Theater Information System

Table of Contents

1. <u>General description of the system</u>

The Afeka Theater located at Mivtza Kadesh St. 38, Tel Aviv. It offers a diverse range of shows, including musicals, comedies, dramas, tragedies, ballets, operas, and more. The theater consists of three halls: Ficus Hall (Hall 101), HaKiriya Hall (Hall 102), and Mapat Hall (Hall 103), each with a different seating capacity.

The theater operates a subscriptions for its customers. Subscribers can choose from various subscription packages, each offering different benefits and durations. The packages are identified by unique package IDs (801, 802, 803, 804) and include details such as the subscription price, the number of shows included, and the maximum number of shows that can be viewed.

Customers can sign up for subscriptions by providing their personal information, such as name and phone number. The customer information is stored in the customer table, and the relationship between customers and their subscribed packages is maintained in the subscriptions table. Each subscription has a start date and an end date, allowing customers to enjoy the benefits of their chosen package for a specific period.

The schedule table holds the information about the shows scheduled at the Afeka Theater. It includes details such as the show ID, show name, genre, show duration, and ticket price. Each show is assigned to a specific hall and has a designated date and time for the performance.

To manage reservations, the theater system utilizes the reservation and 'tickets_in_reservation' tables. Customers can make reservations by providing their phone number, the show they wish to attend, and the number of tickets they want to reserve. The reservation is associated with one or more tickets, which are recorded in the tickets_in_reservation table.

The seat table stores information about the available seats in each hall. Each seat is identified by a unique seat ID and is associated with a specific hall, row, and seat number. This allows the system to track the availability of seats for each show and prevent double bookings.

Overall, the Afeka Theater system provides a platform for managing subscriptions, scheduling shows, making reservations, and tracking ticket sales.

2. Requirements for the current project

- Data Integrity Requirements:
    - The system should enforce data integrity by ensuring that the number of shows left and shows purchased in the subscriptions table are updated accurately and consistently when a reservation is made.
    - The system should maintain the correctness of the number of tickets sold in the schedule table to provide accurate information about ticket availability.
- Reservation Management Requirements:
    - The system should allow the addition of new reservations to the reservation table, including the associated customer's phone number, performance ID, and the number of tickets.
    - The system should automatically create corresponding rows in the tickets_in_reservation table for each reservation, ensuring that the number of tickets matches the reservation.
- Theater Management Requirements:
    - The system should restrict adding more than one row to the theater table, ensuring that there is only one theater record in the system.
- Schedule Management Requirements:
    - The system should allow the addition of new performance entries to the schedule table, including theater ID, show ID, date and time, hall ID, and the initial number of tickets sold.
    - The system will automatically update the number of tickets remaining for the show after placing an reservation.
- Customer Management Requirements:
    - The system should provide the ability to retrieve all reservations made by a specific customer, based on their phone number.
- Ticket Management Requirements:
    - The system should support the insertion of seat IDs into existing records in the tickets_in_reservation table, allowing theater staff to assign specific seats to reservations.
- Subscription Management Requirements:
    - The system will automatically update the number of shows left for the subscription after reservation.

3. Description of 2 user types of the system

- Customers: The customers are individuals who interact with the theater information system to browse shows, make reservations, purchase tickets, and manage their subscriptions.
  Possible customer actions:
  - View the list of available shows and their details, such as show name, genre, duration, and ticket price.
  - Search for shows based on criteria like genre or date.
  - Make reservations for shows by selecting the desired performance, number of tickets, and seating preferences.
  - Browse their reservation history, including past and upcoming reservations.
- Theater Staff: The theater staff members are authorized personnel who have administrative roles within the theater information system. They handle tasks such as managing shows, scheduling performances, and handling reservations.
  Possible theater staff actions:
  - Add or edit shows in the system, including providing details such as show name, genre, duration, and ticket price.
  - Create and manage the schedule of performances, assigning shows to specific theaters, halls, and dates/times.
  - Handle reservations, including viewing and managing reservation details, such as customer information, performance, and the number of tickets.
  - Monitor and update the availability of seats in real-time, ensuring accurate ticket sales and seat allocations.

