EVENT RESERVATION SYSTEM

By -
Sakshi Rana (102215293)
Rahul Gupta (102215298)
Mrikulesh Minhas (102215357)

SYSTEM OVERVIEW

Our project **Event Reservation System** is designed to streamline the booking and management of events at various venues using a Database Management System (DBMS) to ensure data organization and security.

DATABASE USED:

Use of a relational database system i.e **MySQL** for structured data.

DATABASE REQUIREMENTS

1. Event Scheduling
The system allows users to schedule different types of events with specific dates and time, providing an organized way to manage event timelines.

2. Venue Allocation and Management
The system manages detailed information about event venues, including their location, capacity, and cost. It ensures venues are appropriately allocated for events based on availability and requirements.

3. Organiser Coordination
The system manages the details of event organisers including their contact, address and email, enabling efficient communication.

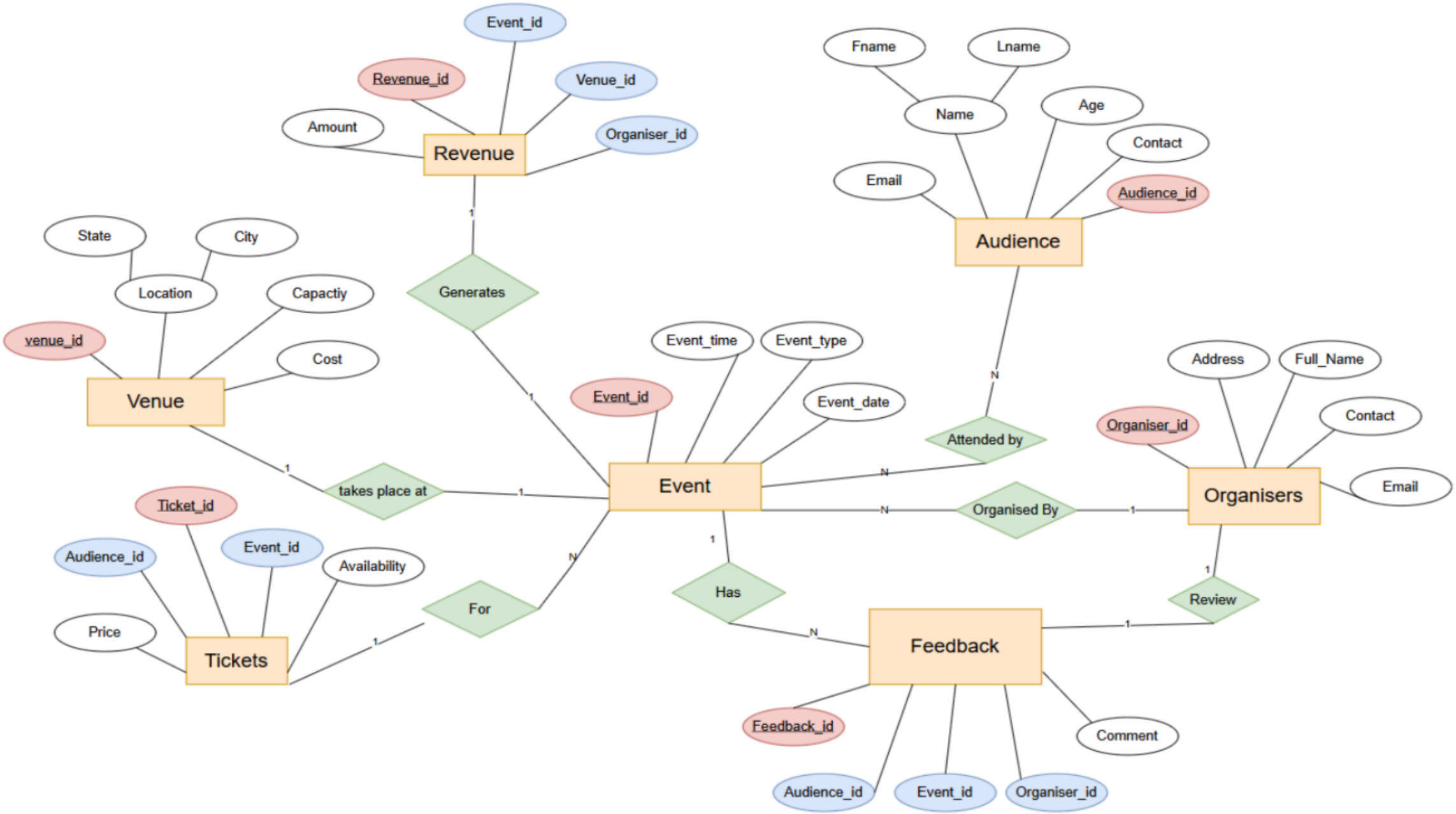
4. Audience Registeration
The system maintains records of attendees for each event, providing a comprehensive view of the audience and ensuring streamlined ticket allocation.

