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

I O N GROUPS PVT LTD

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,

I O N 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 **I O N 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.

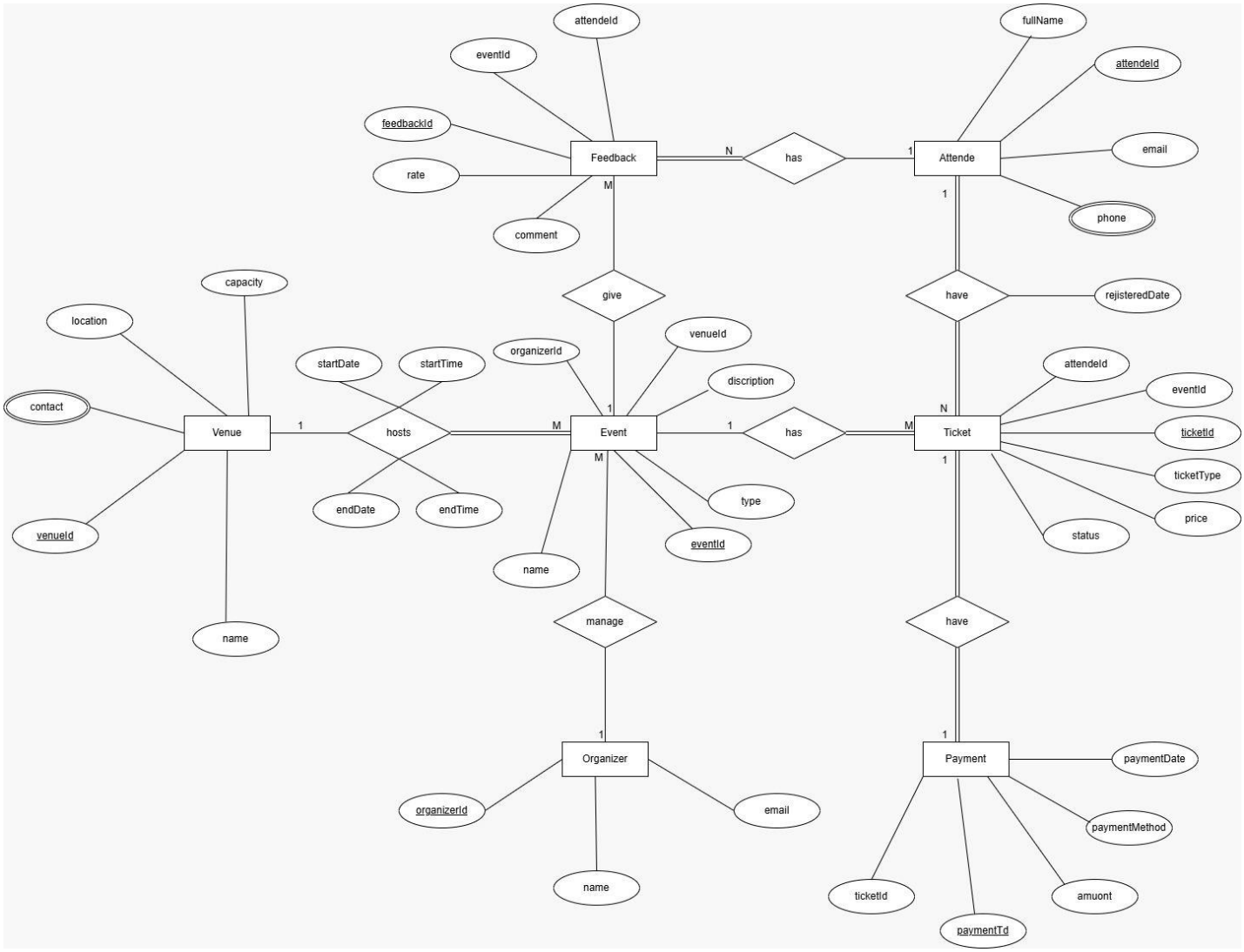


Image 1: Entity-Relationship Diagram for the EventEase Database System

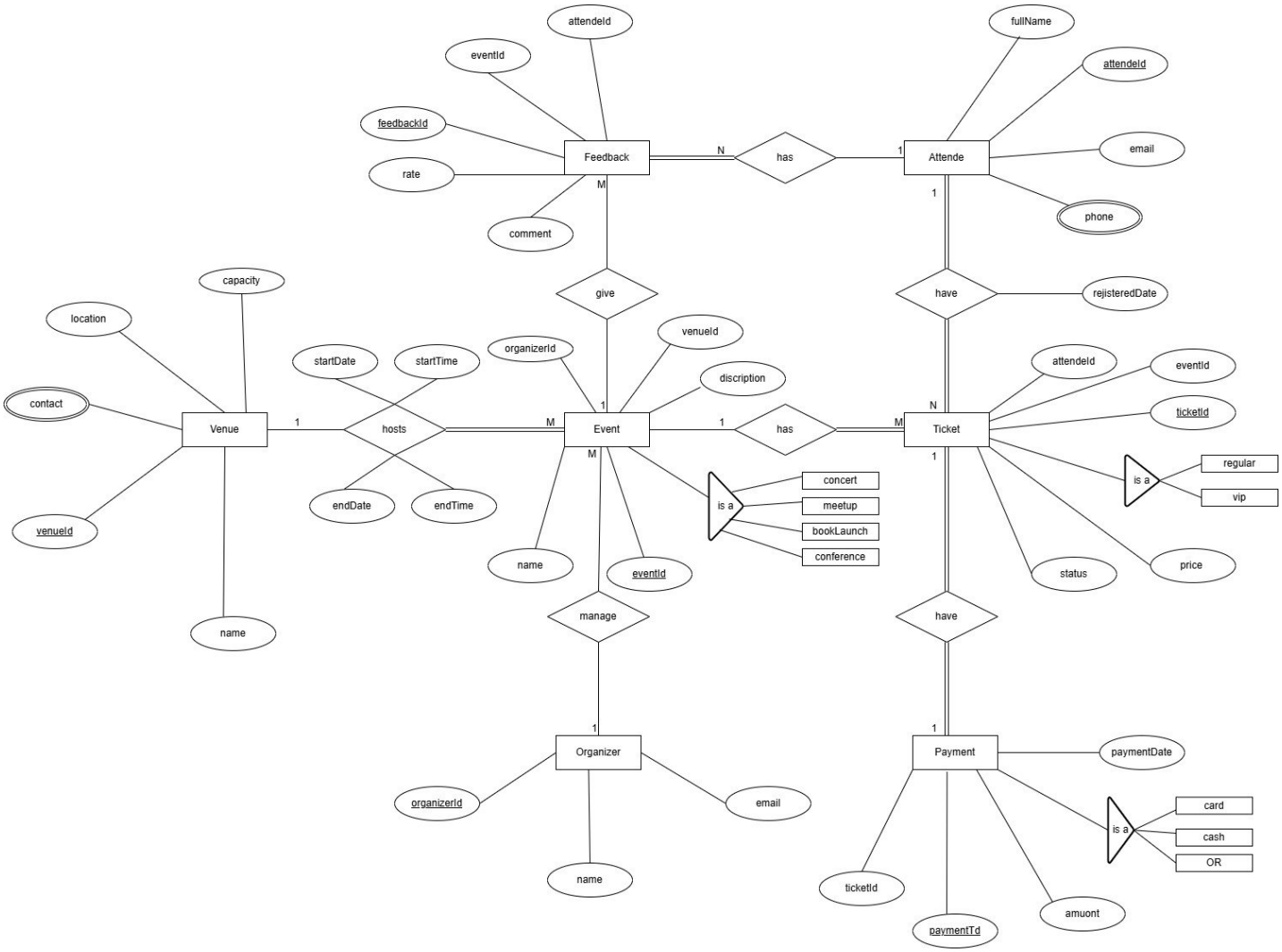


Image 2: Enhanced entity-relationship (EER) Diagram for the EventEase Database System

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. **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.

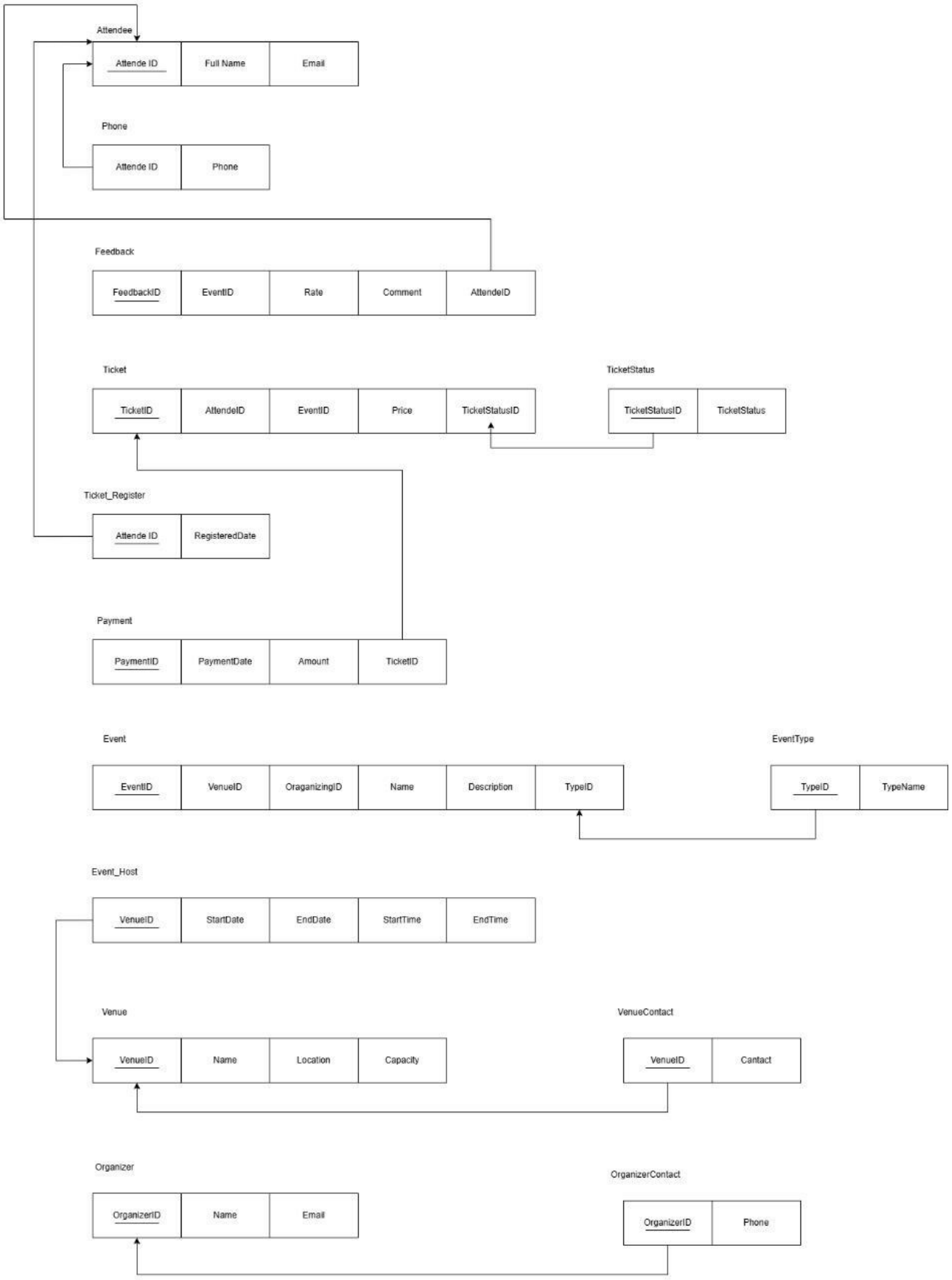

Attendee
Attende ID
Full Name
Email
Phone
Attende ID
Phone
Feedback
FeedbackID
EventID
Rate
Comment
AttendeID
Ticket
TicketID
AttendeID
EventID
Price
TicketStatusID
TicketStatus
TicketStatusID
TicketStatus
Ticket_Register
Attende ID
RegisteredDate
Payment
PaymentID
PaymentDate
Amount
TicketID
Event
EventID
VenueID
OraganizingID
Name
Description
TypeID
EventType
TypeID
TypeName
Event_Host
VenueID
StartDate
EndDate
StartTime
EndTime
Venue
VenueID
Name
Location
Capacity
VenueContact
VenueID
Cantact
Organizer
OrganizerID
Name
Email
OrganizerContact
OrganizerID
Phone

