# Create Database and Tables

-- CREATE DATABASE event;

-- USE event;

CREATE TABLE Role (

ID INT PRIMARY KEY IDENTITY(1,1),

URole VARCHAR(10) CHECK(URole IN ('Admin', 'User')) NOT NULL

);

CREATE TABLE Gender (

ID INT PRIMARY KEY IDENTITY(1,1),

UGender VARCHAR(10) CHECK(UGender IN ('Male', 'Female', 'N/A'))

);

CREATE TABLE "User" (

ID INT PRIMARY KEY IDENTITY(1,1),

URoleID INT,

UGenderID INT,

FullName VARCHAR(100) NOT NULL,

Email VARCHAR(200) UNIQUE NOT NULL,

Phone VARCHAR(20),

Postcode VARCHAR(7),

DateOfBirth DATE CHECK (DateOfBirth < getdate()),

Pass VARCHAR(max) CHECK (LEN(Pass) BETWEEN 8 AND 16)

FOREIGN KEY (URoleID) REFERENCES Role(ID),

FOREIGN KEY (UGenderID) REFERENCES Gender(ID)

);

CREATE TABLE Category (

ID INT PRIMARY KEY IDENTITY(1,1),

Category VARCHAR(max) CHECK (Category IN ('AcademicAndEducationalEvents',

'SocialEvents', 'CulturalAndEntertainmentEvents', 'BusinessAndNetworkingEvents'))

);

CREATE TABLE Venue (

ID INT PRIMARY KEY IDENTITY(1,1),

CategoryID INT,

Title VARCHAR(100) UNIQUE NOT NULL,

CreatedOn DATE DEFAULT GETDATE(),

CreatedBy VARCHAR(200),

FOREIGN KEY (CategoryID) REFERENCES Category(ID)

);

CREATE TABLE Booking (

UserID INT,

VenueID INT,

ReservedDate DATE DEFAULT GETDATE(),

PRIMARY KEY (UserID, VenueID),

FOREIGN KEY (UserID) REFERENCES "User"(ID),

FOREIGN KEY (VenueID) REFERENCES Venue(ID)

);

CREATE TABLE Review (

ID INT PRIMARY KEY IDENTITY(1,1),

UserID INT,

VenueID INT,

Rating INT CHECK (Rating between 1 and 5),

ReviewText TEXT,

ReviewDate DATE DEFAULT GETDATE(),

FOREIGN KEY (UserID) REFERENCES "User"(ID),

FOREIGN KEY (VenueID) references Venue(ID)

);

# Alter Tables

ALTER TABLE Venue add TotalCapacity INT NOT NULL; -- Total capacity of the venue

ALTER TABLE Venue

ADD Vacancy INT CONSTRAINT DF\_Venue\_Vacancy DEFAULT 0; -- Remaining available slots

# Show Tables

SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_TYPE = 'BASE TABLE';

# Drop Tables

drop table if exists booking;

drop table if exists review;

drop table if exists register;

drop table if exists venue;

drop table if exists position;

drop table if exists gender;

drop table if exists category;

# Prevent Booking Existing Hired Venue Trigger

CREATE TRIGGER preventBookingExistingHiredVenue

ON Booking

AFTER INSERT

AS

BEGIN

DECLARE @UserID INT;

DECLARE @VenueID INT;

SELECT @UserID = UserID, @VenueID = VenueID FROM INSERTED;

IF EXISTS (

SELECT 1

FROM Booking

WHERE UserID = @UserID

AND VenueID = @VenueID

)

BEGIN

RAISERROR('An already booked venue cannot be reserved again.', 16,1);

RETURN;

END

IF EXISTS (

SELECT 1

FROM Venue

WHERE ID = @VenueID

AND Vacancy <= 0

)

BEGIN

RAISERROR('The venue does not have available space.', 16,1);

RETURN;

END

END; -- trigger

# Update Vacancy Insert Booking Trigger

CREATE TRIGGER updateVacancyInsertBooking

ON Booking

AFTER INSERT

AS

BEGIN

-- Update the vacancy for all venues affected by the inserted bookings

UPDATE v

SET v.Vacancy = v.Vacancy - 1

FROM Venue v

INNER JOIN inserted i ON v.id = i.VenueID

WHERE v.Vacancy > 0;