5. Revenue Monitoring
The system monitors the total revenue generated from each event.

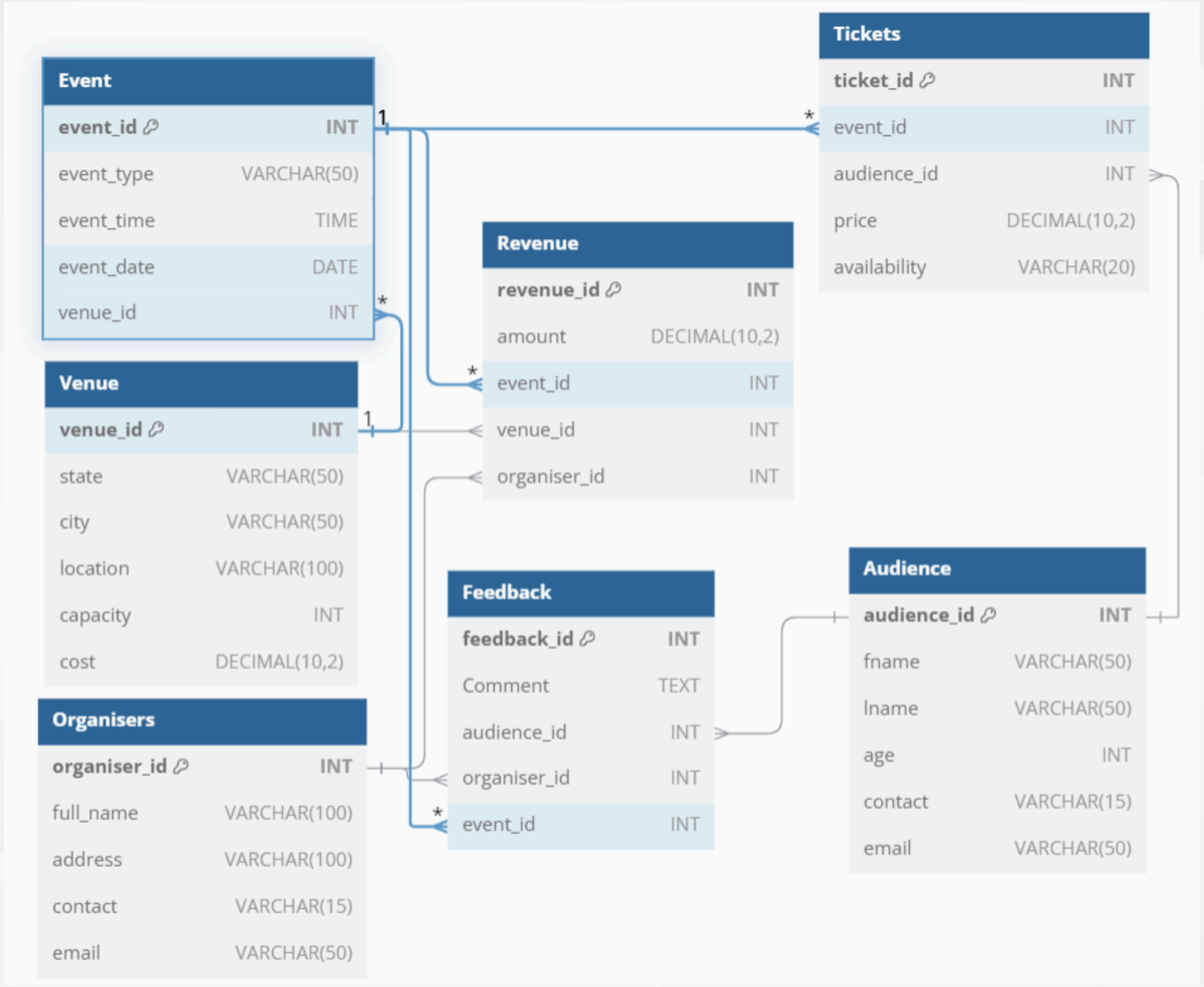
6.Ticket Availability Monitoring
The system records the availability of tickets for different event types along with their prices.