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

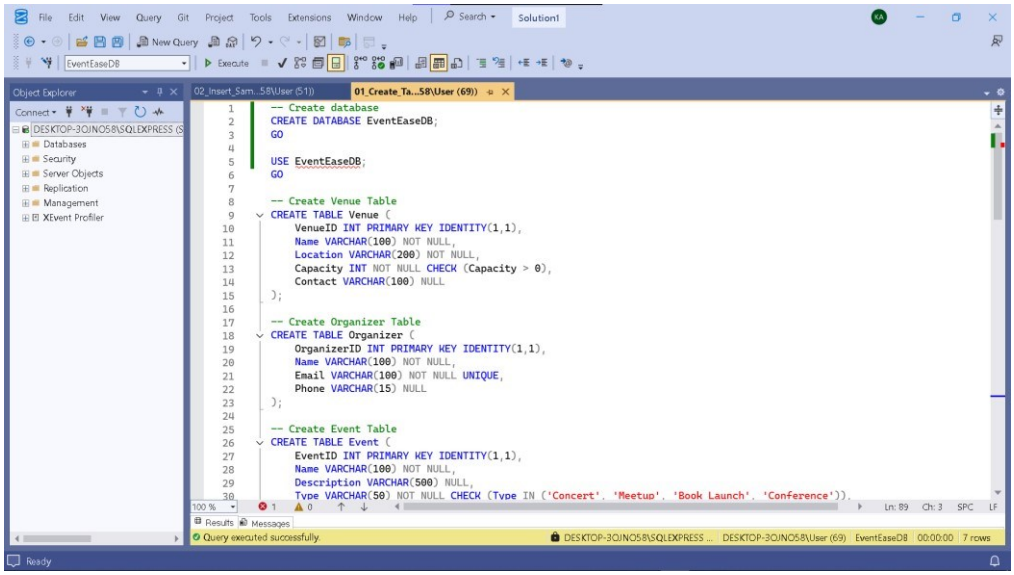
- Ticket status restricted to: Booked, Paid, Cancelled **Unique Constraints**
- Attendee email addresses must be unique
- Organizer email addresses must be unique **Foreign Key Constraints**
- All foreign key relationships enforce referential integrity
- Cascading rules ensure data consistency across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

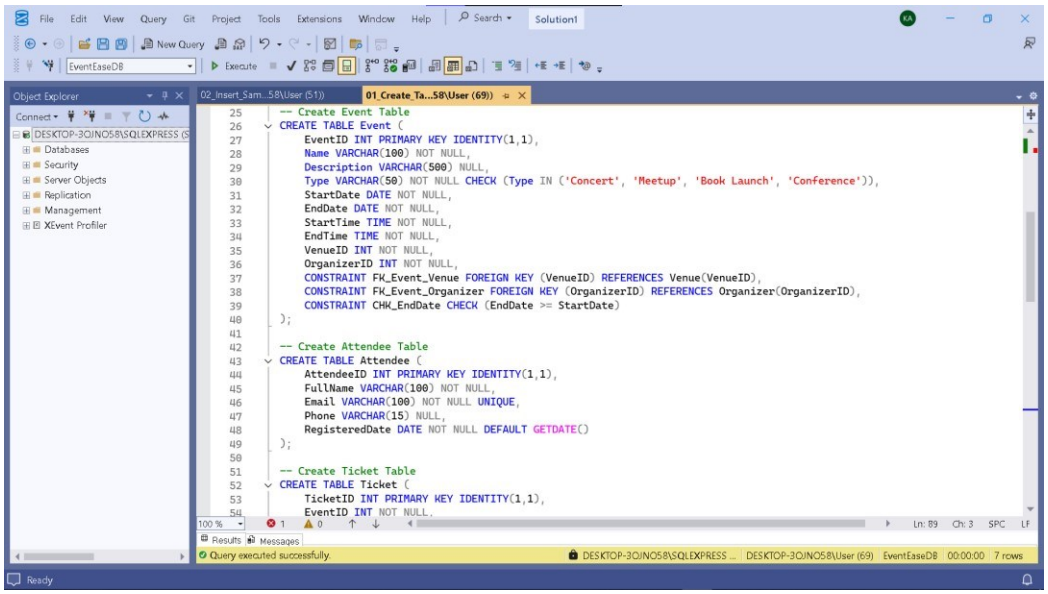
5. Section 2 - Database Implementation

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

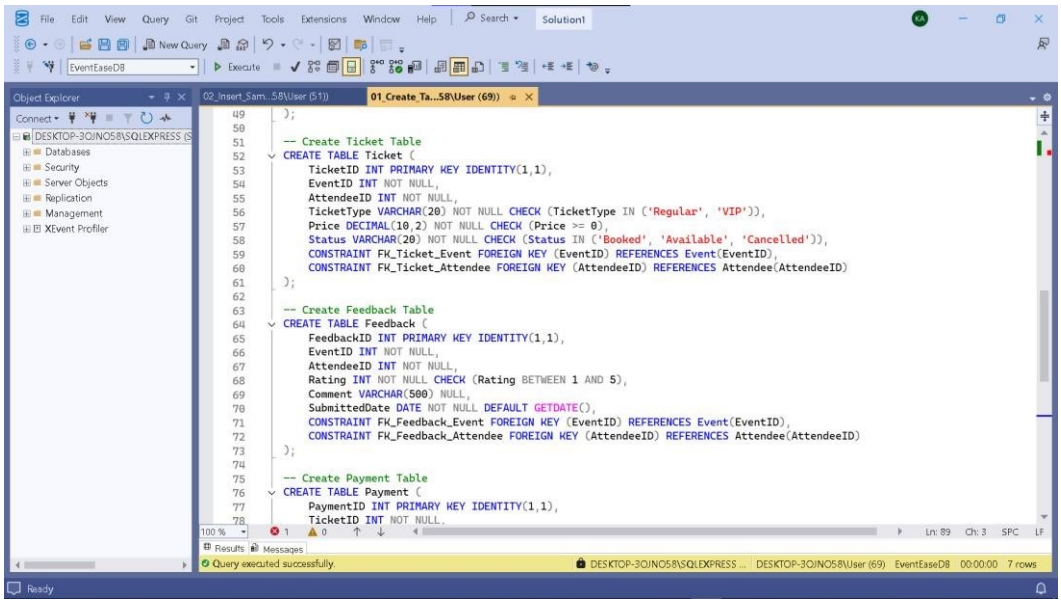
5.1 Database and Table Creation

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.

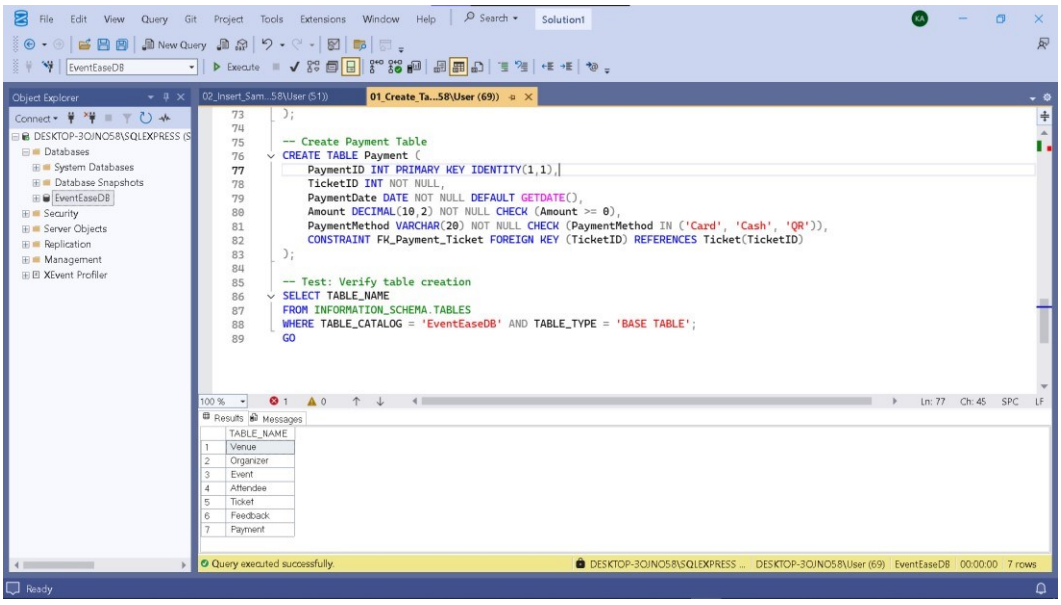




```
-- Create Event Table
CREATE TABLE Event (
    EventID INT PRIMARY KEY IDENTITY(1,1),
    Name VARCHAR(100) NOT NULL,
    Description VARCHAR(500) NULL,
    Type VARCHAR(50) NOT NULL CHECK (Type IN ('Concert', 'Meetup', 'Book Launch', 'Conference')),
    StartDate DATE NOT NULL,
    EndDate DATE NOT NULL,
    StartTime TIME NOT NULL,
    EndTime TIME NOT NULL,
    VenueID INT NOT NULL,
    OrganizerID INT NOT NULL,
    CONSTRAINT FK_Event_Venue FOREIGN KEY (VenueID) REFERENCES Venue(VenueID),
    CONSTRAINT FK_Event_Organizer FOREIGN KEY (OrganizerID) REFERENCES Organizer(OrganizerID),
    CONSTRAINT CHK_EndDate CHECK (EndDate >= StartDate)
);

-- Create Attendee Table
CREATE TABLE Attendee (
    AttendeeID INT PRIMARY KEY IDENTITY(1,1),
    FullName VARCHAR(100) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    Phone VARCHAR(15) NULL,
    RegisteredDate DATE NOT NULL DEFAULT GETDATE()
);

-- Create Ticket Table
CREATE TABLE Ticket (
    TicketID INT PRIMARY KEY IDENTITY(1,1),
    EventID INT NOT NULL,
```



```
);

-- Create Ticket Table
CREATE TABLE Ticket (
    TicketID INT PRIMARY KEY IDENTITY(1,1),
    EventID INT NOT NULL,
    AttendeeID INT NOT NULL,
    TicketType VARCHAR(20) NOT NULL CHECK (TicketType IN ('Regular', 'VIP')),
    Price DECIMAL(10,2) NOT NULL CHECK (Price >= 0),
    Status VARCHAR(20) NOT NULL CHECK (Status IN ('Booked', 'Available', 'Cancelled')),
    CONSTRAINT FK_Ticket_Event FOREIGN KEY (EventID) REFERENCES Event(EventID),
    CONSTRAINT FK_Ticket_Attendee FOREIGN KEY (AttendeeID) REFERENCES Attendee(AttendeeID)
);

-- Create Feedback Table
CREATE TABLE Feedback (
    FeedbackID INT PRIMARY KEY IDENTITY(1,1),
    EventID INT NOT NULL,
    AttendeeID INT NOT NULL,
    Rating INT NOT NULL CHECK (Rating BETWEEN 1 AND 5),
    Comment VARCHAR(500) NULL,
    SubmittedDate DATE NOT NULL DEFAULT GETDATE(),
    CONSTRAINT FK_Feedback_Event FOREIGN KEY (EventID) REFERENCES Event(EventID),
    CONSTRAINT FK_Feedback_Attendee FOREIGN KEY (AttendeeID) REFERENCES Attendee(AttendeeID)
);

-- Create Payment Table
CREATE TABLE Payment (
    PaymentID INT PRIMARY KEY IDENTITY(1,1),
    TicketID INT NOT NULL,
```



