

# Assignment 5

## Ticket Booking System

### Task1:

The screenshot shows the MySQL Workbench interface with the following SQL code:

```
1 /* Assignment 5 */
2 /* Task1 */
3 * CREATE DATABASE TicketBookingSystem;
4 * USE TicketBookingSystem;
5 *
6
7 * CREATE TABLE Venu {
8     venue_id INT PRIMARY KEY,
9     venue_name VARCHAR(255),
10    address VARCHAR(255)
11 } ;
12
13
14 * CREATE TABLE Event {
15     event_id INT PRIMARY KEY,
16     event_name VARCHAR(255),
17     event_date DATE,
18     event_time TIME,
19     venue_id INT,
20     total_seats INT,
21     available_seats INT,
22     ticket_price DECIMAL(10, 2),
23     event_type VARCHAR(50),
24     booking_id INT
25 } ;
26
27
28 * CREATE TABLE Booking {
29     booking_id INT PRIMARY KEY,
30     customer_id INT,
31     event_id INT,
32     num_tickets INT,
33     total_cost DECIMAL(10, 2),
34     booking_date DATE
35 } ;
36
37
38 * CREATE TABLE Customer {
39     customer_id INT PRIMARY KEY,
40     customer_name VARCHAR(255),
41     email VARCHAR(255),
42     phone_number VARCHAR(15),
43     booking_id INT
44 } ;
45
46
47 * ALTER TABLE Customer
48 ADD FOREIGN KEY (booking_id) REFERENCES Booking(booking_id);
49
50
51 * ALTER TABLE Booking
52 ADD (FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
53      FOREIGN KEY (event_id) REFERENCES Event(event_id));
54
55
56 * ALTER TABLE Event
57 ADD (FOREIGN KEY (venue_id) REFERENCES Venu(venue_id),
58      FOREIGN KEY (booking_id) REFERENCES Booking(booking_id));
59
60
61 * ALTER TABLE Event
62 ADD CHECK (event_type IN ('Movie', 'Sports', 'Concert'));
```

```

58
59 * ALTER TABLE Event
60   ADD FOREIGN KEY (venue_id) REFERENCES Venue(venue_id),
61     FOREIGN KEY (booking_id) REFERENCES Booking(booking_id));
62
63 * ALTER TABLE Event
64   ADD CHECK (event_type IN ('Movie', 'Sports', 'Concert'));
65

```

DML			
#	Time	Action	Message
1	15:31:55	CREATE DATABASE TicketBookingSystem	1 row(s) affected
2	15:31:57	USE TicketBookingSystem	0 rows(s) affected
3	15:31:58	CREATE TABLE Venue (venue_id INT PRIMARY KEY, venue_name VARCHAR(255), address VARCHAR(255))	0 rows(s) affected
4	15:32:03	CREATE TABLE Event (event_id INT PRIMARY KEY, event_name VARCHAR(255), event_date DATE)	0 rows(s) affected
5	15:32:05	CREATE TABLE Booking (booking_id INT PRIMARY KEY, customer_id INT, event_id INT, num_tickets INT)	0 rows(s) affected
6	15:32:08	CREATE TABLE Customer (customer_id INT PRIMARY KEY, customer_name VARCHAR(255), email VARCHAR(255))	0 rows(s) affected
7	15:32:18	ALTER TABLE Customer ADD FOREIGN KEY booking_id REFERENCES Booking(booking_id)	0 rows(s) affected Records: 0 Duplicates: 0 Warnings: 0
8	15:32:20	ALTER TABLE Booking ADD FOREIGN KEY (customer_id) REFERENCES Customer(customer_id); FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)	0 rows(s) affected Records: 0 Duplicates: 0 Warnings: 0
9	15:32:23	ALTER TABLE Event ADD FOREIGN KEY (venue_id) REFERENCES Venue(venue_id); FOREIGN KEY (venue_id) REFERENCES Venue(venue_id)	0 rows(s) affected Records: 0 Duplicates: 0 Warnings: 0
10	15:32:40	ALTER TABLE Event ADD CHECK (event_type IN ('Movie', 'Sports', 'Concert'))	0 rows(s) affected Records: 0 Duplicates: 0 Warnings: 0

## TASK 2:

### 1.

```

Management Studio | SQL | Help | Home | Log in to 1000 rows | Back | Forward | Stop | Refresh | My Snippets | Help | Address: http://127.0.0.1:5000/
111 * INSERT INTO Venue (venue_id, venue_name, address) VALUES
112 (1, 'Grand Theater', '123 Park Street, Cityville'),
113 (2, 'City Arena', '456 Center Avenue, Townsville'),
114 (3, 'Sports Stadium', '789 Stadium Road, Sportsborough'),
115 (4, 'Film Palace', '101 Movie Lane, CinemaVille'),
116 (5, 'Concert Hall', '200 Melody Street, Harmonytown'),
117 (6, 'Community Center', '300 Social Square, Gatheringburg'),
118 (7, 'LIVE Lounge', '400 Entertainment Avenue, Funecity'),
119 (8, 'Cinematic Complex', '500 Film Street, Filmington'),
120 (9, 'Soccer Park', '600 Ball Street, Kicksville'),
121 (10, 'Music Box', '700 Harmony Road, Concertburg');

122 *
123 * INSERT INTO Event (event_id, event_name, event_date, event_time, venue_id, total_seats, available_seats, ticket_price, event_type, booking_id)
124 (1, 'Music Night: Invasion!', '2023-01-15', '12:00:00', 1, 150, 120, 220.00, 'Music', NULL),
125 (2, 'Concert: Acoustic Vibes', '2023-02-28', '20:00:00', 2, 200, 150, 120.00, 'Concert', NULL),
126 (3, 'Soccer Match: City Rivals', '2023-03-29', '20:00:00', 3, 200, 180, 110.00, 'Sports', NULL),
127 (4, 'Movie Night: The Great Getaway', '2023-04-10', '21:00:00', 4, 120, 80, 150.00, 'Movie', NULL),
128 (5, 'Concert: Pop Revolution', '2023-05-05', '17:45:00', 5, 250, 200, 350.00, 'Concert', NULL),
129 (6, 'LIVE Music: Rock Fusing', '2023-06-12', '20:00:00', 6, 100, 80, 140.00, 'Concert', NULL),
130 (7, 'Basketball Game: Finals!', '2023-07-08', '18:00:00', 7, 200, 100, 130.00, 'Sports', NULL),
131 (8, 'Movie Night: Classics', '2023-08-29', '20:00:00', 8, 200, 120, 220.00, 'Movie', NULL),
132 (9, 'Soccer Match: International Clash', '2023-09-15', '19:45:00', 9, 200, 100, 120.00, 'Sports', NULL),
133 (10, 'Concert: Rock Revolution', '2023-10-10', '22:00:00', 10, 250, 200, 350.00, 'Concert', NULL);