4. **Description of system entities**

- Theater: The Theater entity represents the physical theater location where shows are performed. It includes information such as the theater ID, theater name, address, and phone number.

- Customer: The Customer entity represents the individuals who purchase tickets for shows at the theater. It includes information such as the customer ID and customer name (first end last).

- Customer Phone: The Customer Phone entity represents the phone numbers associated with customers. It includes information such as the phone number and customer ID.

- Subscription Package: The Subscription Package entity represents the subscription packages offered to customers. It includes information such as the package ID, cost, period in months and number of shows included.

- Subscriptions: The Subscriptions entity represents the customers who have subscriptions to the theater. It includes information such as the customer ID, package ID, start date, end date, number of shows viewed and number of shows left.

- Shows: The Shows entity represents the shows performed at the theater. It includes information such as the show ID, show name, genre, duration and the ticket price.

- Hall: The Hall entity represents the different halls or performance spaces within the theater. It includes information such as the hall ID, theater ID, hall name and seating capacity.

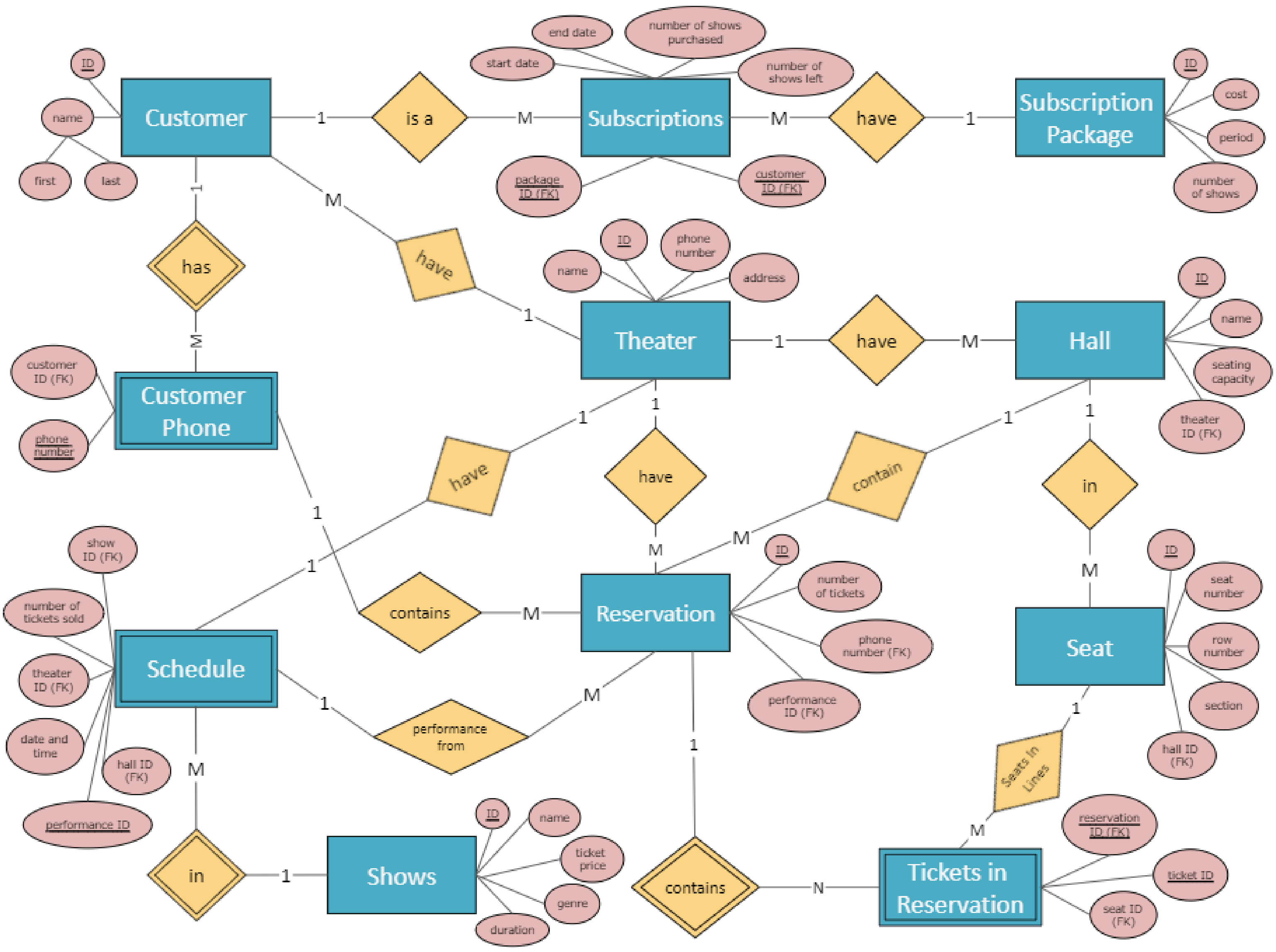
- Schedule: The Schedule entity represents the schedule of shows at the theater. It includes information such as the performance ID, theater ID, show ID, date and time, hall ID and the number of tickets sold.

