Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Εξάμηνο 6ο

Εξαμηνιαία Εργασία στις Βάσεις Δεδομένων

ΟΜΑΔΑ 44

Γεώργιος Γεωργακόπουλος 03120827

Εμμανουήλ Νεοφώτιστος 03119128

Ιωάννα Μπαμπαρούτση 03120081

ER Διάγραμμα

Για τον εννοιολογικό σχεδιασμό της βάσης καταλήξαμε στα εξής entity sets:

school: τα σχολεία που είναι εγγεγραμμένα στο δίκτυο σχολικών βιβλιοθηκών

lib_user: οι χρήστες οι οποίοι έχουν λογαριασμό τη βάση

book: τα βιβλία τα οποία βρίσκονται στη βιβλιοθήκη κάποιου/ων εγγεγραμμένου/ων σχολείου/ων

book_status: οι καταστάσεις κράτησης/ δανεισμού που δημιουργεί ένας user (κατόπιν ανάλογου αιτήματος) και αναφέρεται στο βιβλίο της επιλογής του.

Επίσης χρησιμοποιήθηκαν τα εξής relationship sets:

exists_in: σύνδεση lib_user και school

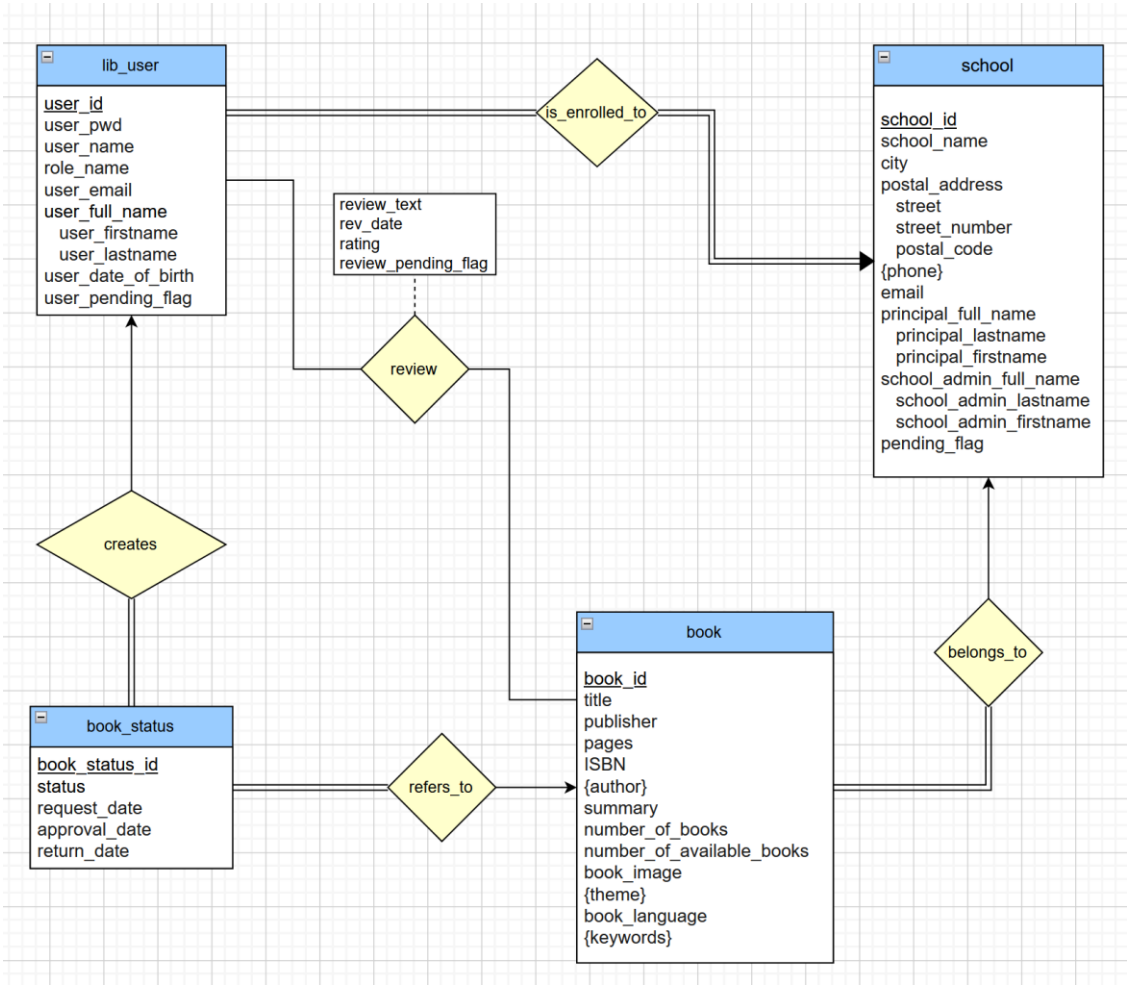
belongs_to: σύνδεση book και school

review: οι αξιολογήσεις που κάνουν οι χρήστες στα βιβλία (σύνδεση book και lib_user) με attributes: review_text, rev_date, rating, review_pending_flag

creates: σύνδεση lib_user με book_status (δημιουργία κράτησης/δανεισμού από τον χρήστη)

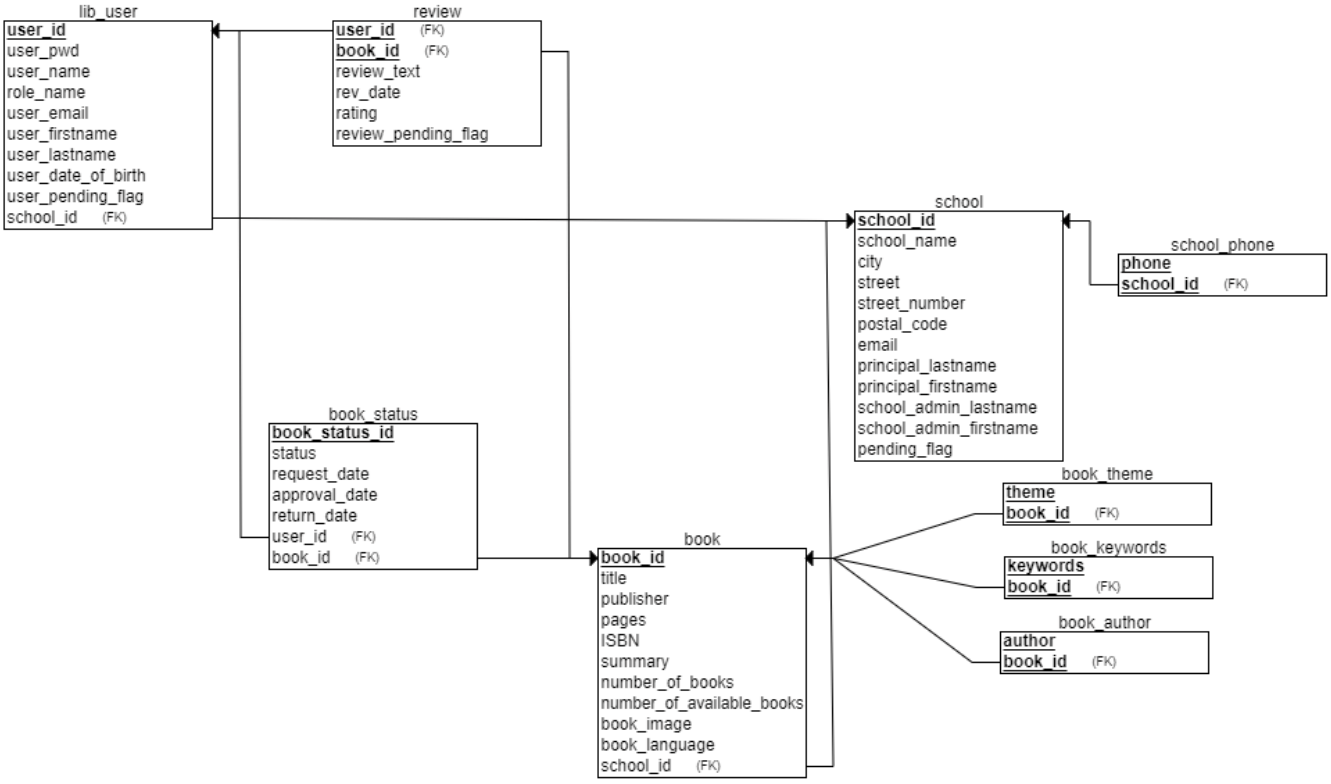
refers_to: σύνδεση book_status και book (ποιο βιβλίο αφορά η κράτηση/ο δανεισμός)

