



# Virtual Event/Conference Manager Database Design

## 1. Entity-Relationship Diagram (ERD)

Our ERD models the key entities and relationships for a virtual conference system. It includes **Event**, **Organizer**, **Attendee**, **Speaker**, **Session**, **TicketType**, **TicketSale**, and two associative (join) entities **AttendeeSession** and **SpeakerSession**. Each **Event** (e.g. a conference) is created by one **Organizer** (1-to-many relationship: an organizer can create many events). Each **Event** has many **Sessions** (one-to-many). **Sessions** belong to exactly one event. **Attendees** can register for multiple sessions and each **Session** can have many attendees – this many-to-many relationship is implemented via the **AttendeeSession** join table [en.wikipedia.org](https://en.wikipedia.org/wiki/Join_table). Similarly, **Speakers** can speak at multiple sessions and a session can have multiple speakers (many-to-many via **SpeakerSession**). **TicketType** (e.g. "General", "VIP") is tied to an **Event** (one event has many ticket types), and **TicketSale** records each ticket purchase by linking a ticket type to an attendee. In summary, we use foreign keys to enforce one-to-many links (e.g. `Session.EventID → Event`) and associative join tables with composite keys for many-to-many links [en.wikipedia.org](https://en.wikipedia.org/wiki/Join_table). This ERD reflects a fully normalized design (each non-key attribute depends only on its table's primary key, avoiding redundancy) [en.wikipedia.org](https://en.wikipedia.org/wiki/Database_normalization).

- **Entities and Keys:**

- *Event*(EventID, OrganizerID, Name, Description, StartDate, EndDate, Location)
- *Organizer*(OrganizerID, Name, ContactName, Email, Phone)
- *Session*(SessionID, EventID, Title, Description, StartTime, EndTime)
- *Attendee*(AttendeeID, FirstName, LastName, Email, Phone)
- *Speaker*(SpeakerID, FirstName, LastName, Bio, Email)

- *TicketType*(TicketTypeID, EventID, Name, Price)
- *TicketSale*(TicketSaleID, TicketTypeID, AttendeeID, Quantity, PurchaseDate)
- **Join Tables (Many-to-Many):**
  - *AttendeeSession*(AttendeeID, SessionID, RegistrationDate) – links Attendees and Sessions [en.wikipedia.org](https://en.wikipedia.org) .
  - *SpeakerSession*(SpeakerID, SessionID) – links Speakers and Sessions [en.wikipedia.org](https://en.wikipedia.org) .

All relationships and cardinalities are shown in the ERD. (For instance, each Session has a foreign key to Event, and the AttendeeSession table has composite PK of AttendeeID+SessionID [en.wikipedia.org](https://en.wikipedia.org) .) The ERD is designed in 3NF: e.g., we separate ticket types from ticket purchases so that price isn't repeated in every sale, avoiding update anomalies [en.wikipedia.org](https://en.wikipedia.org) .

## 2. Database Schema Definition

Each table and its columns (with types, keys, and constraints) are defined as follows. All tables use integer primary keys (auto-increment) unless otherwise noted. Foreign keys reference the related parent table. Key constraints ensure referential integrity and normalization [en.wikipedia.org](https://en.wikipedia.org) .

- **Organizer:** (OrganizerID INT PRIMARY KEY AUTO\_INCREMENT, Name VARCHAR(255) NOT NULL, ContactName VARCHAR(255), Email VARCHAR(255) UNIQUE, Phone VARCHAR(20)) . (Holds companies or people who create events.)
- **Event:** (EventID INT PRIMARY KEY AUTO\_INCREMENT, OrganizerID INT, Name VARCHAR(255) NOT NULL, Description TEXT, StartDate DATE, EndDate DATE, Location VARCHAR(255), FOREIGN KEY (OrganizerID) REFERENCES Organizer(OrganizerID)) . (Each event is linked to one Organizer.)
- **Attendee:** (AttendeeID INT PRIMARY KEY AUTO\_INCREMENT, FirstName VARCHAR(100), LastName VARCHAR(100), Email VARCHAR(255) UNIQUE, Phone VARCHAR(20)) . (Event attendees, with unique email.)

