

Έκθεση ομαδικής εργασίας (πρότζεκτ) - Ομάδα 45

Εφαρμογή κρατήσεων εισιτηρίων πολιτιστικών εκδηλώσεων (π.χ. θεατρική παράσταση, συναυλία κλπ.)

Κωνσταντίνος Γαβαλάς

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών, Πανεπιστήμιο Πατρών, up1072772@upnet.gr

Δημήτριος Γρυλλάκης

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών, Πανεπιστήμιο Πατρών, up1072711@upnet.gr

Η παρούσα Έκθεση περιγράφει την διαδικασία που ακολουθήθηκε και τα αποτελέσματα που επιτεύχθηκαν στα πλαίσια της ομαδικής εργασίας του μαθήματος Βάσεων Δεδομένων (ECE_ΓΚ703). Το θέμα της ομάδας μας είναι “εφαρμογή κρατήσεων εισιτηρίων πολιτιστικών εκδηλώσεων (π.χ. θεατρική παράσταση, συναυλία κλπ.)”. Παρουσιάζεται αρχικά μία σύντομη περιγραφή του μικρόκοσμου και ορίζονται σε γενικές γραμμές οι προδιαγραφές και η μεθοδολογία μας. Ακολουθεί στη συνέχεια ο εννοιολογικός σχεδιασμός καθώς και η πραγματική υλοποίηση της βάσης ενώ περιγράφεται επίσης και η υλοποίηση μίας proof-of-concept εφαρμογής για την ανάδειξη των δυνατοτήτων της. Αξιολογείται τέλος η πληρότητα της προτεινόμενης λύσης. Για την εγκατάσταση της εφαρμογής παρέχονται οδηγίες στο Παράρτημα Α.1.

1 ΕΙΣΑΓΩΓΗ

Μια αναδυόμενη startup εταιρία έχει ήδη έρθει σε επικοινωνία με διοργανωτές διαφόρων πολιτιστικών εκδηλώσεων για την αποκλειστική διανομή των εισιτηρίων τους από το καλοκαίρι του 2023 και μας αναθέτει τον σχεδιασμό της βάσεως δεδομένων του συστήματος κρατήσεων. Η εφαρμογή θα χρησιμοποιείται από τους χρήστες για την αναζήτηση γεγονότων με βάση τα ενδιαφέροντα, την περιοχή κατοικίας τους κλπ. και την άμεση κράτηση διαθέσιμων εισιτηρίων για αυτά.

Ύστερα από επικοινωνία με τους ιδρυτές της εταιρίας προκύπτει ότι η εταιρία έχει ήδη κλείσει συμφωνίες με εξωτερικούς παρόχους υπηρεσιών πληρωμής και άρα δεν έχει περαιτέρω απαιτήσεις για την διαχείριση των πληρωμών (οι κρατήσεις καταγράφονται στη βάση μας αφού ολοκληρωθεί η πληρωμή).

1.1 Προδιαγραφές

Βασικές προδιαγραφές μας για την ανάπτυξη της βάσης και της proof-of-concept εφαρμογής είναι οι εξής:

1. Λειτουργικότητα: Η βάση πρέπει να επιτρέπει την αποθήκευση όλων των απαιτούμενων δεδομένων για την κατάλληλη λειτουργία του τελικού προϊόντος και η εφαρμογή να αναδεικνύει αυτήν την λειτουργικότητα.
2. Ακεραιότητα: Η βάση πρέπει να διασφαλίζει την ακεραιότητα των δεδομένων, τόσο κατά την δημιουργία τους όσο και σε βάθος χρόνου.
3. Ταχύτητα: Η βάση και η προγραμματιστική διεπαφή προς αυτή πρέπει να είναι όσο το δυνατόν ταχύτερες με χρόνους απάντησης ανεπαίσθητους για τον τελικό χρήστη.
4. Ευελιξία: Η βάση και η προγραμματιστική διεπαφή προς αυτή πρέπει να περιορίζουν όσο το δυνατόν λιγότερο τον προγραμματιστή και να επιτρέπουν εύκολα την επέκταση της τελικής εφαρμογής.

Στα πλαίσια της λειτουργικότητας θεωρήθηκε απαραίτητο στην πιλοτική εφαρμογή να δίνονται στον χρήστη οι παρακάτω δυνατότητες:

1. Να αναζητά πολιτιστικές εκδηλώσεις με βάση τα ενδιαφέροντά του
2. Να περιορίζει τα αποτελέσματα με βάση την χρονική και χωρική εγγύτητά τους και
3. Να δημιουργεί κρατήσεις για τις επιλεγμένες εκδηλώσεις και να λαμβάνει τα αντίστοιχα εισιτήρια

1.2 Μεθοδολογία

Για την ολοκληρωμένη αντιμετώπιση του προβλήματος η διαδικασία χωρίστηκε σε τρεις φάσεις:

- Φάση Α: Σχεδιασμός εννοιολογικού μοντέλου
- Φάση Β: Δημιουργία της βάσης
- Φάση Γ: Δημιουργία εφαρμογής για την επίδειξη της λειτουργίας της βάσης

Ο σχεδιασμός του διαγράμματος οντοτήτων-συσχετίσεων και του λογικού σχεσιακού μοντέλου έγινε με εκτεταμένη συνεργασία των δύο συγγραφέων, όπως και η αντίστοιχη δημιουργία της βάσης. Η εφαρμογή χωρίστηκε σε δύο μέρη (την διεπαφή που επικοινωνεί με την βάση και την τελική εφαρμογή με την γραφική διεπαφή) τα οποία αναπτύχθηκαν παράλληλα από τους δύο συγγραφείς.

2 ΕΝΝΟΙΟΛΟΓΙΚΟΣ ΣΧΕΔΙΑΣΜΟΣ

2.1 Διάκριση Όρων

Πρώτο βήμα για τον κατάλληλο εννοιολογικό σχεδιασμό της βάσης μας ήταν η διάκριση ορισμένων κρίσιμων όρων.

Για την καλύτερη οργάνωση της βάσης κρίθηκε απαραίτητο να χωρίσουμε την κάθε πολιτιστική εκδήλωση σε δύο λογικά μέρη:

1. Εκδήλωση (Event): Περιλαμβάνει όλες τις λεπτομέρειες (όνομα, περιγραφή, κατηγορία κλπ.) που χαρακτηρίζουν την παράσταση/συναυλία κλπ.
→ Για παράδειγμα: "Ρωμαίος και Ιουλιέτα", "Rock Believer Tour 2022: Scorpions", ...
2. Γεγονός (Happening): Μια εμφάνιση κάποιας εκδήλωσης σε συγκεκριμένη ημερομηνία και τοποθεσία.
→ Για παράδειγμα: "Ρωμαίος και Ιουλιέτα - Θέατρο Απόλλων - 2022-11-08 21:00"

Σε αυτό το στάδιο εντοπίστηκαν επίσης τέσσερις ακόμα σημαντικές οντότητες που πιθανώς θα συμμετέχουν στην τελική μας λύση:

3. Χρήστης (User): Ένα άτομο με λογαριασμό στην τελική εφαρμογή
4. Κράτηση (Reservation): Μία δέσμευση εισιτηρίων από έναν Χρήστη αφού εκτελεστεί η αντίστοιχη πληρωμή
5. Εισιτήριο (Ticket): Το αποδεικτικό με το οποίο μπορεί κάποιος να προσέλθει στην πολιτιστική εκδήλωση

6. Τοποθεσία (Location): Τα θέατρα, οι συναυλιακοί χώροι κλπ στους οποίους γίνονται πολιτιστικές εκδηλώσεις

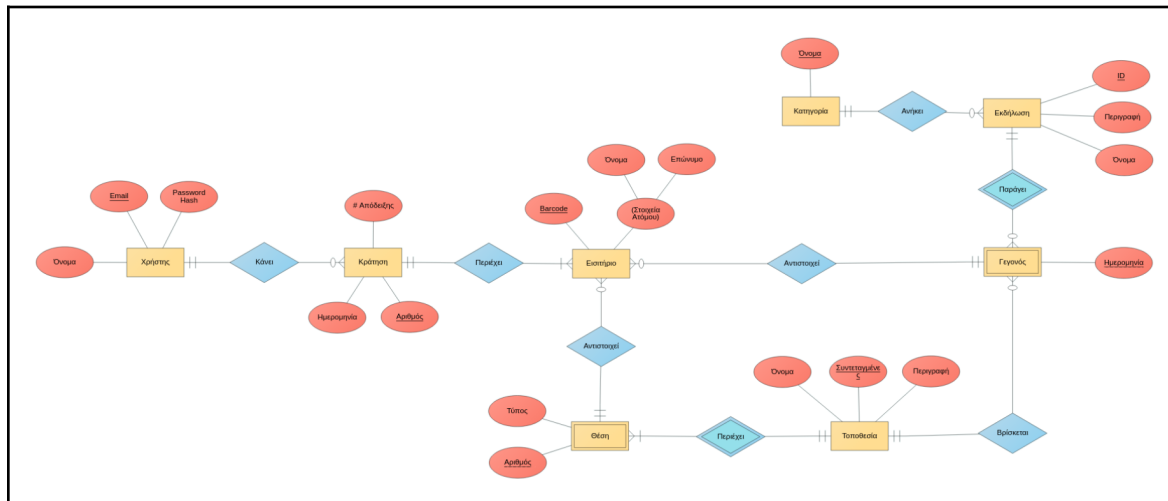
2.2 Διάγραμμα Οντοτήτων-Συσχετίσεων

Έχοντας αναλύσει τώρα τους όρους που διέπουν το πρόβλημά μας μπορούμε να περάσουμε στην ανάπτυξη του διαγράμματος οντοτήτων-συσχετίσεων.

Σχεδιάζοντας αρχικά τις βασικές οντότητες που αναλύσαμε προηγουμένως εντοπίζουμε δύο ακόμα (λίγο πιο αφηρημένες) οντότητες:

7. Θέση (Seat): Μία θέση εντός ενός χώρου εκδηλώσεων (π.χ. τα καθίσματα στο θέατρο). Η καταγραφή τους είναι απαραίτητη για τον υπολογισμό των διαθέσιμων εισιτηρίων για ένα Γεγονός και τον κατάλληλο διαμορισμό αυτών στους ενδιαφερόμενους χρήστες.
8. Κατηγορία (Category): Καθώς η εφαρμογή μας θα διαχειρίζεται πολλούς τύπους Εκδηλώσεων επιλέγουμε να δημιουργήσουμε άλλη μία οντότητα για την κωδικοποίηση των διαφορετικών κατηγοριών (π.χ. Θέατρο, Συναυλία κλπ.)

