ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ
ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Βάσεις Δεδομένων

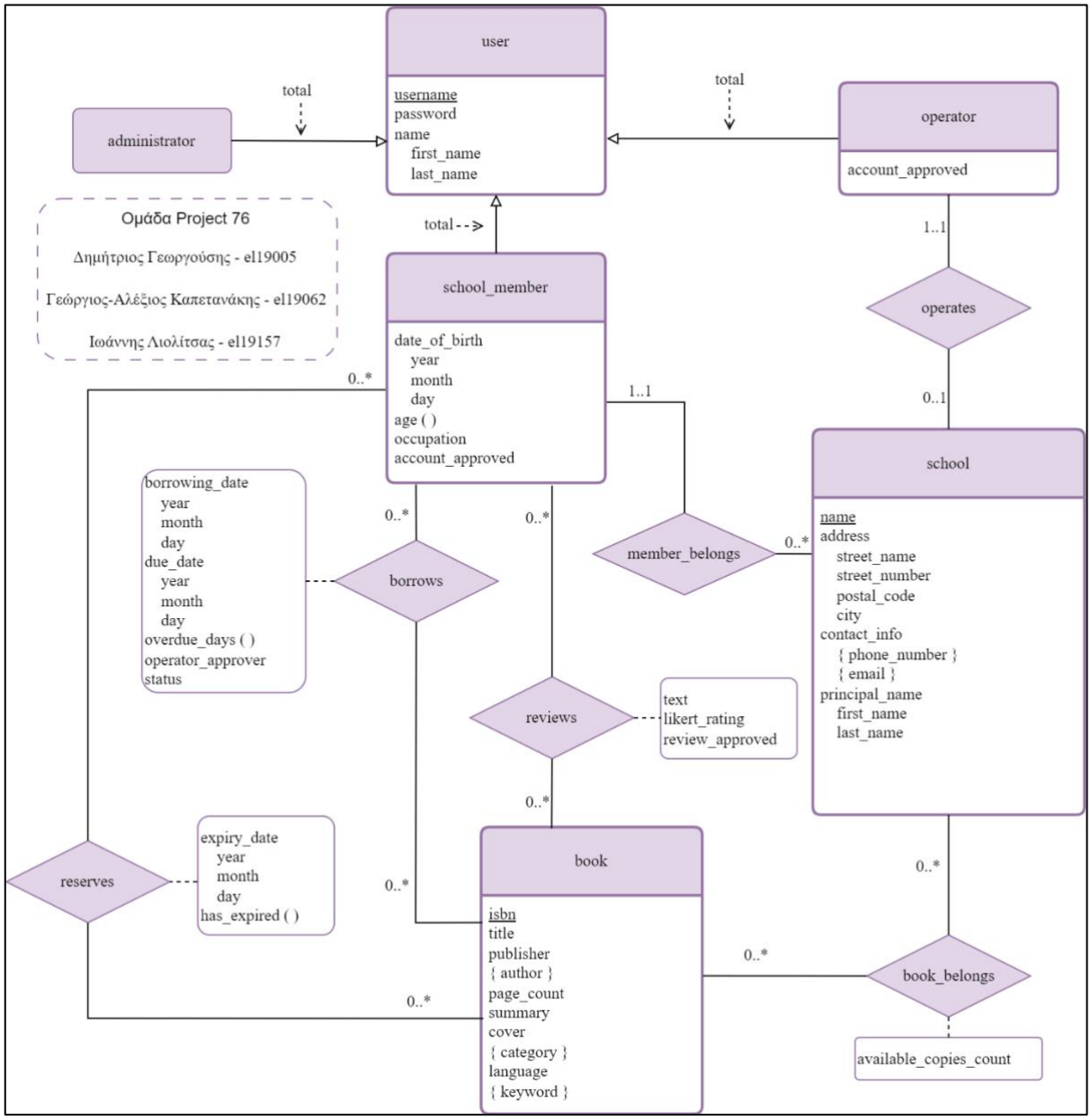
Αναφορά Εξαμηνιαίας Εργασίας Α.Ε. 2022-2023

Ομάδα Project 76

Δημήτριος Γεωργούσης, ΑΜ 03119005
Ιωάννης Λιολίτσας, ΑΜ 03119157
Γεώργιος-Αλέξιος Καπετανάκης, ΑΜ 03119062

1.1. ER & Σχεσιακό διάγραμμα της ΒΔ με αιτιολόγηση και ευρετήρια

ER:



Αιτιολόγηση:

Οντότητες:

- Η βάση μας χρειάζεται μία ‘Σχολική Μονάδα’, οπότε ορίζουμε την οντότητα ‘school’, με διαφοροποίηση από τα ζητούμενα της εκφώνησης ότι δεν υπάρχει το πεδίο του ‘Χειριστή Σχολικής Βιβλιοθήκης’ αφού προκύπτει άμεσα από την σχέση ‘operates’. Επιτρέπουμε πολλαπλά τηλέφωνα και emails.
- Χρειαζόμαστε ‘Βιβλία’, οπότε ορίζουμε την οντότητα ‘book’. Επιτρέπουμε πολλαπλές κατηγορίες και λέξεις-κλειδιά κάνοντας τα αντίστοιχα πεδία multivalued.
- Δημιουργούμε μία γενική οντότητα ‘user’ για τους χρήστες της εφαρμογής μας η οποία εξειδικεύεται στους ‘administrator’, ‘operator’, ‘school_member’ που είναι οι τρεις κύριες κατηγορίες χρηστών της εφαρμογής μας.

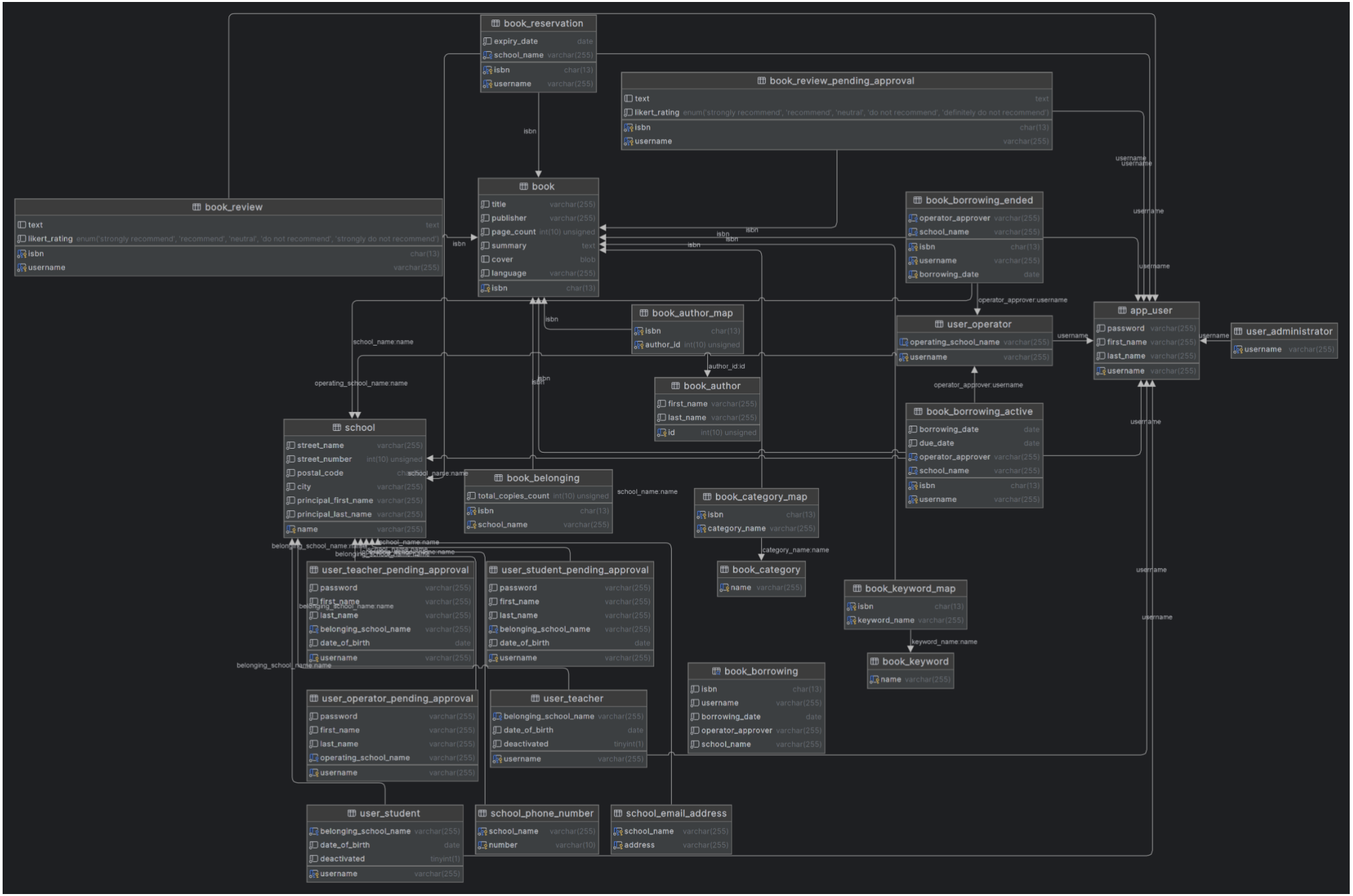
Λειτουργίες:

- Ο ‘Κεντρικός διαχειριστής’ ή, αλλιώς, η οντότητα ‘administrator’ πραγματοποιεί διάφορες ενέργειες στη βάση μέσω της εφαρμογής, οπότε δεν τις αποτυπώνουμε ως σχέσεις στο ER διάγραμμα.
- Με παρόμοια λογική δραστηριότητες που επιτρέπονται στους χρήστες, ανεξαρτήτως κατηγορίας, πάνω στα δεδομένα της βάσης και σχετίζονται με την εκτέλεση ερωτημάτων ή/και CRUD λειτουργιών δεν αποτυπώνονται στο διάγραμμα.
- Για την διευκόλυνση όμως της εκτέλεσης των λειτουργιών αυτών έχουν συμπεριληφθεί πεδία όπως το ‘occupation’ στην οντότητα ‘school_member’ που διαχωρίζει τους ‘teacher’ και τους ‘student’ καθώς και τα διάφορα πεδία σχετικά με την έγκριση (πεδία που περιέχουν λέξεις όπως approved ή approver) σε οντότητες και σχέσεις και παίρνουν τιμές από τους αντίστοιχους υπεύθυνους.

