

Assignment 5

Ticket Booking System

Task1:

```
Assignment5.sql
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 * ALTER TABLE Booking
51 ADD (FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
52     FOREIGN KEY (event_id) REFERENCES Event(event_id));
53
54 * ALTER TABLE Event
55 ADD (FOREIGN KEY (venue_id) REFERENCES Venu(venue_id),
56     FOREIGN KEY (booking_id) REFERENCES Booking(booking_id));
57
58 * ALTER TABLE Event
59 ADD CHECK (event_type IN ('Theatre', 'Sports', 'Concerts'));
```

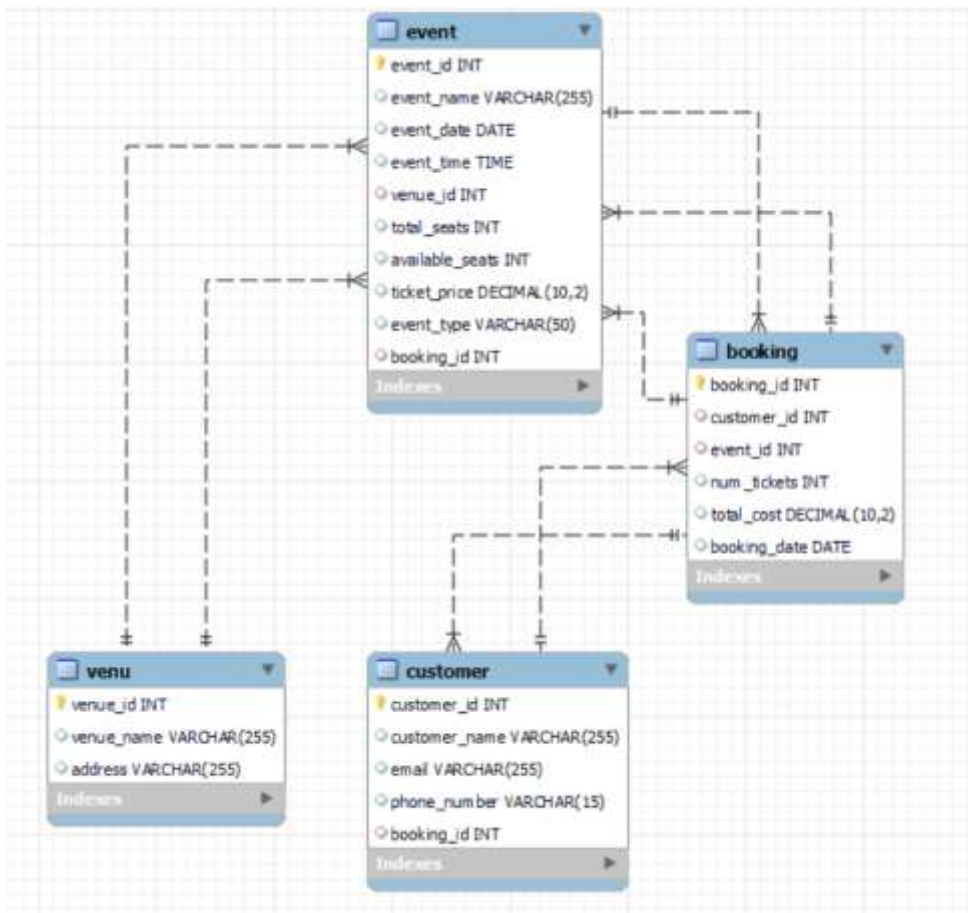
```

50
51 * ALTER TABLE Event
52 ADD (FOREIGN KEY (venue_id) REFERENCES Venue(venue_id),
53 FOREIGN KEY (booking_id) REFERENCES Booking(booking_id));
54
55 * ALTER TABLE Event
56 ADD CHECK (event_type IN ('Movie', 'Sports', 'Concert'));
57

```

Output			
Action Output			
#	Time	Action	Message
1	15:31:55	CREATE DATABASE TicketBookingSystem	1 row(s) affected
2	15:31:57	USE TicketBookingSystem	0 row(s) affected
3	15:31:59	CREATE TABLE Venue (venue_id INT PRIMARY KEY, venue_name VARCHAR(255), address VARCHAR(255))	0 row(s) affected
4	15:32:03	CREATE TABLE Event (event_id INT PRIMARY KEY, event_name VARCHAR(255), event_date DATE, event_time TIME, venue_id INT, total_seats INT, available_seats INT, ticket_price DECIMAL(10,2), event_type VARCHAR(50), booking_id INT)	0 row(s) affected
5	15:32:05	CREATE TABLE Booking (booking_id INT PRIMARY KEY, customer_id INT, event_id INT, num_tickets INT, total_cost DECIMAL(10,2), booking_date DATE)	0 row(s) affected
6	15:32:08	CREATE TABLE Customer (customer_id INT PRIMARY KEY, customer_name VARCHAR(255), email VARCHAR(255), phone_number VARCHAR(15), booking_id INT)	0 row(s) affected
7	15:32:18	ALTER TABLE Customer ADD FOREIGN KEY (booking_id) REFERENCES Booking(booking_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
8	15:32:20	ALTER TABLE Booking ADD FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
9	15:32:23	ALTER TABLE Event ADD FOREIGN KEY (venue_id) REFERENCES Venue(venue_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
10	15:32:40	ALTER TABLE Event ADD CHECK (event_type IN ('Movie', 'Sports', 'Concert'))	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

ERD



TASK 2:

1.

```
Assignment 1
SQL Assistant
My Snippets

66 /* ----- task2 ----- */
67 * INSERT INTO Venue (venue_id, venue_name, address) VALUES
68 (1, 'Grand Theater', '123 Main Street, Cityville'),
69 (2, 'City Arena', '456 Center Avenue, Townsville'),
70 (3, 'Sports Stadium', '789 Stadium Road, SportsTown'),
71 (4, 'Pile Palace', '101 Movie Lane, CinemaVille'),
72 (5, 'Concert Hall', '202 Melody Street, Harmonyton'),
73 (6, 'Community Center', '303 Social Square, Gatherburg'),
74 (7, 'Live Lounge', '404 Entertainment Avenue, Showville'),
75 (8, 'Cinematic Complex', '505 Film Street, Filletown'),
76 (9, 'Soccer Park', '606 Goal Street, Kicksville'),
77 (10, 'Music Dome', '707 Harmony Road, Concertburg')
78
79
80 * INSERT INTO Event (event_id, event_name, event_date, event_time, venue_id, total_seats, available_seats, ticket_price, event_type, booking_id)
81 (1, 'Music Night: Acoustic Vibes', '2023-01-15', '18:00:00', 1, 150, 120, 2200.00, 'Music', NULL),
82 (2, 'Concert: Acoustic Vibes', '2023-02-20', '20:00:00', 2, 300, 250, 1235.00, 'Concert', NULL),
83 (3, 'Soccer Match: City Rivals', '2023-03-25', '19:30:00', 3, 200, 180, 1525.00, 'Sports', NULL),
84 (4, 'Movie Night: The Great Gatsby', '2023-04-10', '21:00:00', 4, 120, 90, 5555.00, 'Movie', NULL),
85 (5, 'Concert: Pop Explosion', '2023-05-05', '17:45:00', 5, 250, 200, 3330.00, 'Concert', NULL),
86 (6, 'Live Music: Jazz Fusion', '2023-06-12', '19:00:00', 6, 100, 80, 1440.00, 'Concert', NULL),
87 (7, 'Basketball Game: Finals', '2023-07-08', '18:30:00', 7, 220, 190, 1730.00, 'Sports', NULL),
88 (8, 'Movie Night: Classic Drama', '2023-08-20', '20:15:00', 8, 150, 120, 2220.00, 'Movie', NULL),
89 (9, 'Soccer Match: International Clash', '2023-09-15', '19:45:00', 9, 280, 230, 1225.00, 'Sports', NULL),
90 (10, 'Concert: Rock Revolution', '2023-10-30', '22:00:00', 10, 200, 180, 3330.00, 'Concert', NULL)
91
```

```
Assignment 1
SQL Assistant
My Snippets

92 * INSERT INTO Customer (customer_id, customer_name, email, phone_number, booking_id) VALUES
93 (1, 'John Doe', 'john.doe@gmail.com', '555-1234', NULL),
94 (2, 'Jane Smith', 'jane.smith@gmail.com', '555-5678', NULL),
95 (3, 'Robert Johnson', 'robert.j@gmail.com', '555-9012', NULL),
96 (4, 'Samantha Brown', 'samantha.b@gmail.com', '555-3456', NULL),
97 (5, 'Chris Miller', 'chris.m@gmail.com', '555-7890', NULL),
98 (6, 'Anna White', 'anna.w@gmail.com', '555-2345', NULL),
99 (7, 'Michael Davis', 'michael.d@gmail.com', '555-6789', NULL),
100 (8, 'Olivia Taylor', 'olivia.t@gmail.com', '555-1234', NULL),
101 (9, 'Daniel Wilson', 'daniel.w@gmail.com', '555-5678', NULL),
102 (10, 'Sophia Adams', 'sophia.a@gmail.com', '555-9012', NULL)
103
104
105 * INSERT INTO Booking (booking_id, customer_id, event_id, num_tickets, total_cost, booking_date) VALUES
106 (1, 1, 1, 2, 4400.00, '2023-01-10'),
107 (2, 2, 2, 3, 3705.00, '2023-01-20'),
108 (3, 3, 3, 1, 1525.00, '2023-03-25'),
109 (4, 4, 4, 4, 6220.00, '2023-04-10'),
110 (5, 5, 5, 2, 6660.00, '2023-05-05'),
111 (6, 6, 6, 1, 4320.00, '2023-06-12'),
112 (7, 7, 7, 3, 5190.00, '2023-07-08'),
113 (8, 8, 8, 1, 2220.00, '2023-08-20'),
114 (9, 9, 9, 2, 2450.00, '2023-09-15'),
115 (10, 10, 10, 3, 9990.00, '2023-10-30')
116
117
118 * UPDATE Event SET booking_id = 1 WHERE event_id = 1;
119 * UPDATE Event SET booking_id = 2 WHERE event_id = 2;
120 * UPDATE Event SET booking_id = 3 WHERE event_id = 3;
121 * UPDATE Event SET booking_id = 4 WHERE event_id = 4;
122 * UPDATE Event SET booking_id = 5 WHERE event_id = 5;
123 * UPDATE Event SET booking_id = 6 WHERE event_id = 6;
124 * UPDATE Event SET booking_id = 7 WHERE event_id = 7;
125 * UPDATE Event SET booking_id = 8 WHERE event_id = 8;
126 * UPDATE Event SET booking_id = 9 WHERE event_id = 9;
127 * UPDATE Event SET booking_id = 10 WHERE event_id = 10;
128
129
130 * UPDATE Customer SET booking_id = 1 WHERE customer_id = 1;
131 * UPDATE Customer SET booking_id = 2 WHERE customer_id = 2;
132 * UPDATE Customer SET booking_id = 3 WHERE customer_id = 3;
133 * UPDATE Customer SET booking_id = 4 WHERE customer_id = 4;
134 * UPDATE Customer SET booking_id = 5 WHERE customer_id = 5;
135 * UPDATE Customer SET booking_id = 6 WHERE customer_id = 6;
136 * UPDATE Customer SET booking_id = 7 WHERE customer_id = 7;
137 * UPDATE Customer SET booking_id = 8 WHERE customer_id = 8;
138 * UPDATE Customer SET booking_id = 9 WHERE customer_id = 9;
139 * UPDATE Customer SET booking_id = 10 WHERE customer_id = 10;
140
```

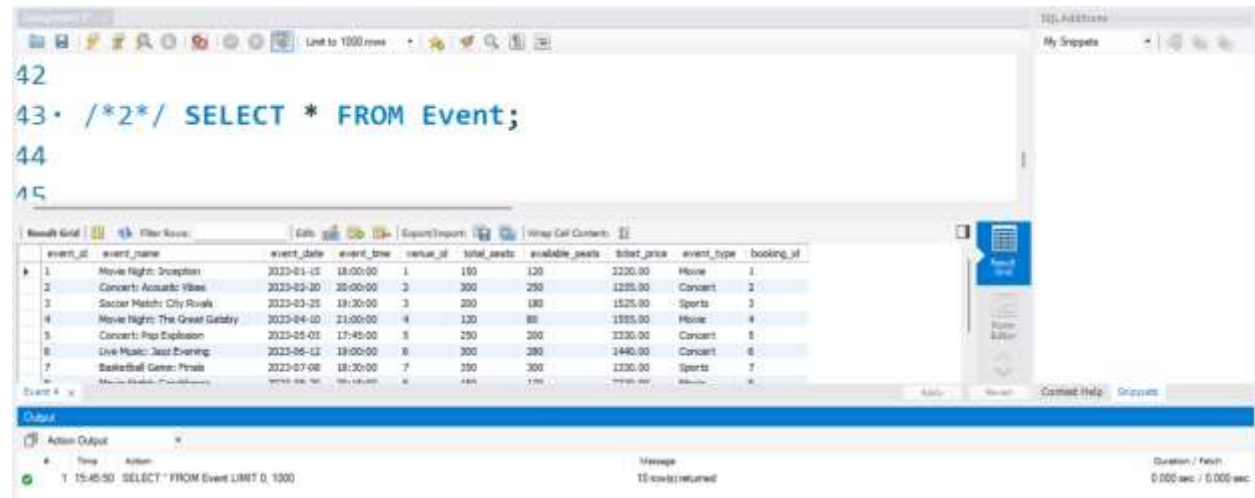
2.

42

43 • /*2*/ SELECT * FROM Event;

44

45



event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
1	Movie Night: Inception	2023-03-15	18:00:00	1	150	120	2200.00	Movie	1
2	Concert: Acoustic Vibes	2023-03-20	20:00:00	2	300	250	1235.00	Concert	2
3	Soccer Match: City Rivals	2023-03-25	19:30:00	3	200	180	1525.00	Sports	3
4	Movie Night: The Great Gatsby	2023-04-01	21:00:00	4	120	80	1555.00	Movie	4
5	Concert: Pop Explosion	2023-04-05	17:45:00	5	250	200	2330.00	Concert	5
6	Live Music: Jazz Evening	2023-06-12	19:00:00	6	300	280	1440.00	Concert	6
7	Basketball Game: Finals	2023-07-08	18:30:00	7	250	300	2330.00	Sports	7
8	Movie Night: Franchise	2023-08-15	19:15:00	8	180	150	2200.00	Movie	8

