

Assignment

Ticket Booking System

Table name	Field	Key
Venue table	venue_id	primary
	venue_name	
	address	
Event table	event_id	primary
	event_name	
	event_date	
	event_time	
	venue_id	foreign
	total_seats	
	available_seats	
Customer table	ticket_price	
	event_type	
	customer_id	primary
	customer_name	
Booking table	email	
	phone_number	
	booking_id	primary
	customer_id	foreign
	event_id	foreign
	num_tickets	
	total_cost	
	booking_date	

Tasks 1: Database Design:

1. Create the database named "TicketBookingSystem"

The screenshot shows the MySQL Workbench interface. In the SQL editor pane, a single query is entered: `create database ticketbookingsystem;`. Below the editor, the Output pane displays the results of the query. It shows an Action Output table with one row: # 1, Time 17:34:45, Action create database ticketbookingsystem. To the right of the table, a Message area indicates "1 row(s) affected".

```
create database ticketbookingsystem;
```

#	Time	Action
1	17:34:45	create database ticketbookingsystem

Message
1 row(s) affected

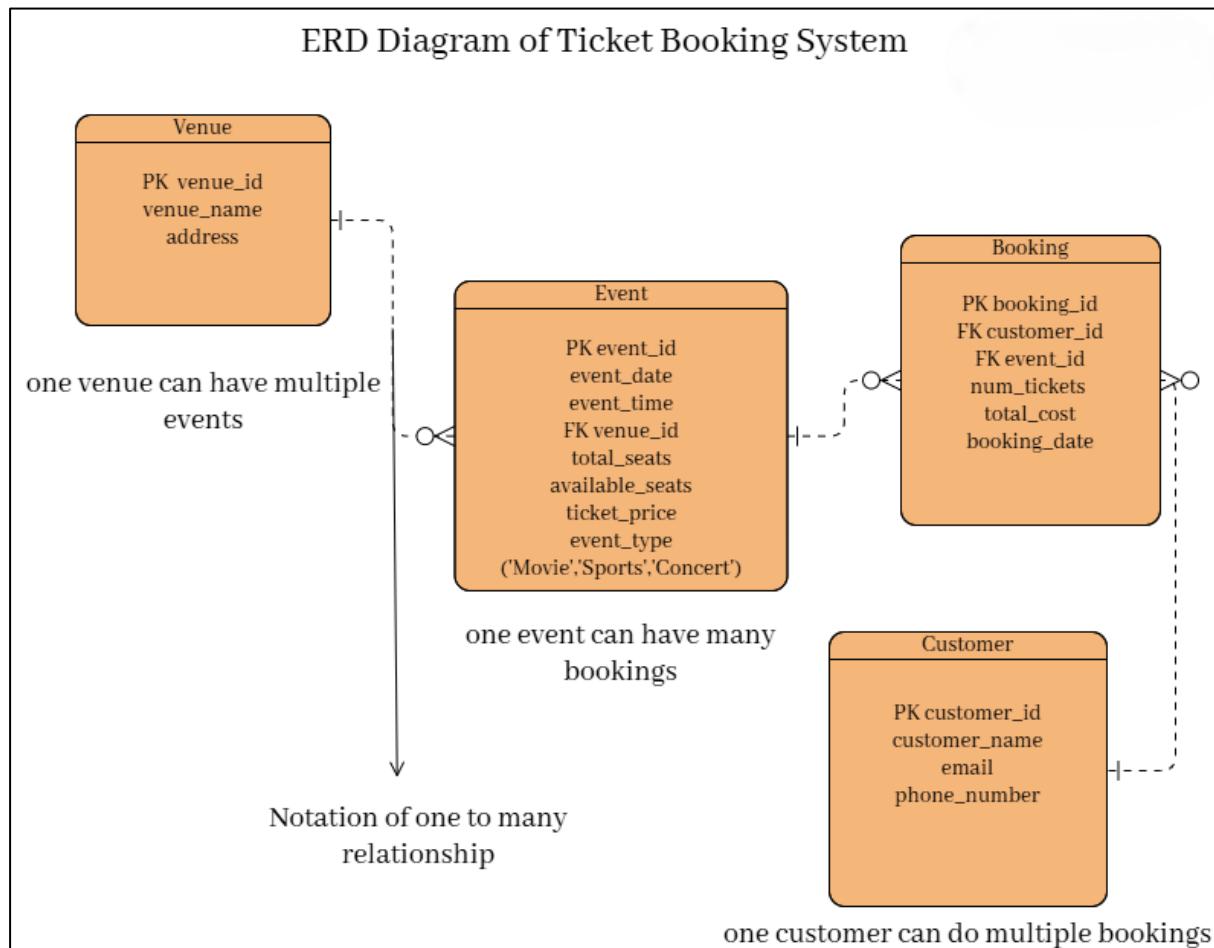
2. Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships. • Venu • Event • Customers • Booking

```

SQL File 1* ×
Limit to 1000 rows
1 • use ticketbookingsystem;
2 • create table venue(venue_id int primary key,venue_name varchar(100),address varchar(100));
3 • create table customer(customer_id int primary key,customer_name varchar(100),email varchar(100),phone_number varchar(10));
4 • create table booking(booking_id int primary key,num_tickets int,total_cost decimal(6,2),booking_date date);
5 • alter table booking add column customer_id int, add constraint foreign key(customer_id) references customer(customer_id);
6 • create table event(event_id int primary key,event_name varchar(100),event_date date,event_time time,total_seats int,available_seats int,ticket_price decimal(6,2),event_type enum('Movie','Sports','Concert'));
7 • alter table event add column venue_id int, add constraint foreign key(venue_id) references venue(venue_id);
8 • alter table booking add column event_id int, add constraint foreign key(event_id) references event(event_id);
10
Output:
Action Output
# Time Action Message Duration / Fetch
1 17:36:48 use ticketbookingsystem 0 row(s) affected 0.016 sec
2 17:36:53 create table venue(venue_id int primary key,venue_name varchar(100),address varchar(100)) 0 row(s) affected 0.016 sec
3 17:36:55 create table customer(customer_id int primary key,customer_name varchar(100),email var... 0 row(s) affected 0.016 sec
4 17:36:59 create table booking(booking_id int primary key,num_tickets int,total_cost decimal(6,2)bo... 0 row(s) affected 0.016 sec
5 17:37:01 alter table booking add column customer_id int, add constraint foreign key(customer_id)... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 0.078 sec
6 17:37:04 create table event(event_id int primary key,event_name varchar(100),event_date.eve... 0 row(s) affected 0.032 sec
7 17:37:08 alter table event add column venue_id int, add constraint foreign key(venue_id) reference... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 0.062 sec
8 17:37:14 alter table booking add column event_id int, add constraint foreign key(event_id) referenc... 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 0.078 sec

```

3. Create an ERD (Entity Relationship Diagram) for the database.



4.Create appropriate Primary Key and Foreign Key constraints for referential integrity.

The screenshot shows the MySQL Workbench interface with an SQL editor window containing the following SQL code:

```

1 •  use ticketbookingsystem;
2 •  create table venue(venue_id int primary key,venue_name varchar(100),address varchar(100));
3 •  create table customer(customer_id int primary key,customer_name varchar(100),email varchar(100),phone_number varchar(10));
4 •  create table booking(booking_id int primary key,num_tickets int,total_cost decimal(6,2),booking_date date);
5 •  alter table booking add column customer_id int, add constraint foreign key(customer_id) references customer(customer_id);
6 •  create table event(event_id int primary key,event_name varchar(100),event_date date,event_time time,total_seats int,available_seats int,
7 ticket_price decimal(6,2),event_type enum('Movie','Sports','Concert'));
8 •  alter table event add column venue_id int, add constraint foreign key(venue_id) references venue(venue_id);
9 •  alter table booking add column event_id int, add constraint foreign key(event_id) references event(event_id);
10

```

The output pane shows the results of the executed statements, including the creation of tables and the addition of columns and constraints. The log details the actions taken, including the creation of the database and each table, and the execution of the ALTER statements to add columns and set up foreign key constraints.