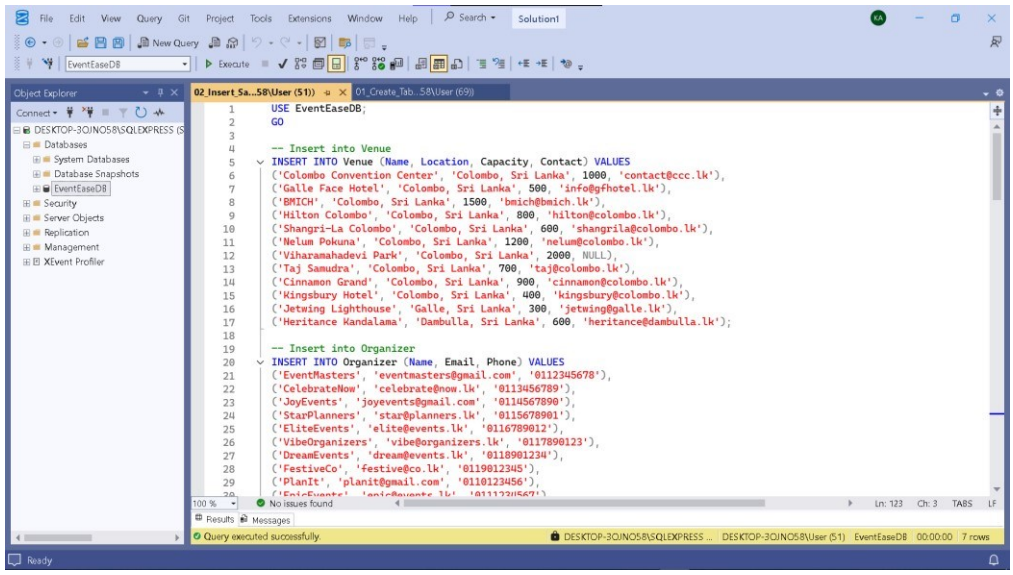
```
);

-- Create Payment Table
CREATE TABLE Payment (
    PaymentID INT PRIMARY KEY IDENTITY(1,1),
    TicketID INT NOT NULL,
    PaymentDate DATE NOT NULL DEFAULT GETDATE(),
    Amount DECIMAL(10,2) NOT NULL CHECK (Amount >= 0),
    PaymentMethod VARCHAR(20) NOT NULL CHECK (PaymentMethod IN ('Card', 'Cash', 'QR')),
    CONSTRAINT FK_Payment_Ticket FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)
);

-- Test: Verify table creation
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_CATALOG = 'EventEaseDB' AND TABLE_TYPE = 'BASE TABLE';
GO
```

	TABLE_NAME
1	Venue
2	Organizer
3	Event
4	Attendee
5	Ticket
6	Feedback
7	Payment

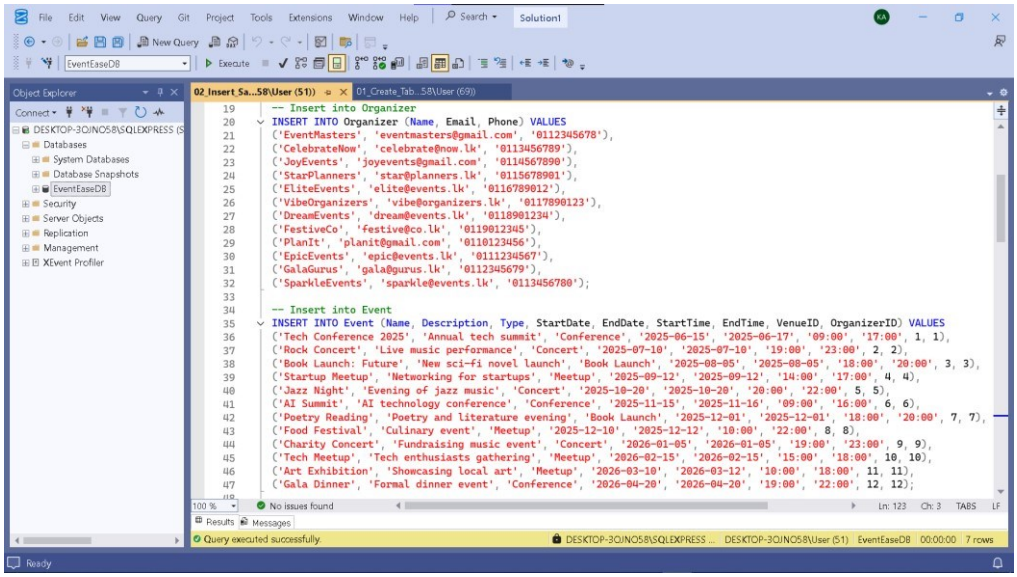
Image 3: Quries for creating the EventEase database and its core tables (Venue, Organizer, Event, Attendee, Ticket, Feedback, Payment).

5.2 Sample Data Insertion

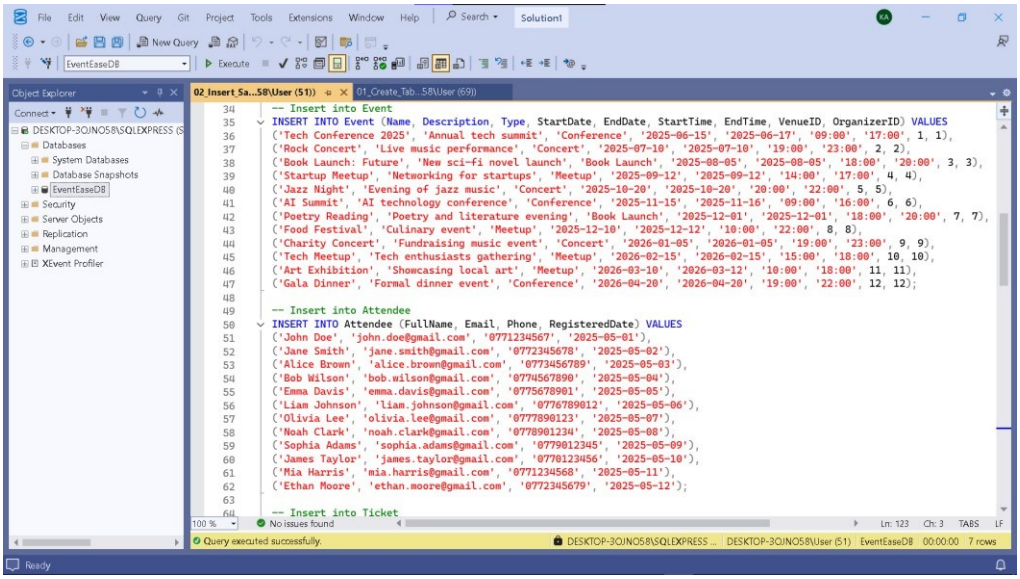
Sample data was inserted into the tables to populate the database for testing and demonstration purposes. This includes data for venues, organizers, events, attendees, tickets, feedback, and payments.



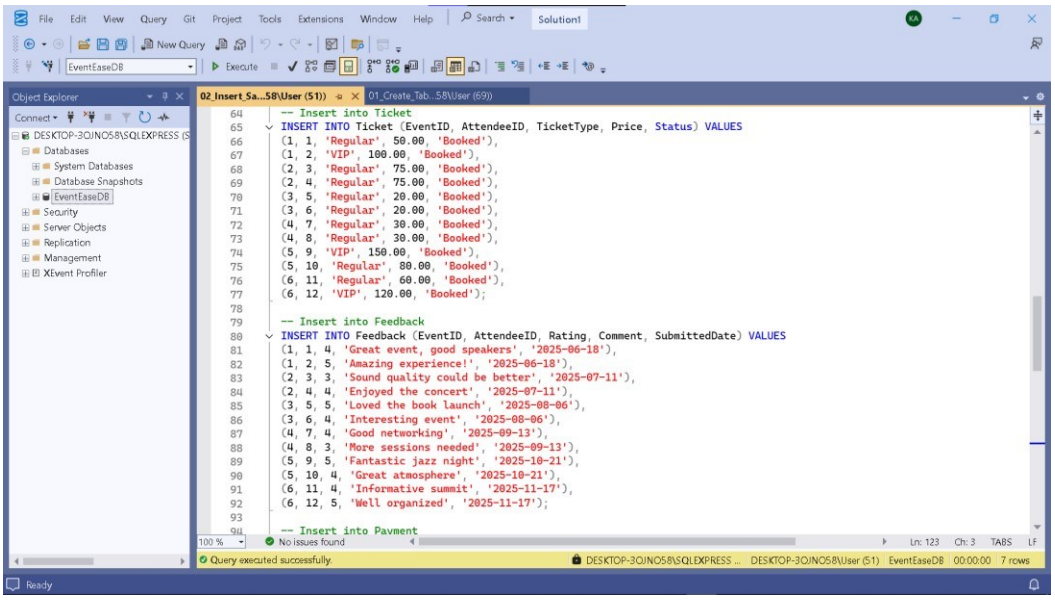
```sql
USE EventEaseDB;
GO

-- 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@gfhotel.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, 'shangrila@colombo.lk'),
('Nelum Pokuna', 'Colombo, Sri Lanka', 1200, 'nelum@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, 'jetwing@galle.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', '0114567890'),
('StarPlanners', 'star@planners.lk', '0115678901'),
('EliteEvents', 'elite@events.lk', '0116789012'),
('VibeOrganizers', 'vibe@organizers.lk', '0117890123'),
('DreamEvents', 'dream@events.lk', '0118901234'),
('FestiveCo', 'festive@co.lk', '01190123456'),
('PlanIt', 'planit@gmail.com', '0110123456'),
```