Output

Action Output

1 15:45:50 SELECT * FROM Event LIMIT 0, 1000

Message

10 rows returned

Duration / Fetch

0.000 sec / 0.000 sec

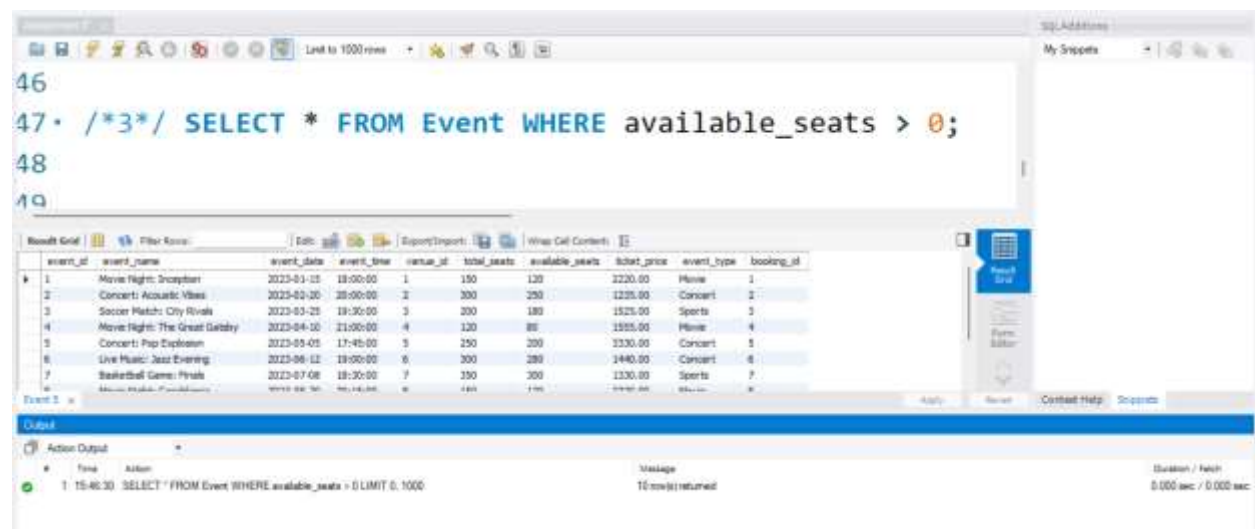
3.

46

47 • /*3*/ SELECT * FROM Event WHERE available_seats > 0;

48

49



event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
1	Movie Night: Inception	2023-03-15	18:00:00	1	150	120	2200.00	Movie	1
2	Concert: Acoustic Vibes	2023-03-20	20:00:00	2	300	250	1235.00	Concert	2
3	Soccer Match: City Rivals	2023-03-25	19:30:00	3	200	180	1525.00	Sports	3
4	Movie Night: The Great Gatsby	2023-04-01	21:00:00	4	120	80	1555.00	Movie	4
5	Concert: Pop Explosion	2023-04-05	17:45:00	5	250	200	2330.00	Concert	5
6	Live Music: Jazz Evening	2023-06-12	19:00:00	6	300	280	1440.00	Concert	6
7	Basketball Game: Finals	2023-07-08	18:30:00	7	250	300	2330.00	Sports	7
8	Movie Night: Franchise	2023-08-15	19:15:00	8	180	150	2200.00	Movie	8

Output

Action Output

1 15:46:30 SELECT * FROM Event WHERE available_seats > 0 LIMIT 0, 1000

Message

10 rows returned

Duration / Fetch

0.000 sec / 0.000 sec

4.

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

```

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

```

The Results grid shows the following columns: event_id, event_name, event_date, event_time, venue_id, total_seats, available_seats, ticket_price, event_type, booking_id. The Results grid is currently empty.

The Output pane shows the following messages:

```

1 15:46:30 SELECT * FROM Event WHERE available_seats > 0 LIMIT 0, 1000 10 rows returned 0.000 sec / 0.000 sec
2 15:46:55 SELECT * FROM Event WHERE event_name LIKE '%cup%' LIMIT 0, 1000 0 rows returned 0.000 sec / 0.000 sec

```

5.

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

```

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

```

The Results grid shows the following columns: event_id, event_name, event_date, event_time, venue_id, total_seats, available_seats, ticket_price, event_type, booking_id. The Results grid contains the following 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	2023-03-15	18:00:00	1	150	120	2200.00	Movie	1
2	Concert: Adele's 30	2023-03-20	20:00:00	2	300	250	1200.00	Concert	2
3	Soccer Match: City vs Arsenal	2023-03-25	19:30:00	3	200	180	1525.00	Sports	3
4	Movie Night: The Great Gatsby	2023-04-01	21:00:00	4	120	80	1550.00	Movie	4
5	Live Music: Jazz Evening	2023-04-12	19:00:00	5	300	280	1440.00	Concert	5
6	Basketball Game: Lakers vs Celtics	2023-03-08	18:30:00	7	350	300	1300.00	Sports	6
7	Movie Night: Casablanca	2023-03-20	20:15:00	8	150	120	2200.00	Movie	7
8	Concert: Beyoncé's Renaissance Tour	2023-05-10	19:00:00	9	400	350	1750.00	Concert	8

The Output pane shows the following messages:

```

1 15:46:30 SELECT * FROM Event WHERE available_seats > 0 LIMIT 0, 1000 10 rows returned 0.000 sec / 0.000 sec
2 15:46:55 SELECT * FROM Event WHERE event_name LIKE '%cup%' LIMIT 0, 1000 0 rows returned 0.000 sec / 0.000 sec
3 15:47:18 SELECT * FROM Event WHERE ticket_price BETWEEN 1000 AND 2500 LIMIT 0, 1000 8 rows returned 0.000 sec / 0.000 sec

```

6.

56

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

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
5	Concert: Pop Explosion	2023-05-05	17:45:00	5	250	200	3330.00	Concert	5
6	Live Music: Jazz Evening	2023-06-12	19:00:00	6	300	280	1440.00	Concert	6

Event 6 is

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	15:46:30	SELECT * FROM Event WHERE available_seats > 0 LIMIT 0, 1000	10 row(s) returned	0.000 sec / 0.000 sec
2	15:46:35	SELECT * FROM Event WHERE event_name LIKE 'Pop%' LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
3	15:47:18	SELECT * FROM Event WHERE ticket_price BETWEEN 1000 AND 2500 LIMIT 0, 1000	8 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	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
2	Concert: Acoustic Vibes	2023-02-20	20:00:00	2	300	250	1225.00	Concert	2
5	Concert: Pop Explosion	2023-05-05	17:45:00	5	250	200	3330.00	Concert	5
6	Live Music: Jazz Evening	2023-06-12	19:00:00	6	300	280	1440.00	Concert	6
10	Concert: Rock Revolution	2023-10-30	22:00:00	10	250	200	3310.00	Concert	10

Event 7 is

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	15:48:11	SELECT * FROM Event WHERE available_seats > 0 AND event_type = 'Concert' LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec

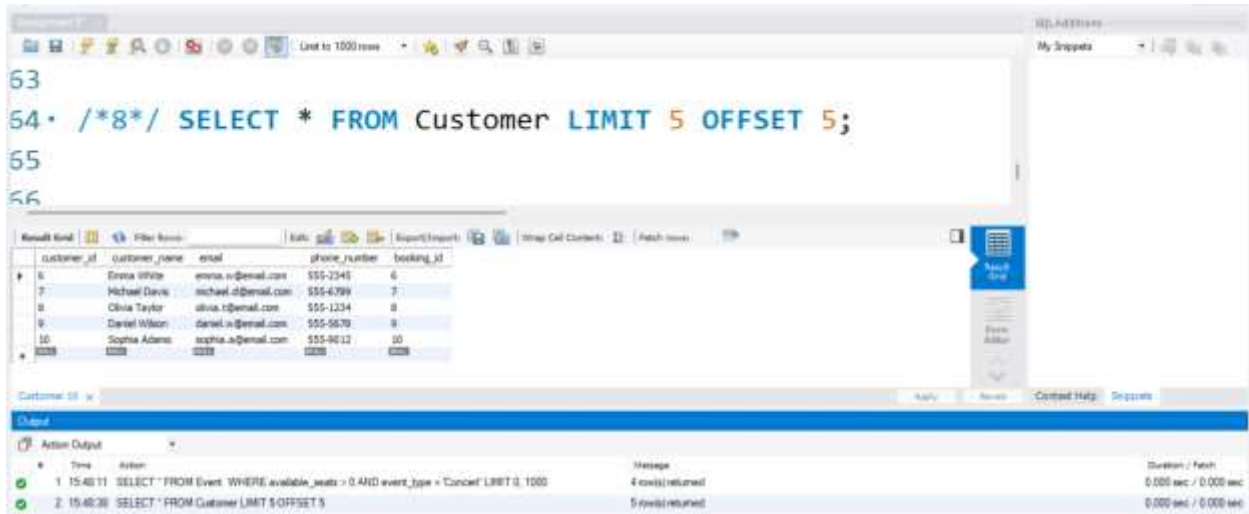
8.

