Entities and Attributes

To create an efficient and normalized database, it is crucial to break the information into related tables based on entity relationships, which allows for easier querying and data integrity. Here are the main entities:

1. **Users**
    - user_id: Primary Key, Unique identifier for each user.
    - first_name: User's first name.
    - last_name: User's last name.
    - email: Email of the user, needs to be unique.
    - password: Password for user authentication.
    - user_type: Defines whether the user is an "admin" or an "attendee".

Justification: This table handles user authentication and identification. We separate the user attributes since not all users are attendees of an event, and admins manage the events.

2. **Venues**
    - venue_id: Primary Key, Unique identifier for each venue.
    - venue_name: Name of the venue.
    - location: Venue's address.
    - capacity: Maximum number of attendees the venue can handle.

Justification: Venue attributes are separated into this table to avoid redundancy (e.g., events can be held at the same venue). The venue's capacity helps with validation for attendee bookings.

3. **Events**
    - event_id: Primary Key, Unique identifier for each event.
    - event_name: Name of the event.
    - event_date: Date of the event.
    - venue_id: Foreign Key from the Venues table (a venue hosts an event).
    - description: Description of the event.

Justification: Events are managed by admins, and attendees register for them. Each event is connected to a venue through the venue_id as a foreign key, ensuring we know where an event is being held.

4. **Attendees**
    - attendee_id: Primary Key, Unique identifier for each attendee registration.
    - user_id: Foreign Key from the Users table (each attendee is a user).

- event_id: Foreign Key from the Events table (each attendee attends an event).
- booking_date: The timestamp when an attendee registered for the event.

Justification: This table tracks which users (attendees) have registered for specific events. The user_id and event_id serve as foreign keys to establish these relationships.

5. **Bookings**
    - booking_id: Primary Key, Unique identifier for each booking.
    - attendee_id: Foreign Key from the Attendees table.
    - event_id: Foreign Key from the Events table.
    - booking_status: Can be 'confirmed' or 'cancelled', depending on the attendee's booking status.

Justification: The Bookings table keeps track of each attendee's booking status for each event. The combination of attendee_id and event_id establishes which attendee is booking for which event.

Foreign Keys Justification:

- **venue_id in Events**: Links events to their respective venues, ensuring that each event occurs at a valid location.
- **user_id in Attendees**: Links each attendee to a user, meaning only valid users can register as attendees.
- **event_id in Attendees**: Ensures that the attendee is tied to a valid event.
- **attendee_id and event_id in Bookings**: Establishes a many-to-many relationship between events and attendees, with the booking_status attribute handling the state of each booking.

Functional Dependencies

- **FD1**: user_id -> first_name, last_name, email, password, user_type
- **FD2**: venue_id -> venue_name, location, capacity
- **FD3**: event_id -> event_name, event_date, venue_id, description
- **FD4**: attendee_id -> user_id, event_id, booking_date
- **FD5**: booking_id -> attendee_id, event_id, booking_status

Normalization Steps

Step 1: First Normal Form (1NF)

1NF Requirements:

- Each table must have atomic values.
- Each attribute must contain only one value (no repeating groups).

Analysis:

All tables are already in **1NF** since they contain atomic values and no repeating groups.

Step 2: Second Normal Form (2NF)

2NF Requirements:

- The table must be in 1NF.
- All non-key attributes must be fully functionally dependent on the primary key.

Analysis:

All tables are in **2NF** since there are no partial dependencies.

Step 3: Third Normal Form (3NF)

3NF Requirements:

- The table must be in 2NF.
- There should be no transitive dependencies (non-key attributes depending on other non-key attributes).

Analysis:

All tables are in **3NF** since there are no transitive dependencies.

