

Tasks 1: Database Design:

Q1. Create the database named "TicketBookingSystem"

The screenshot shows the MySQL Workbench interface with the following SQL code in the Query 1 editor:

```
L488
L489 •  create Database TicketBs;
L490 •  use TicketBs;
L491
L492 •  CREATE TABLE Venue(
L493     venue_id int primary key,
L494     venue_name text,
L495     address text
L496 );
L497 •  CREATE TABLE Event(
L498     event_id int primary key,
L499     event_name text,
L500     event_date date,
L501     event_time time,
L502     venue_id int,
L503     total_seats int,
L504     available_seats int,
L505     ticket_price double,
L506     event_type enum('Music', 'Comedy', 'Concert')
```

The Output pane shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	18:50:24	show tables	2 rows(s) returned	0.062 sec / 0.000 sec
2	18:50:44	create database PetPals	Error Code: 1007. Can't create database 'petpals'; database exists	0.047 sec
3	18:50:59	drop table venu	0 row(s) affected	0.078 sec
4	18:51:11	show tables	1 row(s) returned	0.000 sec / 0.000 sec
5	18:53:41	CREATE TABLE Event(event_id int primary key, event_name text, event_date date, event_time time, venue_id int, tot...)	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server ver... 0.000 sec	

Q2. Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships.

- Venu
- Event
- Customers
- Booking

The screenshot shows the MySQL Workbench interface with the following SQL code in the Query 1 editor:

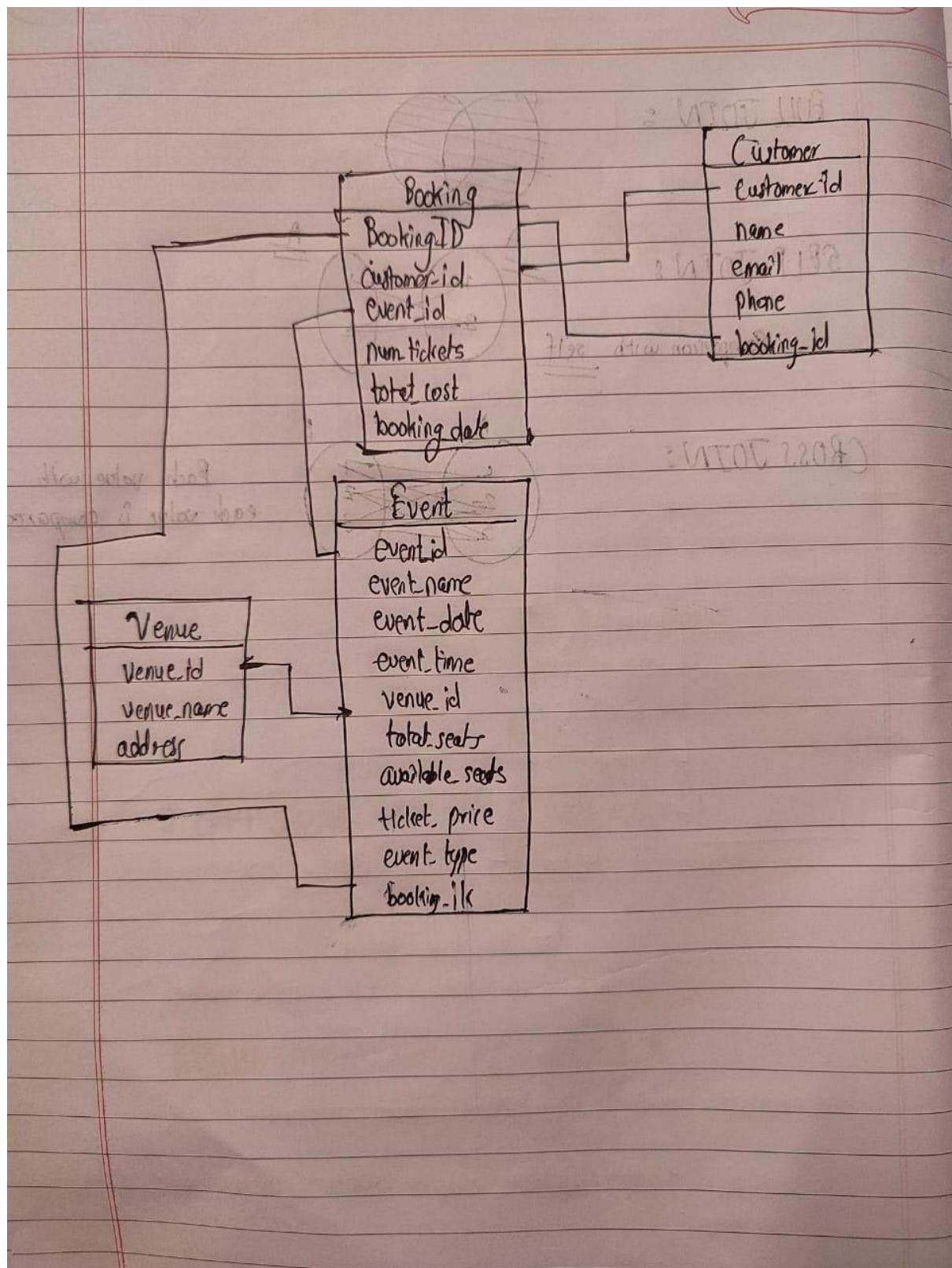
```
L519
L520 );
L521 •  CREATE TABLE Booking(
L522     booking_id int primary key,
L523     customer_id int,
L524     event_id int,
L525     num_tickets int,
L526     total_cost double,
L527     booking_date date,
L528     foreign key (customer_id) references Customer(customer_id),
L529     foreign key (event_id) references Event(event_id)
L530 );

L531
L532 •  ALTER TABLE Event
L533     ADD FOREIGN KEY (booking_id) REFERENCES Booking(booking_id);
L534
L535 •  ALTER TABLE Customer
L536     ADD FOREIGN KEY (booking_id) REFERENCES Booking(booking_id);
```

The Output pane shows the execution results:

#	Time	Action	Message	Duration / Fetch
9	18:55:24	CREATE TABLE Booking(booking_id int primary key, customer_id int, event_id int, num_tickets int, total_cost double, ...)	0 row(s) affected	0.047 sec
10	18:55:34	ALTER TABLE student ADD FOREIGN KEY (booking_id) REFERENCES Booking(booking_id)	Error Code: 1146. Table 'student' doesn't exist	0.000 sec
11	18:55:52	ALTER TABLE Event ADD FOREIGN KEY (booking_id) REFERENCES Booking(booking_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.054 sec
12	18:56:13	ALTER TABLE Customer ADD FOREIGN KEY (booking_id) REFERENCES Booking(booking_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.054 sec

Q3. Create an ERD (Entity Relationship Diagram) for the database.



Q4. Create appropriate Primary Key and Foreign Key constraints for referential integrity

The screenshot shows the MySQL Workbench interface with a query editor window titled "Query1". The code entered is as follows:

```
L519  booking_id int
);
CREATE TABLE Booking(
L520    booking_id int primary key,
L521    customer_id int,
L522    event_id int,
L523    num_tickets int,
L524    total_cost double,
L525    booking_date date,
L526
L527    foreign key (customer_id)references Customer(customer_id),
L528    foreign key (event_id) references Event(event_id)
);
L529
L530
L531
L532 • ALTER TABLE Event
L533   ADD FOREIGN KEY (booking_id) REFERENCES Booking(booking_id);
L534
L535 • ALTER TABLE Customer
L536   ADD FOREIGN KEY (booking_id) REFERENCES Booking(booking_id);
```

The "Output" pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
9	18:55:24	CREATE TABLE Booking(booking_id int primary key, customer_id int, event_id int, num_tickets int, total_cost double, ...)	0 row(s) affected	0.047 sec
10	18:55:34	ALTER TABLE student ADD FOREIGN KEY (booking_id) REFERENCES Booking(booking_id)	Error Code: 1146. Table 'ticketbs.student' doesn't exist	0.000 sec
11	18:55:52	ALTER TABLE Event ADD FOREIGN KEY (booking_id) REFERENCES Booking(booking_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.094 sec
12	18:56:13	ALTER TABLE Customer ADD FOREIGN KEY (booking_id) REFERENCES Booking(booking_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.094 sec