- **Speaker:** (SpeakerID INT PRIMARY KEY AUTO\_INCREMENT, FirstName VARCHAR(100), LastName VARCHAR(100), Bio TEXT, Email VARCHAR(255) UNIQUE) .  
(Speakers/presenters.)
- **Session:** (SessionID INT PRIMARY KEY AUTO\_INCREMENT, EventID INT, Title VARCHAR(255) NOT NULL, Description TEXT, StartTime DATETIME, EndTime DATETIME, FOREIGN KEY (EventID) REFERENCES Event(EventID)) . (Each session belongs to one Event.)
- **TicketType:** (TicketTypeID INT PRIMARY KEY AUTO\_INCREMENT, EventID INT, Name VARCHAR(100), Price DECIMAL(10,2), FOREIGN KEY (EventID) REFERENCES Event(EventID)) . (Types of tickets for an event.)
- **TicketSale:** (TicketSaleID INT PRIMARY KEY AUTO\_INCREMENT, TicketTypeID INT, AttendeeID INT, Quantity INT DEFAULT 1, PurchaseDate DATETIME, FOREIGN KEY (TicketTypeID) REFERENCES TicketType(TicketTypeID), FOREIGN KEY (AttendeeID) REFERENCES Attendee(AttendeeID)) . (Each row is a ticket purchase by an attendee.)
- **AttendeeSession:** (AttendeeID INT, SessionID INT, RegistrationDate DATETIME, PRIMARY KEY (AttendeeID, SessionID), FOREIGN KEY (AttendeeID) REFERENCES Attendee(AttendeeID), FOREIGN KEY (SessionID) REFERENCES Session(SessionID)) .  
(Join table linking attendees to sessions [en.wikipedia.org](https://en.wikipedia.org) .)
- **SpeakerSession:** (SpeakerID INT, SessionID INT, PRIMARY KEY (SpeakerID, SessionID), FOREIGN KEY (SpeakerID) REFERENCES Speaker(SpeakerID), FOREIGN KEY (SessionID) REFERENCES Session(SessionID)) . (Join table linking speakers to sessions.)

Each table's non-key fields depend only on its primary key (e.g. ticket prices are stored once in **TicketType**, avoiding data duplication) [en.wikipedia.org](https://en.wikipedia.org) . All foreign keys enforce valid relationships (e.g. you cannot assign a session to a non-existent event), maintaining a normalized design.

### 3. SQL Table Creation Scripts

The following SQL creates the database and tables (MySQL syntax). Each `CREATE TABLE` includes primary key and foreign key constraints as defined above.

sql

 Copy

```
CREATE DATABASE ConferenceManager;
```

```
USE ConferenceManager;
```

```
CREATE TABLE Organizer (  
    OrganizerID INT AUTO_INCREMENT PRIMARY KEY,  
    Name VARCHAR(255) NOT NULL,  
    ContactName VARCHAR(255),  
    Email VARCHAR(255) UNIQUE,  
    Phone VARCHAR(20)  
);
```

```
CREATE TABLE Attendee (  
    AttendeeID INT AUTO_INCREMENT PRIMARY KEY,  
    FirstName VARCHAR(100),  
    LastName VARCHAR(100),  
    Email VARCHAR(255) UNIQUE,  
    Phone VARCHAR(20)  
);
```

```
CREATE TABLE Speaker (  
    SpeakerID INT AUTO_INCREMENT PRIMARY KEY,  
    FirstName VARCHAR(100),  
    LastName VARCHAR(100),  
    Bio TEXT,  
    Email VARCHAR(255) UNIQUE  
);
```

```
CREATE TABLE Event (  
    EventID INT AUTO_INCREMENT PRIMARY KEY,  
    OrganizerID INT,  
    Name VARCHAR(255) NOT NULL,  
    Description TEXT,  
    StartDate DATE,  
    EndDate DATE,  
    Location VARCHAR(255),  
    FOREIGN KEY (OrganizerID) REFERENCES Organizer(OrganizerID)  
);
```

```
CREATE TABLE Session (  