Προσθέτοντας τώρα τις κατάλληλες σχέσεις μεταξύ των οντοτήτων καταλήγουμε στο διάγραμμα του Σχήματος 1.



Σχήμα 1: Το διάγραμμα Οντοτήτων-Συσχετίσεων που αναπτύχθηκε

Παρατηρούμε ότι η Θέση και το Γεγονός αποτελούν ασθενείς οντότητες, με το Γεγονός να χαρακτηρίζεται από την Εκδήλωση και την ημερομηνία του και την Θέση από την Τοποθεσία και τον αριθμό της. Σε επόμενα στάδια, χάριν απλότητας, όλες οι οντότητες μετατράπηκαν σε ισχυρές με την προσθήκη ενός μοναδικού χαρακτηριστικού (ID).

Δίνεται επίσης μεγάλη έμφαση στις πληθικότητες των διαφόρων σχέσεων. Για παράδειγμα:

1. Ο κάθε Χρήστης μπορεί να κάνει πολλές Κρατήσεις ενώ η κάθε Κράτηση αντιστοιχεί σε έναν μόνο Χρήστη
2. Η κάθε Κράτηση περιέχει ένα ή περισσότερα Εισιτήρια ενώ κάθε Εισιτήριο αντιστοιχεί σε μία μόνο Κράτηση
3. Το κάθε Γεγονός αντιστοιχεί σε μία μόνο Εκδήλωση και μία μόνο Τοποθεσία ενώ μπορούν να υπάρχουν πολλά Εισιτήρια για ένα Γεγονός

2.3 Λογικό Σχεσιακό Μοντέλο

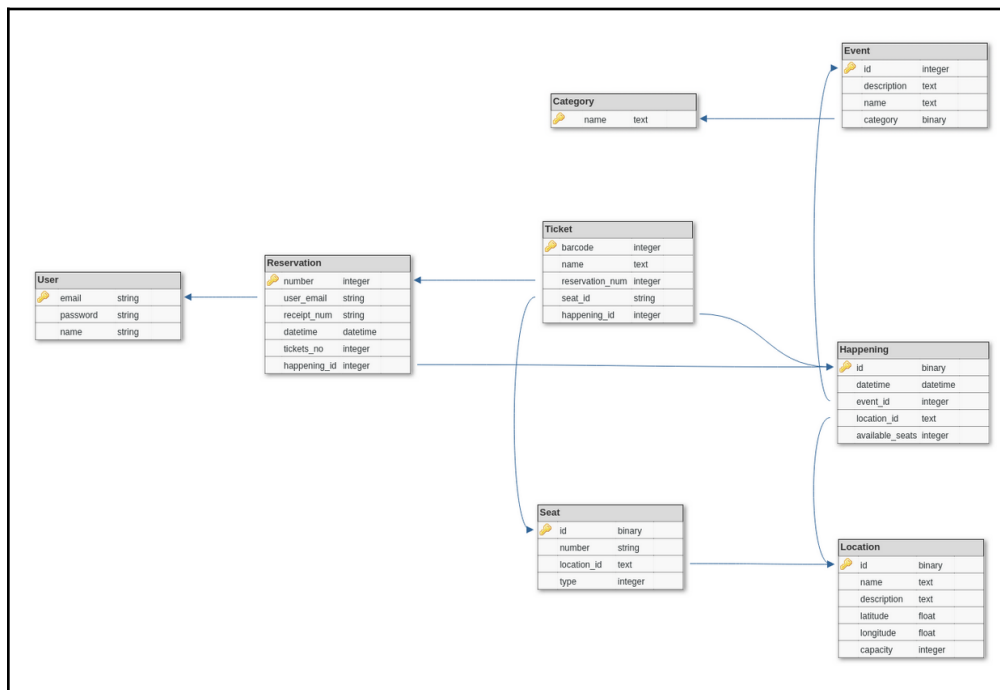
Προχωρώντας στην ανάπτυξη του λογικού σχεσιακού μοντέλου εστιάζουμε περισσότερο στην πρακτική υλοποίηση της βάσης μας. Μετατρέπουμε αρχικά τις ασθενείς οντότητες σε ισχυρές προσθέτοντας ένα μοναδικό γνώρισμα (ID). Στη συνέχεια, εντοπίζουμε και διορθώνουμε τα εξής προβλήματα:

1. Με βάση το διάγραμμα οντοτήτων-συσχετίσεων για να βρούμε το Γεγονός στο οποίο αντιστοιχεί μία Κράτηση πρέπει πρώτα να βρούμε ένα Εισιτήριο που της αντιστοιχεί. Αυτό κρίνεται άτοπο αφού τα εισιτήρια εκδίδονται από τους διοργανωτές μετά την δημιουργία της Κράτησης και άρα για ένα χρονικό διάστημα δε θα γνωρίζουμε σε ποιο Γεγονός αντιστοιχεί η κάθε Κράτηση. Για την διόρθωση προσθέτουμε μία επιπλέον σύνδεση μεταξύ Κράτησης και Γεγονότος (1-N).
2. Με βάση το διάγραμμα οντοτήτων-συσχετίσεων για να υπολογίσουμε τον αριθμό διαθέσιμων θέσεων πρέπει:
 - a. Να βρούμε την τοποθεσία στην οποία βρίσκεται το γεγονός
 - b. Να βρούμε όλες τις θέσεις που αντιστοιχούν σε αυτήν την τοποθεσία (έστω TEMP1)
 - c. Να βρούμε όλα τα εισιτήρια που αντιστοιχούν σε αυτό το γεγονός
 - d. Να βρούμε τις θέσεις που αντιστοιχούν σε αυτά τα εισιτήρια (έστω TEMP2)
 - e. Να αφαιρέσουμε τους δύο πίνακες (TEMP1 - TEMP2) και να μετρήσουμε τα αποτελέσματα

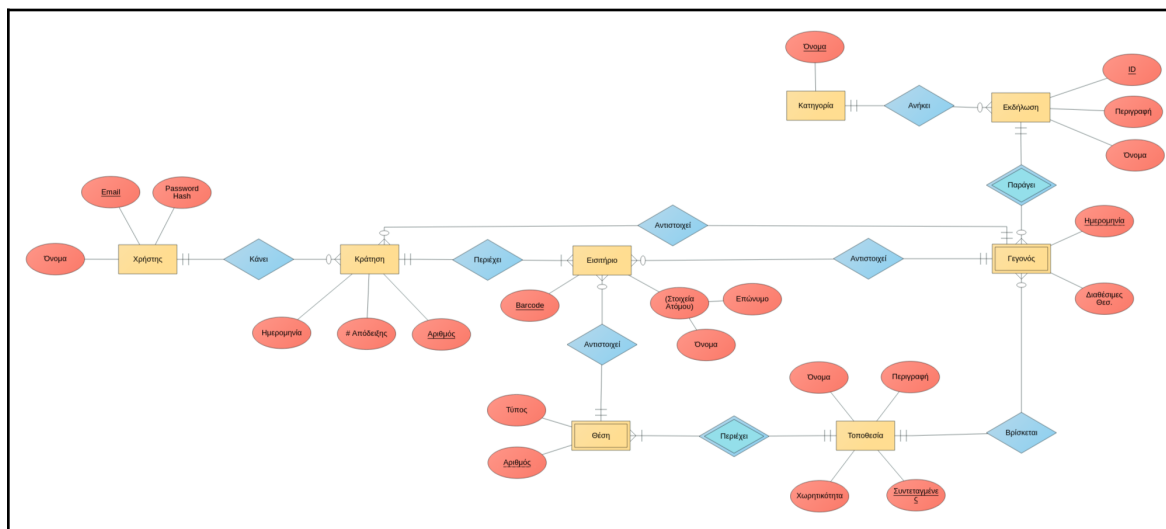
Αυτό κρίνεται επίσης άτοπο αφού η εύρεση διαθέσιμων Γεγονότων (Γεγονότων που έχουν διαθέσιμα εισιτήρια) αναμένουμε να είναι πολύ συχνή διαδικασία. Για τον λόγο αυτό προσθέτουμε το πεδίο `available_seats` στον πίνακα `Happening` που θα ενημερώνεται δυναμικά από την βάση με τον αριθμό των διαθέσιμων θέσεων.

3. Για τον υπολογισμό του μεγέθους ενός χώρου πρέπει να βρούμε και να μετρήσουμε όλες τις θέσεις που αντιστοιχούν σε αυτόν. Για παρόμοιο λόγο με προηγουμένως επιλέγουμε αντιθέτως να προσθέσουμε ένα ακόμα πεδίο στον πίνακα `Happening` (`capacity`) που περιέχει την συνολική χωρητικότητα της κάθε τοποθεσίας.

Το ολοκληρωμένο Λογικό Σχεσιακό Μοντέλο φαίνεται στο Σχήμα 2 ενώ το ενημερωμένο διάγραμμα Οντοτήτων-Συσχετίσεων φαίνεται στο Σχήμα 3. Να σημειωθεί ότι μετά τις αλλαγές η σχέση Εισιτηρίου και Γεγονότος δεν είναι πλέον απαραίτητη ωστόσο διατηρήθηκε για την διευκόλυνση του προγραμματιστή.



Σχήμα 2: Το Λογικό Σχεσιακό Μοντέλο που αναπτύχθηκε



Σχήμα 3: Το τελικό διάγραμμα Οντοτήτων-Συσχετίσεων

3 ΔΗΜΙΟΥΡΓΙΑ ΤΗΣ ΒΑΣΗΣ

Για την υλοποίηση της Βάσης Δεδομένων επιλέχθηκε το σύστημα διαχείρισης βάσεων δεδομένων (DBMS) *SQLite*¹.