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

Q1. Write a SQL query to insert at least 10 sample records into each table.

```
L552     (2, 'Football Match', '2023-02-20', '20:00:00', 2, 500, 500, 25.00, 'Sports', NULL),
L553     (3, 'Concert Live', '2023-03-25', '19:30:00', 3, 300, 300, 50.00, 'Concert', NULL),
L554     (4, 'Cinema Premiere', '2023-04-10', '15:00:00', 4, 150, 150, 12.00, 'Movie', NULL),
L555     (5, 'Basketball Game', '2023-05-05', '17:30:00', 2, 200, 200, 20.00, 'Sports', NULL),
L556     (6, 'Music Festival', '2023-06-15', '16:00:00', 3, 400, 400, 30.00, 'Concert', NULL),
L557     (7, 'Outdoor Movie', '2023-07-20', '20:30:00', 5, 120, 120, 15.00, 'Movie', NULL);

L558 • INSERT INTO Customer (customer_id, name, email, phone, booking_id)
VALUES
L559     (1, 'Alice Johnson', 'alice@email.com', 1234567890, NULL),
L560     (2, 'Bob Smith', 'bob@email.com', 9876543210, NULL),
L561     (3, 'Charlie Brown', 'charlie@email.com', 5551112233, NULL),
L562     (4, 'David Lee', 'david@email.com', 4445556666, NULL),
L563     (5, 'Emily Davis', 'emily@email.com', 7778889999, NULL),
L564     (6, 'Frank Miller', 'frank@email.com', 1112223333, NULL),
L565     (7, 'Grace Taylor', 'grace@email.com', 2223334444, NULL);

L566 • INSERT INTO Booking (booking_id, customer_id, event_id, num_tickets, total_cost, booking_date)
VALUES
```

Q2. Write a SQL query to list all Events.

```
L572     (2, 2, 4, 3, 36.00, '2023-02-25'),
L573     (3, 3, 6, 1, 30.00, '2023-03-30'),
L574     (4, 4, 1, 4, 40.00, '2023-04-30'),
L575     (5, 5, 3, 2, 100.00, '2023-05-30'),
L576     (6, 6, 5, 1, 20.00, '2023-06-30'),
L577     (7, 7, 7, 3, 45.00, '2023-07-30');

L578 • SELECT * FROM Event;
```

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
1	Movie Night	2023-01-15	18:00:00	4	100	100	10	Movie	NULL
2	Football Match	2023-02-20	20:00:00	2	500	500	25	Sports	NULL
3	Concert Live	2023-03-25	19:30:00	3	300	300	50	Concert	NULL
4	Cinema Premiere	2023-04-10	15:00:00	4	150	150	12	Movie	NULL
5	Basketball Game	2023-05-05	17:30:00	2	200	200	20	Sports	NULL
6	Music Festival	2023-06-15	16:00:00	3	400	400	30	Concert	NULL
7	Outdoor Movie	2023-07-20	20:30:00	5	120	120	15	Movie	NULL

Q3. Write a SQL query to select events with available tickets.

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

```
L575    (5, 5, 3, 2, 100.00, '2023-05-30'),
L576    (6, 6, 5, 1, 20.00, '2023-06-30'),
L577    (7, 7, 7, 3, 45.00, '2023-07-30');

INSTANCE ▾
L578 • SELECT * FROM Event;

L581
L582
L583 • SELECT *
```

The results grid shows 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	2023-01-15	18:00:00	4	100	100	10	Movie	
2	Football Match	2023-02-20	20:00:00	2	500	500	25	Sports	
3	Concert Live	2023-03-25	19:30:00	3	300	300	50	Concert	
4	Cinema Premier	2023-04-10	15:00:00	4	150	150	12	Movie	
5	Basketball Game	2023-05-05	17:30:00	2	200	200	20	Sports	
6	Music Festival	2023-06-15	16:00:00	3	400	400	30	Concert	
7	Outdoor Movie	2023-07-20	20:30:00	5	120	120	15	Movie	

The status bar at the bottom right shows the time as 7:31 PM and the date as 12/12/2023.

Q4. Write a SQL query to select events name partial match with 'cup'.

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

```
L582
L583 • SELECT *
FROM Event
WHERE available_seats > 0;
L584
L585
L586
INSTANCE ▾
L587 • SELECT *
FROM Event
WHERE event_name LIKE '%cup%';
L588
L589
L590
```

The results grid shows the following data:

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
*	*	*	*	*	*	*	*	*	*

The status bar at the bottom right shows the time as 7:33 PM and the date as 12/12/2023.

Q5. Write a SQL query to select events with ticket price range is between 1000 to 2500.

MySQL Workbench - Local instance MySQL80

Query 1

```

L582 •  SELECT *
  FROM Event
 WHERE available_seats > 0;
L586
L587 •  SELECT *
  FROM Event
 WHERE event_name LIKE '%cup%';
L588
L589
L590
  
```

Result Grid

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
1	Football Cup	2023-03-10	19:00:00	1	1000	1000	100	Sports	1

Information

No object selected

Action Output

#	Time	Action	Message	Duration / Fetch
1	19:33:22	SELECT * FROM Event WHERE event_name LIKE "%cup%" LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

Object Info

ENG IN 7:33 PM 12/12/2023

Q6. Write a SQL query to retrieve events with dates falling within a specific range.

MySQL Workbench - Local instance MySQL80

Query 1

```

L581
L582
L583 •  SELECT * FROM Event WHERE available_seats > 0;
L584
L585 •  SELECT * FROM Event WHERE event_name LIKE '%cup%';
L586
L587 •  SELECT * FROM Event WHERE ticket_price BETWEEN 1000 AND 2500;
L588
L589 •  SELECT * FROM Event WHERE event_date BETWEEN '2023-03-01' AND '2023-06-30';
  
```

Result Grid

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
3	Concert Live	2023-03-25	19:30:00	3	300	300	50	Concert	1
4	Cinema Premiere	2023-04-10	15:00:00	4	150	150	12	Movie	2
5	Basketball Game	2023-05-05	17:30:00	2	200	200	20	Sports	3
6	Music Festival	2023-06-15	16:00:00	3	400	400	30	Concert	4

Information

No object selected

Action Output

#	Time	Action	Message	Duration / Fetch
1	19:33:22	SELECT * FROM Event WHERE event_name LIKE "%cup%" LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
2	19:36:13	SELECT * FROM Event WHERE ticket_price BETWEEN 1000 AND 2500 LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
3	19:41:08	SELECT * FROM Event WHERE event_date BETWEEN '2023-03-01' AND '2023-06-30' LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec

Object Info

ENG IN 7:41 PM 12/12/2023

Q7. Write a SQL query to retrieve events with available tickets that also have "Concert" in their name.

MySQL Workbench - Local instance MySQL80

Query 1:

```

L583 • SELECT * FROM Event WHERE available_seats > 0;
L584
L585 • SELECT * FROM Event WHERE event_name LIKE '%cup%';
L586
L587 • SELECT * FROM Event WHERE ticket_price BETWEEN 1000 AND 2500;
L588
L589 • SELECT * FROM Event WHERE event_date BETWEEN '2023-03-01' AND '2023-06-30';
L590
L591 • SELECT *
  
```

Result Grid:

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
3	ConcertLive	2023-03-25	19:30:00	3	300	300	50	Concert	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Information:

Event 197 x

No object selected

Action Output

#	Time	Action	Message	Duration / Fetch
1	19:33:22	SELECT * FROM Event WHERE event_name LIKE '%cup%' LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
2	19:36:13	SELECT * FROM Event WHERE ticket_price BETWEEN 1000 AND 2500 LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
3	19:41:08	SELECT * FROM Event WHERE event_date BETWEEN '2023-03-01' AND '2023-06-30' LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
4	19:42:24	SELECT * FROM Event WHERE available_seats > 0 AND event_name LIKE '%Concert%' LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Q8. Write a SQL query to retrieve users in batches of 5, starting from the 6th user.

