Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
Ροή Λ, 6ο εξάμηνο

Βάσεις Δεδομένων

Εξαμηνιαία εργασία

Σπουδαστής
Παπασκαρλάτος Αλέξανδρος(Α.Μ.: 03111097)

Ημερομηνία Υποβολής: 4 Ιουνίου 2023

Git repo: https://github.com/papaskal/Database-systems

Περιεχόμενα

1. Εισαγωγή

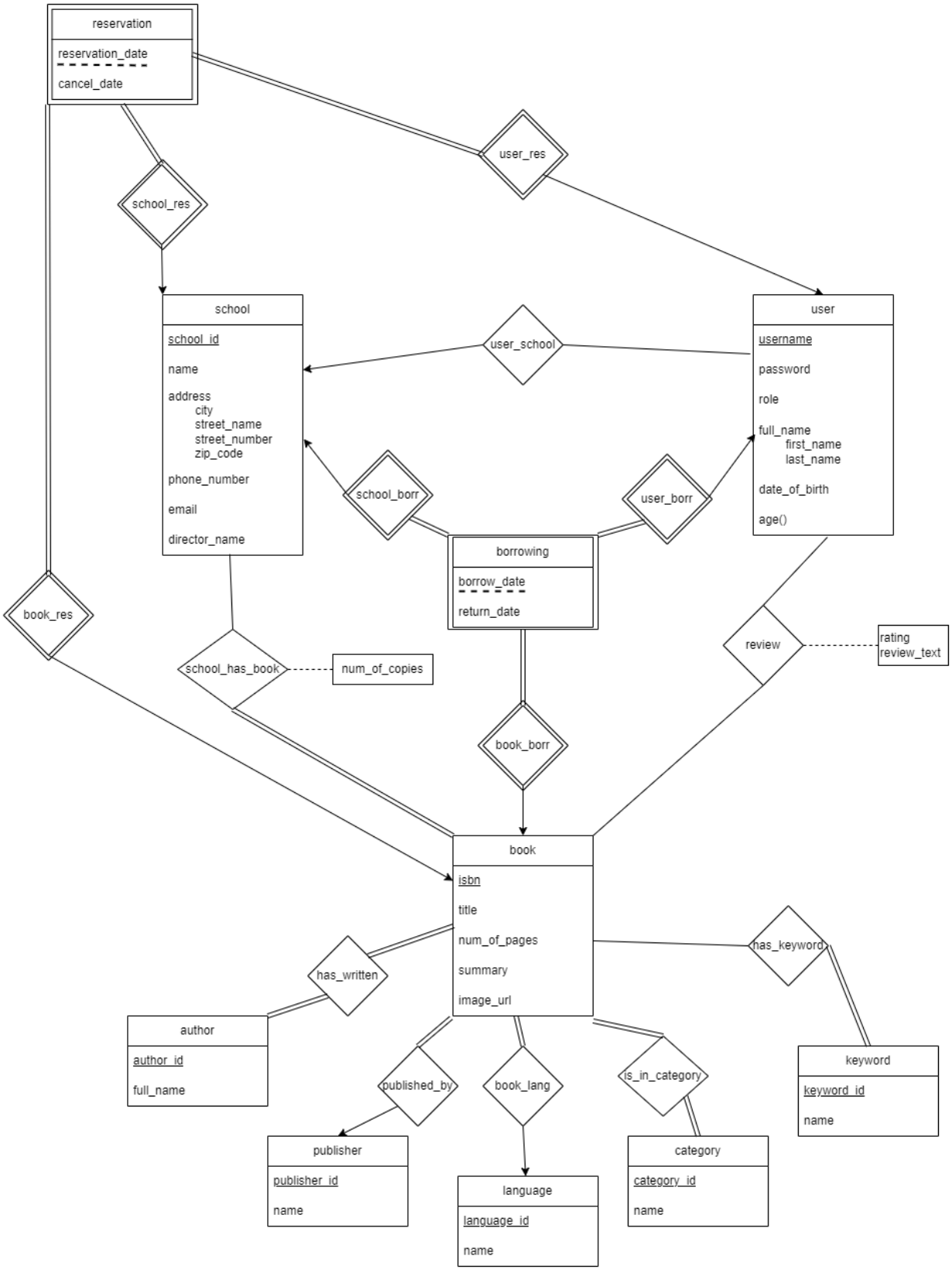
Το παρόν pdf αποτελεί αναφορά της εξαμηνιαίας εργασίας για το μάθημα Βάσεων Δεδομένων, έτος 2022 - 2023.

