

Project phase A (team 1)

Μέλη Ομάδας

CSD4549 Ευθυμίου Γεράσιμος

CSD4676 Μακρογιάννης Δημήτριος

CSD4559 Βιδάλης Δημήτριος

Εννοιολογική Μοντελοποίηση της βάσης

Εισαγωγή

Το παρόν project αφορά τον σχεδιασμό και την υλοποίηση μιας βάσης δεδομένων για τη διαχείριση εκδηλώσεων. Το σύστημα καλύπτει την αποθήκευση βασικών πληροφοριών για εκδηλώσεις, πελάτες, εισιτήρια και κρατήσεις, με σκοπό την παροχή ολοκληρωμένων λειτουργιών όπως εγγραφή νέων πελατών, δημιουργία εκδηλώσεων, αναζήτηση διαθέσιμων θέσεων, κράτηση εισιτηρίων και διαχείριση ακυρώσεων. Επιπλέον, περιλαμβάνονται διαδικασίες που υποστηρίζουν την ανάλυση δεδομένων, όπως υπολογισμός εσόδων, δημοφιλέστερες εκδηλώσεις και προβολή κρατήσεων. Η ανάπτυξη περιλαμβάνει εννοιολογική και σχεσιακή μοντελοποίηση, καθώς και κανονικοποίηση σε τρίτη κανονική μορφή, ώστε να διασφαλιστεί η ακεραιότητα και η αποδοτική διαχείριση των δεδομένων.

Διάγραμμα Οντοτήτων-Σχέσεων (E-R)

Το ER διάγραμμα απεικονίζει τις οντότητες, τις σχέσεις μεταξύ τους και τα χαρακτηριστικά τους.

Οι βασικές οντότητες είναι:

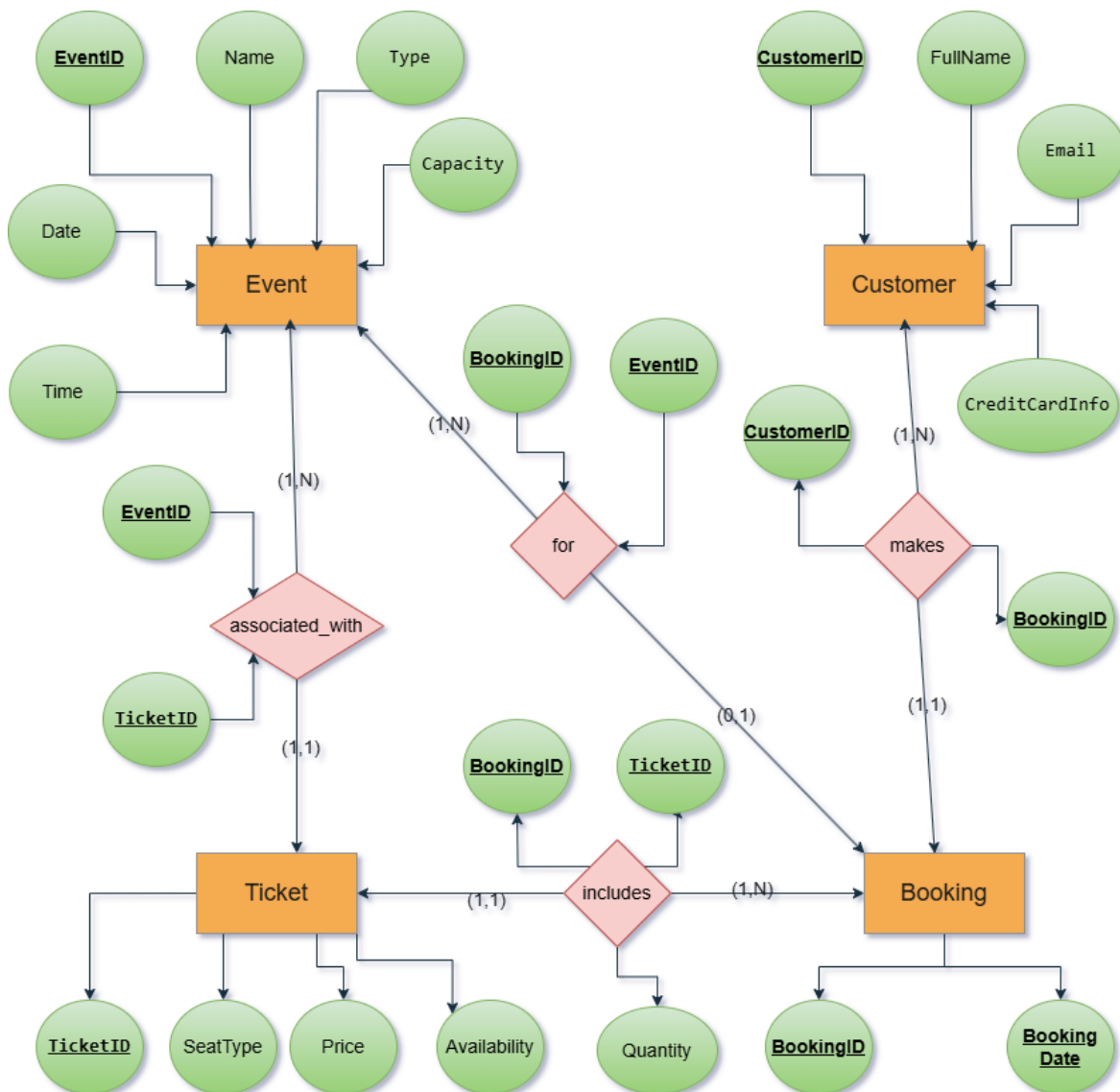
Event (Εκδήλωση)

Customer (Πελάτης)

Ticket (Εισιτήριο)

Booking (Κράτηση)

Διάγραμμα



Γνωρίσματα Οντοτήτων, Σχέσεων

Οντότητες:

Customer

CustomerID (Primary Key, INT),

FullName (VARCHAR),

Email (VARCHAR),

CreditCardInfo (VARCHAR)

Event

EventID (Primary Key, INT),

Name (VARCHAR),

Date (DATE),

Time (TIME),

Type (VARCHAR),

Capacity (INT)

Ticket

TicketID (Primary Key, INT),

SeatType (VARCHAR),

Price (DECIMAL),

Availability (INT)

Booking

BookingID (Primary Key, INT) ,

CustomerID (Foreign Key, INT),

EventID (Foreign Key, INT),

BookingDate (DATE)

Σχέσεις:

1) makes (Customer → Booking)

CustomerID → BookingID

Πληθικότητα: (1, N)

Ένας πελάτης μπορεί να κάνει πολλές κρατήσεις.

Πληθικότητα: (1, 1)

Κάθε κράτηση ανήκει σε έναν πελάτη.

2) for (Booking \rightarrow Event)

EventID \rightarrow BookingID

Πληθικότητα: (1,N)

Μια εκδήλωση μπορεί να έχει πολλές κρατήσεις.

Πληθικότητα: (0, 1)

Κάθε κράτηση αφορά μία μόνο εκδήλωση.

3) associated_with (Ticket \rightarrow Event)

EventID \rightarrow TicketID

Πληθικότητα: (1,N)

Κάθε εκδήλωση έχει πολλούς τύπους εισιτηρίων (π.χ., VIP, γενική είσοδος).

Πληθικότητα: (1, 1)

Κάθε τύπος εισιτηρίου σχετίζεται με μία συγκεκριμένη εκδήλωση.

4) includes (Booking \rightarrow Ticket)

Για να συνδέσουμε τις κρατήσεις με τα εισιτήρια που αγοράστηκαν, δημιουργούμε τη σχέση includes, η οποία περιλαμβάνει τα παρακάτω:

Γνώρισμα

BookingID (Foreign Key, από Booking)

TicketID (Foreign Key, από Ticket)

Quantity (INT, πλήθος εισιτηρίων που περιλαμβάνονται στην κράτηση)

Συνδέει κάθε κράτηση με τα εισιτήρια που έχουν αγοραστεί. Το γνώρισμα Quantity καταγράφει πόσα εισιτήρια στην κράτηση.

Πληθικότητα: (1,N)

Μια κράτηση μπορεί να περιλαμβάνει πολλούς τύπους εισιτηρίων (π.χ., VIP και γενική είσοδος).

Πληθικότητα: (1, 1)

Ένα εισιτήριο μπορεί να εμφανίζεται σε μία κράτηση.

Επεξηγήσεις για Μη Προφανή Γνωρίσματα και Σχέσεις

CreditCardInfo: Χρησιμοποιείται για την ολοκλήρωση πληρωμών και αποθηκεύεται σε ασφαλή μορφή.

Availability (Ticket): Δείχνει τον αριθμό διαθέσιμων εισιτηρίων.

AmountPaid (Booking): Το συνολικό ποσό που πληρώθηκε για την κράτηση.

NumTickets(Booking): Το συνολικό πλήθος των εισιτηρίων που έχουν κρατηθεί. Αποθηκεύεται στο Booking

Μετάφραση στο Σχεσιακό Μοντέλο

Πίνακες:

Event [**EventID**, Name, Date, Time, Type, Capacity]

Customers [**CustomerID**, FullName, Email, CreditCard]

Tickets [**TicketID**, EventID, Type, Price, Availability]

Booking [**BookingID**, CustomerID, EventID, NumTickets, BookingDate, Payment]

Makes [BookingId, CustomerId]

Includes [BookingId, TicketID, Quantity]

Associated_with [EventID, TicketID]

Εντολές γλώσσας ορισμού για τις σχέσεις (DDL)

```
CREATE TABLE Event (  
    EventID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Type VARCHAR(50),  
    Capacity INT,  
    Date DATE,  
    Time TIME  
);
```

```
CREATE TABLE Customer (  
    CustomerID INT PRIMARY KEY,  
    FullName VARCHAR(100),  
    Email VARCHAR(100),  
    CreditCardInfo VARCHAR(50)
```

);

```
CREATE TABLE Ticket (  
    TicketID INT PRIMARY KEY,  
    EventID INT,  
    SeatType VARCHAR(50),  
    Price DECIMAL(10, 2),  
    Availability BOOLEAN,  
    FOREIGN KEY (EventID) REFERENCES Event(EventID)  
);
```

```
CREATE TABLE Booking (  
    BookingID INT PRIMARY KEY,  
    BookingDate DATE,  
    Payment DECIMAL(10, 2),  
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),  
    FOREIGN KEY (EventID) REFERENCES Event(EventID)  
);
```

```
CREATE TABLE BookingTicket (  
    BookingID INT,  
    TicketID INT,  
    Quantity INT,  
    PRIMARY KEY (BookingID, TicketID),  
    FOREIGN KEY (BookingID) REFERENCES Booking(BookingID),  
    FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)  
);
```

Οι πίνακες του συστήματος είναι οι εξής:

- **Event (Εκδήλωση):** Περιέχει λεπτομέρειες για τις εκδηλώσεις, όπως το όνομα, τον τύπο (π.χ. συναυλία, θεατρική παράσταση), τη χωρητικότητα, την ημερομηνία και την ώρα.

