

EventEase Advanced Database Management

Technical Report

By: Kevin Anjalo & Navindu Ashen

Degree Program: BSc (Honours) in Computer Science

Degree Program: BSc (Honours) in Data Science

Table of Contents

1. Authorization Letter.....	2
2. Abstract	2
3. Introduction.....	3
4. Section 1- Database Design	3
4.1 Introduction to the Scenario	3
4.2 Entity-Relationship (ER) Model.....	3
4.3 Additional Assumptions.....	6
4.4 Database Schema and Normalization	7
4.5 Data Normalization.....	7
4.6 Data Dictionary	10
5. Section 2 - Database Implementation.....	13
5.1 Database and Table Creation.....	13
5.2 Sample Data Insertion	15
6. Section 3 – Create Trigger, Functions, Views & Procedure	18
6.1 Triggers	18
6.2 Functions	20
6.3 Views	23
6.4 Procedures	24
7. Section 4 - Web Application & Business Intelligence.....	26
7.1 Application Overview	26
7.2 Application Screenshots.....	27
7.3 Business Intelligence (BI)	29
8. Conclusion	31
9. References.....	31
10. Appendix.....	31
10.1 Code Snippets.....	31

1. Authorization Letter



To
Mr. NAK Dissanayake and Mr. YKA Rathnasiri,
NSBM Green University,
Batch 22.2,
22/MAY/2022

Authorization for EventEase Database Implementation

Dear Mr. NAK Dissanayake and Mr. YAK Rathnasiri,

I O N Groups Pvt Ltd, hereby authorizes you, Mr. N.A.K Dissanayake and Mr. Y.K.A Rathnasiri, as our employees and NSBM Green University students, to develop and implement the EventEase Database System.

The EventEase project aims to create a system to manage events, attendees, ticket bookings, feedback, and payments, with BI features for event analytics. You are tasked with designing the database, implementation and development of Web application and delivering all required documentation.

You have full permission to access necessary company resources, subject to confidentiality agreements.

For support, contact our CEO at induwara@ion-groups.com.

Sincerely,

ION GROUPS (PVT) LTD.


.....
Induwara Wickramasinghe
Director

Co-Founder & CEO
Induwara Wickramasinghe

INNOVATION & TECHNOLOGY COMPANY - PV 00217999

Level 26, East Tower, World Trade Center Echelon Square, Colombo 01, Sri Lanka.
+94 117 573 574 / +94 77 441 771 9 | info@ion-groups.com | www.ion-groups.com

2. Abstract

This report documents the design, implementation, and application development of the **EventEase** Database System. Developed for **ION Groups Pvt Ltd**, this system provides a robust solution for managing event data, including venues, organizers, attendees, tickets, feedback, and payments. Leveraging **Microsoft SQL Server**, the database

incorporates advanced features such as **triggers** for data integrity and automation, user defined **functions and views** for simplified data retrieval and analysis, and stored **procedures** for encapsulating complex logic. A sample Flask based **web application demonstrates basic system interaction**, focusing on event browsing and filtering. The report also details the implementation of key **business intelligence** queries to provide insights into event performance and attendee satisfaction.

3. Introduction

The EventEase database system is designed to manage event-related data for an event management platform, facilitating efficient organization, ticketing, and feedback analysis. The system handles entities such as events, venues, attendees, tickets, and feedback, ensuring robust data integrity and usability. Key facts considered include the need for realtime event filtering, revenue tracking, and attendee satisfaction analysis, as outlined in the project requirements. The solution leverages Microsoft SQL Server for data storage and a Flask-based web application for usability.

This report details the database design, implementation, and application development, covering entity-relationship modeling, normalization, SQL Server scripts, and a sample application. Additional features like triggers, functions, views, and procedures enhance functionality, while a basic business intelligence algorithm provides insights into event popularity. The project was undertaken for I O N Groups Pvt Ltd.

4. Section 1- Database Design

4.1 Introduction to the Scenario

EventEase is an event management platform that organizes concerts, conferences, meetups, and book launches. The system tracks event details (name, type, date, venue), attendee information, ticket sales, and feedback. Key requirements include:

- Storing event and venue data with relationships.
- Ensuring data integrity via constraints.
- Supporting queries for revenue, attendee counts, and feedback summaries.
- Providing a user-friendly interface for event browsing and filtering.

The solution uses a relational database in SQL Server, with a Flask application displaying event details via a web interface. The design prioritizes scalability, data consistency.

4.2 Entity-Relationship (ER) Model

An Entity-Relationship (ER) diagram was developed to visually represent the entities within the EventEase system and the relationships between them. This model serves as the blueprint for the database structure. The main entities identified are,

- **Venue** - Represents the physical location where events are hosted.
- **Organizer** - Represents the individual or organization responsible for managing an event.
- **Event** - Represents an organized activity, including details such as date, time, and type.

- **Attendee** - Represents individuals who register for and attend events.
- **Ticket** - Represents a reservation made by an attendee to attend a specific event.
- **Feedback** - Represents attendee-submitted feedback for events they've attended.
- **Payment** - Represents the transaction related to a booked ticket.

Relationship	Cardinality	Explanation
Venue – Event	1:M	One venue can host many events. Each event is hosted at one venue.
Organizer – Event	1:M	One organizer can manage multiple events. Each event is managed by one.
Event – Ticket	1:M	One event can have multiple tickets booked. Each ticket belongs to one event.
Attendee – Ticket	1:M	One attendee can book multiple tickets. Each ticket is linked to one attendee.
Ticket – Payment	1:1	Each ticket has one payment. Each payment belongs to one ticket.
Attendee – Feedback – Event	M:N (via Feedback)	An attendee can give feedback for many events, and an event can receive feedback from many attendees. Implemented using the Feedback entity as an associative table.

