no_of_tickets) AS total_revenue,
126    IF(SUM(price_per_ticket * no_of_tickets) > 10000,
127       'Revenue has crossed 10000',
128       'Revenue has not crossed 10000') AS revenue_status
129  FROM
130    ticket_details;

```

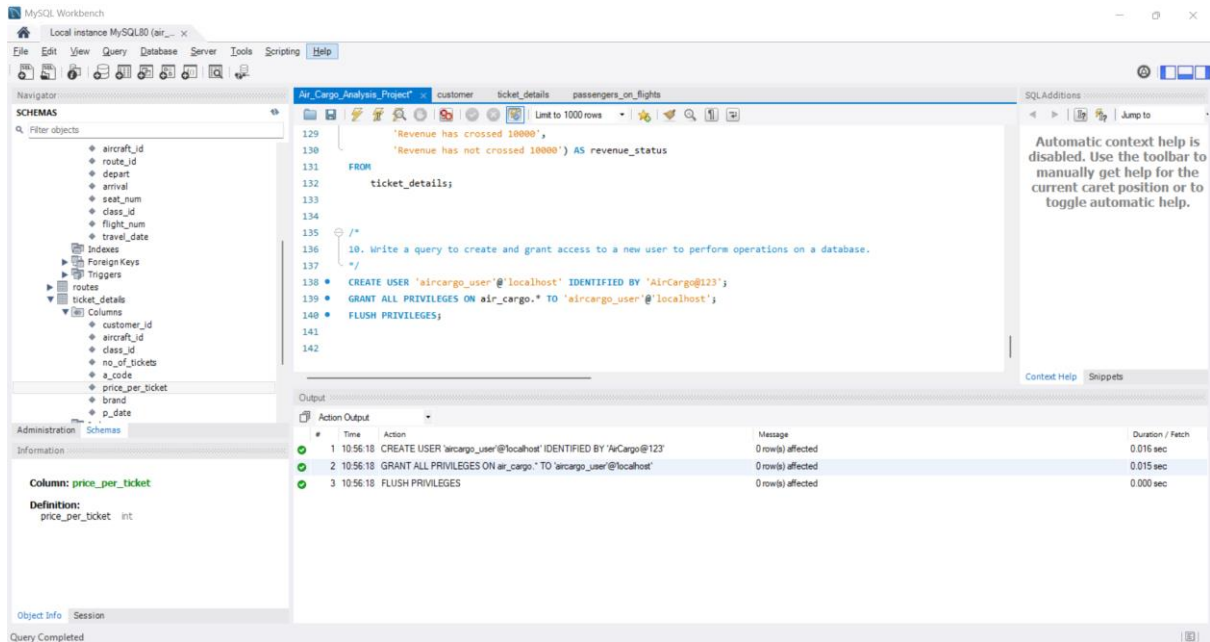
The 'Result Grid' shows the output of the query:

total_revenue	revenue_status
15369	Revenue has crossed 10000

The 'Output' panel shows the execution details:

#	Time	Action	Message	Duration / Fetch
2	10:45:29	SELECT	* FROM ticket_details LIMIT 0, 1000	0.000 sec / 0.000 sec
3	10:47:07	SELECT	SUM(price_per_ticket * no_of_tickets) AS total_revenue FROM ticket_details LIMIT 1 row(s) returned	0.015 sec / 0.000 sec
4	10:48:27	SELECT	SUM(price_per_ticket * no_of_tickets) AS total_revenue, IF(SUM(price_per_ticket * no_of_tickets) > 10000, 'Revenue has crossed 10000', 'Revenue has not crossed 10000') AS revenue_status FROM ticket_details;	0.000 sec / 0.000 sec

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



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'Schemas' tree with 'air\_cargo' selected. The main editor contains a SQL query to create a user and grant privileges. The 'Output' tab at the bottom shows the execution results.

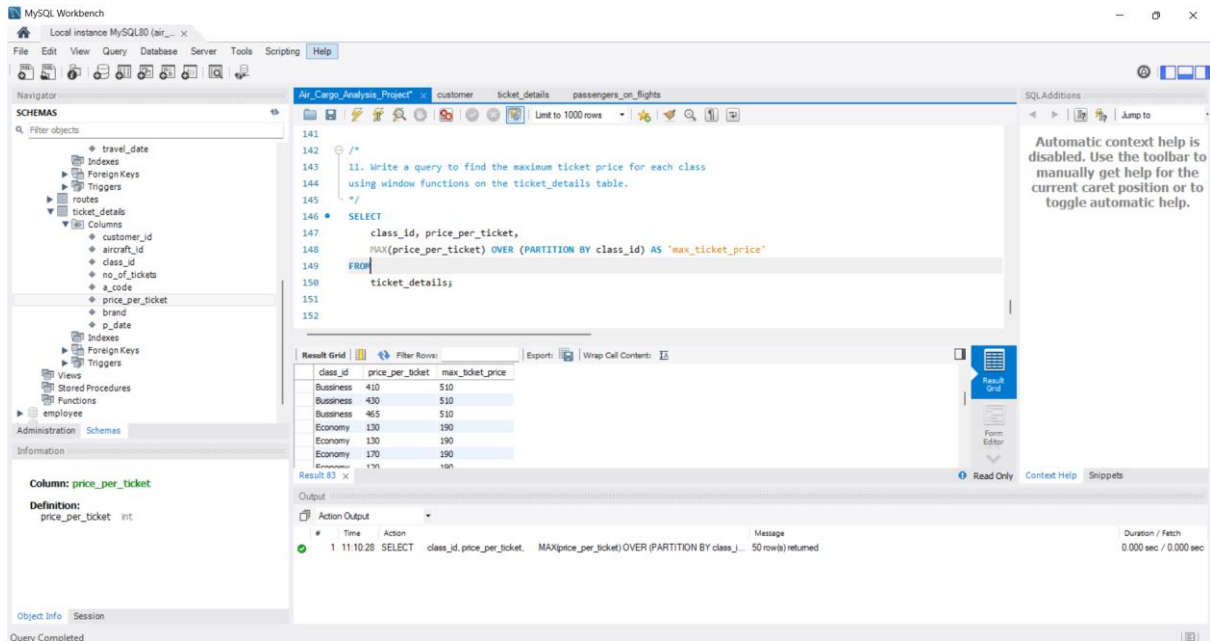
```