Σχέσεις:

- Ο ‘Υπεύθυνος Χειριστής’ ή ‘operator’ ενός σχολείου λειτουργεί ‘operates’ την αντίστοιχη σχολική μονάδα. Η σχέση είναι one-to-one και θεωρούμε ότι μπορεί κάποιο σχολείο να μην έχει λάβει operator ακόμα.
- Ένα ‘Μέλος Σχολικής Μονάδας’ ή ‘school_member’ σε ένα σχολείο μέσω της σχέσης ‘member_belongs’. Η σχέση αυτή λέει ότι κάθε ‘school_member’ ανήκει σε ένα σχολείο και ότι κάθε σχολείο μπορεί να έχει πολλά ‘school_member’. Είναι, δηλαδή, many-to-one.
- Οι σχέσεις ‘reserves’, ‘borrows’, ‘reviews’ προκύπτουν εύκολα από τα δεδομένα της εκφώνησης. Έχουμε συμπεριλάβει και ορισμένα παραγόμενα πεδία, ώστε να ταιριάζουν με ορισμένα ζητούμενα, όπως τα πεδία `borrows`.`overdue_days` και `reserves`.`has_expired`. Σχετικά με τα borrowings και τα reservations υπάρχουν, επιπλέον, περιορισμοί (π.χ. μαθητές μέχρι 2, καθηγητές μέχρι 1, δεν γίνεται να κάνει reservation ή borrowing αν έχεις overdue_borrowing) τα οποία ενσωματώνονται στο σύστημά μας αργότερα και δεν φαίνονται στο ER διάγραμμα.
- Τέλος, υπάρχει και η σχέση ‘book_belongs’ η οποία δείχνει ποιο βιβλίο ανήκει σε ποιο σχολείο και σε πόσα αντίτυπα.

Παρουσιάζουμε το Σχεσιακό Διάγραμμά μας:


book_reservation
expiry_date date
school_name varchar(255)
isbn char(13)
username varchar(255)
book_review_pending_approval
text text
likert_rating enum('strongly recommend', 'recommend', 'neutral', 'do not recommend', 'definitely do not recommend')
isbn char(13)
username varchar(255)
book_review
text text
likert_rating enum('strongly recommend', 'recommend', 'neutral', 'do not recommend', 'strongly do not recommend')
isbn char(13)
username varchar(255)
book
title varchar(255)
publisher varchar(255)
page_count int(10) unsigned
summary text
cover blob
language varchar(255)
isbn char(13)
book_borrowing_ended
operator_approver varchar(255)
school_name varchar(255)
isbn char(13)
username varchar(255)
borrowing_date date
book_author_map
isbn char(13)
author_id int(10) unsigned
book_author
first_name varchar(255)
last_name varchar(255)
id int(10) unsigned
user_operator
operating_school_name varchar(255)
username varchar(255)
app_user
password varchar(255)
first_name varchar(255)
last_name varchar(255)
username varchar(255)
user_administrator
username varchar(255)
book_borrowing_active
borrowing_date date
due_date date
operator_approver varchar(255)
school_name varchar(255)
isbn char(13)
username varchar(255)
school
street_name varchar(255)
street_number int(10) unsigned
postal_code char
city varchar(255)
principal_first_name varchar(255)
principal_last_name varchar(255)
name varchar(255)
book_belonging
total_copies_count int(10) unsigned
isbn char(13)
school_name varchar(255)
book_category_map
isbn char(13)
category_name varchar(255)
book_category
name varchar(255)
book_keyword_map
isbn char(13)
keyword_name varchar(255)
book_keyword
name varchar(255)
user_teacher_pending_approval
password varchar(255)
first_name varchar(255)
last_name varchar(255)
belonging_school_name varchar(255)
date_of_birth date
username varchar(255)
user_student_pending_approval
password varchar(255)
first_name varchar(255)
last_name varchar(255)
belonging_school_name varchar(255)
date_of_birth date
username varchar(255)
user_operator_pending_approval
password varchar(255)
first_name varchar(255)
last_name varchar(255)
operating_school_name varchar(255)
username varchar(255)
user_teacher
belonging_school_name varchar(255)
date_of_birth date
deactivated tinyint(1)
username varchar(255)
book_borrowing
isbn char(13)
username varchar(255)
borrowing_date date
operator_approver varchar(255)
school_name varchar(255)
user_student
belonging_school_name varchar(255)
date_of_birth date
deactivated tinyint(1)
username varchar(255)
school_phone_number
school_name varchar(255)
number varchar(10)
school_email_address
school_name varchar(255)
address varchar(255)
isbn
username
school_name:name
operating_school_name:name
author_id:id
operator_approver:username
category_name:name
keyword_name:name

Αιτιολόγηση:
Οι οντότητες του ER έχουν μετατραπεί σε relations ακολουθώντας τις αντίστοιχες πρακτικές. Αξίζει να σημειώσουμε ότι στο 'school' τα multivalued πεδία 'phone_number' και 'email' μετατράπηκαν σε relations 'school_phone_number' και 'school_email_address' και στο 'book' τα 'author', 'category', 'keyword' μετατράπηκαν σε αντίστοιχα relations. Το πεδίο 'occupation' του 'school_member' οδήγησε στη δημιουργία των 'user_teacher', 'user_student' διακριτών relations. Η σχέση 'operates' μιας και είναι one-to-one ενσωματώθηκε ως πεδίο στον 'operator' και παρομοίως η σχέση 'member_belongs'.

Επίσης, σχέσεις με πεδία ή σχέσεις many-to-many μετατράπηκαν σε relations. Δημιουργήθηκαν relations για την ξεχωριστή αποθήκευση/χειρισμό οντοτήτων που βρίσκονται σε αναμονή π.χ. 'user_student_pending_approval' ή 'book_review_pending_approval'.

Τέλος, η σχέση 'borrows' διαχωρίστηκε σε δύο relations ένα για τους ενεργούς (active) και ένα για τους ολοκληρωμένους (ended) δανεισμούς.

Περιορισμοί:
Στη σημείο αυτό έχουμε δείξει το Σχεσιακό Διάγραμμα, οπότε μπορούμε να αναφερθούμε στους περιορισμούς. Οι περιορισμοί κλειδιών και αναφορικής ακεραιότητας φαίνεται από τα indexes και τα foreign keys τόσο από το διάγραμμα όσο και από τα tables που δημιουργούμε στο script που παράγει την βάση μας παρακάτω. Επίσης, στα tables μπορούμε να δούμε και τους περιορισμούς πεδίου τιμών για τα διάφορα πεδία που λαμβάνουν ιδιαίτερες τιμές.
Οι περιορισμοί στα δεδομένα φαίνονται και από τις επιπλέον λειτουργίες που έχουμε βάλει στα 'on delete' και 'on update' των foreign keys αρκετών από τα relations.
Επιπλέον, περιορισμοί στα δεδομένα της βάσης μας φαίνονται και επιβάλλονται από τα triggers και events που ορίζουμε.

Triggers:
- Ένας καθηγητής δεν μπορεί να είναι μαθητής.
- Ένας μαθητής δεν μπορεί να είναι καθηγητής ή χειριστής.
- Μπορούμε να κάνουμε κράτηση μόνο αν:
    - Δεν έχουμε κάνει ήδη κράτηση για αυτό το βιβλίο
    - Δεν έχουμε ήδη στη κατοχή μας αυτό το βιβλίο
    - Δεν έχουμε αργήσει να επιστρέψουμε κάποιο βιβλίο
    - Δεν έχουμε υπερβεί το όριο κρατήσεων/δανεισμών
- Μπορούμε να δανειστούμε βιβλίο μόνο αν:
    - Δεν έχουν εξαντληθεί τα αντίτυπά του στη βιβλιοθήκη
    - Δεν έχουμε αργήσει να επιστρέψουμε κάποιο βιβλίο
    - Δεν έχουμε υπερβεί το όριο κρατήσεων/δανεισμών
- Όταν δανειζόμαστε ένα βιβλίο σβήνουμε την αντίστοιχη κράτηση
- Όταν επιστρέφουμε ένα βιβλίο σβήνουμε τον ενεργό δανεισμό
- Όταν εγκρίνουμε έναν νέο καθηγητή/μαθητή/χειριστή σβήνουμε το αντίστοιχο row από τον πίνακα αναμονής (pending table).