7. Feedback Analysis
The system records feedback from both the audience and organizers, helping improve future events by analyzing strengths and areas for improvement.

ENTITY-RELATIONSHIP DIAGRAM


Event_id
Revenue_id
Venue_id
Amount
Organiser_id
Revenue
Generates
Fname
Lname
Name
Age
Contact
Email
Audience_id
Audience
State
City
Location
Capactiy
venue_id
Cost
Venue
takes place at
Event_time
Event_type
Event_id
Event_date
Event
Attended by
Address
Full_Name
Organiser_id
Contact
Email
Organisers
Organised By
Ticket_id
Audience_id
Event_id
Availability
Price
Tickets
For
Has
Review
Feedback
Feedback_id
Comment
Audience_id
Event_id
Organiser_id
1
N

RELATIONAL
SCHEMA


Event
event_id INT
event_type VARCHAR(50)
event_time TIME
event_date DATE
venue_id INT
Tickets
ticket_id INT
event_id INT
audience_id INT
price DECIMAL(10,2)
availability VARCHAR(20)
Revenue
revenue_id INT
amount DECIMAL(10,2)
event_id INT
venue_id INT
organiser_id INT
Venue
venue_id INT
state VARCHAR(50)
city VARCHAR(50)
location VARCHAR(100)
capacity INT
cost DECIMAL(10,2)
Feedback
feedback_id INT
Comment TEXT
audience_id INT
organiser_id INT
event_id INT
Audience
audience_id INT
fname VARCHAR(50)
lname VARCHAR(50)
age INT
contact VARCHAR(15)
email VARCHAR(50)
Organisers
organiser_id INT
full_name VARCHAR(100)
address VARCHAR(100)
contact VARCHAR(15)
email VARCHAR(50)
1
*
1
*
1
*
*
*

ER DIAGRAM
EXPLANATION

Our **ER diagram** effectively captures the relationships and dependencies between events, organizers, venues, audiences, and the overall event management system, using proper cardinality to describe how entities interact with one another.

The various entities are:

- **Event Entity:**

The system manages events with attributes like type, date, and time.

- **Venue Entity:**

Venues have unique identifiers, location details, capacity, and rental costs.

- **Ticket Entity:** Tickets are linked to events and audience members, with attributes such as price and availability.

- **Feedback Entity:** Feedback is collected from audience members for events, including comments and identifiers for audience and event

- **Audience Entity:** Audience members are identified by attributes like name, age, and contact details.

- **Organiser Entity:**

Organizers manage events and have details like name, contact, and address.

- **Revenue Tracking:**

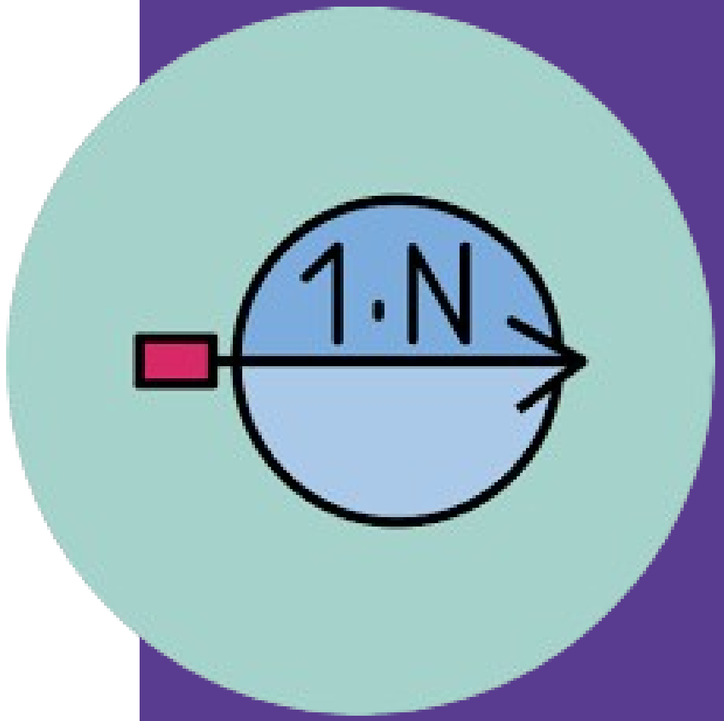
Revenue is generated for each event and linked to the venue.