Tasks 2: Select, Where, Between, AND, LIKE:

1. Write a SQL query to insert at least 10 sample records into each table.

INSERT INTO venue (venue_id, venue_name, address) VALUES

(1, 'Grand Arena', '123 Main Street, Chennai'),
 (2, 'Skyline Stadium', '45 High Road, Mumbai'),
 (3, 'Harmony Hall', '67 Park Avenue, Bengaluru'),
 (4, 'Star Cinemas', '12 MG Road, Pune'),
 (5, 'City Convention Ctr', '88 Race Course, Coimbatore'),
 (6, 'Green Park Theatre', '234 Green Street, Delhi'),
 (7, 'Ocean View Grounds', '98 Marina Bay, Visakhapatnam'),
 (8, 'Metro Multiplex', '76 Nelson Road, Hyderabad'),
 (9, 'Diamond Club', '301 Ruby Street, Jaipur'),
 (10, 'Royal Event Space', '54 Crown Street, Ahmedabad');

```

mysql> select*from venue;
+-----+-----+-----+
| venue_id | venue_name | address |
+-----+-----+-----+
| 1 | Grand Arena | 123 Main Street, Chennai |
| 2 | Skyline Stadium | 45 High Road, Mumbai |
| 3 | Harmony Hall | 67 Park Avenue, Bengaluru |
| 4 | Star Cinemas | 12 MG Road, Pune |
| 5 | City Convention Ctr | 88 Race Course, Coimbatore |
| 6 | Green Park Theatre | 234 Green Street, Delhi |
| 7 | Ocean View Grounds | 98 Marina Bay, Visakhapatnam |
| 8 | Metro Multiplex | 76 Nelson Road, Hyderabad |
| 9 | Diamond Club | 301 Ruby Street, Jaipur |
| 10 | Royal Event Space | 54 Crown Street, Ahmedabad |
+-----+-----+-----+

```

```

INSERT INTO customer (customer_id, customer_name, email, phone_number) VALUES
(1, 'Rahul Sharma', 'rahul@gmail.com', '9876543210'),
(2, 'Aisha Khan', 'aisha.khan@yahoo.com', '9123456000'),
(3, 'Vinay Menon', 'vinay.menon@outlook.com', '9012345678'),
(4, 'Meera Suresh', 'meera@gmail.com', '9988776655'),
(5, 'Arjun Patel', 'arjunp@hotmail.com', '8888665544'),
(6, 'Sneha Rao', 'sneha.rao@rediff.com', '9111223344'),
(7, 'Karthik Iyer', 'karthik@xyz.com', '9345612789'),
(8, 'Riya Das', 'riya.das@gmail.com', '9321456789'),
(9, 'Pranav Joshi', 'pranavj@yahoo.com', '9456721000'),
(10, 'Tanvi Nair', 'tanvi.nair@gmail.com', '9988001122')
(11, 'Noah Kumar', 'noah.kumar@example.com', '9000000000');

```

customer_id	customer_name	email	phone_number
1	Rahul Sharma	rahul@gmail.com	9876543210
2	Aisha Khan	aisha.khan@yahoo.com	9123456000
3	Vinay Menon	vinay.menon@outlook.com	9012345678
4	Meera Suresh	meera@gmail.com	9988776655
5	Arjun Patel	arjunp@hotmail.com	8888665544
6	Sneha Rao	sneha.rao@rediff.com	9111223344
7	Karthik Iyer	karthik@xyz.com	9345612789
8	Riya Das	riya.das@gmail.com	9321456789
9	Pranav Joshi	pranavj@yahoo.com	9456721000
10	Tanvi Nair	tanvi.nair@gmail.com	9988001122
11	Noah Kumar	noah.kumar@example.com	9000000000

11 rows in set (0.00 sec)

```

INSERT INTO Event (event_id, event_name, event_date, event_time, venue_id, total_seats, available_seats, ticket_price, event_type) VALUES
(1, 'Court teaser release', '2025-11-15', '19:00:00', 1, 500, 150, 250.00, 'Movie'),
(2, 'Football Final', '2025-12-10', '20:30:00', 2, 20000, 5000, 120.50, 'Sports'),
(3, 'Coldplay Concert', '2025-11-25', '21:00:00', 3, 10000, 2000, 489.99, 'Concert'),
(4, 'The Batman', '2025-11-20', '18:30:00', 4, 300, 75, 119.99, 'Movie'),
(5, 'World cup Match', '2025-12-05', '18:00:00', 5, 70000, 10000, 150.00, 'Sports'),
(6, 'AR Rahman Concert', '2025-12-15', '19:30:00', 6, 1200, 0, 475.00, 'Concert'),
(7, 'Black Panther 2', '2025-11-18', '20:00:00', 1, 500, 200, 122.50, 'Movie'),
(8, 'U1 drugs', '2025-12-20', '20:00:00', 7, 15000, 2500, 300.00, 'Concert'),
(9, 'Yard Games Championship', '2025-12-12', '19:00:00', 2, 20000, 3000, 200.00, 'Sports'),
(10, 'Zenith Film Premiere', '2025-11-22', '17:30:00', 4, 300, 0, 150.00, 'Movie'),
(11, 'Unpopular Jazz Night', '2025-12-25', '20:00:00', 8, 200, 200, 100.00, 'Concert');

```

```

mysql> select*from event;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| event_id | event_name | event_date | event_time | total_seats | available_seats | ticket_price | event_type | venue_id |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Court teaser release | 2025-11-15 | 19:00:00 | 500 | 150 | 250.00 | Movie | 1 |
| 2 | Football Final | 2025-12-10 | 20:30:00 | 20000 | 5000 | 120.50 | Sports | 2 |
| 3 | Coldplay Concert | 2025-11-25 | 21:00:00 | 10000 | 2000 | 489.99 | Concert | 3 |
| 4 | The Batman | 2025-11-20 | 18:30:00 | 300 | 75 | 119.99 | Movie | 4 |
| 5 | World cup Match | 2025-12-05 | 18:00:00 | 70000 | 10000 | 150.00 | Sports | 5 |
| 6 | AR Rahman Concert | 2025-12-15 | 19:30:00 | 1200 | 0 | 475.00 | Concert | 6 |
| 7 | Black Panther 2 | 2025-11-18 | 20:00:00 | 500 | 200 | 122.50 | Movie | 1 |
| 8 | UI drugs | 2025-12-20 | 20:00:00 | 15000 | 2500 | 300.00 | Concert | 7 |
| 9 | Yard Games Championship | 2025-12-12 | 19:00:00 | 20000 | 3000 | 200.00 | Sports | 2 |
| 10 | Zenith Film Premiere | 2025-11-22 | 17:30:00 | 300 | 0 | 150.00 | Movie | 4 |
| 11 | Unpopular Jazz Night | 2025-12-25 | 20:00:00 | 200 | 200 | 100.00 | Concert | 8 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)