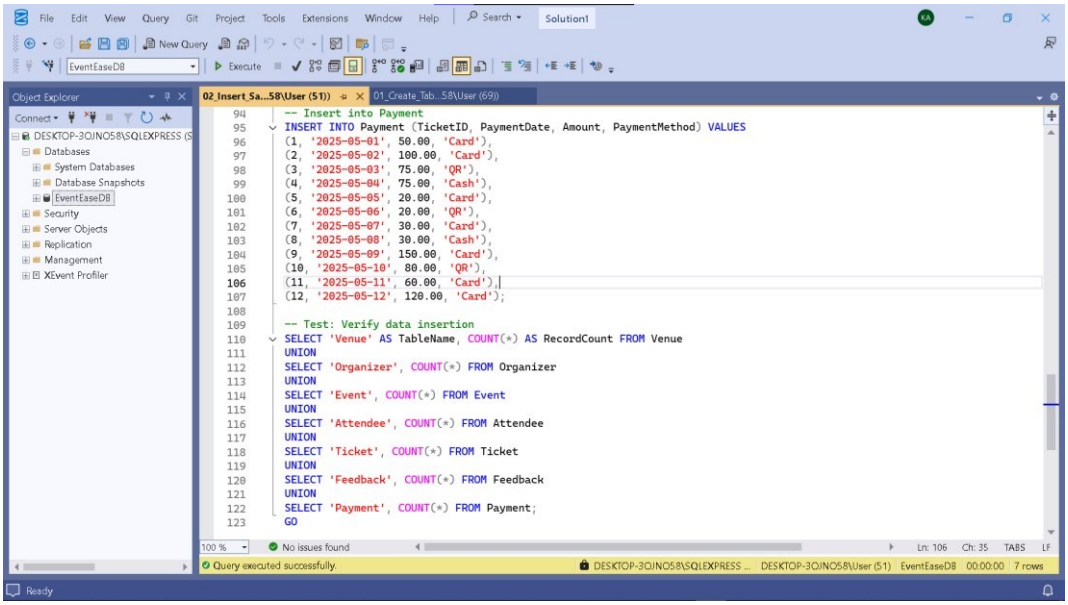
```sql
-- Insert into Organizer
INSERT INTO Organizer (Name, Email, Phone) VALUES
('EventMasters', 'eventmasters@gmail.com', '0112345678'),
('CelebrateNow', 'celebrate@now.lk', '0113456789'),
('JoyEvents', 'joyevents@gmail.com', '0114567890'),
('StarPlanners', 'star@planners.lk', '0115678901'),
('EliteEvents', 'elite@events.lk', '0116789012'),
('VibeOrganizers', 'vibe@organizers.lk', '0117890123'),
('DreamEvents', 'dream@events.lk', '0118901234'),
('FestiveCo', 'festive@co.lk', '01190123456'),
('PlanIt', 'planit@gmail.com', '0110123456'),
('EpicEvents', 'epic@events.lk', '0111234567'),
('GalaGurus', 'gala@gurus.lk', '0112345679'),
('SparkleEvents', 'sparkle@events.lk', '0113456780');

-- 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', '2025-12-12', '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', '2026-03-10', '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);
```



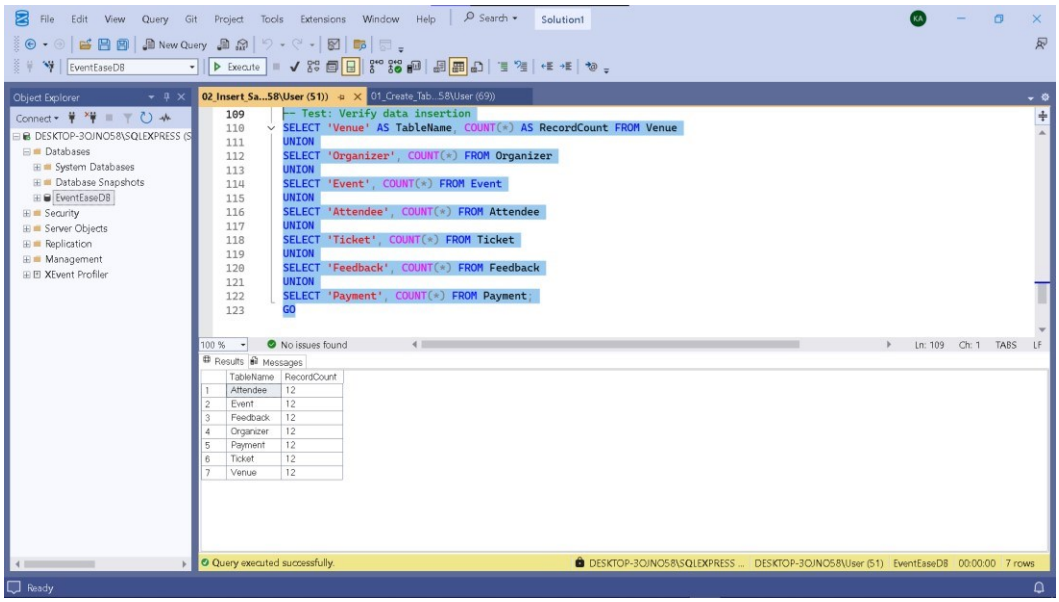
```sql
-- Insert into Event
INSERT INTO Event (Name, Description, Type, StartDate, EndDate, StartTime, EndTime, VenueID, OrganizerID) VALUES
('Tech Conference 2025', 'Annual tech summit', 'Conference', '2025-06-15', '2025-06-17', '09:00', '17:00', 1, 1),
('Rock Concert', 'Live music performance', 'Concert', '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', '2025-12-12', '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', '2026-03-10', '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', '0771234568', '2025-05-11'),
('Ethan Moore', 'ethan.moore@gmail.com', '0772345679', '2025-05-12');

-- Insert into Ticket
```



```
-- Insert into Ticket
INSERT INTO Ticket (EventID, AttendeeID, TicketType, Price, Status) VALUES
(1, 1, 'Regular', 50.00, 'Booked'),
(1, 2, 'VIP', 100.00, 'Booked'),
(2, 3, 'Regular', 75.00, 'Booked'),
(2, 4, 'Regular', 75.00, 'Booked'),
(3, 5, 'Regular', 20.00, 'Booked'),
(3, 6, 'Regular', 20.00, 'Booked'),
(4, 7, 'Regular', 30.00, 'Booked'),
(4, 8, 'Regular', 30.00, 'Booked'),
(5, 9, 'VIP', 150.00, 'Booked'),
(5, 10, 'Regular', 80.00, 'Booked'),
(6, 11, 'Regular', 60.00, 'Booked'),
(6, 12, 'VIP', 120.00, 'Booked');

-- Insert into Feedback
INSERT INTO Feedback (EventID, AttendeeID, Rating, Comment, SubmittedDate) VALUES
(1, 1, 4, 'Great event, good speakers', '2025-06-18'),
(1, 2, 5, 'Amazing experience!', '2025-06-18'),
(2, 3, 3, 'Sound quality could be better', '2025-07-11'),
(2, 4, 4, 'Enjoyed the concert', '2025-07-11'),
(3, 5, 5, 'Loved the book launch', '2025-08-06'),
(3, 6, 4, 'Interesting event', '2025-08-06'),
(4, 7, 4, 'Good networking', '2025-09-13'),
(4, 8, 3, 'More sessions needed', '2025-09-13'),
(5, 9, 5, 'Fantastic jazz night', '2025-10-21'),
(5, 10, 4, 'Great atmosphere', '2025-10-21'),
(6, 11, 4, 'Informative summit', '2025-11-17'),
(6, 12, 5, 'Well organized', '2025-11-17');

-- Insert into Payment
```



```
-- Insert into Payment
INSERT INTO Payment (TicketID, PaymentDate, Amount, PaymentMethod) VALUES
(1, '2025-05-01', 50.00, 'Card'),
(2, '2025-05-02', 100.00, 'Card'),
(3, '2025-05-03', 75.00, 'QR'),
(4, '2025-05-04', 75.00, 'Cash'),
(5, '2025-05-05', 20.00, 'Card'),
(6, '2025-05-06', 20.00, 'QR'),
(7, '2025-05-07', 30.00, 'Card'),
(8, '2025-05-08', 30.00, 'Cash'),
(9, '2025-05-09', 150.00, 'Card'),
(10, '2025-05-10', 80.00, 'QR'),
(11, '2025-05-11', 60.00, 'Card'),
(12, '2025-05-12', 120.00, 'Card');

-- Test: Verify data insertion
SELECT 'Venue' AS TableName, COUNT(*) AS RecordCount FROM Venue
UNION
SELECT 'Organizer', COUNT(*) FROM Organizer
UNION
SELECT 'Event', COUNT(*) FROM Event
UNION
SELECT 'Attendee', COUNT(*) FROM Attendee
UNION
SELECT 'Ticket', COUNT(*) FROM Ticket
UNION
SELECT 'Feedback', COUNT(*) FROM Feedback
UNION
SELECT 'Payment', COUNT(*) FROM Payment;
GO
```



```
-- Test: Verify data insertion
SELECT 'Venue' AS TableName, COUNT(*) AS RecordCount FROM Venue
UNION
SELECT 'Organizer', COUNT(*) FROM Organizer
UNION
SELECT 'Event', COUNT(*) FROM Event
UNION
SELECT 'Attendee', COUNT(*) FROM Attendee
UNION
SELECT 'Ticket', COUNT(*) FROM Ticket
UNION
SELECT 'Feedback', COUNT(*) FROM Feedback
UNION
SELECT 'Payment', COUNT(*) FROM Payment;
GO
```

	TableName	RecordCount
1	Attendee	12
2	Event	12
3	Feedback	12
4	Organizer	12
5	Payment	12
6	Ticket	12
7	Venue	12

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.

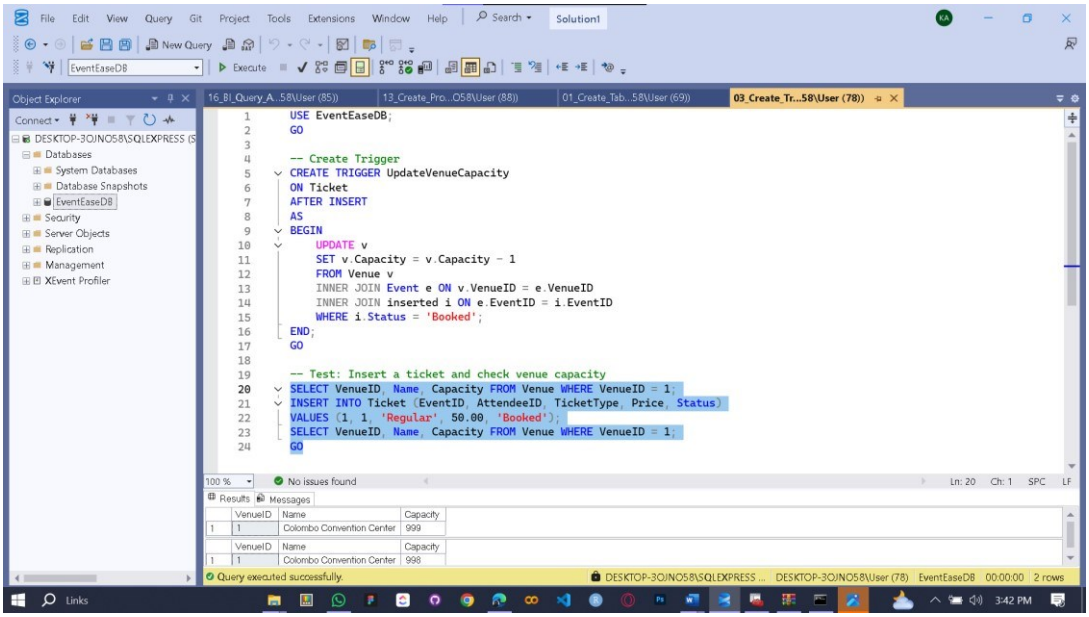


Image 4: The UpdateVenueCapacity trigger and a test showing capacity decrease.

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.