RELATIONSHIPS & CARDINALITIES

- **Each event has relationships, such as taking place at one venue**, being organized by a single organizer, having multiple feedback entries, and being attended by numerous audience members.

- Each **venue hosts one event at a time**, establishing a **one-to-one relationship**.

- **Audience can attend multiple events**, forming a **many-to-many relationship with events.**



- One organizer can organize many events, but each event is organized by only one organizer **(Many-to-One)**.

- Each event generates a specific amount of revenue, which is linked to the venue where it takes place therefore forms **one-to-one relationship**.

- Each event can have multiple tickets, forming a **many-to-one relationship**.

- One event can have multiple feedback entries from different attendees therefore forms **one to many realtionship**.

1NF
NORMALIZATION

- **All columns should be single valued and there should be no repeating groups.**
- **All tables are already in 1NF because each column had single value for a particular record.**
- **Organisers table was modified to violate 1 NF and then steps were taken to implement 1NF.**

Organisers Table

organiser_id	full_name	email
1	Alice Cooper	alice.cooper@google.com
2	Michael Brown	michael.brown@google.com
3	David Lee	david.lee@google.com
4	Sandra White	sandra.white@google.com
5	Lisa Green	lisa.green@google.com
6	John King	john.king@google.com
7	Betty Adams	betty.adams@google.com
8	Charles Harris	charles.harris@google.com
9	Rachel Young	rachel.young@google.com
10	William White	william.white@google.com
11	Susan Clark	susan.clark@google.com
12	Robert Scott	robert.scott@google.com
13	Michael Moore	michael.moore@google.com
14	Laura Turner	laura.turner@google.com
15	Joshua Walker	joshua.walker@google.com

OrganisersAddresses Table

organiser_id	address
1	Address1
1	Address2
10	Address1
10	Address2
5	Address1
5	Address2

OrganisersContacts Table

organiser_id	contact
1	1234567890
1	9876543210
10	6783670197
10	960368650
5	1234567890
5	9876543210

2NF
NORMALIZATION

- The database was made to be in 1NF.
- All non-key attributes in all the tables are fully dependent on the respective primary key. There are no partial dependencies.

EXAMPLE:
In Tickets Table

- The primary key is ticket_id, and all non-key attributes **(event_id, audience_id, price, availability)** depend entirely on **ticket_id.**
- The **foreign keys (event_id and audience_id)** are correctly related to their respective tables and do not cause **partial dependency.**
- There is no partial dependency, as all non-key attributes depend fully on ticket_id.

3NF
NORMALIZATION

- The database is already in **2NF.**
- All non-key attributes are non-transitively dependent on the primary key (i.e., no transitive dependencies).

EXAMPLE:
In Tickets Table

- There are no transitive dependencies because non-key attributes like **price and availability** depend **directly on ticket_id** and not on any other non-key attributes.