MySQL Workbench - Local instance MySQL80

Query 1:

```

L588
L589 • SELECT * FROM Event WHERE event_date BETWEEN '2023-03-01' AND '2023-06-30';
L590
L591 • SELECT *
L592 FROM Event
L593 WHERE available_seats > 0 AND event_name LIKE '%Concert%';
L594
L595
L596 • SELECT * FROM Customer ORDER BY customer_id LIMIT 5 OFFSET 5;
  
```

Result Grid:

customer_id	name	email	phone	booking_id
6	Frank Miller	frank@email.com	1112223333	NULL
7	Grace Taylor	grace@email.com	2223344444	NULL
*	NULL	NULL	NULL	NULL

Information:

Customer 198 x

No object selected

Action Output

#	Time	Action	Message	Duration / Fetch
3	19:41:08	SELECT * FROM Event WHERE event_date BETWEEN '2023-03-01' AND '2023-06-30' LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
4	19:42:24	SELECT * FROM Event WHERE available_seats > 0 AND event_name LIKE '%Concert%' LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
5	19:48:35	SELECT * FROM Users ORDER BY UserID LIMIT 5 OFFSET 5	Error Code: 1146. Table 'ticketbs.users' doesn't exist	0.000 sec
6	19:49:24	SELECT * FROM Customer ORDER BY customer_id LIMIT 5 OFFSET 5	2 row(s) returned	0.000 sec / 0.000 sec

Q9. Write a SQL query to retrieve bookings details contains booked no of ticket more than 1.

The screenshot shows the MySQL Workbench interface with several tabs open:

- Local instance MySQL80 x**: The main workspace showing three queries in the SQL Editor:

 - Query 1 (MANAGEMENT):

```
SELECT * FROM Event WHERE available_seats > 0 AND event_name LIKE '%Concert%';
```
 - Query 2 (INSTANCE):

```
SELECT * FROM Customer ORDER BY customer_id LIMIT 5 OFFSET 5;
```
 - Query 3 (PERFORMANCE):

```
SELECT * FROM Booking WHERE num_tickets > 1;
```

- SQLAdditions**: A toolbar for navigating between queries.
- Result Grid**: A preview of the data from the third query.
- Administration**: A sidebar with various management options.
- Information**: A sidebar showing the current connection details.
- Action Output**: A log of recent actions and errors:

 - 5 19:48:35 SELECT * FROM Users ORDER BY UserID LIMIT 5 OFFSET 5
 - 6 19:49:24 SELECT * FROM Customer ORDER BY customer_id LIMIT 5 OFFSET 5
 - 7 19:52:14 SELECT * FROM Booking WHERE num_tickets > 4 LIMIT 0, 1000
 - 8 19:52:27 SELECT * FROM Booking WHERE num_tickets > 1 LIMIT 0, 1000

- Object Info**: A sidebar showing the selected object is "No object selected".

Q10. Write a SQL query to retrieve customer information whose phone number end with '000'

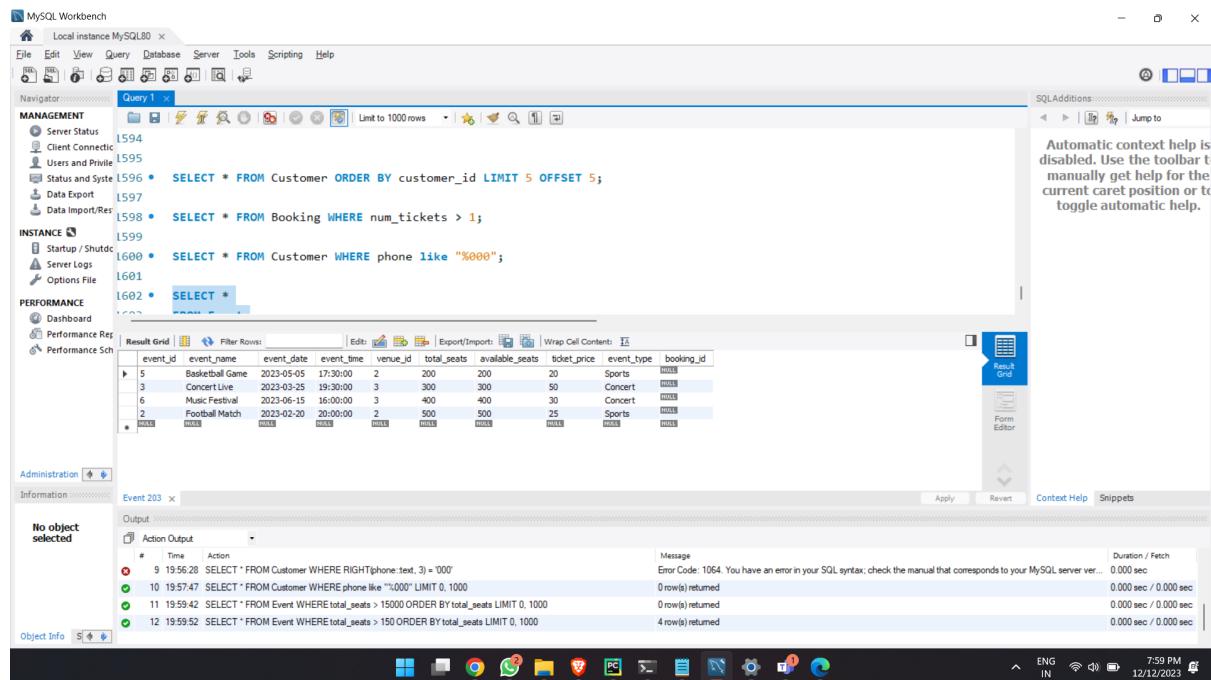
The screenshot shows the MySQL Workbench interface with several windows open:

- MySQL Workbench**: The main window title bar.
- Local instance MySQL80**: A sub-tab under the main window.
- Management**: Shows server status, client connectivity, users and privileges, and data import/export.
- INSTANCE**: Shows startup/shutdown logs and options file.
- PERFORMANCE**: Shows dashboard, performance reports, and schema monitors.
- Administration**: An open panel showing the current schema (Customer 201) and object selection.
- Information**: An open panel showing the results of a query: "SELECT * FROM Customer WHERE phone like '%000'".
- Action Output**: A table showing the execution history of the current session:

#	Time	Action	Message	Duration / Fetch
7	19:52:14	SELECT * FROM Booking WHERE num_tickets > 4 LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
8	19:52:27	SELECT * FROM Booking WHERE num_tickets > 1 LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
9	19:56:28	SELECT * FROM Customer WHERE RIGHT(phone, 3) = '000'	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '000' at line 1	0.000 sec
10	19:57:47	SELECT * FROM Customer WHERE phone like "%000" LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

- SQLAdditions**: A toolbar with various icons for managing queries.
- Jump to**: A button to jump to the current caret position.
- Object Info**: A small panel showing the current object information.
- Bottom Bar**: Standard Windows-style icons for file operations.
- Status Bar**: Shows the date and time (12/12/2023, 7:57 PM).

Q11. Write a SQL query to retrieve the events in order whose seat capacity more than 150.