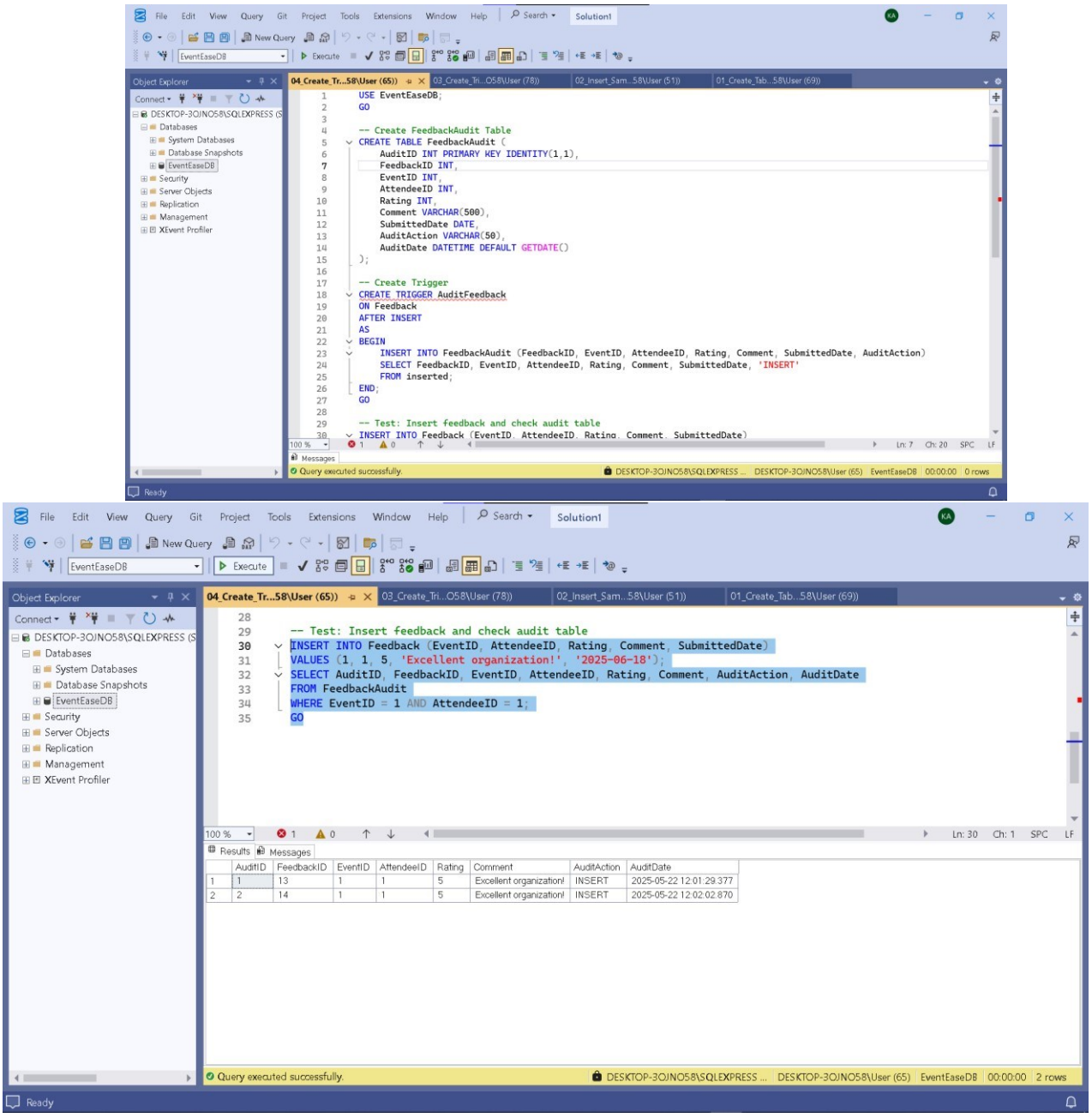


Image 5: The AuditFeedback trigger and a test showing a logged audit entry.

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.

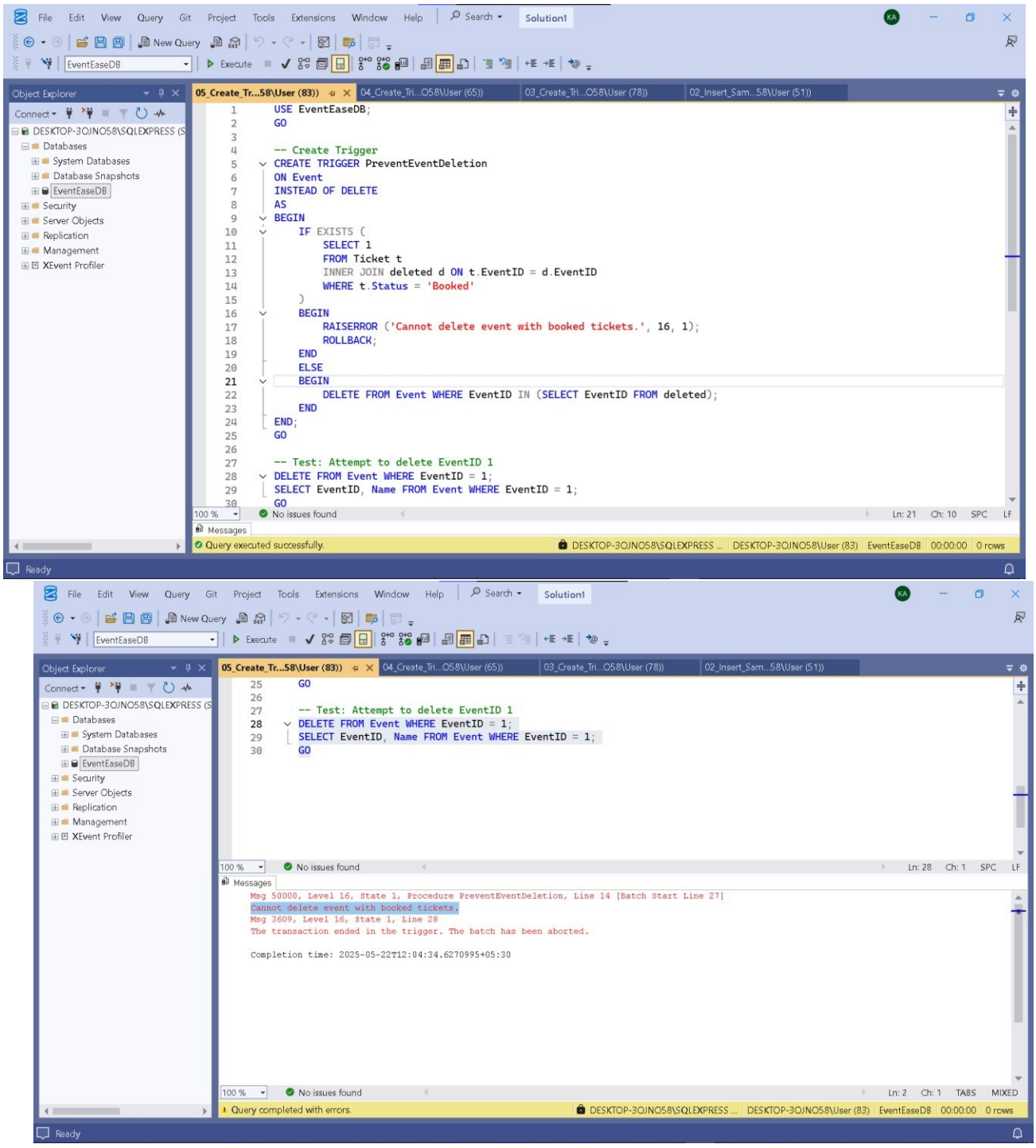


Image 6: The *PreventEventDeletion trigger and a test demonstrating the prevention of deletion for an event with booked tickets.*

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.

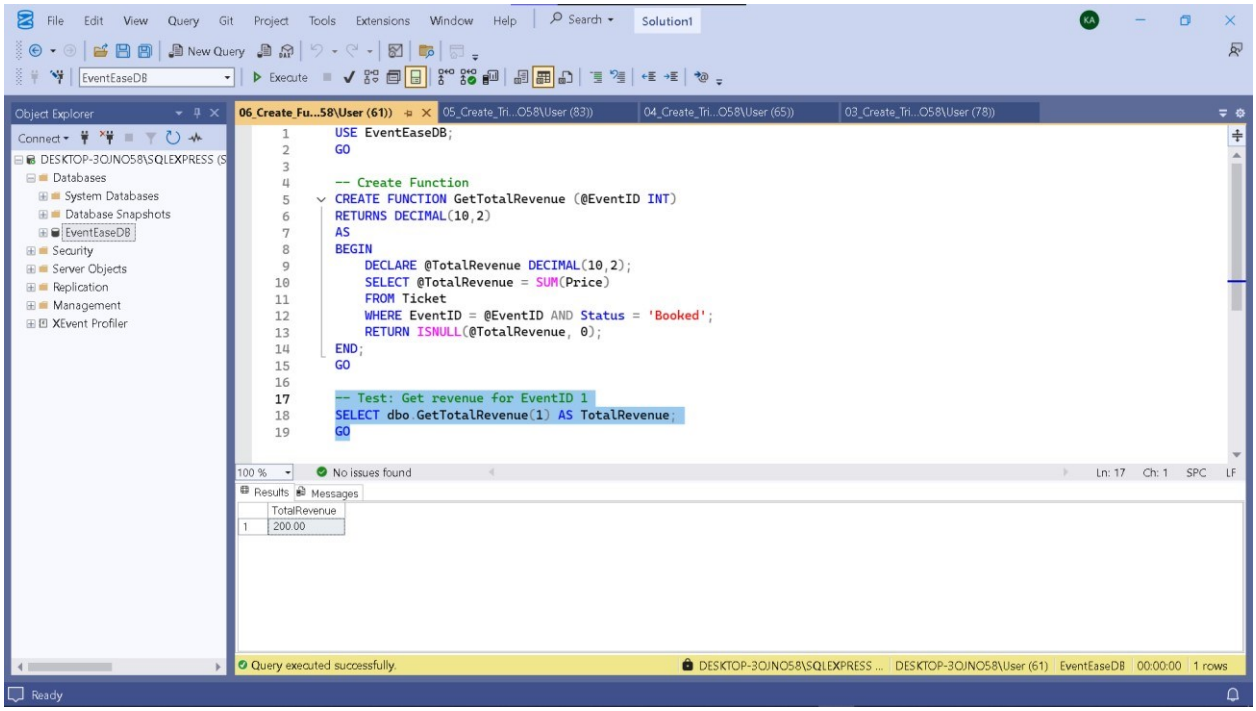


Image 7: GetTotalRevenue Function Script and Test.

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'.

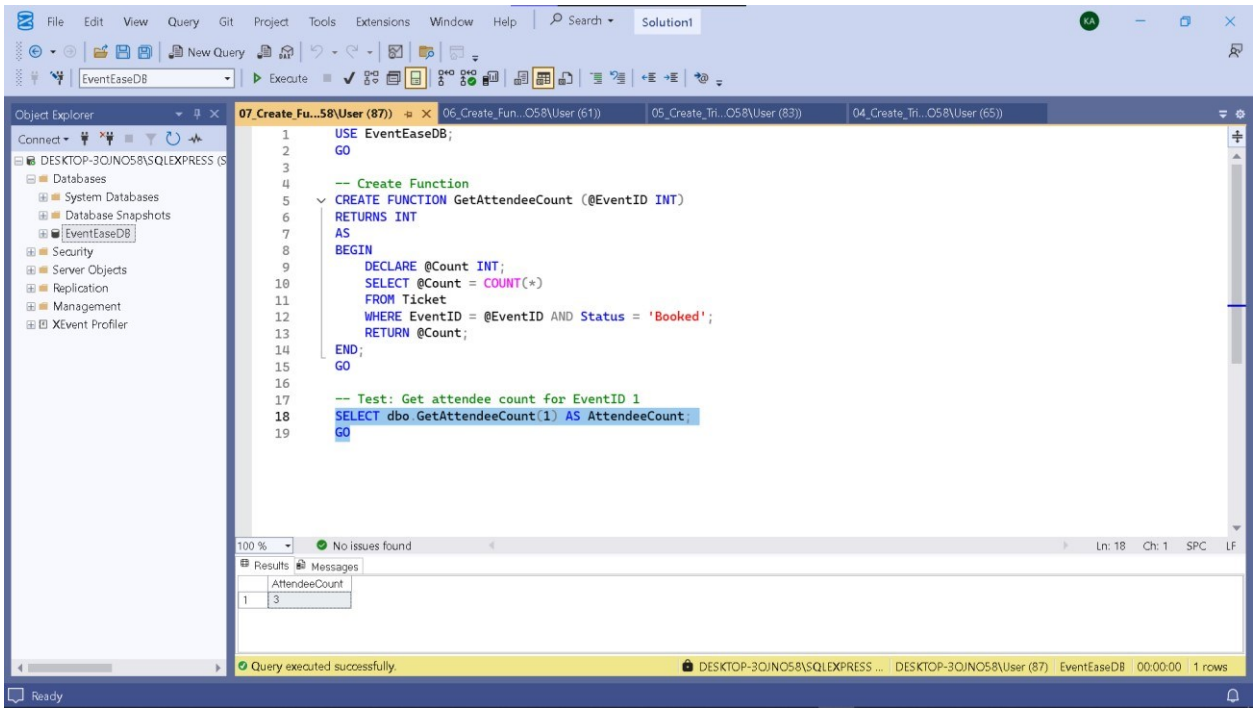


Image 8: GetAttendeeCount Function Script and Test.