Events:
- Κάθε μέρα σβήνουμε αυτόματα τις κρατήσεις που έληξαν.

Ευρετήρια (Δείκτες ή Indexes):
Η βάση δεδομένων μας είναι MariaDB και χρησιμοποιεί το προεπιλεγμένο σύστημα μνήμης της MariaDB, το InnoDB. Το σύστημα μνήμης αυτό δεικτοδοτεί οποιοδήποτε πεδίο χρησιμοποιεί τους όρους 'KEY', 'key', 'INDEX', 'index' (οι οποίοι είναι ανταλλάξιμοι μεταξύ τους στη MariaDB).
Τα primary keys και τα foreign keys δεικτοδοτούνται αυτόματα, αν δεν δημιουργήσουμε ρητά δείκτες για αυτά. Παρ' όλα αυτά σε κάποια σημεία έχουμε διαλέξει να ορίζουμε δείκτες αντί να αφήνουμε το σύστημα μνήμης να πραγματοποιήσει την ενέργεια αυτή για λογαριασμό μας.

Δεικτοδοτούνται, λοιπόν, πεδία που αποτελούν κλειδιά (foreign και μη) διότι χρησιμοποιούνται αρκετά και στα queries μας. Έτσι, βελτιώνεται η απόδοση των queries αυτών.

Ορισμένα από τα queries μας χειρίζονται πεδία όπως τίτλους βιβλίων, ημερομηνίες έναρξης δανεισμού κ.ά. τα οποία δεν έχουμε δεικτοδοτήσει. Η επιλογή μας αυτή βασίζεται στο γεγονός ότι η ύπαρξη δεικτών για κάθε δυνατό πεδίο που μετέχει σε κάποιο ερώτημα διογκώνει και περιπλέκει πολύ το σύστημά μας και τα ερωτήματα πάνω σε τέτοια πεδία δεν έχουν μεγάλη συχνότητα χρήσης, οπότε δικαιολογείται η 'ζημία' απόδοσης αυτών λόγω της απουσίας δεικτών.

Τα ευρετήρια φαίνονται στα scripts σχετικά με την δημιουργία των relations της βάσης μας, που παρουσιάζονται παρακάτω.

Λοιποί Περιορισμοί:
Οι περιορισμοί αυτοί εφαρμόζονται όχι άμεσα από τη βάση αλλά από την εφαρμογή.
Ένας καθηγητής δημιουργεί κατευθείαν review για ένα βιβλίο αν το επιλέξει, ενώ ένας μαθητής θα δημιουργήσει review σε κατάσταση αναμονής έγκρισης, τα οποία θα ελέγξει κάποια στιγμή ο αρμόδιος χειριστής.

Η εφαρμογή μπορεί να χρησιμοποιηθεί μόνο από μη απενεργοποιημένους (non deactivated) λογαριασμούς.

Πριν πραγματοποιηθεί δανεισμός από καθηγητή ή μαθητή δημιουργείται πάντα πρώτα μια κράτηση την οποία πρέπει να εγκρίνει ο αντίστοιχος διαχειριστής.

1.2. DDL και DML script

schema.sql

```sql
drop schema if exists slms; -- school library management system
create database slms
    character set = "utf8mb4"
    collate = "utf8mb4_unicode_ci";
use slms;

-- school related tables

create table school
(
    name                varchar(255) not null,
    street_name         varchar(255) not null,
    street_number       int unsigned not null,
    postal_code         char(5)      not null check (postal_code regexp '^[1-9][0-9]{4}$'),
    city                varchar(255) not null,
    principal_first_name varchar(255) not null,
    principal_last_name  varchar(255) not null,
    primary key (name)
);

create table school_email_address
(
    school_name varchar(255) not null,
    address     varchar(255) not null check (address like '_%@_%._%'),
    primary key (school_name, address),
    constraint fk_school_email_school_name
        foreign key (school_name) references school (name)
            on delete cascade
            on update cascade
);

create table school_phone_number
(
    school_name varchar(255) not null,
    number      varchar(10)  not null check (number regexp '^[0-9]{10}$'),
    primary key (school_name, number),
    constraint fk_school_phone_number_school_name
        foreign key (school_name) references school (name)
            on delete cascade
            on update cascade
);
```

```sql
-- user related tables

create table app_user
(
    username   varchar(255) not null,
    password   varchar(255) not null,
    first_name varchar(255) not null,
    last_name  varchar(255) not null,
    primary key (username)
);

create table user_administrator
(
    username varchar(255) not null,
    primary key (username),
    constraint fk_administrator_username
        foreign key (username) references app_user (username)
);

create table user_operator
(
    username             varchar(255) not null,
    operating_school_name varchar(255),
    primary key (username),
    constraint fk_operator_username
        foreign key (username) references app_user (username)
            on delete cascade
            on update cascade,
    constraint fk_operator_school
        foreign key (operating_school_name) references school (name)
            on delete set null
            on update cascade,

    index (operating_school_name)
);

create table user_student
(
    username             varchar(255)          not null,
    belonging_school_name varchar(255)          not null,
    date_of_birth        date                  not null,
    deactivated          boolean default false not null,
    primary key (username),
    constraint fk_student_username
        foreign key (username) references app_user (username)
            on delete cascade
            on update cascade,
    constraint fk_student_belonging_school_name
        foreign key (belonging_school_name) references school (name)
            on update cascade,

    index (belonging_school_name)
);

create table user_teacher
(
    username             varchar(255)          not null,
    belonging_school_name varchar(255)          not null,
    date_of_birth        date                  not null,
    deactivated          boolean default false not null,
    primary key (username),
    constraint fk_teacher_username
        foreign key (username) references app_user (username)
            on delete cascade
            on update cascade,
    constraint fk_teacher_belonging_school_name
        foreign key (belonging_school_name) references school (name)
            on update cascade,
```

```
    index (belonging_school_name)
);
```

```sql
create table user_operator_pending_approval
(
    username              varchar(255) not null,
    password              varchar(255) not null,
    first_name            varchar(255) not null,
    last_name             varchar(255) not null,
    operating_school_name varchar(255) not null,
    primary key (username),
    constraint fk_operator_pending_school
        foreign key (operating_school_name) references school (name)
            on delete cascade
            on update cascade,

    index (operating_school_name)
);

create table user_teacher_pending_approval
(
    username              varchar(255) not null,
    password              varchar(255) not null,
    first_name            varchar(255) not null,
    last_name             varchar(255) not null,
    belonging_school_name varchar(255) not null,
    date_of_birth         date         not null,
    primary key (username),
    constraint fk_teacher_pending_school
        foreign key (belonging_school_name) references school (name)
            on delete cascade
            on update cascade,

    index (belonging_school_name)
);

create table user_student_pending_approval
(
    username              varchar(255) not null,
    password              varchar(255) not null,
    first_name            varchar(255) not null,
    last_name             varchar(255) not null,
    belonging_school_name varchar(255) not null,
    date_of_birth         date         not null,
    primary key (username),
    constraint fk_student_pending_school
        foreign key (belonging_school_name) references school (name)
            on delete cascade
            on update cascade,

    index (belonging_school_name)
);
```

```sql
-- books related tables --

create table book_author
(
    id         int unsigned auto_increment,
    first_name varchar(255) not null,
    last_name  varchar(255) not null,
    primary key (id)
);

create table book_category
(
    name varchar(255) not null,
    primary key (name)
);

create table book_keyword
(
    name varchar(255) not null,
    primary key (name)
);

create table book
(
    isbn       char(13)     not null check (isbn regexp '^[0-9]{12}[0-9xX]$'),
    title      varchar(255) not null,
    publisher  varchar(255) not null,
    page_count int unsigned not null,
    summary    text         not null,
    cover      blob,
    language   varchar(255) not null,
    primary key (isbn)
);

create table book_author_map
(
    isbn      char(13)     not null,
    author_id int unsigned not null,
    primary key (isbn, author_id),
    constraint fk_book_author_isbn
        foreign key (isbn) references book (isbn)
            on delete cascade
            on update cascade,
    constraint fk_book_author_id
        foreign key (author_id) references book_author (id)
            on delete cascade
            on update cascade
);

create table book_category_map
(
    isbn          char(13)     not null,
    category_name varchar(255) not null,
    primary key (isbn, category_name),
    constraint fk_book_category_isbn
        foreign key (isbn) references book (isbn)
            on delete cascade
            on update cascade,
    constraint fk_book_category_name
        foreign key (category_name) references book_category (name)
            on delete cascade
            on update cascade
);
```

```sql
create table book_keyword_map
(
    isbn         char(13)     not null,
    keyword_name varchar(255) not null,
    primary key (isbn, keyword_name),
    constraint fk_book_keyword_isbn
        foreign key (isbn) references book (isbn)
            on delete cascade
            on update cascade,
    constraint fk_book_keyword_name
        foreign key (keyword_name) references book_keyword (name)
            on delete cascade
            on update cascade
);

create table book_belonging
(
    isbn               char(13)     not null,
    school_name        varchar(255) not null,
    total_copies_count int unsigned not null,
    primary key (isbn, school_name),
    constraint fk_book_belonging_isbn
        foreign key (isbn) references book (isbn)
            on delete restrict
            on update cascade
);

create table book_reservation
(
    isbn        char(13)     not null,
    username    varchar(255) not null,
    expiry_date date         not null default date_add(current_date(), interval 1 week),
    school_name varchar(255) not null,
    primary key (isbn, username),
    constraint fk_reservation_isbn
        foreign key (isbn) references book (isbn)
            on delete cascade
            on update cascade,
    constraint fk_reservation_username
        foreign key (username) references app_user (username)
            on delete cascade
            on update cascade,
    constraint fk_reservation_school_name
        foreign key (school_name) references school (name)
            on delete cascade
            on update cascade,

    index (school_name)
);
```

```sql
create table book_borrowing_active
(
    isbn              char(13)                                          not null,
    username          varchar(255)                                      not null,
    borrowing_date    date default current_date()                       not null,
    due_date          date default date_add(current_date(), interval 1 week) not null,
    operator_approver varchar(255)                                      not null,
    school_name       varchar(255)                                      not null,
    primary key (isbn, username),
    constraint fk_borrowing_active_isbn
        foreign key (isbn) references book (isbn)
            on update cascade,
    constraint fk_borrowing_active_username
        foreign key (username) references app_user (username)
            on update cascade,
    constraint fk_borrowing_active_operator
        foreign key (operator_approver) references user_operator (username)
            on update cascade,
    constraint fk_borrowing_active_school
        foreign key (school_name) references school (name)
            on update cascade,

    index (school_name)
);

create table book_borrowing_ended
(
    isbn              char(13)     not null,
    username          varchar(255) not null,
    borrowing_date    date         not null,
    operator_approver varchar(255) not null,
    school_name       varchar(255) not null,
    primary key (isbn, username, borrowing_date),
    constraint fk_borrowing_ended_isbn
        foreign key (isbn) references book (isbn)
            on update cascade,
    constraint fk_borrowing_ended_username
        foreign key (username) references app_user (username)
            on delete cascade
            on update cascade,
    constraint fk_borrowing_ended_operator
        foreign key (operator_approver) references user_operator (username)
            on delete cascade
            on update cascade,
    constraint fk_borrowing_ended_school
        foreign key (school_name) references school (name)
            on delete cascade
            on update cascade,

    index (school_name)
);
```

```sql
create table book_review
(
    isbn          char(13)             not null,
    username      varchar(255)         not null,
    text          text,
    likert_rating enum (
        'strongly recommend',
        'recommend',
        'neutral',
        'do not recommend',
        'strongly do not recommend') not null,
    primary key (isbn, username),
    constraint fk_book_review_isbn
        foreign key (isbn) references book (isbn)
            on update cascade,
    constraint fk_book_review_username
        foreign key (username) references app_user (username)
            on delete cascade
            on update cascade
);

create table book_review_pending_approval
(
    isbn          char(13)             not null,
    username      varchar(255)         not null,
    text          text,
    likert_rating enum (
        'strongly recommend',
        'recommend',
        'neutral',
        'do not recommend',
        'strongly do not recommend') not null,
    primary key (isbn, username),
    constraint fk_book_review_pending_isbn
        foreign key (isbn) references book (isbn)
            on delete cascade
            on update cascade,
    constraint fk_book_review_pending_username
        foreign key (username) references app_user (username)
            on delete cascade
            on update cascade
);
```