- **Customer (Πελάτης):** Περιέχει τα στοιχεία των πελατών, όπως το όνομα, το email και τις πληροφορίες πιστωτικής κάρτας.
- **Ticket (Εισιτήριο):** Περιέχει τα δεδομένα για τα εισιτήρια, όπως τον τύπο θέσης (π.χ. VIP ή γενική είσοδος), την τιμή, τη διαθεσιμότητα και την ποσότητα για κάθε εκδήλωση.
- **Booking (Κράτηση):** Περιέχει τις πληροφορίες για τις κρατήσεις, όπως την ημερομηνία της κράτησης, το ποσό πληρωμής και την εκδήλωση για την οποία έγινε η κράτηση.
- **BookingTicket (Κράτηση-Εισιτήριο):** Είναι ένας πίνακας σύνδεσης που συνδέει τις κρατήσεις με τα εισιτήρια. Κρατάει την ποσότητα των εισιτηρίων για κάθε κράτηση.

Σχέσεις:

- Κάθε **Εκδήλωση** μπορεί να έχει πολλά **Εισιτήρια**.
- Κάθε **Πελάτης** μπορεί να κάνει πολλές **Κρατήσεις**.
- Κάθε **Κράτηση** μπορεί να περιλαμβάνει πολλά **Εισιτήρια**, και ένα **Εισιτήριο** μπορεί να ανήκει σε πολλές **Κρατήσεις** (αυτό γίνεται μέσω του πίνακα **BookingTicket**).

Για το συγκεκριμένο σύστημα που περιγράφουμε, οι περιορισμοί ακεραιότητας και οι συναρτησιακές εξαρτήσεις είναι κρίσιμες για την αποφυγή ανακολουθιών στα δεδομένα και τη διασφάλιση της σωστής λειτουργίας της βάσης δεδομένων. Παρακάτω ακολουθούν οι αντίστοιχοι περιορισμοί και οι εξαρτήσεις.

Περιορισμοί Ακεραιότητας

Περιορισμός Πρωτεύοντος Κλειδιού (Primary Key Constraint):

Event: Το EventID είναι το πρωτεύον κλειδί για τον πίνακα Event, που διασφαλίζει ότι κάθε εκδήλωση έχει μοναδικό αναγνωριστικό.

Customer: Το CustomerID είναι το πρωτεύον κλειδί για τον πίνακα Customer, διασφαλίζοντας ότι κάθε πελάτης είναι μοναδικός.

Ticket: Το TicketID είναι το πρωτεύον κλειδί για τον πίνακα Ticket, διασφαλίζοντας ότι κάθε εισιτήριο είναι μοναδικό.

Booking: Το BookingID είναι το πρωτεύον κλειδί για τον πίνακα Booking, διασφαλίζοντας ότι κάθε κράτηση έχει μοναδικό αναγνωριστικό.

BookingTicket: Το ζεύγος BookingID και TicketID είναι το σύνθετο πρωτεύον κλειδί για τον πίνακα BookingTicket, εξασφαλίζοντας ότι κάθε κράτηση για εισιτήριο είναι μοναδική.

Περιορισμός Ξένου Κλειδιού (Foreign Key Constraint):

Event: Το EventID στον πίνακα Ticket αναφέρεται στο πρωτεύον κλειδί του πίνακα Event, διασφαλίζοντας ότι κάθε εισιτήριο σχετίζεται με μια υπάρχουσα εκδήλωση.

Customer: Το CustomerID στον πίνακα Booking αναφέρεται στο πρωτεύον κλειδί του πίνακα Customer, διασφαλίζοντας ότι κάθε κράτηση ανήκει σε έναν πελάτη.

Event: Το EventID στον πίνακα Booking αναφέρεται στο πρωτεύον κλειδί του πίνακα Event, διασφαλίζοντας ότι κάθε κράτηση ανήκει σε μια εκδήλωση.

Booking: Το BookingID στον πίνακα BookingTicket αναφέρεται στο πρωτεύον κλειδί του πίνακα Booking, διασφαλίζοντας ότι κάθε κράτηση εισιτηρίου ανήκει σε μια υπάρχουσα κράτηση.

Ticket: Το TicketID στον πίνακα BookingTicket αναφέρεται στο πρωτεύον κλειδί του πίνακα Ticket, διασφαλίζοντας ότι κάθε κράτηση εισιτηρίου ανήκει σε ένα υπάρχον εισιτήριο.

Περιορισμός Μοναδικότητας (Unique Constraint):

Email: Στον πίνακα Customer, το πεδίο Email πρέπει να είναι μοναδικό για κάθε πελάτη. Δεν μπορεί να υπάρχουν δύο πελάτες με το ίδιο email.

Περιορισμός Μη Κενής Τιμής (NOT NULL Constraint):

Τα πεδία που είναι απαραίτητα για τη δημιουργία μιας εγγραφής (όπως EventID, CustomerID, TicketID, κ.λπ.) δεν πρέπει να είναι κενά.

Για παράδειγμα, το πεδίο Name στον πίνακα Event ή το πεδίο FullName στον πίνακα Customer δεν πρέπει να είναι κενά.

Περιορισμός Ελάχιστης και Μέγιστης Τιμής (Check Constraint):

Στον πίνακα Ticket, το πεδίο Price μπορεί να έχει περιορισμό ώστε η τιμή του εισιτηρίου να είναι μεγαλύτερη από το 0 (π.χ., CHECK (Price > 0)).

Στον πίνακα Ticket, το πεδίο Availability μπορεί να είναι boolean, που περιορίζει τις τιμές του σε TRUE ή FALSE.

Συναρτησιακές Εξαρτήσεις

Event:

Το EventID εξαρτάται πλήρως από το Event (πρωτεύον κλειδί).

Τα πεδία Name, Type, Capacity, Date, και Time εξαρτώνται πλήρως από το EventID. Δεν υπάρχει κάποια εξάρτηση μεταξύ τους.

Customer:

Το CustomerID εξαρτάται πλήρως από τον πελάτη.

Τα πεδία FullName, Email, και CreditCardInfo εξαρτώνται πλήρως από το CustomerID.

Ticket:

Το TicketID εξαρτάται πλήρως από το εισιτήριο.

Τα πεδία SeatType, Price, Availability, και Quantity εξαρτώνται πλήρως από το TicketID.

Booking:

Το BookingID εξαρτάται πλήρως από την κράτηση.

Τα πεδία CustomerID, EventID, BookingDate, και Payment εξαρτώνται πλήρως από το BookingID.

BookingTicket:

Το ζεύγος BookingID και TicketID εξαρτάται πλήρως από την εγγραφή στον πίνακα. Το πεδίο Quantity εξαρτάται από αυτό το ζεύγος.

Οι περιορισμοί ακεραιότητας αφορούν τους κανόνες που πρέπει να τηρούνται για να εξασφαλιστεί η ακεραιότητα των δεδομένων στη βάση, ενώ τα κλειδιά αφορούν την μοναδικότητα και τη σύνδεση των δεδομένων μεταξύ των πινάκων. Ας δούμε λοιπόν τους **περιορισμούς ακεραιότητας** για τη συγκεκριμένη βάση δεδομένων, σύμφωνα με τις συναρτησιακές εξαρτήσεις και το μοντέλο.

Περιορισμοί Ακεραιότητας

Ακεραιότητα Οντοτήτων (Entity Integrity):

Περιορισμός: Κάθε πίνακας πρέπει να έχει πρωτεύον κλειδί, και καμία εγγραφή δεν μπορεί να έχει τιμή NULL στο πρωτεύον κλειδί.

Εφαρμογή:

Στους πίνακες Event, Customer, Ticket, Booking και BookingTicket, το πρωτεύον κλειδί δεν μπορεί να είναι NULL.

Επομένως, πεδία όπως το EventID, CustomerID, TicketID, BookingID, και το σύνθετο κλειδί (BookingID, TicketID) δεν επιτρέπεται να είναι NULL.

Ακεραιότητα Σχέσεων (Referential Integrity):

Περιορισμός: Τα εξωτερικά κλειδιά (foreign keys) πρέπει να αντιστοιχούν σε εγγραφές που υπάρχουν στους αντίστοιχους πίνακες.

Εφαρμογή:

Στον πίνακα Booking, το πεδίο CustomerID πρέπει να αντιστοιχεί σε μια υπάρχουσα εγγραφή στον πίνακα Customer.

Στον πίνακα Booking, το πεδίο EventID πρέπει να αντιστοιχεί σε μια υπάρχουσα εγγραφή στον πίνακα Event.

Στον πίνακα BookingTicket, το πεδίο TicketID πρέπει να αντιστοιχεί σε μια υπάρχουσα εγγραφή στον πίνακα Ticket.

Οποιαδήποτε εγγραφή σε αυτούς τους πίνακες που παραβιάζει αυτήν την εξάρτηση (π.χ. εισαγωγή μιας κράτησης με CustomerID που δεν υπάρχει στον πίνακα Customer) θα απορρίπτεται.

Ακεραιότητα Περιορισμών (Domain Integrity):

Περιορισμός: Τα πεδία πρέπει να περιέχουν δεδομένα που ανήκουν σε συγκεκριμένο πεδίο τιμών (domain).

Εφαρμογή:

Τα πεδία όπως το Email στον πίνακα Customer πρέπει να ακολουθούν το σωστό μορφότυπο ενός email (π.χ. user@example.com).

Τα πεδία όπως το Price στον πίνακα Ticket πρέπει να περιέχουν θετικές τιμές (π.χ. Price > 0).