```

```
SessionID INT AUTO_INCREMENT PRIMARY KEY,  
EventID INT,  
Title VARCHAR(255) NOT NULL,  
Description TEXT,  
StartTime DATETIME,  
EndTime DATETIME,  
FOREIGN KEY (EventID) REFERENCES Event(EventID)  
);
```

```
CREATE TABLE TicketType (  
TicketTypeID INT AUTO_INCREMENT PRIMARY KEY,  
EventID INT,  
Name VARCHAR(100),  
Price DECIMAL(10,2),  
FOREIGN KEY (EventID) REFERENCES Event(EventID)  
);
```

```
CREATE TABLE TicketSale (  
TicketSaleID INT AUTO_INCREMENT PRIMARY KEY,  
TicketTypeID INT,  
AttendeeID INT,  
Quantity INT DEFAULT 1,  
PurchaseDate DATETIME,  
FOREIGN KEY (TicketTypeID) REFERENCES TicketType(TicketTypeID),  
FOREIGN KEY (AttendeeID) REFERENCES Attendee(AttendeeID)  
);
```

```
CREATE TABLE AttendeeSession (  
AttendeeID INT,  
SessionID INT,  
RegistrationDate DATETIME,  
PRIMARY KEY (AttendeeID, SessionID),  
FOREIGN KEY (AttendeeID) REFERENCES Attendee(AttendeeID),  
FOREIGN KEY (SessionID) REFERENCES Session(SessionID)  
);
```

```
CREATE TABLE SpeakerSession (  
SpeakerID INT,  
SessionID INT,  
PRIMARY KEY (SpeakerID, SessionID),
```

```
FOREIGN KEY (SpeakerID) REFERENCES Speaker(SpeakerID),
FOREIGN KEY (SessionID) REFERENCES Session(SessionID)
);
```

This script creates each table in an order that respects foreign-key dependencies. All tables use proper data types (e.g. dates, text, decimals for prices) and enforce NOT NULL on required fields. The schema is in 3NF, so update anomalies are avoided [en.wikipedia.org](https://en.wikipedia.org/wiki/Third_normal_form) .

## 4. Sample Data Inserts

Here are example `INSERT` statements with dummy data to populate the major tables:

sql

 Copy

```
-- Insert an organizer and an event
INSERT INTO Organizer (Name, ContactName, Email, Phone) VALUES
('Tech Events Inc.', 'Alice Smith', 'alice@techevents.com', '123-456-7890');

INSERT INTO Event (OrganizerID, Name, Description, StartDate, EndDate, Location)
(1, 'Virtual Tech Conference 2025', 'Annual virtual tech conference', '2025-05-01', '2025-05-05', 'Virtual');

-- Insert attendees
INSERT INTO Attendee (FirstName, LastName, Email, Phone) VALUES
('John', 'Doe', 'john.doe@example.com', '555-0101'),
('Jane', 'Smith', 'jane.smith@example.com', '555-0102');

-- Insert speakers
INSERT INTO Speaker (FirstName, LastName, Bio, Email) VALUES
('Dr.', 'Who', 'Time traveler and keynote speaker', 'dr.who@example.com'),
('Alice', 'Johnson', 'Cloud computing expert', 'alice.j@example.com');

-- Insert sessions (for EventID = 1)
INSERT INTO Session (EventID, Title, Description, StartTime, EndTime) VALUES
(1, 'Opening Keynote', 'Kickoff keynote session', '2025-05-01 09:00:00', '2025-05-01 10:00:00'),
(1, 'Cloud Workshop', 'Hands-on cloud computing workshop', '2025-05-01 11:00:00', '2025-05-01 12:00:00');

-- Insert ticket types for EventID = 1
INSERT INTO TicketType (EventID, Name, Price) VALUES
(1, 'General Admission', 0.00),
```

```

(1, 'VIP Pass', 100.00);

-- Insert ticket sales
INSERT INTO TicketSale (TicketTypeID, AttendeeID, Quantity, PurchaseDate) VALUES
(1, 1, 1, NOW()),
(2, 2, 1, NOW());

-- Attendees register for sessions
INSERT INTO AttendeeSession (AttendeeID, SessionID, RegistrationDate) VALUES
(1, 1, NOW()),
(2, 2, NOW());

-- Assign speakers to sessions
INSERT INTO SpeakerSession (SpeakerID, SessionID) VALUES
(1, 1), -- Dr. Who speaks at Opening Keynote
(2, 2); -- Alice Johnson at Cloud Workshop