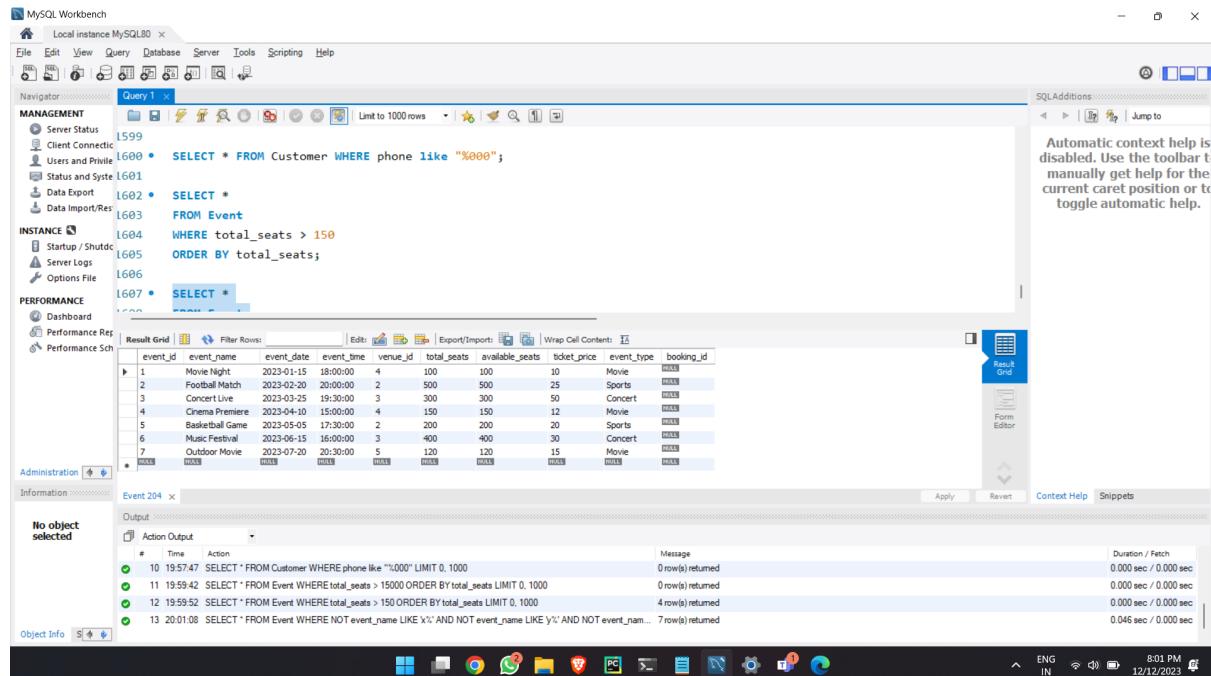
The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database structure with tables like Customer, Booking, and Event.
- Query Editor (Query 1):**

```
L594
L595
L596 •  SELECT * FROM Customer ORDER BY customer_id LIMIT 5 OFFSET 5;
L597
L598 •  SELECT * FROM Booking WHERE num_tickets > 1;
L599
L600 •  SELECT * FROM Customer WHERE phone like "%000";
L601
L602 •  SELECT *
```
- Result Grid:** Displays the results of the last query, showing event details for events with more than 150 seats. The data includes:

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
5	Basketball Game	2023-05-05	17:30:00	2	200	200	20	Sports	
3	Concert Live	2023-03-25	19:30:00	3	300	300	50	Concert	
6	Music Festival	2023-06-15	16:00:00	3	400	400	30	Concert	
2	Football Match	2023-02-20	20:00:00	2	500	500	25	Sports	
- Output Panel:** Shows the execution history and log messages. One message indicates an error: "Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'SELECT * FROM Event WHERE total_seats > 15000 ORDER BY total_seats LIMIT 0, 1000' at line 1".

Q12. Write a SQL query to select events name not start with 'x', 'y', 'z'



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

- Navigator:** Shows the database structure with tables like Customer, Booking, and Event.
- Query Editor (Query 1):**

```
L599
L600 •  SELECT * FROM Customer WHERE phone like "%000";
L601
L602 •  SELECT *
FROM Event
WHERE total_seats > 150
ORDER BY total_seats;
```
- Result Grid:** Displays the results of the last query, showing event details for events with more than 150 seats. The data includes:

event_id	event_name	event_date	event_time	venue_id	total_seats	available_seats	ticket_price	event_type	booking_id
1	Movie Night	2023-01-15	18:00:00	4	100	100	10	Movie	
2	Football Match	2023-02-20	20:00:00	2	500	500	25	Sports	
3	Concert Live	2023-03-25	19:30:00	3	300	300	50	Concert	
4	Cinema Premiere	2023-04-10	15:00:00	4	150	150	12	Movie	
5	Basketball Game	2023-05-05	17:30:00	2	200	200	20	Sports	
6	Music Festival	2023-06-15	16:00:00	3	400	400	30	Concert	
7	Outdoor Movie	2023-07-20	20:30:00	5	120	120	15	Movie	
- Output Panel:** Shows the execution history and log messages. One message indicates an error: "Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'SELECT * FROM Event WHERE NOT event_name LIKE 'x%' AND NOT event_name LIKE 'y%' AND NOT event_name LIKE 'z%" at line 1".

Tasks 3: Aggregate functions, Having, Order By, GroupBy and Joins:

Q1. Write a SQL query to List Events and Their Average Ticket Prices.

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

```
SELECT e.event_id, e.event_name, AVG(b.total_cost / b.num_tickets) AS average_ticket_price
FROM Event e
JOIN Booking b ON e.event_id = b.event_id
GROUP BY e.event_id, e.event_name;
```

The results grid displays the following data:

event_id	event_name	average_ticket_price
1	Movie Night	10
2	Football Match	25
3	Concert Live	50
4	Cinema Premiere	12
5	Basketball Game	20
6	Music Festival	30
7	Outdoor Movie	15

The status bar at the bottom right shows the date and time as 12/12/2023 10:25 PM.

Q2. Write a SQL query to Calculate the Total Revenue Generated by Events.

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

```
SELECT e.event_id, e.event_name, SUM(b.total_cost) AS total_revenue
FROM Event e
JOIN Booking b ON e.event_id = b.event_id
GROUP BY e.event_id, e.event_name;
```

The results grid displays the following data:

event_id	event_name	total_revenue
1	Movie Night	40
2	Football Match	50
3	Concert Live	100
4	Cinema Premiere	36
5	Basketball Game	20
6	Music Festival	30
7	Outdoor Movie	45

The status bar at the bottom right shows the date and time as 12/12/2023 10:27 PM.

Q3. Write a SQL query to find the event with the highest ticket sales.

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

- Query Editor:** Displays the following SQL code:

```
L621 GROUP BY e.event_id, e.event_name;
L622
L623 • 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
ORDER BY total_tickets_sold DESC LIMIT 1;
```
- Result Grid:** Shows the output of the query:

event_id	event_name	total_tickets_sold
1	Movie Night	4
- Output Window:** Displays the execution log:

#	Time	Action	Message	Duration / Fetch
1	22:25:05	SELECT e.event_id, e.event_name, AVG(b.total_cost / b.num_tickets) AS average_ticket_price FROM Event e JOIN Booking b ON e.event_id = b.event_id GROUP BY e.event_id, e.event_name ORDER BY average_ticket_price ASC LIMIT 1;	7 row(s) returned	0.015 sec / 0.000 sec
2	22:26:44	SELECT e.event_id, e.event_name, SUM(b.total_cost) AS total_revenue FROM Event e JOIN Booking b ON e.event_id = b.event_id GROUP BY e.event_id, e.event_name ORDER BY total_revenue DESC LIMIT 1;	7 row(s) returned	0.000 sec / 0.000 sec
3	22:29:07	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 ORDER BY total_tickets_sold DESC LIMIT 1;	1 row(s) returned	0.016 sec / 0.000 sec

Q4. Write a SQL query to Calculate the Total Number of Tickets Sold for Each Event.

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

- Query Editor:** Displays the following SQL code:

```
L625 JOIN Booking b ON e.event_id = b.event_id
L626 GROUP BY e.event_id, e.event_name
L627 ORDER BY total_tickets_sold DESC LIMIT 1;
L628
L629 • 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;
```
- Result Grid:** Shows the output of the query:

event_id	event_name	total_tickets_sold
1	Movie Night	4
2	Football Match	2
3	Concert Live	2
4	Cinema Premiere	3
5	Basketball Game	1
6	Music Festival	1
7	Outdoor Movie	3
- Output Window:** Displays the execution log:

#	Time	Action	Message	Duration / Fetch
1	22:25:05	SELECT e.event_id, e.event_name, AVG(b.total_cost / b.num_tickets) AS average_ticket_price FROM Event e JOIN Booking b ON e.event_id = b.event_id GROUP BY e.event_id, e.event_name ORDER BY average_ticket_price ASC LIMIT 1;	7 row(s) returned	0.015 sec / 0.000 sec
2	22:26:44	SELECT e.event_id, e.event_name, SUM(b.total_cost) AS total_revenue FROM Event e JOIN Booking b ON e.event_id = b.event_id GROUP BY e.event_id, e.event_name ORDER BY total_revenue DESC LIMIT 1;	7 row(s) returned	0.000 sec / 0.000 sec
3	22:29:07	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 ORDER BY total_tickets_sold DESC LIMIT 1;	1 row(s) returned	0.016 sec / 0.000 sec
4	22:30:38	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 ORDER BY total_tickets_sold DESC LIMIT 1;	7 row(s) returned	0.000 sec / 0.000 sec

Q5. Write a SQL query to Find Events with No Ticket Sales.