```

```

INSERT INTO Booking (booking_id, customer_id, event_id, num_tickets, total_cost, booking_date) VALUES
(1, 1, 1, 2, 500.00, '2025-11-10'), (2, 1, 3, 1, 489.99, '2025-11-12'), (3, 1, 5, 4, 600.00, '2025-11-28'),
(4, 2, 2, 4, 482.00, '2025-12-01'), (5, 2, 7, 2, 245.00, '2025-11-15'), (6, 3, 3, 1, 489.99, '2025-11-20'),
(7, 3, 8, 3, 900.00, '2025-12-05'), (8, 4, 4, 3, 359.97, '2025-11-15'), (9, 4, 6, 2, 950.00, '2025-12-10'),
(10, 5, 5, 2, 300.00, '2025-11-30'), (11, 5, 9, 1, 200.00, '2025-12-08'), (12, 6, 6, 2, 950.00, '2025-12-10'),
(13, 6, 10, 4, 600.00, '2025-11-18'), (14, 7, 7, 1, 122.50, '2025-11-12'), (15, 7, 1, 3, 750.00, '2025-11-14'),
(16, 8, 8, 5, 1500.00, '2025-12-15'), (17, 8, 2, 2, 241.00, '2025-12-05'), (18, 9, 9, 3, 600.00, '2025-12-05'),
(19, 9, 4, 1, 119.99, '2025-11-16'), (20, 10, 10, 2, 300.98, '2025-11-18'), (21, 10, 3, 2, 979.98, '2025-11-20');

```

```

mysql> select*from booking;
+-----+-----+-----+-----+-----+-----+
| booking_id | num_tickets | total_cost | booking_date | customer_id | event_id |
+-----+-----+-----+-----+-----+-----+
| 1 | 2 | 500.00 | 2025-11-10 | 1 | 1 |
| 2 | 1 | 489.99 | 2025-11-12 | 1 | 3 |
| 3 | 4 | 600.00 | 2025-11-28 | 1 | 5 |
| 4 | 4 | 482.00 | 2025-12-01 | 2 | 2 |
| 5 | 2 | 245.00 | 2025-11-15 | 2 | 7 |
| 6 | 1 | 489.99 | 2025-11-20 | 3 | 3 |
| 7 | 3 | 900.00 | 2025-12-05 | 3 | 8 |
| 8 | 3 | 359.97 | 2025-11-15 | 4 | 4 |
| 9 | 2 | 950.00 | 2025-12-10 | 4 | 6 |
| 10 | 2 | 300.00 | 2025-11-30 | 5 | 5 |
| 11 | 1 | 200.00 | 2025-12-08 | 5 | 9 |
| 12 | 2 | 950.00 | 2025-12-10 | 6 | 6 |
| 13 | 4 | 600.00 | 2025-11-18 | 6 | 10 |
| 14 | 1 | 122.50 | 2025-11-12 | 7 | 7 |
| 15 | 3 | 750.00 | 2025-11-14 | 7 | 1 |
| 16 | 5 | 1500.00 | 2025-12-15 | 8 | 8 |
| 17 | 2 | 241.00 | 2025-12-05 | 8 | 2 |
| 18 | 3 | 600.00 | 2025-12-05 | 9 | 9 |
| 19 | 1 | 119.99 | 2025-11-16 | 9 | 4 |
| 20 | 2 | 300.98 | 2025-11-18 | 10 | 10 |
| 21 | 2 | 979.98 | 2025-11-20 | 10 | 3 |
+-----+-----+-----+-----+-----+-----+
21 rows in set (0.01 sec)

```

2. Write a SQL query to list all Events

select*from event;

The screenshot shows the MySQL Workbench interface with a query editor window titled "SQL File 1". The query "select*from event;" is entered. Below the editor is the "Result Grid" which displays 11 rows of event data. The columns are: event_id, event_name, event_date, event_time, total_seats, available_seats, ticket_price, event_type, and venue_id. The data includes various events like "Court teaser release", "Football Final", "Coldplay Concert", etc. At the bottom of the grid, it says "event 35 x". Under the "Output" section, there is an "Action Output" table with one row showing the query and its execution time, along with a message indicating 11 rows returned.

event_id	event_name	event_date	event_time	total_seats	available_seats	ticket_price	event_type	venue_id
1	Court teaser release	2025-11-15	19:00:00	500	150	250.00	Movie	1
2	Football Final	2025-12-10	20:30:00	20000	5000	120.50	Sports	2
3	Coldplay Concert	2025-11-25	21:00:00	10000	2000	489.99	Concert	3
4	The Batman	2025-11-20	18:30:00	300	75	119.99	Movie	4
5	World cup Match	2025-12-05	18:00:00	70000	10000	150.00	Sports	5
6	AR Rahman Concert	2025-12-15	19:30:00	1200	0	475.00	Concert	6
7	Black Panther 2	2025-11-18	20:00:00	500	200	122.50	Movie	1
8	U1 drugs	2025-12-20	20:00:00	15000	2500	300.00	Concert	7
9	Yard Games Championship	2025-12-12	19:00:00	20000	3000	200.00	Sports	2
10	Zenith Film Premiere	2025-11-22	17:30:00	300	0	150.00	Movie	4
11	Unpopular Jazz Night	2025-12-25	20:00:00	200	200	100.00	Concert	8

event 35 x

Output

#	Time	Action	Message
1	16:36:54	select*from event LIMIT 0, 1000	11 row(s) returned

3. Write a SQL query to select events with available tickets.

select*from event where available_seats>0;

The screenshot shows the MySQL Workbench interface with a query editor window titled "SQL File 1". The query "select*from event where available_seats>0;" is entered. Below the editor is the "Result Grid" which displays 9 rows of event data, filtering out the row where available_seats is 0. The columns are: event_id, event_name, event_date, event_time, total_seats, available_seats, ticket_price, event_type, and venue_id. The data includes events like "Court teaser release", "Football Final", "Coldplay Concert", etc. At the bottom of the grid, it says "event 37 x". Under the "Output" section, there is an "Action Output" table with one row showing the query and its execution time, along with a message indicating 9 rows returned.

event_id	event_name	event_date	event_time	total_seats	available_seats	ticket_price	event_type	venue_id
1	Court teaser release	2025-11-15	19:00:00	500	150	250.00	Movie	1
2	Football Final	2025-12-10	20:30:00	20000	5000	120.50	Sports	2
3	Coldplay Concert	2025-11-25	21:00:00	10000	2000	489.99	Concert	3
4	The Batman	2025-11-20	18:30:00	300	75	119.99	Movie	4
5	World cup Match	2025-12-05	18:00:00	70000	10000	150.00	Sports	5
7	Black Panther 2	2025-11-18	20:00:00	500	200	122.50	Movie	1
8	U1 drugs	2025-12-20	20:00:00	15000	2500	300.00	Concert	7
9	Yard Games Championship	2025-12-12	19:00:00	20000	3000	200.00	Sports	2
11	Unpopular Jazz Night	2025-12-25	20:00:00	200	200	100.00	Concert	8

event 37 x

Output

#	Time	Action	Message
1	16:37:35	select*from event where available_seats>0 LIMIT 0, 1000	9 row(s) returned

4. Write a SQL query to select events name partial match with ‘cup’.

```
select event_name from event where event_name like '%cup%';
```

The screenshot shows the MySQL Workbench interface. In the top-left pane, there is a code editor window titled "SQL File 1*" containing the SQL query: "select event_name from event where event_name like '%cup%'". Below the code editor is a "Result Grid" table with one row, "World cup Match". At the bottom of the interface, there is an "Output" panel showing the execution log: "1 15:00:23 select event_name from event where event_name like '%cup%' LIMIT 0, 1000" and "1 row(s) returned".