```

```

Management Studio | SQL | Help | Home | Log in to 1000 rows | Back | Forward | Stop | Refresh | My Snippets | Help | Address: http://127.0.0.1:5000/
134 * INSERT INTO Customer (customer_id, customer_name, email, phone_number, booking_id) VALUES
135 (1, 'John Doe', 'john.doe@email.com', '555-1234', NULL),
136 (2, 'Jane Smith', 'jane.smith@email.com', '555-2345', NULL),
137 (3, 'Robert Johnson', 'robert.johnson@email.com', '555-9876', NULL),
138 (4, 'Elizabeth Brown', 'elizabeth.brown@email.com', '555-8987', NULL),
139 (5, 'Chris Miller', 'chris.miller@email.com', '555-7890', NULL),
140 (6, 'Anna White', 'anna.white@email.com', '555-6789', NULL),
141 (7, 'Michael Davis', 'michael.davis@email.com', '555-5678', NULL),
142 (8, 'Elisabeth Taylor', 'elisabeth.taylor@email.com', '555-4321', NULL),
143 (9, 'Daniel Wilson', 'daniel.wilson@email.com', '555-3210', NULL),
144 (10, 'Sophia Adams', 'sophia.adams@email.com', '555-9876', NULL);

145 *
146 * INSERT INTO Booking (booking_id, customer_id, event_id, num_tickets, total_cost, booking_date) VALUES
147 (1, 1, 1, 2, 440.00, '2023-01-15'),
148 (2, 2, 2, 3, 585.00, '2023-02-28'),
149 (3, 3, 3, 1, 120.00, '2023-03-29'),
150 (4, 4, 4, 4, 620.00, '2023-04-10'),
151 (5, 5, 5, 3, 480.00, '2023-05-05'),
152 (6, 6, 6, 2, 430.00, '2023-06-12'),
153 (7, 7, 7, 3, 600.00, '2023-07-08'),
154 (8, 8, 8, 1, 220.00, '2023-08-29'),
155 (9, 9, 9, 1, 240.00, '2023-09-15'),
156 (10, 10, 10, 350.00, '2023-10-10');

157 *
158 * UPDATE Event SET booking_id = 1 WHERE event_id = 1;
159 * UPDATE Event SET booking_id = 2 WHERE event_id = 2;
160 * UPDATE Event SET booking_id = 3 WHERE event_id = 3;
161 * UPDATE Event SET booking_id = 4 WHERE event_id = 4;
162 * UPDATE Event SET booking_id = 5 WHERE event_id = 5;
163 * UPDATE Event SET booking_id = 6 WHERE event_id = 6;
164 * UPDATE Event SET booking_id = 7 WHERE event_id = 7;
165 * UPDATE Event SET booking_id = 8 WHERE event_id = 8;
166 * UPDATE Event SET booking_id = 9 WHERE event_id = 9;
167 * UPDATE Event SET booking_id = 10 WHERE event_id = 10;

168 *
169 * UPDATE Customer SET booking_id = 1 WHERE customer_id = 1;
170 * UPDATE Customer SET booking_id = 2 WHERE customer_id = 2;
171 * UPDATE Customer SET booking_id = 3 WHERE customer_id = 3;
172 * UPDATE Customer SET booking_id = 4 WHERE customer_id = 4;
173 * UPDATE Customer SET booking_id = 5 WHERE customer_id = 5;
174 * UPDATE Customer SET booking_id = 6 WHERE customer_id = 6;
175 * UPDATE Customer SET booking_id = 7 WHERE customer_id = 7;
176 * UPDATE Customer SET booking_id = 8 WHERE customer_id = 8;
177 * UPDATE Customer SET booking_id = 9 WHERE customer_id = 9;
178 * UPDATE Customer SET booking_id = 10 WHERE customer_id = 10;

```

2.

```
42
43 */*2*/ SELECT * FROM Event;
44
45
```

Result Grid | Filter Rows: | Edit | Export|Import: | Wrap Cell Contents: |

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
1	Movie Night: Decipher	2023-01-15	18:00:00	1	100	100	220.00	Movie	1
2	Concert: Acoustic Vibes	2023-02-10	20:00:00	2	300	290	125.00	Concert	2
3	Soccer Match: City Rivals	2023-03-25	19:30:00	3	200	180	152.00	Sports	3
4	Movie Night: The Great Gatsby	2023-04-10	21:00:00	4	120	80	185.00	Movie	4
5	Concert: Pop Explosion	2023-05-01	17:45:00	5	250	200	330.00	Concert	5
6	Live Music: Jazz Evening	2023-06-12	19:00:00	6	300	280	140.00	Concert	6
7	Basketball Game: Finals	2023-07-08	18:30:00	7	300	300	120.00	Sports	7
8	Movie Night: PuddingFest	2023-08-15	20:15:00	8	150	100	175.00	Movie	8

Action Output: \* Time: Action: Message: Duration / Fetch: 1 rows returned 0.000 sec / 0.000 sec