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

- Query Editor:** Displays the following SQL code:

```
L631 JOIN Booking b ON e.event_id = b.event_id
L632 GROUP BY e.event_id, e.event_name;
L634 • SELECT e.event_id, e.event_name
L635 FROM Event e
L636 LEFT JOIN Booking b ON e.event_id = b.event_id
L637 WHERE b.booking_id IS NULL;
L638
L639
```
- Result Grid:** Shows the output of the query, which is an empty table with columns `event_id` and `event_name`.
- Action Output:** Shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
1	22:26:44	SELECT e.event_id, e.event_name, SUM(b.total_cost) AS total_revenue FROM Event e JOIN Booking b ON e.event_id = b.event_id GROUP BY e.event_id, e.event_name;	7 rows(s) returned	0.000 sec / 0.000 sec
2	22:29:07	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;	1 row(s) returned	0.016 sec / 0.000 sec
3	22:30:38	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;	7 row(s) returned	0.000 sec / 0.000 sec
4	22:32:30	SELECT e.event_id, e.event_name FROM Event e LEFT JOIN Booking b ON e.event_id = b.event_id WHERE b.booking_id IS NULL;	0 row(s) returned	0.000 sec / 0.000 sec

Q6. Write a SQL query to Find the User Who Has Booked the Most Tickets.

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

- Query Editor:** Displays the following SQL code:

```
L637 WHERE b.booking_id IS NULL;
L638
L639 • SELECT c.customer_id, c.name, COUNT(b.booking_id) AS total_tickets_booked
L640 FROM Customer c
L641 JOIN Booking b ON c.customer_id = b.customer_id
L642 GROUP BY c.customer_id, c.name
L643 ORDER BY total_tickets_booked DESC LIMIT 1;
```
- Result Grid:** Shows the output of the query, which is a table with columns `customer_id`, `name`, and `total_tickets_booked`. It contains one row: `Alice Johnson 1`.
- Action Output:** Shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
1	22:29:07	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;	7 rows(s) returned	0.016 sec / 0.000 sec
2	22:30:38	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;	1 row(s) returned	0.000 sec / 0.000 sec
3	22:32:30	SELECT e.event_id, e.event_name FROM Event e LEFT JOIN Booking b ON e.event_id = b.event_id WHERE b.booking_id IS NULL;	0 row(s) returned	0.000 sec / 0.000 sec
4	22:34:27	SELECT c.customer_id, c.name, COUNT(b.booking_id) AS total_tickets_booked FROM Customer c JOIN Booking b ON c.customer_id = b.customer_id GROUP BY c.customer_id, c.name ORDER BY total_tickets_booked DESC LIMIT 1;	1 row(s) returned	0.047 sec / 0.000 sec

Q7. Write a SQL query to List Events and the total number of tickets sold for each month.

The screenshot shows the MySQL Workbench interface with a query editor and results grid. The query retrieves event details and ticket sales per month:

```
L642 GROUP BY c.customer_id, c.name
L643 ORDER BY total_tickets_booked DESC LIMIT 1;
L645 • SELECT e.event_id, e.event_name, MONTH(b.booking_date) AS booking_month, YEAR(b.booking_date) AS booking_year, COUNT(b.book
FROM Event e
JOIN Booking b ON e.event_id = b.event_id
GROUP BY e.event_id, e.event_name, booking_month, booking_year
ORDER BY booking_year, booking_month, total_tickets_sold DESC;
```

The results grid displays the following data:

event_id	event_name	booking_month	booking_year	total_tickets_sold
2	Football Match	1	2023	1
4	Cinema Premiere	2	2023	1
6	Music Festival	3	2023	1
1	Movie Night	4	2023	1
3	Concert Live	5	2023	1
5	Basketball Game	6	2023	1
7	Outdoor Movie	7	2023	1

The output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
4	22:30:38	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 WHERE e.event_name = 'Football Match' GROUP BY e.event_id, e.event_name, b.booking_date ORDER BY b.booking_date DESC LIMIT 1;	7 row(s) returned	0.000 sec / 0.000 sec
5	22:32:30	SELECT e.event_id, e.event_name FROM Event e LEFT JOIN Booking b ON e.event_id = b.event_id WHERE e.event_name = 'Football Match' GROUP BY e.event_id, e.event_name, b.booking_date ORDER BY b.booking_date DESC LIMIT 1;	0 row(s) returned	0.000 sec / 0.000 sec
6	22:34:27	SELECT c.customer_id, c.name, COUNT(b.booking_id) AS total_tickets_booked FROM Customer c JOIN Booking b ON c.customer_id = b.customer_id WHERE c.name = 'John Doe' GROUP BY c.customer_id, c.name ORDER BY total_tickets_booked DESC LIMIT 1;	1 row(s) returned	0.047 sec / 0.000 sec
7	22:36:45	SELECT e.event_id, e.event_name, MONTH(b.booking_date) AS booking_month, YEAR(b.booking_date) AS booking_year, COUNT(b.book	7 row(s) returned	0.000 sec / 0.000 sec

Q8. Write a SQL query to calculate the average Ticket Price for Events in Each Venue.

The screenshot shows the MySQL Workbench interface with a query editor and results grid. The query retrieves average ticket prices by venue:

```
L648 GROUP BY e.event_id, e.event_name, booking_month, booking_year
L649 ORDER BY booking_year, booking_month, total_tickets_sold DESC;
L651
L652 • SELECT v.venue_id, v.venue_name, AVG(e.ticket_price) AS average_ticket_price
FROM Venue v
JOIN Event e ON v.venue_id = e.venue_id
GROUP BY v.venue_id, v.venue_name;
```

The results grid displays the following data:

venue_id	venue_name	average_ticket_price
2	Sports Arena	22.5
3	Concert Hall	40
4	Cinema Plex	11
5	Community Center	15

The output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	22:39:10	SELECT v.venue_id, v.venue_name, AVG(e.ticket_price) AS average_ticket_price FROM Venue v JOIN Event e ON v.venue_id = e.venue_id GROUP BY v.venue_id, v.venue_name;	4 row(s) returned	0.047 sec / 0.000 sec

Q9. Write a SQL query to calculate the total Number of Tickets Sold for Each Event Type.

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

- Query Editor:** Displays the following SQL code:

```
L653     FROM Venue v
L654   JOIN Event e ON v.venue_id = e.venue_id
L655   GROUP BY v.venue_id, v.venue_name;
L656
L657 •   SELECT e.event_type, SUM(b.num_tickets) AS total_tickets_sold
L658   FROM Event e
L659   JOIN Booking b ON e.event_id = b.event_id
L660   GROUP BY e.event_type;
L661
```
- Result Grid:** Shows the output of the query:

event_type	total_tickets_sold
Movie	10
Sports	3
Concert	3
- Output Window:** Displays the execution log:

```
# Time Action
1 22:39:10 SELECT v.venue_id, v.venue_name, AVG(e.ticket_price) AS average_ticket_price FROM Venue v JOIN Event e ON v.venue_id = e.venue_id WHERE v.venue_name = 'Theater'; 4 row(s) returned
0.047 sec / 0.000 sec
2 22:41:04 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 WHERE e.event_type = 'Sports'; 3 row(s) returned
0.000 sec / 0.000 sec
3 22:42:40 SELECT YEAR(b.booking_date) AS event_year, SUM(e.ticket_price * b.num_tickets) AS total_revenue FROM Event e JOIN Booking b ON e.event_id = b.event_id WHERE b.booking_date IS NOT NULL; 1 row(s) returned
0.000 sec / 0.000 sec
```

Q10. Write a SQL query to calculate the total Revenue Generated by Events in Each Year.