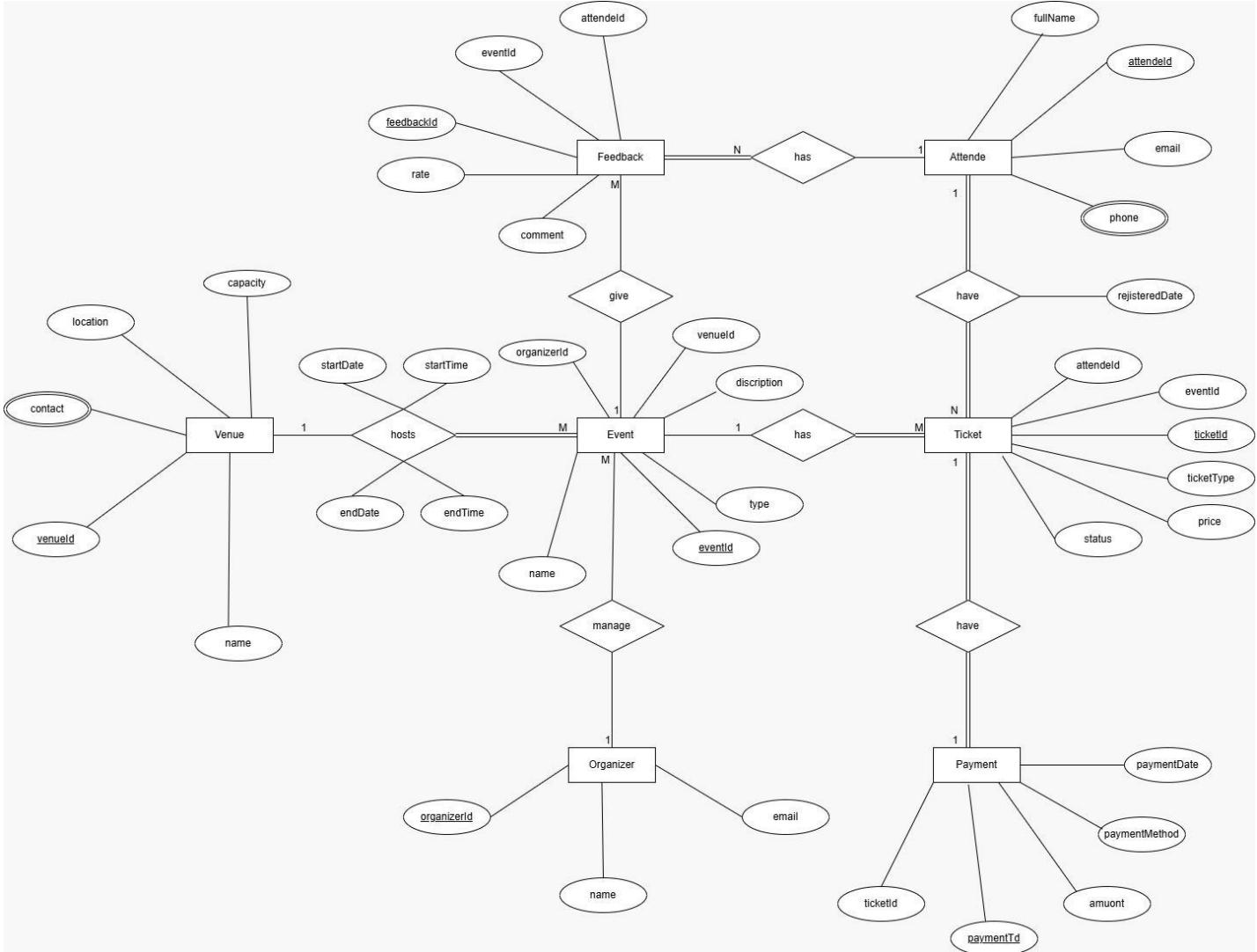


Image 1: Entity-Relationship Diagram for the EventEase Database System

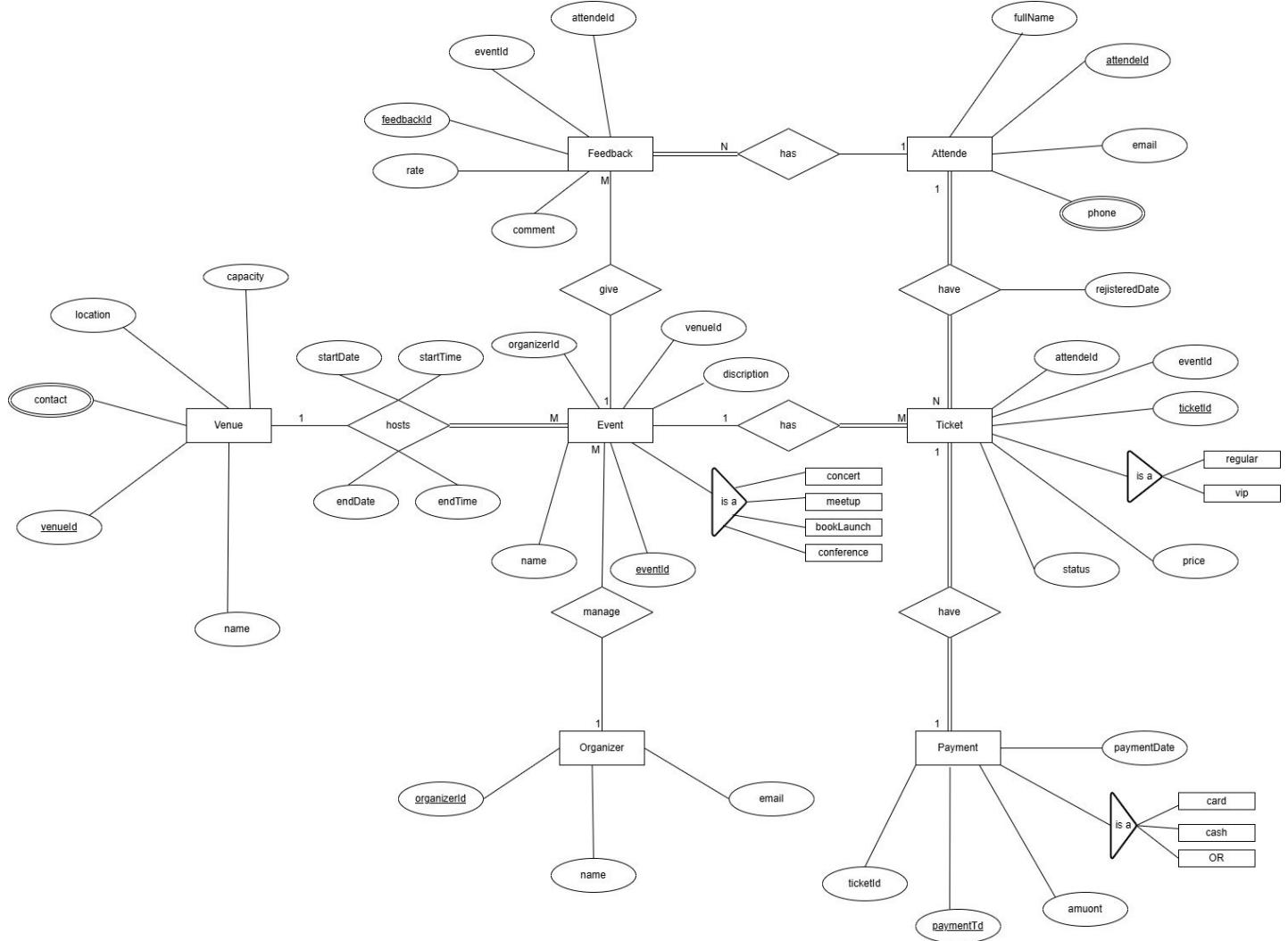


Image 2: Enhanced entity-relationship (EER) Diagram for the EventEase Database System

4.3 Additional Assumptions

- Events are unique by name and date.
- Venue capacity is fixed and updated only via triggers.
- Tickets have a single price per event.
- Feedback ratings are integers (1-5), with optional comments.
- Email addresses are unique for attendees.
- The database supports at least 10 records per table for testing.
- The Flask application uses client-side filtering for performance.

4.4 Database Schema and Normalization

The database schema is designed following normalization principles, aiming for Third Normal Form (3NF) to minimize data redundancy, reduce data anomalies (insertion, update, deletion), and improve data integrity. Each entity from the ER model is represented by a dedicated table, with foreign keys establishing relationships between tables.

Table Name	Purpose
Venue	Stores venue name, location, and capacity
Organizer	Stores organizer name and email
Event	Stores event details; links to Venue, Organizer, and EventType
Attendee	Stores attendee name and email
Ticket	Stores ticket price, status, and links to Event and Attendee
Payment	Stores payment details linked to Ticket
Feedback	Stores attendee ratings and comments linked to events
EventType	Stores categories/types of events (e.g., Meetup, Workshop)
TicketStatus	Stores status labels such as "Booked", "Paid", "Cancelled"
OrganizerContact, AttendeeContact, VenueContact	Separate contact details to avoid repetition
Ticket_Register	Stores registration date details for tickets
Event_Host	Stores event start/end date and time (scheduling info)

4.5 Data Normalization

Normalization ensures data integrity and eliminates redundancy, up to the Third Normal Form (3NF).

Step 1 - First Normal Form (1NF)

- All attributes are atomic (no multi-valued attributes).
- Each table has a PK.
- Example: Event table has atomic attributes (EventName, Type, etc.), with EventID as PK. **Step 2 -**

Second Normal Form (2NF)

- Must be in 1NF.

- All non-key attributes are fully dependent on the PK.
- Example: In Ticket, Price depends on TicketID (not partially on EventID or AttendeeID). All tables are in 2NF.

2NF. Step 3 - Third Normal Form (3NF)

- Must be in 2NF.
- No transitive dependencies (non-key attributes depend only on the PK).
- Example: In Attendee, FirstName, LastName, and Email depend on AttendeeID, not each other. No transitive dependencies exist.

All tables (Event, Venue, Attendee, Ticket, Feedback) are in 3NF, with no redundancy or anomalies.

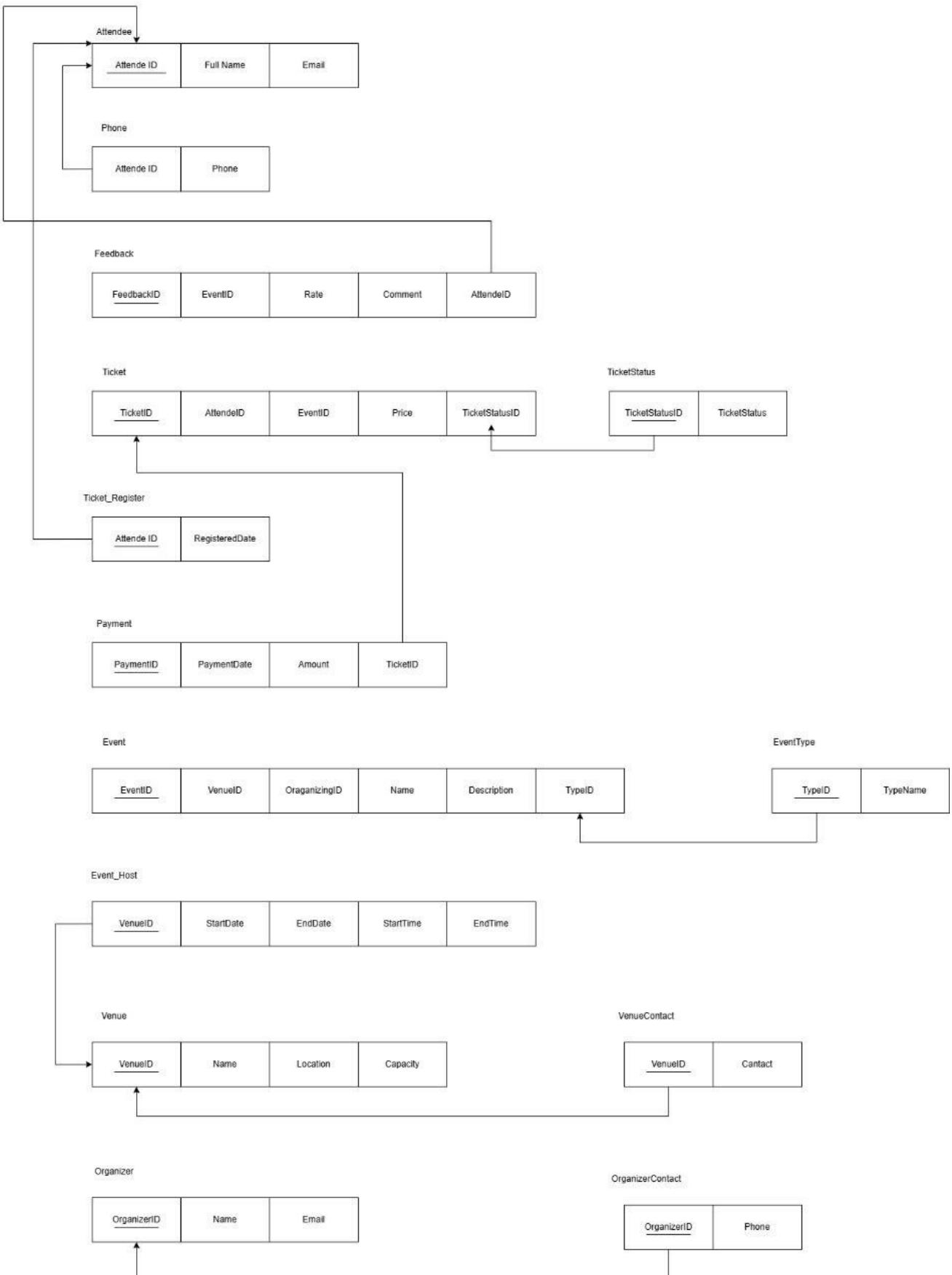


Image 2: Database Schema (Table Structures)

4.6 Data Dictionary

The data dictionary details each table's fields, including data types, constraints, and descriptions. **Event Table**

Field	Data Type	Constraints	Description
EventID	INT	PK, IDENTITY(1,1)	Unique event identifier
VenueID	INT	FK, NOT NULL	References Venue(VenueID)
OrganizerID	INT	FK, NOT NULL	References Organizer(OrganizerID)
Name	VARCHAR(100)	NOT NULL	Name of the event
Description	VARCHAR(500)	NULL	Event description
TypeID	INT	FK, NOT NULL	References EventType(TypeID)

Event_Host Table

Field	Data Type	Constraints	Description
VenueID	INT	FK, NOT NULL	References Venue(VenueID)
StartDate	DATE	NOT NULL	Event start date
EndDate	DATE	NOT NULL	Event end date
StartTime	TIME	NOT NULL	Event start time
EndTime	TIME	NOT NULL	Event end time

EventType Table

Field	Data Type	Constraints	Description
TypeID	INT	PK, IDENTITY(1,1)	Unique event type identifier
TypeName	VARCHAR(50)	NOT NULL, CHECK (TypeName IN ('Concert', 'Meetup', 'Book Launch', 'Conference'))	Event type name

Venue Table

Field	Data Type	Constraints	Description
VenueID	INT	PK, IDENTITY(1,1)	Unique venue identifier
Name	VARCHAR(100)	NOT NULL	Name of the venue
Location	VARCHAR(200)	NOT NULL	Venue location/address
Capacity	INT	NOT NULL, CHECK (Capacity > 0)	Maximum attendee capacity

VenueContact Table

Field	Data Type	Constraints	Description
VenueID	INT	FK, NOT NULL	References Venue(VenueID)
Contact	VARCHAR(15)	NOT NULL	Venue contact phone number

Organizer Table

Field	Data Type	Constraints	Description
OrganizerID	INT	PK, IDENTITY(1,1)	Unique organizer identifier
Name	VARCHAR(100)	NOT NULL	Organizer name
Email	VARCHAR(100)	NOT NULL, UNIQUE	Organizer email address

OrganizerContact Table

Field	Data Type	Constraints	Description
OrganizerID	INT	FK, NOT NULL	References Organizer(OrganizerID)
Phone	VARCHAR(15)	NOT NULL	Organizer phone number

Attendee Table

Field	Data Type	Constraints	Description
AttendeeID	INT	PK, IDENTITY(1,1)	Unique attendee identifier
FullName	VARCHAR(100)	NOT NULL	Attendee's full name
Email	VARCHAR(100)	NOT NULL, UNIQUE	Attendee's email address

Phone Table (Attendee Contact)

Field	Data Type	Constraints	Description
AttendeeID	INT	FK, NOT NULL	References Attendee(AttendeeID)
Phone	VARCHAR(15)	NOT NULL	Attendee phone number

Ticket Table

Field	Data Type	Constraints	Description
TicketID	INT	PK, IDENTITY(1,1)	Unique ticket identifier
AttendeeID	INT	FK, NOT NULL	References Attendee(AttendeeID)
EventID	INT	FK, NOT NULL	References Event(EventID)
Price	DECIMAL(10,2)	NOT NULL, CHECK (Price >= 0)	Ticket price
TicketStatusID	INT	FK, NOT NULL	References TicketStatus(TicketStatusID)

TicketStatus Table

Field	Data Type	Constraints	Description
TicketStatusID	INT	PK, IDENTITY(1,1)	Unique ticket status identifier
TicketStatus	VARCHAR(20)	NOT NULL, CHECK (TicketStatus IN ('Booked', 'Paid', 'Cancelled'))	Ticket status description