Η εργασία έχει ως αντικείμενο το σχεδιασμό και υλοποίηση ενός συστήματος αποθήκευσης και διαχείρισης πληροφοριών που απαιτούνται για τη λειτουργία Σχολικής Βιβλιοθήκης στα Δημόσια Σχολεία.
Ιδιαίτερη βάση δίνεται στα κομμάτια της SQL, και στα ερωτήματα που υλοποιούνται σύμφωνα με την εκφώνηση.

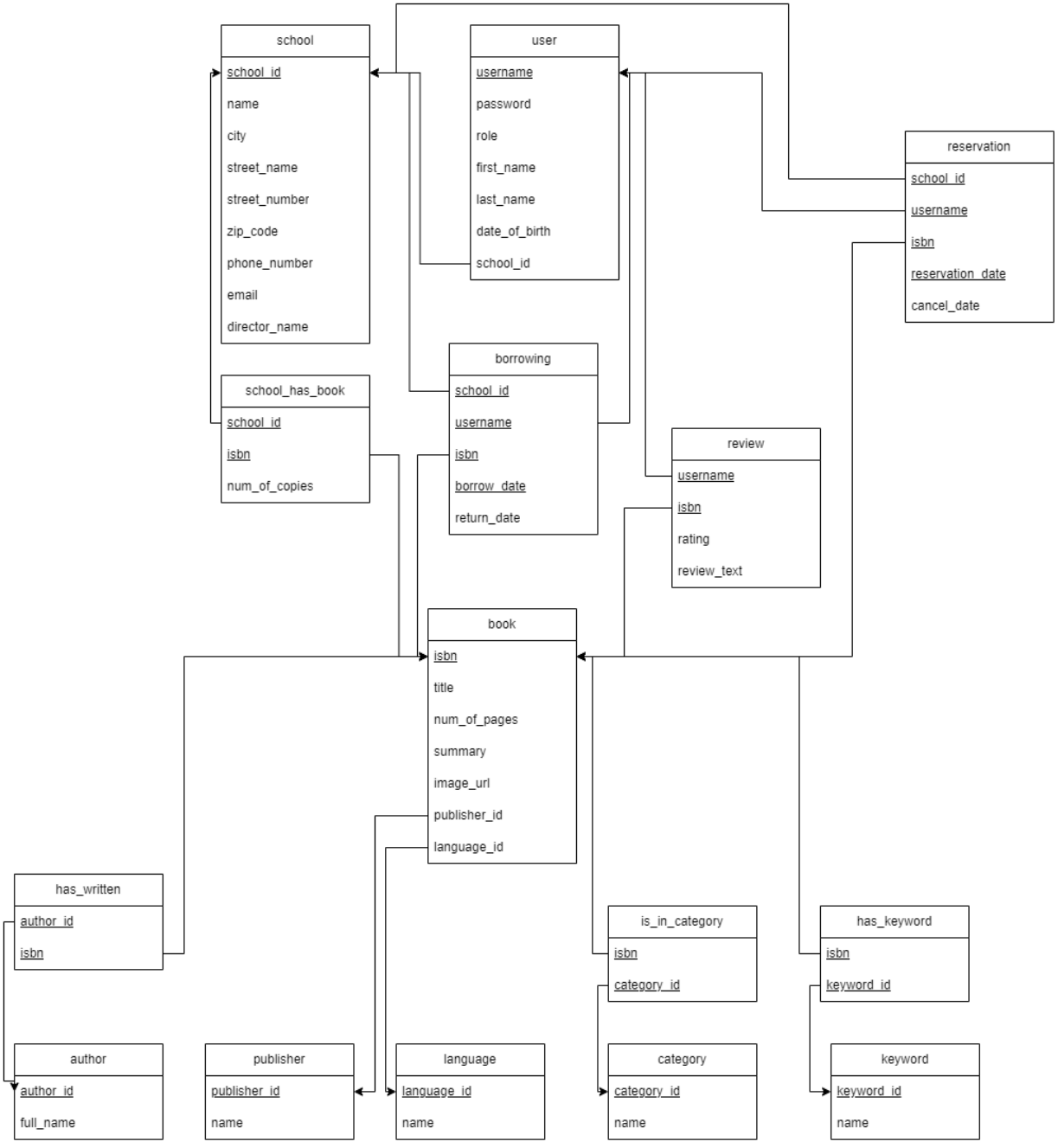
Όλα τα κομμάτια της εργασίας βρίσκονται στο git repo https://github.com/papaskal/Database-systems που φαίνεται και στην πρώτη σελίδα.