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.

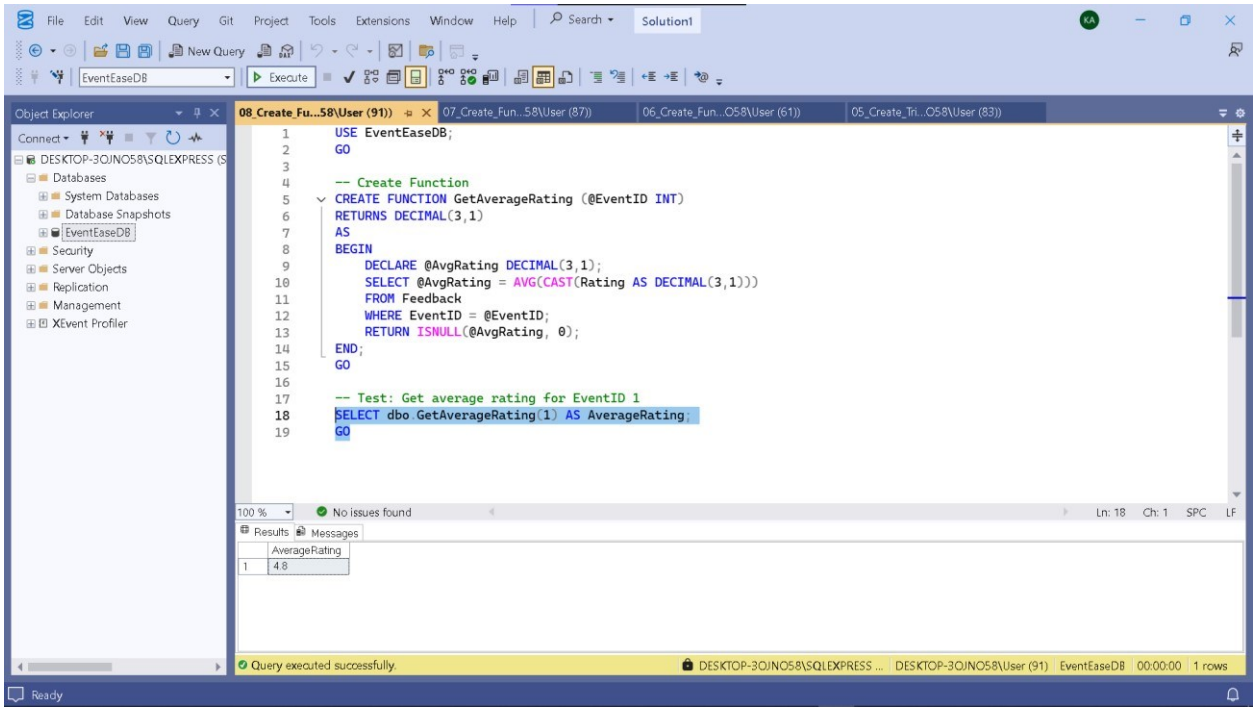


Image 9: GetAverageRating Function Query and Test

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.

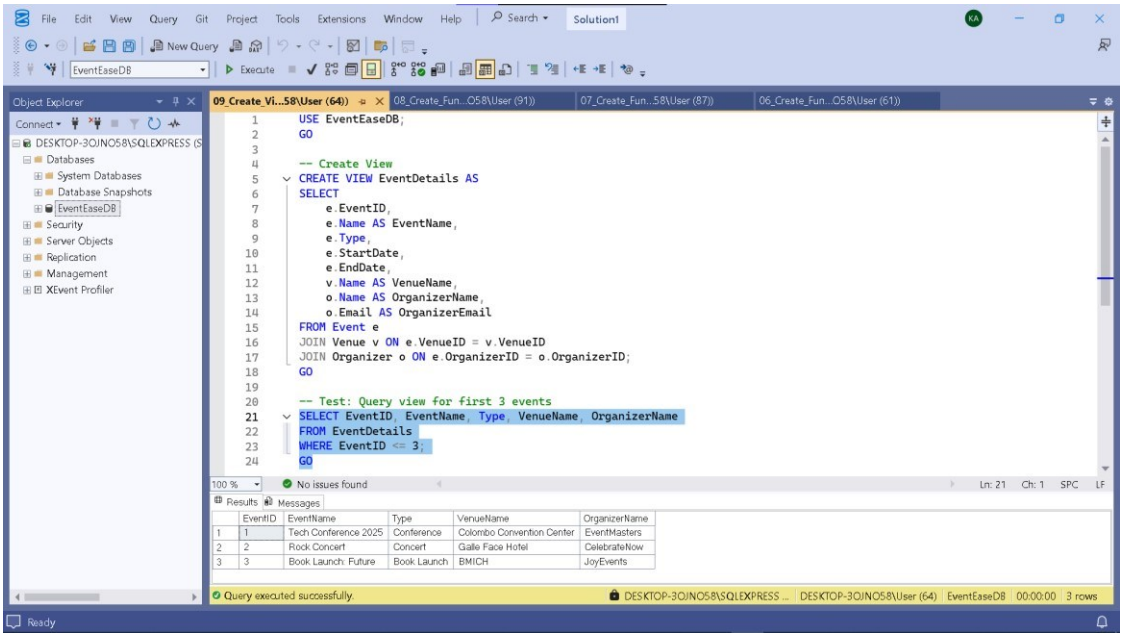


Image 10: EventDetails View Query and Test.

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.

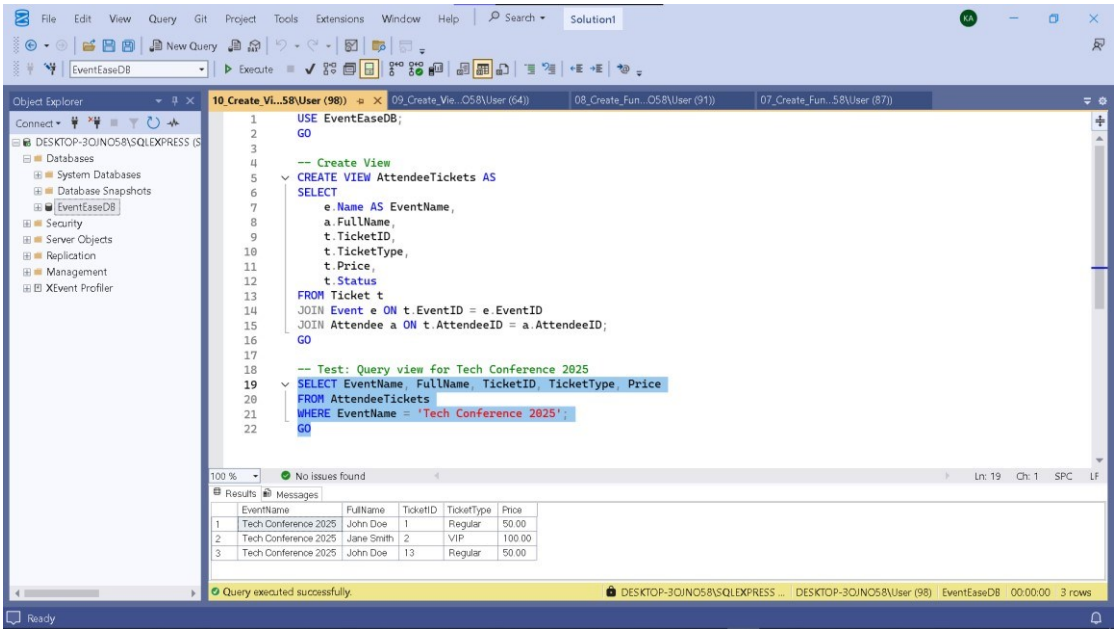


Image 11: AttendeeTickets View Query and Test

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.

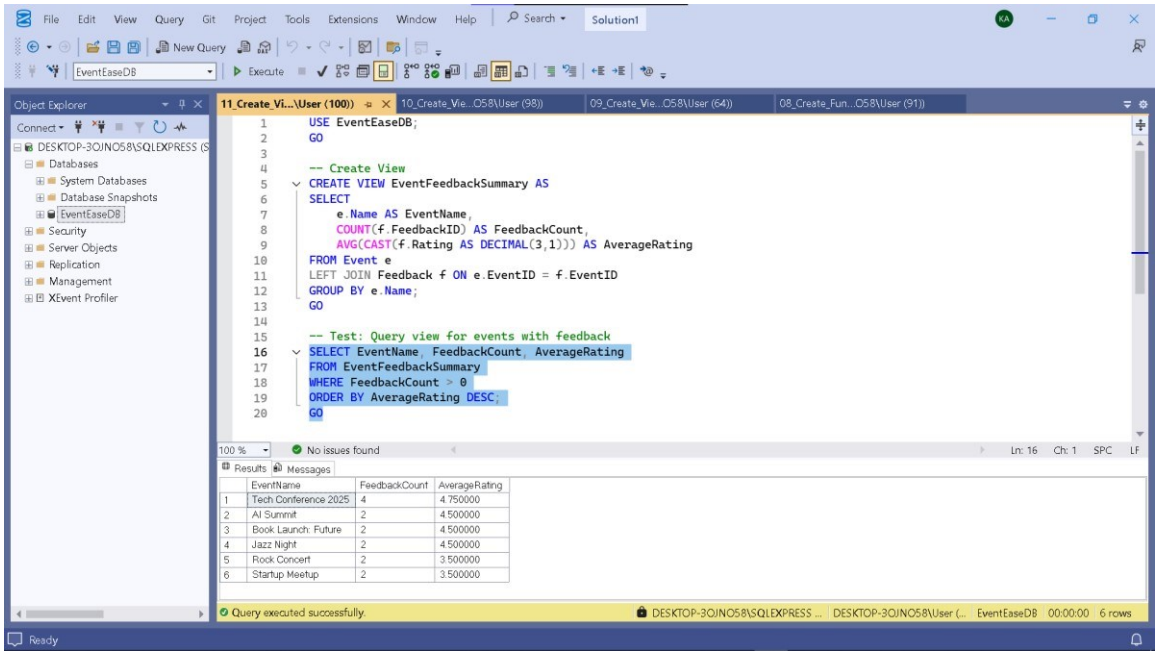


Image 13: EventFeedbackSummary View Query and Test

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.