Event 4

Contact Help | Support | Documentation

3.

```
46
47 /*3*/ SELECT * FROM Event WHERE available_seats > 0;
48
49
```

Result Grid | Filter Rows: | Edit | Export|Import: | Wrap Cell Contents: |

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
1	Movie Night: Decipher	2023-01-15	18:00:00	1	100	100	220.00	Movie	1
2	Concert: Acoustic Vibes	2023-02-10	20:00:00	2	300	290	125.00	Concert	2
3	Soccer Match: City Rivals	2023-03-25	19:30:00	3	200	180	152.00	Sports	3
4	Movie Night: The Great Gatsby	2023-04-10	21:00:00	4	120	80	185.00	Movie	4
5	Concert: Pop Explosion	2023-05-01	17:45:00	5	250	200	330.00	Concert	5
6	Live Music: Jazz Evening	2023-06-12	19:00:00	6	300	280	140.00	Concert	6
7	Basketball Game: Finals	2023-07-08	18:30:00	7	300	300	120.00	Sports	7
8	Movie Night: PuddingFest	2023-08-15	20:15:00	8	150	100	175.00	Movie	8

Action Output: \* Time: Action: Message: Duration / Fetch: 10 rows returned 0.000 sec / 0.000 sec

Event 2

Contact Help | Support | Documentation

4.

The screenshot shows the MySQL Workbench interface with a query editor and results pane. The query is:

```
50
51 */*4*/ SELECT * FROM Event
52 WHERE event_name LIKE '%cup%';
53
```

The results pane displays the structure of the 'Event' table and the data for events where the name contains 'cup'. The output shows two rows:

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
1	Movie Night: Inception	2023-01-15	18:00:00	1	150	120	220.00	Movie	1
2	Concert: Acoustic Vibes	2023-02-20	20:00:00	2	300	250	125.00	Concert	2

Below the results, the 'Output' tab shows the execution log with three entries corresponding to the queries above.

5.

The screenshot shows the MySQL Workbench interface with a query editor and results pane. The query is:

```
53
54 */*5*/ SELECT * FROM Event
55 WHERE ticket_price BETWEEN 1000 AND 2500;
56
```

The results pane displays the structure of the 'Event' table and the data for events with ticket prices between 1000 and 2500. The output shows eight rows:

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
1	Movie Night: Inception	2023-01-15	18:00:00	1	150	120	220.00	Movie	1
2	Concert: Acoustic Vibes	2023-02-20	20:00:00	2	300	250	125.00	Concert	2
3	Soccer Match: City Rivals	2023-03-25	19:00:00	3	200	180	150.00	Sports	3
4	Movie Night: The Great Gatsby	2023-04-01	21:00:00	4	120	80	180.00	Movie	4
5	Live Music: Jazz Evening	2023-04-12	19:00:00	5	300	280	140.00	Concert	5
6	Basketball Game: Finals	2023-05-08	18:00:00	6	250	200	130.00	Sports	6
7	Movie Night: Casablanca	2023-06-10	20:00:00	7	150	120	220.00	Movie	7
8	Movie Night: Casablanca	2023-06-10	20:00:00	8	150	120	220.00	Movie	8

Below the results, the 'Output' tab shows the execution log with three entries corresponding to the queries above.

6.

```
56
57- /*6*/ SELECT * FROM Event
58 WHERE event_date BETWEEN '2023-05-01' AND '2023-06-30';
59
```

Result Grid		Filter Rows		Edit		Report/Export		Wrap Cell Context	
event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
5	Concert: Rock Explosion	2023-05-09	17:45:00	1	250	250	300.00	Concert	5
6	Live Music: Jazz Evening	2023-06-12	19:00:00	6	300	280	144.00	Concert	6

Event 2 -				Apply	Revert	Comment	Help	Shutdown
Details								
Action Output								
•	Time	Action		Message			Duration / Result	
1	15:46:30	SELECT * FROM Event WHERE available_millis > 0 LIMIT 0, 1000		10 row(s) returned.			0.000 sec / 0.000 sec	
2	15:46:55	SELECT ? FROM Event WHERE event_name LIKE 'Sup%' LIMIT 0, 1000		0 row(s) returned.			0.000 sec / 0.000 sec	
3	15:47:10	SELECT ? FROM Event WHERE ticket_price BETWEEN 1000 AND 2500 LIMIT 0, 1000		0 row(s) returned.			0.000 sec / 0.000 sec	
4	15:47:45	SELECT ? FROM Event WHERE event_date BETWEEN '2023-05-01' AND '2023-06-30' LIMIT 0, 1000		2 row(s) returned.			0.000 sec / 0.000 sec	

7.

```
59
60+ /*7*/ SELECT * FROM Event
61 WHERE available_seats > 0 AND event_type = 'Concert';
62
```

Event ID	Event Name	Event Date		Venue ID	Total Seats	Available Seats	Ticket Price	Event Type	Booking Status
		Start Date	End Date						
1	Concert: Acoustic Vibes	2023-02-20	20:00:00	2	300	230	1225.00	Concert	Open
5	Concert: Rock Explosion	2023-03-05	19:00:00	5	250	200	3330.00	Concert	Open
6	Live Music Jazz Evening	2023-06-12	19:00:00	6	300	280	1400.00	Concert	Open
10	Concert: Rock Revolution	2023-10-20	22:00:00	10	250	230	3310.00	Concert	Open
11	Virtual Reality Experience	2023-07-15	18:00:00	11	150	120	1000.00	Virtual	Open
12	Art Show: Abstract Expressions	2023-09-10	17:00:00	12	200	180	800.00	Art Show	Open

Action Output	Time	Action	Message	Duration / Result
1 15:46:11 SELECT * FROM Event WHERE available_units > 0 AND event_type = 'Cancel' LIMIT 0,1000			4 rows(s) returned.	0.000 sec / 0.000 sec

8.