Το προγραμματιστικό μέρος της εργασίας υλοποιήθηκε σε mysql (MariaDB) και nodeJS. Περισσότερες πληροφορίες για την εγκατάσταση, την εκτέλεση και τη λειτουργία της εφαρμογής βρίσκονται στο user manual στο τέλος της αναφοράς.

2. Entity - Relationship diagram



3. Relational diagram



4. DDL

4.1. Tables

Οι πίνακες της SQL που δημιουργήθηκαν για τη βάση βρίσκονται στο αρχείο:
https://github.com/papaskal/Database-systems/blob/main/SQL/data.sql
Πρόκειται για τον κάτωθι κώδικα:

```sql
DROP DATABASE IF EXISTS db_2023;


CREATE DATABASE db_2023;


USE db_2023;


CREATE TABLE school (
    school_id INT UNSIGNED AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    city VARCHAR(255) NOT NULL,
    street_name VARCHAR(255) NOT NULL,
    street_number INT NOT NULL,
    zip_code VARCHAR(10) NOT NULL,
    phone_number VARCHAR(15) NOT NULL,
    email VARCHAR(255) NOT NULL,
    director_name VARCHAR(255) NOT NULL,
    PRIMARY KEY (school_id)
);


CREATE TABLE language (
    language_id INT UNSIGNED AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    PRIMARY KEY (language_id)
);


CREATE TABLE publisher(
    publisher_id INT UNSIGNED AUTO_INCREMENT,
    name VARCHAR(255),
    PRIMARY KEY(publisher_id)
);
```

```sql
CREATE TABLE book (
    isbn VARCHAR(20),
    title VARCHAR(255) NOT NULL,
    num_of_pages INT NOT NULL,
    summary TEXT NOT NULL,
    image_url VARCHAR(255) NOT NULL DEFAULT 'placeholder',
    publisher_id INT UNSIGNED,
    language_id INT UNSIGNED,
    PRIMARY KEY (isbn),
    FOREIGN KEY (language_id) REFERENCES language(language_id) ON DELETE
CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (publisher_id) REFERENCES publisher(publisher_id) ON
DELETE CASCADE ON UPDATE CASCADE
);


CREATE TABLE category(
    category_id INT UNSIGNED AUTO_INCREMENT,
    name VARCHAR(255),
    PRIMARY KEY(category_id)
);


CREATE TABLE keyword(
    keyword_id INT UNSIGNED AUTO_INCREMENT,
    name VARCHAR(255),
    PRIMARY KEY(keyword_id)
);


CREATE TABLE is_in_category(
    isbn VARCHAR(20),
    category_id INT UNSIGNED,
    PRIMARY KEY(isbn, category_id),
    FOREIGN KEY(isbn) REFERENCES book(isbn) ON UPDATE CASCADE ON DELETE
CASCADE,
    FOREIGN KEY(category_id) REFERENCES category(category_id) ON UPDATE
CASCADE ON DELETE CASCADE
```

```
);


CREATE TABLE has_keyword(
    isbn VARCHAR(20),
    keyword_id INT UNSIGNED,
    PRIMARY KEY(isbn, keyword_id),
    FOREIGN KEY(isbn) REFERENCES book(isbn) ON UPDATE CASCADE ON DELETE
CASCADE,
    FOREIGN KEY(keyword_id) REFERENCES keyword(keyword_id) ON UPDATE
CASCADE ON DELETE CASCADE
);


CREATE TABLE author(
    author_id INT UNSIGNED AUTO_INCREMENT,
    full_name VARCHAR(255),
    PRIMARY KEY(author_id)
);


CREATE TABLE has_written(
    author_id INT UNSIGNED,