- Seat: The Seat entity represents each individual seat in the theater. It includes information such as the seat ID, seat number, hall ID, row, and section.

- Reservation: The Reservation entity represents the reservations made by customers for shows at the theater. It includes information such as the reservation ID, customer phone number, performance ID and number of tickets.

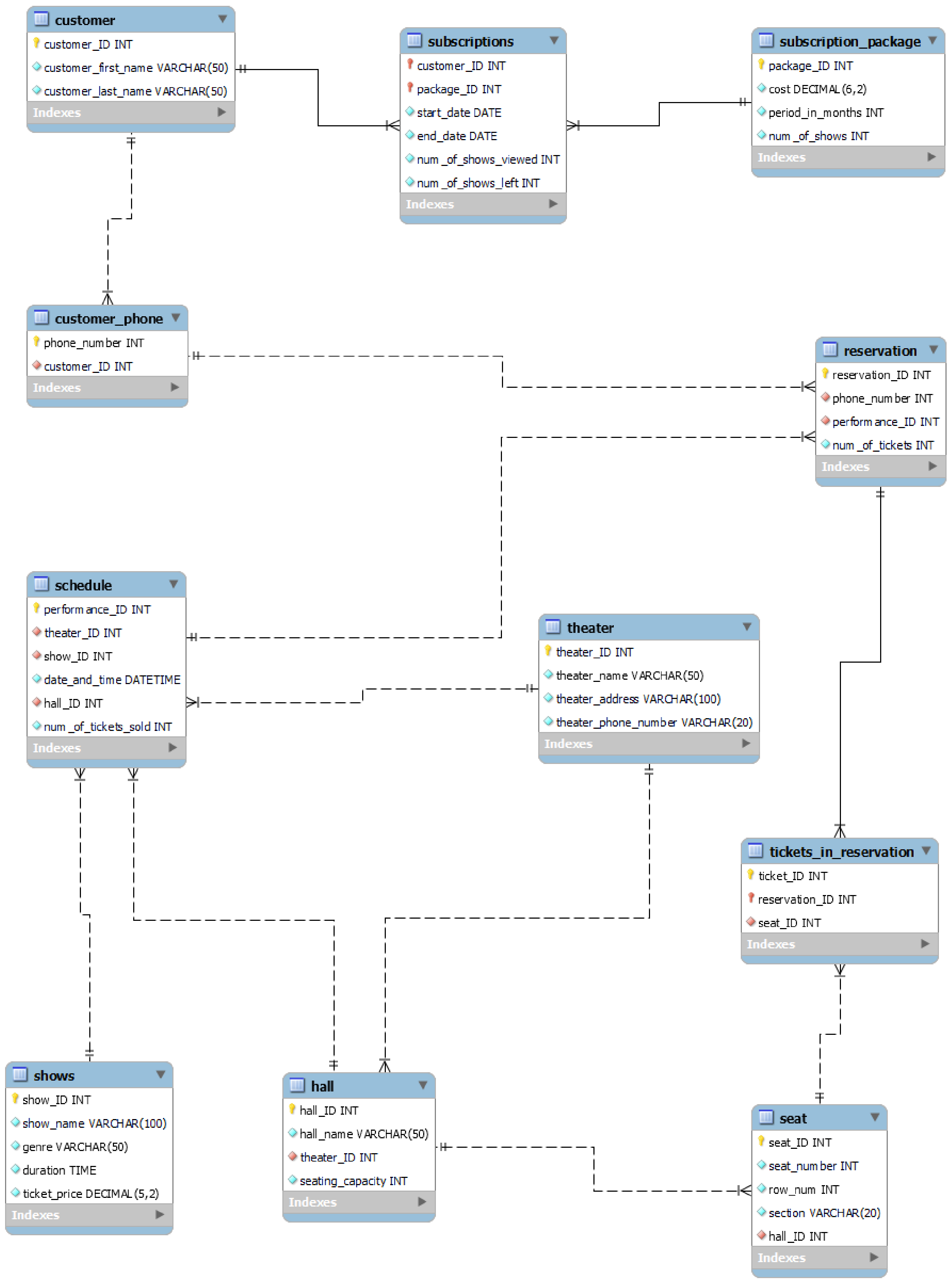
- Tickets in Reservation: The "Tickets in Reservation" entity represents the specific groups of seats that are reserved together as part of a single reservation. It captures the seating arrangement within a reservation that consists of multiple seats grouped together in a specific order. It includes information such as the ticket ID, reservation ID and seat number.