The screenshot shows the MySQL Workbench interface with the following details:

- SQL Editor:** The code entered is:

```
63
64 */*8*/ SELECT * FROM Customer LIMIT 5 OFFSET 5;
65
66
```
- Result Grid:** A table titled "Customer" is displayed with columns: customer\_id, customer\_name, email, phone\_number, booking\_id. The data rows are:

customer_id	customer_name	email	phone_number	booking_id
6	Emma White	emma.w@email.com	555-1234	6
7	Michael Davis	michael.d@email.com	555-6799	7
8	Olivia Taylor	olivia.t@email.com	555-1234	8
9	Daniel Wilson	daniel.w@email.com	555-9879	9
10	Sophia Adams	sophia.a@email.com	555-8612	10
- Logs:** The log shows two actions:

Action	Time	Message	Duration / Fetch	
1	15:48:11	SELECT * FROM Event WHERE available_seats > 0 AND event_type = 'Concert' LIMIT 0, 1000	4 rows  returned	0.000 sec / 0.000 sec
2	15:48:38	SELECT * FROM Customer LIMIT 5 OFFSET 5	5 rows  returned	0.000 sec / 0.000 sec

9.

The screenshot shows the MySQL Workbench interface with the following details:

- SQL Editor:** The code entered is:

```
67
68 */*9*/ SELECT * FROM Booking WHERE num_tickets > 4;
69
70
```
- Result Grid:** A table titled "Booking" is displayed with columns: booking\_id, customer\_id, event\_id, num\_tickets, total\_cost, booking\_date. The data row is:

booking_id	customer_id	event_id	num_tickets	total_cost	booking_date
7	7	7	5	M\$50.00	2023-07-08
- Logs:** The log shows three actions:

Action	Time	Message	Duration / Fetch	
1	15:48:11	SELECT * FROM Event WHERE available_seats > 0 AND event_type = 'Concert' LIMIT 0, 1000	4 rows  returned	0.000 sec / 0.000 sec
2	15:48:38	SELECT * FROM Customer LIMIT 5 OFFSET 5	5 rows  returned	0.000 sec / 0.000 sec
3	15:49:01	SELECT * FROM Booking WHERE num_tickets > 4 LIMIT 0, 1000	1 rows  returned	0.000 sec / 0.000 sec

## 10.

The screenshot shows the MySQL Workbench interface. In the top-left pane, there is a code editor with the following SQL query:

```
70
71 */*10*/ SELECT * FROM Customer
72 WHERE phone_number LIKE '%000';
73
```

In the bottom-right pane, there is a results grid titled "Customer 17" with the following columns: customer\_id, customer\_name, email, phone\_number, booking\_id. The grid is currently empty, showing only the header row.

Below the results grid, the "Action Output" section displays the following information:

#	Time	Action	Message	Duration / Fetch
1	15:49:32	SELECT * FROM Customer WHERE phone_number LIKE '%000' LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

## 11.

The screenshot shows the MySQL Workbench interface. In the top-left pane, there is a code editor with the following SQL query:

```
73
74 */*11*/ SELECT * FROM Event
75 WHERE total_seats > 15000 ORDER BY total_seats;
76
```

In the bottom-right pane, there is a results grid titled "Event 17" with the following columns: event\_id, event\_name, event\_date, event\_time, venue\_id, total\_seats, available\_seats, total\_price, event\_type, booking\_id. The grid is currently empty, showing only the header row.

Below the results grid, the "Action Output" section displays the following information:

#	Time	Action	Message	Duration / Fetch
1	15:50:05	SELECT * FROM Event WHERE total_seats > 15000 ORDER BY total_seats LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

## 12.

The screenshot shows the MySQL Workbench interface with a query editor and a results grid. The query is:

```
76
77 */*12*/ SELECT * FROM Event
78 WHERE NOT (event_name LIKE 'x%' OR
79 event_name LIKE 'y%' OR event_name LIKE 'z%');
```

The results grid displays 10 rows of event data:

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
1	Movie Night: Inception	2023-01-21	18:00:00	1	100	100	220.00	Movie	1
2	Concert: Acoustic Vibes	2023-02-01	20:00:00	2	200	200	125.00	Concert	2
3	Soccer Match: City Rivals	2023-03-20	19:30:00	3	200	180	150.00	Sports	3
4	Movie Night: The Great Gatsby	2023-04-10	21:00:00	4	100	80	180.00	Movie	4
5	Concert: Pop Explosion	2023-05-05	17:45:00	5	250	200	230.00	Concert	5
6	Live Music: Jazz Evening	2023-06-12	19:00:00	6	300	280	140.00	Concert	6
7	Basketball Game: Finals	2023-07-08	18:30:00	7	300	300	130.00	Sports	7
8	Musical Performance: Puccini's Dream	2023-08-15	20:15:00	8	400	400	300.00	Concert	8

The status bar at the bottom right indicates a duration of 0.000 sec / 0.000 sec.

## TASK3:

1.

The screenshot shows the MySQL Workbench interface with a query editor and results grid. The query is:

```
181  /*-----Task 3-----*/
182  /*1*/ SELECT event_id, event_name, AVG(ticket_price) AS average_ticket_price
183  FROM Event
184  GROUP BY event_id, event_name;
185
```

The results grid displays the following data:

event_id	event_name	average_ticket_price
1	Movie Night: Inception	2220.000000
2	Concert: Acoustic Week	1215.000000
3	Soccer Match: City Rivals	1125.000000
4	Movie Night: The Green Gables	1553.000000
5	Concert: Pop Explosion	2330.000000
6	Live Music: Jazz Evening	1440.000000
7	Basketball Game: Finals	1130.000000
8	Movie Night: Thrillers	1111.000000

The output pane shows the query and its execution details:

```
1 15:53:02 SELECT event_id, event_name, AVG(ticket_price) AS average_ticket_price  FROM Event  GROUP BY e... 10 rows selected
```

Duration / Fetch: 0.000 sec / 0.000 sec

2.

The screenshot shows the MySQL Workbench interface with a query editor and results grid. The query is:

```
185
186
187  /*2*/ SELECT SUM(Booking.num_tickets * Event.ticket_price) AS total_revenue
188  FROM Booking
189  JOIN Event ON Booking.event_id = Event.event_id;
```

The results grid displays the following data:

total_revenue
48120.00

The output pane shows the query and its execution details:

```
1 15:55:14 SELECT SUM(Booking.num_tickets * Event.ticket_price) AS total_revenue  FROM Booking  JOIN Event ... 1 row selected
```

Duration / Fetch: 0.000 sec / 0.000 sec

3.