Ticket_Register Table

Field	Data Type	Constraints	Description
AttendeeID	INT	FK, NOT NULL	References Attendee(AttendeeID)
RegisteredDate	DATE	NOT NULL	Date when ticket was registered

Payment Table

Field	Data Type	Constraints	Description
PaymentID	INT	PK, IDENTITY(1,1)	Unique payment identifier
PaymentDate	DATE	NOT NULL	Date of payment
Amount	DECIMAL(10,2)	NOT NULL, CHECK (Amount >= 0)	Payment amount
TicketID	INT	FK, NOT NULL	References Ticket(TicketID)

Feedback Table

Field	Data Type	Constraints	Description
FeedbackID	INT	PK, IDENTITY(1,1)	Unique feedback identifier
EventID	INT	FK, NOT NULL	References Event(EventID)
Rate	INT	NOT NULL, CHECK (Rate BETWEEN 1 AND 5)	Feedback rating (1-5 scale)
Comment	VARCHAR(500)	NULL	Optional feedback comment
AttendeeID	INT	FK, NOT NULL	References Attendee(AttendeeID)

Relationships

Primary Relationships

- Venue ↔ Event:** One-to-Many (One venue can host multiple events)
- Organizer ↔ Event:** One-to-Many (One organizer can manage multiple events)
- Event ↔ Ticket:** One-to-Many (One event can have multiple tickets)
- Attendee ↔ Ticket:** One-to-Many (One attendee can book multiple tickets)
- Ticket ↔ Payment:** One-to-One (Each ticket has one payment)
- EventType ↔ Event:** One-to-Many (One event type can categorize multiple events)
- TicketStatus ↔ Ticket:** One-to-Many (One status can apply to multiple tickets)

Many-to-Many Relationships

- Attendee ↔ Event:** Many-to-Many through Feedback table (Attendees can provide feedback for multiple events, events can receive feedback from multiple attendees)

Contact Information

- Separate contact tables maintain normalized structure for phone numbers
- Email addresses are stored directly in main entity tables with unique constraints

Data Integrity Rules

Check Constraints

- Venue capacity must be greater than 0
- Ticket price must be non-negative
- Payment amount must be non-negative
- Feedback rating must be between 1 and 5
- Event types restricted to: Concert, Meetup, Book Launch, Conference

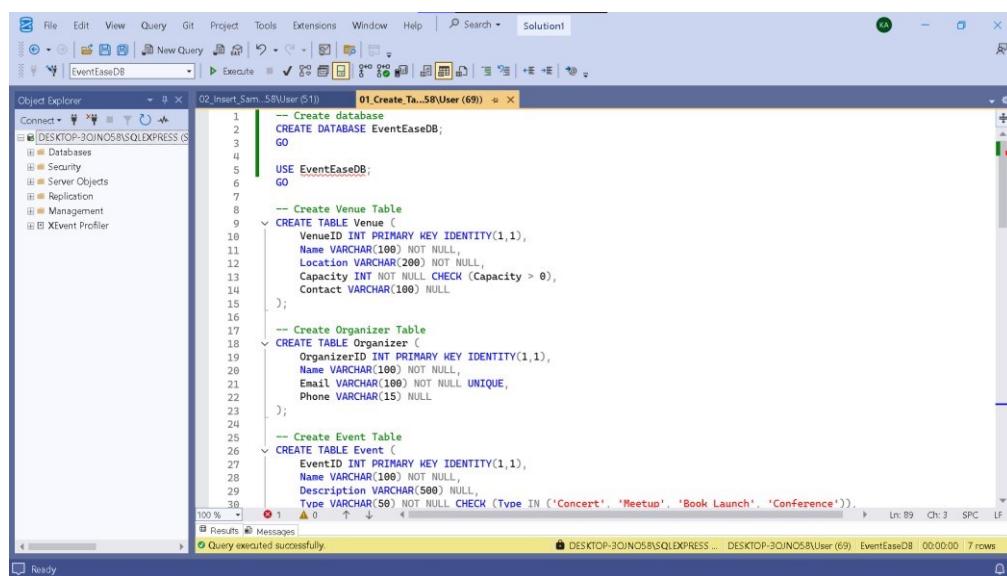
- Ticket status restricted to: Booked, Paid, Cancelled **Unique Constraints**
- Attendee email addresses must be unique
- Organizer email addresses must be unique **Foreign Key Constraints**
- All foreign key relationships enforce referential integrity
- Cascading rules ensure data consistency across related tables

5. Section 2 - Database Implementation

The database schema and objects were implemented using SQL scripts in Microsoft SQL Server Management Studio (SSMS).

5.1 Database and Table Creation

The first step was to create the EventEaseDB database and then define each table with its columns, data types, primary keys, identity specifications, foreign keys, and check constraints.



```

File Edit View Query Git Project Tools Extensions Window Help | Search | Solution1
File Edit View Query Git Project Tools Extensions Window Help | Search | Solution1
Object Explorer | 02_Insert_Sam...SBUser (51) | 01_Create Ta...SBUser (69) |
Connect | New Query | Run | Stop | Refresh | Back | Forward | Home | Favorites | Task List | Help | Options | Exit | 
[ DESKTOP-3OINOS5\SQLEXPRESS ] [ EventEaseDB ] | Execute | 
-- Create database
CREATE DATABASE EventEaseDB;
GO
USE EventEaseDB;
GO
-- Create Venue Table
CREATE TABLE Venue (
    VenueID INT PRIMARY KEY IDENTITY(1,1),
    Name VARCHAR(100) NOT NULL,
    Location VARCHAR(200) NOT NULL,
    Capacity INT NOT NULL CHECK (Capacity > 0),
    Contact VARCHAR(100) NULL
);
-- Create Organizer Table
CREATE TABLE Organizer (
    OrganizerID INT PRIMARY KEY IDENTITY(1,1),
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    Phone VARCHAR(15) NULL
);
-- Create Event Table
CREATE TABLE Event (
    EventID INT PRIMARY KEY IDENTITY(1,1),
    Name VARCHAR(100) NOT NULL,
    Description VARCHAR(500) NULL,
    Type VARCHAR(50) NOT NULL CHECK (Type IN ('Concert', 'Meetup', 'Book Launch', 'Conference'))
);

```

Results | Messages | DESKTOP-3OINOS5\SQLEXPRESS... DESKTOP-3OINOS5\SBUser (69) EventEaseDB 00:00:00 | 7 rows

Query executed successfully.

File Edit View Query Git Project Tools Extensions Window Help Search Solution

EventEaseDB 02_Insert_Sam_58\User (51) 01.Create Ta...58\User (69)

```

25  -- Create Event Table
26  CREATE TABLE Event (
27      EventID INT PRIMARY KEY IDENTITY(1,1),
28      Name VARCHAR(100) NOT NULL,
29      Description VARCHAR(500) NULL,
30      Type VARCHAR(50) NOT NULL CHECK (Type IN ('Concert', 'Meetup', 'Book Launch', 'Conference')),
31      StartDate DATE NOT NULL,
32      EndDate DATE NOT NULL,
33      StartTime TIME NOT NULL,
34      EndTime TIME NOT NULL,
35      VenueID INT NOT NULL,
36      OrganizerID INT NOT NULL,
37      CONSTRAINT FK_Event_Venue FOREIGN KEY (VenueID) REFERENCES Venue(VenueID),
38      CONSTRAINT FK_Event_Organizer FOREIGN KEY (OrganizerID) REFERENCES Organizer(OrganizerID),
39      CONSTRAINT CHK_EndDate CHECK (EndDate >= StartDate)
40  );
41
42  -- Create Attendee Table
43  CREATE TABLE Attendee (
44      AttendeeID INT PRIMARY KEY IDENTITY(1,1),
45      FullName VARCHAR(100) NOT NULL,
46      Email VARCHAR(100) NOT NULL UNIQUE,
47      Phone VARCHAR(15) NULL,
48      RegisteredDate DATE NOT NULL DEFAULT GETDATE()
49  );
50
51  -- Create Ticket Table
52  CREATE TABLE Ticket (
53      TicketID INT PRIMARY KEY IDENTITY(1,1),
54      EventID INT NOT NULL,
55      AttendeeID INT NOT NULL,
56      TicketTypeID INT NOT NULL,
57      Price DECIMAL(10,2) NOT NULL CHECK (Price >= 0),
58      Status VARCHAR(20) NOT NULL CHECK (Status IN ('Booked', 'Available', 'Cancelled')),
59      CONSTRAINT FK_Ticket_Event FOREIGN KEY (EventID) REFERENCES Event(EventID),
60      CONSTRAINT FK_Ticket_Attendee FOREIGN KEY (AttendeeID) REFERENCES Attendee(AttendeeID)
61  );
62
63  -- Create Feedback Table
64  CREATE TABLE Feedback (
65      FeedbackID INT PRIMARY KEY IDENTITY(1,1),
66      EventID INT NOT NULL,
67      AttendeeID INT NOT NULL,
68      Rating INT NOT NULL CHECK (Rating BETWEEN 1 AND 5),
69      Comment VARCHAR(500) NULL,
70      SubmittedDate DATE NOT NULL DEFAULT GETDATE(),
71      CONSTRAINT FK_Feedback_Event FOREIGN KEY (EventID) REFERENCES Event(EventID),
72      CONSTRAINT FK_Feedback_Attendee FOREIGN KEY (AttendeeID) REFERENCES Attendee(AttendeeID)
73  );
74
75  -- Create Payment Table
76  CREATE TABLE Payment (
77      PaymentID INT PRIMARY KEY IDENTITY(1,1),
78      TicketID INT NOT NULL,
79      PaymentDate DATE NOT NULL DEFAULT GETDATE(),
80      Amount DECIMAL(10,2) NOT NULL CHECK (Amount >= 0),
81      PaymentMethod VARCHAR(20) NOT NULL CHECK (PaymentMethod IN ('Card', 'Cash', 'QR')),
82      CONSTRAINT FK_Payment_Ticket FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)
83  );
84
85  -- Test: Verify table creation
86  SELECT TABLE_NAME
87  FROM INFORMATION_SCHEMA.TABLES
88  WHERE TABLE_CATALOG = 'EventEaseDB' AND TABLE_TYPE = 'BASE TABLE';
89  GO

```

Results Messages

Query executed successfully.