SQL IMPLEMENTATION

1. **Create Users Table**:

```
CREATE TABLE Users (
user_id INT PRIMARY KEY AUTO_INCREMENT,
first_name VARCHAR(100),
last_name VARCHAR(100),
email VARCHAR(100) UNIQUE,
password VARCHAR(100)
user_type ENUM('admin', 'attendee') DEFAULT 'attendee' );
```

Insert Values into Venues Table:

```
INSERT INTO Users (first_name, last_name, email, password, user_type)
VALUES
('John', 'Doe', 'john.doe@example.com', 'password123', 'attendee'),
('Jane', 'Smith', 'jane.smith@example.com', 'password456', 'admin'),
('Alice', 'Brown', 'alice.brown@example.com', 'password789', 'attendee');
```

2. **Create Venues Table:**

```
CREATE TABLE Venues (
   venue_id INT PRIMARY KEY AUTO_INCREMENT,
   venue_name VARCHAR(100),
   location VARCHAR(255),
   capacity INT
);
```

Insert Values into Venues Table

```
INSERT INTO Venues (venue_name, location, capacity)
VALUES
('Conference Hall A', '123 Main Street', 500),
('Outdoor Arena', '45 Park Ave', 1000),
('Exhibition Center', '99 Center Blvd', 750);
```

3. **Create Events Table:**

```
CREATE TABLE Events (
   event_id INT PRIMARY KEY AUTO_INCREMENT,
   event_name VARCHAR(100),
   event_date DATE,
   venue_id INT,
   description TEXT,
   FOREIGN KEY (venue_id) REFERENCES Venues(venue_id)
);
```

Insert Values into Events Table

```
INSERT INTO Events (event_name, event_date, venue_id, description)
 VALUES
('Tech Conference 2024', '2024-05-15', 1, 'A conference focusing on new technologies.'),
('Music Festival', '2024-06-20', 2, 'A grand music event with various artists.'),
('Art Expo', '2024-08-10', 3, 'An exhibition of modern art.');
```

4. **Create Attendees Table:**

```
CREATE TABLE Attendees (
   attendee_id INT PRIMARY KEY AUTO_INCREMENT,
   user_id INT,
   event_id INT,
   booking_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (user_id) REFERENCES Users(user_id),
   FOREIGN KEY (event_id) REFERENCES Events(event_id)
);
```

Insert Values into Attendees Table

```
INSERT INTO Attendees (user_id, event_id, booking_date)
VALUES
(1, 1, '2024-01-15'),
(1, 2, '2024-02-12'),
(3, 1, '2024-03-01');
```

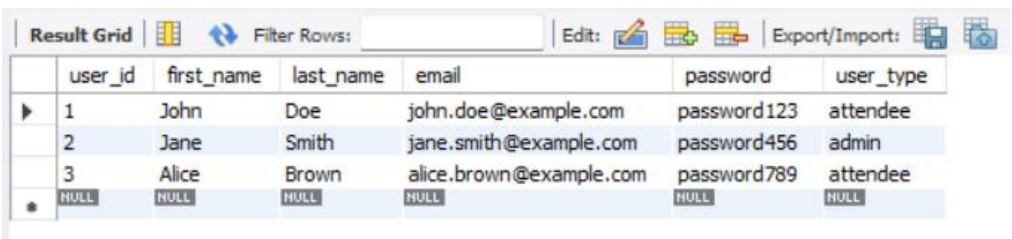
5. **Create Bookings Table:**

```
CREATE TABLE Bookings (
   booking_id INT PRIMARY KEY AUTO_INCREMENT,
   attendee_id INT,
   event_id INT,
   booking_status ENUM('confirmed', 'cancelled') DEFAULT 'confirmed',
   FOREIGN KEY (attendee_id) REFERENCES Attendees(attendee_id),
   FOREIGN KEY (event_id) REFERENCES Events(event_id)
);
```