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

```
192 * /*3*/ SELECT e.event_id, e.event_name, SUM(b.num_tickets) AS total_ticket_sales
193     FROM Booking b
194     JOIN Event e ON b.event_id = e.event_id
195 GROUP BY e.event_id, e.event_name
196 ORDER BY total_ticket_sales DESC
197 LIMIT 1;
```

The results grid shows one row of data:

event_id	event_name	total_ticket_sales
7	Basketball Game: Finals	3

The status bar at the bottom indicates the duration of the query execution.

4.

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

```
199
200 * /*4*/ SELECT e.event_id, e.event_name, SUM(b.num_tickets) AS total_tickets_sold
201     FROM Booking b
202     JOIN Event e ON b.event_id = e.event_id
203 GROUP BY e.event_id, e.event_name;
204
```

The results grid shows the data from the first query:

event_id	event_name	total_tickets_sold
1	Movie Night: Inception	2
2	Concert: Acoustic Vibes	3
3	Soccer Match: City Rivals	1
4	Movie Night: The Great Gatsby	4
5	Concert: Pop Explosion	2
6	Live Music: Jazz Evening	3
7	Basketball Game: Finals	5
8	Movie Night: Casablanca	1
9	Concert: Rock Revolution!	7

The status bar at the bottom indicates the duration of the query execution.

## 5.

The screenshot shows the MySQL Workbench interface with a query editor and results pane. The query is:

```
284
285 * /*5*/ SELECT e.event_id, e.event_name
286     FROM Event e
287     LEFT JOIN Booking b ON e.event_id = b.event_id
288 WHERE b.event_id IS NULL;
289
```

The results pane shows a table with two columns: event\_id and event\_name. The table contains one row with event\_id 1 and event\_name 'Football Match'.

Below the results is the 'Detail' tab showing the 'Admin Output' of the query execution:

Time	Action	Message	Duration / Fetch
1 15:57:24	SELECT e.event_id, e.event_name, SUM(b.num_tickets) AS total_tickets_sold ... FROM Booking b ... JOIN E... 1 rows(s) returned.		0.282 sec / 0.000 sec
2 15:58:12	SELECT e.event_id, e.event_name, SUM(b.num_tickets) AS total_tickets_sold ... FROM Booking b ... JOIN E... 10 rows(s) returned.		0.000 sec / 0.000 sec
3 15:58:45	SELECT e.event_id, e.event_name ... FROM Event e ... LEFT JOIN Booking b ON event_id = b.event_id ... 2 rows(s) returned.		0.000 sec / 0.000 sec

## 6.

The screenshot shows the MySQL Workbench interface with a query editor and results pane. The query is:

```
211     FROM Booking b
212     JOIN Customer c ON b.customer_id = c.customer_id
213     GROUP BY c.customer_id, c.customer_name
214     ORDER BY total_tickets_booked DESC
215     LIMIT 1;
216
```

The results pane shows a table with three columns: customer\_id, customer\_name, and total\_tickets\_booked. The table contains one row with customer\_id 1 and customer\_name 'Michael Davis'.

Below the results is the 'Detail' tab showing the 'Admin Output' of the query execution:

Time	Action	Message	Duration / Fetch
1 15:59:11	SELECT c.customer_id, c.customer_name, SUM(b.num_tickets) AS total_tickets_booked ... FROM Booking b ... JOIN C... 1 rows(s) returned.		0.000 sec / 0.000 sec

7.

The screenshot shows the MySQL Workbench interface with the following details:

SQL Editor Content:

```
216
217 • /*7*/ SELECT MONTH(b.booking_date) AS month, e.event_id, e.event_name, SUM(b.num_tickets) /
218     FROM Booking b
219     JOIN Event e ON b.event_id = e.event_id
220     GROUP BY MONTH(b.booking_date), e.event_id, e.event_name;
221
```

Result Grid:

month	event_id	event_name	total_tickets_sold
1	1	Movie Night: Inception	2
2	2	Concert: Acoustic Vibes	3
3	3	Soccer Match: City Rivalry	1
4	4	Movie Night: The Great Gatsby	4
5	5	Concert: Pop Explosion	2
6	6	Live Music: Jazz Evening	3
7	7	Basketball Game: Finals	5
8	8	Movie Night: Casablanca	1
9	9	Concert: Metallica International Month	7

Action Output:

- 1 15:59:11 SELECT c.customer\_id, c.customer\_name, SUM(b.num\_tickets) AS total\_tickets\_booked FROM Booking b ... 1 rows(s) returned Duration / Fetch: 0.000 sec / 0.000 sec
- 2 15:59:40 SELECT MONTH(b.booking\_date) AS month, e.event\_id, e.event\_name, SUM(b.num\_tickets) AS total\_tickets\_... 10 rows(s) returned Duration / Fetch: 0.000 sec / 0.000 sec

8.