63

64 • /*8*/ SELECT * FROM Customer LIMIT 5 OFFSET 5;

65

66



customer_id	customer_name	email	phone_number	booking_id
6	Emma White	emma.w@gmail.com	555-2345	6
7	Michael Davis	michael.d@gmail.com	555-6789	7
8	Olivia Taylor	olivia.t@gmail.com	555-1234	8
9	Daniel Wilson	daniel.w@gmail.com	555-5678	9
10	Sophia Adams	sophia.a@gmail.com	555-9012	10

Customer 10 x

Output

Action Output

Time	Action	Message	Duration / Fetch
1 15:48:11	SELECT * FROM Event WHERE available_seats > 0 AND event_type = 'Concert' LIMIT 5, 1000	4 row(s) returned	0.000 sec / 0.000 sec
2 15:48:36	SELECT * FROM Customer LIMIT 5 OFFSET 5	5 row(s) returned	0.000 sec / 0.000 sec

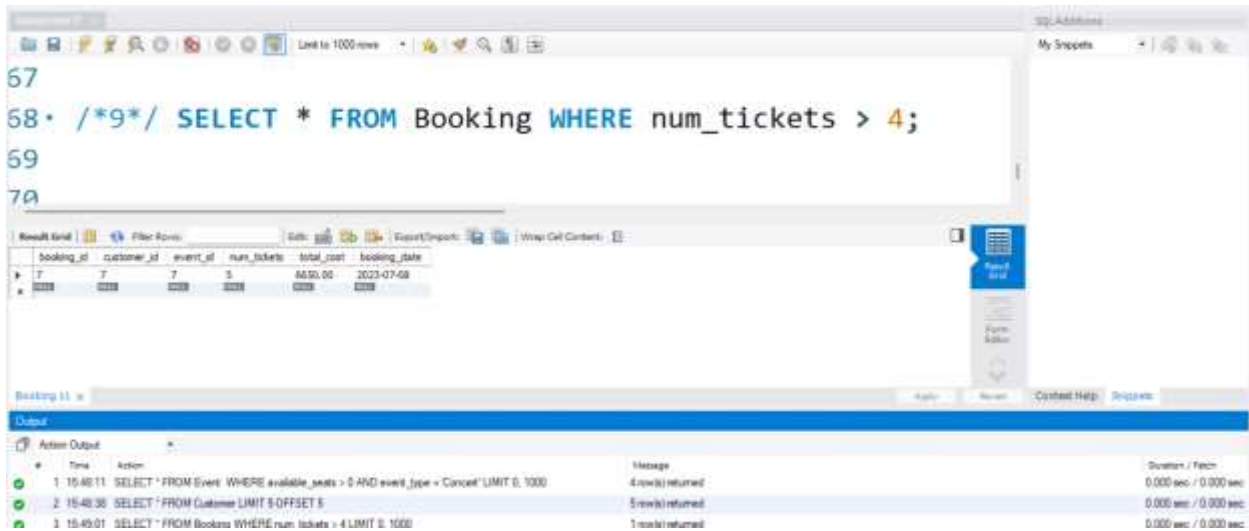
9.

67

68 • /*9*/ SELECT * FROM Booking WHERE num_tickets > 4;

69

70



booking_id	customer_id	event_id	num_tickets	total_cost	booking_date
7	7	7	5	6630.00	2023-07-08

Booking 11 x

Output

Action Output

Time	Action	Message	Duration / Fetch
1 15:48:11	SELECT * FROM Event WHERE available_seats > 0 AND event_type = 'Concert' LIMIT 5, 1000	4 row(s) returned	0.000 sec / 0.000 sec
2 15:48:36	SELECT * FROM Customer LIMIT 5 OFFSET 5	5 row(s) returned	0.000 sec / 0.000 sec
3 15:49:01	SELECT * FROM Booking WHERE num_tickets > 4 LIMIT 5, 1000	1 row(s) returned	0.000 sec / 0.000 sec

10.

The screenshot shows the SQL Anywhere 12.0.1 IDE. The main editor contains the following SQL query:

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

Below the editor, the 'Result Grid' is visible, showing a table with the following columns: customer_id, customer_name, email, phone_number, and booking_id. The first row of data is displayed.

The 'Output' pane at the bottom shows the execution log:

#	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 SQL Anywhere 12.0.1 IDE. The main editor contains the following SQL query:

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

Below the editor, the 'Result Grid' is visible, showing a table with the following columns: event_id, event_name, event_date, event_time, venue_id, total_seats, available_seats, ticket_price, event_type, and booking_id. The first row of data is displayed.

The 'Output' pane at the bottom shows the execution log:

#	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 a database IDE interface. The top pane contains a SQL query:

```
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 bottom pane displays the 'Result Grid' with the following data:

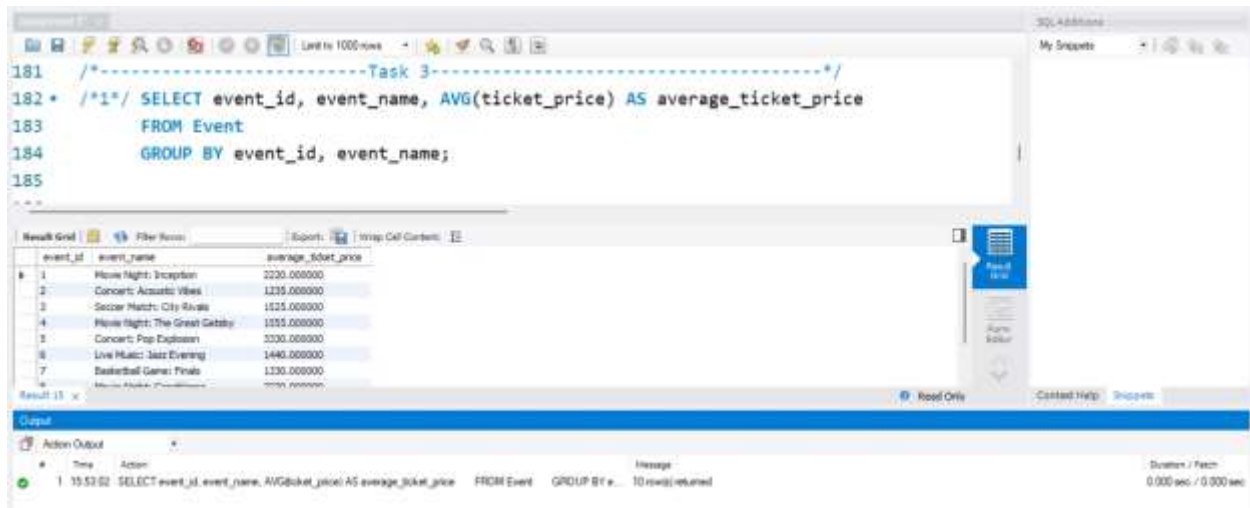
event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
1	Movie Night: Inception	2023-03-15	18:00:00	1	150	120	2200.00	Movie	1
2	Concert: Acoustic Vibes	2023-02-20	20:00:00	2	300	250	1225.00	Concert	2
3	Soccer Match: City Rivals	2023-03-25	19:30:00	3	200	180	1525.00	Sports	3
4	Movie Night: The Great Gatsby	2023-04-10	21:00:00	4	120	80	1950.00	Movie	4
5	Concert: Pop Explosion	2023-05-05	17:45:00	5	250	200	3330.00	Concert	5
6	Live Music: Jazz Evening	2023-06-12	19:00:00	6	300	280	2440.00	Concert	6
7	Basketball Game: Finals	2023-07-08	18:30:00	7	150	300	1330.00	Sports	7
8	Movie Night: Phenomenon	2023-08-15	19:15:00	8	180	150	2790.00	Movie	8

The bottom pane also shows the 'Output' section with the following message:

```
1 15:50:58 SELECT * FROM Event WHERE NOT (event_name LIKE 'x%' OR event_name LIKE 'y%' OR event_name LIKE 'z%') returned 10 row(s) in 0.000 sec / 0.000 sec
```

TASK3:

1.