Insert Values into Bookings Table

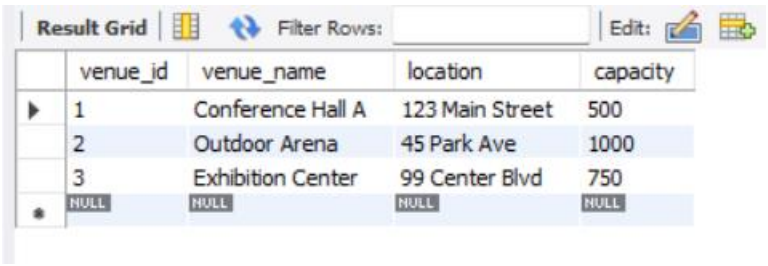
```
INSERT INTO Bookings (attendee_id, event_id, booking_status)
VALUES
(1, 1, 'confirmed'),
(2, 2, 'confirmed'),
(3, 1, 'cancelled');
```

USERS TABLE:



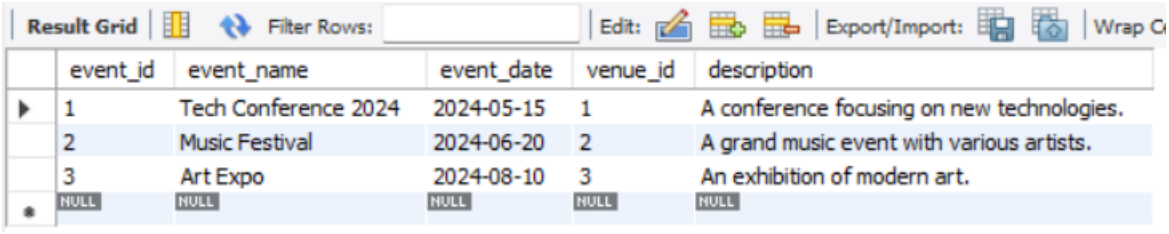
user_id	first_name	last_name	email	password	user_type
1	John	Doe	john.doe@example.com	password123	attendee
2	Jane	Smith	jane.smith@example.com	password456	admin
3	Alice	Brown	alice.brown@example.com	password789	attendee
NULL	NULL	NULL	NULL	NULL	NULL

VENUES TABLE:



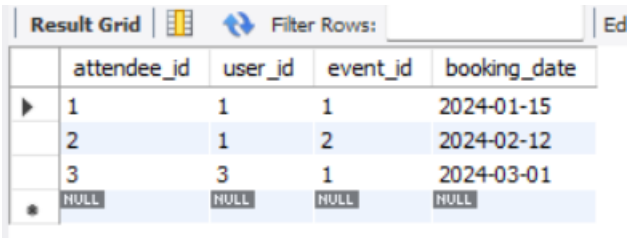
venue_id	venue_name	location	capacity
1	Conference Hall A	123 Main Street	500
2	Outdoor Arena	45 Park Ave	1000
3	Exhibition Center	99 Center Blvd	750
NULL	NULL	NULL	NULL

EVENTS TABLE:



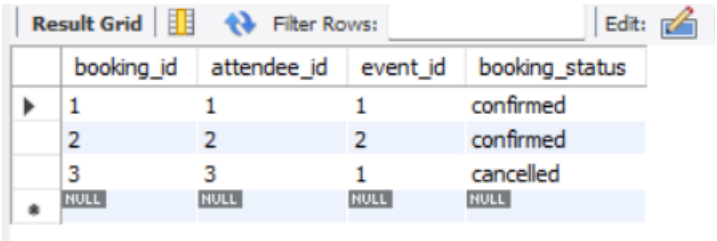
event_id	event_name	event_date	venue_id	description
1	Tech Conference 2024	2024-05-15	1	A conference focusing on new technologies.
2	Music Festival	2024-06-20	2	A grand music event with various artists.
3	Art Expo	2024-08-10	3	An exhibition of modern art.
NULL	NULL	NULL	NULL	NULL

ATTENDEES TABLE:



attendee_id	user_id	event_id	booking_date
1	1	1	2024-01-15
2	1	2	2024-02-12
3	3	1	2024-03-01
NULL	NULL	NULL	NULL