DESKTOP-3OINOS8\SQLEXPRESS ... DESKTOP-3OINOS8\User (69) EventEaseDB 00:00:00 7 rows

Ready

File Edit View Query Git Project Tools Extensions Window Help Search Solution

EventEaseDB 02_Insert_Sam_58\User (51) 01.Create Ta...58\User (69)

```

49  );
50
51  -- Create Ticket Table
52  CREATE TABLE Ticket (
53      TicketID INT PRIMARY KEY IDENTITY(1,1),
54      EventID INT NOT NULL,
55      AttendeeID INT NOT NULL,
56      TicketTypeID INT NOT NULL,
57      Price DECIMAL(10,2) NOT NULL CHECK (Price >= 0),
58      Status VARCHAR(20) NOT NULL CHECK (Status IN ('Booked', 'Available', 'Cancelled')),
59      CONSTRAINT FK_Ticket_Event FOREIGN KEY (EventID) REFERENCES Event(EventID),
60      CONSTRAINT FK_Ticket_Attendee FOREIGN KEY (AttendeeID) REFERENCES Attendee(AttendeeID)
61  );
62
63  -- Create Feedback Table
64  CREATE TABLE Feedback (
65      FeedbackID INT PRIMARY KEY IDENTITY(1,1),
66      EventID INT NOT NULL,
67      AttendeeID INT NOT NULL,
68      Rating INT NOT NULL CHECK (Rating BETWEEN 1 AND 5),
69      Comment VARCHAR(500) NULL,
70      SubmittedDate DATE NOT NULL DEFAULT GETDATE(),
71      CONSTRAINT FK_Feedback_Event FOREIGN KEY (EventID) REFERENCES Event(EventID),
72      CONSTRAINT FK_Feedback_Attendee FOREIGN KEY (AttendeeID) REFERENCES Attendee(AttendeeID)
73  );
74
75  -- Create Payment Table
76  CREATE TABLE Payment (
77      PaymentID INT PRIMARY KEY IDENTITY(1,1),
78      TicketID INT NOT NULL,
79      PaymentDate DATE NOT NULL DEFAULT GETDATE(),
80      Amount DECIMAL(10,2) NOT NULL CHECK (Amount >= 0),
81      PaymentMethod VARCHAR(20) NOT NULL CHECK (PaymentMethod IN ('Card', 'Cash', 'QR')),
82      CONSTRAINT FK_Payment_Ticket FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)
83  );
84
85  -- Test: Verify table creation
86  SELECT TABLE_NAME
87  FROM INFORMATION_SCHEMA.TABLES
88  WHERE TABLE_CATALOG = 'EventEaseDB' AND TABLE_TYPE = 'BASE TABLE';
89  GO

```

Results Messages

Query executed successfully.

DESKTOP-3OINOS8\SQLEXPRESS ... DESKTOP-3OINOS8\User (69) EventEaseDB 00:00:00 7 rows

Ready

File Edit View Query Git Project Tools Extensions Window Help Search Solution

EventEaseDB 02_Insert_Sam_58\User (51) 01.Create Ta...58\User (69)

```

73  );
74
75  -- Create Payment Table
76  CREATE TABLE Payment (
77      PaymentID INT PRIMARY KEY IDENTITY(1,1),
78      TicketID INT NOT NULL,
79      PaymentDate DATE NOT NULL DEFAULT GETDATE(),
80      Amount DECIMAL(10,2) NOT NULL CHECK (Amount >= 0),
81      PaymentMethod VARCHAR(20) NOT NULL CHECK (PaymentMethod IN ('Card', 'Cash', 'QR')),
82      CONSTRAINT FK_Payment_Ticket FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)
83  );
84
85  -- Test: Verify table creation
86  SELECT TABLE_NAME
87  FROM INFORMATION_SCHEMA.TABLES
88  WHERE TABLE_CATALOG = 'EventEaseDB' AND TABLE_TYPE = 'BASE TABLE';
89  GO

```

Results Messages

Query executed successfully.

DESKTOP-3OINOS8\SQLEXPRESS ... DESKTOP-3OINOS8\User (69) EventEaseDB 00:00:00 7 rows

Ready

Image 3: Queries for creating the EventEase database and its core tables (Venue, Organizer, Event, Attendee, Ticket, Feedback, Payment).

5.2 Sample Data Insertion

Sample data was inserted into the tables to populate the database for testing and demonstration purposes. This includes data for venues, organizers, events, attendees, tickets, feedback, and payments.

File Edit View Query Git Project Tools Extensions Window Help Search Solution

Object Explorer

```

USE EventEaseDB;
GO
-- Insert into Venue
INSERT INTO Venue (Name, Location, Capacity, Contact) VALUES
('Colombo Convention Center', 'Colombo, Sri Lanka', 1000, 'contact@ccc.lk'),
('Galle Face Hotel', 'Colombo, Sri Lanka', 500, 'info@gallehotel.lk'),
('BMICH', 'Colombo, Sri Lanka', 1500, 'bmich@bmich.lk'),
('Hilton Colombo', 'Colombo, Sri Lanka', 800, 'hilton@colombo.lk'),
('Shangri-La Colombo', 'Colombo, Sri Lanka', 600, 'shangri@colombo.lk'),
('Neum Pokuna', 'Colombo, Sri Lanka', 1200, 'neum@colombo.lk'),
('Viharamahadevi Park', 'Colombo, Sri Lanka', 2000, NULL),
('Taj Samudra', 'Colombo, Sri Lanka', 700, 'taj@colombo.lk'),
('Cinnamon Grand', 'Colombo, Sri Lanka', 900, 'cinnamon@colombo.lk'),
('Kingsbury Hotel', 'Colombo, Sri Lanka', 400, 'kingsbury@colombo.lk'),
('Jetwing Lighthouse', 'Galle, Sri Lanka', 300, 'jetwinggalle.lk'),
('Heritance Kandalama', 'Dambulla, Sri Lanka', 600, 'heritance@dambulla.lk');

```

-- Insert into Organizer

```

INSERT INTO Organizer (Name, Email, Phone) VALUES
('EventMasters', 'eventmasters@gmail.com', '0112345678'),
('CelebrateNow', 'celebrate@now.lk', '0113456789'),
('JoyEvents', 'joyevents@gmail.com', '0145678901'),
('StarPlanners', 'star@planners.lk', '0115678901'),
('EliteEvents', 'elite@events.lk', '0117890123'),
('VibeOrganizers', 'vibe@organizers.lk', '0117890123'),
('DreamEvents', 'dream@events.lk', '0118901234'),
('FestiveCo', 'festive@co.lk', '01198012345'),
('PlanIt', 'planit@gmail.com', '0110123456'),
('EventEvents', 'event@events.lk', '0111234567')

```

Query executed successfully.

File Edit View Query Git Project Tools Extensions Window Help Search Solution

Object Explorer

```

-- Insert into Organizer
INSERT INTO Organizer (Name, Email, Phone) VALUES
('EventMasters', 'eventmasters@gmail.com', '0112345678'),
('CelebrateNow', 'celebrate@now.lk', '0113456789'),
('JoyEvents', 'joyevents@gmail.com', '0145678901'),
('StarPlanners', 'star@planners.lk', '0115678901'),
('EliteEvents', 'elite@events.lk', '01167890123'),
('VibeOrganizers', 'vibe@organizers.lk', '0117890123'),
('DreamEvents', 'dream@events.lk', '0118901234'),
('FestiveCo', 'festive@co.lk', '01198012345'),
('PlanIt', 'planit@gmail.com', '0110123456'),
('EventEvents', 'event@events.lk', '0111234567'),
('GalaGurus', 'gala@gurus.lk', '0112345678'),
('SparkleEvents', 'sparkle@events.lk', '0113456789');

```

-- Insert into Event

```

INSERT INTO Event (Name, Description, Type, StartDate, EndDate, StartTime, EndTime, VenueID, OrganizerID) VALUES
('Tech Conference 2025', 'Annual tech summit', 'Conference', '2025-06-15', '2025-06-17', '09:00', '17:00', 1, 1),
('Rock Concert', 'Live music performance', 'Concert', '2025-07-10', '2025-07-10', '19:00', '23:00', 2, 2),
('Book Launch: Future', 'New sci-fi novel launch', 'Book Launch', '2025-08-05', '2025-08-05', '18:00', '20:00', 3, 3),
('Startup Meetup', 'Networking for startups', 'Meetup', '2025-09-12', '2025-09-12', '14:00', '17:00', 4, 4),
('Jazz Night', 'Evening of jazz music', 'Concert', '2025-10-20', '2025-10-20', '20:00', '22:00', 5, 5),
('AI Summit', 'AI technology conference', 'Conference', '2025-11-15', '2025-11-16', '09:00', '16:00', 6, 6),
('Poetry Reading', 'Poetry and literature evening', 'Book Launch', '2025-12-01', '2025-12-01', '18:00', '20:00', 7, 7),
('Food Festival', 'Culinary event', 'Meetup', '2025-12-10', '2025-12-10', '10:00', '22:00', 8, 8),
('Charity Concert', 'Fundraising music event', 'Concert', '2026-01-05', '2026-01-05', '19:00', '23:00', 9, 9),
('Tech Meetup', 'Tech enthusiasts gathering', 'Meetup', '2026-02-15', '2026-02-15', '15:00', '18:00', 10, 10),
('Art Exhibition', 'Showcasing local art', 'Meetup', '2026-03-12', '2026-03-12', '10:00', '18:00', 11, 11),
('Gala Dinner', 'Formal dinner event', 'Conference', '2026-04-20', '2026-04-20', '19:00', '22:00', 12, 12);

```

Query executed successfully.

File Edit View Query Git Project Tools Extensions Window Help Search Solution

Object Explorer

```

-- Insert into Event
INSERT INTO Event (Name, Description, Type, StartDate, EndDate, StartTime, EndTime, VenueID, OrganizerID) VALUES
('Tech Conference 2025', 'Annual tech summit', 'Conference', '2025-06-15', '2025-06-17', '09:00', '17:00', 1, 1),
('Rock Concert', 'Live music performance', 'Concert', '2025-07-10', '2025-07-10', '19:00', '23:00', 2, 2),
('Book Launch: Future', 'New sci-fi novel launch', 'Book Launch', '2025-08-05', '2025-08-05', '18:00', '20:00', 3, 3),
('Startup Meetup', 'Networking for startups', 'Meetup', '2025-09-12', '2025-09-12', '14:00', '17:00', 4, 4),
('Jazz Night', 'Evening of jazz music', 'Concert', '2025-10-20', '2025-10-20', '20:00', '22:00', 5, 5),
('AI Summit', 'AI technology conference', 'Conference', '2025-11-15', '2025-11-16', '09:00', '16:00', 6, 6),
('Poetry Reading', 'Poetry and literature evening', 'Book Launch', '2025-12-01', '2025-12-01', '18:00', '20:00', 7, 7),
('Food Festival', 'Culinary event', 'Meetup', '2025-12-10', '2025-12-10', '10:00', '22:00', 8, 8),
('Charity Concert', 'Fundraising music event', 'Concert', '2026-01-05', '2026-01-05', '19:00', '23:00', 9, 9),
('Tech Meetup', 'Tech enthusiasts gathering', 'Meetup', '2026-02-15', '2026-02-15', '15:00', '18:00', 10, 10),
('Art Exhibition', 'Showcasing local art', 'Meetup', '2026-03-12', '2026-03-12', '10:00', '18:00', 11, 11),
('Gala Dinner', 'Formal dinner event', 'Conference', '2026-04-20', '2026-04-20', '19:00', '22:00', 12, 12);

-- Insert into Attendee
INSERT INTO Attendee (FullName, Email, Phone, RegisteredDate) VALUES
('John Doe', 'john.doe@gmail.com', '0771234567', '2025-05-01'),
('Jane Smith', 'jane.smith@gmail.com', '0772345678', '2025-05-02'),
('Alice Brown', 'alice.brown@gmail.com', '0773456789', '2025-05-03'),
('Bob Wilson', 'bob.wilson@gmail.com', '0774567890', '2025-05-04'),
('Emma Davis', 'emma.davis@gmail.com', '0775678901', '2025-05-05'),
('Liam Johnson', 'liam.johnson@gmail.com', '0776789012', '2025-05-06'),
('Olivia Lee', 'olivia.lee@gmail.com', '0777890123', '2025-05-07'),
('Noah Clark', 'noah.clark@gmail.com', '0778901234', '2025-05-08'),
('Sophia Adams', 'sophia.adams@gmail.com', '0779012345', '2025-05-09'),
('James Taylor', 'james.taylor@gmail.com', '0770123456', '2025-05-10'),
('Mia Harris', 'mia.harris@gmail.com', '0771234567', '2025-05-11'),
('Ethan Moore', 'ethan.moore@gmail.com', '0772345679', '2025-05-12');

-- Insert into Ticket

```

Query executed successfully.