views.sql

```
use slms;

drop view if exists book_borrowing;
create view book_borrowing as
    select isbn, username, borrowing_date, operator_approver, school_name
    from book_borrowing_active
    union
    select isbn, username, borrowing_date, operator_approver, school_name
    from book_borrowing_ended;
```

events.sql

```
use slms;

-- delete expired reservations every day --
drop event if exists delete_expired_reservations;
create event delete_expired_reservations
    on schedule every 1 day do
    delete
    from book_reservation
    where expiry_date < current_date();
```

triggers.sql

```
use slms;

-- a teacher can't be a student
drop trigger if exists new_teacher;
delimiter $$
create trigger new_teacher
    before insert
    on user_teacher
    for each row
begin
    if exists (select student.username
               from user_student as student
               where student.username = new.username) then
        signal sqlstate '45000'
            set message_text = 'Error: Teacher is already a student';
    end if;
end $$
delimiter ;

-- a student can't be an operator or teacher
drop trigger if exists new_student;
delimiter $$
create trigger new_student
    before insert
    on user_student
    for each row
begin
    if exists (select operator.username
               from user_operator as operator
               where operator.username = new.username) then
        signal sqlstate '45000'
            set message_text = 'Error: Student is already an operator';
    elseif exists (select teacher.username
                   from user_teacher as teacher
                   where teacher.username = new.username) then
        signal sqlstate '45000'
            set message_text = 'Error: Student is already a teacher';
    end if;
end $$
delimiter ;
```

```
-- checks made before a reservation is created
drop trigger if exists new_reservation_pre_register;
delimiter $$
create trigger new_reservation_pre_register
    before insert
    on book_reservation
    for each row
begin
    if exists (select *
               from book_reservation as bb
               where bb.username = new.username
                 and bb.isbn = new.isbn) then
        signal sqlstate '45000'
            set message_text = 'Error: There is a reservation for this book already pending';
    elseif exists (select *
                   from book_borrowing_active as bba
                   where bba.username = new.username
                     and bba.isbn = new.isbn) then
        signal sqlstate '45000'
            set message_text = 'Error: User has already borrowed this book';
    elseif exists (select *
                   from book_borrowing_active as bba
                   where bba.username = new.username
                     and bba.due_date < current_date()) then
        signal sqlstate '45000'
            set message_text = 'Error: User has an overdue borrowing';
    elseif 2 <= (select count(*)
                 from book_borrowing_active as bba
                 where bba.username = new.username
                   and exists (select *
                               from user_student as us
                               where us.username = new.username)
                 group by username) + (select count(*)
                                       from book_reservation as br
                                       where br.username = new.username
                                         and exists (select *
                                                     from user_student as us
                                                     where us.username = new.username)
                                       group by username) then
        signal sqlstate '45000'
            set message_text = 'Error: Student has reached maximum borrowing capacity';
    elseif 1 <= (select count(*)
                 from book_borrowing_active as bba
                 where bba.username = new.username
                   and exists (select *
                               from user_teacher as ut
                               where ut.username = new.username)
                 group by username) + (select count(*)
                                       from book_reservation as br
                                       where br.username = new.username
                                         and exists (select *
                                                     from user_teacher as ut
                                                     where ut.username = new.username)
                                       group by username) then
        signal sqlstate '45000'
            set message_text = 'Error: Teacher has reached maximum borrowing capacity';
    end if;
end $$
delimiter ;
```

```
-- checks made before a borrowing is created
drop trigger if exists book_borrowing_pre_start;
delimiter $$
create trigger book_borrowing_pre_start
    before insert
    on book_borrowing_active
    for each row
begin
    delete
    from book_reservation
    where isbn = new.isbn
      and username = new.username;

    if ((select count(*)
         from book_borrowing_active as bba
         where bba.isbn = new.isbn
           and bba.school_name = new.school_name
         group by isbn) >=
        (select bb.total_copies_count
         from book_belonging as bb
         where bb.isbn = new.isbn
           and bb.school_name = new.school_name)
        ) then
        signal sqlstate '45000'
            set message_text = 'Error: There are no copies of this book available';
    elseif exists (select *
                   from book_borrowing_active as bba
                   where bba.username = new.username
                     and bba.due_date < current_date()) then
        signal sqlstate '45000'
            set message_text = 'Error: User has an overdue borrowing';
    elseif 2 <= (select count(*)
                 from book_borrowing_active as bba
                 where bba.username = new.username
                   and exists (select *
                               from user_student as us
                               where us.username = new.username)
                 group by username) + (select count(*)
                                       from book_reservation as br
                                       where br.username = new.username
                                         and exists (select *
                                                     from user_student as us
                                                     where us.username = new.username)
                                       group by username) then
        signal sqlstate '45000'
            set message_text = 'Error: Student has reached maximum borrowing capacity';
    elseif 1 <= (select count(*)
                 from book_borrowing_active as bba
                 where bba.username = new.username
                   and exists (select *
                               from user_teacher as ut
                               where ut.username = new.username)
                 group by username) + (select count(*)
                                       from book_reservation as br
                                       where br.username = new.username
                                         and exists (select *
                                                     from user_teacher as ut
                                                     where ut.username = new.username)
                                       group by username) then
        signal sqlstate '45000'
            set message_text = 'Error: Teacher has reached maximum borrowing capacity';
    end if;
end $$
delimiter ;
```

```sql
-- when a reservation is created, automatically fill its `school_name` field
drop trigger if exists autofill_reservation_school;
delimiter $$
create trigger autofill_reservation_school
    before insert
    on book_reservation
    for each row
begin
    set @school_name = (select belonging_school_name
                        from user_teacher
                        where username = new.username);
    set @school_name =
            ifnull(@school_name, (select belonging_school_name
                                  from user_student
                                  where username = new.username));
    set new.school_name = @school_name;
end $$
delimiter ;

-- after borrowing a book delete the reservation
drop trigger if exists book_borrowing_after_start;
create trigger book_borrowing_after_start
    after insert
    on book_borrowing_active
    for each row
    delete
    from book_reservation
    where username = new.username
      and school_name = new.school_name
      and isbn = new.isbn;

-- after returning a book delete the book_borrowing_active row
drop trigger if exists book_borrowing_after_end;
create trigger book_borrowing_after_end
    after insert
    on book_borrowing_ended
    for each row
    delete
    from book_borrowing_active
    where username = new.username
      and school_name = new.school_name
      and isbn = new.isbn;

-- after registering a review delete the pending row
drop trigger if exists book_review_after_start;
create trigger book_review_after_start
    after insert
    on book_review
    for each row
    delete
    from book_review_pending_approval
    where isbn = new.isbn
      and username = new.username;
```

```
-- after creating an operator delete the pending row
drop trigger if exists user_operator_after_creation;
create trigger user_operator_after_creation
    after insert
    on user_operator
    for each row
    delete
    from user_operator_pending_approval
    where username = new.username;

-- after creating a teacher delete the pending row
drop trigger if exists user_teacher_after_creation;
create trigger user_teacher_after_creation
    after insert
    on user_teacher
    for each row
    delete
    from user_teacher_pending_approval
    where username = new.username;

-- after creating a student delete the pending row
drop trigger if exists user_student_after_creation;
create trigger user_student_after_creation
    after insert
    on user_student
    for each row
    delete
    from user_student_pending_approval
    where username = new.username;
```