BOOKINGS TABLE:



booking_id	attendee_id	event_id	booking_status
1	1	1	confirmed
2	2	2	confirmed
3	3	1	cancelled
NULL	NULL	NULL	NULL

QUERIES

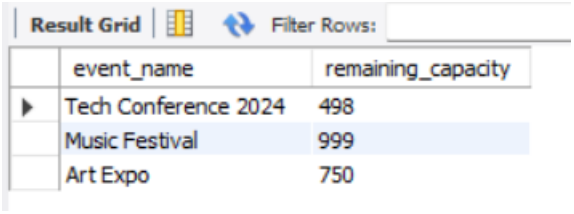
1. Arithmetic Operators

Q) Calculate the remaining capacity for an event based on venue capacity and the number of attendees

QUERY:

```
SELECT e.event_name, v.capacity - COUNT(a.attendee_id) AS remaining_capacity
FROM Events e
JOIN Venues v ON e.venue_id = v.venue_id
LEFT JOIN Attendees a ON e.event_id = a.event_id
GROUP BY e.event_id, v.capacity;
```

OUTPUT:



Result Grid | Filter Rows:

event_name	remaining_capacity
Tech Conference 2024	498
Music Festival	999
Art Expo	750

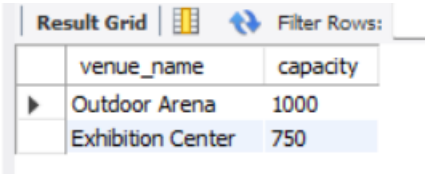
2. Comparison Operators

Q) Find all venues with a capacity greater than 700

QUERY:

```
SELECT venue_name, capacity
FROM Venues
WHERE capacity > 700;
```

OUTPUT:



Result Grid | Filter Rows:

venue_name	capacity
Outdoor Arena	1000
Exhibition Center	750

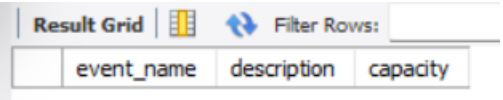
3. Logical Operators

Q) Find all events hosted in venues with a capacity greater than 700 AND with a description containing the word "conference"

QUERY:

```
SELECT event_name, description, capacity
FROM Events e
JOIN Venues v ON e.venue_id = v.venue_id
WHERE v.capacity > 700 AND e.description LIKE '%conference%';
```

OUTPUT:



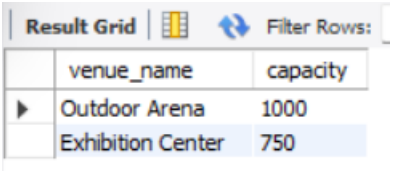
4. BETWEEN Operator

Q) Find all venues with a capacity between 600 and 1000

QUERY:

```
SELECT venue_name, capacity
FROM Venues
WHERE capacity BETWEEN 600 AND 1000;
```

OUTPUT:



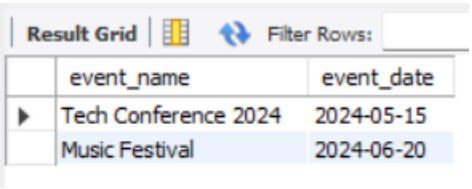
5. IN Operator

Q) Find all events held in specific venues by venue IDs (1, 2)

QUERY:

```
SELECT event_name, event_date
FROM Events
WHERE venue_id IN (1, 2);
```

OUTPUT:



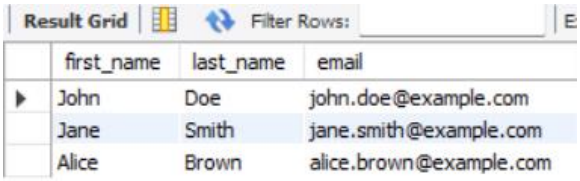
6. LIKE Operator

Q) Find all users with email addresses that contain "example"

QUERY:

```
SELECT first_name, last_name, email
FROM Users
WHERE email LIKE '%example%';
```