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

- Query Editor:** Displays the following SQL code:

```
L660   GROUP BY e.event_type;
L661
L662 •   SELECT YEAR(b.booking_date) AS event_year, SUM(e.ticket_price * b.num_tickets) AS total_revenue
L663   FROM Event e
L664   JOIN Booking b ON e.event_id = b.event_id
L665   WHERE b.booking_date IS NOT NULL
L666   GROUP BY event_year;
L667
L668
```
- Result Grid:** Shows the output of the query:

event_year	total_revenue
2023	321
- Output Window:** Displays the execution log:

```
# Time Action
1 22:39:10 SELECT v.venue_id, v.venue_name, AVG(e.ticket_price) AS average_ticket_price FROM Venue v JOIN Event e ON v.venue_id = e.venue_id WHERE v.venue_name = 'Theater'; 4 row(s) returned
0.047 sec / 0.000 sec
2 22:41:04 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 WHERE e.event_type = 'Sports'; 3 row(s) returned
0.000 sec / 0.000 sec
3 22:42:40 SELECT YEAR(b.booking_date) AS event_year, SUM(e.ticket_price * b.num_tickets) AS total_revenue FROM Event e JOIN Booking b ON e.event_id = b.event_id WHERE b.booking_date IS NOT NULL; 1 row(s) returned
0.000 sec / 0.000 sec
```

Q11. Write a SQL query to list users who have booked tickets for multiple events.

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

- Query Editor:** Shows a multi-line query window with the following SQL code:

```
SELECT c.customer_id, c.name AS customer_name, COUNT(DISTINCT b.event_id) AS num_events_booked
FROM Customer c
JOIN Booking b ON c.customer_id = b.customer_id
GROUP BY c.customer_id, c.name
HAVING num_events_booked > 1;
```
- Result Grid:** Displays the results of the query in a tabular format with columns: customer_id, customer_name, and num_events_booked. The result is empty (0 rows).
- Output Window:** Shows the execution log with one entry:

#	Time	Action	Message	Duration / Fetch
1	22:44:56	SELECT c.customer_id, c.name AS customer_name, COUNT(DISTINCT b.event_id) AS num_events_booked F...	0 row(s) returned	0.000 sec / 0.000 sec

Q12. Write a SQL query to calculate the Total Revenue Generated by Events for Each User.

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

- Query Editor:** Shows a modified version of the previous query:

```
HAVING num_events_booked > 1;
SELECT c.customer_id, c.name AS customer_name, SUM(b.total_cost) AS total_revenue
FROM Customer c
JOIN Booking b ON c.booking_id = b.booking_id
GROUP BY c.customer_id, c.name;
```
- Result Grid:** Displays the results of the query in a tabular format with columns: customer_id, customer_name, and total_revenue. The result is empty (0 rows).
- Output Window:** Shows the execution log with two entries:

#	Time	Action	Message	Duration / Fetch
1	22:44:56	SELECT c.customer_id, c.name AS customer_name, COUNT(DISTINCT b.event_id) AS num_events_booked F...	0 row(s) returned	0.000 sec / 0.000 sec
2	22:46:28	SELECT c.customer_id, c.name AS customer_name, SUM(b.total_cost) AS total_revenue FROM Customer c J...	0 row(s) returned	0.000 sec / 0.000 sec

Q13. Write a SQL query to calculate the Average Ticket Price for Events in Each Category and Venue.

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

- Query Editor:** Displays two queries. The first query calculates total revenue per customer:


```
L674 • SELECT c.customer_id, c.name AS customer_name, SUM(b.total_cost) AS total_revenue
      FROM Customer c
      JOIN Booking b ON c.booking_id = b.booking_id
      GROUP BY c.customer_id, c.name;
```
- Result Grid:** Shows the output of the first query:

event_type	venue_name	average_ticket_price
Movie	Cinema Plex	11
Sports	Sports Arena	22.5
Concert	Concert Hall	40
Movie	Community Center	15
- Output Window:** Shows the execution log with three entries:

#	Time	Action	Message	Duration / Fetch
1	22:44:56	SELECT c.customer_id, c.name AS customer_name, COUNT(DISTINCT b.event_id) AS num_events_booked ... 0 row(s) returned		0.000 sec / 0.000 sec
2	22:46:28	SELECT c.customer_id, c.name AS customer_name, SUM(b.total_cost) AS total_revenue FROM Customer c ... 0 row(s) returned		0.000 sec / 0.000 sec
3	22:48:45	SELECT e.event_type, v.venue_name, AVG(e.ticket_price) AS average_ticket_price FROM Event e JOIN ... 4 row(s) returned		0.000 sec / 0.000 sec

Q14. Write a SQL query to list Users and the Total Number of Tickets They've Purchased in the Last 30 Days.

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

- Query Editor:** Displays a query to list users and their total tickets purchased in the last 30 days:


```
L688
      e.event_type, v.venue_name;
L689
L690 • SELECT c.customer_id, c.name AS user_name, COUNT(b.num_tickets) AS total_tickets_purchased
      FROM Customer c
      JOIN Booking b ON c.customer_id = b.customer_id
      WHERE b.booking_date >= CURDATE() - INTERVAL 30 DAY
      GROUP BY c.customer_id, c.name;
```
- Result Grid:** Shows the output of the query:

customer_id	user_name	total_tickets_purchased
-------------	-----------	-------------------------
- Output Window:** Shows the execution log with four entries:

#	Time	Action	Message	Duration / Fetch
1	22:44:56	SELECT c.customer_id, c.name AS customer_name, COUNT(DISTINCT b.event_id) AS num_events_booked ... 0 row(s) returned		0.000 sec / 0.000 sec
2	22:46:28	SELECT c.customer_id, c.name AS customer_name, SUM(b.total_cost) AS total_revenue FROM Customer c ... 0 row(s) returned		0.000 sec / 0.000 sec
3	22:48:45	SELECT e.event_type, v.venue_name, AVG(e.ticket_price) AS average_ticket_price FROM Event e JOIN ... 4 row(s) returned		0.000 sec / 0.000 sec
4	22:50:54	SELECT c.customer_id, c.name AS user_name, COUNT(b.num_tickets) AS total_tickets_purchased FROM Customer c ... 0 row(s) returned		0.000 sec / 0.000 sec

Tasks 4: Subquery and its types

Q1. Calculate the Average Ticket Price for Events in Each Venue Using a Subquery.

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

Query Editor:

```
1693 WHERE b.booking_date >= CURDATE() - INTERVAL 30 DAY
1694 GROUP BY c.customer_id, c.name;
1695
1696
1697 • 1697 SELECT V.venue_id, V.venue_name, AVG(E.ticket_price) AS average_ticket_price
1698 FROM Venue V
1699 JOIN Event E ON V.venue_id = E.venue_id
1700 GROUP BY V.venue_id, V.venue_name;
1701
```

Result Grid:

venue_id	venue_name	average_ticket_price
2	Sports Arena	22.5
3	Concert Hall	40
4	Cinema Plex	11
5	Community Center	15

Output Window:

#	Time	Action	Message	Duration / Fetch
2	22:46:28	SELECT c.customer_id, c.name AS customer_name, SUM(b.total_cost) AS total_revenue FROM Customer c JOIN Booking b ON c.customer_id = b.customer_id WHERE b.booking_date >= CURDATE() - INTERVAL 30 DAY GROUP BY c.customer_id, c.name	0 row(s) returned	0.000 sec / 0.000 sec
3	22:48:45	SELECT e.event_type, v.venue_name, AVG(e.ticket_price) AS average_ticket_price FROM Event e JOIN Venue v ON e.venue_id = v.venue_id GROUP BY e.event_type, v.venue_name	4 row(s) returned	0.000 sec / 0.000 sec
4	22:50:54	SELECT c.customer_id, c.name AS user_name, COUNT(b.num_tickets) AS total_tickets_purchased FROM Customer c JOIN Booking b ON c.customer_id = b.customer_id WHERE b.booking_date >= CURDATE() - INTERVAL 30 DAY GROUP BY c.customer_id, c.name	0 row(s) returned	0.000 sec / 0.000 sec
5	22:53:41	SELECT V.venue_id, V.venue_name, AVG(E.ticket_price) AS average_ticket_price FROM Venue V JOIN Event E ON V.venue_id = E.venue_id GROUP BY V.venue_id, V.venue_name	4 row(s) returned	0.000 sec / 0.000 sec

Q2. Find Events with More Than 50% of Tickets Sold using subquery.