5. Write a SQL query to select events with ticket price range is between 100 to 250.

```
select*from event where ticket_price between 100 and 250;
```

The screenshot shows the MySQL Workbench interface. In the top-left pane, there is a code editor window titled "SQL File 1*" containing the SQL query: "select*from event where ticket_price between 100 and 250;". Below the code editor is a "Result Grid" table displaying 8 rows of event data. At the bottom of the interface, there is an "Output" panel showing the execution log: "1 16:39:16 select*from event where ticket_price between 100 and 250 LIMIT 0, 1000" and "8 row(s) returned".

event_id	event_name	event_date	event_time	total_seats	available_seats	ticket_price	event_type	venue_id
1	Court teaser release	2025-11-15	19:00:00	500	150	250.00	Movie	1
2	Football Final	2025-12-10	20:30:00	20000	5000	120.50	Sports	2
4	The Batman	2025-11-20	18:30:00	300	75	119.99	Movie	4
5	World cup Match	2025-12-05	18:00:00	70000	10000	150.00	Sports	5
7	Black Panther 2	2025-11-18	20:00:00	500	200	122.50	Movie	1
9	Yard Games Championship	2025-12-12	19:00:00	20000	3000	200.00	Sports	2
10	Zenith Film Premiere	2025-11-22	17:30:00	300	0	150.00	Movie	4
11	Unpopular Jazz Night	2025-12-25	20:00:00	200	200	100.00	Concert	8

6. Write a SQL query to retrieve events with dates falling within a specific range.

```
select*from event where event_date between '2025-11-05' and '2025-11-20';
```

The screenshot shows the MySQL Workbench interface with a SQL editor window titled "SQL File 1". The query is:

```
1     select*from event where event_date between '2025-11-05' and '2025-11-20';
2
```

The results grid displays three rows of event data:

event_id	event_name	event_date	event_time	total_seats	available_seats	ticket_price	event_type	venue_id
1	Court teaser release	2025-11-15	19:00:00	500	150	250.00	Movie	1
4	The Batman	2025-11-20	18:30:00	300	75	119.99	Movie	4
7	Black Panther 2	2025-11-18	20:00:00	500	200	122.50	Movie	1

The output pane shows the executed query and the result message:

```
Action Output
# Time Action
1 15:02:24 select*from event where event_date between '2025-11-05' and '2025-11-20' LIMIT 0, 1... 3 row(s) returned
```

7. Write a SQL query to retrieve events with available tickets that also have "Concert" in their name.

```
select*from event where available_seats>0 and event_type='Concert';
```

The screenshot shows the MySQL Workbench interface with a SQL editor window titled "SQL File 1". The query is:

```
1 •    select*from event where available_seats>0 and event_type='Concert';
2
```

The results grid displays three rows of event data:

event_id	event_name	event_date	event_time	total_seats	available_seats	ticket_price	event_type	venue_id
3	Coldplay Concert	2025-11-25	21:00:00	10000	2000	489.99	Concert	3
8	U1 drugs	2025-12-20	20:00:00	15000	2500	300.00	Concert	7
11	Unpopular Jazz Night	2025-12-25	20:00:00	200	200	100.00	Concert	8

The output pane shows the executed query and the result message:

```
Action Output
# Time Action
1 16:40:44 select*from event where available_seats>0 and event_type='Concert' LIMIT 0, 1000 3 row(s) returned
```

8. Write a SQL query to retrieve users in batches of 5, starting from the 6th user.

```
select*from customer limit 5 offset 5;
```

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

```
1 • select*from customer
2     limit 5 offset 5;
3
```

The results grid displays 5 rows of customer data:

customer_id	customer_name	email	phone_number
6	Sneha Rao	sneha.rao@rediff.com	9111223344
7	Karthik Iyer	karthik@xyz.com	9345612789
8	Riya Das	riya.das@gmail.com	9321456789
9	Pranav Joshi	pranavj@yahoo.com	9456721000
10	Tanvi Nair	tanvi.nair@qmail.com	9988001122

The output pane shows the executed query and the message "5 row(s) returned".

9. Write a SQL query to retrieve bookings details contains booked no of ticket more than 3

```
select*from booking where num_tickets>3;
```

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

```
1 • select*from booking where num_tickets>3;
2
3
```

The results grid displays 4 rows of booking data:

booking_id	num_tickets	total_cost	booking_date	customer_id	event_id
3	4	600.00	2025-11-28	1	5
4	4	482.00	2025-12-01	2	2
13	4	600.00	2025-11-18	6	10
16	5	1500.00	2025-12-15	8	8

The output pane shows the executed query and the message "4 row(s) returned".

10. Write a SQL query to retrieve customer information whose phone number end with '000'

```
select*from customer where phone_number like '%000';
```

The screenshot shows the SQL Developer interface with a query editor and a results grid.

Query Editor:

```
SQL File 1* 
1 • select*from customer where phone_number like '%000';
```

Results Grid:

	customer_id	customer_name	email	phone_number
▶	2	Aisha Khan	aisha.khan@yahoo.com	9123456000
▶	9	Pranav Joshi	pranavj@yahoo.com	9456721000
▶	11	Noah Kumar	noah.kumar@example.com	9000000000

Action Output:

#	Time	Action	Message
1	16:43:09	select*from customer where phone_number like '%000' LIMIT 0, 1000	3 row(s) returned

11. Write a SQL query to retrieve the events in order whose seat capacity more than 15000.

```
select*from event where total_seats>15000;
```

The screenshot shows the SQL Developer interface with a query editor and a results grid.

Query Editor:

```
SQL File 1* 
1 • select*from event where total_seats>15000;
```

Results Grid:

	event_id	event_name	event_date	event_time	total_seats	available_seats	ticket_price	event_type	venue_id
▶	2	Football Final	2025-12-10	20:30:00	20000	5000	120.50	Sports	2
▶	5	World cup Match	2025-12-05	18:00:00	70000	10000	150.00	Sports	5
▶	9	Yard Games Championship	2025-12-12	19:00:00	20000	3000	200.00	Sports	2

Action Output:

#	Time	Action	Message
1	15:08:06	select*from event where total_seats>15000 LIMIT 0, 1000	3 row(s) returned

12. Write a SQL query to select events name not start with 'x', 'y', 'z'

```
select event_name from event where event_name not like 'x%' and  
event_name not like 'y%' and  
event_name not like 'z%';
```

SQL File 1* x

The screenshot shows the MySQL Workbench interface with a query editor window titled "SQL File 1*". The query is:

```
1  select event_name from event where event_name not like 'x%' and  
2  event_name not like 'y%' and  
3  event_name not like 'z%';  
4
```

Below the query, the "Result Grid" displays the following data:

event_name
Court teaser release
Football Final
Coldplay Concert
The Batman
World cup Match
AR Rahman Concert
Black Panther 2
U1 drugs
Unpopular Jazz Night

Task 3: Aggregate functions, Having, Order By, Group By and Joins:

1. Write a SQL query to List Events and Their Average Ticket Prices.

```
select e.event_name,round(avg(e.ticket_price),2) as avg_ticketprice  
from event e  
group by event_name;
```

The screenshot shows an SQL interface with the following details:

SQL File 1 window:

```
1 • select e.event_name,round(avg(e.ticket_price),2) as avg_ticketprice  
2   from event e  
3   group by event_name;
```

Result Grid table:

event_name	avg_ticketprice
Coldplay Concert	489.99
The Batman	119.99
World cup Match	150.00
AR Rahman Concert	475.00
Black Panther 2	122.50
UI drugs	300.00
Yard Games Championship	200.00
Zenith Film Premiere	150.00
Unpopular Jazz Night	100.00

Action Output log:

#	Time	Action	Message
1	16:51:33	select e.event_name,round(avg(e.ticket_price),2) as avg_ticketprice from event e group... 11 row(s) returned	

2. Write a SQL query to Calculate the Total Revenue Generated by Events.

```
select event_name, sum(b.total_cost) as total_revenue  
from event e  
join booking b on  
b.event_id=e.event_id  
group by e.event_name;
```

The screenshot shows an SQL interface with the following details:

SQL File 1 window:

```
1 select event_name, sum(b.total_cost) as total_revenue  
2   from event e  
3   join booking b on  
4     b.event_id=e.event_id  
5   group by e.event_name;
```