The screenshot shows the MySQL Workbench interface with the following details:

SQL Editor Content:

```
221
222 • /*8*/ SELECT e.venue_id, AVG(e.ticket_price) AS average_ticket_price
223     FROM Event e
224     GROUP BY e.venue_id;
225
226
```

Result Grid:

venue_id	average_ticket_price
1	1220.000000
2	1250.000000
3	1525.000000
4	1555.000000
5	1330.000000
6	1440.000000
7	1330.000000
8	2220.000000
9	1714.000000

Action Output:

- 1 15:59:11 SELECT c.customer\_id, c.customer\_name, SUM(b.num\_tickets) AS total\_tickets\_booked FROM Booking b ... 1 rows(s) returned Duration / Fetch: 0.000 sec / 0.000 sec
- 2 15:59:40 SELECT MONTH(b.booking\_date) AS month, e.event\_id, e.event\_name, SUM(b.num\_tickets) AS total\_tickets\_... 10 rows(s) returned Duration / Fetch: 0.000 sec / 0.000 sec
- 3 16:00:01 SELECT e.venue\_id, AVG(ticket\_price) AS average\_ticket\_price FROM Event e GROUP BY e.venue\_id ... 10 row(s) returned Duration / Fetch: 0.018 sec / 0.000 sec

## 9.

The screenshot shows the MySQL Workbench interface with the following details:

SQL Editor content:

```
226
227 * /*9*/ SELECT e.event_type, SUM(b.num_tickets) AS total_tickets_sold
228   FROM Event e
229     JOIN Booking b ON e.event_id = b.event_id
230   GROUP BY e.event_type;
231
```

Result Grid:

event_type	total_tickets_sold
Movie	7
Concert	11
Sports	8

Action Output:

#	Time	Action	Message	Duration / Fetch
1	15:59:11	SELECT c.customer_id, c.customer_name, SUM(b.num_tickets) AS total_tickets_booked FROM Booking b	1 rows(1 returned)	0.000 sec / 0.000 sec
2	15:59:40	SELECT MONTH(b.booking_date) AS month, e.event_id, e.event_name, SUM(b.num_tickets) AS total_tickets_...	10 rows(10 returned)	0.000 sec / 0.000 sec
3	16:00:01	SELECT e.venue_id, AVG(b.ticket_price) AS average_ticket_price FROM Event e GROUP BY e.venue_id	10 rows(10 returned)	0.018 sec / 0.000 sec
4	16:00:37	SELECT e.event_name, COUNT(b.num_tickets) AS total_tickets_sold FROM Event e JOIN Booking b ON e.event_id = b.event_id	1 rows(1 returned)	0.001 sec / 0.000 sec

## 10.

The screenshot shows the MySQL Workbench interface with the following details:

SQL Editor content:

```
232
233 * /*10*/ SELECT YEAR(b.booking_date) AS year, SUM(b.num_tickets * e.ticket_price) AS total_revenue
234   FROM Booking b
235     JOIN Event e ON b.event_id = e.event_id
236   GROUP BY YEAR(b.booking_date);
237
```

Result Grid:

year	total_revenue
2023	49120.00

Action Output:

#	Time	Action	Message	Duration / Fetch
1	16:00:50	SELECT YEAR(b.booking_date) AS year, SUM(b.num_tickets * e.ticket_price) AS total_revenue FROM Booking b JOIN Event e ON b.event_id = e.event_id GROUP BY YEAR(b.booking_date);	1 rows(1 returned)	0.000 sec / 0.000 sec

## 11.

The screenshot shows a MySQL Workbench interface with a query editor and a results grid. The query is:

```
239 * /*11*/ SELECT c.customer_id, c.customer_name
240     FROM Booking b
241     JOIN Customer c ON b.customer_id = c.customer_id
242     GROUP BY c.customer_id, c.customer_name
243     HAVING COUNT(DISTINCT b.event_id) > 1;
244
```

The results grid shows two columns: customer\_id and customer\_name. The output section shows two rows of execution logs:

Time	Action	Message	Duration / Hash
1 16:00:55	SELECT YEAR(b.booking_date) AS year, SUM(b.num_tickets * e.ticket_price) AS total_revenue FROM Booking b JOIN Customer c ON b.customer_id = c.customer_id GROUP BY c.customer_id, c.customer_name;	1 row(s) returned	0.000 sec / 0.000 sec
2 16:01:18	SELECT c.customer_id, c.customer_name FROM Booking b JOIN Customer c ON b.customer_id = c.customer_id GROUP BY c.customer_id, c.customer_name;	0 row(s) returned	0.000 sec / 0.000 sec

## 12.

The screenshot shows a MySQL Workbench interface with a query editor and a results grid. The query is:

```
245 * /*12*/ SELECT c.customer_id, c.customer_name, SUM(b.num_tickets * e.ticket_price) AS total_revenue
246     FROM Booking b
247     JOIN Customer c ON b.customer_id = c.customer_id
248     JOIN Event e ON b.event_id = e.event_id
249     GROUP BY c.customer_id, c.customer_name;
250
```

The results grid shows three columns: customer\_id, customer\_name, and total\_revenue. The output section shows one row of execution logs:

Time	Action	Message	Duration / Hash
1 16:01:50	SELECT c.customer_id, c.customer_name, SUM(b.num_tickets * e.ticket_price) AS total_revenue FROM Booking b JOIN Customer c ON b.customer_id = c.customer_id JOIN Event e ON b.event_id = e.event_id GROUP BY c.customer_id, c.customer_name;	10 row(s) returned	0.000 sec / 0.000 sec

13.

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
251
252 * /*13*/ SELECT e.venue_id, e.event_type, AVG(e.ticket_price) AS average_ticket_price
253     FROM Event e
254     GROUP BY e.venue_id, e.event_type;
255
256
```

The results grid displays the following data:

venue_id	event_type	average_ticket_price
1	Movie	2220.00000
2	Concert	1235.00000
3	Sports	1523.00000
4	Movie	1553.00000
5	Concert	3300.00000
6	Concert	1840.00000
7	Sports	1330.00000
8	Movie	2220.00000
9	Movie	1774.00000

The output pane shows the executed query and its result:

```
1 16:02:20 SELECT e.venue_id, e.event_type, AVG(e.ticket_price) AS average_ticket_price    FROM Event e   GROUP BY e.venue_id, e.event_type; 10 rows returned
```

14.

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
257 * /*14*/ SELECT c.customer_id, c.customer_name, SUM(b.num_tickets)
258     AS total_tickets_purchased
259     FROM Booking b
260     JOIN Customer c ON b.customer_id = c.customer_id
261     WHERE b.booking_date >= CURDATE() - INTERVAL 30 DAY
262     GROUP BY c.customer_id, c.customer_name;
```