File Edit View Query Git Project Tools Extensions Window Help

EventEaseDB

Object Explorer

```

64 -- Insert into Ticket
65 INSERT INTO Ticket (EventID, AttendeeID, TicketType, Price, Status) VALUES
66 (1, 1, 'Regular', 50.00, 'Booked'),
67 (1, 2, 'VIP', 100.00, 'Booked'),
68 (2, 3, 'Regular', 75.00, 'Booked'),
69 (2, 4, 'Regular', 75.00, 'Booked'),
70 (3, 5, 'Regular', 20.00, 'Booked'),
71 (3, 6, 'Regular', 20.00, 'Booked'),
72 (4, 7, 'Regular', 30.00, 'Booked'),
73 (4, 8, 'Regular', 30.00, 'Booked'),
74 (5, 9, 'VIP', 150.00, 'Booked'),
75 (5, 10, 'Regular', 80.00, 'Booked'),
76 (6, 11, 'Regular', 60.00, 'Booked'),
77 (6, 12, 'VIP', 120.00, 'Booked');
78
79 -- Insert into Feedback
80 INSERT INTO Feedback (EventID, AttendeeID, Rating, Comment, SubmittedDate) VALUES
81 (1, 1, 4, 'Great event, good speakers', '2025-06-18'),
82 (1, 2, 5, 'Amazing experience!', '2025-06-18'),
83 (2, 3, 3, 'Sound quality could be better', '2025-07-11'),
84 (2, 4, 4, 'Enjoyed the concert!', '2025-07-11'),
85 (3, 5, 5, 'Loved the book launch!', '2025-08-06'),
86 (3, 6, 4, 'Interesting event!', '2025-08-06'),
87 (4, 7, 4, 'Good networking', '2025-09-13'),
88 (4, 8, 3, 'More sessions needed', '2025-09-13'),
89 (5, 9, 5, 'Fantastic jazz night!', '2025-10-21'),
90 (5, 10, 4, 'Great atmosphere!', '2025-10-21'),
91 (6, 11, 4, 'Informative summit', '2025-11-17'),
92 (6, 12, 5, 'Well organized', '2025-11-17');
93
94 -- Insert into Payment
95 
```

No issues found

Query executed successfully.

DESKTOP-3QJNOS8\SQLEXPRESS ... DESKTOP-3QJNOS8\User (51) EventEaseDB 00:00:00 7 rows

File Edit View Query Git Project Tools Extensions Window Help

EventEaseDB

Object Explorer

```

91 -- Insert into Payment
92 INSERT INTO Payment (TicketID, PaymentDate, Amount, PaymentMethod) VALUES
93 (1, '2025-05-01', 50.00, 'Card'),
94 (2, '2025-05-02', 100.00, 'Card'),
95 (3, '2025-05-03', 75.00, 'QR'),
96 (4, '2025-05-04', 75.00, 'Cash'),
97 (5, '2025-05-05', 20.00, 'Card'),
98 (6, '2025-05-06', 20.00, 'QR'),
99 (7, '2025-05-07', 30.00, 'Card'),
100 (8, '2025-05-08', 30.00, 'Cash'),
101 (9, '2025-05-09', 150.00, 'Card'),
102 (10, '2025-05-10', 80.00, 'QR'),
103 (11, '2025-05-11', 60.00, 'Card'),
104 (12, '2025-05-12', 120.00, 'Card');

105
106 -- Test: Verify data insertion
107 SELECT 'Venue' AS TableName, COUNT(*) AS RecordCount FROM Venue
108 UNION
109 SELECT 'Organizer', COUNT(*) FROM Organizer
110 UNION
111 SELECT 'Event', COUNT(*) FROM Event
112 UNION
113 SELECT 'Attendee', COUNT(*) FROM Attendee
114 UNION
115 SELECT 'Ticket', COUNT(*) FROM Ticket
116 UNION
117 SELECT 'Feedback', COUNT(*) FROM Feedback
118 UNION
119 SELECT 'Payment', COUNT(*) FROM Payment;
120 GO
121
122 
```

No issues found

Query executed successfully.

DESKTOP-3QJNOS8\SQLEXPRESS ... DESKTOP-3QJNOS8\User (51) EventEaseDB 00:00:00 7 rows

File Edit View Query Git Project Tools Extensions Window Help

EventEaseDB

Object Explorer

```

109 -- Test: Verify data insertion
110 SELECT 'Venue' AS TableName, COUNT(*) AS RecordCount FROM Venue
111 UNION
112 SELECT 'Organizer', COUNT(*) FROM Organizer
113 UNION
114 SELECT 'Event', COUNT(*) FROM Event
115 UNION
116 SELECT 'Attendee', COUNT(*) FROM Attendee
117 UNION
118 SELECT 'Ticket', COUNT(*) FROM Ticket
119 UNION
120 SELECT 'Feedback', COUNT(*) FROM Feedback
121 UNION
122 SELECT 'Payment', COUNT(*) FROM Payment;
123 GO
124 
```

No issues found

TableName	RecordCount
Attendee	12
Event	12
Feedback	12
Organizer	12
Payment	12
Ticket	12
Venue	12

Query executed successfully.

DESKTOP-3QJNOS8\SQLEXPRESS ... DESKTOP-3QJNOS8\User (51) EventEaseDB 00:00:00 7 rows

Image 3: Queries for inserting sample data into the EventEase tables to populate the database.

6. Section 3 – Create Trigger, Functions, Views & Procedure

6.1 Triggers

Triggers were implemented to automate actions and maintain data integrity in response to data modifications.

I. Update Venue Capacity Trigger

A trigger named UpdateVenueCapacity is created on the Ticket table. It fires AFTER INSERT. When a new ticket is inserted with Status = 'Booked', it decrements the Capacity of the corresponding Venue by 1. This ensures the available capacity is updated in real-time upon booking.

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, the database 'EventEaseDB' is selected. In the center pane, a script window titled '03_Create.Tr...58(User (78))' contains the T-SQL code for creating the trigger:

```
USE EventEaseDB;
GO

-- Create Trigger
CREATE TRIGGER UpdateVenueCapacity
ON Ticket
AFTER INSERT
AS
BEGIN
    UPDATE v
    SET v.Capacity = v.Capacity - 1
    FROM Venue v
    INNER JOIN Event e ON v.VenueID = e.VenueID
    INNER JOIN inserted i ON e.EventID = i.EventID
    WHERE i.Status = 'Booked';
END;
GO

-- Test: Insert a ticket and check venue capacity
SELECT VenueID, Name, Capacity FROM Venue WHERE VenueID = 1;
INSERT INTO Ticket (EventID, AttendeeID, TicketType, Price, Status)
VALUES (1, 1, 'Regular', 50.00, 'Booked');
SELECT VenueID, Name, Capacity FROM Venue WHERE VenueID = 1;
```

Below the script, the results of the test query are shown in a table:

VenueID	Name	Capacity
1	Colombo Convention Center	999
1	Colombo Convention Center	998

The status bar at the bottom indicates 'Query executed successfully.'

Image 4: The UpdateVenueCapacity trigger and a test showing capacity decrease.

II. Audit Feedback Trigger

A trigger named AuditFeedback is created on the Feedback table. It fires AFTER INSERT. When new feedback is inserted, this trigger logs the details of the inserted row into a separate FeedbackAudit table, along with the action ('INSERT') and a timestamp. This provides a basic audit trail for feedback entries.

```

USE EventEaseDB;
GO
-- Create FeedbackAudit Table
CREATE TABLE FeedbackAudit (
    AuditID INT PRIMARY KEY IDENTITY(1,1),
    FeedbackID INT,
    EventID INT,
    AttendeeID INT,
    Rating INT,
    Comment VARCHAR(500),
    SubmittedDate DATE,
    AuditAction VARCHAR(50),
    AuditDate DATETIME DEFAULT GETDATE()
);
-- Create Trigger
CREATE TRIGGER AuditFeedback
ON Feedback
AFTER INSERT
AS
BEGIN
    INSERT INTO FeedbackAudit (FeedbackID, EventID, AttendeeID, Rating, Comment, SubmittedDate, AuditAction)
    SELECT FeedbackID, EventID, AttendeeID, Rating, Comment, SubmittedDate, 'INSERT'
    FROM inserted;
END;
GO
-- Test: Insert feedback and check audit table
INSERT INTO Feedback (EventID, AttendeeID, Rating, Comment, SubmittedDate)
VALUES (1, 1, 5, 'Excellent organization!', '2025-06-18');
SELECT AuditID, FeedbackID, EventID, AttendeeID, Rating, Comment, AuditAction, AuditDate
FROM FeedbackAudit
WHERE EventID = 1 AND AttendeeID = 1;
GO

```

The screenshot shows two separate queries in the SQL Server Management Studio Object Explorer. The top query creates a 'FeedbackAudit' table with columns for AuditID (primary key), FeedbackID, EventID, AttendeeID, Rating, Comment, SubmittedDate, AuditAction, and AuditDate. It also creates a trigger named 'AuditFeedback' on the 'Feedback' table that inserts records into 'FeedbackAudit' whenever a new record is inserted into 'Feedback'. The bottom query tests this by inserting a record into 'Feedback' with EventID 1 and AttendeeID 1, rating 5, and comment 'Excellent organization!'. It then selects all records from 'FeedbackAudit' where EventID is 1 and AttendeeID is 1, showing two rows: one for the insert and one for the trigger's insertion.

Image 5: The AuditFeedback trigger and a test showing a logged audit entry.

III. Prevent Event Deletion Trigger

A trigger named PreventEventDeletion is created on the Event table. It uses INSTEAD OF DELETE. Before an event is deleted, the trigger checks if there are any associated tickets with Status = 'Booked'. If booked tickets exist, it raises an error message (RAISERROR) and rolls back the transaction, preventing the deletion. Otherwise, it proceeds with the deletion of the event.

```

USE EventEaseDB;
GO

-- Create Trigger
CREATE TRIGGER PreventEventDeletion
ON Event
INSTEAD OF DELETE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM Ticket t
        INNER JOIN deleted d ON t.EventID = d.EventID
        WHERE t.Status = 'Booked'
    )
    BEGIN
        RAISERROR ('Cannot delete event with booked tickets.', 16, 1);
        ROLLBACK;
    END
    ELSE
    BEGIN
        DELETE FROM Event WHERE EventID IN (SELECT EventID FROM deleted);
    END
END;
GO

-- Test: Attempt to delete EventID 1
DELETE FROM Event WHERE EventID = 1;
SELECT EventID, Name FROM Event WHERE EventID = 1;
GO

```

Query executed successfully.

```

GO

-- Test: Attempt to delete EventID 1
DELETE FROM Event WHERE EventID = 1;
SELECT EventID, Name FROM Event WHERE EventID = 1;
GO

```

No issues found

Completion time: 2025-05-22T12:04:34.6270995+05:30

Query completed with errors.

Image 6: The *PreventEventDeletion* trigger and a test demonstrating the prevention of deletion for an event with booked tickets.

6.2 Functions

User defined scalar functions were created to encapsulate reusable calculations.

I. Get Total Revenue Function

The GetTotalRevenue function accepts an @EventID as input and returns the total revenue generated from Booked tickets for that event. It sums the Price of tickets where the EventID matches and the Status is 'Booked'. It returns 0 if no booked tickets are found.