Result Grid table:

event_name	total_revenue
Court teaser release	1250.00
Football Final	723.00
Coldplay Concert	1959.96
The Batman	479.96
World cup Match	900.00
AR Rahman Concert	1900.00
Black Panther 2	367.50
UI drugs	2400.00
Yard Games Championship	800.00
Zenith Film Premiere	900.98

Action Output log:

#	Time	Action	Message
1	20:05:27	select event_name, sum(b.total_cost) as total_revenue from event e join booking b on b... 10 row(s) returned	

3. Write a SQL query to find the event with the highest ticket sales.

```
select e.event_name,sum(b.num_tickets) as highticket_sales from event e  
join booking b on e.event_id=b.event_id  
group by event_name order by highticket_sales desc limit 1;
```

The screenshot shows the MySQL Workbench interface. The top window is titled "SQL File 1" and contains the SQL query. The bottom window is titled "Result Grid" and displays the results of the query. The output window at the bottom shows the execution message.

```
SQL File 1*  
1 • select e.event_name,sum(b.num_tickets) as highticket_sales  
2   from event e  
3   join booking b on  
4     e.event_id=b.event_id  
5   group by event_name  
6   order by highticket_sales desc  
7   limit 1;  
8  
9
```

event_name	highticket_sales
U1 drugs	8

```
Result 78 x  
Output:  
Action Output # Time Action Message  
1 15:26:51 select e.event_name,sum(b.num_tickets) as highticket_sales from event e join booking ... 1 row(s) returned
```

4. Write a SQL query to Calculate the Total Number of Tickets Sold for Each Event.

```
select e.event_name,sum(b.num_tickets) as total_tickets from event e  
join booking b on e.event_id=b.event_id  
group by event_name;  
order by total_tickets desc;
```

The screenshot shows the MySQL Workbench interface. The top window is titled "SQL File 1" and contains the SQL query. The bottom window is titled "Result Grid" and displays the results of the query. The output window at the bottom shows the execution message.

```
SQL File 1*  
1 • select e.event_name,sum(b.num_tickets) as total_tickets  
2   from event e  
3   join booking b on  
4     e.event_id=b.event_id  
5   group by event_name
```

event_name	total_tickets
U1 drugs	8
Football Final	6
World cup Match	6
Zenith Film Premiere	6
Court teaser release	5
Coldplay Concert	4
The Batman	4
AR Rahman Concert	4
Yard Games Championship	4
Black Panther 2	3

```
Result 79 x  
Output:  
Action Output # Time Action Message  
1 15:27:33 select e.event_name,sum(b.num_tickets) as total_tickets from event e join booking b o... 10 row(s) returned
```

5. Write a SQL query to Find Events with No Ticket Sales.

```
select e.event_id,e.event_name from event e  
left join booking b on b.event_id=e.event_id  
where b.booking_id is null;
```

The screenshot shows the MySQL Workbench interface with a query editor titled "SQL File 1". The query is:

```
1 •  select e.event_id,e.event_name  
2   from event e  
3   left join booking b on  
4     b.event_id=e.event_id  
5   where b.booking_id is null;
```

The results are displayed in a "Result Grid" table:

event_id	event_name
11	Unpopular Jazz Night

The "Output" pane shows the execution log:

#	Time	Action	Message
1	16:56:24	select e.event_id,e.event_name from event e left join booking b on b.event_id=e.event_id... where b.booking_id is null;	1 row(s) returned

6. Write a SQL query to Find the User Who Has Booked the Most Tickets

```
select c.customer_name,sum(b.num_tickets) as max_ticket  
from customer c  
join booking b on c.customer_id=b.customer_id  
group by customer_name order by max_ticket desc limit 1;
```

The screenshot shows the MySQL Workbench interface with a query editor titled "SQL File 1". The query is:

```
1 •  select c.customer_name,sum(b.num_tickets) as max_ticket  
2   from customer c  
3   join booking b on  
4     c.customer_id=b.customer_id  
5   group by customer_name  
6   order by max_ticket desc  
7   limit 1;
```

The results are displayed in a "Result Grid" table:

customer_name	max_ticket
Rahul Sharma	7

The "Output" pane shows the execution log:

#	Time	Action	Message
1	15:29:02	select c.customer_name,sum(b.num_tickets) as max_ticket from customer c join booking b on c.customer_id=b.customer_id group by customer_name order by max_ticket desc limit 1;	1 row(s) returned

7. Write a SQL query to List Events and the total number of tickets sold for each month.

```
select e.event_name,sum(b.num_tickets) as total_tickets,monthname(e.event_date) as sold_month  
from event e
```

```
join booking b on e.event_id=b.event_id  
group by event_name,event_date  
order by total_tickets;
```

The screenshot shows a SQL IDE interface with a code editor and a result grid. The code editor contains the SQL query from question 7. The result grid displays the output with columns: event_name, total_tickets, and sold_month. The output window shows the execution details and a message indicating 10 rows returned.

event_name	total_tickets	sold_month
Black Panther 2	3	November
Coldplay Concert	4	November
The Batman	4	November
AR Rahman Concert	4	December
Yard Games Championship	4	December
Court teaser release	5	November
Football Final	6	December
World cup Match	6	December
Zenith Film Premiere	6	November
U1 drugs	8	December

Result 81 x
Output
Action Output
Time Action Message
1 15:29:55 select e.event_name,sum(b.num_tickets) as total_tickets,monthname(e.event_date) as ... 10 row(s) returned

8. Write a SQL query to calculate the average Ticket Price for Events in Each Venue.

```
select e.event_name,round(avg(e.ticket_price),2) as avgticket_price,v.venue_name  
from event e  
join venue v on  
e.venue_id=v.venue_id  
group by event_name,venue_name  
order by avgticket_price;
```

The screenshot shows a SQL query in the 'SQL File 1*' window:

```

1 • select e.event_name,round(avg(e.ticket_price),2) as avgticket_price,v.venue_name
2   from event e
3   join venue v on
4     e.venue_id=v.venue_id
5   group by event_name,venue_name
6   order by avgticket_price;
7

```

The results are displayed in the 'Result Grid' table:

event_name	avgticket_price	venue_name
Unpopular Jazz Night	100.00	Metro Multiplex
The Batman	119.99	Star Cinemas
Football Final	120.50	Skyline Stadium
Black Panther 2	122.50	Grand Arena
Zenith Film Premiere	150.00	Star Cinemas
World cup Match	150.00	City Convention Ctr
Yard Games Championship	200.00	Skyline Stadium
Court teaser release	250.00	Grand Arena
U1 drugs	300.00	Ocean View Grounds
AR Rahman Concert	475.00	Green Park Theatre
Coldplay Concert	489.99	Harmony Hall

The output window shows the execution details:

```

Result 60 x
Output:
Action Output
# Time Action
1 16:58:38 select e.event_name,round(avg(e.ticket_price),2) as avgticket_price,v.venue_name fro... 11 row(s) returned

```

9. Write a SQL query to calculate the total Number of Tickets Sold for Each Event Type.

```

select e.event_type,sum(b.num_tickets) as total_ticketssold
from event e
join booking b on
e.event_id=b.event_id
group by event_type
order by total_ticketssold;

```

The screenshot shows a SQL query in the 'SQL File 1*' window:

```

1 • select e.event_type,sum(b.num_tickets) as total_ticketssold
2   from event e
3   join booking b on
4     e.event_id=b.event_id
5   group by event_type
6   order by total_ticketssold;
7
8
9
10
11

```

The results are displayed in the 'Result Grid' table:

event_type	total_ticketssold
Sports	16
Concert	16
Movie	18

The output window shows the execution details:

```

Result 83 x
Output:
Action Output
# Time Action
1 15:33:43 select e.event_type,sum(b.num_tickets) as total_ticketssold from event e join booking b ... 3 row(s) returned

```

10. Write a SQL query to calculate the total Revenue Generated by Events in Each Year.

```
select year(e.event_date) as yearr , sum(b.total_cost) as totalcost  
from event e  
join booking b on  
e.event_id=b.event_id  
group by yearr  
order by totalcost;
```

The screenshot shows the MySQL Workbench interface with two panes. The top pane is titled "SQL File 1" and contains the SQL code for question 10. The bottom pane is titled "Result Grid" and displays the query's output. The output shows a single row with the year 2025 and a total cost of 11681.40. The "Result 84" pane at the bottom shows the message "1 row(s) returned".

yearr	totalcost
2025	11681.40

Result 84

Action Output

1 15:34:27 select year(e.event_date) as yearr , sum(b.total_cost) as totalcost from event e join book...