```
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
```

event_id	event_name	average_ticket_price
1	Movie Night: Inception	2020.000000
2	Concert: Acoustic Vibes	1235.000000
3	Soccer Match: City Rivals	1525.000000
4	Movie Night: The Great Gatsby	1555.000000
5	Concert: Pop Explosion	2030.000000
6	Live Music: Jazz Evening	1440.000000
7	Basketball Game: Finals	1330.000000
8	Movie Night: Sci-Fi Adventure	2000.000000

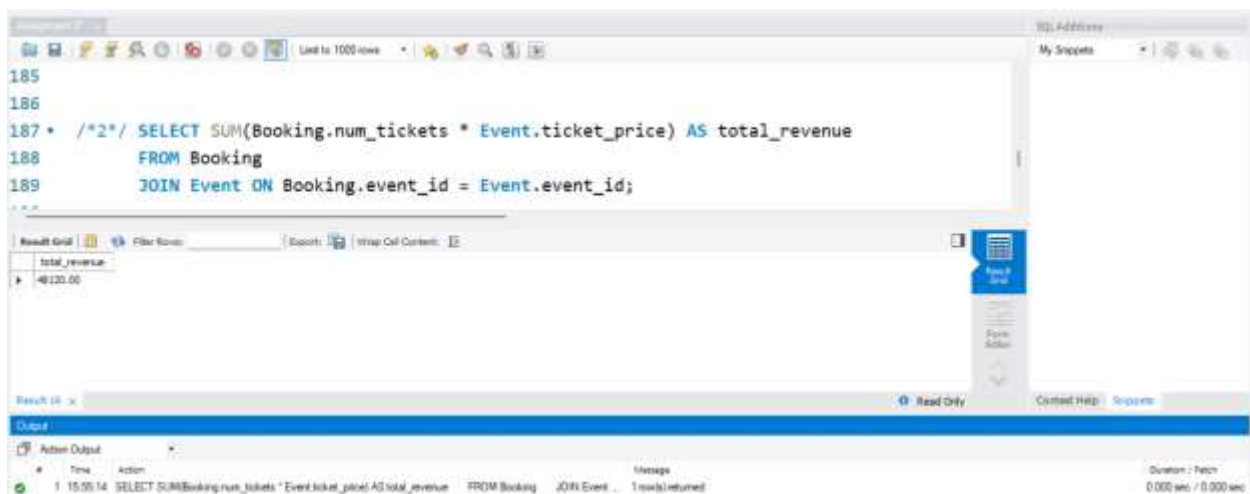
Result 13: x

Output

Action Output

Time	Action	Message	Duration / Fetch
15:52:02	SELECT event_id, event_name, AVG(ticket_price) AS average_ticket_price FROM Event GROUP BY event_id, event_name	10 row(s) returned	0.000 sec / 0.000 sec

2.



```
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;

```

total_revenue
49120.00

Result 14: x

Output

Action Output

Time	Action	Message	Duration / Fetch
15:55:14	SELECT SUM(Booking.num_tickets * Event.ticket_price) AS total_revenue FROM Booking JOIN Event ON Booking.event_id = Event.event_id	1 row(s) returned	0.000 sec / 0.000 sec

3.

The screenshot shows a SQL IDE with a query editor and a results pane. The query is as follows:

```
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 pane displays a single row:

event_id	event_name	total_ticket_sales
7	Basketball Game: Finals	5

The bottom pane shows the execution log with the following entry:

Time	Action	Message	Duration / Fetch
1 15:57:24	SELECT e.event_id, e.event_name, SUM(b.num_tickets) AS total_ticket_sales FROM Booking b JOIN E...	1 row(s) returned	0.282 sec / 0.000 sec

4.

The screenshot shows a SQL IDE with a query editor and a results pane. The query is as follows:

```
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 pane displays a list of events and their total tickets sold:

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	Soccer Match: International Clash	1

The bottom pane shows the execution log with the following entries:

Time	Action	Message	Duration / Fetch
1 15:57:24	SELECT e.event_id, e.event_name, SUM(b.num_tickets) AS total_ticket_sales FROM Booking b JOIN E...	1 row(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 row(s) returned	0.000 sec / 0.000 sec

5.

The screenshot shows a SQL IDE interface. The query editor contains the following SQL code:

```
204
205 * /*5*/ SELECT e.event_id, e.event_name
206       FROM Event e
207      LEFT JOIN Booking b ON e.event_id = b.event_id
208     WHERE b.event_id IS NULL;
209
```

The Results grid is empty, with columns 'event_id' and 'event_name'. The Action Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	15:57:26	SELECT e.event_id, e.event_name, SUM(b.num_tickets) AS total_tickets_sold FROM Booking b JOIN E...	1 row(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 row(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 e.event_id = b.event_id	0 row(s) returned	0.000 sec / 0.000 sec

6.

The screenshot shows a SQL IDE interface. The query editor contains the following SQL code:

```
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 grid shows one row:

customer_id	customer_name	total_tickets_booked
7	Michael Davis	5

The Action Output pane shows the execution log:

#	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 row(s) returned	0.000 sec / 0.000 sec

7.

The screenshot shows the SQL Anywhere 12.0.1.2 interface. The SQL editor contains the following query:

```

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

```

The Results Grid shows the following data:

month	event_id	event_name	total_tickets_sold
1	1	Horse Night: Inception	2
2	2	Concert: Acoustic Vibes	3
3	3	Soccer Match: City Rivals	1
4	4	Horse Night: The Great Escape	4
5	5	Concert: Pop Explosion	2
6	4	Live Music: Jazz Evening	3
7	7	Basketball Game: Finals	5
8	8	Horse Night: Casablanca	1
9	6	Soccer Match: International Clash	7

The Output pane shows the execution log:

#	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 row(s) 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 row(s) returned	0.000 sec / 0.000 sec

8.

The screenshot shows the SQL Anywhere 12.0.1.2 interface. The SQL editor contains the following query:

```

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

```

The Results Grid shows the following data:

venue_id	average_ticket_price
1	2220.000000
2	1238.000000
3	1525.000000
4	1555.000000
5	3220.000000
6	1440.000000
7	1330.000000
8	2220.000000
9	1775.000000

The Output pane shows the execution log:

#	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 row(s) 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 row(s) returned	0.000 sec / 0.000 sec
3	16:00:01	SELECT e.venue_id, AVG(e.ticket_price) AS average_ticket_price FROM Event e GROUP BY e.venue...	10 row(s) returned	0.018 sec / 0.000 sec

9.

The screenshot shows the SQL Sublime editor with a query to calculate the total tickets sold for each event type. The query is as follows:

```
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
```

The result grid shows the following data:

event_type	total_tickets_sold
Movie	7
Concert	11
Sports	8

The output pane shows the execution log with the following messages:

Time	Action	Message	Duration / Fetch
15:59:11	SELECT c.customer_id, c.customer_name, SUM(b.num_tickets) AS total_tickets_sold FROM Booking b	1 row(s) returned	0.000 sec / 0.000 sec
15:59:40	SELECT MONTH(b.booking_date) AS month, e.event_id, e.event_name, SUM(b.num_tickets) AS total_tickets_sold FROM Event e	10 row(s) returned	0.000 sec / 0.000 sec
16:00:01	SELECT e.venue_id, AVG(e.ticket_price) AS average_ticket_price FROM Event e GROUP BY e.venue_id	10 row(s) returned	0.016 sec / 0.000 sec
16:00:17	SELECT e.event_name, SUM(b.num_tickets) AS total_tickets_sold FROM Event e JOIN Booking b ON e.event_id = b.event_id	1 row(s) returned	0.000 sec / 0.000 sec

10.