5. **ERD**



ID
name
first
last
Customer
1
is a
M
end date
number of shows purchased
start date
number of shows left
Subscriptions
M
have
1
ID
cost
period
number of shows
Subscription Package
package ID (FK)
customer ID (FK)
1
has
M
M
have
1
ID
phone number
name
address
Theater
1
have
M
ID
name
seating capacity
theater ID (FK)
Hall
customer ID (FK)
phone number
Customer Phone
1
have
1
have
M
1
contain
M
1
in
M
1
1
show ID (FK)
number of tickets sold
theater ID (FK)
date and time
hall ID (FK)
performance ID
Schedule
contains
M
ID
number of tickets
phone number (FK)
performance ID (FK)
Reservation
ID
seat number
row number
section
hall ID (FK)
Seat
1
performance from
M
1
Seats in Lines
M
M
in
1
ID
name
ticket price
genre
duration
Shows
contains
N
reservation ID (FK)
ticket ID
seat ID (FK)
Tickets in Reservation

6. Tables



customer
customer_ID INT
customer_first_name VARCHAR(50)
customer_last_name VARCHAR(50)
Indexes
subscriptions
customer_ID INT
package_ID INT
start_date DATE
end_date DATE
num_of_shows_viewed INT
num_of_shows_left INT
Indexes
subscription_package
package_ID INT
cost DECIMAL(6,2)
period_in_months INT
num_of_shows INT
Indexes
customer_phone
phone_number INT
customer_ID INT
Indexes
reservation
reservation_ID INT
phone_number INT
performance_ID INT
num_of_tickets INT
Indexes
schedule
performance_ID INT
theater_ID INT
show_ID INT
date_and_time DATETIME
hall_ID INT
num_of_tickets_sold INT
Indexes
theater
theater_ID INT
theater_name VARCHAR(50)
theater_address VARCHAR(100)
theater_phone_number VARCHAR(20)
Indexes
tickets_in_reservation
ticket_ID INT
reservation_ID INT
seat_ID INT
Indexes
shows
show_ID INT
show_name VARCHAR(100)
genre VARCHAR(50)
duration TIME
ticket_price DECIMAL(5,2)
Indexes
hall
hall_ID INT
hall_name VARCHAR(50)
theater_ID INT
seating_capacity INT
Indexes
seat
seat_ID INT
seat_number INT
row_num INT
section VARCHAR(20)
hall_ID INT
Indexes