Το πεδίο Quantity στον πίνακα BookingTicket πρέπει να περιέχει μη αρνητικές ακέραιες τιμές (π.χ. Quantity >= 0).

Το πεδίο Availability στον πίνακα Ticket πρέπει να είναι είτε Available είτε Unavailable.

Ακεραιότητα Αξιοπιστίας (Consistency Integrity):

Περιορισμός: Οι κανόνες της επιχείρησης πρέπει να τηρούνται για να διασφαλιστεί ότι τα δεδομένα παραμένουν συνεπή.

Εφαρμογή:

Ο αριθμός των εισιτηρίων που ζητούνται στην κράτηση (στο πεδίο Quantity στον πίνακα BookingTicket) δεν μπορεί να υπερβαίνει τον αριθμό των διαθέσιμων εισιτηρίων (στο πεδίο Availability στον πίνακα Ticket).

Το ποσό πληρωμής για μια κράτηση πρέπει να είναι τουλάχιστον το συνολικό ποσό για τα εισιτήρια που έχουν αγοραστεί.

Εάν μια εκδήλωση ακυρωθεί, όλες οι σχετικές κρατήσεις πρέπει να ακυρωθούν και οι πληρωμές να επιστραφούν, διατηρώντας τη συνεπή κατάσταση των δεδομένων.

Ακεραιότητα Ενημέρωσης (Update Integrity):

Περιορισμός: Οι αλλαγές σε δεδομένα δεν πρέπει να προκαλούν ανεπιθύμητες συνέπειες ή διαταραχές στις σχέσεις.

Εφαρμογή:

Εάν κάποιος πελάτης αλλάξει τα στοιχεία του (π.χ. Email ή CreditCardInfo στον πίνακα Customer), οι σχετικές εγγραφές στις κρατήσεις ή στις πληρωμές θα πρέπει να ενημερώνονται αυτόματα χωρίς να χάνονται δεδομένα.

Αν μια εκδήλωση αλλάξει (π.χ. ημερομηνία ή ώρα), οι κρατήσεις για αυτή την εκδήλωση πρέπει να ενημερωθούν για να αντικατοπτρίζουν τη νέα ημερομηνία/ώρα.

Ακεραιότητα Διαγραφής (Deletion Integrity):

Περιορισμός: Η διαγραφή μιας εγγραφής δεν πρέπει να οδηγεί σε «ορφανές» εγγραφές που δεν συνδέονται με άλλες.

Εφαρμογή:

Αν διαγραφεί μια εκδήλωση από τον πίνακα Event, τότε όλες οι σχετικές κρατήσεις στον πίνακα Booking και τα εισιτήρια στον πίνακα BookingTicket πρέπει να διαγραφούν ή να ενημερωθούν με τρόπο ώστε να μην υπάρχουν «ορφανές» εγγραφές.

Αν διαγραφεί ένας πελάτης από τον πίνακα Customer, όλες οι κρατήσεις που σχετίζονται με αυτόν τον πελάτη στον πίνακα Booking θα πρέπει να διαγραφούν ή να ενημερωθούν αναλόγως.

Ακολουθεί μια περιγραφή των πιθανών ερωτήσεων που μπορούμε να θέσουμε στη βάση δεδομένων, μαζί με παραδείγματα εντολών SQL που τις υλοποιούν:

1. Κατάσταση διαθέσιμων και κρατημένων θέσεων ανά εκδήλωση

Θέλουμε να δούμε πόσες θέσεις είναι διαθέσιμες και κρατημένες για κάθε εκδήλωση.

```
SELECT
    e.Name AS EventName,
    t.SeatType,
    t.Availability AS TotalSeats,
    SUM(bt.Quantity) AS ReservedSeats,
    (t.Availability - SUM(bt.Quantity)) AS AvailableSeats
FROM
    Event e
JOIN
    Ticket t ON e.EventID = t.EventID
LEFT JOIN
    BookingTicket bt ON t.TicketID = bt.TicketID
GROUP BY
```

e.Name, t.SeatType, t.Availability;

2. Έσοδα από πωλήσεις ανά εκδήλωση

Υπολογίζουμε τα συνολικά έσοδα από πωλήσεις εισιτηρίων για κάθε εκδήλωση.

```
SELECT
    e.Name AS EventName,
    SUM(bt.Quantity * t.Price) AS TotalRevenue
FROM
    BookingTicket bt
JOIN
    Ticket t ON bt.TicketID = t.TicketID
JOIN
    Event e ON t.EventID = e.EventID
GROUP BY
    e.Name;
```

3. Δημοφιλέστερη εκδήλωση βάσει κρατήσεων

Βρίσκουμε την εκδήλωση με τις περισσότερες κρατήσεις.

```
SELECT
    e.Name AS EventName,
    COUNT(b.BookingID) AS TotalBookings
FROM
```

```
Booking b
JOIN
Event e ON b.EventID = e.EventID
GROUP BY
e.Name
ORDER BY
TotalBookings DESC
LIMIT 1;
```

4. Εκδήλωση με τα περισσότερα έσοδα σε ένα χρονικό εύρος

Βρίσκουμε την εκδήλωση που έχει αποφέρει τα περισσότερα έσοδα σε ένα συγκεκριμένο χρονικό διάστημα.

```
SELECT
e.Name AS EventName,
SUM(bt.Quantity * t.Price) AS TotalRevenue
FROM
BookingTicket bt
JOIN
Ticket t ON bt.TicketID = t.TicketID
JOIN
Event e ON t.EventID = e.EventID
JOIN
Booking b ON bt.BookingID = b.BookingID
WHERE
b.BookingDate BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY
e.Name
ORDER BY
TotalRevenue DESC
LIMIT 1;
```


5. Προβολή κρατήσεων ανά χρονική περίοδο

Βρίσκουμε όλες τις κρατήσεις που έγιναν σε συγκεκριμένο χρονικό διάστημα.

SELECT

b.BookingID,

c.FullName AS CustomerName,

e.Name AS EventName,

b.BookingDate,

SUM(bt.Quantity * t.Price) AS TotalPayment

FROM

Booking b

JOIN

Customer c ON b.CustomerID = c.CustomerID

JOIN

Event e ON b.EventID = e.EventID

JOIN

BookingTicket bt ON b.BookingID = bt.BookingID

JOIN

Ticket t ON bt.TicketID = t.TicketID

WHERE

b.BookingDate BETWEEN '2024-01-01' AND '2024-12-31'

GROUP BY

b.BookingID, c.FullName, e.Name, b.BookingDate;

```

SELECT
    b.BookingID,
    c.FullName AS CustomerName,
    e.Name AS EventName,
    b.BookingDate,
    SUM(bt.Quantity * t.Price) AS TotalPayment
FROM
    Booking b
JOIN
    Customer c ON b.CustomerID = c.CustomerID
JOIN
    Event e ON b.EventID = e.EventID
JOIN
    BookingTicket bt ON b.BookingID = bt.BookingID
JOIN
    Ticket t ON bt.TicketID = t.TicketID
WHERE
    b.BookingDate BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY
    b.BookingID, c.FullName, e.Name, b.BookingDate;

```

6. Συνολικά έσοδα από VIP ή γενικά εισιτήρια ανά εκδήλωση

Υπολογίζουμε τα έσοδα από συγκεκριμένο τύπο εισιτηρίων (π.χ. VIP) για κάθε εκδήλωση.

```

SELECT
    e.Name AS EventName,
    t.SeatType,
    SUM(bt.Quantity * t.Price) AS Revenue
FROM
    BookingTicket bt
JOIN
    Ticket t ON bt.TicketID = t.TicketID
JOIN
    Event e ON t.EventID = e.EventID
WHERE
    t.SeatType = 'VIP'
GROUP BY
    e.Name, t.SeatType;

```

7. Πελάτες που έχουν κάνει κρατήσεις για συγκεκριμένη εκδήλωση

Βρίσκουμε όλους τους πελάτες που έχουν κάνει κράτηση για μια συγκεκριμένη εκδήλωση.

```

SELECT
    c.FullName AS CustomerName,
    c.Email,
    b.BookingID,
    SUM(bt.Quantity * t.Price) AS TotalPayment
FROM
    Customer c
JOIN
    Booking b ON c.CustomerID = b.CustomerID
JOIN
    BookingTicket bt ON b.BookingID = bt.BookingID
JOIN
    Ticket t ON bt.TicketID = t.TicketID

```

JOIN

Event e ON b.EventID = e.EventID

WHERE

e.Name = 'Concert XYZ'

GROUP BY

c.FullName, c.Email, b.BookingID;

8. Συνολικά έσοδα για όλες τις εκδηλώσεις

Βρίσκουμε το συνολικό ποσό που έχει συγκεντρωθεί από όλες τις πωλήσεις εισιτηρίων.

SELECT

SUM(bt.Quantity * t.Price) AS TotalRevenue

FROM

BookingTicket bt

JOIN

Ticket t ON bt.TicketID = t.TicketID;

Με αυτές τις ερωτήσεις, μπορούμε να ανακτήσουμε κρίσιμα δεδομένα για την παρακολούθηση της απόδοσης της επιχείρησης και τη διαχείριση των εκδηλώσεων!

Όψη για Έσοδα ανά Εκδήλωση

Περιγραφή: Η όψη "EventRevenueView" εμφανίζει τα συνολικά έσοδα από τις κρατήσεις εισιτηρίων για κάθε εκδήλωση. Χρησιμοποιείται για την ταχύτερη εξαγωγή αναφορών σχετικά με τα έσοδα.

Χρήση: Απλοποιεί την εξαγωγή οικονομικών δεδομένων, χωρίς να χρειάζεται να γράφονται συνεχώς πολύπλοκα ερωτήματα.

Εντολή SQL:

CREATE VIEW EventRevenueView AS

SELECT

```
e.Name AS EventName,  
SUM(bt.Quantity * t.Price) AS TotalRevenue  
FROM  
    BookingTicket bt  
JOIN  
    Ticket t ON bt.TicketID = t.TicketID  
JOIN  
    Event e ON t.EventID = e.EventID  
GROUP BY  
    e.Name;
```

Όψη για Πελάτες με Πολλαπλές Κρατήσεις

Περιγραφή: Η όψη "FrequentCustomersView" εμφανίζει τους πελάτες που έχουν κάνει περισσότερες από μία κρατήσεις.