The screenshot shows the SQL Sublime editor with a query to calculate the total revenue for each year. The query is as follows:

```
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
```

The result grid shows the following data:

year	total_revenue
2023	40120.00

The output pane shows the execution log with the following messages:

Time	Action	Message	Duration / Fetch
16:00:50	SELECT YEAR(b.booking_date) AS year, SUM(b.num_tickets * e.ticket_price) AS total_revenue FROM Booking b	1 row(s) returned	0.000 sec / 0.000 sec

11.

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

```
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 is empty, showing columns: customer_id, customer_name.

The Output pane shows the following action output:

#	Time	Action	Message	Duration / Fetch
1	16:06:55	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	1 row(s) returned	0.000 sec / 0.000 sec
2	16:07:16	SELECT c.customer_id, c.customer_name FROM Booking b JOIN Customer c ON b.customer_id = c.customer_id	0 row(s) returned	0.000 sec / 0.000 sec

12.

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

```
245 * /*12*/ SELECT c.customer_id, c.customer_name, SUM(b.num_tickets * e.ticket_price) AS total
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 displays the following data:

customer_id	customer_name	total_revenue
1	John Doe	4940.00
2	Jane Smith	3705.00
3	Robert Johnson	1525.00
4	Samantha Brown	6220.00
5	Chris Miller	9560.00
6	Emma White	4320.00
7	Michael Davis	6650.00
8	Olivia Taylor	2220.00
9	Prakash Reddy	NA

The Output pane shows the following action output:

#	Time	Action	Message	Duration / Fetch
1	16:07:50	SELECT c.customer_id, c.customer_name, SUM(b.num_tickets * e.ticket_price) AS total_revenue FROM Booking b JOIN Event e ON b.event_id = e.event_id	13 row(s) returned	0.000 sec / 0.000 sec

13.

The screenshot shows the SQL Anywhere 12.0.1.2 interface. The query editor contains the following SQL code:

```
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 Result Grid displays the following data:

venue_id	event_type	average_ticket_price
1	Movie	2220.000000
2	Concert	1235.000000
3	Sports	1525.000000
4	Movie	1555.000000
5	Concert	2030.000000
6	Concert	1440.000000
7	Sports	1330.000000
8	Movie	2220.000000
9	Movie	1775.000000

The Output pane shows the execution details:

#	Time	Action	Message	Duration / Fetch
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 row(s) returned	0.000 sec / 0.000 sec

14.

The screenshot shows the SQL Anywhere 12.0.1.2 interface. The query editor contains the following SQL code:

```
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;  
263
```

The Result Grid displays the following data:

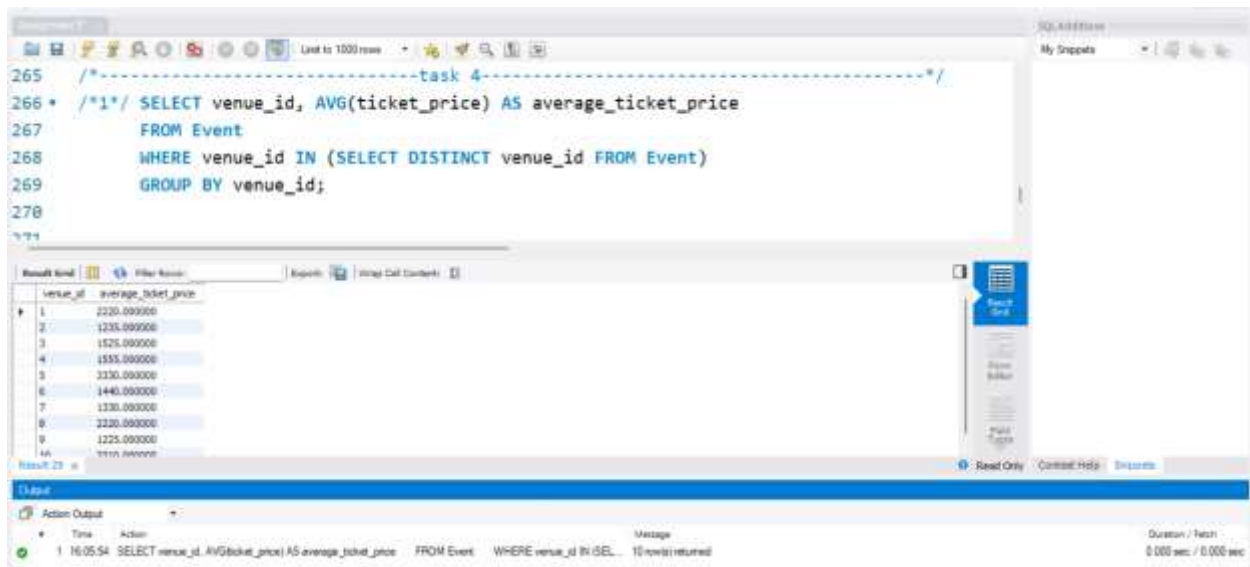
customer_id	customer_name	total_tickets_purchased
-------------	---------------	-------------------------

The Output pane shows the execution details:

#	Time	Action	Message	Duration / Fetch
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	0 row(s) returned	0.015 sec / 0.000 sec

Task4:

1.



```
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
```

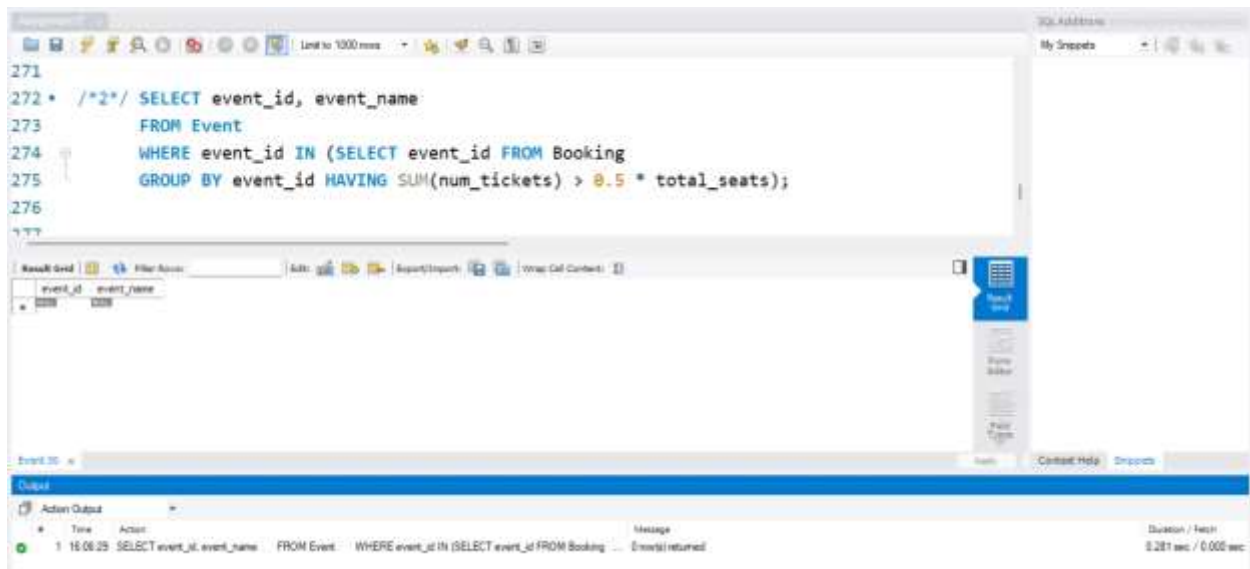
venue_id	average_ticket_price
1	2220.000000
2	1235.000000
3	1525.000000
4	1555.000000
5	2230.000000
6	1440.000000
7	1230.000000
8	2220.000000
9	1225.000000
10	1225.000000

Result: 23

Output

Time	Action	Message	Duration / Fetch
16:05:54	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

2.



