DROP DATABASE IF EXISTS CinemaDB;

CREATE DATABASE CinemaDB;

USE CinemaDB;

CREATE TABLE Movies (

movie\_id INT PRIMARY KEY AUTO\_INCREMENT,

title VARCHAR(255) NOT NULL,

genre VARCHAR(100),

duration\_min INT,

director VARCHAR(100),

actors TEXT,

language VARCHAR(50),

age\_rating VARCHAR(10),

subtitle\_dubbing VARCHAR(50),

release\_date DATE

);

CREATE TABLE Rooms (

room\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

capacity INT,

room\_type VARCHAR(50) COMMENT '2D, 3D, IMAX',

equipment TEXT

);

CREATE TABLE Seats (

seat\_id INT PRIMARY KEY AUTO\_INCREMENT,

room\_id INT,

seat\_number VARCHAR(10) NOT NULL,

seat\_type VARCHAR(50) COMMENT 'thường, VIP',

FOREIGN KEY (room\_id) REFERENCES Rooms(room\_id)

);

CREATE TABLE Showtimes (

showtime\_id INT PRIMARY KEY AUTO\_INCREMENT,

movie\_id INT,

room\_id INT,

show\_time DATETIME NOT NULL,

base\_price DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (movie\_id) REFERENCES Movies(movie\_id),

FOREIGN KEY (room\_id) REFERENCES Rooms(room\_id)

);

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY AUTO\_INCREMENT,

full\_name VARCHAR(100) NOT NULL,

birth\_date DATE,

email VARCHAR(100) UNIQUE,

phone\_number VARCHAR(20),

membership\_rank VARCHAR(50) DEFAULT 'Standard',

points INT DEFAULT 0

);

CREATE TABLE StaffMembers (

staff\_id INT PRIMARY KEY AUTO\_INCREMENT,

full\_name VARCHAR(100) NOT NULL,

position VARCHAR(50),

username VARCHAR(50) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

manager\_id INT,

FOREIGN KEY (manager\_id) REFERENCES StaffMembers(staff\_id)

);

CREATE TABLE Payments (

payment\_id INT PRIMARY KEY AUTO\_INCREMENT,

payment\_method VARCHAR(50),

transaction\_date DATETIME,

amount DECIMAL(10, 2),

status VARCHAR(50) COMMENT 'Completed, Pending, Failed'

);

CREATE TABLE Bookings (

booking\_id INT PRIMARY KEY AUTO\_INCREMENT,

customer\_id INT,

staff\_id INT,

payment\_id INT UNIQUE,

booking\_time DATETIME,

total\_amount DECIMAL(10, 2),

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id),

FOREIGN KEY (staff\_id) REFERENCES StaffMembers(staff\_id),

FOREIGN KEY (payment\_id) REFERENCES Payments(payment\_id)

);

CREATE TABLE Tickets (

ticket\_id INT PRIMARY KEY AUTO\_INCREMENT,

booking\_id INT,

showtime\_id INT,

seat\_id INT,

price DECIMAL(10, 2),

status VARCHAR(50) DEFAULT 'Sold' COMMENT 'Sold, Cancelled',

FOREIGN KEY (booking\_id) REFERENCES Bookings(booking\_id),

FOREIGN KEY (showtime\_id) REFERENCES Showtimes(showtime\_id),

FOREIGN KEY (seat\_id) REFERENCES Seats(seat\_id)

);

DELIMITER //

CREATE TRIGGER trg\_AfterPaymentUpdate\_AddPointsOnComplete

AFTER UPDATE ON Payments

FOR EACH ROW

BEGIN

DECLARE customer\_id\_to\_update INT;

DECLARE total\_amount\_for\_points DECIMAL(10, 2);

IF OLD.status <> 'Completed' AND NEW.status = 'Completed' THEN

SELECT b.customer\_id, b.total\_amount INTO customer\_id\_to\_update, total\_amount\_for\_points

FROM Bookings b WHERE b.payment\_id = NEW.payment\_id;

IF customer\_id\_to\_update IS NOT NULL THEN

UPDATE Customers SET points = points + FLOOR(total\_amount\_for\_points / 1000)

WHERE customer\_id = customer\_id\_to\_update;

END IF;

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeCustomerUpdate\_CheckRank

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

IF NEW.points > OLD.points THEN

IF NEW.points >= 5000 THEN SET NEW.membership\_rank = 'Platinum';

ELSEIF NEW.points >= 2000 THEN SET NEW.membership\_rank = 'Gold';

ELSEIF NEW.points >= 500 THEN SET NEW.membership\_rank = 'Silver';

END IF;

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_AfterTicketUpdate\_DeductPointsOnCancel

AFTER UPDATE ON Tickets

FOR EACH ROW

BEGIN

DECLARE customer\_id\_to\_update INT;

DECLARE points\_to\_deduct INT;

IF OLD.status <> 'Cancelled' AND NEW.status = 'Cancelled' THEN

SELECT b.customer\_id INTO customer\_id\_to\_update

FROM Bookings b WHERE b.booking\_id = NEW.booking\_id;