3.1 Δημιουργία Πινάκων

Ακολουθώντας το λογικό σχεσιακό μοντέλο που αναπτύχθηκε ξεκινάμε ορίζοντας τους οκτώ πίνακες της βάσης μας:

```
CREATE TABLE IF NOT EXISTS "User" (  
    "email"      TEXT,  
    "password"   TEXT NOT NULL,  
    "name"       TEXT NOT NULL,  
    PRIMARY KEY("email")  
);
```

```
CREATE TABLE IF NOT EXISTS "Category" (  
    "name"       TEXT,  
    PRIMARY KEY("name")  
);
```

Ο πίνακας User έχει ως κύριο κλειδί το email και τα άλλα δύο πεδία του ορισμένα ως NOT NULL τιμές. Προφανώς, σε μία πραγματική βάση το πεδίο password πρέπει να αποθηκεύεται με ασφαλή τρόπο (χρήση κατάλληλων μεθόδων hashing και salting). Ο πίνακας Category στη συγκεκριμένη υλοποίησή περιέχει μόνο το όνομα της κατηγορίας.

```
CREATE TABLE IF NOT EXISTS "Event" (  
    "id"         INTEGER,  
    "description" TEXT,  
    "name"       TEXT NOT NULL,  
    "category"   TEXT,  
    PRIMARY KEY("id" AUTOINCREMENT),
```

¹ <https://www.sqlite.org/index.html>

```

FOREIGN KEY("category") REFERENCES "Category"("name") ON DELETE SET NULL ON UPDATE
CASCADE
);

```

Ο πίνακας Event έχει ως πρωτεύον κλειδί το id του, το οποίο μάλιστα ορίζεται και ως AUTOINCREMENT. Προσθέτοντας το συγκεκριμένο keyword στον ορισμό του πρωτεύοντος κλειδιού επιτρέπουμε στον χρήστη της βάσης να προσθέτει νέες εγγραφές χωρίς να ορίζει την τιμή του πεδίου-κλειδιού, με το πεδίο να λαμβάνει τιμή αυτόματα από τη βάση. Για τη σύνδεση της κάθε εγγραφής του πίνακα Event με την αντίστοιχη εγγραφή του πίνακα Category το πεδίο category ορίζεται ως ξένο κλειδί με αναφορά στο πεδίο name του πίνακα Category. Σε περίπτωση διαγραφής της αντίστοιχης κατηγορίας κατευθύνουμε τη βάση να αντικαθιστά την τιμή του πεδίου με NULL, ενώ στην περίπτωση αντίστοιχης ενημέρωσης να ενημερώνει και το πεδίο.

```

CREATE TABLE IF NOT EXISTS "Location" (
    "id"      INTEGER,
    "name"    TEXT NOT NULL,
    "description"  INTEGER,
    "latitude"    REAL NOT NULL CHECK("latitude" >= -90 AND "latitude" <= 90),
    "longitude"   REAL NOT NULL CHECK("longitude" >= -180 AND "longitude" <= 180),
    "capacity"    INTEGER,
    PRIMARY KEY("id" AUTOINCREMENT)
);

```

Ο πίνακας Location περιέχει τις πληροφορίες για κάθε χώρο εκδηλώσεων. Πέρα από το όνομα και την περιγραφή του περιέχει επίσης πληροφορία για την τοποθεσία του με μορφή συντεταγμένων (γεωγραφικό πλάτος και μήκος), καθώς και πληροφορία για τη μέγιστη χωρητικότητά του. Στον συγκεκριμένο πίνακα βλέπουμε επίσης για πρώτη φορά τη χρήση του keyword CHECK με το οποίο μπορούμε να ορίσουμε περιορισμούς για τη διασφάλιση της ακεραιότητας των δεδομένων. Έτσι, στη συγκεκριμένη περίπτωση απαγορεύουμε τη δημιουργία εγγραφών με κακοσχηματισμένες τιμές συντεταγμένων.

```

CREATE TABLE IF NOT EXISTS "Happening" (
    "id"      INTEGER,
    "event_id"    INTEGER NOT NULL,
    "datetime"    TEXT NOT NULL,
    "location_id"  INTEGER NOT NULL,
    "available_seats"  INTEGER NOT NULL DEFAULT 0 CHECK("available_seats" >= 0),
    PRIMARY KEY("id" AUTOINCREMENT),
    FOREIGN KEY("location_id") REFERENCES "Location"("id") ON DELETE RESTRICT ON UPDATE
    CASCADE,

```

```
FOREIGN KEY("event_id") REFERENCES "Event"("id") ON DELETE CASCADE ON UPDATE CASCADE
);
```

Έχοντας τώρα ορίσει τους πίνακες Location και Event μπορούμε να ορίσουμε επίσης τον πίνακα Γεγονότων Happening. Και πάλι παρατηρούμε τη χρήση του CHECK για τον περιορισμό του πεδίου διαθέσιμων θέσεων (available_seats) σε μη-αρνητικές τιμές, ενώ εμφανίζεται για πρώτη φορά και το keyword DEFAULT με το οποίο ορίζουμε μία προεπιλεγμένη τιμή.

Για το ξένο κλειδί location_id περιορίζουμε τη δυνατότητα του διαχειριστή/συστήματος να διαγράψει Εγγραφές του πίνακα Location σε περίπτωση που υπάρχει κάποιο Γεγονός προγραμματισμένο στη συγκεκριμένη τοποθεσία, ενώ επιτρέπουμε την αλλαγή του κλειδιού μίας τοποθεσίας αλλάζοντας απλά την τιμή του πεδίου. Στην περίπτωση διαγραφής της αναφερόμενης Εκδήλωσης αντίθετα, επιτρέπουμε το cascading της διαγραφής και στις εγγραφές του πίνακα Happening.

```
CREATE TABLE IF NOT EXISTS "Reservation" (
    "number"      INTEGER,
    "user_email"   TEXT NOT NULL,
    "receipt_num"  INTEGER NOT NULL,
    "datetime"     TEXT NOT NULL,
    "tickets_no"   INTEGER NOT NULL DEFAULT 1 CHECK("tickets_no" > 0),
    "happening_id" INTEGER NOT NULL,
    FOREIGN KEY("user_email") REFERENCES "User"("email") ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY("number" AUTOINCREMENT),
    FOREIGN KEY("happening_id") REFERENCES "Happening"("id") ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Ο πίνακας κρατήσεων Reservation περιέχει, πέρα από το κλειδί του και τα ξένα κλειδιά στους πίνακες User και Happening, και την ημερομηνία και ώρα (timestamp) που δημιουργήθηκε η κράτηση. Όπως και στον πίνακα Happening, το πεδίο αυτό είναι ορισμένο με τύπο TEXT (με το τυπικό format "YYYY-MM-DD hh:mm:ss") καθώς το SQLite δεν υποστηρίζει εξειδικευμένους τύπους για την αποθήκευση χρονικών στιγμών (π.χ. DATETIME).

Μέσα από τους περιορισμούς ξένων κλειδιών αποτρέπουμε τη διαγραφή Γεγονότων σε περίπτωση ύπαρξης κρατήσεων για αυτά (προφανώς θα πρέπει πρώτα να ενημερωθούν οι επηρεαζόμενοι χρήστες και να γίνει η διαγραφή των κρατήσεων), ενώ επιτρέπουμε τη διαγραφή κρατήσεων σε περίπτωση διαγραφής του υπεύθυνου χρήστη από τη βάση.

```
CREATE TABLE IF NOT EXISTS "Seat" (
```



```

    "id"      INTEGER,
    "number"   TEXT NOT NULL,
    "location_id"  INTEGER NOT NULL,
    "type"     INTEGER,
    FOREIGN KEY("location_id") REFERENCES "Location"("id") ON DELETE CASCADE ON UPDATE
    CASCADE,
    PRIMARY KEY("id" AUTOINCREMENT)
);

CREATE TABLE IF NOT EXISTS "Ticket" (
    "barcode"    INTEGER,
    "name"       TEXT NOT NULL,
    "reservation_num"  INTEGER NOT NULL,
    "seat_id"    INTEGER NOT NULL,
    "happening_id"  INTEGER NOT NULL,
    FOREIGN KEY("happening_id") REFERENCES "Happening"("id") ON DELETE RESTRICT ON
    UPDATE CASCADE,
    FOREIGN KEY("seat_id") REFERENCES "Seat"("id") ON DELETE RESTRICT ON UPDATE CASCADE,
    PRIMARY KEY("barcode" AUTOINCREMENT),
    FOREIGN KEY("reservation_num") REFERENCES "Reservation"("number") ON DELETE CASCADE
    ON UPDATE CASCADE
);

```

Ο πίνακας θέσεων Seat αναφέρεται με ξένο κλειδί μόνο στην τοποθεσία στην οποία βρίσκεται η κάθε θέση, κατά τη διαγραφή της οποίας είναι λογικό να διαγράφονται και οι ίδιες. Το πεδίο type χρησιμοποιείται για την κοστολόγηση της κάθε θέσης και έχει μία ακέραια τιμή (π.χ. ένας χώρος μπορεί να δίνει τύπο 0 στα κανονικά του εισιτήρια και 1 στα VIP).

Τέλος, ο πίνακας εισιτηρίων Ticket αναφέρεται στους πίνακες Happening, Seat και Reservation. Και πάλι, απαγορεύουμε τη διαγραφή Γεγονότων και Θέσεων σε περίπτωση που υπάρχουν εισιτήρια που τους αντιστοιχούν ενώ αντιθέτως επιτρέπουμε τη διαγραφή των εισιτηρίων στην περίπτωση που διαγραφεί η Κράτηση στην οποία αντιστοιχούσαν.

3.2 Έλεγχοι Ακεραιότητας

Πέρα από τους ελέγχους ακεραιότητας που ορίστηκαν κατά τη δημιουργία των πινάκων (βλ. και Πίνακα 1), για τη διασφάλιση της σωστής λειτουργίας της βάσης μας και την αποφυγή προγραμματιστικών λαθών κατά την ανάπτυξη της εφαρμογής έγινε χρήση και της δυνατότητα ορισμού triggers του SQLite.

Πίνακας 1: Έλεγχοι ακεραιότητας με τη χρήση CHECK

Πεδίο	Περιορισμός	Επεξήγηση
Location.latitude	"latitude" >= -90 AND "latitude" <= 90	Το πεδίο ορισμού του γεωγραφικού πλάτους είναι το [-90°, 90°].
Location.longitude	"longitude" >= -180 AND "longitude" <= 180	Το πεδίο ορισμού του γεωγραφικού μήκους είναι το [-180°, 180°].
Happening.available_seats	"available_seats" >= 0	Ένα γεγονός δε γίνεται να έχει αρνητικό αριθμό διαθέσιμων θέσεων.
Reservation.tickets_no	"tickets_no" > 0	Μία κράτηση δεν μπορεί να αντιστοιχεί σε λιγότερα από 1 εισιτήριο.

Συγκεκριμένα δημιουργήθηκαν τέσσερα σεντ triggers:

1. Για τον περιορισμό του αριθμού εισιτηρίων που μπορούν να εκδοθούν για μία Κράτηση
2. Για την επιβεβαίωση ότι η θέση που αντιστοιχήθηκε σε ένα εισιτήριο ανήκει στην ίδια τοποθεσία με αυτή του αντίστοιχου Γεγονότος
3. Για την επικαιροποίηση του πεδίου available_seats του πίνακα Happening και
4. Για την επιβεβαίωση ότι η ημερομηνία Κράτησης είναι πριν την ημερομηνία του Γεγονότος στο οποίο αντιστοιχεί

Κάθε ένας από τους περιορισμούς υλοποιήθηκε με συνδυασμό δύο triggers (ένα για insert και ένα για update) καθώς το DBMS μας δεν υποστηρίζει τη χρήση του συντελεστή OR στο clause ON των triggers. Ακολουθούν οι ορισμοί των triggers (μόνο για την περίπτωση του insert):