Message

1 row(s) returned

11. Write a SQL query to list users who have booked tickets for multiple events.

```
select customer_name,count(b.event_id) as booked_events  
from customer c  
join booking b on b.customer_id = c.customer_id  
group by customer_name  
having count(b.event_id)>1  
order by booked_events;
```

The screenshot shows the SQL Developer interface with a query window titled "SQL File 1*". The query is:

```

1 • select customer_name, count(b.event_id) as booked_events
2   from customer c
3   join booking b on
4     b.customer_id=c.customer_id
5   group by customer_name
6   having count(b.event_id)>1
7   order by booked_events;

```

The result grid displays the following data:

customer_name	booked_events
Aisha Khan	2
Vinay Menon	2
Meera Suresh	2
Arjun Patel	2
Sneha Rao	2
Karthik Iyer	2
Riya Das	2
Pranav Joshi	2
Tanvi Nair	2
Rahul Sharma	3

The output pane shows the message: "1 16:35:46 select customer_name, count(b.event_id) as booked_events from customer c join booki... 10 row(s) returned".

12. Write a SQL query to calculate the Total Revenue Generated by Events for Each User.

```

select c.customer_id,c.customer_name,SUM(b.total_cost) AS total_revenue_generated
from customer c
join booking b ON
c.customer_id = b.customer_id
group by customer_id, customer_name
order by total_revenue_generated DESC;

```

The screenshot shows the SQL Developer interface with a query window titled "SQL File 1*". The query is:

```

1 • select c.customer_id,c.customer_name,sum(b.total_cost) AS total_revenue_generated
2   from customer c
3   join booking b ON
4     c.customer_id = b.customer_id
5   group by customer_id, customer_name
6   order by total revenue generated DESC;

```

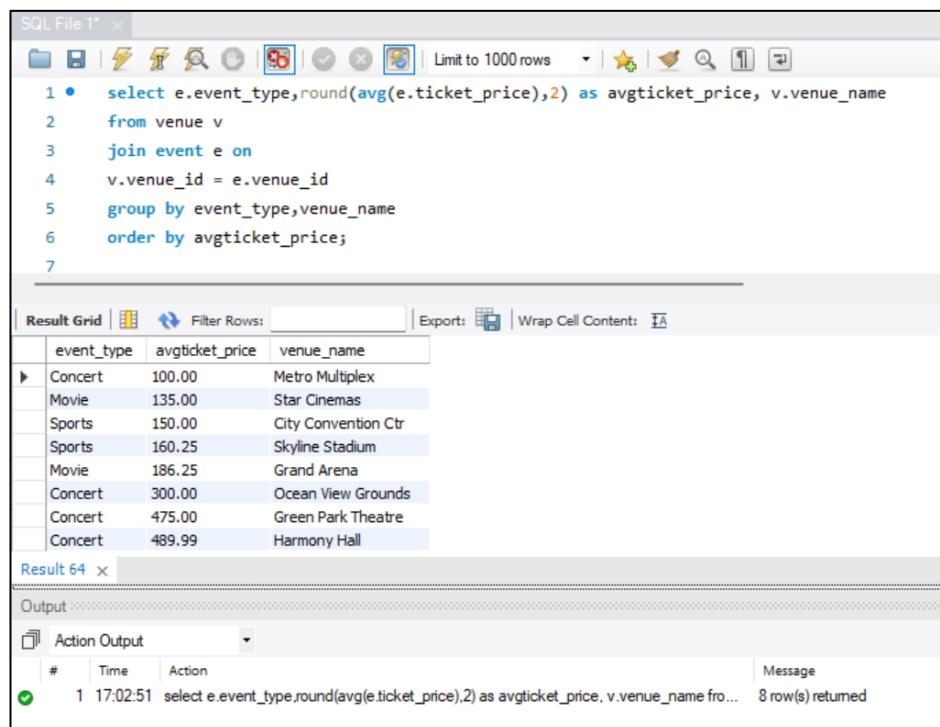
The result grid displays the following data:

customer_id	customer_name	total_revenue_generated
8	Riya Das	1741.00
1	Rahul Sharma	1589.99
6	Sneha Rao	1550.00
3	Vinay Menon	1389.99
4	Meera Suresh	1309.97
10	Tanvi Nair	1280.96
7	Karthik Iyer	872.50
2	Aisha Khan	727.00
9	Pranav Joshi	719.99
5	Arjun Patel	500.00

The output pane shows the message: "1 15:35:09 select c.customer_id,c.customer_name,sum(b.total_cost) AS total_revenue_generated... 10 row(s) returned".

13. Write a SQL query to calculate the Average Ticket Price for Events in Each Category and Venue.

```
select e.event_type,round(avg(e.ticket_price),2) as avgticket_price, v.venue_name
from venue v
join event e on
v.venue_id = e.venue_id
group by event_type,venue_name
order by avgticket_price;
```



The screenshot shows a SQL IDE interface with the following details:

- SQL File 1**: The code area contains the SQL query provided above.
- Result Grid**: The results are displayed in a grid format:

event_type	avgticket_price	venue_name
Concert	100.00	Metro Multiplex
Movie	135.00	Star Cinemas
Sports	150.00	City Convention Ctr
Sports	160.25	Skyline Stadium
Movie	186.25	Grand Arena
Concert	300.00	Ocean View Grounds
Concert	475.00	Green Park Theatre
Concert	489.99	Harmony Hall

- Output**: The output pane shows the message: "8 row(s) returned".

14. Write a SQL query to list Users and the Total Number of Tickets They've Purchased in the Last 30 days

```
select c.customer_id,c.customer_name,sum(b.num_tickets) as total_tickets_purchased
from customer c
left join booking b on b.customer_id=c.customer_id
and b.booking_date>='2025-11-01' and b.booking_date<='2025-11-30'
group by c.customer_id,c.customer_name
order by total_tickets_purchased desc;
```

```

SQL File 1* 
1 • select c.customer_id, c.customer_name, sum(b.num_tickets) as total_tickets_purchased
2   from customer c
3   left join booking b on
4     b.customer_id=c.customer_id
5   and b.booking_date>='2025-11-01'
6   and b.booking_date<='2025-11-30'
7   group by c.customer_id,c.customer_name
8   order by total_tickets_purchased desc;

```

Result Grid

customer_id	customer_name	total_tickets_purchased
1	Rahul Sharma	7
6	Sneha Rao	4
7	Karthik Iyer	4
10	Tanvi Nair	4
4	Meera Suresh	3
2	Aisha Khan	2
5	Arjun Patel	2
3	Vinay Menon	1
9	Pranav Joshi	1
8	Riya Das	HULL
11	Noah Kumar	HULL

Tasks 4: Subquery and its types

1. Calculate the Average Ticket Price for Events in Each Venue Using a Subquery.

```

select v.venue_name,
       select round(avg(e.ticket_price),2)
  from event e where e.venue_id=v.venue_id) as avgticket_price
   from venue v;

```

```

SQL File 1* 
1 • select v.venue_name,
2       select round(avg(e.ticket_price),2)
3       from event e
4     where e.venue_id=v.venue_id)as avgticket_price
5       from venue v;