    isbn VARCHAR(20),
    PRIMARY KEY(author_id, isbn),
    FOREIGN KEY(author_id) REFERENCES author(author_id) ON UPDATE CASCADE
ON DELETE CASCADE,
    FOREIGN KEY(isbn) REFERENCES book(isbn) ON UPDATE CASCADE ON DELETE
CASCADE
);


CREATE TABLE user (
    username VARCHAR(256),
    password VARCHAR(255) NOT NULL,
    role ENUM(
        'administrator',
        'operator',
        'student',
        'professor'
```

```
    ) NOT NULL,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    date_of_birth DATE NOT NULL,
    school_id INT UNSIGNED,
    PRIMARY KEY (username),
    FOREIGN KEY (school_id) REFERENCES school(school_id) ON DELETE CASCADE
ON UPDATE CASCADE
);


CREATE TABLE school_has_book (
    school_id INT UNSIGNED NOT NULL,
    isbn VARCHAR(20) NOT NULL,
    num_of_copies INT UNSIGNED NOT NULL,
    PRIMARY KEY (school_id, isbn),
    FOREIGN KEY (school_id) REFERENCES school(school_id) ON DELETE CASCADE
ON UPDATE CASCADE,
    FOREIGN KEY (isbn) REFERENCES book(isbn) ON DELETE CASCADE ON UPDATE
CASCADE
);


CREATE TABLE borrowing (
    school_id INT UNSIGNED NOT NULL,
    username VARCHAR(256) NOT NULL,
    isbn VARCHAR(20) NOT NULL,
    borrow_date DATE NOT NULL DEFAULT CURDATE(),
    return_date DATE,
    PRIMARY KEY (school_id, username, isbn, borrow_date),
    FOREIGN KEY (school_id) REFERENCES school(school_id) ON DELETE CASCADE
ON UPDATE CASCADE,
    FOREIGN KEY (username) REFERENCES user(username) ON DELETE CASCADE ON
UPDATE CASCADE,
    FOREIGN KEY (isbn) REFERENCES book(isbn) ON DELETE CASCADE ON UPDATE
CASCADE,
    CONSTRAINT valid_return_date CHECK (return_date IS NULL OR return_date
>= borrow_date)
);
```

```sql
CREATE TABLE reservation (
    school_id INT UNSIGNED NOT NULL,
    username VARCHAR(256) NOT NULL,
    isbn VARCHAR(20) NOT NULL,
    reservation_date DATE NOT NULL DEFAULT CURDATE(),
    cancel_date DATE DEFAULT DATE_ADD(reservation_date, INTERVAL 1 WEEK),
    PRIMARY KEY (school_id, username, isbn, reservation_date),
    FOREIGN KEY (school_id) REFERENCES school(school_id) ON DELETE CASCADE
ON UPDATE CASCADE,
    FOREIGN KEY (username) REFERENCES user(username) ON DELETE CASCADE ON
UPDATE CASCADE,
    FOREIGN KEY (isbn) REFERENCES book(isbn) ON DELETE CASCADE ON UPDATE
CASCADE,
    CONSTRAINT valid_cancel_date CHECK (cancel_date >= reservation_date)
);


CREATE TABLE review (
    username VARCHAR(256) NOT NULL,
    isbn VARCHAR(20) NOT NULL,
    rating INT UNSIGNED NOT NULL,
    review_text TEXT,
    PRIMARY KEY (username, isbn),
    FOREIGN KEY (username) REFERENCES user(username) ON DELETE CASCADE ON
UPDATE CASCADE,
    FOREIGN KEY (isbn) REFERENCES book(isbn) ON DELETE CASCADE ON UPDATE
CASCADE,
    CONSTRAINT valid_rating CHECK (rating BETWEEN 1 AND 5)
);
```