general_procedures.sql

```sql
use slms;

-- browse books that match the given criteria
-- all parameters are optional EXCEPT `given_school_name`
drop procedure if exists browse_school_books;
delimiter $$
create procedure browse_school_books(
    in given_school_name varchar(255),
    in given_title varchar(255),
    in given_category_name varchar(255),
    in given_author_name varchar(255),
    in given_total_copies int unsigned
)
    language sql
    reads sql data
begin
    set @given_title = ifnull(given_title, '%');
    set @given_category_name = ifnull(given_category_name, '%');
    set @given_author_name = ifnull(given_author_name, '%');
    set @given_total_copies = ifnull(given_total_copies, 0);
    -- isbn and copies of books of the given school that satisfy title constraint and copies count
constraint
    with school_book_satisfy_title_and_count
             as (select bb.isbn, total_copies_count
                 from book_belonging bb
                          join book b on bb.isbn = b.isbn
                 where school_name = given_school_name
                   and lower(title) like lower(concat('%', @given_title, '%'))
                   and total_copies_count >= @given_total_copies),
         -- isbn of books that satisfy author constraint
         school_book_satisfy_author
             as (select sbstac.isbn
                 from school_book_satisfy_title_and_count sbstac
                          join book_author_map bam on sbstac.isbn = bam.isbn
                          join book_author ba on bam.author_id = ba.id
                 where lower(concat(first_name, ' ', last_name)) like
                       lower(@given_author_name)
                 group by sbstac.isbn),
         -- isbn of books that satisfy category constraint
         school_book_satisfy_category
             as (select sbstac.isbn
                 from school_book_satisfy_title_and_count sbstac
                          join book_category_map bcm on sbstac.isbn = bcm.isbn
                 where lower(category_name) like lower(@given_category_name)
                 group by sbstac.isbn),
         -- isbn and count of all active borrowings
         school_book_borrowings
             as (select sbstac.isbn, count(*) as active_count
                 from school_book_satisfy_title_and_count sbstac
                          join book_borrowing_active bba on sbstac.isbn = bba.isbn
                 where bba.school_name = @op_school_name
                 group by sbstac.isbn)
    select b.isbn,
           title,
           publisher,
           page_count,
           summary,
           cover,
           language,
           group_concat(distinct concat(a.first_name, ' ', a.last_name) separator ', ') as authors,
           group_concat(distinct bcm.category_name separator ', ')                      as
categories,
           (cast(sbstac.total_copies_count as int) -
            cast(coalesce(sbb.active_count, 0) as int))                                 as
available_copies
    from book b
             -- get only isbns of books that satisfy the constraints
             join school_book_satisfy_title_and_count sbstac on b.isbn = sbstac.isbn
```

```
            join school_book_satisfy_author sbsa on b.isbn = sbsa.isbn
            join school_book_satisfy_category sbsc on b.isbn = sbsc.isbn
            -- get author and category names of authors and categories of books that satisfy the
constraints
            join book_author_map bam on sbsa.isbn = bam.isbn
            join book_author a on bam.author_id = a.id
            join book_category_map bcm on sbsc.isbn = bcm.isbn
            left join school_book_borrowings sbb on b.isbn = sbb.isbn
    group by b.isbn;
end $$
delimiter ;
```

```sql
-- create a book and all other entities related to it
drop procedure if exists create_book;
delimiter $$
create procedure create_book(
    in new_isbn varchar(255),
    in new_title varchar(255),
    in new_publisher varchar(255),
    in new_page_count int unsigned,
    in new_summary text,
    in new_cover blob,
    in new_language varchar(255),
    in new_author_first_name varchar(255),
    in new_author_last_name varchar(255),
    in new_category_name varchar(255),
    in new_keyword_name varchar(255),
    in op_username varchar(255),
    in new_copies varchar(255)
)
begin
    start transaction;

    -- create book
    insert into book (isbn, title, publisher, page_count, summary, cover, language)
    values (new_isbn, new_title, new_publisher, new_page_count, new_summary, new_cover,
new_language);

    -- create author
    set @author_id = ifnull((select id
                             from book_author
                             where lower(first_name) = lower(new_author_first_name)
                               and lower(last_name) = lower(new_author_last_name)), 'none');
    if @author_id = 'none'
    then
        insert into book_author (first_name, last_name)
        values (new_author_first_name, new_author_last_name);
        set @author_id = last_insert_id();
    end if;
    insert into book_author_map (isbn, author_id)
    values (new_isbn, @author_id);

    -- create category
    if not exists (select name
                   from book_category
                   where lower(name) = lower(new_category_name))
    then
        insert into book_category (name)
        values (new_category_name);
    end if;
    insert into book_category_map (isbn, category_name)
    values (new_isbn, new_category_name);

    -- create keyword
    if not exists (select name
                   from book_keyword
                   where lower(name) = lower(new_keyword_name))
    then
        insert into book_keyword (name)
        values (new_keyword_name);
    end if;
    insert into book_keyword_map (isbn, keyword_name)
    values (new_isbn, new_keyword_name);

    -- create belonging IF a username is given
    if op_username is not null
    then
        set @op_school_name = (select operating_school_name from user_operator where username =
op_username);
        insert into book_belonging (isbn, school_name, total_copies_count)
        values (new_isbn, @op_school_name, new_copies);
    end if;
```

```
    commit;
end $$
delimiter ;
-- view student/teacher reservations
drop procedure if exists school_member_reservations_showcase;
create procedure school_member_reservations_showcase(in sm_username varchar(255))
    language sql
    reads sql data
with reservation as (select isbn, expiry_date
                     from book_reservation
                     where username = sm_username)
select book.isbn,
       book.title,
       book.publisher,
       book.page_count,
       book.summary,
       book.cover,
       book.language,
       reservation.expiry_date
from reservation
         join book
              on reservation.isbn = book.isbn;

-- get all students/teachers of an operator's school
drop procedure if exists school_member_list;
delimiter $$
create procedure school_member_list(in op_username varchar(255))
    language sql
    reads sql data
begin
    set @op_school_name = (select operating_school_name
                           from user_operator
                           where username = op_username);
    with school_member as (select username, 'teacher' as occupation
                           from user_teacher
                           where belonging_school_name = @op_school_name
                           union
                           select username, 'student' as occupation
                           from user_student
                           where belonging_school_name = @op_school_name)
    select school_member.username,
           school_member.occupation,
           app_user.first_name,
           app_user.last_name
    from school_member
             join app_user on school_member.username = app_user.username;
end $$
delimiter ;

-- get all pending reviews of an operator's school
drop procedure if exists get_school_pending_reviews;
delimiter $$
create procedure get_school_pending_reviews(in op_username varchar(255))
    language sql
    reads sql data
begin
    set @op_school_name = (select operating_school_name
                           from user_operator
                           where username = op_username);
    with school_member as (select username
                           from user_student
                           where belonging_school_name = @op_school_name
                           union
                           select username
                           from user_teacher
                           where belonging_school_name = @op_school_name)
    select brpa.isbn, brpa.username, brpa.text, brpa.likert_rating
    from book_review_pending_approval brpa
             join school_member on brpa.username = school_member.username;
```

```
end $$
delimiter ;
```