```
CREATE TRIGGER check_ticket_count_before_insert BEFORE INSERT ON Ticket
BEGIN
    SELECT
        CASE
            WHEN (SELECT tickets_no FROM Reservation WHERE number = NEW.reservation_num) - 1
            < (SELECT COUNT(*) FROM Ticket WHERE reservation_num = NEW.reservation_num)
            THEN RAISE(ABORT, 'Number of tickets exceeds reservation ticket count')
        END;
END;
```

Για τον περιορισμό των αριθμό εισιτηρίων που μπορούν να εκδοθούν για μία Κράτηση, μετράμε τον αριθμό των εισιτηρίων που ήδη έχουν εκδοθεί για αυτήν την Κράτηση, προσθέτουμε 1 για την εύρεση του νέου αριθμού εισιτηρίων και συγκρίνουμε με τον αριθμό των εισιτηρίων που πρέπει να εκδοθούν συνολικά (tickets_no). Σε περίπτωση ανακολουθίας εγείρουμε αντίστοιχο σφάλμα.

```
CREATE TRIGGER check_seat_location_before_insert BEFORE INSERT ON Ticket
BEGIN
    SELECT
        CASE
            WHEN (SELECT location_id FROM Seat WHERE id = NEW.seat_id) <> (SELECT location_id
FROM Happening WHERE id = NEW.happening_id) THEN
                RAISE(ABORT, 'Seat does not belong to happening location')
            END;
END;
```

Για τον έλεγχο της τοποθεσίας μίας θέσης ενός εισιτηρίου σε σχέση με την τοποθεσία του αντίστοιχου Γεγονότος εκτελούμε δύο απλά queries και συγκρίνουμε το αποτέλεσμα τους.

```
CREATE TRIGGER available_seats_after_insert AFTER INSERT ON Reservation
BEGIN
    UPDATE Happening
    SET available_seats = available_seats - NEW.tickets_no
    WHERE id = NEW.happening_id;
END;
```

Για τον υπολογισμό των διαθέσιμων εισιτηρίων για ένα γεγονός μετά από μία καινούργια Κράτηση αρκεί να μειώσουμε από τον αριθμό των διαθέσιμων θέσεων τον αριθμό των εισιτηρίων που αντιστοιχούν στην κράτηση. Λόγω του περιορισμού του πεδίου τιμών του πεδίου available_seats (βλ. Πίνακα 1), η βάση μας αποτρέπει επίσης να δημιουργήσουμε κράτηση που περιέχει παραπάνω εισιτήρια από όσα είναι διαθέσιμα για το συγκεκριμένο Γεγονός. Για αυτόν τον κανόνα προσέχουμε επίσης τον ορισμό στην περίπτωση ενημέρωσης όπου στον αριθμό θέσεων πρέπει και να προστεθεί η παλιά τιμή του tickets_no.

```
CREATE TRIGGER reservation_date_check_before_insert BEFORE INSERT ON Reservation
BEGIN
    SELECT
        CASE
```

```

        WHEN NEW.datetime > (SELECT h.datetime FROM Happening h WHERE h.id =
NEW.happening_id) THEN
            RAISE(ABORT, 'Reservation date must be earlier than happening date')
        END;
END;

```

Ο έλεγχος της ημερομηνίας κράτησης με την ημερομηνία ενός γεγονότος τέλος γίνεται με πολύ απλοϊκό τρόπο με τη χρήση του τελεστή > και ενός απλού query.

3.3 Δημιουργία Ευρετηρίων

Για τη βελτιστοποίηση της απόδοσης της βάσης δημιουργήθηκαν επίσης τα ευρετήρια που παρουσιάζονται στον Πίνακα 2.

Πίνακας 2: Ευρετήρια

Πεδίο Ευρετηρίου	Αιτιολογία
Event.name	Για την ταχύτερη αναζήτηση Εκδηλώσεων με βάση το όνομα.
Happening.event_id	Για την ταχύτερη εύρεση Γεγονότων που αντιστοιχούν σε μία Εκδήλωση.
Happening.datetime	Για την ταχύτερη αναζήτηση Γεγονότων με βάση τη χρονική εγγύτητα.
Happening.location_id	Για την ταχύτερη αναζήτηση Γεγονότων με βάση την τοποθεσία και τη βελτιστοποίηση του κανόνα check_seat_location_before_insert.
Happening.available_seats	Για την ταχύτερη εύρεση διαθέσιμων Γεγονότων.
Location.name*	Για την ταχύτερη αναζήτηση χώρων εκδηλώσεων με βάση το όνομά τους.
Reservation.user_email*	Για την ταχύτερη εύρεση όλων των Κρατήσεων ενός χρήστη.
Reservation.happening_id*	Για την ταχύτερη εύρεση Κρατήσεων για ένα συγκεκριμένο γεγονός.
Seat.location_id	Για την ταχύτερη εύρεση των Θέσεων που βρίσκονται σε μία συγκεκριμένη τοποθεσία.
Seat.number*	Για την ταχύτερη εύρεση μίας Θέσης εντός ενός συγκεκριμένου χώρου.
Ticket.reservation_num	Για την ταχύτερη εύρεση των Εισιτηρίων που αντιστοιχούν σε μία συγκεκριμένη κράτηση και τη βελτιστοποίηση του κανόνα check_ticket_count_before_insert.
Ticket.seat_id*	Για την ταχύτερη εύρεση εισιτηρίων που αντιστοιχούν σε μία συγκεκριμένη θέση.
Ticket.happening_id*	Για την ταχύτερη εύρεση εισιτηρίων που έχουν εκδοθεί για ένα συγκεκριμένο Γεγονός.

* Η χρησιμότητα των ευρετηρίων που ακολουθούνται από αστερίσκο (*) δεν αξιολογήθηκε από την πιλοτική μας εφαρμογή

Τα πεδία γεωγραφικών συντεταγμένων latitude και longitude του πίνακα Happening χρησιμοποιούνται μόνο για τον υπολογισμό αποστάσεων με τη χρήση του τύπου του Haversine που θεωρήθηκε ότι δε θα επωφελούνταν από τη δημιουργία ευρετηρίων.

Η επίδοση των ελέγχων ακεραιότητας και των ευρετηρίων εξετάζεται στην Παράγραφο 6.

4 ΣΥΛΛΟΓΗ ΚΑΙ ΔΗΜΙΟΥΡΓΙΑ ΔΕΔΟΜΕΝΩΝ

Για την ορθότερη δοκιμή της βάσης κρίθηκε απαραίτητη η συλλογή (σχετικά) μεγάλου αριθμού δεδομένων. Για την εύρεση και την παραγωγή τους έγινε χρήση διαφόρων τεχνικών, ανάλογων πάντα και της φύσεως των διαφόρων δεδομένων.

4.1 Δημιουργία Μη-Εξαρτώμενων Δεδομένων

Ξεκινώντας από τον πίνακα Category επιλέγουμε - χωρίς βλάβη της γενικότητας - να εστιάσουμε για τις δοκιμές μας σε δύο μόνο τύπους πολιτιστικών εκδηλώσεων, το θέατρο ("Theater") και τις συναυλίες ("Concert"). Προσθέτοντας τις δύο τιμές στον πίνακα Category μπορούμε να περάσουμε τώρα στους πίνακες που δεν εμφανίζουν εξάρτηση (ξένα κλειδιά) από άλλους πίνακες. Οι τρεις πίνακες που δεν εμφανίζουν ξένα κλειδιά είναι:

1. Ο πίνακας User
2. Ο πίνακας Location και
3. Ο πίνακας Event (που αναφέρεται μόνο στο όνομα της κατηγορίας του)

Για τη δημιουργία δεδομένων για τον πίνακα χρηστών User επιλέχθηκε η χρήση της πλατφόρμας δημιουργίας ψευδοδεδομένων (mock data) *Mockaroo*². Χρησιμοποιώντας έτσι τους τύπου δεδομένων Full Name, Email Address και Password δημιουργούμε ένα csv αρχείο με ψευδοδεδομένα 5000 χρηστών. Ένα μικρό δείγμα των δεδομένων φαίνεται στον Πίνακα 3.

Πίνακας 3: Μερικά ψευδοδεδομένα του πίνακα User

email	password	name
sdearle4@patch.com	nbIDTciUVWf	Shea Dearle
hlockton5@ebay.com	MI2osc	Harley Lockton
earthur6@webnode.com	FcXX7PirBH41	Emmeline Arthur
cmacillrick7@yellowbook.com	6xDuSl	Chicky MacIllrick
tsails8@economist.com	sWf4nqzk2Go	Tana Sails

Για τον πίνακα Location χρησιμοποιούμε δύο μεθόδους:

1. Για τα θέατρα χρησιμοποιούμε και πάλι το Mockaroo προσθέτοντας στον τύπο δεδομένων Street Name το επίθεμα "Theater" και δημιουργώντας τυχαίες γεωγραφικές συντεταγμένες. Για τις περιγραφές των χώρων χρησιμοποιούμε το πρόγραμμα μηχανικής μάθησης της OpenAI *ChatGPT*³

² <https://mockaroo.com/>

³ <https://chat.openai.com/chat>

ζητώντας του με φυσική γλώσσα να μας παράξει μία λίστα πολύ γενικών περιγραφών για θεατρικούς χώρους. Προσθέτουμε τέλος χειροκίνητα μερικά Ελληνικά θέατρα.

2. Για τους συναυλιακούς χώρους ζητάμε αυτήν τη φορά από το ChatGPT τη δημιουργία ολόκληρων εγγραφών του πίνακά μας στη μορφή λιστών της Python.

Για την κατάλληλη συμπλήρωση του πεδίου `capacity` χρησιμοποιούμε ένα μικρό πρόγραμμα σε Python (βλ. αρχείο `helper/fill_capacities.py`). Μερικές από τις τελικές εγγραφές του πίνακα φαίνονται στον Πίνακα 4. Να σημειωθεί ότι οι συντεταγμένες που δημιούργησε το ChatGPT είναι ψευδοτυχαίες και δεν αντιστοιχούν στις πραγματικές τοποθεσίες των συναυλιακών κέντρων⁴.

Πίνακας 4: Μερικά ψευδοδεδομένα του πίνακα Location

name	description	latitude	longitude	capacity
Artisan Theater	A theater with a grand and elegant atmosphere, featuring plush seating and intricate details	45.6011009	20.1415393	840
Katie Theater	A theater with seating in small, cozy booths or cabins, creating a more intimate atmosphere for live performances	48.8171378	2.4196694	410
Apollon Theatre	Designed by the famous German architect Ernst Ziller, it was completed in 1872. The Apollon is located east of Georgiou I Square, one of Patras' popular squares. The theatre is a micrograph of the La Scala in Milan and is the oldest existing enclosed theatre of the same era.	38.2465988	21.7354299	300
Madison Square Garden	A multi-purpose arena with a seating capacity of over 20,000, known for hosting major sporting events and concerts	47.2457073	-122.4773432	20000
OAKA Olympic Stadium	A multi-purpose stadium with a seating capacity of over 70,000, located in Athens, Greece, known for hosting major sporting events and concerts	38.0394620	23.78491936	70000

Ο πίνακας δεδομένων Event γεμίζεται επίσης με αποκλειστική χρήση του ChatGPT (βλ. Πίνακα 5).

⁴ Οι συντεταγμένες των Ελληνικών συναυλιακών κέντρων διορθώθηκαν χειροκίνητα για την καλύτερη παρουσίαση της λειτουργίας των φίλτρων.

Πίνακας 5: Μερικά ψευδοδεδομένα του πίνακα Event

name	description	category
Villain, The (Le Vilain)	An action-packed and thrilling performance that will have you rooting for the heroes	Theater
Johnny Tremain	A thought-provoking and meaningful performance that will leave you reflecting and considering new perspectives	Theater
Mitchell	A hilarious and light-hearted performance that will have you laughing out loud	Theater
The Sound of the Forest	Join us for a night of unbridled fun and non-stop hits!	Concert
Music in the Air	Experience the energy and excitement of live music!	Concert

Η αποφυγή χρήσης πραγματικών δεδομένων και η επιστράτευση του μοντέλου μηχανικής μάθησης ChatGPT (που βασίζεται στο GPT 3.5) έγινε, χωρίς βλάβη της γενικότητας, για την εξοικονόμηση χρόνου και την εξοικείωση των συγγραφέων με τις νεότερες τεχνολογίες.

4.2 Δημιουργία Εξαρτώμενων Δεδομένων

Έχοντας δημιουργήσει τώρα τα δεδομένα των πινάκων Category, User, Location και Event περνάμε στους πίνακες που εμφανίζουν εξάρτηση (ξένα κλειδιά) από άλλους πίνακες.

4.2.1 Χωρίς Σύνδεση Στη Βάση

Δύο πίνακές μας, ο πίνακας Seat και ο πίνακας Happening, περιέχουν δεδομένα που δεν απαιτούν πολύπλοκα ερωτήματα από τη βάση (πέρα από τη γνώση δηλαδή των τιμών των άλλων πινάκων) και άρα τα δεδομένα τους μπορούν να δημιουργηθούν, μέσω Python, πριν την ανάπτυξη της διεπαφής εφαρμογών για τη βάση μας (API).

Για τη δημιουργία δεδομένων για τον πίνακα Seat ακολουθήθηκε η λογική του Αλγορίθμου 1 (βλ. αρχείο *generators/seats.py*).

ΑΛΓΟΡΙΘΜΟΣ 1: Δημιουργία Δεδομένων Θέσεων