4.2. Triggers

Οι σκανδαλιστές που υλοποιήθηκαν για τη βάση βρίσκονται στο αρχείο:
https://github.com/papaskal/Database-systems/blob/main/SQL/triggers.sql
Πρόκειται για τον κάτωθι κώδικα:

```
USE db_2023;

DELIMITER ;;

CREATE TRIGGER borrowing_overdue_check
BEFORE INSERT ON borrowing
FOR EACH ROW
BEGIN
    DECLARE overdueCount INT;

    SELECT COUNT(*) INTO overdueCount
    FROM borrowing
    WHERE username = NEW.username AND return_date IS NULL AND
DATEDIFF(CURDATE(), borrow_date) > 7;

    IF (overdueCount > 0) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A user cannot borrow a book, if they currently
have an overdue book';
    END IF;
END ;;


CREATE TRIGGER reservation_overdue_check
BEFORE INSERT ON reservation
FOR EACH ROW
BEGIN
    DECLARE overdueCount INT;

    SELECT COUNT(*) INTO overdueCount
    FROM borrowing
    WHERE username = NEW.username AND return_date IS NULL AND
DATEDIFF(CURDATE(), borrow_date) > 7;

    IF (overdueCount > 0) THEN
        SIGNAL SQLSTATE '45000'
```

```
        SET MESSAGE_TEXT = 'A user cannot reserve a book, if they
currently have an overdue book';
    END IF;
END ;;


CREATE TRIGGER borrowing_not_already_borrowed_or_reserved
BEFORE INSERT ON borrowing
FOR EACH ROW
BEGIN
    DECLARE countBorrowed INT;
    DECLARE countReserved INT;

    SELECT COUNT(*) INTO countBorrowed
    FROM borrowing
    WHERE username = NEW.username AND isbn = NEW.isbn AND return_date IS
NULL;

    SELECT COUNT(*) INTO countReserved
    FROM reservation
    WHERE username = NEW.username AND isbn = NEW.isbn AND cancel_date >
CURDATE();

    IF (countBorrowed > 0 OR countReserved > 0) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'A user cannot borrow a
book if they already have it reserved or borrowed';
    END IF;
END ;;


CREATE TRIGGER reservation_not_already_borrowed_or_reserved
BEFORE INSERT ON reservation
FOR EACH ROW
BEGIN
    DECLARE countBorrowed INT;
    DECLARE countReserved INT;

    SELECT COUNT(*) INTO countBorrowed
    FROM borrowing
    WHERE username = NEW.username AND isbn = NEW.isbn AND return_date IS
```

```
NULL;

    SELECT COUNT(*) INTO countReserved
    FROM reservation
    WHERE username = NEW.username AND isbn = NEW.isbn AND cancel_date >
CURDATE();

    IF (countBorrowed > 0 OR countReserved > 0) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'A user cannot reserve
a book if they already have it reserved or borrowed';
    END IF;
END ;;


CREATE TRIGGER reservation_limit_check
BEFORE INSERT ON reservation
FOR EACH ROW
BEGIN
    DECLARE countReserved INT;
    DECLARE userRole VARCHAR(256);

    SELECT COUNT(*) INTO countReserved
    FROM reservation
    WHERE username = NEW.username AND cancel_date > CURDATE();

    SELECT role INTO userRole
    FROM user
    WHERE username = NEW.username;

    IF (userRole NOT IN ('student', 'professor')) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Only a student or a professor can reserve
books';
    ELSEIF (userRole = 'student' AND countReserved >= 2) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A student can only have up to 2 books reserved
at a time';
    ELSEIF (userRole = 'professor' AND countReserved >= 1) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A professor can only have up to 1 book
```

```
reserved at a time';
    END IF;
END ;;


CREATE TRIGGER borrowing_limit_check
BEFORE INSERT ON borrowing
FOR EACH ROW
BEGIN
    DECLARE countBorrowed INT;
    DECLARE userRole ENUM('administrator', 'operator', 'student',
'professor');

    SELECT COUNT(*) INTO countBorrowed
    FROM borrowing
    WHERE username = NEW.username AND return_date IS NULL;

    SELECT role INTO userRole
    FROM user
    WHERE username = NEW.username;

    IF (userRole NOT IN ('student', 'professor')) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Only a student or a professor can borrow
books';
    ELSEIF (userRole = 'student' AND countBorrowed >= 2) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A student can only have up to 2 books borrowed
at a time';
    ELSEIF (userRole = 'professor' AND countBorrowed >= 1) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A professor can only have up to 1 book
borrowed at a time';
    END IF;
END ;;


CREATE TRIGGER book_to_reserve_exists_in_school
BEFORE INSERT ON reservation
FOR EACH ROW
```

```
BEGIN
    DECLARE available_books INT;

    SELECT COUNT(*) INTO available_books
    FROM school_has_book
    WHERE school_id = NEW.school_id AND isbn = NEW.isbn;

    IF available_books = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'This school does not have this book';
    END IF;
END ;;


CREATE TRIGGER book_to_borrow_exists_in_school
BEFORE INSERT ON borrowing
FOR EACH ROW
BEGIN
    DECLARE available_books INT;

    SELECT COUNT(*) INTO available_books
    FROM school_has_book
    WHERE school_id = NEW.school_id AND isbn = NEW.isbn;

    IF available_books = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'This school does not have this book';
    END IF;
END ;;


CREATE TRIGGER available_copy_check
BEFORE INSERT ON borrowing
FOR EACH ROW
BEGIN
    DECLARE borrowed_books INT;
    DECLARE total_books INT;

    SELECT num_of_copies INTO total_books
    FROM school_has_book
```

```
    WHERE school_id = NEW.school_id
    AND isbn = NEW.isbn;

    SELECT COUNT(*) INTO borrowed_books
    FROM borrowing
    WHERE school_id = NEW.school_id
    AND isbn = NEW.isbn
    AND return_date IS NULL;

    IF borrowed_books >= total_books THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No available copies';
    END IF;
END ;;


CREATE TRIGGER pending_borrowing_check
BEFORE DELETE ON user
FOR EACH ROW
BEGIN
    DECLARE pending INT;
    SELECT COUNT(*) INTO pending
    FROM borrowing
    WHERE username = OLD.username AND return_date IS NULL;

    IF pending > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot delete user with pending borrowings';
    END IF;
END ;;

DELIMITER ;
```