The results grid displays the following data:

customer_id	customer_name	total_tickets_purchased

The output pane shows the executed query and its result:

```
1 16:03:33 SELECT c.customer_id, c.customer_name, SUM(b.num_tickets) AS total_tickets_purchased    FROM Booking b JOIN Customer c ON b.customer_id = c.customer_id WHERE b.booking_date >= CURDATE() - INTERVAL 30 DAY GROUP BY c.customer_id, c.customer_name; 8 rows returned
```

## Task4:

### 1.

The screenshot shows the MySQL Workbench interface with a query editor and results pane. The query is:

```
265  /*-----task 4-----*/
266  /*1*/ SELECT venue_id, AVG(ticket_price) AS average_ticket_price
267  FROM Event
268  WHERE venue_id IN (SELECT DISTINCT venue_id FROM Event)
269  GROUP BY venue_id;
270
271
```

The results grid shows the following data:

venue_id	average_ticket_price
1	2220.000000
2	1235.000000
3	1525.000000
4	1533.000000
5	1330.000000
6	1440.000000
7	1330.000000
8	2220.000000
9	1225.000000
10	1111.000000

The status bar at the bottom indicates the query took 0.000 sec / 0.000 sec.

### 2.

The screenshot shows the MySQL Workbench interface with a query editor and results pane. The query is:

```
271
272  /*2*/ SELECT event_id, event_name
273  FROM Event
274  WHERE event_id IN (SELECT event_id FROM Booking
275  GROUP BY event_id HAVING SUM(num_tickets) > 0.5 * total_seats);
276
277
```

The results grid shows the following data:

event_id	event_name
200	200

The status bar at the bottom indicates the query took 0.281 sec / 0.000 sec.

3.

The screenshot shows the MySQL Workbench interface with a query editor and results grid. The query is:

```
277
278 * /*3*/ SELECT e.event_id, e.event_name, SUM(b.num_tickets) AS total_tickets_sold
279   FROM Event e
280   JOIN Booking b ON e.event_id = b.event_id
281 GROUP BY e.event_id, e.event_name;
282
283
```

The results grid displays the following data:

event_id	event_name	total_tickets_sold
1	Movie Night: Inception	2
2	Concert: Atlantic Vibes	3
3	Soccer Match: City Rivals	1
4	Movie Night: The Great Gatsby	4
5	Concert: Pop Explosion	2
6	Live Music: Jazz Evening	3
7	Basketball Games: Finals	5
8	Movie Night: Casablanca	1
9	Soccer Match: International Clash	3
10	Concert: Rock Revolution	5

The output pane shows the executed SQL statement and its duration:

```
1 16:08:27 SELECT e.event_id, e.event_name, SUM(b.num_tickets) AS total_tickets_sold
   FROM Event e
   JOIN Booking b ON e.event_id = b.event_id
 GROUP BY e.event_id, e.event_name;
```

Duration / Fetch: 0.000 sec / 0.000 sec

4.

The screenshot shows the MySQL Workbench interface with a query editor and results grid. The first query is:

```
284
285 * /*4*/ SELECT customer_id, customer_name
286   FROM Customer
287 WHERE NOT EXISTS (SELECT 1 FROM Booking WHERE
288 Customer.customer_id = Booking.customer_id);
289
290
```

The results grid displays the following data:

customer_id	customer_name
1000	John Doe

The output pane shows the executed SQL statements and their durations:

```
1 16:08:27 SELECT e.event_id, e.event_name, SUM(b.num_tickets) AS total_tickets_sold
   FROM Event e
   JOIN Booking b ON e.event_id = b.event_id
 GROUP BY e.event_id, e.event_name;
2 16:08:56 SELECT customer_id, customer_name
   FROM Customer
 WHERE NOT EXISTS (SELECT 1 FROM Booking WHERE
 Customer.customer_id = Booking.customer_id);
```

Duration / Fetch: 0.000 sec / 0.000 sec; 0.019 sec / 0.000 sec

5.

The screenshot shows the MySQL Workbench interface with the following details:

- SQL Editor:** Contains the following SQL code:

```
298
299 * /*5*/ SELECT event_id, event_name
300      FROM Event
301     WHERE event_id NOT IN (SELECT event_id FROM Booking);
302
303
```
- Result Grid:** Displays the results of the query, showing two columns: event\_id and event\_name. The data is:

event_id	event_name
1	Movie
2	Concert
3	Sports
4	Movie
5	Concert
6	Sports
7	Movie
8	Sports
9	Concert
- Action Output:** Shows the execution log with three entries:

#	Time	Action	Message	Duration / Fetch
1	10:08:27	SELECT event_id, event_name, SUM(num_tickets) AS total_tickets_sold...	FROM Event n JOIN Booking b ON n.event_id = b.event_id WHERE NOT EXISTS (SELECT 1 FROM Booking b WHERE b.event_id = n.event_id) 10 row(s) returned	0.000 sec / 0.000 sec
2	10:08:56	SELECT customer_id, customer_name...	FROM Customer c WHERE NOT EXISTS (SELECT 1 FROM Booking b WHERE b.customer_id = c.customer_id) 3 row(s) returned	0.015 sec / 0.000 sec
3	10:09:22	SELECT event_id, event_name...	FROM Event n WHERE event_id NOT IN (SELECT event_id FROM Booking b) 5 row(s) returned	0.000 sec / 0.000 sec

6.

The screenshot shows the MySQL Workbench interface with the following details:

- SQL Editor:** Contains the following SQL code:

```
296
297 * /*6*/ SELECT event_type, (SELECT SUM(num_tickets)
298      FROM Booking WHERE Booking.event_id = Event.event_id) AS total_tickets_sold
299
300
301
302
```
- Result Grid:** Displays the results of the query, showing two columns: event\_type and total\_tickets\_sold. The data is:

event_type	total_tickets_sold
Movie	2
Concert	3
Sports	1
Movie	4
Concert	2
Concert	3
Sports	1
Movie	1
Sports	2
Concert	9
- Action Output:** Shows the execution log with one entry:

#	Time	Action	Message	Duration / Fetch
1	10:09:58	SELECT event_type, (SELECT SUM(num_tickets)...	FROM Booking WHERE Booking.event_id = Event.event_id... 10 row(s) returned	0.000 sec / 0.000 sec

7.