The screenshot shows the SQL Server Management Studio (SSMS) interface. The Object Explorer on the left shows a connection to 'DESKTOP-3OJN058\SQLEXPRESS' with the database 'EventEaseDB' selected. The main pane displays a script for creating a function named 'GetTotalRevenue'. The script uses a cursor to sum the 'Price' column for rows where 'EventID' matches the input parameter '@EventID' and 'Status' is 'Booked'. If the result is null, it returns 0. The script concludes with a test query to select the total revenue for EventID 1, which returns a value of 200.00. The bottom status bar indicates the query was executed successfully.

```
USE EventEaseDB;
GO

-- Create Function
CREATE FUNCTION GetTotalRevenue (@EventID INT)
RETURNS DECIMAL(10,2)
AS
BEGIN
    DECLARE @TotalRevenue DECIMAL(10,2);
    SELECT @TotalRevenue = SUM(Price)
    FROM Ticket
    WHERE EventID = @EventID AND Status = 'Booked';
    RETURN ISNULL(@TotalRevenue, 0);
END;
GO

-- Test: Get revenue for EventID 1
SELECT dbo.GetTotalRevenue(1) AS TotalRevenue;
GO
```

Results

TotalRevenue
1 200.00

Query executed successfully.

Image 7: GetTotalRevenue Function Script and Test.

II. Get Attendee Count Function

The GetAttendeeCount function accepts an @EventID as input and returns the number of Booked tickets for that event. It counts the number of tickets where the EventID matches and the Status is 'Booked'.

The screenshot shows the SQL Server Management Studio (SSMS) interface. In the Object Explorer, the database 'EventEaseDB' is selected. In the center pane, a query window titled '07.Create_Fu...58(User (87))' contains the following T-SQL script:

```

1 USE EventEaseDB;
2 GO
3
4 -- Create Function
5 CREATE FUNCTION GetAttendeeCount (@EventID INT)
6 RETURNS INT
7 AS
8 BEGIN
9     DECLARE @Count INT;
10    SELECT @Count = COUNT(*)
11    FROM Ticket
12    WHERE EventID = @EventID AND Status = 'Booked';
13    RETURN @Count;
14 END;
15 GO
16
17 -- Test: Get attendee count for EventID 1
18 SELECT dbo.GetAttendeeCount(1) AS AttendeeCount;
19 GO

```

Below the script, the results pane shows a single row with 'AttendeeCount' having a value of 3.

Image 8: GetAttendeeCount Function Script and Test.

III. Get Average Rating Function

The GetAverageRating function accepts an @EventID as input and returns the average Rating from feedback received for that event. It calculates the average of the Rating column in the Feedback table for the specified EventID. It handles cases where no feedback exists by returning 0.

The screenshot shows the SQL Server Management Studio (SSMS) interface. In the Object Explorer, the database 'EventEaseDB' is selected. In the center pane, a query window titled '08.Create_Fu...58(User (91))' contains the following T-SQL script:

```

1 USE EventEaseDB;
2 GO
3
4 -- Create Function
5 CREATE FUNCTION GetAverageRating (@EventID INT)
6 RETURNS DECIMAL(3,1)
7 AS
8 BEGIN
9     DECLARE @AvgRating DECIMAL(3,1);
10    SELECT @AvgRating = AVG(CAST(Rating AS DECIMAL(3,1)))
11    FROM Feedback
12    WHERE EventID = @EventID;
13    RETURN ISNULL(@AvgRating, 0);
14 END;
15 GO
16
17 -- Test: Get average rating for EventID 1
18 SELECT dbo.GetAverageRating(1) AS AverageRating;
19 GO

```

Below the script, the results pane shows a single row with 'AverageRating' having a value of 4.8.

Image 9: GetAverageRating Function Query and Test

6.3 Views

Views were created to simplify complex join operations and provide convenient access to aggregated or combined data.

I. Event Details View

The EventDetails view joins the Event, Venue, and Organizer tables. It provides a consolidated view of event information, including the event name, type, start/end dates, and the names and email addresses of the venue and organizer.

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, the database 'EventEaseDB' is selected. In the center pane, a query window titled '09_Create_Vi...58\User (64)' contains the following T-SQL code:

```
1 USE EventEaseDB;
2 GO
3
4 -- Create View
5 CREATE VIEW EventDetails AS
6 SELECT
7     e.EventID,
8     e.Name AS EventName,
9     e.Type,
10    e.StartDate,
11    e.EndDate,
12    v.Name AS VenueName,
13    o.Name AS OrganizerName,
14    o.Email AS OrganizerEmail
15 FROM Event e
16 JOIN Venue v ON e.VenueID = v.VenueID
17 JOIN Organizer o ON e.OrganizerID = o.OrganizerID;
18 GO
19
20 -- Test: Query view for first 3 events
21 SELECT EventID, EventName, Type, VenueName, OrganizerName
22 FROM EventDetails
23 WHERE EventID <= 3;
24 GO
```

The results pane shows the output of the test query:

EventID	EventName	Type	VenueName	OrganizerName
1	Tech Conference 2025	Conference	Colombo Convention Center	EventMasters
2	Rock Concert	Concert	Galle Face Hotel	CelebrateNow
3	Book Launch	Future	BMICH	JoyEvents

The status bar at the bottom indicates 'Query executed successfully.'

Image 10: EventDetails View Query and Test.

II. Attendee Tickets View

The AttendeeTickets view joins the Ticket, Event, and Attendee tables. It provides a view showing which attendees have booked which tickets for which events, including the ticket details (ID, Type, Price, Status) and attendee name.

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, the database 'EventEaseDB' is selected. In the center pane, a query window titled '10_Create_Vi...58\User (98)' contains the following T-SQL code:

```
1 USE EventEaseDB;
2 GO
3
4 -- Create View
5 CREATE VIEW AttendeeTickets AS
6 SELECT
7     e.Name AS EventName,
8     a.FullName,
9     t.TicketID,
10    t.TicketType,
11    t.Price,
12    t.Status
13 FROM Ticket t
14 JOIN Event e ON t.EventID = e.EventID
15 JOIN Attendee a ON t.AttendeeID = a.AttendeeID;
16 GO
17
18 -- Test: Query view for Tech Conference 2025
19 SELECT EventName, FullName, TicketID, TicketType, Price
20 FROM AttendeeTickets
21 WHERE EventName = 'Tech Conference 2025';
22 GO
```

The results pane shows the output of the test query:

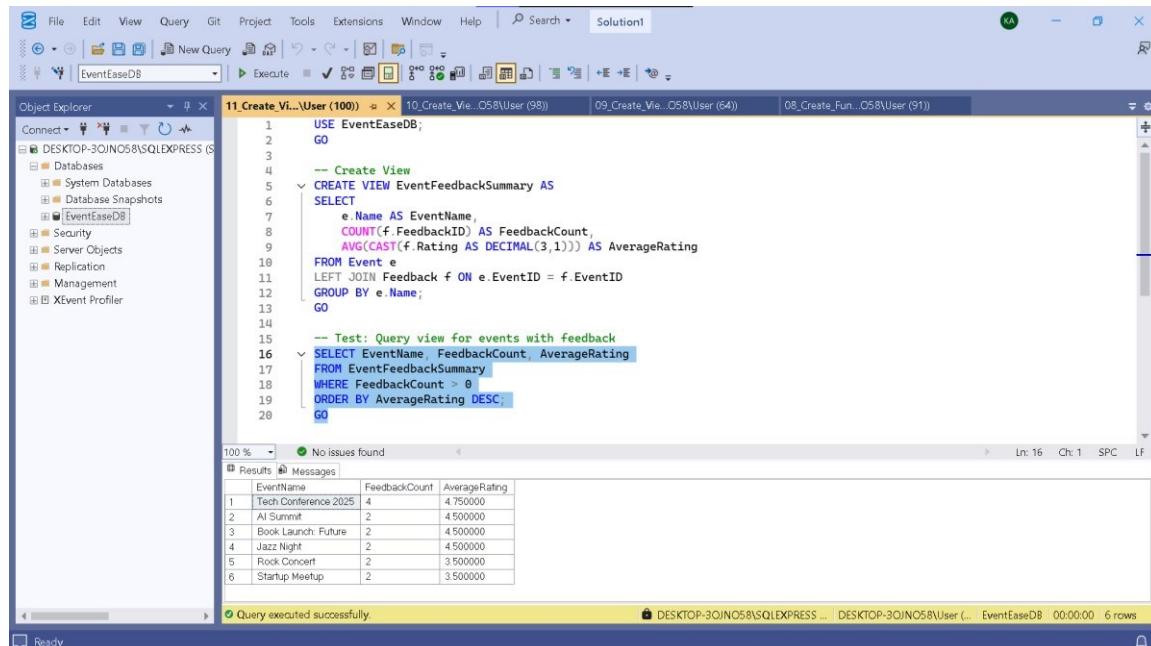
EventName	FullName	TicketID	TicketType	Price
Tech Conference 2025	John Doe	1	Regular	50.00
Tech Conference 2025	Jane Smith	2	VIP	100.00
Tech Conference 2025	John Doe	13	Regular	50.00

The status bar at the bottom indicates 'Query executed successfully.'

Image 11: AttendeeTickets View Query and Test

III. Event Feedback Summary View

The EventFeedbackSummary view joins the Event and Feedback tables. It summarizes feedback per event, showing the event name, the count of feedback entries received (FeedbackCount), and the average rating (AverageRating). It uses a LEFT JOIN to include events that may not have received feedback yet.



The screenshot shows the SSMS interface with the Object Explorer on the left and a query editor window on the right. The query editor contains the following SQL code:

```

USE EventEaseDB;
GO

-- Create View
CREATE VIEW EventFeedbackSummary AS
SELECT
    e.Name AS EventName,
    COUNT(f.FeedbackID) AS FeedbackCount,
    AVG(CAST(f.Rating AS DECIMAL(3,1))) AS AverageRating
FROM Event e
LEFT JOIN Feedback f ON e.EventID = f.EventID
GROUP BY e.Name;
GO

-- Test: Query view for events with feedback
SELECT EventName, FeedbackCount, AverageRating
FROM EventFeedbackSummary
WHERE FeedbackCount > 0
ORDER BY AverageRating DESC;
GO

```

The results pane shows a table with the following data:

	EventName	FeedbackCount	AverageRating
1	Tech Conference 2025	4	4.750000
2	AI Summit	2	4.500000
3	Book Launch Future	2	4.500000
4	Jazz Night	2	4.500000
5	Rock Concert	2	3.500000
6	Startup Meetup	2	3.500000

At the bottom of the results pane, it says "Query executed successfully."

Image 13: EventFeedbackSummary View Query and Test

6.4 Procedures

Stored procedures were created to encapsulate business logic and execute a series of SQL statements as a single unit.

I. Procedure for Inserting Events

The InsertEvent procedure handles the insertion of new events. It takes all event details as parameters. Before inserting, it validates the provided VenueID and OrganizerID by checking if they exist in their respective tables. It also checks if the EndDate is not before the StartDate. If any validation fails, it raises an error and returns, preventing the insert. Otherwise, it inserts the new event record.

The screenshot shows two instances of SQL Server Management Studio (SSMS) running side-by-side.