```

current_id      0
seats           []
for each location do
    numbering_style = choose_random_numbering_style()
    for i from 0 to location.capacity do
        number, type = numbering_style.next()

```

```

        seat = new Seat()
        seat.number = number
        seat.location_id = location.id
        seat.type = type
        seats[current_id] = seat
        current_id++
    end
end

```

Για την καλύτερη παρουσίαση των δυνατοτήτων της βάσης μας υλοποιούμε τρεις διαφορετικούς τρόπους αρίθμησης θέσεων ως παράδειγμα, γνωρίζοντας ότι κάθε χώρος εκδηλώσεων επιλέγει διαφορετική μέθοδο ονοματοδοσίας θέσεων. Μερικά παραδείγματα των παραγόμενων δεδομένων φαίνονται στον Πίνακα 5.

Πίνακας 6: Μερικά ψευδοδεδομένα του πίνακα Seat

id	number	location_id	type
0	FRO-0	1	9
1	REAR-1	1	2
2	MID-2	1	5
3	REAR-3	1	2
411	O241	2	8
412	S242	2	8
413	M243	2	8
2650	0	6	10
2651	1	6	5
2652	2	6	2

Για τη δημιουργία δεδομένων για τον πίνακα Happening ακολουθήθηκε η λογική του Αλγορίθμου 2 (βλ. αρχείο *generators/happenings.py*).

ΑΛΓΟΡΙΘΜΟΣ 2: Δημιουργία Δεδομένων Γεγονότων

```

current_id    0
happenings    []
for each event do
    for i from 0 to random(max=10) do
        location = random_location_based_on_category(event.category)
        happening = new Happening()
        happening.event_id = event.id
        happening.datetime = random_time_in_2023()
        happening.location_id = location.id
        happening.available_seats = location.capacity
        happenings[current_id] = happening
        current_id++
    end
end

```


end

Δημιουργούμε δηλαδή έναν τυχαίο αριθμό Γεγονότων για κάθε Εκδήλωση (από 1 μέχρι 10) με βάση μία από της τοποθεσίες που δημιουργήσαμε προηγουμένως. Μερικά από τα αποτελέσματα φαίνονται στον Πίνακα 6.

Πίνακας 7: Μερικά ψευδοδεδομένα του πίνακα Happening

event_id	datetime	location_id	available_seats
1	2023-04-22 21:08:00	57	690
1	2023-04-27 16:26:00	102	600
2	2023-09-10 21:42:00	18	430
2	2023-12-01 14:26:00	41	280
2	2023-03-27 19:38:00	43	210

4.2.2 Με Σύνδεση Στη Βάση

Για το γέμισμα των δύο τελευταίων πινάκων μας με ψευδοδεδομένα θεωρήθηκε ευκολότερο να χρησιμοποιήσουμε άμεσα τη διεπαφή προγραμματισμού εφαρμογών (API) που είχαμε αναπτύξει για τη βάση μας (βλ. Παράγραφο 5.2). Βασική στρατηγική είναι η εξομοίωση χρηστών που ζητούν τη δημιουργία νέων κρατήσεων και την έκδοση στη συνέχεια των κατάλληλων εισιτηρίων. Ακολουθούν οι δύο αλγόριθμοι που χρησιμοποιήθηκαν για τη δημιουργία των δεδομένων των πινάκων Reservation και Ticket (Αλγόριθμος 3 και 4).

ΑΛΓΟΡΙΘΜΟΣ 3: Δημιουργία Δεδομένων Κρατήσεων

```
app = initiate_connection_to_api()
users = app.get_users()
for each user do
    for i from 0 to random(max=10) do
        reservation = create_random_reservation(user)
        app.add_reservation(reservation)
    end
end
```

Η υπορουτίνα `create_random_reservation` του Αλγορίθμου 3 ζητάει από τη βάση ένα τυχαίο Γεγονός με διαθέσιμα εισιτήρια και επιστρέφει όλες τις πληροφορίες της κράτησης (βλ. και αρχείο “*generators/reservations.py*”)

ΑΛΓΟΡΙΘΜΟΣ 4: Δημιουργία Δεδομένων Εισιτηρίων

```
app = initiate_connection_to_api()
reservations = app.get_reservations()
for each reservation do
    available_seats = app.get_available_seat_ids_for_happening(reservation.happening_id)
```

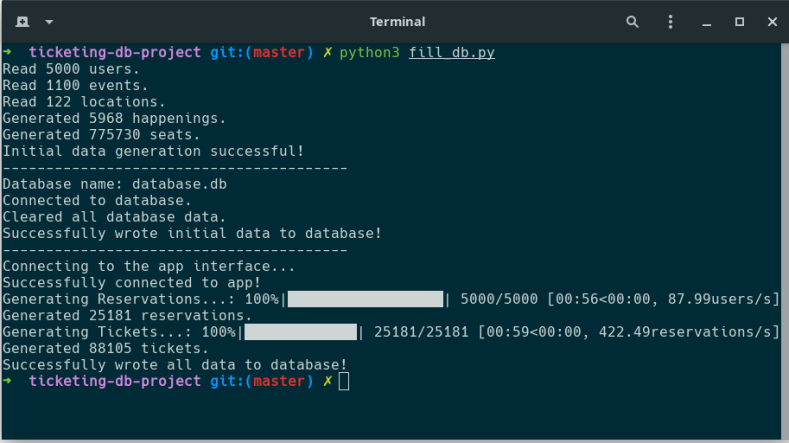
```

    for i from 0 to reservation.tickets_no do
        ticket = new Ticket()
        ticket.name = random_name()
        ticket.reservation_num = reservation.id
        ticket.seat_id = available_seats[i]
        ticket.happening_id = reservation.happening_id
        app.create_ticket(ticket)
    end
end
end

```

Η υλοποίηση του Αλγορίθμου 2 σε Python βρίσκεται στο αρχείο “*generators/reservations.py*”.

Συνδυάζοντας τα modules που αναπτύχθηκαν σε ένα script (“*fill_db.py*”) μπορούμε να δημιουργούμε όποτε χρειάζεται μαζικά όλα τα δεδομένα της βάσης μας. Το αποτέλεσμα εκτέλεσης του προγράμματος φαίνεται στο Σχήμα 4.