7. **"CREATE TABLE" commands**

```
USE theater_information_system;

CREATE TABLE theater
(
        theater_ID INT PRIMARY KEY,
        theater_name VARCHAR(50) NOT NULL,
        theater_address VARCHAR(100) NOT NULL,
        theater_phone_number VARCHAR(20) NOT NULL
)ENGINE = InnoDB;

CREATE TABLE customer (
          customer_ID INT PRIMARY KEY,
          customer_first_name VARCHAR(50) NOT NULL,
          customer_last_name VARCHAR(50) NOT NULL
)ENGINE = InnoDB;

CREATE TABLE customer_phone (
          phone_number INT PRIMARY KEY,
          customer_ID INT NOT NULL,
          FOREIGN KEY (customer_ID) REFERENCES customer(customer_ID)
)ENGINE = InnoDB;

CREATE TABLE subscription_package (
          package_ID INT PRIMARY KEY,
          cost DECIMAL(6,2) NOT NULL,
          period_in_months INT NOT NULL,
          num_of_shows INT NOT NULL
)ENGINE = InnoDB;

CREATE TABLE subscriptions (
         customer_ID INT NOT NULL,
          package_ID INT NOT NULL,
          start_date DATE NOT NULL,
          end_date DATE NOT NULL,
          num_of_shows_purchased INT NOT NULL,
          num_of_shows_left INT NOT NULL,
          PRIMARY KEY (customer_ID, package_ID),
          FOREIGN KEY (customer_ID) REFERENCES customer(customer_ID),
          FOREIGN KEY (package_ID) REFERENCES subscription_package(package_ID)
)ENGINE = InnoDB;
```

```
CREATE TABLE shows (
            show_ID INT PRIMARY KEY,
              show_name VARCHAR(100) NOT NULL,
               genre VARCHAR(50) NOT NULL,
              duration TIME NOT NULL,
              ticket_price DECIMAL(5,2) NOT NULL
)ENGINE = InnoDB;

CREATE TABLE hall (
            hall_ID INT PRIMARY KEY,
             hall_name VARCHAR(50) NOT NULL,
            theater_ID INT NOT NULL,
            seating_capacity INT NOT NULL,
            FOREIGN KEY (theater_ID) REFERENCES theater(theater_ID)
)ENGINE = InnoDB;

CREATE TABLE schedule (
            performance_ID INT PRIMARY KEY,
            theater_ID INT NOT NULL,
            show_ID INT NOT NULL,
            date_and_time DATETIME NOT NULL,
            hall_ID INT NOT NULL,
            num_of_tickets_sold INT NOT NULL,
            FOREIGN KEY (theater_ID) REFERENCES theater(theater_ID),
            FOREIGN KEY (show_ID) REFERENCES shows(show_ID),
   FOREIGN KEY (hall_ID) REFERENCES hall(hall_ID)
);

CREATE TABLE seat (
            seat_ID INT PRIMARY KEY,
            seat_number INT NOT NULL,
            row_num INT NOT NULL,
             section VARCHAR(20) NOT NULL,
            hall_ID INT NOT NULL,
            FOREIGN KEY (hall_ID) REFERENCES hall(hall_ID)
)ENGINE = InnoDB;

CREATE TABLE reservation (
             reservation_ID INT PRIMARY KEY,
            phone_number INT NOT NULL,
            performance_ID INT NOT NULL,
            num_of_tickets INT NOT NULL,
            FOREIGN KEY (phone_number) REFERENCES customer_phone(phone_number),
            FOREIGN KEY (performance_ID) REFERENCES schedule(performance_ID)
)ENGINE = InnoDB;

CREATE TABLE tickets_in_reservation (
            ticket_ID INT NOT NULL,
            reservation_ID INT NOT NULL,
            seat_ID INT,
            PRIMARY KEY (ticket_ID, reservation_ID),
            FOREIGN KEY (reservation_ID) REFERENCES reservation(reservation_ID),
            FOREIGN KEY (seat_ID) REFERENCES seat(seat_ID)
);
```