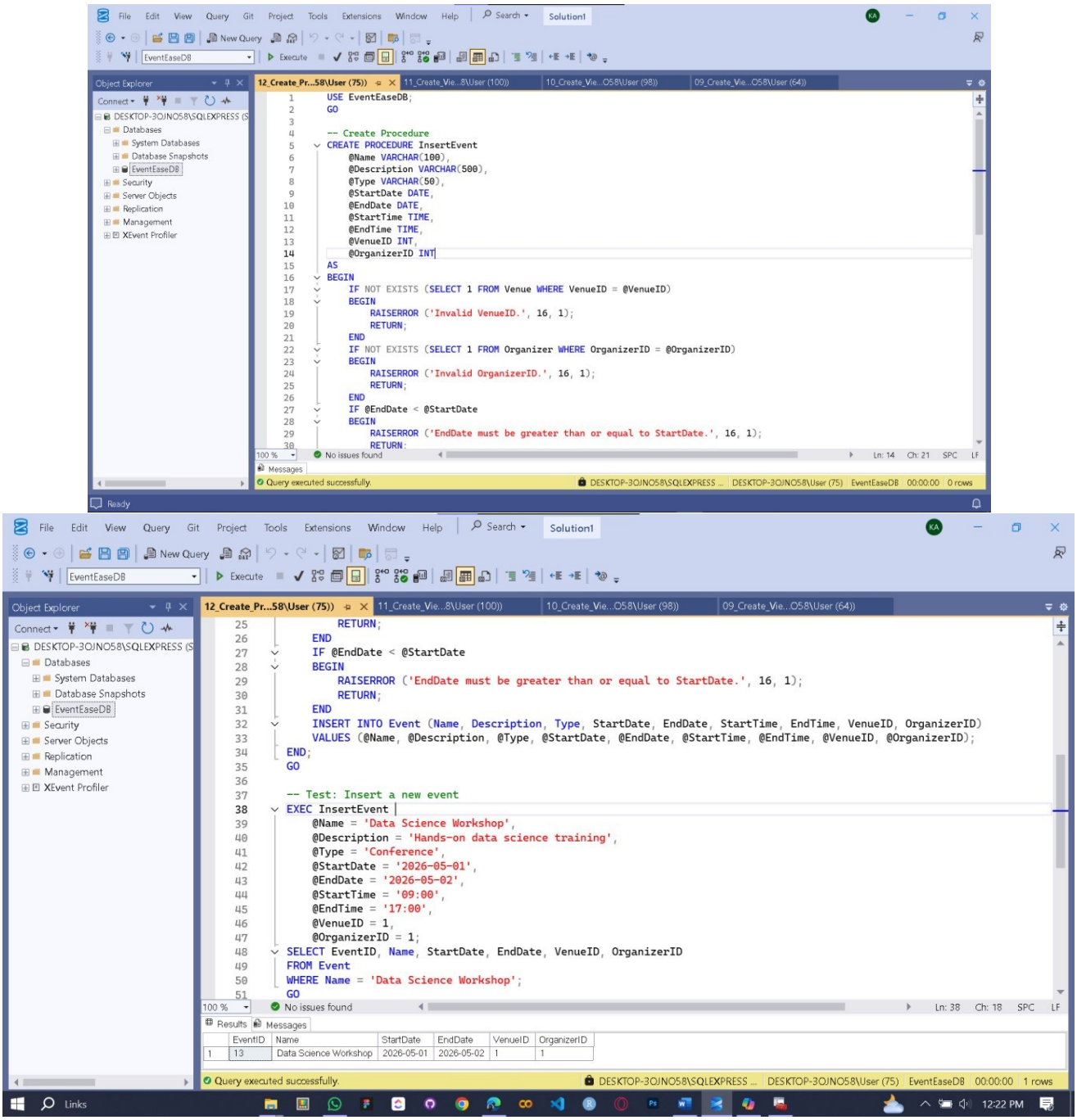


Image 14: InsertEvent Procedure Query and Test II.

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.

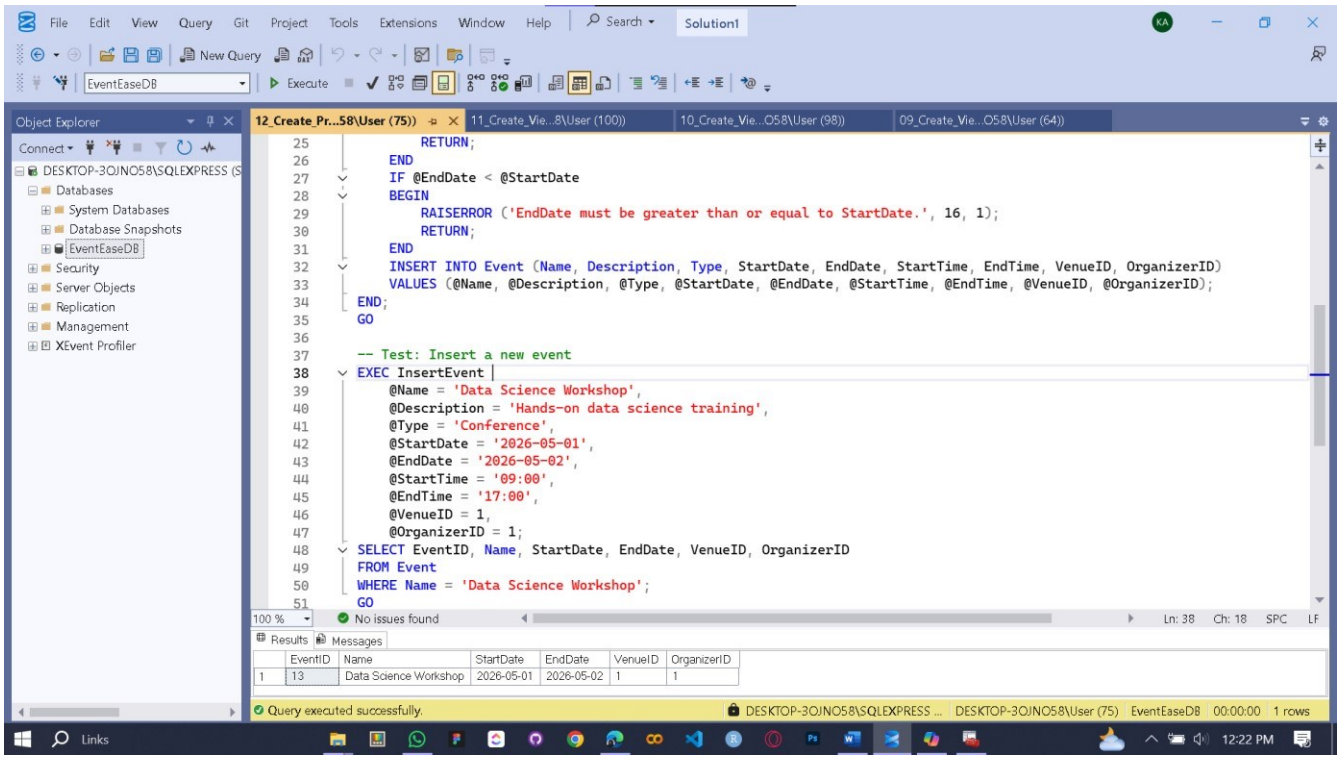


Image 15: TicketSalesReport Procedure Query and Test

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

```python
from flask import Flask, render_template
import pyodbc

app = Flask(__name__)

# SQL Server connection configuration
conn_str = (
    "DRIVER={ODBC Driver 17 for SQL Server};"
    "SERVER=localhost\\SQLEXPRESS;"
    "DATABASE=EventEaseDB;"
    "Trusted_Connection=yes;"
)

@app.route('/')
def index():
    try:
        # Connect to EventEaseDB
        conn = pyodbc.connect(conn_str)
        cursor = conn.cursor()

        # Query EventDetails view
        cursor.execute("""
            SELECT EventID, EventName, Type, StartDate, VenueName
            FROM EventDetails
            ORDER BY StartDate
        """)
        events = cursor.fetchall()

        # Convert to list of dictionaries for template
        events_list = [
            {
                'EventID': row.EventID,
                'EventName': row.EventName,
                'Type': row.Type,
                'StartDate': row.StartDate.strftime('%Y-%m-%d'),
                'VenueName': row.VenueName
            } for row in events
        ]

        cursor.close()
        conn.close()

        return render_template('index.html', events=events_list)

    except Exception as e:
        return f"Error: {str(e)}", 500

if __name__ == '__main__':
    app.run(debug=True)
```