IMPLEMENTATION
WITH QUERIES

Revenue Monitoring:
Find the total revenue generated by each event

```
SELECT E.event_type, SUM(R.amount) AS
total_revenue
FROM Revenue R
JOIN Event E ON R.event_id = E.event_id
GROUP BY E.event_type;
```

event_type	total_revenue
Concert	5000.00
Wedding	15700.00
Conference	3200.00
Trade Show	4000.00
Festival	6000.00
Exhibition	5800.00
Seminar	7100.00
Workshop	3600.00
Product Launch	3000.00
Birthday Party	4500.00

QUERIES
AND THIER OUTPUTS

Feedback Analysis:
Find Feedback Given by a Specific Audience

```
SELECT F.feedback_id, F.comment, E.event_type, A.fname, A.lname
FROM Feedback F
JOIN Audience A ON F.audience_id = A.audience_id
JOIN Event E ON F.event_id = E.event_id
WHERE A.audience_id = 1;
```

feedback_id	comment	event_type	fname	lname
1	Great concert, would love to attend again!	Concert	John	Doe

QUERIES
AND THIER OUTPUTS

-- Identify Events with Feedback Keywords (e.g., "Amazing" or "Better")

```
SELECT E.event_type, F.comment
FROM Feedback F
JOIN Event E ON F.event_id = E.event_id
WHERE F.comment LIKE '%Amazing%' OR
F.comment LIKE '%Better%';
```

	event_type	comment
▶	Concert	Amazing experience, well organized.
	Concert	Not bad, but could be better organized.
	Festival	Amazing music festival! Loved every performance!

QUERIES
AND THIER OUTPUTS

-- Find Events with Missing Feedback

```
SELECT E.event_id, E.event_type,
E.event_date
FROM Event E
LEFT JOIN Feedback F ON E.event_id
= F.event_id
WHERE F.feedback_id IS NULL;
```

event_id	event_type	event_date
8	Workshop	2025-04-05
9	Wedding	2025-05-10
10	Product Launch	2025-06-15
11	Birthday Party	2025-07-20
12	Charity Event	2025-08-01
13	Networking	2025-09-10
14	Job Fair	2025-10-22
15	Art Show	2025-11-18
16	Health Expo	2025-12-14
17	Food Festival	2025-01-28
18	Music Festival	2025-02-18
19	Film Screening	2025-03-12
20	Tech Conference	2025-04-01

QUERIES
AND THIER OUTPUTS

Ticket Availability:
Retrieve the Number of Available Tickets for Each Event

```
SELECT E.event_id, E.event_type,
COUNT(T.ticket_id) AS
available_tickets
FROM Tickets T
JOIN Event E ON T.event_id =
E.event_id
WHERE T.availability = 'Available'
GROUP BY E.event_id, E.event_type;
```

event_id	event_type	available_tickets
1	Concert	2
2	Wedding	2
3	Conference	2
4	Trade Show	2
5	Festival	2
6	Exhibition	2
7	Seminar	1

QUERIES
AND THIER OUTPUTS

-- Find the Average Ticket Price for Each Event Type

```
SELECT E.event_type, AVG(T.price) AS
average_ticket_price
FROM Tickets T
JOIN Event E ON T.event_id =
E.event_id
GROUP BY E.event_type;
```

event_type	average_ticket_price
Concert	50.000000
Wedding	100.000000
Conference	75.000000
Trade Show	30.000000
Festival	150.000000
Exhibition	120.000000
Seminar	60.000000

QUERIES
AND THIER OUTPUTS

Audience Registration:
Retrieve Events with Audience Count

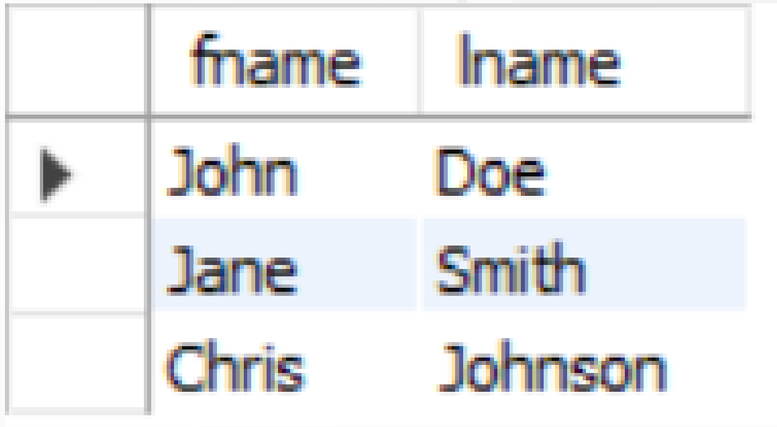
```
SELECT E.event_id, E.event_type,
COUNT(T.ticket_id) AS audience_count
FROM Event E
LEFT JOIN Tickets T ON E.event_id =
T.event_id
GROUP BY E.event_id, E.event_type;
```

	event_id	event_type	audience_count
▶	1	Concert	3
	2	Wedding	3
	3	Conference	3
	4	Trade Show	3
	5	Festival	3
	6	Exhibition	3
	7	Seminar	2
	8	Workshop	0
	9	Wedding	0
	10	Product Launch	0