```

Terminal
+ ticketing-db-project git:(master) * python3 fill_db.py
Read 5000 users.
Read 1100 events.
Read 122 locations.
Generated 5968 happenings.
Generated 775730 seats.
Initial data generation successful!
-----
Database name: database.db
Connected to database.
Cleared all database data.
Successfully wrote initial data to database!
-----
Connecting to the app interface...
Successfully connected to app!
Generating Reservations...: 100%|██████████| 5000/5000 [00:56<00:00, 87.99users/s]
Generated 25181 reservations.
Generating Tickets...: 100%|██████████| 25181/25181 [00:59<00:00, 422.49reservations/s]
Generated 88105 tickets.
Successfully wrote all data to database!
+ ticketing-db-project git:(master) *

```

Σχήμα 4: Τυπική εκτέλεση του προγράμματος *fill_db.py*

5 ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ

Η τελική εφαρμογή που αναπτύχθηκε για την παρουσίαση της λειτουργίας της βάσης αποτελείται από 3 διακριτά μέρη:

1. Το μέρος που διαχειρίζεται τη διασύνδεση με τη βάση (module connection)
2. Τη διεπαφή προγραμματισμού εφαρμογών - API (module app)
3. Τη γραφική διεπαφή που χρησιμοποιεί ο χρήστης (script gui)

5.1 Σύνδεση Με Τη Βάση

Για τη σύνδεση με τη βάση μας αποφασίσαμε να αναπτύξουμε μία νέα κλάση (Connection) που αποτελεί τη μοναδική μας διέξοδο προς τη βάση (όλα τα ερωτήματα πρέπει να περνούν από αντικείμενα τύπου Connection). Με αυτόν τον τρόπο γίνεται ευκολότερη η αξιολόγηση της απόδοσης και η καταγραφή όλων των παραγόμενων ερωτημάτων προς τη βάση. Τα αντικείμενα τύπου Connection δημιουργούνται με κλήσεις της μορφής:

```
connection = Connection(dbname, log=False)
```

Ενώ απλά ερωτήματα εκτελούνται στη συνέχεια με τον παρακάτω τρόπο:

```
result = connection.execute_and_fetch(query, *args)
```

Ενδιαφέρον έχει επίσης η μέθοδος `insert_to_table` που μας επιτρέπει τη γρήγορη προσθήκη δεδομένων σε έναν πίνακα της βάσης.

5.2 Ανάπτυξη Διεπαφής Εφαρμογών

Επόμενο επίπεδο αφαιρετικότητας αποτελεί η διεπαφή προγραμματισμού εφαρμογών (API). Υλοποιημένη ως μία απλή κλάση (App) η διεπαφή μας επιτρέπει την απόκρυψη πολύπλοκων λογικών από τον προγραμματιστή της τελικής εφαρμογής κάνοντας παράλληλα τον κώδικα πιο ευανάγνωστο. Τα αντικείμενα τύπου App δημιουργούνται και αυτά με κλήσεις της μορφής:

```
app = App(dbname, log=False)
```

Στη συνέχεια, απλά και πολύπλοκα ερωτήματα στη βάση μετατρέπονται σε απλές κλήσεις μεθόδων:

```
users = app.get_users()
ticket_data = app.get_ticket_info(ticket_id)
descriptions = app.get_happening_descriptions(filters)
```

Τα δεδομένα που επιστρέφονται από το App έχουν πάντα τη μορφή λεξικών όταν η μέθοδος αναμένεται να επιστρέψει μία μόνο εγγραφή ή λεξικά λεξικών όταν η μέθοδος αναμένεται να επιστρέψει παραπάνω από μία εγγραφές. Για παράδειγμα, η μεταβλητή `users` μετά την κλήση της `get_users()` όπως φαίνεται παραπάνω θα έχει τη μορφή:

```
users = {
    "up1072772@upnet.gr": {"name": "Konstantinos Gavalas", "password":
        "FunO4CWF5rZ"},
    "up1072711@upnet.gr": {"name": "Jim Grillakis", "password": "Xn929FSSej"},
    ...
}
```

Με αυτόν τον τρόπο ο προγραμματιστής έχει γρήγορη πρόσβαση στα δεδομένα που τον ενδιαφέρουν με συντακτικό της μορφής `table[id][column]`.

5.2.1 Σύστημα Φίλτρων

Στο τρίτο παράδειγμα της προηγούμενης παραγράφου (`descriptions=app.get_happening_descriptions(filters)`) υπονοήθηκε επίσης η ύπαρξη ενός συστήματος φίλτρων. Ένας από τους στόχους της τελικής εφαρμογής στην οποία θα ενταχθεί η βάση είναι οι χρήστες να έχουν τη δυνατότητα να αναζητούν Γεγονότα με τη χρήση διάφορων φίλτρων (π.χ. λεκτικών, χρονικών, χωρικών κλπ).

Προφανώς, ένας τρόπος υλοποίησης της συγκεκριμένης λειτουργίας είναι η εφαρμογή του κάθε φίλτρου με ένα `SELECT` στη βάση και η προσθήκη κάθε επόμενου φίλτρου σαν εμφωλευμένο `SELECT` στο τελικό ερώτημα. Η λύση αυτή ωστόσο απορρίφθηκε νωρίς στη διαδικασία καθώς θεωρήθηκε ότι η βελτιστοποίηση των παραγόμενων ερωτημάτων από την εφαρμογή μας ήταν μείζονος σημασίας. Καταλήξαμε έτσι στην ανάπτυξη του `module filters`.

Η λογική είναι ότι διαφορετικά φίλτρα μπορούν να υλοποιηθούν ως υποκλάσεις της κλάσης `Filter` με τρόπο τέτοιο ώστε να είναι απλός ο συνδυασμός τους για τη δημιουργία του τελικού `query`. Παράδειγμα τρόπου χρήσης του συγκεκριμένου `module` φαίνεται παρακάτω:

```
f1 = filters.HappeningAvailable()
f2 = filters.HappeningTime("2023-01-31")
f3 = filters.HappeningDistance(lat=0, long=0, dist=250)
happenings = app.get_happenings([f1, f2, f3])
```

Στη συγκεκριμένη περίπτωση η μεταβλητή `happenings` θα περιέχει όλα τα Γεγονότα που πληρούν τα παρακάτω φίλτρα:

1. Το Γεγονός έχει διαθέσιμες θέσεις
2. Το Γεγονός συμβαίνει πριν τις 31/01/2023
3. Το Γεγονός συμβαίνει σε απόσταση μικρότερη από 250km από την αρχή των γεωγραφικών συντεταγμένων

Το ερώτημα στη βάση που παράγεται από τη συγκεκριμένη ακολουθία εντολών είναι αυτό που φαίνεται παρακάτω:

```
SELECT ALL * FROM Happening JOIN Location ON Happening.location_id=Location.id WHERE
(available_seats > 0) AND (Happening.datetime < datetime('2023-01-31')) AND ((6371 *
acos(cos(radians(0)) * cos(radians(latitude)) * cos(radians(longitude) - radians(0)) +
sin(radians(0)) * sin(radians(latitude)))) < 250)
```

Με αντίστοιχο τρόπο δημιουργούνται αρκετά ακόμα φίλτρα, ενώ δίνεται επίσης η δυνατότητα στον προγραμματιστή να ορίσει καινούργια (`custom`) φίλτρα καλώντας άμεσα τον `constructor` της κλάσης `Filter`, σε περίπτωση εξειδικευμένων αναγκών σε συγκεκριμένα σημεία του κώδικα της τελικής εφαρμογής.

5.3 Ανάπτυξη Γραφικής Διεπαφής Χρήστη

Τελευταίο επίπεδο αφαιρετικότητας αποτελεί η τελική γραφική διεπαφή που χρησιμοποιεί ο χρήστης. Η διεπαφή επιλέχθηκε να είναι γραφική για την καλύτερη παρουσίαση των δυνατοτήτων της εφαρμογής. Και αυτό το κομμάτι της εφαρμογής είναι υλοποιημένο με χρήση της Python και συγκεκριμένα με εκτεταμένη χρήση της βιβλιοθήκης PySimpleGUI.

Η εφαρμογή μας αποτελείται από 2 απλοϊκές σελίδες, μία στην οποία ο χρήστης διαλέγει την Εκδήλωση και το Γεγονός που τον ενδιαφέρει και μία στην οποία κάνει την τελική κράτηση των εισιτηρίων του. Οι δύο σελίδες αποτελούν δύο διαφορετικά columns του τελικού layout ένα εκ των δύο είναι πάντα κρυμμένο.

Λόγω της καλής υλοποίησης των δύο άλλων τμημάτων της εφαρμογής, η γραφική διεπαφή υλοποιείται σε λιγότερο από 350 γραμμές κώδικα.

Για τυπικά παραδείγματα χρήσης της εφαρμογής βλ. Παράρτημα Α.2.

6 ΕΛΕΓΧΟΣ ΚΑΙ ΑΞΙΟΛΟΓΗΣΗ

Ο έλεγχος και η αξιολόγηση της βάσης μας έγινε, με βάση τις προδιαγραφές που τέθηκαν στην αρχή της εργασίας μας, γύρω από τρεις βασικούς άξονες:

1. Η βάση πρέπει να αποτρέπει την δημιουργία κακοσχηματισμένων δεδομένων αποτρέποντας τον προγραμματιστή της εφαρμογής από το να κάνει κοστοβόρα λάθη.
2. Η βάση πρέπει να είναι ικανοποιητικά γρήγορη για την κάλυψη των αναγκών της εφαρμογής.
3. Η βάση και η διεπαφή εφαρμογών που αναπτύχθηκε πρέπει να επιτρέπουν μεγάλη ευελιξία στον προγραμματιστή της εφαρμογής.

6.1 Αξιολόγηση Ελέγχων Ακεραιότητας

Για τον έλεγχο των κανόνων ελέγχου ακεραιότητας που αναπτύχθηκαν (βλ. Παράγραφο 3) έγιναν πολυάριθμες (χειροκίνητες) δοκιμές. Μερικές από αυτές παρουσιάζονται στο αρχείο *tests.txt* που συνοδεύει την παρούσα έκθεση. Ως παράδειγμα παρουσιάζεται εδώ η ακολουθία ερωτημάτων που αναπτύχθηκε για τον έλεγχο του `trigger check_ticket_count_before_insert`:

```
INSERT INTO Reservation(number, user_email, receipt_num, datetime, tickets_no,
happening_id)
```

```
VALUES(99999, 'mmckinlay0@vinaora.com', 12345, '2022-01-01 10:00:00', 2, 1);
```