END;

# Update Vacancy Cancel Booking Trigger

CREATE TRIGGER updateVacancyCancelBooking

ON Booking

AFTER DELETE

AS

BEGIN

-- Update the vacancy for venues associated with the deleted bookings

UPDATE v

SET v.Vacancy = v.Vacancy + 1

FROM Venue v

INNER JOIN deleted d ON v.ID = d.VenueID

WHERE v.Vacancy >= 0; -- Ensure vacancy doesn't go negative

END;

# Insert Into Tables

INSERT INTO position (urole)

VALUES ('Admin'), ('User');

INSERT INTO gender (ugender)

VALUES ('Male'), ('Female'), ('N/A');

INSERT INTO category (category)

VALUES ('AcademicAndEducationalEvents'),

('SocialEvents'),

('CulturalAndEntertainmentEvents'),

('BusinessAndNetworkingEvents');

-- Inserting venues with the new columns: totalCapacity and vacancy

INSERT INTO venue (categoryid, title, createdBy, totalCapacity, vacancy)

VALUES (1, 'Science fairs', 'john.doe@example.com', 100, 100), -- Venue 1, Category 1, with 100 total capacity and 100 vacancies

(2, 'Wedding', 'jane.smith@example.com', 200, 200); -- Venue 2, Category 2, with 200 total capacity and 200 vacancies

INSERT INTO register (uroleid, ugender, fullName, email, phone, postcode, dateOfBirth, pass)

VALUES (1, 1, 'John Doe', 'john.doe@example.com', '1234567890', '1234567', '1990-05-15', 'password123'),

(2, 2, 'Jane Smith', 'jane.smith@example.com', '0987654321', '7654321', '2000-01-25', 'password456');

INSERT INTO booking (userid, venueid)

VALUES (1, 1),

(2, 2);

INSERT INTO review (userid, venueid, rating, reviewtext)

VALUES (1, 1, 5, 'Great event space, highly recommend!'),

(2, 2, 4, 'Nice venue but could improve acoustics.');

# Select queries

SELECT \* FROM venue;

SELECT v.id AS VenueID, v.createdBy, c.category AS VenueCategory

FROM venue v

JOIN category c ON v.categoryid = c.id;

SELECT b.userid, r.fullName AS 'Full Name', r.email, r.uroleid, b.venueid, b.reserveddate

FROM booking b

INNER JOIN register r ON b.userid = r.id

WHERE b.userid = 1;

SELECT r.rating, r.reviewtext, r.reviewdate

FROM review r

WHERE r.venueid = 1;

SELECT r.fullName, r.email, v.createdBy AS VenueName

FROM register r

JOIN booking b ON r.id = b.userid

JOIN venue v ON b.venueid = v.id;

# Join queries

SELECT b.userid, r.fullName, v.createdBy AS VenueName, b.reserveddate

FROM booking b

JOIN register r ON b.userid = r.id

JOIN venue v ON b.venueid = v.id;

SELECT r.fullName, v.createdBy AS VenueName, rev.rating, rev.reviewtext

FROM review rev

JOIN register r ON rev.userid = r.id

JOIN venue v ON rev.venueid = v.id;

SELECT v.createdBy AS VenueName, c.category AS VenueCategory, v.totalCapacity

FROM venue v

JOIN category c ON v.categoryid = c.id;

SELECT r.fullName, rev.reviewtext, v.createdBy AS VenueName

FROM review rev

JOIN register r ON rev.userid = r.id

JOIN venue v ON rev.venueid = v.id;

SELECT b.userid, r.fullName AS 'Full Name', p.urole As 'Role',

b.venueid, v.title AS 'Venue Name', v.totalCapacity, v.vacancy, b.reserveddate

FROM booking b

JOIN register r ON b.userid = r.id

JOIN venue v ON b.venueid = v.id

JOIN position p ON r.uroleid = p.id;