8. SELECT commands

```
USE theater_information_system;

/*-----------------------------------------------------*/

# S1
# Select the show names and genres from the shows table sorted by genre in ascending order:

SELECT show_name AS 'Show Name', genre AS 'Genre'
FROM shows
ORDER BY genre ASC;

/*-----------------------------------------------------*/

# S2
# Select the shows where the number of tickets sold is greater than 1:

SELECT performance_ID AS 'Performance ID',  show_name AS 'Show Name',
num_of_tickets_sold AS 'Number of Tickets Sold'
FROM shows
JOIN schedule ON shows.show_ID = schedule.show_ID
WHERE schedule.num_of_tickets_sold > 1;

/*-----------------------------------------------------*/

# S3
# Select the top subscription packages with the highest number of subscriptions:

SELECT subscription_package.package_ID AS 'Package ID', COUNT(*) AS
'Number of Subscriptions'
FROM subscription_package
JOIN subscriptions ON subscription_package.package_ID =
subscriptions.package_ID
GROUP BY subscription_package.package_ID
ORDER BY COUNT(*) DESC
LIMIT 1;

/*-----------------------------------------------------*/
```

```
# S4
# Select the subscriptions names and the total number of tickets they have purchased:

SELECT CONCAT(customer.customer_first_name, ' ',
customer.customer_last_name) AS 'Customer Name',
COALESCE(SUM(reservation.num_of_tickets), 0) AS 'Total Tickets Purchased'
FROM customer
LEFT JOIN customer_phone ON customer.customer_ID =
customer_phone.customer_ID
LEFT JOIN reservation ON customer_phone.phone_number =
reservation.phone_number
LEFT JOIN subscriptions ON customer.customer_ID = subscriptions.customer_ID
WHERE CURDATE() BETWEEN subscriptions.start_date AND
subscriptions.end_date
GROUP BY CONCAT(customer.customer_first_name, ' ',
customer.customer_last_name)
ORDER BY COALESCE(SUM(reservation.num_of_tickets), 0) DESC;

/*-----------------------------------------------------*/

# S5
# Select the total number of people who visited each hall:

SELECT hall.hall_ID, hall_name, SUM(IFNULL(reservation.num_of_tickets, 0)) AS
'Total Number of Visitors'
FROM hall
LEFT JOIN schedule ON hall.hall_ID = schedule.hall_ID
LEFT JOIN reservation ON schedule.performance_ID = reservation.performance_ID
GROUP BY hall.hall_ID, hall.hall_name;

/*-----------------------------------------------------*/

# S6
# Select customers who have more than one phone number:

SELECT customer.customer_ID AS 'ID', customer_first_name AS 'First Name',
customer_last_name AS 'Last Name', phone_number AS 'Phone Number'
FROM customer
JOIN customer_phone phone ON customer.customer_ID = phone.customer_ID
WHERE customer.customer_ID IN (
   SELECT customer_ID
   FROM customer_phone
   GROUP BY customer_ID
   HAVING COUNT(*) > 1
);

/*-----------------------------------------------------*/

# S7
# Select all the customers who have made reservations for 'The Broken Algorithm'
shows:

SELECT CONCAT(customer_first_name, ' ', customer_last_name) AS 'Customer
Name', date_and_time AS 'Date of the Performance'
FROM customer
JOIN customer_phone ON customer.customer_ID = customer_phone.customer_ID
JOIN reservation ON customer_phone.phone_number = reservation.phone_number
JOIN schedule ON reservation.performance_ID = schedule.performance_ID
WHERE schedule.show_ID = 304
ORDER BY date_and_time ASC;

/*-----------------------------------------------------*/

# S8
# Select the names of customers who have an active subscription package:
SELECT CONCAT(customer_first_name, ' ', customer_last_name) AS 'Customer
Name', start_date AS 'Start Date', end_date AS 'End Date'
FROM customer
JOIN subscriptions ON customer.customer_ID = subscriptions.customer_ID
WHERE CURDATE() BETWEEN start_date AND end_date;

/*-----------------------------------------------------*/

# S9
# Select the top 5 customers who have made the most reservations:

SELECT CONCAT(customer_first_name, ' ', customer_last_name) AS 'Customer
Name', COUNT(reservation.reservation_ID) AS 'Reservation Count'
FROM customer
JOIN customer_phone ON customer.customer_ID = customer_phone.customer_ID
JOIN reservation ON customer_phone.phone_number = reservation.phone_number
GROUP BY customer.customer_ID
ORDER BY COUNT(reservation.reservation_ID) DESC
LIMIT 5;

/*-----------------------------------------------------*/
```

```
# S10
# Select the seats that are currently available for a specific performance:

SELECT seat.seat_ID AS 'Seat ID', seat.seat_number AS 'Seat Number',
seat.row_num AS 'Row Number', seat.section AS 'Section'
FROM seat
JOIN hall ON seat.hall_ID = hall.hall_ID
WHERE seat.hall_ID = 103
  AND seat.seat_ID NOT IN (
    SELECT tickets_in_reservation.seat_ID
    FROM tickets_in_reservation
    JOIN reservation ON tickets_in_reservation.reservation_ID =
reservation.reservation_ID
    WHERE reservation.performance_ID = 410
  );
```