Top Window (Query 1):

```

1 USE EventEaseDB;
2 GO
3
4 -- Create Procedure
5 CREATE PROCEDURE InsertEvent
6     @Name VARCHAR(100),
7     @Description VARCHAR(500),
8     @StartDate DATE,
9     @EndDate DATE,
10    @StartTime TIME,
11    @EndTime TIME,
12    @VenueID INT,
13    @OrganizerID INT
14
15 AS
16 BEGIN
17     IF NOT EXISTS (SELECT 1 FROM Venue WHERE VenueID = @VenueID)
18     BEGIN
19         RAISERROR ('Invalid VenueID.', 16, 1);
20         RETURN;
21     END
22     IF NOT EXISTS (SELECT 1 FROM Organizer WHERE OrganizerID = @OrganizerID)
23     BEGIN
24         RAISERROR ('Invalid OrganizerID.', 16, 1);
25         RETURN;
26     END
27     IF @EndDate < @StartDate
28     BEGIN
29         RAISERROR ('EndDate must be greater than or equal to StartDate.', 16, 1);
30         RETURN;
31     END
32     INSERT INTO Event (Name, Description, Type, StartDate, EndDate, StartTime, EndTime, VenueID, OrganizerID)
33     VALUES (@Name, @Description, @Type, @StartDate, @EndDate, @StartTime, @EndTime, @VenueID, @OrganizerID);
34 END;
35 GO
36
37 -- Test: Insert a new event
38 EXEC InsertEvent
39     @Name = 'Data Science Workshop',
40     @Description = 'Hands-on data science training',
41     @Type = 'Conference',
42     @StartDate = '2026-05-01',
43     @EndDate = '2026-05-02',
44     @StartTime = '09:00',
45     @EndTime = '17:00',
46     @VenueID = 1,
47     @OrganizerID = 1;
48
49 SELECT EventID, Name, StartDate, EndDate, VenueID, OrganizerID
50 FROM Event
51 WHERE Name = 'Data Science Workshop';
52 GO

```

Bottom Window (Query 2):

```

100 % No issues found
Results Messages
EventID Name StartDate EndDate VenueID OrganizerID
1 13 Data Science Workshop 2026-05-01 2026-05-02 1 1

```

Both windows show the message "Query executed successfully." at the bottom.

Image 14: InsertEvent Procedure Query and Test II.

Procedure for Ticket Sales Report

The TicketSalesReport procedure generates a summary report of ticket sales for Booked tickets. It joins the Event and Ticket tables, groups by event name, and calculates the total TicketsSold (count of booked tickets) and TotalRevenue (sum of prices of booked tickets). The results are ordered by total revenue descending.

```

25      RETURN;
26  END
27  IF @EndDate < @StartDate
28  BEGIN
29      RAISERROR ('End Date must be greater than or equal to Start Date.', 16, 1);
30      RETURN;
31  END
32  INSERT INTO Event (Name, Description, Type, StartDate, EndDate, StartTime, EndTime, VenueID, OrganizerID)
33  VALUES (@Name, @Description, @Type, @StartDate, @EndDate, @StartTime, @EndTime, @VenueID, @OrganizerID);
34  END;
35  GO
36
37  -- Test: Insert a new event
38  EXEC InsertEvent
39  @Name = 'Data Science Workshop',
40  @Description = 'Hands-on data science training',
41  @Type = 'Conference',
42  @StartDate = '2026-05-01',
43  @EndDate = '2026-05-02',
44  @StartTime = '09:00',
45  @EndTime = '17:00',
46  @VenueID = 1,
47  @OrganizerID = 1;
48  SELECT EventID, Name, StartDate, EndDate, VenueID, OrganizerID
49  FROM Event
50  WHERE Name = 'Data Science Workshop';
51  GO

```

No issues found

EventID	Name	StartDate	EndDate	VenueID	OrganizerID
1	Data Science Workshop	2026-05-01	2026-05-02	1	1

Query executed successfully.

Image 15: TicketSalesReport Procedure Query and Test

7. Section 4 - Web Application & Business Intelligence

7.1 Application Overview

A sample web application was developed using the Flask framework in Python to demonstrate interaction with the EventEase database. This application serves as a basic user interface for browsing available events.

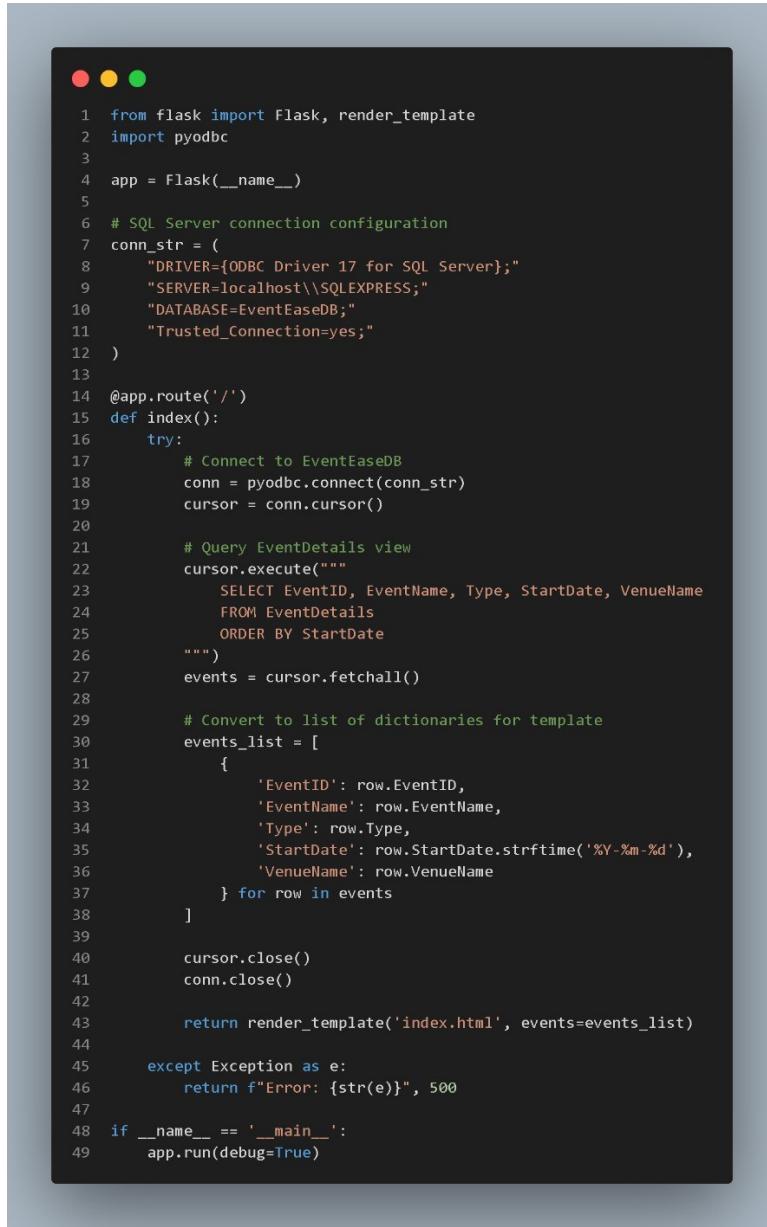
Functionality

The application demonstrates the following functionality,

- **Event Listing** - Displays a list of events available in the database.
- **Event Filtering** - Allows users to filter the list of events based on event type (e.g., Concert, Conference).

The application connects to the SQL Server database to fetch event data and present it to the user through web pages.

7.2 Application Screenshots



```
 1  from flask import Flask, render_template
 2  import pyodbc
 3
 4  app = Flask(__name__)
 5
 6  # SQL Server connection configuration
 7  conn_str = (
 8      "DRIVER={ODBC Driver 17 for SQL Server};"
 9      "SERVER=localhost\\SQLEXPRESS;"
10     "DATABASE=EventEaseDB;"
11     "Trusted_Connection=yes;"
12 )
13
14 @app.route('/')
15 def index():
16     try:
17         # Connect to EventEaseDB
18         conn = pyodbc.connect(conn_str)
19         cursor = conn.cursor()
20
21         # Query EventDetails view
22         cursor.execute("""
23             SELECT EventID, EventName, Type, StartDate, VenueName
24             FROM EventDetails
25             ORDER BY StartDate
26         """)
27         events = cursor.fetchall()
28
29         # Convert to list of dictionaries for template
30         events_list = [
31             {
32                 'EventID': row.EventID,
33                 'EventName': row.EventName,
34                 'Type': row.Type,
35                 'StartDate': row.StartDate.strftime('%Y-%m-%d'),
36                 'VenueName': row.VenueName
37             } for row in events
38         ]
39
40         cursor.close()
41         conn.close()
42
43         return render_template('index.html', events=events_list)
44
45     except Exception as e:
46         return f"Error: {str(e)}", 500
47
48 if __name__ == '__main__':
49     app.run(debug=True)
```