QUERIES
AND THIER OUTPUTS

-- Retrieve the list of all attendees for a specific event

```
SELECT A.fname, A.lname
FROM Tickets T
JOIN Audience A ON T.audience_id =
A.audience_id
WHERE T.event_id = 1;
```



	fname	lname
▶	John	Doe
	Jane	Smith
	Chris	Johnson

QUERIES
AND THIER OUTPUTS

-- Find Average Age of Attendees for Each Event

```
SELECT E.event_type, AVG(A.age) AS
avg_age
FROM Tickets T
JOIN Audience A ON T.audience_id =
A.audience_id
JOIN Event E ON T.event_id = E.event_id
GROUP BY E.event_type;
```

event_type	avg_age
Concert	30.0000
Wedding	33.3333
Conference	44.3333
Trade Show	47.0000
Festival	33.3333
Exhibition	36.6667
Seminar	37.5000

QUERIES
AND THIER OUTPUTS

-- Retrieve Events with Total Ticket Sales Above a Certain Threshold

```
SELECT E.event_type, SUM(T.price) AS
total_sales
FROM Tickets T
JOIN Event E ON T.event_id =
E.event_id
GROUP BY E.event_type
HAVING SUM(T.price) > 50;
```

event_type	total_sales
Concert	150.00
Wedding	300.00
Conference	225.00
Trade Show	90.00
Festival	450.00
Exhibition	360.00
Seminar	120.00

QUERIES
AND THIER OUTPUTS

Venue Allocation and Management: Count Events by Venue with Average Cost and Total Capacity

```
SELECT E.event_type, A.fname, A.lname,
F.SELECT V.state, V.city,
COUNT(E.event_id) AS total_events,
AVG(V.cost) AS avg_cost,
SUM(V.capacity) AS total_capacity
FROM Venue V
LEFT JOIN Event E ON V.venue_id =
E.venue_id
GROUP BY V.state, V.city;
```

state	city	total_events	avg_cost	total_capacity
California	Los Angeles	1	10000.000000	5000
New York	New York City	1	15000.000000	3000
Texas	Austin	1	12000.000000	7000
Florida	Miami	1	11000.000000	4000
Nevada	Las Vegas	1	13000.000000	6000
Illinois	Chicago	1	20000.000000	10000
Ohio	Cleveland	1	500.000000	3500
Michigan	Detroit	1	14000.000000	8000
Arizona	Phoenix	1	12000.000000	5000
Colorado	Denver	1	11500.000000	4500

QUERIES
AND THIER OUTPUTS

Organiser Coordination:
List Events with Multiple Organisers

```
SELECT E.event_type,
COUNT(O.organiser_id) AS
organiser_count
FROM Event E
JOIN Feedback F ON E.event_id =
F.event_id
JOIN Organisers O ON F.organiser_id =
O.organiser_id
GROUP BY E.event_type
HAVING COUNT(O.organiser_id) > 1;
```

	event_type	organiser_count
▶	Concert	3
	Wedding	3
	Conference	3
	Trade Show	3
	Festival	3
	Exhibition	3
	Seminar	2

CONCLUSION

The Event Reservation System efficiently organizes and tracks events, venues, audiences, and revenue, providing a seamless experience for event organizers. With features like event scheduling, venue allocation, ticket availability monitoring, and feedback analysis, it offers a structured approach to managing various aspects of event planning. This project emphasizes the importance of a well-designed database, ensuring accurate storage, retrieval, and processing of event-related data, ultimately simplifying the complexities of organizing and executing successful events.

THANK YOU !