OUTPUT:



first_name	last_name	email
John	Doe	john.doe@example.com
Jane	Smith	jane.smith@example.com
Alice	Brown	alice.brown@example.com

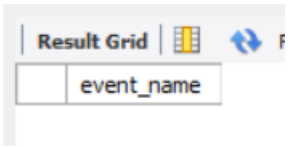
7. IS NULL / IS NOT NULL Operator

Q) Find all events where the description is missing (NULL)

QUERY:

```
SELECT event_name
FROM Events
WHERE description IS NULL;
```

OUTPUT:



event_name

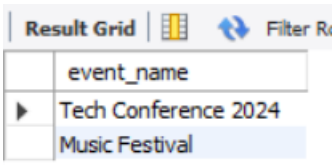
8. DISTINCT Operator

Q) Find all distinct event names attended by users

QUERY:

```
SELECT DISTINCT e.event_name
FROM Events e
JOIN Attendees a ON e.event_id = a.event_id;
```

OUTPUT:



event_name
Tech Conference 2024
Music Festival

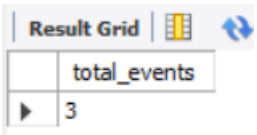
9. COUNT, SUM, AVG Operators

Q) Count the total number of events

QUERY:

```
SELECT COUNT(event_id) AS total_events
FROM Events;
```

OUTPUT:



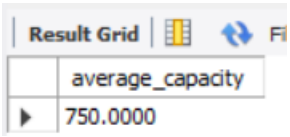
	total_events
▶	3

Q) Find the average capacity of all venues

QUERY:

```
SELECT AVG(capacity) AS average_capacity
FROM Venues;
```

OUTPUT:



	average_capacity
▶	750.0000

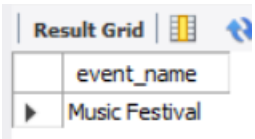
10. WILDCARDS

Q) Search for events with the word "Music"

QUERY:

```
SELECT event_name FROM Events
WHERE event_name LIKE '%Music%';
```

OUTPUT:



	event_name
▶	Music Festival

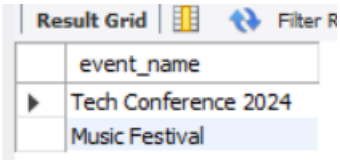
11. NESTED QUERIES

Q) Find the Events Attended by John Doe (Using Subquery to Fetch user_id)

QUERY:

```
SELECT event_name
FROM Events
WHERE event_id IN (
   SELECT event_id
   FROM Attendees
   WHERE user_id = (SELECT user_id FROM Users WHERE first_name = 'John' AND last_name = 'Doe')
);
```

OUTPUT:



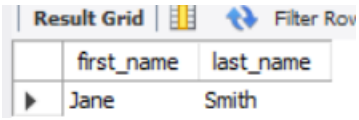
event_name
Tech Conference 2024
Music Festival

Q) Find Users Who Have Not Booked Any Events (Using NOT IN Subquery)

QUERY:

```
SELECT first_name, last_name
FROM Users
WHERE user_id NOT IN (
  SELECT user_id
  FROM Attendees
);
```

OUTPUT:



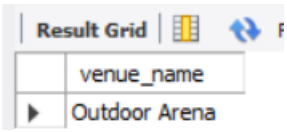
first_name	last_name
Jane	Smith

Q) Find All Venues Where Capacity Is Higher Than the Average Event Capacity

QUERY:

```
SELECT venue_name
FROM Venues
WHERE capacity > (
  SELECT AVG(capacity)
  FROM Venues
);
```

OUTPUT:



venue_name
Outdoor Arena

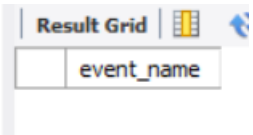
Q) Get the Events Created by the Admin Jane Smith (Using Subquery to Fetch user_id of Admin)