Αυτά τα triggers έχουν στόχο να διατηρήσουν την ακεραιότητα της βάσης σύμφωνα με τους κανόνες που ορίζονται στην εκφώνηση της εργασίας.

borrowing_overdue_check
Διασφαλίζει πως ένας χρήστης δεν μπορεί να δανειστεί ένα βιβλίο εάν εκκρεμεί δανεισμός που καθυστέρησε.
Πριν από την εισαγωγή ενός νέου δανεισμού στη βάση, εξετάζονται όλοι οι δανεισμοί του χρήστη, και αν υπάρχει εκκρεμής δανεισμός διάρκειας μεγαλύτερης από 7 μέρες, τότε ο νέος δανεισμός απορρίπτεται.

reservation_overdue_check
Όμοια με borrowing_overdue_check, αλλά για κρατήσεις.

borrowing_not_already_borrowed_or_reserved
Διασφαλίζει πως ένας χρήστης δεν μπορεί να δανειστεί ένα βιβλίο, εάν υπάρχει εν ισχύ δανεισμός ή κράτηση του ίδιου τίτλου.
Πριν από την εισαγωγή ενός νέου δανεισμού στη βάση, εξετάζονται όλοι οι δανεισμοί και οι κρατήσεις του χρήστη, και αν υπάρχει εκκρεμής δανεισμός ή κράτηση του ίδιου βιβλίου, τότε ο νέος δανεισμός απορρίπτεται.

reservation_not_already_borrowed_or_reserved
Όμοια με borrowing_not_already_borrowed_or_reserved, αλλά για κρατήσεις.

borrowing_limit_check
Διασφαλίζει το πλήθος βιβλίων που επιτρέπεται να δανειστεί ένας χρήστης.
Ένας μαθητής μπορεί να δανειστεί μέχρι 2 βιβλία τη φορά.
Ένας καθηγητής μπορεί να δανειστεί μέχρι 1 βιβλίο τη φορά.
Κανένας άλλος χρήστης δεν μπορεί να δανειστεί βιβλία.
Πριν από την εισαγωγή ενός νέου δανεισμού στη βάση, εξετάζονται ο ρόλος του χρήστη και όλοι οι εκκρεμείς δανεισμοί του, και αν παραβιάζονται οι παραπάνω συνθήκες, τότε ο νέος δανεισμός απορρίπτεται.

reservation_limit_check
Όμοια με borrowing_limit_check, αλλά για κρατήσεις.

book_to_borrow_exists_in_school
Διασφαλίζει πως το συγκεκριμένο βιβλίο υπάρχει στο συγκεκριμένο σχολείο.

book_to_borrow_exists_in_school
Όμοια με book_to_borrow_exists_in_school, αλλά για κρατήσεις.

available_copy_check
Διασφαλίζει πως το συγκεκριμένο σχολείο έχει τουλάχιστον ένα διαθέσιμο αντίτυπο του βιβλίου προς δανεισμό.
Πριν από την εισαγωγή ενός νέου δανεισμού στη βάση, εξετάζονται το πλήθος των συνολικών αντιτύπων του βιβλίου και το πλήθος των δανεισμένων αντιτύπων του βιβλίου, προκειμένου να υπολογισθεί το πλήθος των διαθέσιμων αντιτύπων. Εάν δεν υπάρχουν διαθέσιμα αντίτυπα, τότε ο νέος δανεισμός απορρίπτεται.

pending_borrowing_check
Διασφαλίζει πως ένας χρήστης δεν μπορεί να διαγραφεί, εάν υπάρχει εν ισχύ δανεισμός.
Πριν από τη διαγραφή ενός χρήστη, εξετάζονται όλοι οι δανεισμοί του χρήστη, και αν υπάρχει εκκρεμής δανεισμός, τότε η διαγραφή απορρίπτεται.

4.3 Indexes

Τα ευρετήρια που υλοποιήθηκαν για τη βάση βρίσκονται στο SQL αρχείο:
https://github.com/papaskal/Database-systems/blob/main/SQL/indexes.sql
Πρόκειται για τον κάτωθι κώδικα:

```
USE db_2023;

CREATE INDEX user_role_idx ON user (role);

CREATE INDEX school_has_book_num_of_copies_idx ON school_has_book
(num_of_copies);

CREATE INDEX borrowing_return_date_idx ON borrowing (return_date);

CREATE INDEX book_title_idx ON book (title);

CREATE INDEX author_full_name_idx ON author (full_name);
```

Αυτά τα index έχουν σκοπό να επιταχύνουν τα queries που απαιτούνται για τις λειτουργίες που ζητούνται στην εκφώνηση της εργασίας.