129      'Revenue has crossed 100000',
130      'Revenue has not crossed 100000') AS revenue_status
131  FROM
132      ticket_details;
133
134  /*
135   * 10. Write a query to create and grant access to a new user to perform operations on a database.
136   */
137
138  CREATE USER 'aircargo_user'@'localhost' IDENTIFIED BY 'AirCargo@123';
139  GRANT ALL PRIVILEGES ON air_cargo.* TO 'aircargo_user'@'localhost';
140  FLUSH PRIVILEGES;
141
142

```

#	Time	Action	Message	Duration / Fetch
1	10:56:18	CREATE USER 'aircargo_user'@'localhost' IDENTIFIED BY 'AirCargo@123'	0 row(s) affected	0.016 sec
2	10:56:18	GRANT ALL PRIVILEGES ON air_cargo.* TO 'aircargo_user'@'localhost'	0 row(s) affected	0.015 sec
3	10:56:18	FLUSH PRIVILEGES	0 row(s) affected	0.000 sec

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



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'Schemas' tree with 'air\_cargo' selected. The main editor contains a SQL query to find the maximum ticket price for each class using window functions. The 'Result Grid' shows the output of the query.

```

141
142  /*
143   * 11. Write a query to find the maximum ticket price for each class
144   * using window functions on the ticket_details table.
145   */
146
147  SELECT
148      class_id, price_per_ticket,
149      MAX(price_per_ticket) OVER (PARTITION BY class_id) AS 'max_ticket_price'
150  FROM
151      ticket_details;
152

```

class_id	price_per_ticket	max_ticket_price
Business	410	510
Business	430	510
Business	465	510
Economy	130	190
Economy	130	190
Economy	170	190
Economy	170	190

12. Write a query to extract the passengers whose route ID is 4 by improving the speed and performance of the passengers\_on\_flights table.

## Performance before indexing:

The screenshot shows the MySQL Workbench interface. The query editor contains the following SQL code:

```
149 FROM
150 ticket_details;
151
152 /*
153 12. Write a query to extract the passengers
154 whose route ID is 4 by improving the speed and performance of the passengers_on_flights table.
155 */
156 SELECT *
157 FROM passengers_on_flights
```

The Visual Explain tab shows the execution plan for the query. The plan indicates a 'Full Table Scan' on the passengers\_on\_flights table, with a query cost of 5.25 and 50 rows returned. The output table shows the results of the query, including the column 'price\_per\_ticket'.

#	Time	Action	Message	Duration / Fetch
1	23:28:04	SELECT * FROM passengers_on_flights WHERE route_id = 4 LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
2	23:28:07	EXPLAIN SELECT * FROM passengers_on_flights WHERE route_id = 4	OK	0.000 sec
3	23:28:07	EXPLAIN FORMAT=JSON SELECT * FROM passengers_on_flights WHERE route_id = 4	OK	0.000 sec

## Performance before indexing:

The screenshot shows the MySQL Workbench interface. The query editor contains the following SQL code:

```
153 12. Write a query to extract the passengers
154 whose route ID is 4 by improving the speed and performance of the passengers_on_flights table.
155 */
156 SELECT *
157 FROM passengers_on_flights
158 WHERE route_id = 4;
159
160 CREATE INDEX idx_route_id ON passengers_on_flights(route_id);
161 SELECT *
162 FROM passengers_on_flights
163 WHERE route_id = 4;
```

The Visual Explain tab shows the execution plan for the query. The plan indicates a 'Non-Unique Key Lookup' on the passengers\_on\_flights table, with a query cost of 1.05 and 3 rows returned. The output table shows the results of the query, including the column 'route\_id'.

#	Time	Action	Message	Duration / Fetch
5	23:32:05	SELECT * FROM passengers_on_flights WHERE route_id = 4 LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
6	23:32:08	EXPLAIN SELECT * FROM passengers_on_flights WHERE route_id = 4	OK	0.000 sec
7	23:32:09	EXPLAIN FORMAT=JSON SELECT * FROM passengers_on_flights WHERE route_id = 4	OK	0.000 sec

## Observation:

Before indexing, the query to extract passengers whose route\_id = 4 performed a **full table scan**, resulting in a higher execution cost (**5.25**) and slower performance as the entire dataset had to be scanned sequentially.

After creating an **index** on the route\_id column, the execution plan showed that the query used the index which significantly **reduced the number of rows scanned** and improved the overall query execution speed (**1.05**).

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

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with the 'passengers\_on\_flights' table selected. The main editor shows a query window with the following SQL code:

```
163 WHERE route_id = 4;
164
165 /*
166 13. For the route ID 4, write a query to view the execution plan of the passengers_on_flights table.
167 */
168 * EXPLAIN SELECT *
169 FROM passengers_on_flights
170 WHERE route_id = 4;
```

Below the query editor, the 'Result Grid' is displayed, showing the execution plan for the query. The grid has columns: id, select\_type, table, partitions, type, possible\_keys, key, key\_len, ref, rows, filtered, Extra. The first row shows a 'SIMPLE' select type, 'passengers\_on\_flights' table, 'ref' type, and 'idx\_route\_id' key.

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	passengers_on_flights		ref	idx_route_id	idx_route_id	5	const	3	100.00	

The bottom panel shows the 'Output' tab with the following message:

```
1 23:43:49 EXPLAIN SELECT * FROM passengers_on_flights WHERE route_id = 4
Message: 1 row(s) returned
Duration / Fetch: 0.000 sec / 0.000 sec
```

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

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```

171
172
173 14. Write a query to calculate the total price of all tickets booked by a customer
174 across different aircraft IDs using rollup function.
175 */
176 SELECT
177     customer_id,
178     aircraft_id,
179     SUM(no_of_tickets * price_per_ticket) AS total_price
180 FROM
181     ticket_details
182 GROUP BY customer_id, aircraft_id WITH ROLLUP;

```

The Result Grid shows the following data:

customer_id	aircraft_id	total_price
1	CRJ900	320
1	ERJ142	250
1	ERJ142	570
2	767-30 IER	130
2	A321	505
2	ERJ142	635
4	767-30 IER	780
4	ERJ142	780
5	767-30 IER	430
5	ERJ142	240

The Output pane shows the execution of the query, indicating that 75 rows were returned.