admin_procedures.sql

```
use slms;

-- 3.1.1 --
/* View all borrowings per school in between start_date and end_date */
drop procedure if exists lendings_per_school;
create procedure lendings_per_school(in start_date date, in end_date date)
    language sql
    reads sql data
select school_name, count(*) as book_lendings_count
from book_borrowing
where borrowing_date between start_date and end_date
group by school_name;


-- 3.1.2 --
/* View all authors of a category and all teachers who borrowed
   a book of this category the past year */
drop procedure if exists author_teacher_by_category;
create procedure author_teacher_by_category(in given_category_name varchar(255))
    language sql
    reads sql data
    -- books belonging to the given category
with category_book as (select isbn
                       from book_category_map
                       where category_name = given_category_name),
     -- borrowings made in the past year
     past_year_borrowing as (select username, isbn
                             from book_borrowing
                             where borrowing_date >= date_format(now(), '%Y-01-01')),
     -- users who borrowed books of the given category in the past year
     category_book_borrower as (select username
                                from past_year_borrowing pyb
                                         join category_book cb on pyb.isbn = cb.isbn),
     -- teacher usernames who borrowed books of the given category in the past year
     category_teacher_borrower as (select ut.username
                                   from user_teacher ut
                                            join category_book_borrower cbb on ut.username =
cbb.username),
     -- author ids of the given category
     category_author as (select author_id as id
                         from book_author_map bam
                                  join category_book cb on bam.isbn = cb.isbn)
-- author names of the given category
select ba.id as identification, first_name, last_name, 'author' as occupation
from book_author ba
         join category_author ca on ba.id = ca.id
union
-- teacher names who borrowed books of that category in the past year
select au.username as identification, first_name, last_name, 'teacher' as occupation
from app_user au
         join category_teacher_borrower ctb on au.username = ctb.username;
```

```sql
-- 3.1.3 --
/* View all teachers less than max_age years old who have borrowed the most books,
   and the number of books they have borrowed */
drop procedure if exists find_young_teachers;
create procedure find_young_teachers(in max_age int)
    language sql
    reads sql data
-- teachers less than `max_age` years old
with young_teacher as (select username
                       from user_teacher
                       where date_of_birth > date_sub(current_date, interval max_age year)),
     -- borrowings of young teachers
     young_teacher_borrowing as (select yt.username
                                 from book_borrowing bb
                                          join young_teacher yt on bb.username = yt.username),
     -- borrowings count of young teachers
     young_teacher_borrowing_count as (select ytb.username, count(*) as borrowing_count
                                       from young_teacher_borrowing ytb
                                       group by ytb.username),
     -- borrowings of the teachers with the maximum borrowings
     young_teacher_max_borrower as (select username, max(borrowing_count) as max_borrowing_count
                                    from young_teacher_borrowing_count
                                    group by username)
select au.username,
       au.first_name,
       au.last_name,
       ytmb.max_borrowing_count
from young_teacher_max_borrower ytmb
         join app_user au on au.username = ytmb.username;

-- 3.1.4 --
/* View all authors whose books have never been borrowed */
drop procedure if exists find_unpopular_authors;
create procedure find_unpopular_authors()
    language sql
    reads sql data
-- author ids of authors without borrowings
with unpopular_author as (select distinct author_id as id
                          from book_author_map bam
                                   left join book_borrowing bb on bam.isbn = bb.isbn
                          group by id
                          having count(bb.isbn) = 0)
select ba.id, ba.first_name, ba.last_name
from unpopular_author ua
         join book_author ba on ba.id = ua.id;

-- 3.1.5 --
/* View all operators with more than 20 approvals that have
   approved the same number of borrowings in the year year_no */
drop procedure if exists find_high_value_operators;
create procedure find_high_value_operators(in year_no int)
    language sql
    reads sql data
select operator_approver as username, count(*) as approvals_count
from book_borrowing bb
where year(bb.borrowing_date) = year_no
group by operator_approver
having approvals_count > 20
order by approvals_count desc;
```

```sql
-- 3.1.6 --
/* View the top pair_cnt most common genre pairs in borrowings */
drop procedure if exists find_most_popular_category_pairs;
create procedure find_most_popular_category_pairs(in pair_cnt int)
    language sql
    reads sql data
-- all category pairs (no permutations)
with book_category_pair as (select bcm1.isbn,
                                   bcm1.category_name as category1,
                                   bcm2.category_name as category2
                            from book_category_map as bcm1
                                     join book_category_map as bcm2
                                          on bcm1.isbn = bcm2.isbn
                            where bcm1.category_name < bcm2.category_name),
     book_borrowing_count as (select isbn, count(*) as borrowings_count
                              from book_borrowing
                              group by isbn)
select bcp.category1, bcp.category2, borrowings_count
from book_borrowing_count bbc
         join book_category_pair bcp on bbc.isbn = bcp.isbn
order by bbc.borrowings_count desc
limit pair_cnt;

-- 3.1.7 --
/* View all authors who have written at least `book_dif` books
   less than the author with the most books written */
drop procedure if exists authors_with_some_books_less_than_top;
create procedure authors_with_some_books_less_than_top(in book_dif int)
    language sql
    reads sql data
-- number of books written by each author
with author_book_count as (select author_id, count(*) as book_count
                           from book_author_map
                           group by author_id),
     -- most books written by an author
     most_written_book as (select max(book_count) as count
                           from author_book_count
                           limit 1),
     -- id of the requested authors
     requested_author as (select author_id as id, book_count
                          from author_book_count
                          where book_count <= ((select count from most_written_book) - book_dif))
select ba.id, first_name, last_name, book_count
from requested_author ra
         join book_author ba on ra.id = ba.id
order by book_count desc;
```

operator_procedures.sql

```sql
use slms;

-- 3.2.1 --
/* Browse all books */
drop procedure if exists operator_present_books;
delimiter $$
create procedure operator_present_books(
    in op_username varchar(255),
    in given_title varchar(255),
    in given_category_name varchar(255),
    in given_author_name varchar(255),
    in given_total_copies int unsigned
)
    language sql
    reads sql data
begin
    set @op_school_name = (select operating_school_name from user_operator where username =
op_username);
    call browse_school_books(
            @op_school_name,
            given_title,
            given_category_name,
            given_author_name,
            given_total_copies
        );
end $$
delimiter ;

-- 3.2.2 --
/* View all overdue borrowers of the operator's school */
drop procedure if exists find_overdue_borrowers;
delimiter $$
create procedure find_overdue_borrowers(
    in op_username varchar(255),
    in given_first_name varchar(255),
    in given_last_name varchar(255),
    in given_days_overdue int unsigned
)
    language sql
    reads sql data
begin
    set @given_first_name = ifnull(given_first_name, '%');
    set @given_last_name = ifnull(given_last_name, '%');
    set @given_days_overdue = ifnull(given_days_overdue, 0);
    set @op_school_name = (select operating_school_name from user_operator where username =
op_username);
    select isbn, au.username, first_name, last_name, timestampdiff(day, due_date, current_date()) as
days_overdue
    from app_user au
             join book_borrowing_active as bba on au.username = bba.username
    where school_name = @op_school_name
      and current_date() > due_date
      and au.first_name like @given_first_name
      and au.last_name like @given_last_name
      and timestampdiff(day, due_date, current_date()) >= @given_days_overdue
    order by days_overdue desc;
end $$
delimiter ;
```

```sql
-- 3.2.3 --
/* View the average review score per user and per category */
drop procedure if exists find_avg_review_score;
delimiter $$
create procedure find_avg_review_score(
    in given_username varchar(255),
    in given_category_name varchar(255)
)
    language sql
    reads sql data
begin
    set @given_username = ifnull(given_username, '%');
    set @given_category_name = ifnull(given_category_name, '%');
    select au.username, first_name, last_name, category_name, avg(likert_rating) as avg_rating
    from book_review br
             join app_user au on br.username = au.username
             join book_category_map bcm on br.isbn = bcm.isbn
             join book_category bc on bcm.category_name = bc.name
    where category_name like @given_category_name
      and au.username like @given_username
    group by username, category_name;
end $$
delimiter ;
```

user_procedures.sql

```
use slms;

-- 3.3.1 --
/* Browse all books of the user's school */
drop procedure if exists school_member_present_books;
delimiter $$
create procedure school_member_present_books(
    in sm_username varchar(255),
    in given_title varchar(255),
    in given_category varchar(255),
    in given_author_name varchar(255)
)
    language sql
    reads sql data
begin
    set @school_name = (select belonging_school_name
                        from (select username, belonging_school_name
                              from user_teacher
                              union
                              select username, belonging_school_name
                              from user_student) as school_member
                        where username = sm_username);
    call browse_school_books(
            @school_name,
            given_title,
            given_category,
            given_author_name,
            null
        );
end $$
delimiter ;

-- 3.3.2 --
/* View user's borrowing history */
drop procedure if exists school_member_borrowings_showcase;
create procedure school_member_borrowings_showcase(in sm_username varchar(255))
    language sql
    reads sql data
with borrowed_book as (select isbn, borrowing_date
                       from book_borrowing bb
                       where username = sm_username),
     borrowed_book_author as (select bb.isbn,
                                     group_concat(distinct concat(first_name, ' ', last_name)
separator
                                                  ', ') as names
                              from borrowed_book bb
                                       join book_author_map bam on bb.isbn = bam.isbn
                                       join book_author ba on bam.author_id = ba.id
                              group by bb.isbn),
     borrowed_book_category as (select bb.isbn, group_concat(distinct bcm.category_name separator ',
') as names
                                from borrowed_book bb
                                         join book_category_map bcm on bb.isbn = bcm.isbn
                                         join book_category bc on bcm.category_name = bc.name
                                group by bb.isbn)
select b.isbn,
       title,
       publisher,
       page_count,
       summary,
       cover,
       language,
       bba.names as authors,
       bbc.names as categories,
       bb.borrowing_date
from book b
         join borrowed_book bb on b.isbn = bb.isbn
         join borrowed_book_author bba on b.isbn = bba.isbn
```

```
        join borrowed_book_category bbc on b.isbn = bbc.isbn
order by bb.borrowing_date;
```

Για την εισαγωγή δεδομένων στη βάση δημιουργήσαμε ένα script χρησιμοποιώντας την JavaScript βιβλιοθήκη ‘faker’. Στις οδηγίες εγκατάστασης της εφαρμογής υπάρχουν και οδηγίες για τη δημιουργία και εισαγωγή δεδομένων στη βάση χρησιμοποιώντας αυτό.