9. Triggers

```
USE theater_information_system;

/*-------------------------------------------------------*/

# T1
# Trigger to update the number of shows left in the subscriptions table when a reservation is
made:

DELIMITER $$
CREATE TRIGGER update_shows_left
AFTER INSERT
ON reservation
FOR EACH ROW
BEGIN
  UPDATE subscriptions
  SET num_of_shows_left = num_of_shows_left - NEW.num_of_tickets
  WHERE customer_ID = (SELECT customer_ID FROM customer_phone WHERE
phone_number = NEW.phone_number);
END $$
DELIMITER ;

/*-------------------------------------------------------*/

# T2
# Trigger to update the number of shows purchased in the subscriptions table when a
reservation is made:

DELIMITER $$
CREATE TRIGGER update_shows_purchased
AFTER INSERT
ON reservation
FOR EACH ROW
BEGIN
  UPDATE subscriptions
  SET num_of_shows_purchased = num_of_shows_purchased + NEW.num_of_tickets
  WHERE customer_ID = (SELECT customer_ID FROM customer_phone WHERE
phone_number = NEW.phone_number);
END $$
DELIMITER ;

/*-------------------------------------------------------*/

# T3
# Trigger that updates the number of tickets sold in the schedule table when a reservation is
made:
```

```
DELIMITER $$
CREATE TRIGGER update_num_of_tickets_sold
AFTER INSERT
ON reservation
FOR EACH ROW
BEGIN
  UPDATE schedule
  SET num_of_tickets_sold = num_of_tickets_sold + NEW.num_of_tickets
  WHERE performance_ID = NEW.performance_ID;
END $$
DELIMITER ;

/*------------------------------------------------------*/

# T4
# Trigger to automatically create new rows in the tickets_in_reservation table when a
reservation is made:
# (there is a procedure to update the specific seats in tickets)

DELIMITER $$
CREATE TRIGGER create_tickets_in_reservation
AFTER INSERT
ON reservation
FOR EACH ROW
BEGIN
  DECLARE i INT DEFAULT 1;
  DECLARE next_ticket_ID INT;

  SET next_ticket_ID = (SELECT COALESCE(MAX(ticket_ID), 0) + 1 FROM
tickets_in_reservation);

  WHILE i <= NEW.num_of_tickets DO
    INSERT INTO tickets_in_reservation (ticket_ID, reservation_ID, seat_ID)
    VALUES (next_ticket_ID, NEW.reservation_ID, NULL);
    SET i = i + 1;
    SET next_ticket_ID = next_ticket_ID + 1;
  END WHILE;
END $$
DELIMITER ;
```