IF customer\_id\_to\_update IS NOT NULL THEN

SET points\_to\_deduct = FLOOR(NEW.price / 1000);

UPDATE Customers SET points = points - points\_to\_deduct

WHERE customer\_id = customer\_id\_to\_update;

END IF;

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeCustomerDelete\_CheckFutureTickets

BEFORE DELETE ON Customers

FOR EACH ROW

BEGIN

DECLARE future\_tickets INT;

SELECT COUNT(\*) INTO future\_tickets

FROM Bookings b

JOIN Tickets t ON b.booking\_id = t.booking\_id

JOIN Showtimes s ON t.showtime\_id = s.showtime\_id

WHERE b.customer\_id = OLD.customer\_id AND s.show\_time >= NOW() AND t.status <> 'Cancelled';

IF future\_tickets > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Không thể xóa tài khoản này. Khách hàng vẫn còn vé cho các suất chiếu trong tương lai.';

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeTicketInsert\_PreventDoubleBooking

BEFORE INSERT ON Tickets

FOR EACH ROW

BEGIN

DECLARE seat\_is\_taken INT;

SELECT COUNT(\*) INTO seat\_is\_taken

FROM Tickets

WHERE showtime\_id = NEW.showtime\_id AND seat\_id = NEW.seat\_id AND status <> 'Cancelled';

IF seat\_is\_taken > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Lỗi: Ghế này đã được người khác đặt. Vui lòng chọn ghế khác.';

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeBookingUpdate\_VerifyTotalAmount

BEFORE UPDATE ON Bookings

FOR EACH ROW

BEGIN

DECLARE actual\_ticket\_sum DECIMAL(10, 2);

IF OLD.total\_amount <> NEW.total\_amount THEN

SELECT SUM(price) INTO actual\_ticket\_sum

FROM Tickets WHERE booking\_id = OLD.booking\_id;

IF NEW.total\_amount <> actual\_ticket\_sum THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Lỗi Toàn Vẹn: Tổng tiền hóa đơn không khớp với tổng giá trị các vé.';

END IF;

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeTicketUpdate\_PreventPriceChange

BEFORE UPDATE ON Tickets

FOR EACH ROW

BEGIN

IF OLD.price <> NEW.price THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Hành Động Bị Cấm: Không được phép thay đổi giá của một vé đã được bán.';

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeMovieDelete\_CheckShowtimes

BEFORE DELETE ON Movies

FOR EACH ROW

BEGIN

DECLARE future\_showtimes INT;

SELECT COUNT(\*) INTO future\_showtimes

FROM Showtimes WHERE movie\_id = OLD.movie\_id AND show\_time >= NOW();

IF future\_showtimes > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Không thể xóa phim này. Phim vẫn còn các suất chiếu trong tương lai.';

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeShowtimeUpdate\_CheckTickets

BEFORE UPDATE ON Showtimes

FOR EACH ROW

BEGIN

DECLARE tickets\_sold INT;

SELECT COUNT(\*) INTO tickets\_sold

FROM Tickets WHERE showtime\_id = OLD.showtime\_id AND status <> 'Cancelled';

IF tickets\_sold > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Không thể cập nhật. Suất chiếu này đã có vé được bán.';

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeShowtimeDelete\_CheckTickets

BEFORE DELETE ON Showtimes

FOR EACH ROW

BEGIN

DECLARE tickets\_sold INT;

SELECT COUNT(\*) INTO tickets\_sold

FROM Tickets WHERE showtime\_id = OLD.showtime\_id AND status <> 'Cancelled';

IF tickets\_sold > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Không thể xóa. Suất chiếu này đã có vé được bán.';

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeRoomDelete\_CheckFutureShowtimes

BEFORE DELETE ON Rooms

FOR EACH ROW

BEGIN

DECLARE future\_showtimes INT;

SELECT COUNT(\*) INTO future\_showtimes

FROM Showtimes WHERE room\_id = OLD.room\_id AND show\_time >= NOW();

IF future\_showtimes > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Không thể xóa phòng này. Vẫn còn suất chiếu trong tương lai được lên lịch tại đây.';

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeSeatDelete\_CheckFutureBookings

BEFORE DELETE ON Seats

FOR EACH ROW

BEGIN

DECLARE future\_bookings\_for\_seat INT;

SELECT COUNT(\*) INTO future\_bookings\_for\_seat

FROM Tickets t

JOIN Showtimes s ON t.showtime\_id = s.showtime\_id

WHERE t.seat\_id = OLD.seat\_id AND s.show\_time >= NOW() AND t.status = 'Sold';

IF future\_bookings\_for\_seat > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Không thể xóa ghế. Ghế này đã được đặt cho một suất chiếu trong tương lai.';

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_AfterTicketUpdate\_LogCancellation

AFTER UPDATE ON Tickets

FOR EACH ROW

BEGIN

IF OLD.status <> 'Cancelled' AND NEW.status = 'Cancelled' THEN

INSERT INTO AuditLogs (performing\_user, action\_type, table\_name, record\_id, details)