```

These example inserts populate one organizer, one event, two attendees, two speakers, two sessions, two ticket types, ticket sales, and registrations. They demonstrate foreign-key use (e.g. Session rows reference the correct `EventID` and AttendeeSession uses existing AttendeeID/SessionID pairs).

## 5. C# Interface Outline

A basic C# application (e.g. WinForms, WPF, or ASP.NET MVC) would provide forms or pages for managing the above data. Example UI components include:

- **Event Management Form:** Fields to enter event details (Name, Description, dates, Location) and a dropdown to select the **Organizer**. On submission, the form executes an `INSERT` into the `Event` table (using a SQL command or ORM).
- **Session Management Form:** A screen where the user selects an **Event** (via dropdown) and enters session details (title, times). There is also a multi-select for **Speaker** (populated from the `Speaker` table) to associate speakers. Submitting adds a row to `Session` and entries to `SpeakerSession` (establishing the many-to-many link).

- **Attendee Registration Form:** Allows new attendees to sign up (inserting into `Attendee` with name/email). The attendee can then register for sessions (adding rows to `AttendeeSession`) and/or purchase a ticket (selecting a ticket type from `TicketType` and creating a `TicketSale`). For example, clicking "Register" might run an `INSERT` into `AttendeeSession`.
- **Ticket Purchase Interface:** Shows available ticket types (from `TicketType`) for an event. The attendee selects one and confirms purchase, which inserts a row into `TicketSale`.
- **Data Access:** Under the hood, the C# code can use ADO.NET or an ORM like Entity Framework to interact with the database. For instance, Entity Framework can map each table to a class, letting developers work with objects instead of raw SQL [c-sharpcorner.com](https://c-sharpcorner.com). This decouples the UI code from SQL details and ensures changes in the database schema can be managed in one place [c-sharpcorner.com](https://c-sharpcorner.com).

The forms would validate user input and display database-driven lists (e.g. a `DataGridView` of events or sessions). In summary, the C# interface provides standard CRUD forms for each entity (Create/Read/Update/Delete events, sessions, attendees, etc.), and uses data bindings or commands to execute the SQL scripts outlined above. This gives a complete system where users can add events and sessions, register attendees, and manage tickets in a user-friendly way.

**Sources:** We followed standard database design and normalization principles (an ERD visualizes entities and relationships [visual-paradigm.com](https://visual-paradigm.com), many-to-many relationships require join tables [en.wikipedia.org](https://en.wikipedia.org), and a 3NF design avoids update anomalies [en.wikipedia.org](https://en.wikipedia.org)). For C# data access, Entity Framework is an example ORM that abstracts a SQL database into C# objects [c-sharpcorner.com](https://c-sharpcorner.com).

---

## Citations

### W Many-to-many (data model) - Wikipedia

[https://en.wikipedia.org/wiki/Many-to-many\\_\(data\\_model\)](https://en.wikipedia.org/wiki/Many-to-many_(data_model))

### W Database normalization - Wikipedia

[https://en.wikipedia.org/wiki/Database\\_normalization](https://en.wikipedia.org/wiki/Database_normalization)

## Entity Framework using C#





<https://www.c-sharpcorner.com/article/entity-framework-introduction-using-c-sharp-part-one/>

## What is Entity Relationship Diagram (ERD)?

<https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/>

### All Sources

 en.wikipedia  c-sharpcorner  visual-paradigm