```
-- Should work
```

```
INSERT INTO Ticket(name, reservation_num, seat_id, happening_id)
```

```
VALUES('Kostas', 99999, 1120, 1), ('Dimitris', 99999, 1121, 1);
```

```
-- Should produce an error
```

```
INSERT INTO Ticket(name, reservation_num, seat_id, happening_id)
```

```
VALUES('Giorgos', 99999, 1122, 1);
```

Δημιουργούμε αρχικά μία νέα εγγραφή στον πίνακα κρατήσεων Reservation που αντιστοιχεί σε 2 εισιτήρια. Στη συνέχεια, δημιουργούμε δύο καινούργια εισιτήρια που αντιστοιχούν σε αυτήν την κράτηση χωρίς κάποιο μήνυμα σφάλματος από την βάση. Προσπαθούμε τέλος να εκδώσουμε άλλο ένα εισιτήριο για την συγκεκριμένη κράτηση. Η βάση ωστόσο αυτήν την φορά αρνείται το αίτημα με μήνυμα λάθους “Number of tickets exceeds reservation ticket count” καθώς ήδη έχει εκδοθεί ο κατάλληλος αριθμός εισιτηρίων.

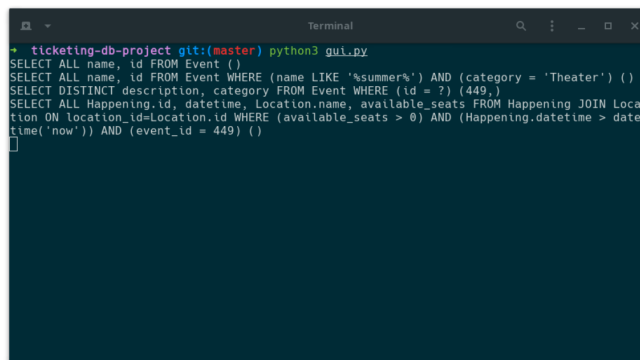
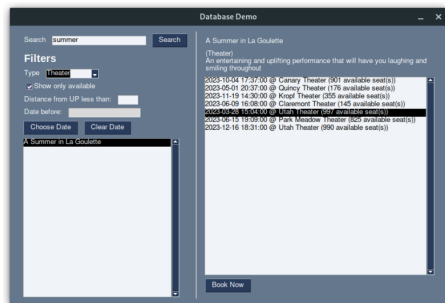
Λόγω της ορθής λειτουργίας κανόνων όπως αυτού στο επίπεδο της SQL, πράγματι αποφεύχθηκε μεγάλος αριθμός λογικών σφαλμάτων προγραμματισμού κατά την διαδικασία ανάπτυξης του API αλλά και της γραφικής διεπαφής. Η μεγάλη εμπιστοσύνη στη σωστή διατήρηση της ακεραιότητας των δεδομένων από την βάση, μας επέτρεψε επίσης μεγαλύτερη ελευθερία στον πειραματισμό και την δοκιμή διαφόρων τεχνικών δημιουργίας ερωτημάτων προς αυτήν.

6.2 Αξιολόγηση Ταχύτητας

Για την βελτιστοποίηση της ταχύτητας της τελικής μας εφαρμογής εστιάσαμε σε δύο βασικά ζητήματα:

1. Την ελαχιστοποίηση των ερωτημάτων που δημιουργούνται προς την βάση και
2. Την μεγιστοποίηση της ταχύτητας απόκρισης της βάσης

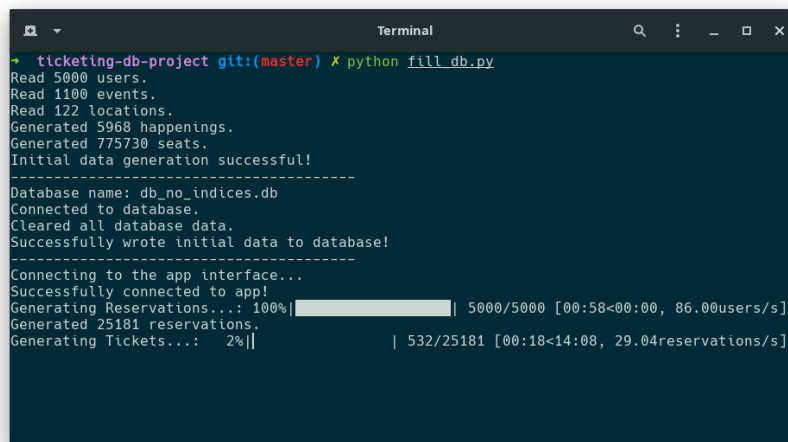
Στην ελαχιστοποίηση των ερωτημάτων βασικό ρόλο διαδραμάτισε η δημιουργία του συστήματος φίλτρων (βλ. Παράγραφο 5.2.1). Για τον έλεγχο των παραγόμενων ερωτημάτων προστέθηκε απλοϊκός κώδικας παρακολούθησης (logging) στις μεθόδους insert_to_table και execute_and_fetch της κλάσης Connection. Έτσι κατά την εκτέλεση του προγράμματος (π.χ. της γραφικής διεπαφής) εμφανίζονται στην κονσόλα όλα τα ερωτήματα που γίνονται προς την βάση (βλ. Σχήμα 5).



Σχήμα 5: Η γραφική διεπαφή και τα αντίστοιχα ερωτήματα προς την βάση

Με αυτήν την μέθοδο αξιολογούμε το πλήθος των παραγόμενων ερωτημάτων που, μετά από αρκετές αλλαγές, κρίνεται αρκετά ικανοποιητικό.

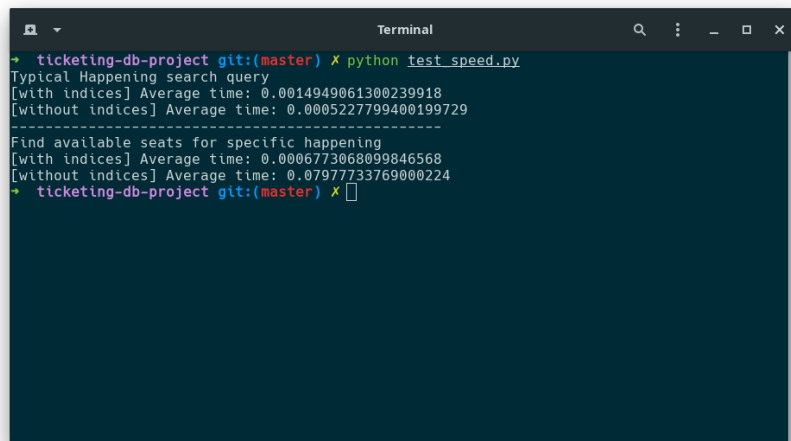
Στη βελτιστοποίηση της ταχύτητας απόκρισης της βάσης βασικό ρόλο διαδραμάτισε η δημιουργία των ευρετηρίων (βλ. Παράγραφο 3.3). Πρώτη εμφανής διαφορά της απόδοσης της βάσης μας πριν και μετά την δημιουργία τους ήταν κατά την δημιουργία των δεδομένων (μέσω του script *fill_db.py*). Στην περίπτωση χωρίς τα ευρετήρια (βλ. Σχήμα 6) η ταχύτητα, ιδιαίτερα δημιουργίας εισιτηρίων, είναι κατά πολύ μικρότερη από αυτήν στην περίπτωση ύπαρξης ευρετηρίων (βλ. Σχήμα 4).



Σχήμα 6: Τυπική εκτέλεση του προγράμματος *fill_db.py* χωρίς την ύπαρξη ευρετηρίων

Για τον έλεγχο της επιρροής των ευρετηρίων σε πιο συνηθισμένα ερωτήματα, αναπτύσσουμε ένα μικρό πρόγραμμα σε Python που συνδέεται σε δύο βάσεις (μία χωρίς και μία με ευρετήρια αλλά ίδια δεδομένα) και

εκτελεί πολλές φορές δύο τυπικά ερωτήματα σε αυτές μέσω της διεπαφής. Τα αποτελέσματα φαίνονται στο Σχήμα 7.



```
Terminal
ticketing-db-project git:(master) * python test_speed.py
Typical Happening search query
[with indices] Average time: 0.0014949061300239918
[without indices] Average time: 0.0005227799400199729
-----
Find available seats for specific happening
[with indices] Average time: 0.0006773068099846568
[without indices] Average time: 0.07977733769000224
ticketing-db-project git:(master) *
```

Σχήμα 7: Τυπικοί (μέσοι) χρόνοι για την εκτέλεση δύο βασικών ερωτημάτων σε βάση με και χωρίς ευρετήρια

Μη αναμενόμενα, η εκτέλεση ενός απλού ερωτήματος εύρεσης Γεγονότων με βάση κάποια φίλτρα (σαν το δεύτερο ερώτημα που παρουσιάζεται στο Παράρτημα Α.2) γίνεται ταχύτερα στην περίπτωση έλλειψης των ευρετηρίων (κατά μία τάξη μεγέθους). Ερωτήματα σαν αυτά δημιουργούνται πολύ συχνά προς την βάση μας και η βελτιστοποίησή τους θα μας απασχολούσε σίγουρα αρκετά σε επόμενες αναθεωρήσεις.

Στην αντίθετη περίπτωση βέβαια, για την εκτέλεση πολυπλοκότερων ερωτημάτων (όπως για την εύρεση όλων των διαθέσιμων θέσεων για ένα συγκεκριμένο γεγονός), η διαφορά που κάνουν τα ευρετήρια είναι και πάλι αισθητή με χρόνους μικρότερους κατά δύο τάξεις μεγέθους.

Συνολικά, η επιρροή των ευρετηρίων στα ερωτήματα μας είχε πολύ θετικό χαρακτήρα καθώς μόνον εξαιτίας αυτών μας επιτράπηκε η δημιουργία μεγάλου αριθμού δεδομένων (ιδιαίτερα για τον πίνακα tickets).

6.3 Αξιολόγηση Ευελιξίας

Η ευελιξία της βάσης και του κώδικα που αναπτύχθηκε για την διαχείριση αυτής κρίθηκε εξ αρχής βασικό κριτήριο για την πληρότητα της τελικής λύσης. Κάποια μέτρα που λήφθηκαν (κυρίως κατά την ανάπτυξη της διεπαφής προγραμματισμού εφαρμογών) είναι:

1. Η επιλογή χρήσης αντικειμενοστραφούς προγραμματισμού
2. Η δομή του κώδικα σε πολλά επίπεδα αφαιρετικότητας και
3. Η ανάπτυξη του επεκτάσιμου συστήματος φίλτρων

Η δημιουργία του κώδικα με την ευελιξία σαν βασικό κριτήριο από το πρώτο βήμα, μας επέτρεψε τον ταχύτερο πειραματισμό σε επόμενα στάδια γεγονός που αν και μη-μετρήσιμο συνέβαλλε κατά πολύ στην τελική μορφή της εργασίας μας.

A ΠΑΡΑΡΤΗΜΑΤΑ

A.1 Οδηγίες Εγκατάστασης

1. Ο απαραίτητος κώδικας για την εκτέλεση της εφαρμογής βρίσκεται στο Github. Για την αντιγραφή του εκτελούμε:

```
git clone https://github.com/GavalasDev/ticket-booking-app
cd ticket-booking-app
```

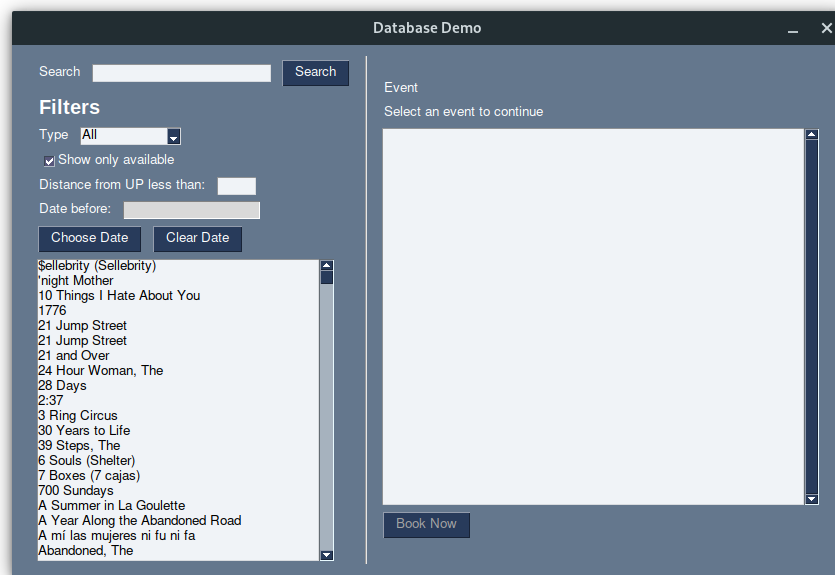
2. Οι απαραίτητες βιβλιοθήκες αναφέρονται στο αρχείο *requirements.txt*. Για την γρήγορη εγκατάστασή τους εκτελούμε την εντολή:

```
pip install -r requirements.txt
```

3. Για την δημιουργία της βάσης έχουμε δύο επιλογές:
 - a. Την χρήση της προϋπάρχουσας βάσης ("*database_filled.db*") που συνοδεύει την αναφορά και περιέχει ήδη όλους τους πίνακες και τα δεδομένα που χρησιμοποιήσαμε μετονομάζοντας την σε *database.db* ή
 - b. Την δημιουργία καινούργιας βάσης για την δοκιμή και του προγράμματος *fill_db.py* (προτεινόμενο):
 - i. Ανοίγοντας το *DB Browser for SQLite* επιλέγουμε File > Import > Database from SQL file...
 - ii. Στο παράθυρο που ανοίγει επιλέγουμε το αρχείο *schema.sql* που συνοδεύει την παρούσα έκθεση.
 - iii. Αποθηκεύουμε την βάση μας ως *database.db* στον ίδιο φάκελο με τα αρχεία *fill_db.py* και *gui.py*.
 - iv. Χρησιμοποιώντας την συντόμευση Ctrl+S αποθηκεύουμε όλες τις αλλαγές στον δίσκο.
 - v. Από την γραμμή εντολών εκτελούμε, στον ίδιο φάκελο με την βάση, το αρχείο *fill_db.py*. Μετά από λίγο χρόνο πρέπει να βλέπουμε εικόνα παρόμοια με αυτή του Σχήματος 4.
4. Για την εκτέλεση της εφαρμογής τώρα εκτελούμε το αρχείο *gui.py*. Η εικόνα που βλέπουμε πρέπει να είναι παρόμοια με το Σχήμα 8.

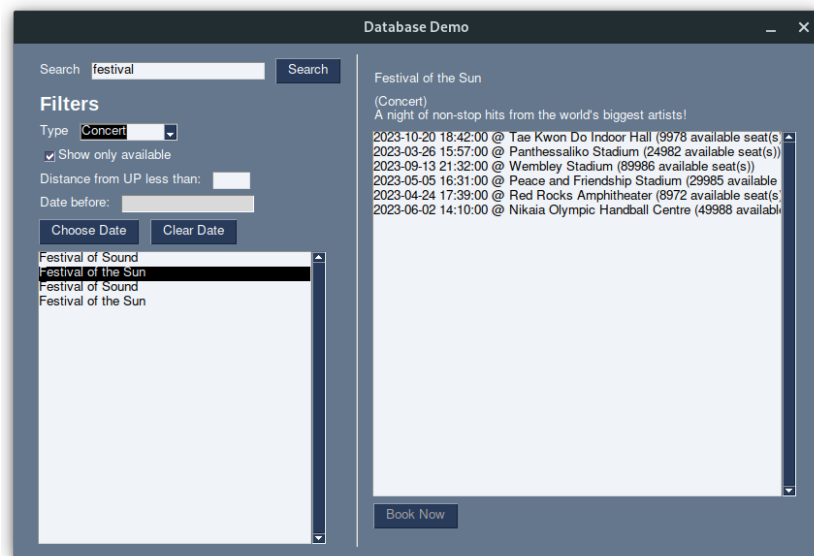
A.2 Παραδείγματα Χρήσης του Προγράμματος

Η αρχική σελίδα της εφαρμογής παρουσιάζει τις διαθέσιμες Εκδηλώσεις και επιτρέπει στον χρήστη την εφαρμογή διάφορων φίλτρων (βλ. Σχήμα 8).



Σχήμα 8: Η αρχική κατάσταση της εφαρμογής

Αναζητώντας μία συγκεκριμένη συναυλία ο χρήστης επιλέγει τον τύπο “Concert” και πληκτρολογεί στο πεδίο “Search”. Επιλέγοντας την Εκδήλωση που τον ενδιαφέρει εμφανίζονται στο δεξί παράθυρο τα διάφορα Γεγονότα (βλ. Σχήμα 9).



Σχήμα 9: Αναζήτηση Εκδηλώσεων

Για να περιορίσει τα αποτελέσματα ο χρήστης επιλέγει να δει μόνο εκδηλώσεις που βρίσκονται σε απόσταση μικρότερη των 300 χ.μ. από το Πανεπιστήμιο Πατρών⁵ και ημερομηνία πριν το τέλος του θερινού εξαμήνου (βλ. Σχήμα 10).

Αφού ο χρήστης επιλέξει το Γεγονός που τον ενδιαφέρει πατώντας το κουμπί "Book Now" μεταφέρεται στη σελίδα δημιουργίας Κρατήσεων, όπου συμπληρώνει τα στοιχεία των εισιτηρίων που θέλει να αγοράσει (βλ. Σχήμα 11).

Τέλος πατώντας το κουμπί "BOOK" ενημερώνεται ότι τα 2 εισιτήρια του εκδόθηκαν με επιτυχία και μεταφέρεται και πάλι στην αρχική σελίδα της εφαρμογής. Τα εισιτήρια του βρίσκονται (σε μορφή pdf) στον ίδιο φάκελο με αυτόν της εφαρμογής (βλ. Σχήμα 12). Προφανώς στην πραγματική εφαρμογή θα μεσολαβούσε και η διαδικασία επιλογής θέσεων και πληρωμής του αντίστοιχου ποσού μέσω εξωτερικού παρόχου.

⁵ Χάριν απλότητας της γραφικής διεπαφής η απόσταση επιλέγεται πάντα από το Πανεπιστήμιο Πατρών αν και η διεπαφή εφαρμογών επιτρέπει την επιλογή οποιασδήποτε τοποθεσίας ως αναφορά.

Database Demo

Search:

Filters

Type:

☒ Show only available

Distance from UP less than:

Date before:

Festival of Sound
Festival of the Sun
Festival of Sound
Festival of the Sun

Festival of the Sun
(Concert)
A night of non-stop hits from the world's biggest artists!
2023-03-26 15:57:00 @ Panthessaliko Stadium (24982 available seat(s))
2023-05-05 16:31:00 @ Peace and Friendship Stadium (29985 available)

Σχήμα 10: Φιλτράρισμα γεγονότων

Database Demo

←

Festival of the Sun

(Concert)
A night of non-stop hits from the world's biggest artists!

Time: 2023-05-05 16:31:00
Place: Peace and Friendship Stadium

Available seat(s): 29985

Number of tickets:

[Ticket #1] Name:

[Ticket #2] Name:

Σχήμα 11: Συμπλήρωση στοιχείων Εισιτηρίων



Τμήμα Ηλεκτρολόγων Μηχανικών
& Τεχνολογίας Υπολογιστών



Name: Dimitrios Grillakis

Event: Festival of the Sun

Date: 2023-05-05 16:31:00

Seat: E18



Τμήμα Ηλεκτρολόγων Μηχανικών
& Τεχνολογίας Υπολογιστών



Name: Konstantinos Gavalas

Event: Festival of the Sun

Date: 2023-05-05 16:31:00

Seat: Y17

Σχήμα 12: Τα εισιτήρια που δημιουργήθηκαν από την παραπάνω διαδικασία.
Οι κωδικοί QR αντιστοιχούν στο id του εισιτηρίου στη βάση επιτρέποντας τον γρήγορο έλεγχο τους.

Ενδεικτικά κάποια ερωτήματα που δημιουργήθηκαν από την εφαρμογή προς την βάση φαίνονται παρακάτω:

1. Για την αναζήτηση Εκδήλωσης:

```
SELECT ALL name, id FROM Event WHERE (name LIKE '%festival%') AND (category = 'Concert')
```

2. Για την αναζήτηση Γεγονότος:

```
SELECT ALL Happening.id, datetime, Location.name, available_seats FROM Happening JOIN  
Location ON Happening.location_id=Location.id WHERE (available_seats > 0) AND ((6371 *  
acos(cos(radians(38.289365)) * cos(radians(latitude)) * cos(radians(longitude) -  
radians(21.784715)) + sin(radians(38.289365)) * sin(radians(latitude)))) < 300) AND
```

```
(Happening.datetime > datetime('now')) AND (Happening.datetime < datetime('2023-06-02  
13:03:20')) AND (event_id = 1022)
```

3. Για την δημιουργία κράτησης και την έκδοση εισιτηρίων:

```
INSERT INTO Reservation VALUES (?, ?, ?, ?, ?, ?) [None, 'mmckinlay0@vinaora.com', 10780,  
datetime.datetime(2023, 1, 7, 12, 54, 39, 544965), 2, 5535]
```

```
INSERT INTO Ticket VALUES (?, ?, ?, ?, ?) [None, 'Konstantinos Gavalas', 25182, 470745,  
5535]
```

```
INSERT INTO Ticket VALUES (?, ?, ?, ?, ?) [None, 'Dimitrios Grillakis', 25182, 470746, 5535]
```