10. Stored Procedures

```
USE theater_information_system;

/*------------------------------------------------------*/

# P1
# Procedure that not allow adding more than one row to the theater table:

DELIMITER //
CREATE PROCEDURE insert_theater_row(
   IN p_theater_ID INT,
   IN p_theater_name VARCHAR(50),
   IN p_theater_address VARCHAR(100),
   IN p_theater_phone_number VARCHAR(20)
)
BEGIN
   DECLARE theater_count INT;
   SELECT COUNT(*) INTO theater_count
   FROM theater;

   IF theater_count = 1 THEN
      SELECT 'Insert operation not allowed. Theater table already contains a row.' AS message;
   END IF;
END //
DELIMITER ;

CALL insert_theater_row (2,'Afeka Theater 2','Benei Ephraim St. 218 , Tel Aviv','03-7688600');

/*------------------------------------------------------*/

# P2
# Procedure that allows to add a new perfornamce entry to the schedule table:

DELIMITER //
CREATE PROCEDURE add_to_schedule(
   IN performance_ID INT,
   IN theater_ID INT,
   IN show_ID INT,
   IN date_and_time DATETIME,
   IN hall_ID INT,
   IN num_of_tickets_sold INT
)
BEGIN
   INSERT INTO schedule (performance_ID, theater_ID, show_ID, date_and_time, hall_ID,
num_of_tickets_sold)
```

```
   VALUES (performance_ID, theater_ID, show_ID, date_and_time, hall_ID,
num_of_tickets_sold);
END //
DELIMITER ;

CALL add_to_schedule(423,1,309,'2023-06-17 21:30',102,0); #added

/*-----------------------------------------------------------*/

# P3
# Procedure that allows to add a new reservation to the reservation table:

DELIMITER //
CREATE PROCEDURE add_new_reservation(
   IN reservation_ID INT,
   IN phone_number INT,
   IN performance_ID INT,
   IN num_of_tickets INT
)
BEGIN
   INSERT INTO reservation (reservation_ID, phone_number, performance_ID,
num_of_tickets)
   VALUES (reservation_ID, phone_number, performance_ID, num_of_tickets);
END //
DELIMITER ;

CALL add_new_reservation(505,0522014447,406,2); #added

/*-----------------------------------------------------------*/

# P4
# Procedure that retrieves all the reservations made by a specific customer:

DELIMITER //
CREATE PROCEDURE get_customer_reservations(IN customer_phone_number INT)
BEGIN
   SELECT reservation.reservation_ID, shows.show_name, schedule.date_and_time,
theater.theater_name, hall.hall_name
   FROM reservation
   INNER JOIN schedule ON reservation.performance_ID = schedule.performance_ID
   INNER JOIN shows ON schedule.show_ID = shows.show_ID
   INNER JOIN hall ON schedule.hall_ID = hall.hall_ID
   INNER JOIN theater ON schedule.theater_ID = theater.theater_ID
   WHERE reservation.phone_number = customer_phone_number;
END //
DELIMITER ;
```

```
CALL get_customer_reservations(0502847715);

/*------------------------------------------------------*/

# P5
# Procedure that retrieves the available seats for a specific performance in a particular hall:

DELIMITER //
CREATE PROCEDURE get_available_seats(
  IN performance_ID INT,
  IN hall_ID INT
)
BEGIN
  SELECT seat.seat_ID, seat.seat_number, seat.row_num, seat.section
  FROM seat
  LEFT JOIN tickets_in_reservation ON seat.seat_ID = tickets_in_reservation.seat_ID
  WHERE seat.hall_ID = hall_ID
  AND (tickets_in_reservation.ticket_ID IS NULL OR tickets_in_reservation.reservation_ID
NOT IN (
    SELECT reservation_ID
    FROM reservation
    WHERE performance_ID = performance_ID
  ));
END //
DELIMITER ;

CALL get_available_seats(414, 102);

/*------------------------------------------------------*/

# P6
# Procedure that retrieves reservations for a specific performance:

DELIMITER //
CREATE PROCEDURE get_reservations_by_performance (IN performanceID INT)
BEGIN
  SELECT *
  FROM reservation
  WHERE performance_ID = performanceID;
END //
DELIMITER ;

CALL get_reservations_by_performance(401);

/*------------------------------------------------------*/
```

```
# P7
# Procedure that inserts a seat_ID into an existing record from the tickets_in_reservation table:
# (there is a trigger to insert automatically tickets without seats numbers)

DELIMITER //
CREATE PROCEDURE insert_seat_ID(
            IN p_ticket_ID INT,
   IN p_seat_ID INT
)
BEGIN
   UPDATE tickets_in_reservation
   SET seat_ID = p_seat_ID
   WHERE ticket_ID = p_ticket_ID;
END //
DELIMITER ;

CALL insert_seat_ID(610, 235);
CALL insert_seat_ID(611, 236);
```

11. INSERT / UPDATE / DELETE commands

```
USE theater_information_system;

/*-------------------------------------------------------*/

# 1
# INSERT a new show to the "shows" table:

INSERT INTO shows VALUES
(313, 'The Horror of Physics', 'Drama', '02:30:00', 180.00);

/*-------------------------------------------------------*/

# 2
# DELETE shows from the "schedule" table whose date has already passed:

DELETE FROM schedule
WHERE date_and_time < NOW();

/*-------------------------------------------------------*/

# 3
# UPDATE all prices of subscription packages in the "subscription_package" table by
increasing them by 50%

UPDATE subscription_package
SET cost = cost * 1.5;
```