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

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```

185
186 15. Write a query to create a view with only
187 business class customers along with the brand of airlines.
188 */
189 CREATE VIEW business_class_cust AS
190 SELECT
191     customer_id, class_id AS 'Class', brand
192 FROM
193     ticket_details
194 WHERE
195     class_id = 'business';
196 SELECT * FROM business_class_cust;

```

The Result Grid shows the following data:

customer_id	Class	brand
21	Business	British Airways
7	Business	Emirates
11	Business	Emirates
25	Business	Emirates
24	Business	Qatar Airways
29	Business	Qatar Airways
2	Business	Qatar Airways
29	Business	Jet Airways
5	Business	Emirates
15	Business	Qatar Airways

The Output pane shows the execution of the query, indicating that 13 rows were returned.

16. 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.

The screenshot shows the MySQL Workbench interface with a local instance of MySQL80. The 'Schemas' pane on the left shows the 'air\_cargo' schema. The main editor displays the following SQL code:

```

DELIMITER //
CREATE PROCEDURE get_passenger_details(IN start_route INT, IN end_route INT)
BEGIN
    IF (SELECT COUNT(*)
        FROM passengers_on_flights) >= 0 THEN
        SELECT *
        FROM passengers_on_flights
        WHERE route_id BETWEEN start_route AND end_route;
    ELSE
        SELECT 'Error: Table passengers_on_flights does not exist' AS message;
    END IF;
END //
DELIMITER ;

CALL get_passenger_details(5, 10);

```

The 'Result Grid' shows the output of the procedure call, displaying a table with columns: customer\_id, aircraft\_id, route\_id, depart, arrival, seat\_num, class\_id, flight\_num, travel\_date. The table contains 10 rows of data.

The 'Action Output' pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
12	01:10:39	CREATE PROCEDURE get_passenger_details(IN start_route INT, IN end_route INT) BEGIN ...	0 row(s) affected	0.016 sec
13	01:10:49	CALL get_passenger_details(5, 10)	6 row(s) returned	0.047 sec / 0.000 sec

17. 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.

The screenshot shows the MySQL Workbench interface with a local instance of MySQL80. The 'Schemas' pane on the left shows the 'air\_cargo' schema. The main editor displays the following SQL code:

```

/*
17. 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 get_routes_over_2000()
BEGIN
    SELECT *
    FROM routes
    WHERE distance_miles > 2000
    ORDER BY distance_miles DESC;
END //
DELIMITER ;

CALL get_routes_over_2000();

```

The 'Result Grid' shows the output of the procedure call, displaying a table with columns: route\_id, flight\_num, origin\_airport, destination\_airport, aircraft\_id, distance\_miles. The table contains 6 rows of data.

The 'Action Output' pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	01:19:47	CREATE PROCEDURE get_routes_over_2000() BEGIN SELECT * FROM routes WHERE data...	0 row(s) affected	0.031 sec
2	01:19:55	CALL get_routes_over_2000()	24 row(s) returned	0.000 sec / 0.000 sec



18. 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  $\geq 0$  AND  $\leq 2000$  miles, intermediate distance travel (IDT) for  $> 2000$  AND  $\leq 6500$ , and long-distance travel (LDT) for  $> 6500$ .

The screenshot shows the MySQL Workbench interface. The main editor displays the following SQL code:

```
DELIMITER //
CREATE PROCEDURE categorised_flight_distance ()
BEGIN
    SELECT
        flight_num,
        distance_miles,
        CASE
            WHEN distance_miles >= 0 AND distance_miles <= 2000 THEN 'SDT'
            WHEN distance_miles > 2000 AND distance_miles <= 6500 THEN 'IDT'
            ELSE 'LDT'
        END AS 'distance_category'
    FROM routes
    ORDER BY distance_miles;
END //
DELIMITER ;
CALL categorised_flight_distance();
```

The left sidebar shows the 'SCHEMAS' panel with a tree view of the 'air\_cargo' database, including tables, views, stored procedures, and functions. The 'Result Grid' shows the output of the query, displaying flight numbers and their corresponding distance categories.

flight_num	distance_miles	distance_category
1138	246	SDT
1142	246	SDT
1137	578	SDT
1141	660	SDT
1157	675	SDT
1155	676	SDT

The bottom panel shows the 'Action Output' tab, which displays the execution details of the stored procedure:

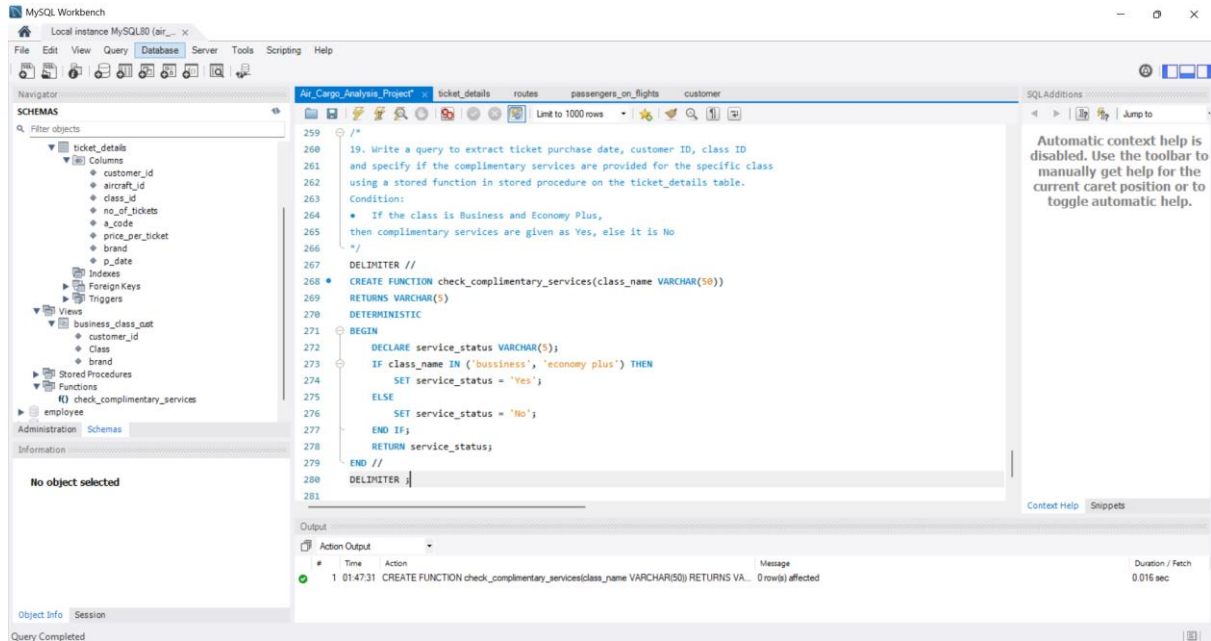
#	Time	Action	Message	Duration / Fetch
1	01:37:16	CREATE PROCEDURE categorised_flight_distance () BEGIN SELECT flight_num, ...	0 row(s) affected	0.015 sec
2	01:37:32	CALL categorised_flight_distance()	49 row(s) returned	0.016 sec / 0.000 sec



19. 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*

## Creating Function:



The screenshot shows the MySQL Workbench interface with the 'Air\_Cargo\_Analysis\_Project' database selected. The 'ticket\_details' table is highlighted in the Schemas pane. The SQL editor contains the following code:

```

259  /*
260  19. Write a query to extract ticket purchase date, customer ID, class ID
261  and specify if the complimentary services are provided for the specific class
262  using a stored function in stored procedure on the ticket_details table.
263  Condition:
264  • If the class is Business and Economy Plus,
265  then complimentary services are given as Yes, else it is No
266  */
267  DELIMITER //
268  CREATE FUNCTION check_complimentary_services(class_name VARCHAR(50))
269  RETURNS VARCHAR(5)
270  DETERMINISTIC
271  BEGIN
272      DECLARE service_status VARCHAR(5);
273      IF class_name IN ('business', 'economy plus') THEN
274          SET service_status = 'Yes';
275      ELSE
276          SET service_status = 'No';
277      END IF;
278      RETURN service_status;
279  END //
280  DELIMITER ;
281

```

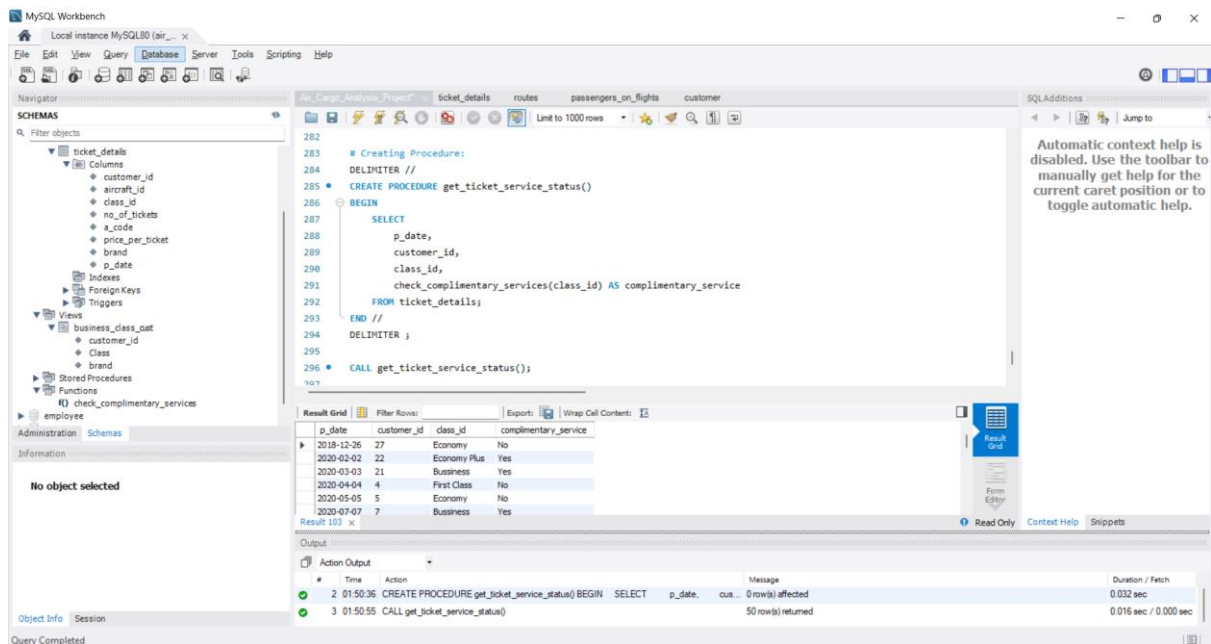
The Output pane shows the successful execution of the function creation:

```

1 01:47:31 CREATE FUNCTION check_complimentary_services(class_name VARCHAR(50)) RETURNS VA... 0 row(s) affected
0.016 sec

```

## Creating PROCEDURE:



The screenshot shows the MySQL Workbench interface with the 'Air\_Cargo\_Analysis\_Project' database selected. The 'ticket\_details' table is highlighted in the Schemas pane. The SQL editor contains the following code:

```

282  /* Creating Procedure:
283  DELIMITER //
284  CREATE PROCEDURE get_ticket_service_status()
285  BEGIN
286      SELECT
287          p_date,
288          customer_id,
289          class_id,
290          check_complimentary_services(class_id) AS complimentary_service
291      FROM ticket_details;
292  END //
293  DELIMITER ;
294
295  CALL get_ticket_service_status();
296

```

The Output pane shows the successful execution of the procedure creation and its call:

```

2 01:50:36 CREATE PROCEDURE get_ticket_service_status() BEGIN SELECT p_date, cus... 0 row(s) affected 0.032 sec
3 01:50:55 CALL get_ticket_service_status() 50 row(s) returned 0.016 sec / 0.000 sec

```

The Result Grid shows the output of the procedure call:

p_date	customer_id	class_id	complimentary_service
2018-12-26	27	Economy	No
2020-02-02	22	Economy Plus	Yes
2020-03-03	21	Business	Yes
2020-04-04	4	First Class	No
2020-05-05	5	Economy	No
2020-07-07	7	Business	Yes

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

NOTE: Cursor operations were not covered during Live Class, hence this query was not implemented

### **Conclusion:**

In this project, the *Air Cargo* database was created and analysed through a series of SQL operations designed to enhance the company's route and customer management. The tasks included:

- Designing an ER diagram and defining primary and foreign key relationships.
- Writing queries to extract passenger, route, and ticket details.
- Applying conditional logic using IF, CASE, and aggregation functions.
- Creating views, stored procedures, and stored functions for automated and reusable operations.
- Using indexing and the EXPLAIN plan to improve query performance.
- Implementing advanced SQL concepts such as ROLLUP, WINDOW FUNCTIONS, and runtime parameterised procedures.

Through these implementations, the project demonstrates practical applications of SQL in database design, performance optimisation, and analytical reporting — enabling more efficient decision-making for route planning, customer offers, and revenue tracking in the aviation domain.