Χρήση: Βοηθά στη δημιουργία αναφορών για τους πιο συχνούς πελάτες, ώστε να γίνουν προωθητικές ενέργειες.

Εντολή SQL:

```
CREATE VIEW FrequentCustomersView AS  
SELECT  
    c.CustomerID,  
    c.FullName,  
    COUNT(b.BookingID) AS TotalBookings  
FROM  
    Customer c  
JOIN  
    Booking b ON c.CustomerID = b.CustomerID  
GROUP BY  
    c.CustomerID, c.FullName  
HAVING  
    COUNT(b.BookingID) > 1;
```

3: Όψη για Διαθέσιμες Θέσεις Ανά Εκδήλωση

Περιγραφή: Η όψη "AvailableSeatsView" δείχνει τις διαθέσιμες θέσεις ανά εκδήλωση και τύπο εισιτηρίου.

Χρήση: Διευκολύνει την άμεση προβολή της διαθεσιμότητας χωρίς να γίνονται σύνθετες ενώσεις στους πίνακες.

Εντολή SQL:

```
CREATE VIEW AvailableSeatsView AS
SELECT
    e.Name AS EventName,
    t.SeatType,
    (t.Availability - COALESCE(SUM(bt.Quantity), 0)) AS AvailableSeats
FROM
    Ticket t
JOIN
    Event e ON t.EventID = e.EventID
LEFT JOIN
    BookingTicket bt ON t.TicketID = bt.TicketID
GROUP BY e.Name, t.SeatType, t.Availability;
```

Αυτές είναι μερικές όψεις οι οποίες μπορούν να χρησιμοποιηθούν στην δικιά μας βάση δεδομένων αλλά δεν είναι σίγουρο πως θα χρησιμοποιηθούν

Η περιγραφή σε ψευδοκώδικα των διαδικασιών

1. Εγγραφή Νέου Πελάτη

Διαδικασία Εγγραφή_Νέου_Πελάτη

Εμφάνισε "Εισαγάγετε τα στοιχεία του πελάτη"

Διάβασε FullName, Email, CreditCardInfo

Αν FullName KENO Ή Email KENO Ή CreditCardInfo KENO Τότε

Εμφάνισε "Η εγγραφή απέτυχε. Όλα τα πεδία είναι υποχρεωτικά."

Επιστροφή

Τέλος Αν

Αποθήκευσε τα στοιχεία του πελάτη στον πίνακα Customer

Εμφάνισε "Η εγγραφή ολοκληρώθηκε επιτυχώς"

Τέλος Διαδικασίας

2. Δημιουργία Νέας Εκδήλωσης

Διαδικασία Δημιουργία_Νέας_Εκδήλωσης

Εμφάνισε "Εισαγάγετε τα στοιχεία της εκδήλωσης"

Διάβασε EventName, EventDate, EventTime, EventType, Capacity

Αν EventName KENO Ή EventDate KENO Ή Capacity ≤ 0 Τότε

Εμφάνισε "Η δημιουργία της εκδήλωσης απέτυχε. Ελέγξτε τα στοιχεία."

Επιστροφή

Τέλος Αν

Αποθήκευσε την εκδήλωση στον πίνακα Event

Εμφάνισε "Η εκδήλωση δημιουργήθηκε επιτυχώς"

Τέλος Διαδικασίας

3. Αναζήτηση Διαθέσιμων Θέσεων

Διαδικασία Αναζήτηση_Διαθέσιμων_Θέσεων

Εμφάνισε "Εισαγάγετε το όνομα της εκδήλωσης"

Διάβασε EventName

Αν EventName δεν υπάρχει στον πίνακα Event Τότε

Εμφάνισε "Η εκδήλωση δεν βρέθηκε"

Επιστροφή

Τέλος Αν

Αναζήτησε στον πίνακα Ticket τις διαθέσιμες θέσεις για την εκδήλωση EventName

Εμφάνισε τις διαθέσιμες θέσεις και τους τύπους εισιτηρίων

Τέλος Διαδικασίας

4. Κράτηση Εισιτηρίων

Διαδικασία Κράτηση_Εισιτηρίων

Εμφάνισε "Εισαγάγετε το EventName, τον αριθμό εισιτηρίων και τον τύπο θέσης"

Διάβασε EventName, Quantity, SeatType

Αν EventName δεν υπάρχει στον πίνακα Event Τότε

Εμφάνισε "Η εκδήλωση δεν βρέθηκε"

Επιστροφή

Τέλος Αν

Αναζήτησε τη διαθεσιμότητα εισιτηρίων στον πίνακα Ticket για τον συγκεκριμένο τύπο θέσης

Αν Quantity > Διαθέσιμα_Εισιτήρια Τότε

Εμφάνισε "Ανεπαρκής διαθεσιμότητα εισιτηρίων"

Επιστροφή

Τέλος Αν

Μείωσε τα διαθέσιμα εισιτήρια στον πίνακα Ticket

Καταχώρισε την κράτηση στον πίνακα Booking

Εμφάνισε "Η κράτηση ολοκληρώθηκε επιτυχώς"

Τέλος Διαδικασίας

5. Ακύρωση Κράτησης

Διαδικασία Ακύρωση_Κράτησης

Εμφάνισε "Εισαγάγετε το BookingID για ακύρωση"

Διάβασε BookingID

Αν BookingID δεν υπάρχει στον πίνακα Booking Τότε

Εμφάνισε "Η κράτηση δεν βρέθηκε"

Επιστροφή

Τέλος Αν

Επίστρεψε τα χρήματα στον Customer

Ενημέρωσε τα διαθέσιμα εισιτήρια στον πίνακα Ticket

Διαγραφή της εγγραφής από τον πίνακα Booking

Εμφάνισε "Η κράτηση ακυρώθηκε επιτυχώς"

Τέλος Διαδικασίας

6. Ακύρωση Εκδήλωσης

Διαδικασία Ακύρωση_Εκδήλωσης

Εμφάνισε "Εισαγάγετε το EventID για ακύρωση"

Διάβασε EventID

Αν EventID δεν υπάρχει στον πίνακα Event Τότε

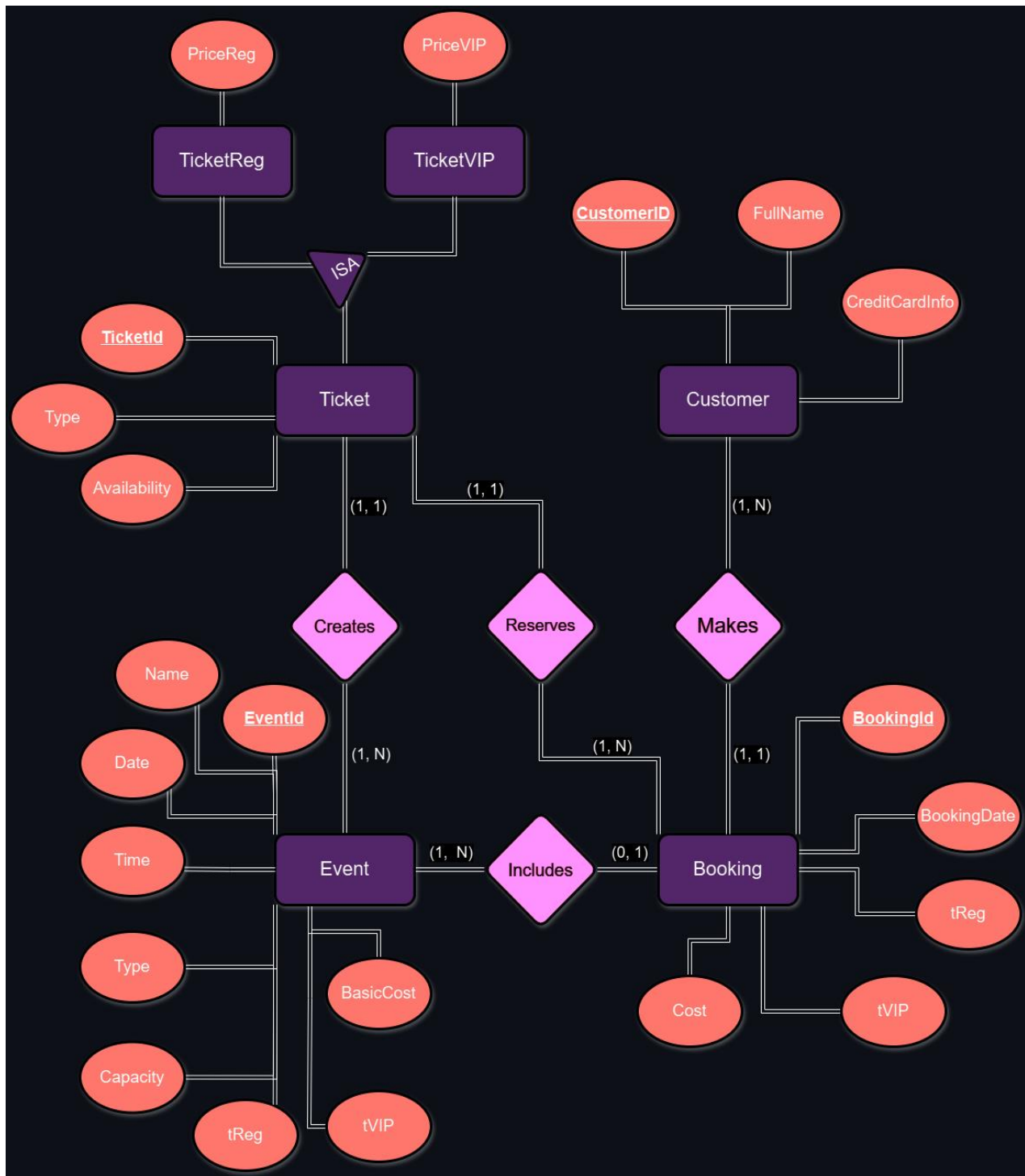
Εμφάνισε "Η εκδήλωση δεν βρέθηκε"

Επιστροφή
Τέλος Αν
Διαγραφή όλων των σχετικών κρατήσεων από τον πίνακα Booking
Ενημέρωσε τα διαθέσιμα εισιτήρια στον πίνακα Ticket
Διαγραφή της εκδήλωσης από τον πίνακα Event
Εμφάνισε "Η εκδήλωση ακυρώθηκε επιτυχώς"
Τέλος Διαδικασίας

Αυτή η αναφορά αφορά τη Φάση 1 του έργου, όπου παρουσιάζονται η εννοιολογική μοντελοποίηση και οι βασικές προδιαγραφές του συστήματος. Το πρότζεκτ μπορεί να υλοποιηθεί με διάφορους τρόπους, ανάλογα με τις ανάγκες και τις απαιτήσεις που θα προκύψουν. Υπάρχει μεγάλη πιθανότητα να αλλάξουν αρκετά πράγματα κατά τη διάρκεια της υλοποίησης, είτε λόγω νέων αναγκών είτε λόγω προσαρμογών που θα γίνουν για τη βελτίωση του τελικού αποτελέσματος.