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

Query Editor:

```
1698 FROM Venue V
1699 JOIN Event E ON V.venue_id = E.venue_id
1700 GROUP BY V.venue_id, V.venue_name;
1701
1702
1703 • 1703 SELECT E.event_id, E.event_name, E.total_seats, E.available_seats, ((E.total_seats - E.available_seats) / E.total_seats) * 100 AS percentage_tickets_sold
1704 FROM Event E
1705 WHERE ((E.total_seats - E.available_seats) / E.total_seats) * 100 > 50;
1706
```

Result Grid:

event_id	event_name	total_seats	available_seats	percentage_tickets_sold
----------	------------	-------------	-----------------	-------------------------

Output Window:

#	Time	Action	Message	Duration / Fetch
3	22:48:45	SELECT e.event_type, v.venue_name, AVG(e.ticket_price) AS average_ticket_price FROM Event e JOIN Venue v ON e.venue_id = v.venue_id GROUP BY e.event_type, v.venue_name	4 row(s) returned	0.000 sec / 0.000 sec
4	22:50:54	SELECT c.customer_id, c.name AS user_name, COUNT(b.num_tickets) AS total_tickets_purchased FROM Customer c JOIN Booking b ON c.customer_id = b.customer_id WHERE b.booking_date >= CURDATE() - INTERVAL 30 DAY GROUP BY c.customer_id, c.name	0 row(s) returned	0.000 sec / 0.000 sec
5	22:53:41	SELECT V.venue_id, V.venue_name, AVG(E.ticket_price) AS average_ticket_price FROM Venue V JOIN Event E ON V.venue_id = E.venue_id GROUP BY V.venue_id, V.venue_name	4 row(s) returned	0.000 sec / 0.000 sec
6	22:56:46	SELECT E.event_id, E.event_name, E.total_seats, E.available_seats, ((E.total_seats - E.available_seats) / E.total_seats) * 100 AS percentage_tickets_sold	0 row(s) returned	0.000 sec / 0.000 sec

Q3. Calculate the Total Number of Tickets Sold for Each Event.

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

```

    WHERE ((E.total_seats - E.available_seats) / E.total_seats) * 100 > 50;
    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;
  
```

The result grid shows the following data:

event_id	event_name	total_tickets_sold
1	Movie Night	4
2	Football Match	2
3	Concert Live	2
4	Cinema Premiere	3
5	Basketball Game	1
6	Music Festival	1
7	Outdoor Movie	3

The output pane shows the execution log:

- 4 22:50:54 SELECT c.customer_id,c.name AS user_name,COUNT(b.num_tickets) AS total_tickets_purchased FROM Customer c JOIN Booking b ON c.customer_id = b.customer_id WHERE b.event_id = 1 GROUP BY c.customer_id,c.name; 0 row(s) returned
- 5 22:53:41 SELECT V.venue_id, V.venue_name, AVG(E.ticket_price) AS average_ticket_price FROM Venue V JOIN Event E ON V.venue_id = E.venue_id WHERE E.event_id = 1 GROUP BY V.venue_id, V.venue_name; 4 row(s) returned
- 6 22:56:46 SELECT E.event_id, E.event_name, E.total_seats, E.available_seats, ((E.total_seats - E.available_seats) / E.total_seats) * 100 AS percentage FROM Event E WHERE E.event_id = 1; 0 row(s) returned
- 7 22:59:31 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 WHERE E.event_id = 1 GROUP BY E.event_id, E.event_name; 7 row(s) returned

Q4. Find Users Who Have Not Booked Any Tickets Using a NOT EXISTS Subquery.

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

```

    FROM Event E
    JOIN Booking B ON E.event_id = B.event_id
    GROUP BY E.event_id, E.event_name;
    SELECT C.customer_id, C.name
    FROM Customer C
    WHERE NOT EXISTS (SELECT 1 FROM Booking B WHERE B.customer_id = C.customer_id);
  
```

The result grid shows the following data:

customer_id	name
NULL	NULL

The output pane shows the execution log:

- 7 22:59:31 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 WHERE E.event_id = 1 GROUP BY E.event_id, E.event_name; 7 row(s) returned
- 8 23:01:14 SELECT U.user_id, U.name FROM Users U WHERE NOT EXISTS (SELECT 1 FROM Booking B WHERE B.customer_id = U.user_id) AND U.user_id = 1; Error Code: 1146. Table 'tickets.users' doesn't exist 0.063 sec
- 9 23:01:54 SELECT U.user_id, U.name FROM Users U WHERE NOT EXISTS (SELECT 1 FROM Booking B WHERE B.customer_id = U.user_id) AND U.user_id = 2; Error Code: 1146. Table 'tickets.users' doesn't exist 0.015 sec
- 10 23:04:17 SELECT C.customer_id, C.name FROM Customer C WHERE NOT EXISTS (SELECT 1 FROM Booking B WHERE B.customer_id = C.customer_id) AND C.customer_id = 1; 0 row(s) returned 0.000 sec / 0.000 sec

Q5. List Events with No Ticket Sales Using a NOT IN Subquery.

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

```
L715 FROM Customer C
L716 WHERE NOT EXISTS (SELECT 1 FROM Booking B WHERE B.customer_id = C.customer_id);
L719 • SELECT E.event_id, E.event_name
L720 FROM Event E
L721 WHERE E.event_id NOT IN (SELECT B.event_id FROM Booking B);
L722
L723
```

The results grid shows two columns: event_id and event_name. The output pane displays the execution log with the following rows:

event_id	event_name	Duration / Fetch
1	Music	0.063 sec
2	Sports	0.015 sec
3	Concert	0.000 sec / 0.000 sec
4	Movie	0.063 sec / 0.000 sec

The status bar at the bottom right indicates the time as 11:07 PM and the date as 12/12/2023.

Q6. Calculate the Total Number of Tickets Sold for Each Event Type Using a Subquery in the FROM Clause.

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

```
L720 FROM Event E
L721 WHERE E.event_id NOT IN (SELECT B.event_id FROM Booking B);
L723 • SELECT ET.event_type, SUM(B.num_tickets) AS total_tickets_sold
L724 FROM (SELECT DISTINCT event_type FROM Event) ET
L725 LEFT JOIN Booking B ON ET.event_type = B.event_id
L726 GROUP BY ET.event_type;
L727
L728
```

The results grid shows the event type and the total number of tickets sold:

event_type	total_tickets_sold
Movie	4
Sports	2
Concert	2

The output pane displays the execution log with the following rows:

#	Time	Action	Message	Duration / Fetch
8	23:01:14	SELECT User_id, U.name FROM Users U WHERE NOT EXISTS (SELECT 1 FROM Booking B... Error Code: 1146. Table 'ticketbs.users' doesn't exist	0.063 sec
9	23:01:54	SELECT User_id, U.name FROM Users U WHERE NOT EXISTS (SELECT 1 FROM Booking B... Error Code: 1146. Table 'ticketbs.users' doesn't exist	0.015 sec
10	23:04:17	SELECT Customer_id, C.name FROM Customer C WHERE NOT EXISTS (SELECT 1 FROM Booking B... 0 row(s) returned	0.000 sec / 0.000 sec
11	23:06:36	SELECT Event_id, E.event_name FROM Event E WHERE Event_id NOT IN (SELECT B.event_id FROM Booking B... 0 row(s) returned	0.063 sec / 0.000 sec
12	23:09:52	SELECT ET.event_type, SUM(B.num_tickets) AS total_tickets_sold FROM (SELECT DISTINCT event_type F...	3 row(s) returned	0.046 sec / 0.000 sec

The status bar at the bottom right indicates the time as 11:10 PM and the date as 12/13/2023.