1.3. User Manual

Πρώτη επαφή με την εφαρμογή:
Στο πάνω μέρος της αρχικής σελίδας (‘/’) υπάρχει μία μπάρα πλοήγησης με 4 πλήκτρα ‘Home’, ‘About’, ‘Log In’ και ‘Register’. Ο χρήστης επισκέπτεται την ιστοσελίδα της εφαρμογής και αντικρίζει την κεντρική σελίδα, το ‘Home’ page, όπου εμφανίζεται ένα μήνυμα καλωσορίσματος.

- ‘About’: γενικές πληροφορίες για την ομάδα μας και την εφαρμογή
- ‘Log In’: μία κλασσική φόρμα εισόδου σε εφαρμογή. Ο χρήστης υποβάλει τα διαπιστευτήρια ταυτότητάς του και εισέρχεται στην εφαρμογή ή λαμβάνει μήνυμα λάθους αν κάτι πήγε στραβά ή διαλέγει να δημιουργήσει λογαριασμό μέσω της επιλογής ‘I don’t have an account’ οπότε και κατευθύνεται στη ‘Register’ σελίδα
- ‘Register’: ο χρήστης συμπληρώνει τα πεδία με τις προσωπικές του πληροφορίες και υποβάλει αίτηση δημιουργίας λογαριασμού, την οποία πρέπει να εξετάσει ο χειριστής της αντίστοιχης σχολικής μονάδας. Ο χρήστης ενημερώνεται για την επιτυχία/αποτυχία υποβολής του αιτήματός του. Μέσω την επιλογής ‘I already have an account’ ο χρήστης μεταφέρεται στην ‘Log In’ σελίδα.

Μετά την είσοδο στην εφαρμογή, η μπάρα πλοήγησης έχει το πλήκτρο ‘Log Out’ στη θέση του ‘Log In’ και το πλήκτρο ‘Account’ στη θέση του ‘Register’ και ελέγχεται ο ρόλος του χρήστη και ο χρήστης καθοδηγείται στο αντίστοιχο περιβάλλον.
Οι δυνατοί ρόλοι είναι:

- admin
- operator
- teacher
- student

Στην ‘Account’ σελίδα όλοι οι χρήστες μπορούν να τροποποιήσουν τον κωδικό τους, ενώ οι καθηγητές μπορούν επιπλέον να τροποποιήσουν τη Σχολική Μονάδα στην οποία ανήκουν και την ημερομηνία γέννησής τους.

Είσοδος ως admin:
Παρέχονται οι εξής λειτουργίες μέσω κουμπιών στο ‘Home’ page:

- Create new school
- Create database backup
- Restore database backup
- View and approve pending operator accounts
- View total borrowings per school in a date range (3.1.1)
- View authors with books of a category and teachers with borrowings of that category in the past year (3.1.2)
- View teachers younger than 40 with the most borrowings (3.1.3)
- View authors whose books have never been borrowed (3.1.4)
- View operators with same number of borrowing approvals that have more than 20 approvals (3.1.5)
- View the top 3 category pairs in borrowings (3.1.6)
- View authors who have written at least 5 books less than the author with the most books written (3.1.7)

Είσοδος ως operator:
Παρέχονται οι εξής λειτουργίες μέσω κουμπιών στο ‘Home’ page:

- Create new book and optionally add it to your school
- View and approve pending reservations made by members of your school
- View and mark as returned due borrowings of members of your school
- View and approve pending teacher accounts of members of your school
- View and approve pending student accounts of members of your school
- View and deactivate or delete school member accounts of members of your school
- View and approve pending reviews of members of your school
- View books available at your school (3.2.1)
- View and mark as returned overdue borrowings of members of your school (3.2.2)
- View average score of reviews of members of all schools (3.2.3)

Είσοδος ως teacher/student στο ‘Home’ page:

- Create a review
- View and delete your active reservations
- View and reserve books available at your school (3.3.1)
- View your borrowing history (3.3.2)

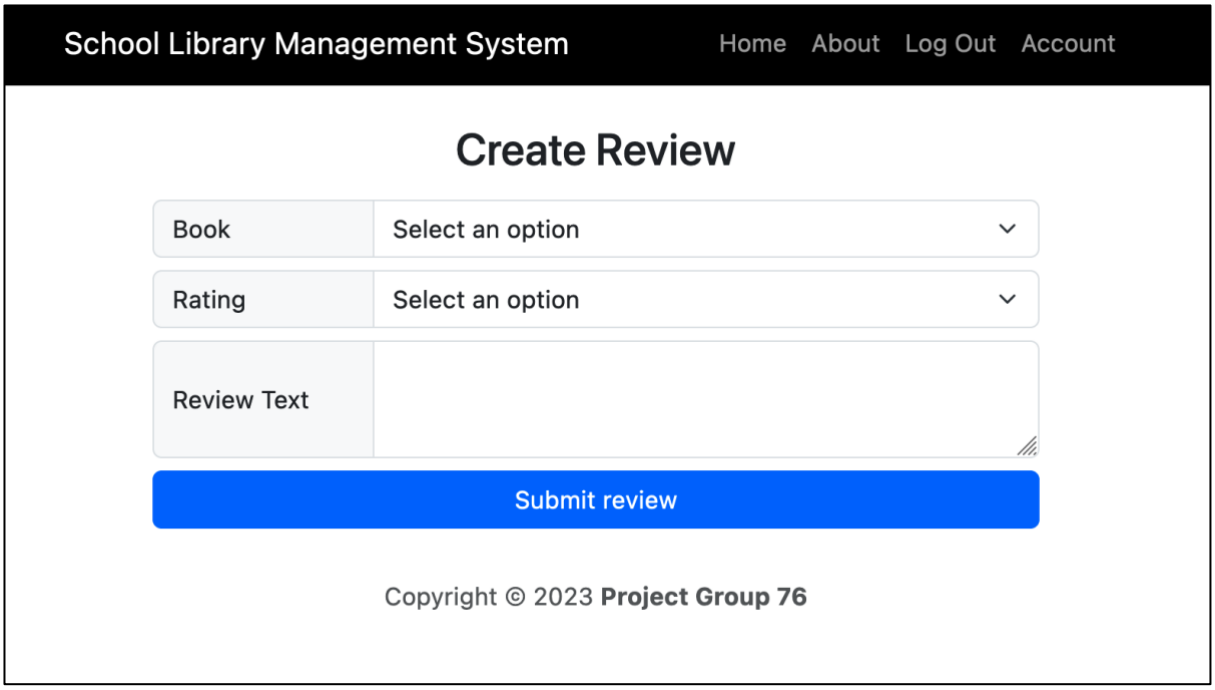
Οι λειτουργίες των παραπάνω κουμπιών γίνονται εμφανείς από το κείμενο τους.
Τα παραπάνω κουμπιά όταν πατηθούν οδηγούν τον χρήστη σε κάποια φόρμα, όπου του ζητείται η συμπλήρωση των αντίστοιχων πεδίων, αν πρόκειται για ενέργειες δημιουργίας rows/’οντοτήτων’/’αντικειμένων’ στη βάση.

Οι υπόλοιπες ενέργειες σχετίζονται με την προβολή στον χρήστη λίστας δεδομένων. Όταν πατιέται ένα κουμπί, αν υπάρχει η δυνατότητα εφαρμογής επιπρόσθετων φίλτρων, εμφανίζεται στον χρήστη σχετικό pop-up παράθυρο και έπειτα οδηγείται στην αντίστοιχη λίστα.