Project phase B (team 1)

Τελική Μορφή Εννοιολογικού Μοντέλου



1. Πρώτη Κανονική Μορφή (1NF) Η βάση δεδομένων είναι σε 1NF επειδή:

Όλες οι τιμές των ιδιοτήτων είναι ατομικές (atomic). Δεν υπάρχουν π.χ. λίστες, σύνολα ή πολυτιμές ιδιότητες. Για παράδειγμα, το Customer.FullName είναι μία απλή τιμή, όχι π.χ. λίστα ονομάτων. Το ίδιο ισχύει και για άλλες ιδιότητες όπως BookingDate, PriceReg, κλπ. 2. Δεύτερη Κανονική Μορφή (2NF) Η βάση δεδομένων είναι σε 2NF επειδή:

Είναι ήδη σε 1NF. Δεν υπάρχουν μερικές εξαρτήσεις (partial dependencies). Δηλαδή, κάθε μη πρωτεύουσα ιδιότητα εξαρτάται πλήρως από το πρωτεύον κλειδί της οντότητας ή της σχέσης. Παράδειγμα:

Στην οντότητα Booking, η ιδιότητα Cost εξαρτάται απόλυτα από το BookingId. Στην οντότητα TicketReg, η ιδιότητα PriceReg εξαρτάται απόλυτα από το TicketId. Δεν υπάρχει περίπτωση μία ιδιότητα να εξαρτάται μόνο από ένα μέρος ενός σύνθετου πρωτεύοντος κλειδιού (αν υπήρχε).

3. Τρίτη Κανονική Μορφή (3NF) Η βάση δεδομένων είναι σε 3NF επειδή:

Είναι ήδη σε 2NF. Δεν υπάρχουν μεταβατικές εξαρτήσεις (transitive dependencies). Δηλαδή, καμία μη πρωτεύουσα ιδιότητα δεν εξαρτάται από άλλη μη πρωτεύουσα ιδιότητα. Παράδειγμα:

Στην οντότητα Event, η ιδιότητα BasicCost εξαρτάται μόνο από το EventId. Δεν εξαρτάται από άλλες μη πρωτεύουσες ιδιότητες, όπως το Name ή το Date. Στην οντότητα Customer, οι ιδιότητες FullName και CreditCardInfo εξαρτώνται μόνο από το CustomerId. Οι εξαρτήσεις είναι καθαρές και συνδέονται μόνο με τα πρωτεύοντα κλειδιά.

Είχαμε εντελώς λάθος approach στην Α φάση έτσι αλλάξαμε όλα τα γνωρίσματα των οντοτήτων και καταλάβαμε ότι είναι λάθος τα γνωρίσματα στις σχέσεις. Έτσι το νέο μας μοντέλο περιλαμβάνει:

Οντότητες και Ιδιότητες

Ticket (Εισιτήριο)

Ιδιότητες:

TicketId: Μοναδικό αναγνωριστικό για κάθε εισιτήριο.

Type: Τύπος εισιτηρίου (π.χ., Κανονικό ή VIP).

Availability: Διαθεσιμότητα του εισιτηρίου.

Υποκατηγορίες:

TicketReg: Εισιτήριο Κανονικό (Regular). Περιλαμβάνει την ιδιότητα PriceReg (κόστος κανονικού εισιτηρίου).

TicketVIP: Εισιτήριο VIP. Περιλαμβάνει την ιδιότητα PriceVIP (κόστος VIP εισιτηρίου).

Customer (Πελάτης)

Ιδιότητες:

CustomerId: Μοναδικό αναγνωριστικό πελάτη.

FullName: Ονοματεπώνυμο πελάτη.

CreditCardInfo: Πληροφορίες πιστωτικής κάρτας.

Event (Εκδήλωση)

Ιδιότητες:

EventId: Μοναδικό αναγνωριστικό εκδήλωσης.

Name: Όνομα της εκδήλωσης.

Date: Ημερομηνία διεξαγωγής.

Time: Ώρα διεξαγωγής.

Type: Τύπος εκδήλωσης (π.χ., συναυλία, θέατρο).

Capacity: Συνολική χωρητικότητα.

BasicCost: Βασικό κόστος συμμετοχής.

tReg: Αριθμός διαθέσιμων κανονικών εισιτηρίων.

tVIP: Αριθμός διαθέσιμων VIP εισιτηρίων.

Booking (Κράτηση)

Ιδιότητες:

BookingId: Μοναδικό αναγνωριστικό κράτησης.

BookingDate: Ημερομηνία κράτησης.

Cost: Κόστος κράτησης.

tReg: Αριθμός κανονικών εισιτηρίων που έχουν κρατηθεί.

tVIP: Αριθμός VIP εισιτηρίων που έχουν κρατηθεί.

Σχέσεις

Creates (Δημιουργεί)

Σχέση μεταξύ του Event και του Ticket.

Ένα Event δημιουργεί N Ticket.

Includes (Περιλαμβάνει)

Σχέση μεταξύ του Event και του Booking.

Κάθε Event μπορεί να περιλαμβάνει από 0 έως πολλές κρατήσεις (0, N).

Κάθε Booking σχετίζεται με ακριβώς ένα Event (1, 1).

Reserves (Κρατά)

Σχέση μεταξύ του Booking και του Ticket.

Ένα booking μπορεί να κάνει μία ή περισσότερες κρατήσεις εισιτηρίων (1, N).

Κάθε εισιτήριο μπορεί να συνδεθεί μόνο με ένα booking(1, 1).

Makes (Πραγματοποιεί)

Σχέση μεταξύ του Customer και του Booking.

Ένας πελάτης μπορεί να πραγματοποιήσει πολλές κρατήσεις (1, N).

Κάθε κράτηση ανήκει σε έναν μόνο πελάτη (1, 1).

Ενδεικτικά αποτελέσματα από την εκτέλεση των διαδικασιών

1) ADD/Delete Event

Event Name

Eminem

VIP Tickets

20

Event Date (yyyy-MM-dd)

2025-12-12

Regular Tickets

80

Event Time (hh:mm:ss)

22:00:00

Cost

100

Event Type

Concert

Event Capacity

100

Submit

2) Add/Delete Customer

Full Name

Email

Credit Card Info

Customers Recommended Input

Όνομα: Δημήτρης Παπαδόπουλος

Email: d.papadopoulos@example.com

IBAN: GR1601101250000000012300695

Όνομα: Μαρία Καραγιάννη

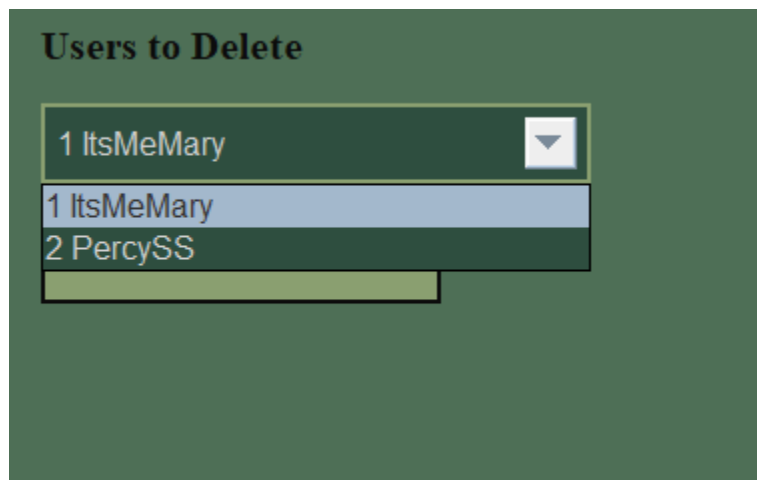
Email: m.karagianni@example.com

IBAN: GR3202602000000000200012345

Όνομα: Νικόλαος Γεωργίου

Email: n.georgiou@example.com

IBAN: GR7801002300000000009876543



3) Add/Delete Booking

Choose Client

1 ItsMeMary



Choose Event

2 Eminem



Booking Date (yyyy-MM-dd)

2024-12-12

Submit

VIP Tickets available:20

5

Regular Tickets available:80

10

Submit

VIP Tickets available:15

Regular Tickets available:70

Choose Client

2 PercySS



Submit

Choose Booking

1 2024-12-12



Submit

4) Additional Features

Income of Event

3 Anna Vissi

Submit

Income: 2500.0

Choose Event

2 Eminem

Choose Ticket Type

VIP

Submit

The event profits are: 1500.0

Most Popular Event is:

Eminem

Submit

Date start (yyyy-MM-dd)

2024-01-01

Date end (yyyy-MM-dd)

2025-12-12

Submit

Eminem

Screenshots from PHP MyAdmin

Tables

<input type="checkbox"/>	bookings	★	Περιήγηση	Δομή	Αναζήτηση	Προσθήκη	Άδειασμα	Διαγραφή	4	InnoDB	utf8mb4_general_ci	48,0 KB	-
<input type="checkbox"/>	customers	★	Περιήγηση	Δομή	Αναζήτηση	Προσθήκη	Άδειασμα	Διαγραφή	3	InnoDB	utf8mb4_general_ci	16,0 KB	-
<input type="checkbox"/>	events	★	Περιήγηση	Δομή	Αναζήτηση	Προσθήκη	Άδειασμα	Διαγραφή	2	InnoDB	utf8mb4_general_ci	16,0 KB	-
<input type="checkbox"/>	ticketsregular	★	Περιήγηση	Δομή	Αναζήτηση	Προσθήκη	Άδειασμα	Διαγραφή	160	InnoDB	utf8mb4_general_ci	48,0 KB	-
<input type="checkbox"/>	ticketsvip	★	Περιήγηση	Δομή	Αναζήτηση	Προσθήκη	Άδειασμα	Διαγραφή	40	InnoDB	utf8mb4_general_ci	48,0 KB	-

Table Bookings

←T→

▼

Click the drop-down arrow to toggle column's visibility.