Q7. Find Events with Ticket Prices Higher Than the Average Ticket Price Using a Subquery in the WHERE Clause.

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

- Query Editor:** Displays the following SQL code:


```
L724 FROM (SELECT DISTINCT event_type FROM Event) ET
L725 LEFT JOIN Booking B ON ET.event_type = B.event_id
L726 GROUP BY ET.event_type;
L728 • SELECT event_name, ticket_price
L729 FROM Event
L730 WHERE ticket_price > (SELECT AVG(ticket_price) FROM Event);
```
- Result Grid:** Shows the results of the query:

event_name	ticket_price
Football Match	25
Concert Live	50
Music Festival	30
- Output Window:** Shows the execution log for the session:

#	Time	Action	Message	Duration / Fetch
10	23:04:17	SELECT customer_id, C.name FROM Customer C WHERE NOT EXISTS (SELECT 1 FROM Booking B WHERE customer_id = B.customer_id)	0 rows(s) returned	0.000 sec / 0.000 sec
11	23:06:36	SELECT Event_Id, E.event_name FROM Event E WHERE Event_Id NOT IN (SELECT B.event_id FROM Booking B WHERE customer_id = B.customer_id)	0 rows(s) returned	0.063 sec / 0.000 sec
12	23:09:52	SELECT ET.event_type, SUM(B.num_tickets) AS total_tickets_sold FROM (SELECT DISTINCT event_type F...	3 row(s) returned	0.046 sec / 0.000 sec
13	23:12:43	SELECT event_name, ticket_price FROM Event WHERE ticket_price > (SELECT AVG(ticket_price) FROM Event)	3 row(s) returned	0.047 sec / 0.000 sec

Q8. Calculate the Total Revenue Generated by Events for Each User Using a Correlated Subquery.

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

- Query Editor:** Displays the following SQL code:


```
L730 WHERE ticket_price > (SELECT AVG(ticket_price) FROM Event);
L731
L732 • SELECT c.name AS user_name, e.event_name, SUM(b.total_cost) AS total_revenue
L733 FROM Customer c
L734 JOIN Booking b ON c.customer_id = b.customer_id
L735 JOIN Event e ON b.event_id = e.event_id
L736 GROUP BY c.name, e.event_name;
```
- Result Grid:** Shows the results of the query:

user_name	event_name	total_revenue
Alice Johnson	Football Match	50
Bob Smith	Cinema Premiere	36
Charlie Brown	Music Festival	30
Daniel Lee	Nightlife	40
Emily Davis	Concert Live	100
Frank Miller	Basketball Game	20
Grace Taylor	Outdoor Movie	45
- Output Window:** Shows the execution log for the session:

#	Time	Action	Message	Duration / Fetch
12	23:09:52	SELECT ET.event_type, SUM(B.num_tickets) AS total_tickets_sold FROM (SELECT DISTINCT event_type F...	3 row(s) returned	0.046 sec / 0.000 sec
13	23:12:43	SELECT event_name, ticket_price FROM Event WHERE ticket_price > (SELECT AVG(ticket_price) FROM Event)	3 row(s) returned	0.047 sec / 0.000 sec
14	23:14:22	SELECT u.name AS user_name, e.event_name, SUM(b.total_cost) AS total_revenue FROM User u JOIN Booking b ON u.customer_id = b.customer_id WHERE e.event_name > (SELECT AVG(event_name) FROM Event)	Error Code: 1146: Table 'ticketsbs.user' doesn't exist	0.000 sec
15	23:15:43	SELECT c.name AS user_name, e.event_name, SUM(b.total_cost) AS total_revenue FROM Customer c JOIN Booking b ON c.customer_id = b.customer_id WHERE e.event_name > (SELECT AVG(event_name) FROM Event)	7 row(s) returned	0.000 sec / 0.000 sec

Q9. List Users Who Have Booked Tickets for Events in a Given Venue Using a Subquery in the WHERE Clause.

The screenshot shows the MySQL Workbench interface with a query editor containing the following SQL code:

```
SELECT DISTINCT u.name AS user_name
FROM Customer u
JOIN Booking b ON u.customer_id = b.customer_id
JOIN Event e ON b.event_id = e.event_id
WHERE e.venue_id = (SELECT venue_id FROM Venue WHERE venue_name = 'Pune');
```

The results grid shows a single column named "user_name" with one row: "user_name".

The status bar at the bottom right indicates the date and time: 12/12/2023, 11:21 PM.

Q10. Calculate the Total Number of Tickets Sold for Each Event Category Using a Subquery with GROUP BY.

The screenshot shows the MySQL Workbench interface with a query editor containing the following SQL code:

```
SELECT DISTINCT u.name AS user_name
FROM Customer u
JOIN Booking b ON u.customer_id = b.customer_id
JOIN Event e ON b.event_id = e.event_id
WHERE e.venue_id = (SELECT venue_id FROM Venue WHERE venue_name = 'Pune');
```

Below this, another query is shown:

```
SELECT e.event_type AS event_category,
       COUNT(b.booking_id) AS total_tickets_sold
FROM Booking b
JOIN Event e ON b.event_id = e.event_id
GROUP BY e.event_type;
```

The results grid shows two columns: "event_category" and "total_tickets_sold". The data is:

event_category	total_tickets_sold
Movie	3
Sports	2
Concert	2

The status bar at the bottom right indicates the date and time: 12/12/2023, 11:22 PM.

Q11. Find Users Who Have Booked Tickets for Events in each Month Using a Subquery with DATE_FORMAT.

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

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


```
L748 JOIN Booking b ON e.event_id = b.event_id
      GROUP BY e.event_type;
L752 • SELECT c.name AS user_name, DATE_FORMAT(b.booking_date, '%Y-%m') AS booking_month
      FROM Customer c
      JOIN Booking b ON c.customer_id = b.customer_id
      GROUP BY c.customer_id, booking_month;
```
- Result Grid:** Displays the results of the query:

user_name	booking_month
Alice Johnson	2023-01
Bob Smith	2023-02
Charlie Brown	2023-03
David Lee	2023-04
Emily Davis	2023-05
Frank Miller	2023-06
Grace Taylor	2023-07
- Output Tab:** Shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	23:26:12	SELECT c.name AS user_name, DATE_FORMAT(b.booking_date, '%Y-%m') AS booking_month FROM Customer c JOIN Booking b ON c.customer_id = b.customer_id GROUP BY c.customer_id, booking_month;	Customer... 7 row(s) returned	0.000 sec / 0.000 sec

Q12. Calculate the Average Ticket Price for Events in Each Venue Using a Subquery

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

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


```
L754 JOIN Booking b ON c.customer_id = b.customer_id
      GROUP BY c.customer_id, booking_month;
L758 • SELECT v.venue_name, AVG(e.ticket_price) AS average_ticket_price
      FROM Venue v
      JOIN Event e ON v.venue_id = e.venue_id
      GROUP BY v.venue_id, v.venue_name;
```
- Result Grid:** Displays the results of the query:

venue_name	average_ticket_price
Cinema Plex	11
Sports Arena	22.5
Concert Hall	40
Community Center	15
- Output Tab:** Shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	23:26:12	SELECT c.name AS user_name, DATE_FORMAT(b.booking_date, '%Y-%m') AS booking_month FROM Customer c JOIN Booking b ON c.customer_id = b.customer_id GROUP BY c.customer_id, booking_month;	Customer... 7 row(s) returned	0.000 sec / 0.000 sec
2	23:28:24	SELECT v.venue_name, AVG(e.ticket_price) AS average_ticket_price FROM Venue v JOIN Event e ON v.venue_id = e.venue_id GROUP BY v.venue_id, v.venue_name;	4 row(s) returned	0.000 sec / 0.000 sec