Image 16: code snippet – app.py

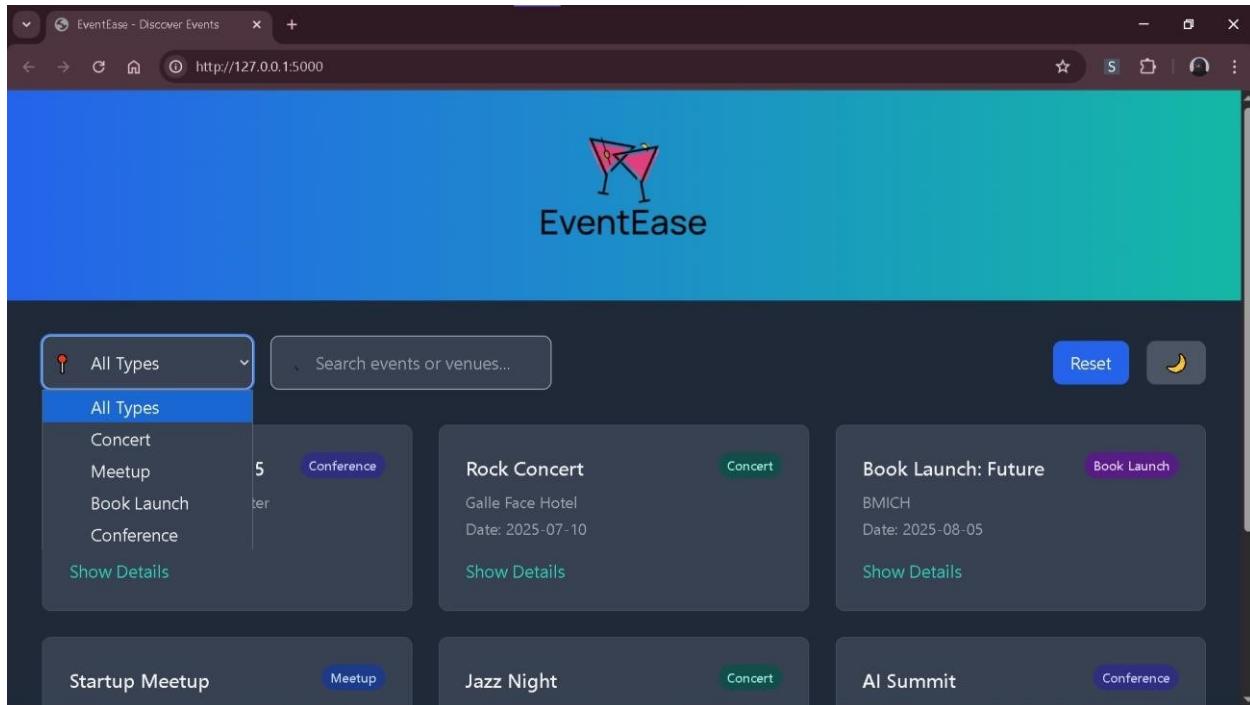


Image 17: Web Application Screenshot - All Events Listing

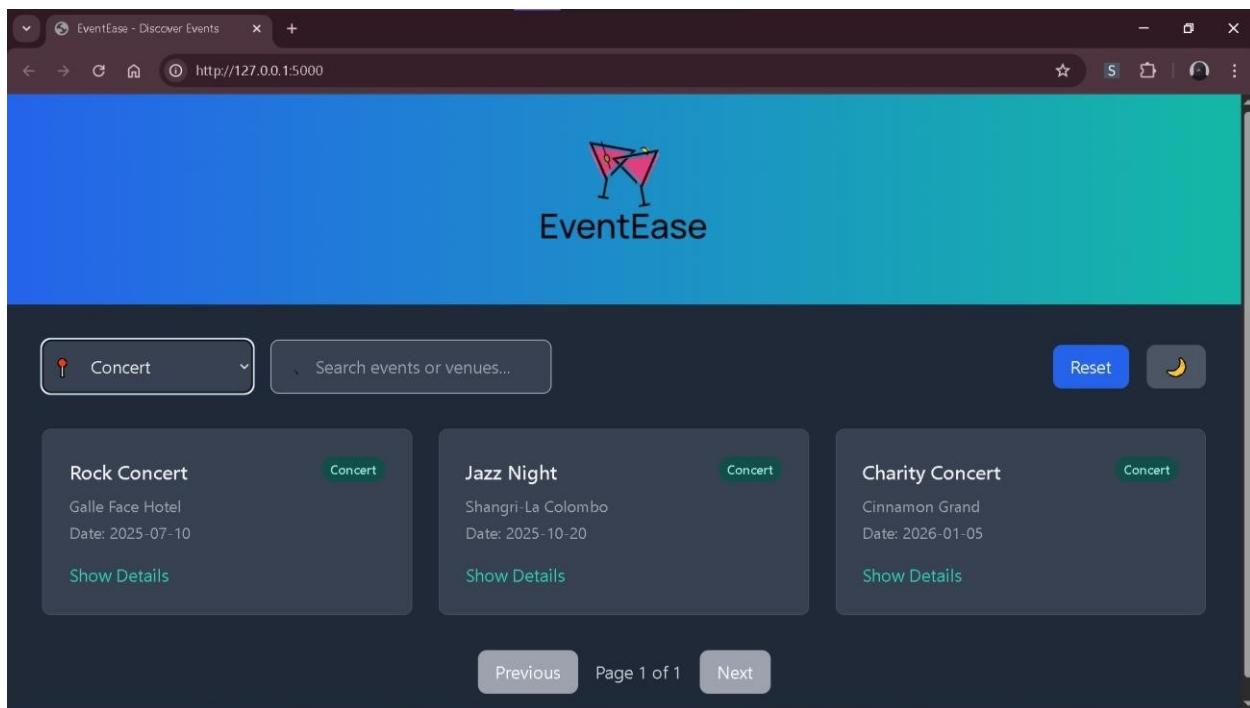


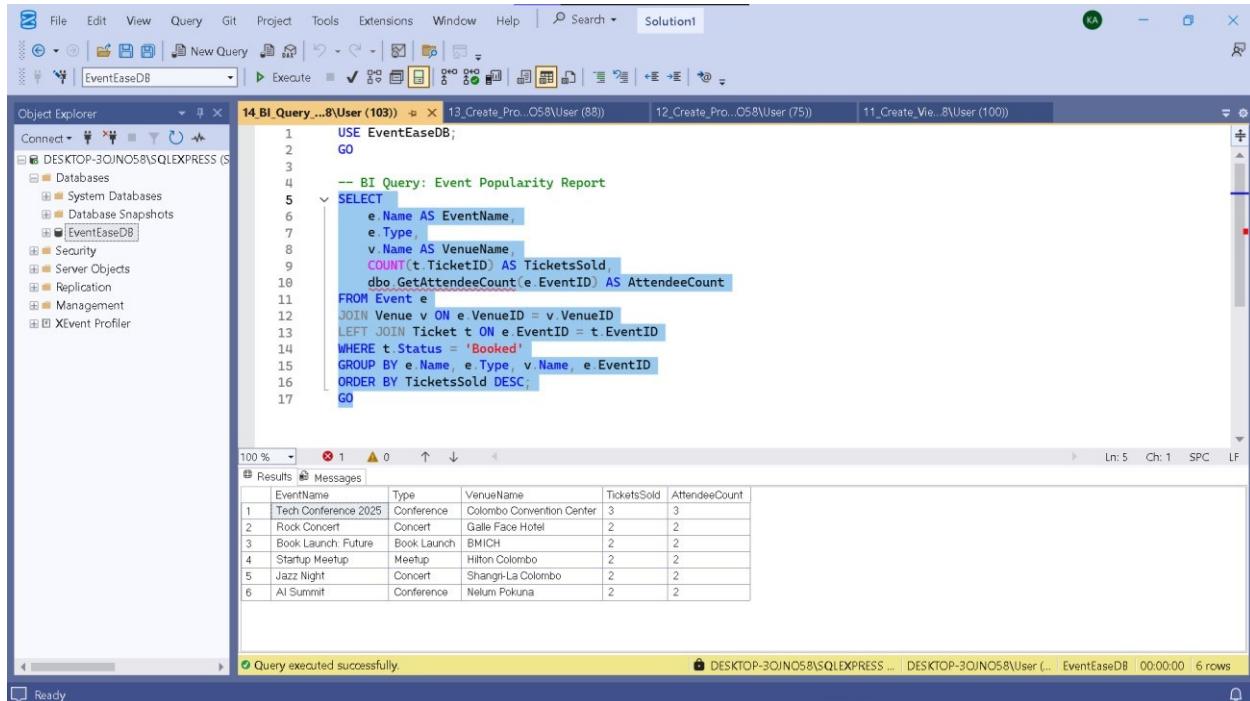
Image 18: Web Application Screenshot - Events Filtered by Type

7.3 Business Intelligence (BI)

The database design and implemented objects support various business intelligence queries to gain insights into event performance and attendee behavior. Specific BI queries were executed to demonstrate this capability.

I. Event Popularity Report

This query reports on event popularity by combining event, venue, and ticket information. It shows the event name, type, venue, the total count of booked tickets (TicketsSold), and the total count of unique attendees for booked tickets (AttendeeCount), utilizing the GetAttendeeCount function. The results are ordered by the number of tickets sold in descending order.



The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the EventEaseDB database. The central pane displays a T-SQL query titled 'Event Popularity Report'. The query uses joins between Event, Venue, and Ticket tables to calculate the total number of tickets sold and unique attendees for each event. The results are ordered by TicketsSold in descending order. The bottom pane shows the execution results, which are as follows:

	EventName	Type	VenueName	TicketsSold	AttendeeCount
1	Tech Conference 2025	Conference	Colombo Convention Center	3	3
2	Rock Concert	Concert	Galle Face Hotel	2	2
3	Book Launch: Future	Book Launch	BMICH	2	2
4	Startup Meetup	Meetup	Hilton Colombo	2	2
5	Jazz Night	Concert	Shangi-La Colombo	2	2
6	AI Summit	Conference	Nelum Pokuna	2	2

A message at the bottom indicates the query was executed successfully.

Image 19: Event Popularity BI Query and Result

II. Ticket Sales Summary

This report directly uses the TicketSalesReport stored procedure to provide a summary of sales and revenue for events with booked tickets. It lists each event name along with the number of tickets sold and the total revenue generated from those booked tickets, ordered by total revenue descending.

The screenshot shows a SQL Server Management Studio (SSMS) window. The Object Explorer on the left shows the database structure for 'EventEaseDB'. The central pane displays a query titled '15_BI_Query...58(User (56))' with the following code:

```

1 USE EventEaseDB;
2 GO
3
4 -- BI Query: Ticket Sales Summary
5 EXEC TicketSalesReport;
6 GO

```

The results pane shows a table with the following data:

	EventName	TicketsSold	TotalRevenue
1	Jazz Night	2	230.00
2	Tech Conference 2025	3	200.00
3	AI Summit	2	180.00
4	Rock Concert	2	150.00
5	Startup Meetup	2	60.00
6	Book Launch: Future	2	40.00

The status bar at the bottom indicates 'Query executed successfully'.

Image 20: Ticket Sales Summary BI Query and Result

III. Attendee Satisfaction Analysis

This query provides insights into attendee satisfaction per event. It joins the Event, Feedback, and Ticket tables to display the event name, the average rating received (AverageRating) using the GetAverageRating function, the total count of feedback entries (FeedbackCount), and the total count of unique attendees for booked tickets (Attendees) using the GetAttendeeCount function. The results are ordered by average rating descending.

The screenshot shows a SQL Server Management Studio (SSMS) window. The Object Explorer on the left shows the database structure for 'EventEaseDB'. The central pane displays a query titled '16_BI_Query...58(User (85))' with the following code:

```

1 USE EventEaseDB;
2 GO
3
4 -- BI Query: Attendee Satisfaction Analysis
5
6 SELECT
7     e.Name AS EventName,
8     dbo.GetAverageRating(e.EventID) AS AverageRating,
9     COUNT(f.FeedbackID) AS FeedbackCount,
10    COUNT(DISTINCT t.AttendeeID) AS Attendees
11 FROM Event e
12 LEFT JOIN Feedback f ON e.EventID = f.EventID
13 LEFT JOIN Ticket t ON e.EventID = t.EventID
14 WHERE t.Status = 'Booked'
15 GROUP BY e.Name, e.EventID
16 ORDER BY AverageRating DESC;
17 GO

```

The results pane shows a table with the following data:

	EventName	AverageRating	FeedbackCount	Attendees
1	Tech Conference 2025	4.8	12	2
2	Book Launch: Future	4.5	4	2
3	Jazz Night	4.5	4	2
4	AI Summit	4.5	4	2
5	Startup Meetup	3.5	4	2
6	Rock Concert	3.5	4	2

The status bar at the bottom indicates 'Query executed successfully'.

Image 21: Attendee Satisfaction Analysis BI Query and Result

8. Conclusion

The EventEase Advanced Database Management System successfully implements a relational database solution in Microsoft SQL Server to manage event data. The design, based on ER modeling and normalization, ensures data integrity and a structured foundation. The implementation incorporates advanced SQL features, including triggers for automated data management (venue capacity update, feedback auditing, prevention of deletion with dependencies), user-defined functions for reusable calculations (revenue, attendee count, average rating), views for simplified data access and reporting (event details, attendee tickets, feedback summary), and stored procedures for complex logic encapsulation (validated event insertion, ticket sales reporting).

A sample Flask web application demonstrates the usability of the database, providing a basic interface for browsing and filtering events. Furthermore, the system effectively supports business intelligence requirements through specific queries that provide valuable insights into event popularity, sales performance, and attendee satisfaction. The project successfully addresses the key requirements outlined for an efficient event management platform, providing a solid database backend for future development.

9. References

- I. Microsoft Docs. (2025). - <https://learn.microsoft.com/en-us/sql/sql-server>
- II. IBM Knowledge Center. (2025). *Database normalization: 1NF, 2NF, 3NF*. - <https://www.ibm.com/docs/en/db2-for-zos/12.0.0?topic=modeling-normalization-in-database-design>
- III. Draw.io. (2025). *Diagram Design Tool*. - <https://www.drawio.com/>
- IV. Microsoft SQL Server data sources with Kerberos and in different Oss. - <https://www.jetbrains.com/help/webstorm/db-tutorial-connecting-to-ms-sql-server.html#connect-by-using-windows-domain-authentication>

10. Appendix

10.1 Code Snippets

- Complete SQL Queries, Test application source code and backup of database (.bak) are available in supplementary repository: https://nsbm365-my.sharepoint.com/:f/g/personal/ykarathnasiri_students_nsbm_ac_lk/EgARQf54qi9EmHnYVCPc_gUBfZrACZiY_H9i8b2oOrNq_g?e=ggr5lI