		BookingId	BookingDate	CustomerId	EventId	tReg	tVIP	Cost
	Αντιγραφή Διαγραφή	1	2024-12-12	2	2	0	5	1500.00
<input type="checkbox"/> Επεξεργασία	Αντιγραφή Διαγραφή	2	2024-12-13	1	1	5	5	2000.00
<input type="checkbox"/> Επεξεργασία	Αντιγραφή Διαγραφή	3	2024-12-15	3	1	10	0	1000.00
<input type="checkbox"/> Επεξεργασία	Αντιγραφή Διαγραφή	4	2024-12-15	3	2	15	10	4500.00

Table Customers

←T→

▼

	CustomerId	FullName	Email	CreditCardInfo
<input type="checkbox"/> Επεξεργασία	1	Δημήτρης Παπαδόπουλος	d.papadopoulos@example.com	GR1601101250000000012300695
<input type="checkbox"/> Επεξεργασία	2	Μαρία Καραγιάννη	m.karagianni@example.com	GR3202602000000000200012345
<input type="checkbox"/> Επεξεργασία	3	Νικόλαος Γεωργίου	n.georgiou@example.com	GR7801002300000000009876543

Table Events

←T→

▼

	EventId	Name	Date	Time	Type	Capacity	tReg	tVIP	Price
<input type="checkbox"/> Επεξεργασία	1	Eminem	2025-12-12	12:00:00	Concert	100	80	20	100.00
<input type="checkbox"/> Επεξεργασία	2	Anna Vlsi	2025-12-13	12:00:00	Concert	100	80	20	100.00

Table Tickets Vip




























































































































← T →					TicketId	Type	Price	Availability	EventId	BookingId
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	1	VIP	300.00	0	1	2
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	2	VIP	300.00	0	1	2
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	3	VIP	300.00	0	1	2
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	4	VIP	300.00	0	1	2
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	5	VIP	300.00	0	1	2
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	6	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	7	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	8	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	9	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	10	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	11	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	12	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	13	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	14	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	15	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	16	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	17	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	18	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	19	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	20	VIP	300.00	1	1	NULL
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	21	VIP	300.00	0	2	1
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	22	VIP	300.00	0	2	1
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	23	VIP	300.00	0	2	1
<input type="checkbox"/>		Επεξεργασία	 Αντιγραφή	 Διαγραφή	24	VIP	300.00	0	2	1

Table TicketsRegular

←T→				TicketId	Type	Price	Availability	EventId	BookingId
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	1	Regular	100.00	0	1	2
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	2	Regular	100.00	0	1	2
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	3	Regular	100.00	0	1	2
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	4	Regular	100.00	0	1	2
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	5	Regular	100.00	0	1	2
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	6	Regular	100.00	0	1	3
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	7	Regular	100.00	0	1	3
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	8	Regular	100.00	0	1	3
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	9	Regular	100.00	0	1	3
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	10	Regular	100.00	0	1	3
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	11	Regular	100.00	0	1	3
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	12	Regular	100.00	0	1	3
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	13	Regular	100.00	0	1	3
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	14	Regular	100.00	0	1	3
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	15	Regular	100.00	0	1	3
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	16	Regular	100.00	1	1	NULL
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	17	Regular	100.00	1	1	NULL
<input type="checkbox"/>	 Επεξεργασία	 Αντιγραφή	 Διαγραφή	18	Regular	100.00	1	1	NULL

Περιγραφή των περιορισμών της υλοποίησής σας και των δυνατοτήτων βελτίωσής του.

Περιορισμοί:

Ασφάλεια Δεδομένων: Η εφαρμογή δεν παρέχει κρυπτογράφηση για ευαίσθητα δεδομένα όπως οι πληροφορίες πιστωτικής κάρτας. Χρήσιμο θα ήταν να ενσωματωθούν σύγχρονες μεθόδους κρυπτογράφησης για την προστασία αυτών των δεδομένων.

Αποδοτικότητα Ερωτημάτων SQL: Στην τρέχουσα υλοποίηση, πολλά ερωτήματα SQL (όπως τα SELECT *) μπορούν να επιβαρύνουν την απόδοση της βάσης δεδομένων. Θα ήταν καλύτερο να χρησιμοποιούνται πιο συγκεκριμένα ερωτήματα για την αποφυγή περιττών δεδομένων.

Διαχείριση Λαθών: Η εφαρμογή δεν παρέχει επαρκή διαχείριση λαθών για όλες τις περιπτώσεις αποτυχίας των διαδικασιών (π.χ., κατά την εκτέλεση συναλλαγών ή ενημερώσεων). Είναι σημαντικό να εισαχθεί καλύτερη διαχείριση εξαιρέσεων.

Χρήση Εφαρμογής μέσω GUI: Η εφαρμογή βασίζεται στην εμφάνιση μηνυμάτων μέσω GUI για την επικοινωνία με τον χρήστη, κάτι που μπορεί να περιορίσει την επεκτασιμότητα και την ευχρηστία της εφαρμογής σε μεγάλες παραγωγικές συνθήκες.

Δυνατότητες Βελτίωσης:

Κρυπτογράφηση Δεδομένων: Για την προστασία των χρηστών και τη συμμόρφωση με τα πρότυπα ασφαλείας, θα μπορούσε να ενσωματωθεί κρυπτογράφηση για τις πληροφορίες πιστωτικών καρτών και άλλων ευαίσθητων δεδομένων.

Αναβάθμιση του UI/UX: Η διεπαφή χρήστη (UI) μπορεί να γίνει πιο φιλική και λειτουργική με τη χρήση πιο σύγχρονων εργαλείων ή βιβλιοθηκών για την ανάπτυξη εφαρμογών GUI.

Βελτίωση Σύνδεσης με τη Βάση Δεδομένων: Η εφαρμογή θα μπορούσε να επωφεληθεί από τεχνικές βελτίωσης της απόδοσης στη σύνδεση και στα ερωτήματα με την προσθήκη βελτιστοποιήσεων όπως indexing, prepared statements ή caching.

Αυτόματη Διαχείριση Κρατήσεων και Ενημερώσεων: Η δυνατότητα αυτόματης αποκατάστασης κρατήσεων και τη διαχείριση με ταχύτερους μηχανισμούς για ενημερώσεις στη βάση δεδομένων θα μπορούσε να βελτιώσει την εμπειρία του χρήστη και να μειώσει τα σφάλματα.

Database (Creation Of db and tables)

```
package db;
```

```
import javax.swing.*;
```

```
import java.sql.*;
```

```
public class DataBase {
```

```
    static Connection connection = null;
```

```
    private final static String driver = "com.mysql.cj.jdbc.Driver";
```

```
    private final static String url = "jdbc:mysql://localhost";
```

```
    private final static String user = "root";
```

```
    private final static String password = "";
```

```
    private final static String dbName = "agency_db";
```

```
    private final static int port = 3306;
```

```
// ===== DataBase Methods =====
```

```
public static void createDb() {
```

```
    try {
```

```
        Class.forName(driver);
```

```

        Connection con = DriverManager.getConnection(url + ":" + port + "/", user,
password);

        Statement stm = con.createStatement();

        stm.executeUpdate("DROP DATABASE IF EXISTS agency_db");

        stm.executeUpdate("CREATE DATABASE IF NOT EXISTS agency_db");

        System.out.println("Database created");

    } catch (Exception e) {

        System.out.println("Error: " + e);

    }

}

public static Connection update(String query) {

    try {

        Class.forName(driver);

        if (connection == null) {

            connection = DriverManager.getConnection(url + ":" + port + "/" + dbName +
"?characterEncoding=UTF-8", user, password);

        }

        Statement stm = connection.createStatement();

        stm.executeUpdate(query);

        System.out.println("Database updated");

        return connection;

    } catch (Exception e) {

        throw new RuntimeException(e);

    }

}

```

```

public static ResultSet get(String query) {
    try {
        Class.forName(driver);
        if (connection == null) {
            connection = DriverManager.getConnection(url + ":" + port + "/" + dbName +
"?characterEncoding=UTF-8", user, password);
        }
        Statement stm = connection.createStatement();
        ResultSet rs = stm.executeQuery(query);
        System.out.println("Got from DB");
        return rs;
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}

```

```

public static void dropDb() {
    update("DROP DATABASE IF EXISTS agency_db");
}

```

```

public void close() {
    try {
        connection.close();
    } catch (Exception e) {
        System.out.println("Error: " + e);
    }
}

```

```
}  
}
```

```
public static void initDb() {
```

```
    update("CREATE TABLE IF NOT EXISTS events (" +  
        "EventId INT AUTO_INCREMENT PRIMARY KEY," +  
        "Name VARCHAR(255)," +  
        "Date DATE," +  
        "Time TIME," +  
        "Type VARCHAR(255)," +  
        "Capacity INT," +  
        "tReg INT," +  
        "tVIP INT," +  
        "Price DEC(10, 2)" +  
        ")");
```

```
    update("CREATE TABLE IF NOT EXISTS customers (" +  
        "CustomerId INT AUTO_INCREMENT PRIMARY KEY," +  
        "FullName VARCHAR(255)," +  
        "Email VARCHAR(255)," +  
        "CreditCardInfo VARCHAR(255)" +  
        ")");
```

```
    update("CREATE TABLE IF NOT EXISTS bookings (" +  
        "BookingId INT AUTO_INCREMENT PRIMARY KEY," +  
        "BookingDate DATE," +
```

```
"CustomerId INT," +  
"EventId INT," +  
"tReg INT," +  
"tVIP INT," +  
"Cost DEC(10, 2)," +  
"FOREIGN KEY (CustomerId) REFERENCES customers(CustomerId) ON DELETE  
CASCADE," +  
"FOREIGN KEY (EventId) REFERENCES events(EventId) ON DELETE CASCADE" +  
");
```

```
update("CREATE TABLE IF NOT EXISTS ticketsVIP (" +  
"TicketId INT AUTO_INCREMENT PRIMARY KEY," +  
"Type VARCHAR(255)," +  
"Price DEC(10, 2)," +  
"Availability INT," +  
"EventId INT," +  
"BookingId INT," +  
"FOREIGN KEY (EventId) REFERENCES events(EventId) ON DELETE CASCADE," +  
"FOREIGN KEY (BookingId) REFERENCES bookings(BookingId) ON DELETE  
CASCADE" +  
");
```

```
update("CREATE TABLE IF NOT EXISTS ticketsRegular (" +  
"TicketId INT AUTO_INCREMENT PRIMARY KEY," +  
"Type VARCHAR(255)," +  
"Price DEC(10, 2)," +
```

```

        "Availability INT," +
        "EventId INT," +
        "BookingId INT," +
        "FOREIGN KEY (EventId) REFERENCES events(EventId) ON DELETE CASCADE," +
        "FOREIGN KEY (BookingId) REFERENCES bookings(BookingId) ON DELETE
CASCADE" +
        "));
    }