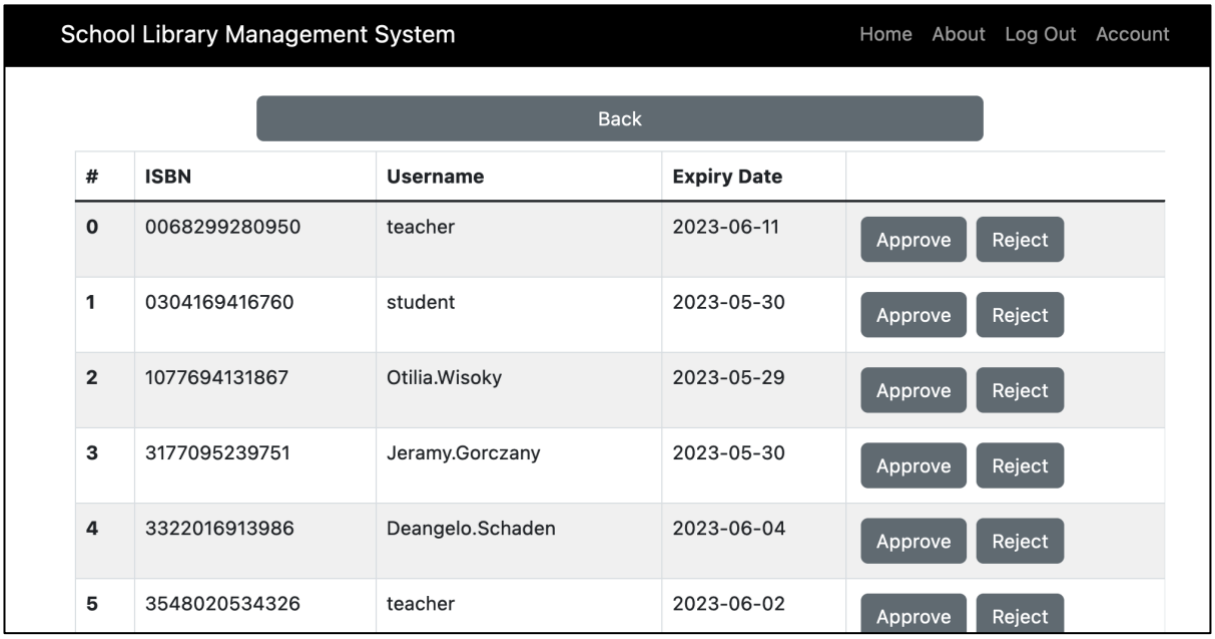
Υπάρχει πλήκτρο επιστροφής ‘Back’ ενώ σε αρκετές λίστες, όπου είναι δυνατή η επεξεργασία των δεδομένων, υπάρχουν αντίστοιχα κουμπιά στην τελευταία στήλη της λίστας. Τα κουμπιά αυτά όταν πατηθούν εκτελούν μία αντίστοιχη ενέργεια στο row της βάσης που αντιστοιχίζεται με τα συνοδεύονται δεδομένα τους και οδηγούν τον χρήστη σε μία σελίδα επιτυχίας/αποτυχίας αναλόγως την έκβαση του αντίστοιχο αιτήματος.

Για καλύτερη εξήγηση του τελευταίου δίνουμε ένα παράδειγμα:
Ένας χειριστής (operator) πατάει το κουμπί ‘Approve teacher accounts’ και οδηγείται σε μία λίστα καθηγητών με λογαριασμούς σε αναμονή. Στη λίστα αυτή δίπλα από κάθε εγγραφή υπάρχει ένα κουμπί ‘Approve’ με το οποίο ο χειριστής διαλέγει να εγκρίνει το αντίστοιχο αίτημα.

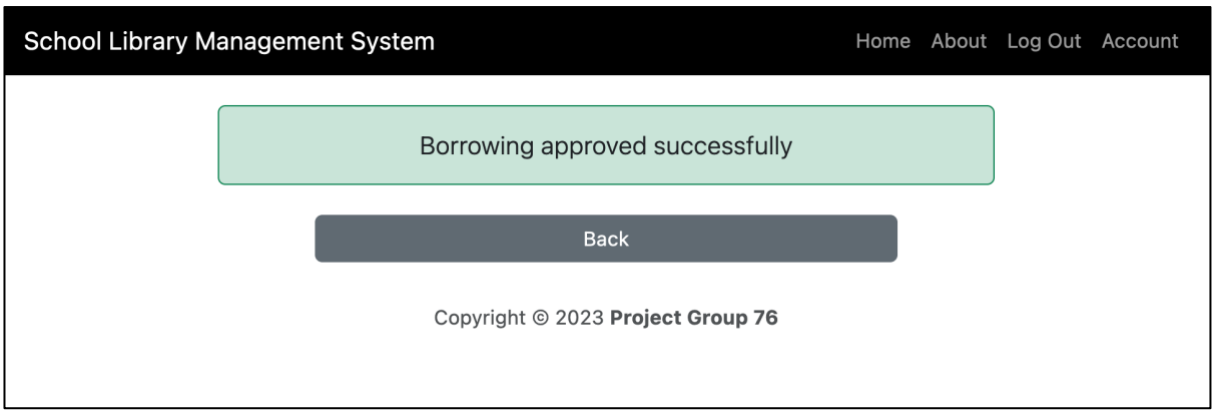
Παράδειγμα σελίδας με φόρμα εισαγωγής δεδομένων:



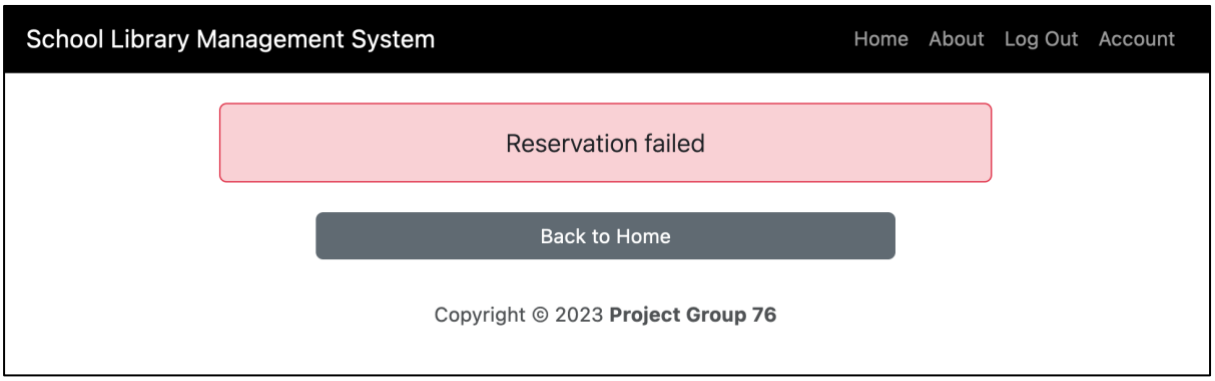
Παράδειγμα σελίδας με λίστα δεδομένων:



Παράδειγμα σελίδας επιτυχίας:



Παράδειγμα σελίδας αποτυχίας:



1.4. Αναλυτικά βήματα εγκατάστασης της εφαρμογής μας

Τα αρχεία που απαιτούνται για την εγκατάσταση της εφαρμογής είναι πολλά και αποτελούνται από χιλιάδες γραμμές κώδικα, οπότε δεν θα τα συμπεριλάβουμε εδώ. Υπάρχουν όμως στο GitHub του project.
Αναλυτικές οδηγίες εγκατάστασης της εφαρμογής υπάρχουν στο README.md στο GitHub. Αυτό παρατίθεται και εδώ πέρα:

Setup

Requirements

- node.js (v20.2.0)
- MariaDB (v10.11.3)
- Clone the repository. For the rest of the steps it is assumed that it was cloned in folder `/`.

Instructions for the backend

1. `cd` into `/backend` and run `npm install`.
2. Create a file called `.env` and fill it according to `.env.sample`.
   Be sure to set `DB_NAME=slms`, because the database name is hardcoded in the SQL files.
3. Start the app in development mode using `npm run dev` or in production mode using `node app.js`.
4. The app should now be accessible on `http://APP_HOST:APP_PORT/`. Of course `APP_HOST` should be `localhost` when running it locally.

Instructions for the database

1. `cd` into `/populate_db` and run `npm install`.
2. Run `node fill_db.js <output_name>` where `<output_name>` is the name that will be given to the output file of the process.
   The output will be an SQL file containing dummy data to be inserted into the database. We recommend selecting a filename similar to `../database/dummy_data.sql`, meaning inside the `/database` folder as it will make inserting the data easier.
3. `cd` into `/database` and give execution permission for `init_db.sh`.
4. Run `node init_db.js <db_username> <db_password> [<data_file_1> <data_file_2> ...]` where the first two arguments are the username and the password you use to access MariaDB, while the rest are the names of the data files to be inserted in the database. Give the name of the file created on step 2.
5. (Alternative) If you don't want to run the script or if it doesn't work you can parse the SQL files one by one using the command `mariadb --user=<db_username> --password=<db_password> < sql_file.sql`. Start with `schema.sql` and end with the dummy data file. The intermediate order does not matter.

Notes

- To change the amount of database rows generated for any table, change the values inside the `generate()` function in `/populate_db/generate.js`.
- The dummy data generation script always creates the following users, which are useful for testing the application:
  - Username `admin`, password `admin`, is an admin
  - Username `operator`, password `operator`, is an operator
  - Username `teacher`, password `teacher`, is a teacher
  - Username `student`, password `student`, is a student

1.5. Σύνδεσμος για το GitHub repo της εφαρμογής μας
https://github.com/ntua-el19062/ece-ntua-databases-group-76