```
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
```

event_id	event_name
1008	1008

Event: 20

Output

Time	Action	Message	Duration / Fetch
16:06:29	SELECT event_id, event_name FROM Event WHERE event_id IN (SELECT event_id FROM Booking GROUP BY event_id HAVING SUM(num_tickets) > 0.5 * total_seats)	1 rows returned	0.281 sec / 0.000 sec

3.

The screenshot shows the SQL Anywhere 12.0.1 interface. The SQL editor contains the following query:

```
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
```

The Results Grid displays the following data:

event_id	event_name	total_tickets_sold
1	Movie Night: Inception	2
2	Concert: Arctic Vibe	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	Soccer Match: International Clash	2
10	Concert: Rock Band Revue	5

The Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
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	10 row(s) returned	0.000 sec / 0.000 sec

4.

The screenshot shows the SQL Anywhere 12.0.1 interface. The SQL editor contains the following query:

```
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
```

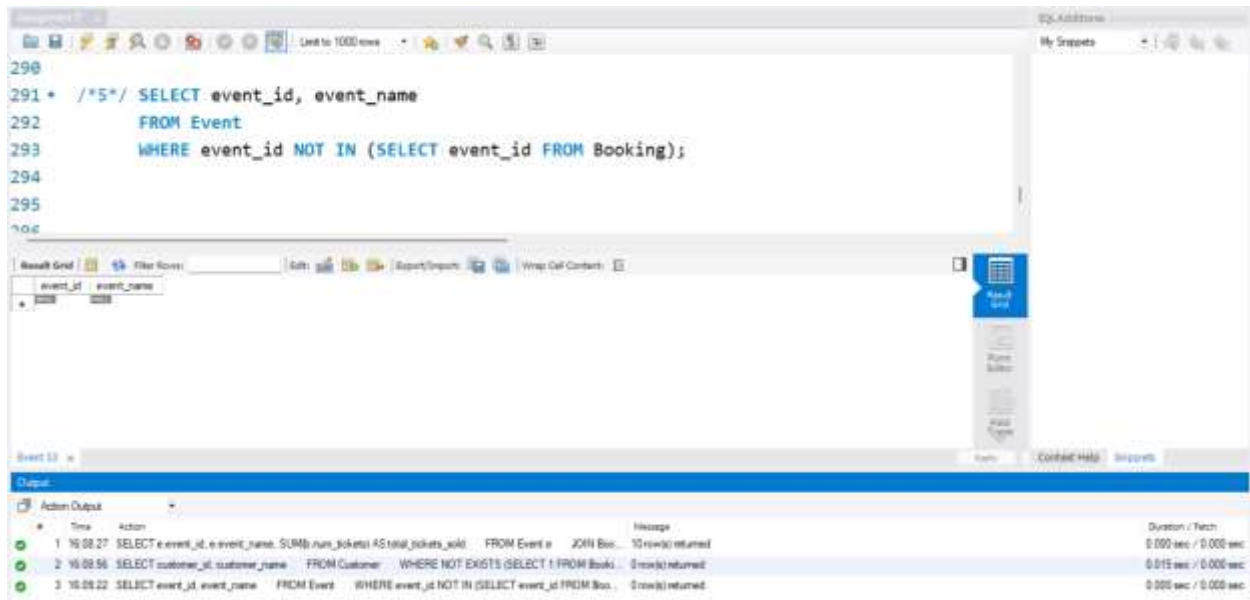
The Results Grid displays the following data:

customer_id	customer_name
1	John Doe

The Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
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	10 row(s) returned	0.000 sec / 0.000 sec
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)	0 row(s) returned	0.019 sec / 0.000 sec

5.



The screenshot shows the SQL Anywhere IDE with a query window containing the following SQL code:

```

290
291 * /*5*/ SELECT event_id, event_name
292       FROM Event
293      WHERE event_id NOT IN (SELECT event_id FROM Booking);
294
295
296

```

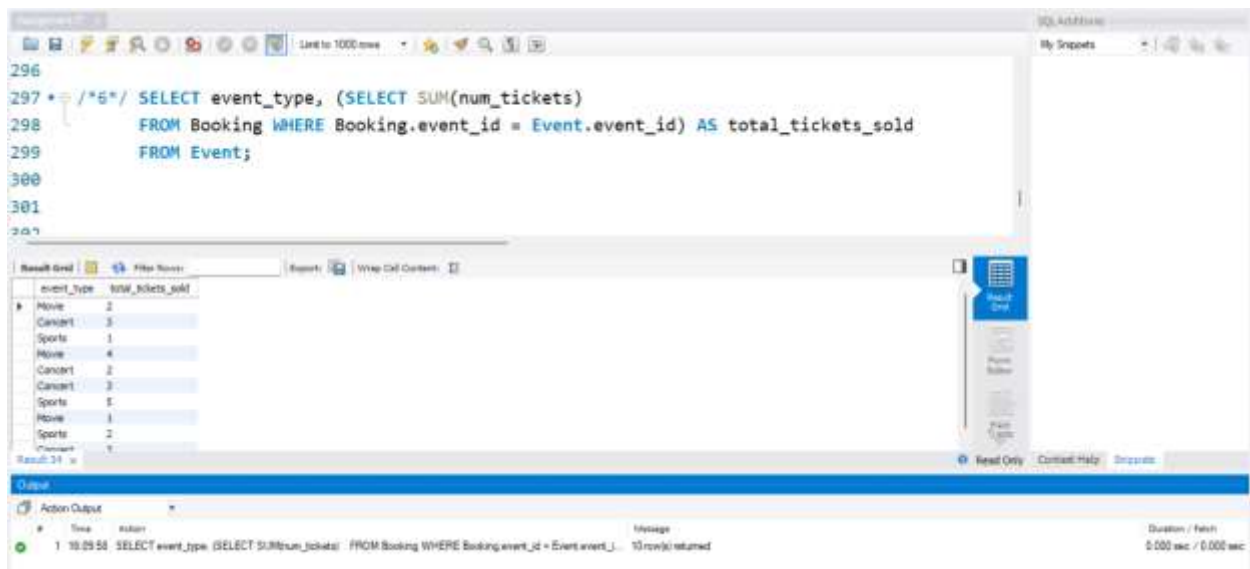
The Results Grid below the query window shows the following data:

event_id	event_name
1	Movie

The Output window at the bottom shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	10.08.27	SELECT e.event_id, e.event_name, SUM(b.num_tickets) AS total_tickets_sold FROM Event e	JOIN Boo... 10 row(s) returned	0.000 sec / 0.000 sec
2	10.08.56	SELECT customer_id, customer_name FROM Customer WHERE NOT EXISTS (SELECT 1 FROM Booking WHERE Booking.event_id = Customer.event_id)	0 row(s) returned	0.015 sec / 0.000 sec
3	10.08.22	SELECT event_id, event_name FROM Event WHERE event_id NOT IN (SELECT event_id FROM Booking)	0 row(s) returned	0.000 sec / 0.000 sec

6.



The screenshot shows the SQL Anywhere IDE with a query window containing 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       FROM Event;
300
301
302