```

```

public static void dropTable(String tableName) {
    update("DROP TABLE IF EXISTS " + tableName);
}

```

```

public static void cleanTable(String tableName) {
    update("DELETE * FROM " + tableName);
}

```

```

public static int[] booked_availableTickets(int EventId) throws SQLException {
    ResultSet rs = get("SELECT * FROM events WHERE EventId = " + EventId);
    rs.next();
    int tReg = rs.getInt("tReg");
    int tVIP = rs.getInt("tVIP");

    rs = get("SELECT * FROM bookings WHERE EventId = " + EventId);
    while (rs.next()) {
        tReg -= rs.getInt("tReg");
    }
}

```



```
        tVIP -= rs.getInt("tVIP");
    }

    return new int[]{tReg, tVIP};
}
```

```
public static float eventIncome(int EventId) throws SQLException {
    ResultSet rs = get("SELECT * FROM bookings WHERE EventId = " + EventId);
    float income = 0;
    while (rs.next()) {
        income += rs.getFloat("Cost");
    }

    return income;
}
```

```
public static String mostPopularEvent() throws SQLException {
    // most popular event based on bookings

    ResultSet rs = get("SELECT EventId, COUNT(EventId) AS count FROM bookings
GROUP BY EventId ORDER BY count DESC LIMIT 1");

    rs.next();

    int eventId = rs.getInt("EventId");

    rs = get("SELECT * FROM events WHERE EventId = " + eventId);
    rs.next();
}
```

```
        return rs.getString("Name");
    }
}
```

```
public static String mostProfitableEvent(Date d1, Date d2) throws SQLException {
    // most profitable event based in a certain time period

    // check if d1 pre dates d2
    if (d1.after(d2)) {
        JOptionPane.showMessageDialog(null, "Invalid date range", "Error",
JOptionPane.ERROR_MESSAGE);
        return null;
    }
}
```

```
        ResultSet rs = get("SELECT EventId, SUM(Cost) AS sum FROM bookings WHERE
BookingDate BETWEEN '" + d1 + "' AND '" + d2 + "' GROUP BY EventId ORDER BY sum DESC
LIMIT 1");

        rs.next();

        int eventId = rs.getInt("EventId");

        rs = get("SELECT * FROM events WHERE EventId = " + eventId);

        rs.next();

        return rs.getString("Name");
    }
}
```

```
public static float[] profitsTickets(int EventId) throws SQLException {
    float profitR = 0, profitV = 0;
```

```

ResultSet rs;

if (EventId == 0) {
    // all events

    rs = get("SELECT * FROM ticketsRegular WHERE AVAILABILITY = 0");

    while (rs.next()) {
        profitR += rs.getFloat("Price");
    }

    rs = get("SELECT * FROM ticketsVIP WHERE AVAILABILITY = 0");

} else {
    // one event

    rs = get("SELECT * FROM ticketsRegular WHERE EventId = " + EventId + " AND
AVAILABILITY = 0");

    while (rs.next()) {
        profitR += rs.getFloat("Price");
    }

    rs = get("SELECT * FROM ticketsVIP WHERE EventId = " + EventId + " AND
AVAILABILITY = 0");

}

while (rs.next()) {
    profitV += rs.getFloat("Price");
}

return new float[]{profitR, profitV};

```

```
}
```

```
}
```

Event (add-delete-available tickets)

```
package classes;
```

```
import javax.swing.*;
```

```
import java.sql.*;
```

```
import static classes.Booking.deleteBooking;
```

```
import static classes.Ticket.*;
```

```
import static db.dataBase.*;
```

```
public class Event {
```

```
    public static boolean addEvent(String Name, Date Date, Time Time, String Type, int Capacity, int tReg, int tVIP, float Price) throws SQLException {
```

```
        ResultSet rs;
```

```
        StringBuilder made = new StringBuilder();
```

```
        made.append("Event not created: ").append(Name);
```

```
        // check for valid input
```

```
        if (Name == null || Date == null || Time == null || Type == null || Capacity <= 0 || tReg < 0 || tVIP < 0 || Price < 0) {
```

```
            System.out.println(made.append("Invalid input"));
```

```

        return false;
    }

    // check if the event is in the past
    Date today = new Date(System.currentTimeMillis());
    if (Date.before(today)) {
        System.out.println(made.append("Invalid Date"));
        return false;
    }

    // check for valid time
    if (Time.before(java.sql.Time.valueOf("00:00:00")) ||
        Time.after(java.sql.Time.valueOf("23:59:59"))) {
        made.append("Invalid Time");

        JOptionPane.showMessageDialog(null, made, "Error",
            JOptionPane.ERROR_MESSAGE);

        return false;
    }

    // check if there are enough tickets available
    if (tReg + tVIP != Capacity) {
        made.append("Invalid number of tickets Regular tickets and VIP tickets must be
        equal to capacity");

        JOptionPane.showMessageDialog(null, made, "Error",
            JOptionPane.ERROR_MESSAGE);

        return false;
    }

```

```

// check if the event is already in the database

rs = get("SELECT * FROM events WHERE Name = '" + Name + "' AND Date = '" + Date + "'
AND Time = '" + Time + "'");

if (rs.next()) {

    made.append("Event already exists");

    JOptionPane.showMessageDialog(null, made, "Error",
JOptionPane.ERROR_MESSAGE);

    return false;

}

made.setLength(0);

made.append("Event created: ").append(Name);

int generatedId = -1;

update("INSERT INTO events (Name, Date, Time, Type, Capacity, tReg, tVIP, Price)
VALUES ('" + Name + "', '" + Date + "', '" + Time + "', '" + Type + "', " + Capacity + ", " + tReg + ",
" + tVIP + ", " + Price + "')");

// I get the last generated id from the database

rs = get("SELECT * FROM events ORDER BY EventId DESC LIMIT 1");

if (rs.next()) {

    generatedId = rs.getInt("EventId");

}

for (int i = 0; i < tReg; i++) {

    addTicketRegular(Price, 1, generatedId);

}

```

```
for (int i = 0; i < tVIP; i++) {  
    addTicketVIP(3*Price, 1, generatedId);  
}
```

```
JOptionPane.showMessageDialog(null, made, "Event created",  
JOptionPane.INFORMATION_MESSAGE);  
  
return true;  
}
```

```
public static boolean deleteEvent(int EventId) throws SQLException {  
    // check if the event exists  
  
    ResultSet rs = get("SELECT * FROM events WHERE EventId = " + EventId);  
    if (!rs.next()) {  
        JOptionPane.showMessageDialog(null, "Event not found", "Error",  
JOptionPane.ERROR_MESSAGE);  
        return false;  
    }
```

```
    // delete all bookings for this event  
  
    rs = get("SELECT * FROM bookings WHERE EventId = " + EventId);  
    while (rs.next()) {  
        deleteBooking(rs.getInt("BookingId"));  
    }
```

```
    // delete all tickets for this event
```

```
rs = get("SELECT * FROM ticketsRegular WHERE EventId = " + EventId);  
while (rs.next()) {  
    deleteTicket(rs.getInt("EventId"));  
}
```

```
rs = get("SELECT * FROM ticketsVIP WHERE EventId = " + EventId);  
while (rs.next()) {  
    deleteTicketVIP(rs.getInt("EventId"));  
}
```

```
update("DELETE FROM events WHERE EventId = " + EventId);
```

```
JOptionPane.showMessageDialog(null, "Event deleted", "Event deleted",  
JOptionPane.INFORMATION_MESSAGE);  
  
return true;  
}
```

```
public static int availableTickets(int EventId, int Type) throws SQLException {  
    ResultSet rs;  
    int count = 0;  
  
    if (Type == 1) {  
        rs = get("SELECT * FROM ticketsRegular WHERE EventId = " + EventId + " AND  
Availability = 1");  
    } else {  
        rs = get("SELECT * FROM ticketsVIP WHERE EventId = " + EventId + " AND Availability  
= 1");  
    }
```



```
}
```

```
while (rs.next()) {
```

```
    count++;
```

```
}
```

```
return count;
```

```
}
```

```
}
```

Ticket (add-delete)

```
public static boolean addTicket(String Type, float Price, int Availability, int EventId)  
throws SQLException {
```

```
    ResultSet rs;
```

```
    StringBuilder made = new StringBuilder();
```

```
    made.append("Ticket not created: ");
```

```
    // check for valid input
```

```
    if (Price < 0 || Availability < 0 || EventId < 0 || Type == null || Type.isEmpty()) {
```

```
        System.out.println(made.append("Invalid input"));
```

```
        return false;
```

```
    }
```

```
    // check if the event exists
```

```
rs = get("SELECT * FROM events WHERE EventId = " + EventId);  
if (!rs.next()) {  
    System.out.println("Event does not exist");  
    return false;  
}
```

```
made.setLength(0);  
made.append("Ticket created:");
```

```
if (Type.equals("VIP")) {  
    addTicketVIP(Price, Availability, EventId);  
} else {  
    addTicketRegular(Price, Availability, EventId);  
}  
return true;  
}
```

```
public static void addTicketVIP(float Price, int Availability, int EventId) throws  
SQLException {  
    update("INSERT INTO ticketsVIP (Price, Availability, EventId, BookingId, Type) VALUES  
(" + Price + ", " + Availability + ", " + EventId + ", NULL, 'VIP')");  
}
```

```
public static void addTicketRegular(float Price, int Availability, int EventId) throws  
SQLException {  
    update("INSERT INTO ticketsRegular (Price, Availability, EventId, BookingId, Type)  
VALUES (" + Price + ", " + Availability + ", " + EventId + ", NULL, 'Regular')");  
}
```

```
}
```

```
public static boolean deleteTicket(int EventId) throws SQLException {
```

```
    ResultSet rs;
```

```
    StringBuilder made = new StringBuilder();
```

```
    made.append("Ticket not deleted: ");
```

```
    // check for valid input
```

```
    if (EventId < 0) {
```

```
        System.out.println(made.append("Invalid input"));
```

```
        return false;
```

```
    }
```

```
    // check if the event exists
```

```
    rs = get("SELECT * FROM events WHERE EventId = " + EventId);
```

```
    if (!rs.next()) {
```

```
        System.out.println(made.append("Event does not exist"));
```

```
        return false;
```

```
    }
```

```
    made.setLength(0);
```

```
    // see if it is a VIP or Regular ticket
```

```
    rs = get("SELECT * FROM ticketsVIP WHERE EventId = " + EventId);
```

```
    if (rs.next()) {
```

```
        deleteTicketVIP(rs.getInt("TicketId"));
```

```
    } else {
```

```

        rs = get("SELECT * FROM ticketsRegular WHERE EventId = " + EventId);
        if (rs.next()) {
            deleteTicketRegular(rs.getInt("TicketId"));
        }
    }

    System.out.println(made.append("Ticket deleted"));
    return true;
}