Αξίζει να αναφέρουμε τα εξής σημεία:

- Κάθε primary key ενός table γίνεται indexed αυτόματα από το DBMS (MariaDB), χωρίς να χρειάζεται να δημιουργήσουμε νέο index εμείς. Αυτό είναι πολύ χρήσιμο στα περισσότερα queries.
- Κάθε column ενός table, το οποίο αποτελεί foreign key ενός άλλου table, γίνεται indexed αυτόματα από το DBMS (MariaDB), χωρίς να χρειάζεται να δημιουργήσουμε νέο index εμείς. Αυτό είναι πολύ χρήσιμο στα JOIN operations που υλοποιούμε στον κώδικά μας.
- Ένα composite index σε πολλαπλά columns, είναι χρήσιμο και για οποιοδήποτε prefix αυτών των columns.
- Ένα index σε ένα column δεν βοηθάει σε μια συνθήκη της μορφής LIKE ‘%*input_string*%’ , καθώς το αρχικό % δεν επιτρέπει τη γρήγορη αναζήτηση μέσω index. Αντ’ αυτού απαιτείται full table scan.
  Στην παρούσα εργασία, όταν για μια λειτουργία απαιτούνταν ένα κριτήριο αναζήτησης, συνήθως χρησιμοποιήσαμε συνθήκη της παραπάνω μορφής. Συνεπώς, ένα index δεν μπορεί να βοηθήσει πχ για την αναζήτηση βιβλίων βάση κατηγορίας.
- Ένα index σε ένα column δεν μπορεί να βοηθήσει εάν η συνθήκη απαιτεί αυτό το column να είναι μέσα σε ένα function. Για παράδειγμα, για την εύρεση δανεισμών σε ένα έτος, χρησιμοποιούμε συνθήκη της μορφής YEAR(borrow_date). Ένα index στο borrow_date δεν θα ήταν χρήσιμο.

user_role_idx
Αρκετές από τις απαιτούμενες λειτουργίες ζητούν πληροφορίες για τους καθηγητές. Συνεπώς, αυτό το index επιταχύνει queries με συνθήκες της μορφής
WHERE user.role = ‘professor’

school_has_book_num_of_copies_idx
Στην πρώτη λειτουργία που ζητείται για το χειριστή, απαιτείται (πέρα από τα υπόλοιπα) αναζήτηση βιβλίων με κριτήριο το πλήθος των διαθέσιμων αντιτύπων.

book_title_idx
Όπως προαναφέραμε, αυτό το index δε βοηθάει στην αναζήτηση βιβλίου με κριτήριο τον τίτλο καθώς το υλοποιούμε με συνθήκη της μορφής LIKE ‘%*partial_title*%’ .
Ωστόσο, στην πρώτη λειτουργία του χειριστή απαιτείται και η προαιρετική ταξινόμηση των βιβλίων κατά τίτλο. Σε αυτό πράγματι βοηθάει το index.

author_full_name_idx
Όμοια με το book_title_idx, αυτό το index δε βοηθάει στην αναζήτηση βιβλίου με κριτήριο τον συγγραφέα, αλλά βοηθάει στην προαιρετική ταξινόμηση των βιβλίων κατά συγγραφέα.

borrowing_return_date_idx
Βοηθάει πολύ σε λειτουργίες όπου ψάχνουμε εκκρεμείς δανεισμούς, καθώς σε ένα εκκρεμή δανεισμό το πεδίο return_date θα είναι null.

5. DML

Λόγω του μεγέθους του σχετικού κώδικα, δεν μπορεί να παρατεθεί εξ ολοκλήρου στην αναφορά.
Ο σχετικός κώδικας βρίσκεται στα εξής 2 αρχεία:

- https://github.com/papaskal/Database-systems/blob/main/SQL/data.sql
  Πρόκειται για ένα SQL script με χιλιάδες insertion για την αρχικοποίηση της βάσης με dummy δεδομένα.
  Τα δεδομένα παρήχθησαν με τη χρήση του nodeJS script:
  https://github.com/papaskal/Database-systems/blob/main/data_generator/datagen.js
  με τη βοήθεια της βιβλιοθήκης faker.

- https://github.com/papaskal/Database-systems/blob/main/app/app.js
  Αυτό αποτελεί το “κύριο” nodejs script της εφαρμογής μας.
  Σε αυτό υλοποιούνται οι λειτουργίες που ζητούνται στην εκφώνηση της εργασίας.
  Αξιοποιείται η βιβλιοθήκη mariadb για την αποστολή SQL εντολών υπό μορφή string στη βάση.