Image 16: code snippet – app.py

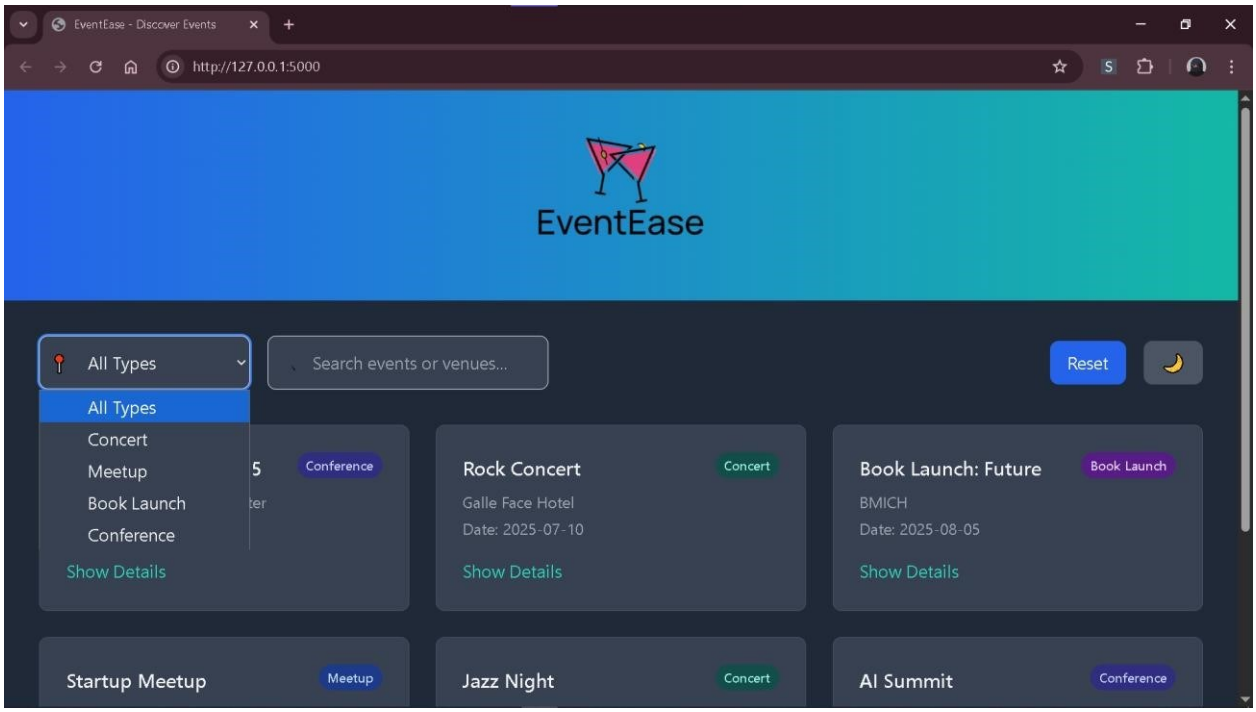


Image 17: Web Application Screenshot - All Events Listing

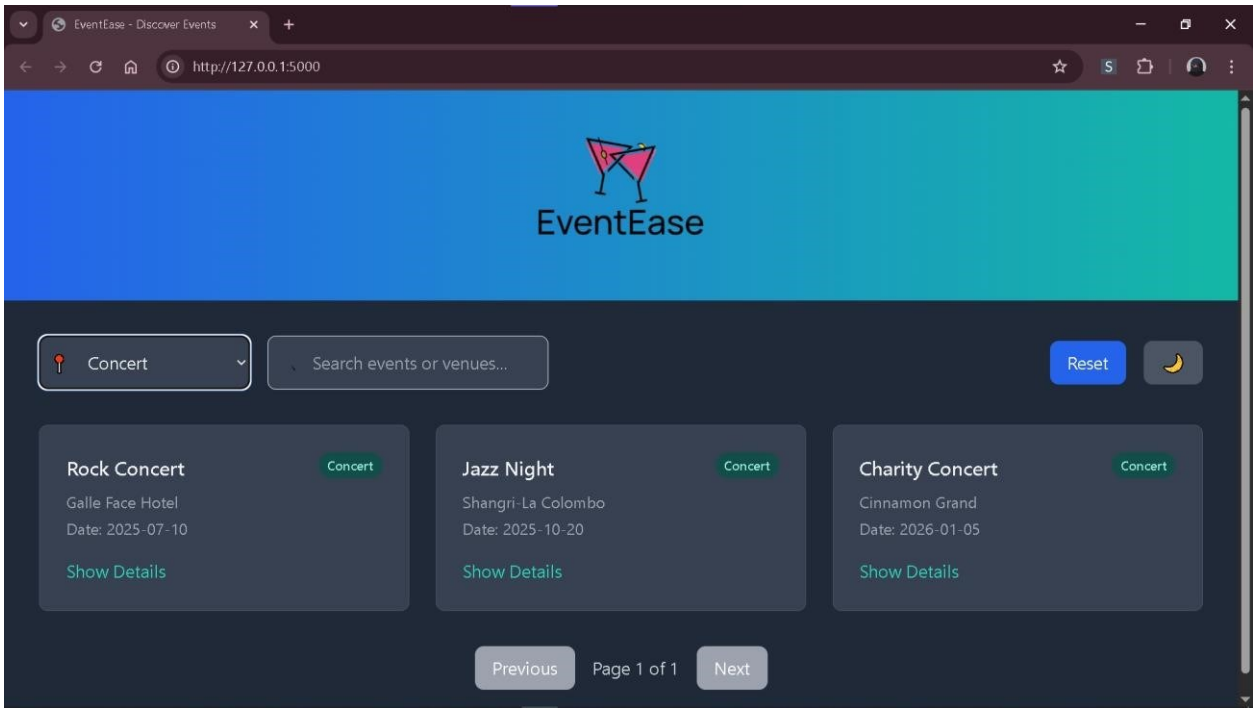


Image 18: Web Application Screenshot - Events Filtered by Type

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.

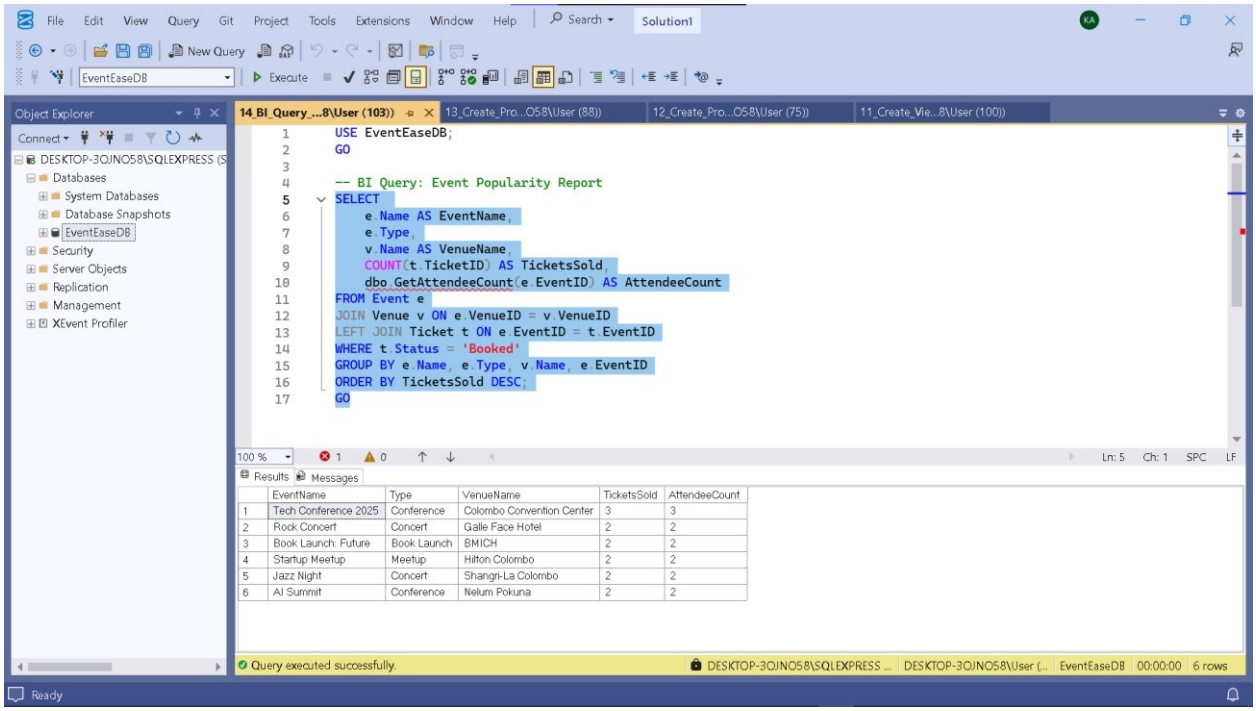


Image 19: Event Popularity BI Query and Result

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.

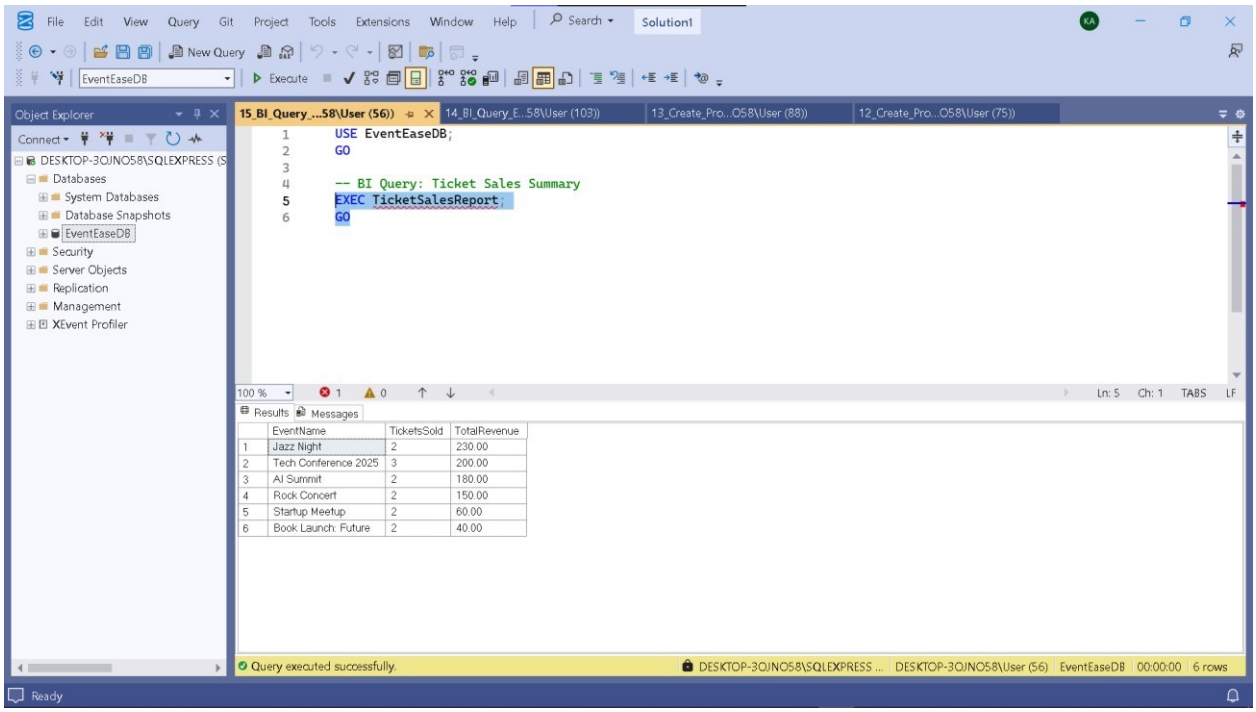


Image 20: Ticket Sales Summary BI Query and Result

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.

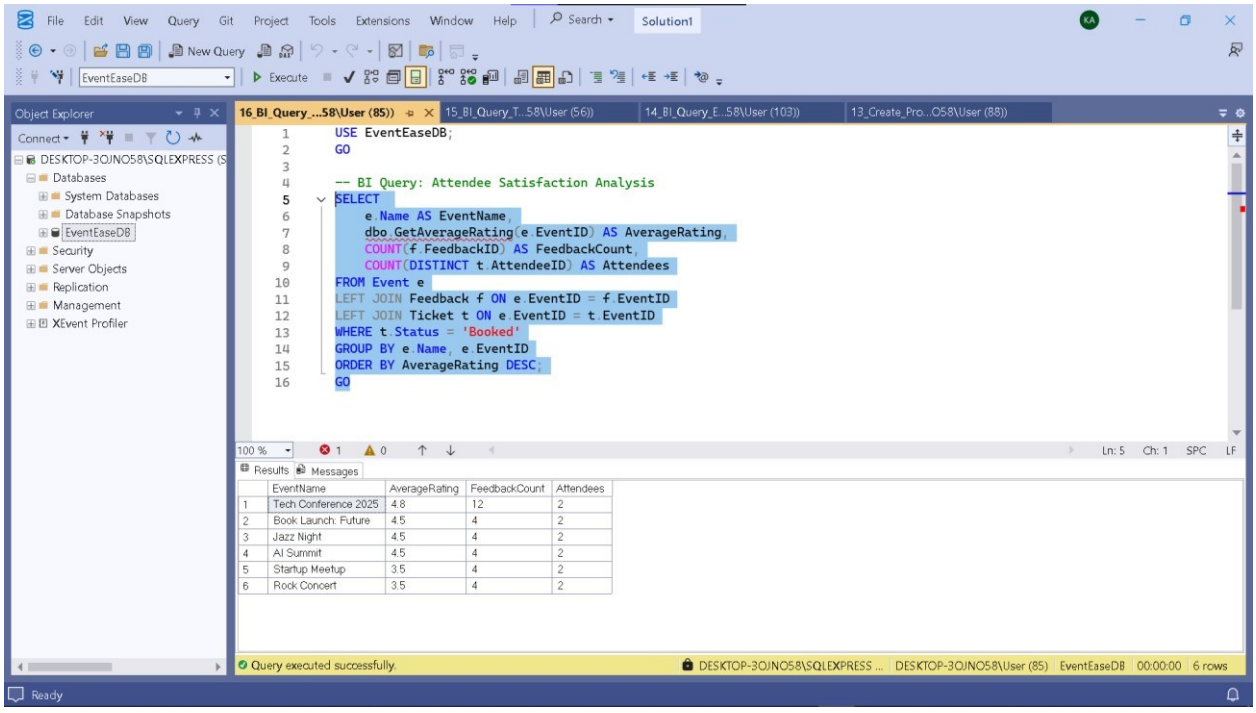


Image 21: Attendee Satisfaction Analysis BI Query and Result

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-usingwindows-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_H9i8b2oQrNq_g?e=ggr5lI*