QUERY:

```
SELECT event_name
FROM Events
WHERE venue_id IN (
  SELECT venue_id
  FROM Venues
  WHERE venue_id IN (
    SELECT venue_id
    FROM Events
```

```
    WHERE event_id IN (
      SELECT event_id
      FROM Attendees
      WHERE user_id = (SELECT user_id FROM Users WHERE first_name = 'Jane' AND last_name =
'Smith')
    )
  )
);
```

OUTPUT:



event_name

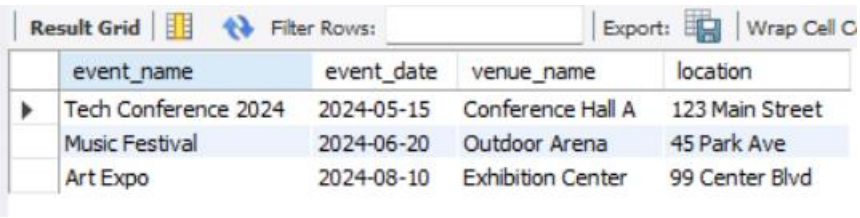
12. JOIN QUERIES

Q) **Inner Join** (Joining Events and Venues to show Event details with Venue names)

QUERY:

```
SELECT e.event_name, e.event_date, v.venue_name, v.location
FROM Events e
INNER JOIN Venues v ON e.venue_id = v.venue_id;
```

OUTPUT:



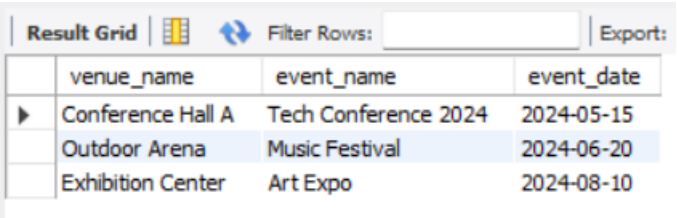
event_name	event_date	venue_name	location
Tech Conference 2024	2024-05-15	Conference Hall A	123 Main Street
Music Festival	2024-06-20	Outdoor Arena	45 Park Ave
Art Expo	2024-08-10	Exhibition Center	99 Center Blvd

Q) **Left Join** (Listing all Venues and the Events hosted at each Venue, showing NULL for Venues with no events)

QUERY:

```
SELECT v.venue_name, e.event_name, e.event_date
FROM Venues v
LEFT JOIN Events e ON v.venue_id = e.venue_id;
```

OUTPUT:



venue_name	event_name	event_date
Conference Hall A	Tech Conference 2024	2024-05-15
Outdoor Arena	Music Festival	2024-06-20
Exhibition Center	Art Expo	2024-08-10

Q) **Right Join** (Listing all Events and the Users who booked them, showing NULL for events with no users booked)

QUERY:

```
SELECT e.event_name, u.first_name, u.last_name
FROM Events e
RIGHT JOIN Attendees a ON e.event_id = a.event_id
RIGHT JOIN Users u ON a.user_id = u.user_id;
```

OUTPUT:

Result Grid | Filter Rows:

event_name	first_name	last_name
Tech Conference 2024	John	Doe
Music Festival	John	Doe
NULL	Jane	Smith
Tech Conference 2024	Alice	Brown

Q) **Full Outer Join** (Listing all Users and the Events they attended, showing NULL where a user has not attended an event or an event has no attendees)

QUERY:

```
SELECT u.first_name, u.last_name, e.event_name
FROM Users u
LEFT JOIN Attendees a ON u.user_id = a.user_id
LEFT JOIN Events e ON a.event_id = e.event_id

UNION

SELECT u.first_name, u.last_name, e.event_name
FROM Users u
RIGHT JOIN Attendees a ON u.user_id = a.user_id
RIGHT JOIN Events e ON a.event_id = e.event_id;
```

OUTPUT:

Result Grid | Filter Rows:

first_name	last_name	event_name
John	Doe	Music Festival
John	Doe	Tech Conference 2024
Jane	Smith	NULL
Alice	Brown	Tech Conference 2024
NULL	NULL	Art Expo