6. User manual

Εγκατάσταση της εφαρμογής

- Απαιτείται το XAMPP για τη χρήση του DBMS MySQL (MariaDB)
- Απαιτείται το node, npm για το backend
- Υποθέτουμε πως το mysql και το node έχουν γίνει add to path
- Πλοηγούμαστε στο φάκελο “~/app/” με την εντολή
  cd “~/app/”
  στο terminal
- Εκτελούμε την εντολή
  npm install
  για την εγκατάσταση των απαραίτητων node_modules για τη λειτουργία της εφαρμογής

Αρχικοποίηση της βάσης

- Ανοίγουμε το XAMPP και εκκινούμε τον MySQL server
- Πλοηγούμαστε στο φάκελο “~/SQL/” με την εντολή
  cd “~/SQL”
  στο terminal
- Εκτελούμε με τη σειρά τις παρακάτω εντολές
  mysql -u root -p <”./tables.sql”
  mysql -u root -p <”./indexes.sql”
  mysql -u root -p <”./data.sql”
  mysql -u root -p <”./triggers.sql”
  Προσοχή: Με τις παραπάνω εντολές, θα δημιουργηθεί (και θα γεμίσει) μια νέα βάση με όνομα db_2023. Εάν προϋπήρχε μια βάση με το ίδιο όνομα, θα χαθεί.

Εκτέλεση της εφαρμογής

- Ανοίγουμε το XAMPP και εκκινούμε τον MySQL server στην port 3306
- Πλοηγούμαστε στο φάκελο “~/app/” με την εντολή
  cd “~/app/”
  στο terminal
- Εκτελούμε την εντολή
  node app.js
- Σε έναν browser ανοίγουμε τη σελίδα
  http://localhost:3000/

Λειτουργία της εφαρμογής

Βρισκόμαστε στο http://localhost:3000/
Η λειτουργία της εφαρμογής είναι αρκετά απλή και διαισθητική.
Αρχικά, επιλέγουμε το ρόλο που μας ενδιαφέρει.
Άπαξ και επιλέξουμε ένα ρόλο, οδηγούμαστε στην αντίστοιχη σελίδα.
Κάθε τέτοια σελίδα, περιέχει τις λειτουργίες που ζητούνται στην εκφώνηση της εργασίας με τη σειρά.
Για να εκτελέσουμε μια από τις λειτουργίες, συμπληρώνουμε την αντίστοιχη φόρμα (αν υπάρχει και απαιτείται) και πατάμε το κουμπί “Submit”.

Συγκεκριμένα, η αντιστοιχία των ερωτημάτων της εκφώνησης (τρίτο μέρος) με τις λειτουργίες της εφαρμογής είναι η εξής (**οι επιπλέον λειτουργίες που δεν αποτελούν συγκεκριμένα ερωτήματα στο τρίτο μέρος της εκφώνησης, τονίζονται με bold**) :

1. Administrator
    1 : Loans per school
    2 : Authors and teachers by category
    3 : Young teachers who borrowed most books
    4 : Authors whose books have not been borrowed
    5 : Loans per operator above 20
    6 : Top 3 category pairs in borrowings
    7 : Authors with at least 5 books less than the author with most books

2. Operator
    1: Search books
    2: Borrowers with delays
    3: Average ratings
    Προσθήκη νέου χρήστη : Add new user

3. User
    1: Search books
        Σημειώνουμε πως εδώ, πέρα από τις απαιτήσεις της εκφώνησης, με την επιλογή ενός βιβλίου, **βλέπουμε όλες τις πληροφορίες για το βιβλίο.**
        Επίσης, **μπορούμε να δούμε όλα τα reviews για το βιβλίο, καθώς και να δημιουργήσουμε νέα reviews.**
        Επιπλέον, **μπορούμε να επιτελέσουμε τις λειτουργίες της κράτησης και του δανεισμού βιβλίου** (εφόσον δεν παραβιάζονται περιορισμοί).
    2: View user info
        Σημειώνουμε πως εδώ, πέρα από τις απαιτήσεις της εκφώνησης, με την επιλογή ενός χρήστη, **βλέπουμε όλες τις πληροφορίες του χρήστη, καθώς επίσης και τους δανεισμούς και κρατήσεις του χρήστη** (παρελθόντες και παρόντες) .
        Επίσης, **μπορούμε να κάνουμε update τις πληροφορίες του χρήστη ή ακόμα και να διαγράψουμε το χρήστη από τη βάση**.
        Επιπλέον, **μπορούμε να ολοκληρώσουμε εκκρεμείς δανεισμούς και κρατήσεις.**