```

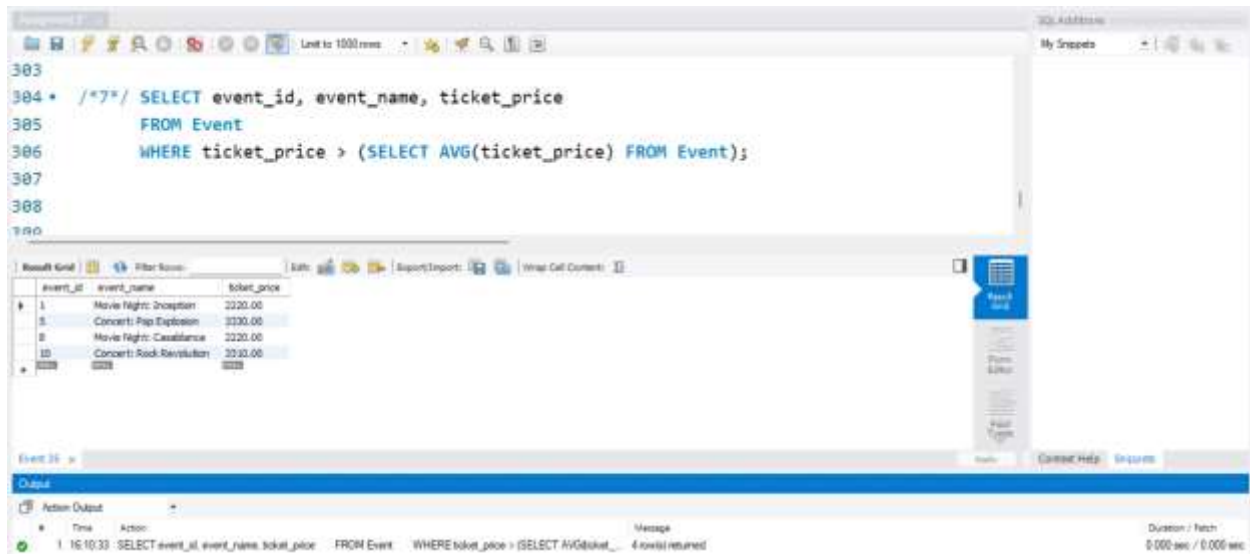
The Results Grid below the query window shows the following data:

event_type	total_tickets_sold
Movie	2
Concert	3
Sports	1
Movie	4
Concert	2
Concert	3
Sports	5
Sports	1
Movie	2
Concert	1

The Output window at the bottom shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	10.09.56	SELECT event_type, (SELECT SUM(num_tickets) FROM Booking WHERE Booking.event_id = Event.event_id) AS total_tickets_sold FROM Event	10 row(s) returned	0.000 sec / 0.000 sec

7.



The screenshot shows the SQL Server Enterprise Manager interface. The query editor at the top contains the following SQL code:

```

303
304 * /*7*/ SELECT event_id, event_name, ticket_price
305     FROM Event
306     WHERE ticket_price > (SELECT AVG(ticket_price) FROM Event);
307
308
309

```

The Results pane below the query editor displays the following data:

event_id	event_name	ticket_price
1	Movie Night: Inception	2020.00
5	Concert: Pop Explosion	2030.00
8	Movie Night: Casablanca	2020.00
10	Concert: Rock Revolution	2030.00

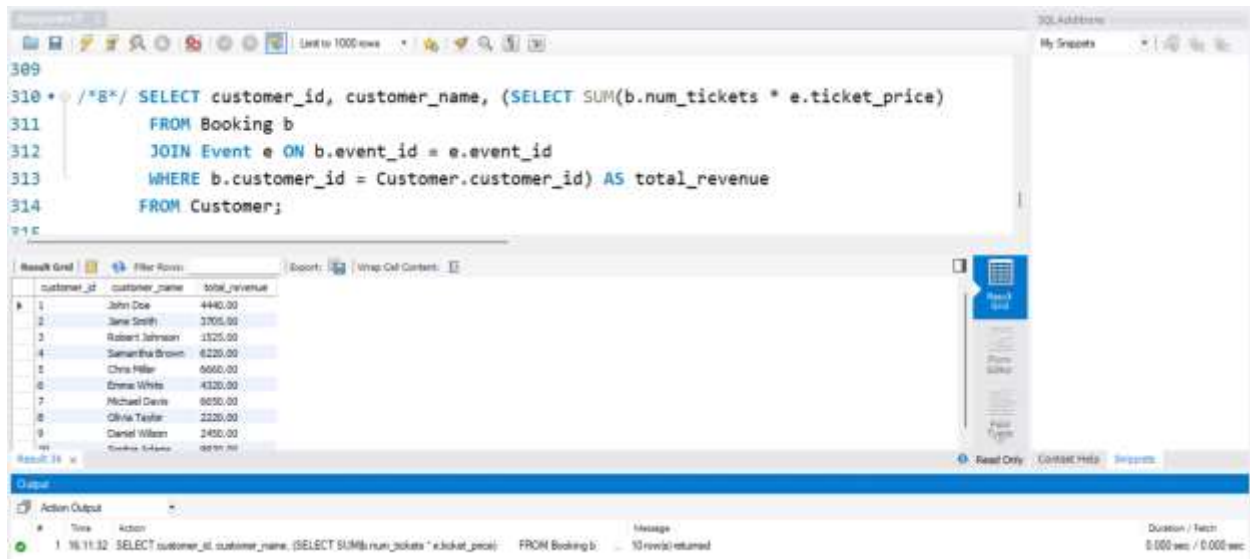
The Output pane at the bottom shows the execution details:

```

1 16:10:33 SELECT event_id, event_name, ticket_price FROM Event WHERE ticket_price > (SELECT AVG(ticket_... 4 row(s) returned
Duration / Fetch: 0.000 sec / 0.000 sec

```

8.



The screenshot shows the SQL Server Enterprise Manager interface. The query editor at the top contains the following SQL code:

```

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

```

The Results pane below the query editor displays the following data:

customer_id	customer_name	total_revenue
1	John Doe	4440.00
2	Jane Smith	3705.00
3	Robert Johnson	1525.00
4	Samantha Brown	6220.00
5	Chris Miller	6660.00
6	Elena White	4320.00
7	Michael Davis	6050.00
8	Olivia Taylor	2220.00
9	Daniel Wilson	2450.00

The Output pane at the bottom shows the execution details:

```

1 16:11:32 SELECT customer_id, customer_name, (SELECT SUM(b.num_tickets * e.ticket_price) FROM Booking b ... 10 row(s) returned
Duration / Fetch: 0.000 sec / 0.000 sec

```


9.

The screenshot shows the SQL Developer interface. The query editor contains the following SQL statement:

```
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 shows the following data:

customer_id	customer_name
1	John Doe
2228	2228

The Output pane shows the execution details:

Time	Action	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 row(s) returned	0.021 sec / 0.000 sec

10.

The screenshot shows the SQL Developer interface. The query editor contains the following SQL statement:

```
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
```

The Results Grid shows the following data:

event_type	total_tickets_sold
Movie	7
Concert	11
Sports	8

The Output pane shows the execution details:

Time	Action	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 row(s) returned	0.000 sec / 0.000 sec

11.

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

```

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	Elena White	6
7	Michael Davis	7
8	Olivia Taylor	8
9	Daniel Wilson	9
10	Sophia Adams	10

The Output pane shows the execution message: "1 16:17:15 SELECT DISTINCT c.customer_id, c.customer_name, MONTH(b.booking_date) AS month FROM Customer... 10 rows returned".

12.

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

```

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
344
345

```

The Results Grid displays the following data:

venue_id	average_ticket_price
1	2220.000000
2	1220.000000
3	1520.000000
4	1855.000000
5	3200.000000
6	1440.000000
7	1300.000000
8	2220.000000
9	1220.000000
10	1800.000000

The Output pane shows the execution message: "1 16:17:31 SELECT venue_id, AVG(ticket_price) AS average_ticket_price FROM Event WHERE venue_id IN (SE... 10 rows returned".