VALUES (CURRENT\_USER(), 'TICKET CANCELLED', 'Tickets', NEW.ticket\_id,

CONCAT('Ticket ID ', NEW.ticket\_id, ' for booking ID ', NEW.booking\_id, ' was cancelled.'));

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeBookingDelete\_Log

BEFORE DELETE ON Bookings

FOR EACH ROW

BEGIN

INSERT INTO AuditLogs (performing\_user, action\_type, table\_name, record\_id, details)

VALUES (CURRENT\_USER(), 'BOOKING DELETED', 'Bookings', OLD.booking\_id,

CONCAT('Booking ID ', OLD.booking\_id, ' for customer ID ', IFNULL(OLD.customer\_id, 'N/A'),

' with total amount ', OLD.total\_amount, ' was deleted.'));

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_AfterMovieInsert\_Log

AFTER INSERT ON Movies

FOR EACH ROW

BEGIN

INSERT INTO AuditLogs (performing\_user, action\_type, table\_name, record\_id, details)

VALUES (CURRENT\_USER(), 'MOVIE CREATED', 'Movies', NEW.movie\_id, CONCAT('New movie added: ', NEW.title));

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_AfterMovieUpdate\_Log

AFTER UPDATE ON Movies

FOR EACH ROW

BEGIN

IF OLD.title <> NEW.title THEN

INSERT INTO AuditLogs (performing\_user, action\_type, table\_name, record\_id, details)

VALUES (CURRENT\_USER(), 'MOVIE UPDATED', 'Movies', NEW.movie\_id,

CONCAT('Title changed from "', OLD.title, '" to "', NEW.title, '".'));

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_AfterStaffInsert\_Log

AFTER INSERT ON StaffMembers

FOR EACH ROW

BEGIN

INSERT INTO AuditLogs (performing\_user, action\_type, table\_name, record\_id, details)

VALUES (CURRENT\_USER(), 'STAFF CREATED', 'StaffMembers', NEW.staff\_id,

CONCAT('New staff account created for: ', NEW.full\_name, ' with position: ', NEW.position));

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_AfterStaffUpdate\_Log

AFTER UPDATE ON StaffMembers

FOR EACH ROW

BEGIN

IF OLD.position <> NEW.position THEN

INSERT INTO AuditLogs (performing\_user, action\_type, table\_name, record\_id, details)

VALUES (CURRENT\_USER(), 'STAFF ROLE CHANGED', 'StaffMembers', NEW.staff\_id,

CONCAT('Position for ', NEW.full\_name, ' changed from "', OLD.position, '" to "', NEW.position, '".'));

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_AfterStaffDelete\_Log

AFTER DELETE ON StaffMembers

FOR EACH ROW

BEGIN

INSERT INTO AuditLogs (performing\_user, action\_type, table\_name, record\_id, details)

VALUES (CURRENT\_USER(), 'STAFF DELETED', 'StaffMembers', OLD.staff\_id,

CONCAT('Staff account deleted. Name: ', OLD.full\_name, ', Username: ', OLD.username));

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_AfterCustomerUpdate\_LogChanges

AFTER UPDATE ON Customers

FOR EACH ROW

BEGIN

DECLARE changes TEXT DEFAULT '';

IF OLD.email <> NEW.email THEN

SET changes = CONCAT(changes, 'Email changed. ');

END IF;

IF OLD.phone\_number <> NEW.phone\_number THEN

SET changes = CONCAT(changes, 'Phone number changed. ');

END IF;

IF changes <> '' THEN

INSERT INTO AuditLogs (performing\_user, action\_type, table\_name, record\_id, details)

VALUES (CURRENT\_USER(), 'CUSTOMER PROFILE UPDATED', 'Customers', NEW.customer\_id,

CONCAT('Customer ID ', NEW.customer\_id, ' updated profile. Details: ', changes));

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeStaffUpdate\_ProtectLastManager

BEFORE UPDATE ON StaffMembers

FOR EACH ROW

BEGIN

DECLARE manager\_count INT;

IF OLD.position = 'Quản lý rạp' AND NEW.position <> 'Quản lý rạp' THEN

SELECT COUNT(\*) INTO manager\_count FROM StaffMembers WHERE position = 'Quản lý rạp';

IF manager\_count <= 1 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Không thể thay đổi vai trò. Đây là Quản lý cuối cùng của hệ thống.';

END IF;

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER trg\_BeforeStaffDelete\_ProtectLastManager

BEFORE DELETE ON StaffMembers

FOR EACH ROW

BEGIN

DECLARE manager\_count INT;

IF OLD.position = 'Quản lý rạp' THEN

SELECT COUNT(\*) INTO manager\_count FROM StaffMembers WHERE position = 'Quản lý rạp';

IF manager\_count <= 1 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Không thể xóa. Đây là Quản lý cuối cùng của hệ thống.';

END IF;

END IF;

END //

DELIMITER ;