```

Result Grid

venue_name	avgticket_price
Grand Arena	186.25
Skyline Stadium	160.25
Harmony Hall	489.99
Star Cinemas	135.00
City Convention Ctr	150.00
Green Park Theatre	475.00
Ocean View Grounds	300.00
Metro Multiplex	100.00
Diamond Club	HULL
Royal Event Space	HULL

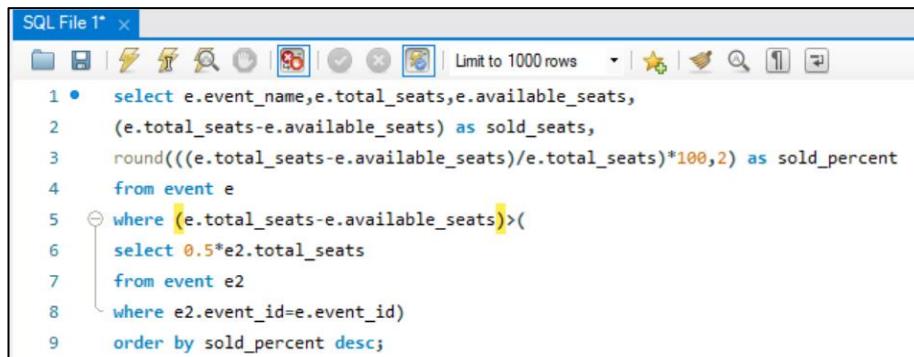
Result 68 x

Output

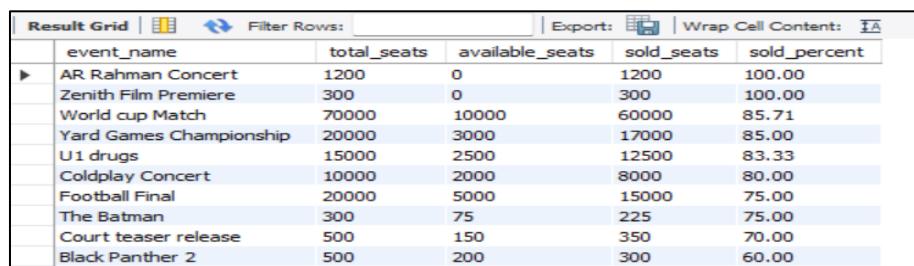
#	Time	Action	Message
1	17:26:13	select v.venue_name,(select round(avg(e.ticket_price),2) from event e where e.venue...	10 row(s) returned

2. Find Events with More Than 50% of Tickets Sold using subquery.

```
select e.event_name,e.total_seats,e.available_seats,
(e.total_seats-e.available_seats) as sold_seats,
round(((e.total_seats-e.available_seats)/e.total_seats)*100,2)as sold_percent
from event e
where (e.total_seats-e.available_seats)>(
select 0.5*e2.total_seats
from event e2
where e2.event_id=e.event_id)
order by sold_percent desc;
```



```
SQL File 1* 
1 •   select e.event_name,e.total_seats,e.available_seats,
2     (e.total_seats-e.available_seats) as sold_seats,
3     round(((e.total_seats-e.available_seats)/e.total_seats)*100,2) as sold_percent
4   from event e
5   where (e.total_seats-e.available_seats)>
6     select 0.5*e2.total_seats
7     from event e2
8     where e2.event_id=e.event_id
9   order by sold_percent desc;
```



event_name	total_seats	available_seats	sold_seats	sold_percent
AR Rahman Concert	1200	0	1200	100.00
Zenith Film Premiere	300	0	300	100.00
World cup Match	70000	10000	60000	85.71
Yard Games Championship	20000	3000	17000	85.00
U1 drugs	15000	2500	12500	83.33
Coldplay Concert	10000	2000	8000	80.00
Football Final	20000	5000	15000	75.00
The Batman	300	75	225	75.00
Court teaser release	500	150	350	70.00
Black Panther 2	500	200	300	60.00

3. Calculate the Total Number of Tickets Sold for Each Event.

```
select event_name,
(select sum(b.num_tickets) from booking b
where b.event_id=e.event_id) as tickets_sold
from event e
order by tickets_sold desc;
```

SQL File 1*

```

1 • ⏷ select event_name,
2     select sum(b.num_tickets)
3     from booking b
4     where b.event_id=e.event_id)
5     as tickets_sold
6     from event e
7     order by tickets_sold desc;

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

event_name	tickets_sold
U1 drugs	8
Football Final	6
World cup Match	6
Zenith Film Premiere	6
Court teaser release	5
Coldplay Concert	4
The Batman	4
AR Rahman Concert	4
Yard Games Championship	4
Black Panther 2	3
Unpopular Jazz Night	HULL

4. Find Users Who Have Not Booked Any Tickets Using a NOT EXISTS Subquery.

```

select c.customer_name
from customer c
where not exists(
    select 1 from booking b where b.customer_id=c.customer_id);

```

SQL File 1*

```

3 • ⏷ where not exists(
4     select 1
5     from booking b
6     where b.customer_id=c.customer_id);
7

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

customer_name
Noah Kumar

customer 73 x

Output:

Action Output

#	Time	Action	Message
1	17:38:26	select c.customer_name from customer c where not exists(select 1 from booking b wh...	1 row(s) returned

5. List Events with No Ticket Sales Using a NOT IN Subquery.

```
select e.event_name  
from event e  
where event_id not in(  
    select event_id from booking);
```

The screenshot shows the MySQL Workbench interface with two panes. The top pane is titled 'SQL File 1*' and contains the SQL code:

```
1 •  select e.event_name  
2   from event e  
3   where event_id not in  
4       select event_id from booking;
```

The bottom pane is titled 'Result Grid' and displays the output:

event_name
Unpopular Jazz Night

The 'Output' pane at the bottom right shows the log entry:

#	Time	Action	Message
1	17:41:36	select e.event_name from event e where event_id not in(select event_id from booking) ...	1 row(s) returned

6. Calculate the Total Number of Tickets Sold for Each Event Type Using a Subquery in the FROM Clause.

```
select event_type, SUM(tickets_sold) AS total_tickets  
from(  
    select e.event_type,  
    (select sum(b.num_tickets) from booking b  
     where b.event_id=e.event_id) as tickets_sold  
    from event e  
) as sales  
group by event_type;
```

SQL File 1*

```

1 •   select event_type, SUM(tickets_sold) AS total_tickets
2   • from (
3       select e.event_type,
4             (select SUM(b.num_tickets)
5              from booking b
6             where b.event_id = e.event_id) as tickets_sold
7       from event e
8   ) as sales
9   group by event_type;

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

event_type	total_tickets
Movie	18
Sports	16
Concert	16

Result 105 x

Output:

Action Output

#	Time	Action	Message
1	16:17:04	SELECT event_type, SUM(tickets_sold) AS total_tickets FROM (SELECT ...	3 row(s) returned

7. Find Events with Ticket Prices Higher Than the Average Ticket Price Using a Subquery in the WHERE Clause.