Παρακάτω παρατίθεται το ER Diagram με τους πίνακες, τα χαρακτηριστικά (attributes) τους και τις μεταξύ τους σχέσεις.



Σχεσιακό Διάγραμμα

Παρακάτω παραθέτουμε το σχεσιακό διάγραμμα, το οποίο προέκυψε άμεσα από τη δομή και τη διάρθρωση του ER διαγράμματος:



Οι κανόνες που χρησιμοποιήσαμε για την κατασκευή του σχεσιακού διαγράμματος από το ER-διάγραμμα που παρουσιάσαμε παραπάνω, είναι οι εξής:

• Για κάθε οντότητα (entity) δημιουργούμε ένα table το οποίο περιέχει όλα τα simple attributes αυτής.

• Όταν έχουμε μία δυαδική 1:N σχέση στο ER-διάγραμμα, τότε στο σχεσιακό διάγραμμα βάζουμε το primary key της “1” οντότητας ως foreign key στο table της “N” οντότητας.

• Όταν έχουμε μία δυαδική M:N σχέση στο ER-διάγραμμα, τότε στο σχεσιακό διάγραμμα δημιουργούμε ένα ξεχωριστό table του οποίου το primary key ειναι ο συνδυασμος των primary keys των δυο οντοτητων που συμμετεχουν στη σχεση. Το table αυτο περιεχει, επισης , οποιοδηποτε attribute της ιδιας της σχεσης που εμφανιζονται στο ER- διαγραμμα.

• Για κάθε multivalued attribute σε ένα entity set, δημιουργούμε ξεχωριστό table. Το primary key του συνόλου οντοτήτων μπαίνει ως foreign key σε αυτό το table. Η τιμή του attribute σε συνδυασμό με το foreign key, αποτελούν το primary key του παραγόμενου πίνακα.

Παρατηρούμε λοιπόν, οτι στο σχεσιακό διάγραμμα προστίθενται οι ακόλουθοι πίνακες:

school_phone,

book_keywords,

book_theme,

book_author: διότι αντιστοιχούν σε πλειότιμα χαρακτηριστικά(multivalued attributes) των entity sets

και ο πίνακας:

review ως many-to-many relationships του ER.

Από την άλλη, οι one-to-many relationships **belongs_to**, **exists_in** έχουν σαν αποτέλεσμα την προσθήκη των foreign keys που επισημαίνονται στο διάγραμμα.

Indexes

Προκειμένου να βελτιώσουμε την απόδοση των queries της βάσης μας και να μειώσουμε τον χρόνο που απαιτεί η αναζήτηση των απαραίτητων κάθε φορά δεδομένων, προσθέτουμε στη βάση τα παρακάτω indexes.

```
CREATE INDEX idx_book_status_status ON book_status (status);
CREATE INDEX idx_book_theme_theme ON book_theme (theme);
CREATE INDEX idx_book_author_author ON book_author (author);
CREATE INDEX idx_book_status_user_id ON book_status (user_id);
CREATE INDEX idx_book_status_approval_date ON book_status (approval_date);
CREATE INDEX idx_lib_user_name ON lib_user (user_firstname, user_lastname);
CREATE INDEX idx_book_title ON book (title);
```

Τα indexes, λοιπόν, λειτουργούν σαν γρήγορα ευρετήρια των δεδομένων στα οποία απαιτείται συχνή πρόσβαση από τα queries.