```

```

public static void deleteTicketVIP(int TicketId) throws SQLException {
    update("DELETE FROM ticketsVIP WHERE TicketId = " + TicketId);
}

```

```

public static void deleteTicketRegular(int TicketId) throws SQLException {
    update("DELETE FROM ticketsRegular WHERE TicketId = " + TicketId);
}
}

```

Customer (add-delete)

```

public class Customer {

    public static boolean addCustomer(String FullName, String email, String CreditCardInfo)
    throws SQLException {

        ResultSet rs;

        StringBuilder made = new StringBuilder();
    }
}

```

```
made.append("Customer not created: ");

// check for valid input

if (FullName == null || email == null || CreditCardInfo == null || FullName.isEmpty() ||
email.isEmpty() || CreditCardInfo.isEmpty()) {

    JOptionPane.showMessageDialog(null, made.append("Invalid input"), "Error",
JOptionPane.ERROR_MESSAGE);

    return false;
}

// check if the customer is already in the database

rs = get("SELECT * FROM customers WHERE FullName = '" + FullName + "' AND email =
'" + email + "'");

if (rs.next()) {

    JOptionPane.showMessageDialog(null, made.append("Customer already exists"),
"Error", JOptionPane.ERROR_MESSAGE);

    return false;
}

// check if the email is valid form

if (!email.contains("@") || !email.contains(".")) {

    JOptionPane.showMessageDialog(null, made.append("Invalid email"), "Error",
JOptionPane.ERROR_MESSAGE);

    return false;
}

// check if the credit card info is valid form (2 first char GR followed by 16 digits)
```

```

        if (!CreditCardInfo.startsWith("GR") || CreditCardInfo.length() != 27) {

            JOptionPane.showMessageDialog(null, made.append("Invalid Credit Card Info"),
"Error", JOptionPane.ERROR_MESSAGE);

            return false;

        }

        made.setLength(0);

        made.append("Customer created: ").append(FullName);

        update("INSERT INTO customers (FullName, email, CreditCardInfo) VALUES ('" +
FullName + "', '" + email + "', '" + CreditCardInfo + "')");

        JOptionPane.showMessageDialog(null, made, "Success",
JOptionPane.INFORMATION_MESSAGE);

        return true;

    }

    public static boolean deleteCustomer(int CustomerId) throws SQLException {

        // check if the customer exists

        ResultSet rs = get("SELECT * FROM customers WHERE CustomerId = " + CustomerId);

        if (!rs.next()) {

            JOptionPane.showMessageDialog(null, "Customer not found", "Error",
JOptionPane.ERROR_MESSAGE);

            return false;

        }

        // delete all bookings of this customer

        rs = get("SELECT * FROM bookings WHERE CustomerId = " + CustomerId);

```

```

while (rs.next()) {
    deleteBooking(rs.getInt("BookingId"));
}

update("DELETE FROM customers WHERE CustomerId = " + CustomerId);

JOptionPane.showMessageDialog(null, "Customer deleted", "Success",
JOptionPane.INFORMATION_MESSAGE);

return true;
}

```

Booking (add-delete)

```

public static boolean addBooking(int CustomerId, Date Date, int EventId, int tReg, int tVIP)
throws SQLException {

    ResultSet rs;

    StringBuilder made = new StringBuilder();

    int availableRegular = 0, availableVIP = 0;

    float cost = 0;

    made.append("Booking not created: ");

    // check if the customer exists

    rs = get("SELECT * FROM customers WHERE CustomerId = " + CustomerId);

    if (!rs.next()) {

        JOptionPane.showMessageDialog(null, made.append("Customer does not exist"),
        "Error", JOptionPane.ERROR_MESSAGE);

        return false;
    }
}

```

```

// check if the event exists

rs = get("SELECT * FROM events WHERE EventId = " + EventId);

if (!rs.next()) {

    JOptionPane.showMessageDialog(null, made.append("Event does not exist"),
"Error", JOptionPane.ERROR_MESSAGE);

    return false;

}


// check if there enough tickets available

rs = get("SELECT * FROM ticketsRegular WHERE EventId = " + EventId + " AND
Availability = 1");

while (rs.next()) {

    availableRegular++;

}


rs = get("SELECT * FROM ticketsVIP WHERE EventId = " + EventId + " AND Availability =
1");

while (rs.next()) {

    availableVIP++;

}


if (tReg > availableRegular || tVIP > availableVIP) {

    JOptionPane.showMessageDialog(null, made.append("Not enough tickets
available"), "Error", JOptionPane.ERROR_MESSAGE);

    return false;

}

```



```
update("INSERT INTO bookings (BookingDate, CustomerId, EventId, tReg, tVIP, Cost)
VALUES (" + Date + ", " + CustomerId + ", " + EventId + ", " + tReg + ", " + tVIP + ", " + cost +
");
```

```
// store the booking id from last added booking
```

```
rs = get("SELECT * FROM bookings ORDER BY BookingId DESC LIMIT 1");
```

```
rs.next();
```

```
int Boo = rs.getInt("BookingId");
```

```
// make tickets unavailable and assign them to the booking and add the cost to the
booking cost
```

```
rs = get("SELECT * FROM ticketsRegular WHERE EventId = " + EventId + " AND
Availability = 1 LIMIT " + tReg);
```

```
while (rs.next()) {
```

```
    update("UPDATE ticketsRegular SET Availability = 0, BookingId = " + Boo + " WHERE
TicketId = " + rs.getInt("TicketId"));
```

```
    cost += rs.getFloat("Price");
```

```
}
```

```
rs = get("SELECT * FROM ticketsVIP WHERE EventId = " + EventId + " AND Availability =
1 LIMIT " + tVIP);
```

```
while (rs.next()) {
```

```
    update("UPDATE ticketsVIP SET Availability = 0, BookingId = " + Boo + " WHERE
TicketId = " + rs.getInt("TicketId"));
```

```
    cost += rs.getFloat("Price");
```

```
}
```

```
JOptionPane.showMessageDialog(null, "Booking added successfully", "Success",  
JOptionPane.INFORMATION_MESSAGE);
```

```
update("UPDATE bookings SET Cost = " + cost + " WHERE BookingId = " + Boo);
```

```
return true;
```

```
}
```

```
public static boolean deleteBooking(int BookingId) throws SQLException {
```

```
String creditcard;
```

```
int customerId;
```

```
// check if the booking exists
```

```
ResultSet rs = get("SELECT * FROM bookings WHERE BookingId = " + BookingId);
```

```
if (!rs.next()) {
```

```
JOptionPane.showMessageDialog(null, "Booking does not exist", "Error",  
JOptionPane.ERROR_MESSAGE);
```

```
return false;
```

```
} else {
```

```
customerId = rs.getInt("CustomerId");
```

```
}
```

```
rs = get("SELECT * FROM customers WHERE CustomerId = " + customerId);
```

```
rs.next();
```

```
creditcard = rs.getString("CreditCardInfo");
```

```
rs = get("SELECT * FROM bookings WHERE BookingId = " + BookingId);
```

```
rs.next();
```

```
JOptionPane.showMessageDialog(null, "The cost: " + rs.getFloat("Cost") + "will be  
refunded to account: " + creditcard + ".", "Refund",  
JOptionPane.INFORMATION_MESSAGE);
```

```
// make all tickets included in this booking available again  
  
rs = get("SELECT * FROM ticketsRegular WHERE BookingId = " + BookingId);  
  
while (rs.next()) {  
  
    update("UPDATE ticketsRegular SET Availability = 1, BookingId = NULL WHERE  
TicketId = " + rs.getInt("TicketId"));  
  
}  
  
rs = get("SELECT * FROM ticketsVIP WHERE BookingId = " + BookingId);  
  
while (rs.next()) {  
  
    update("UPDATE ticketsVIP SET Availability = 1, BookingId = NULL WHERE TicketId = "  
+ rs.getInt("TicketId"));  
  
}
```

```
JOptionPane.showMessageDialog(null, "Booking deleted successfully", "Success",  
JOptionPane.INFORMATION_MESSAGE);  
  
update("DELETE FROM bookings WHERE BookingId = " + BookingId);  
  
return true;  
  
}
```

Το συγκεκριμένο project ήταν αρκετά απαιτητικό, καθώς περιλάμβανε λεπτομερή σχεδίαση βάσεων δεδομένων, ανάλυση σύνθετων σχέσεων μεταξύ οντοτήτων και εφαρμογή των αρχών κανονικοποίησης μέχρι την 3NF. Παρά την πολυπλοκότητα, ήταν ένα πολύ ενδιαφέρον και challenging έργο, που μας έδωσε την ευκαιρία να εξελίξουμε τις γνώσεις μας στον σχεδιασμό συστημάτων. Η διαδικασία μας βοήθησε να κατανοήσουμε καλύτερα πώς η θεωρία συνδέεται με την πράξη, ενώ παράλληλα αναπτύξαμε νέες δεξιότητες επίλυσης προβλημάτων .