```

select event_name,ticket_price
from event
where ticket_price > ( select avg(ticket_price) from event);

```

SQL File 1*

```

1 •   select event_name,ticket_price
2     from event
3   • where ticket_price > (
4       select avg(ticket_price)
5         from event
6   );

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

event_name	ticket_price
Court teaser release	250.00
Coldplay Concert	489.99
AR Rahman Concert	475.00
U1 drugs	300.00

event 106 x

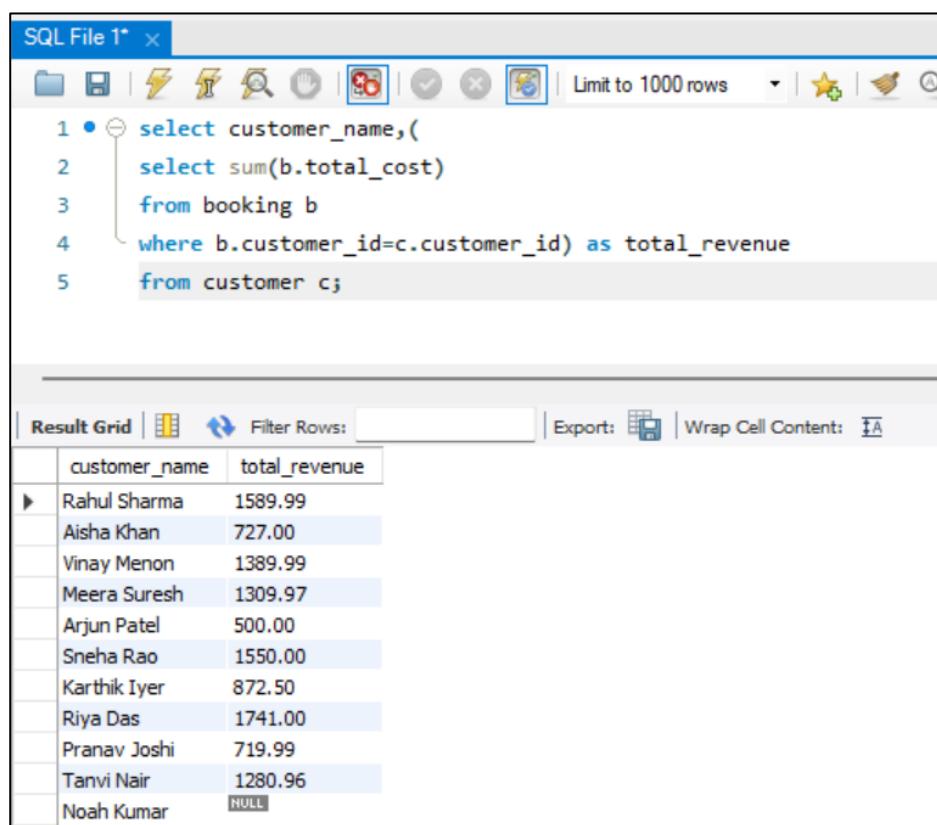
Output:

Action Output

#	Time	Action	Message
1	16:20:35	SELECT event_name, ticket_price FROM event WHERE ticket_price > (SELE...	4 row(s) returned

8. Calculate the Total Revenue Generated by Events for Each User Using a Correlated Subquery.

```
select customer_name,
       select sum(b.total_cost)
  from booking b where b.customer_id=c.customer_id) as total_revenue
   from customer c;
```



The screenshot shows a SQL editor window titled "SQL File 1" with the following query:

```
1 • ⚏ select customer_name,
2      select sum(b.total_cost)
3      from booking b
4     where b.customer_id=c.customer_id) as total_revenue
5      from customer c;
```

Below the query is a "Result Grid" table with two columns: "customer_name" and "total_revenue". The data is as follows:

customer_name	total_revenue
Rahul Sharma	1589.99
Aisha Khan	727.00
Vinay Menon	1389.99
Meera Suresh	1309.97
Arjun Patel	500.00
Sneha Rao	1550.00
Karthik Iyer	872.50
Riya Das	1741.00
Pranav Joshi	719.99
Tanvi Nair	1280.96
Noah Kumar	NULL

9. List Users Who Have Booked Tickets for Events in a Given Venue Using a Subquery in the WHERE Clause.

```
select c.customer_name from customer c
  where exists(
    select booking_id from booking b
      join event e on b.event_id = e.event_id
      where b.customer_id=c.customer_id
        and e.venue_id=7 );
```

```

SQL File 1* 
1 •  select c.customer_name
2   from customer c
3   where exists (
4     select booking_id
5       from booking b
6       join event e on b.event_id = e.event_id
7      where b.customer_id = c.customer_id
8      and e.venue_id = 7
9   );

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

customer_name
Vinay Menon
Riya Das

customer 117 x

Output:

Action Output

#	Time	Action
1	17:19:44	select c.customer_name from customer c where exists (select booking_id from boo... 2 row(s) returned

10. Calculate the Total Number of Tickets Sold for Each Event Category Using a Subquery with GROUP BY.

```

select e.event_type,
(select SUM(b.num_tickets) from booking b
join event e2 on b.event_id = e2.event_id
where e2.event_type = e.event_type) as tickets_sold
from event e group by e.event_type;

```

```

SQL File 1* 
1 •  select e.event_type,
2   (select SUM(b.num_tickets)
3    from booking b
4    join event e2 on b.event_id = e2.event_id
5    where e2.event_type = e.event_type) as tickets_sold
6   from event e
7   group by e.event_type;

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

event_type	tickets_sold
Movie	18
Sports	16
Concert	16

Result 121 x

Output:

Action Output

#	Time	Action
1	17:27:40	select e.event_type, (select SUM(b.num_tickets) from booking b join event e2 on ... 3 row(s) returned

11. Find Users Who Have Booked Tickets for Events in each Month Using a Subquery with DATE_FORMAT.

```
select distinct b.customer_id, c.customer_name, b.booking_date
from booking b
join customer c on b.customer_id=c.customer_id
where not exists (
    select 1 from (
        select distinct DATE_FORMAT(booking_date, '%m') as booking_month
        from Booking ) all_months
    where not exists (
        select 1 from Booking b2
        where b2.customer_id = b.customer_id
        and DATE_FORMAT(b2.booking_date, '%m') = all_months.booking_month)
);

```

SQL File 1*

```
1 ●   select distinct b.customer_id, c.customer_name, b.booking_date
2   from Booking b
3   join customer c on
4     b.customer_id=c.customer_id
5   where not exists (
6     select 1 from (
7       select distinct DATE_FORMAT(booking_date, '%m') as booking_month
8       from Booking
9     ) all_months
10    where not exists (
11      select 1 from Booking b2
12      where b2.customer_id = b.customer_id and DATE_FORMAT(b2.booking_date, '%m') = all_months.booking_month
13    ));
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	customer_id	customer_name	booking_date
▶	2	Aisha Khan	2025-12-01
	2	Aisha Khan	2025-11-15
	3	Vinay Menon	2025-11-20
	3	Vinay Menon	2025-12-05
	4	Meera Suresh	2025-11-15
	4	Meera Suresh	2025-12-10
	5	Arjun Patel	2025-11-30
	5	Arjun Patel	2025-12-08
	6	Sneha Rao	2025-12-10
	6	Sneha Rao	2025-11-18
	9	Pranav Joshi	2025-12-05

12. Calculate the Average Ticket Price for Events in Each Venue Using a Subquery

```
select v.venue_name,
       select round(avg(e.ticket_price),2)
  from event e where e.venue_id=v.venue_id) as avgticket_price
   from venue v;
```

The screenshot shows a SQL interface with two panes. The top pane, titled "SQL File 1*", contains the SQL query:

```
1 • 1. select v.venue_name,
2      select round(avg(e.ticket_price),2)
3      from event e
4     where e.venue_id=v.venue_id)as avgticket_price
5      from venue v;
```

The bottom pane, titled "Result Grid", displays the results of the query:

venue_name	avgticket_price
Grand Arena	186.25
Skyline Stadium	160.25
Harmony Hall	489.99
Star Cinemas	135.00
City Convention Ctr	150.00
Green Park Theatre	475.00
Ocean View Grounds	300.00
Metro Multiplex	100.00
Diamond Club	HULL
Royal Event Space	HULL

The "Result 68" pane shows the execution details:

#	Time	Action	Message
1	17:26:13	select v.venue_name,(select round(avg(e.ticket_price),2) from event e where e.venue...	10 row(s) returned