The screenshot shows the MySQL Workbench interface with a query editor and a results grid. The query retrieves event IDs, names, and ticket prices from the Event table where the ticket price is greater than the average ticket price. The results grid displays four rows of data.

```
384 * /*7*/ SELECT event_id, event_name, ticket_price
385   FROM Event
386 WHERE ticket_price > (SELECT AVG(ticket_price) FROM Event);
387
388
389
```

event_id	event_name	ticket_price
1	Movie Night: Inception	220.00
2	Concert: Pop Explosion	330.00
3	Movie Night: Excellence	220.00
10	Concert: Rock Revolution	330.00

Event 16: x

Action Output

#	Time	Action	Message	Duration / Fetch
1	16:10:33	SELECT event_id, event_name, ticket_price FROM Event WHERE ticket_price > (SELECT AVG(ticket_...	4 rows returned	0.000 sec / 0.000 sec

8.

The screenshot shows the MySQL Workbench interface with a query editor and a results grid. The query joins the Booking and Event tables to calculate the total revenue for each customer by summing the product of the number of tickets and the ticket price. The results grid displays ten rows of data.

```
309
310 * /*8*/ SELECT customer_id, customer_name, (SELECT SUM(b.num_tickets * e.ticket_price)
311   FROM Booking b
312   JOIN Event e ON b.event_id = e.event_id
313   WHERE b.customer_id = Customer.customer_id) AS total_revenue
314   FROM Customer;
315
```

customer_id	customer_name	total_revenue
1	John Doe	440.00
2	Jane Smith	370.00
3	Robert Johnson	1525.00
4	Samantha Brown	625.00
5	Chris Miller	6660.00
6	Emma White	4320.00
7	Michael Davis	6650.00
8	Olivia Taylor	2220.00
9	Daniel Wilson	2450.00
10	Victoria Adams	6875.00

Event 16: x

Action Output

#	Time	Action	Message	Duration / Fetch
1	16:11:32	SELECT customer_id, customer_name, (SELECT SUM(b.num_tickets * e.ticket_price) ... FROM Booking b ... 10 rows returned		0.000 sec / 0.000 sec

9.

The screenshot shows the MySQL Workbench interface with a query editor and a results grid. The query is:

```
317
318 * /*9*/ SELECT DISTINCT c.customer_id, c.customer_name
319   FROM Customer c
320 WHERE EXISTS (SELECT 1 FROM Booking b
321   WHERE b.customer_id = c.customer_id AND b.event_id IN
322     (SELECT event_id FROM Event WHERE venue_id = 1));
323
```

The results grid displays:

customer_id	customer_name
1	John Doe
2	Jane Smith

The output pane shows the execution details:

Action	Time	Message	Duration / Fetch
1	16:14:37	SELECT DISTINCT c.customer_id, c.customer_name FROM Customer c WHERE EXISTS (SELECT 1 FROM Booking b WHERE b.customer_id = c.customer_id AND b.event_id IN (SELECT event_id FROM Event WHERE venue_id = 1)); 1 rows(s) returned.	0.031 sec / 0.000 sec

10.

The screenshot shows the MySQL Workbench interface with a query editor and a results grid. The query is:

```
323
324 * /*10*/ SELECT e.event_type, SUM(b.num_tickets) AS total_tickets_sold
325   FROM Event e
326 JOIN Booking b ON e.event_id = b.event_id
327 GROUP BY e.event_type;
328
329
```

The results grid displays:

event_type	total_tickets_sold
Movie	7
Concert	11
Sports	8

The output pane shows the execution details:

Action	Time	Message	Duration / Fetch
1	16:16:44	SELECT e.event_type, SUM(b.num_tickets) AS total_tickets_sold FROM Event e JOIN Booking b ON e.event_id = b.event_id GROUP BY e.event_type; 3 rows(s) returned.	0.000 sec / 0.000 sec

## 11.

The screenshot shows the MySQL Workbench interface with a query editor and results pane. The query selected is:

```
332
333 * /*11*/ SELECT DISTINCT c.customer_id, c.customer_name, MONTH(b.booking_date) AS month
334     FROM Customer c
335     JOIN Booking b ON c.customer_id = b.customer_id;
336
337
338
```

The results grid displays the following data:

customer_id	customer_name	month
1	John Doe	1
2	Jane Smith	2
3	Robert Johnson	3
4	Samantha Brown	4
5	Chris Miller	5
6	Emma White	6
7	Michael Davis	7
8	Olivia Taylor	8
9	Daniel Wilson	9
10	Brooke Adams	10

The details pane shows the execution log:

Time	Action	Message	Duration / Fetch
16:17:15	SELECT DISTINCT c.customer_id, c.customer_name, MONTH(b.booking_date) AS month FROM Customer c JOIN Booking b ON c.customer_id = b.customer_id;	10 rows returned	0.000 sec / 0.000 sec

## 12.

The screenshot shows the MySQL Workbench interface with a query editor and results pane. The query selected is:

```
337
338
339 * /*12*/ SELECT venue_id, AVG(ticket_price) AS average_ticket_price
340     FROM Event
341     WHERE venue_id IN (SELECT DISTINCT venue_id FROM Event)
342     GROUP BY venue_id;
343
```

The results grid displays the following data:

venue_id	average_ticket_price
1	2222.000000
2	1222.000000
3	1522.000000
4	1355.000000
5	1320.000000
6	1440.000000
7	1320.000000
8	2222.000000
9	1222.000000
10	1222.000000

The details pane shows the execution log:

Time	Action	Message	Duration / Fetch
16:17:31	SELECT venue_id, AVG(ticket_price) AS average_ticket_price FROM Event WHERE venue_id IN (SELECT DISTINCT venue_id FROM Event) GROUP BY venue_id;	10 rows returned	0.000 sec / 0.000 sec