Σχετικά με τα ευρετήρια (indexes) η MySQL δημιουργεί από μόνη της ευρετήρια για όλα τα Primary Keys. Επίσης, για εκδόσεις της MySQL μεγαλύτερες της 5.5, όπου η InnoDB είναι η default storage engine, δημιουργούνται αυτόματα ευρετήρια και για όλα τα foreign keys. Αυτό ισχύει και στη δική μας περίπτωση.

```
george@glaptop:~/Workshop/uni/dblab$ mysql -V
mysql  Ver 8.0.33-0ubuntu0.22.04.2 for Linux on x86_64 ((Ubuntu))
```

```
MariaDB [library]> SELECT SUPPORT FROM INFORMATION_SCHEMA.ENGINES WHERE ENGINE = "InnoDB";
+---------+
| SUPPORT |
+---------+
| DEFAULT |
+---------+
1 row in set (0,001 sec)
```

Έτσι, λοιπόν, δεν χρειάζεται να δημιουργήσουμε ευρετήρια για τα primary και foreign keys της βάσης μας (κάτι το οποίο θα ήταν απαραίτητο έτσι ώστε να επιταχυνθούν οι εντολές JOIN).

Περιορισμοί (constraints) και παραδοχές για την ακεραιότητα και την λογική ορθότητα της βάσης

Ο super_admin δεν έχει δικαίωμα δανεισμού καθώς θεωρείται πως δεν ανήκει στο διδακτικό προσωπικό, αλλά είναι διοικητικός υπάλληλος/γραμματειακή υποστήριξη που όμως έχει έδρα σε κάποιο σχολείο.

Ο admin εφόσον είναι καθηγητής έχει τα ίδια δικαιώματα δανεισμού με όλους τους καθηγητές.

Ένας καθηγητής (άρα και ο admin) πρέπει να είναι τουλάχιστον 23 ετών.

Ένας μαθητής πρέπει να είναι τουλάχιστον 7 ετών.

Στους μαθητές δεν βάζουμε ανώτατο όριο ηλικίας, καθώς λαμβάνουμε υπόψη μαθητές που έχουν χάσει χρονιές, φοιτούν σε σχολεία 2ης ευκαιρίας κλπ.

Τρόπος υλοποίησης κράτησης και δανεισμού:

Σε περίπτωση που το επιθυμητό βιβλίο είναι διαθέσιμο, τότε ο χρήστης θα μπορεί να δεσμεύσει ένα αντίτυπο (γίνεται reserved την request_date).
Ο δανεισμός καταχωρείται από τον υπεύθυνο χειριστή, όταν ο χρήστης προσέλθει στη βιβλιοθήκη για να παραλάβει το βιβλίο του (γίνεται borrowed την approval_date).

Τόσο οι δανεισμοί όσο και οι κρατήσεις (χωρίς διαθέσιμα αντίτυπα) ακολουθούν τους κατωτέρω περιορισμούς:
2/εβδομάδα για μαθητή
1/εβδομάδα για καθηγητή

Να επισημάνουμε ότι στους δανεισμούς προσμετρώνται τόσο η ηλεκτρονική δέσμευση ενός διαθέσιμου αντιτύπου (reserved) όσο και ο απευθείας (δια ζώσης) δανεισμός (borrowed). Για παράδειγμα, στην περίπτωση του μαθητή το ανώτατο εβδομαδιαίο όριο των 2 βιβλίων μπορεί να προκύψει με τους παρακάτω πιθανούς συνδυασμούς:

1. Δύο βιβλία δανεισμένα αυτή την εβδομάδα
2. Δύο διαθέσιμα βιβλία ηλεκτρονικά δεσμευμένα αυτή την εβδομάδα
3. Ένα βιβλίο δανεισμένο και ένα άλλο ηλεκτρονικά δεσμευμένο αυτή την εβδομάδα

Σύμφωνα με την χρονική αλληλουχία της διαδικασίας δανεισμού, προκύπτουν οι ακόλουθοι περιορισμοί: η ημερομηνία ηλεκτρονικής κράτησης (request_date) πρέπει να είναι προγενέστερη της ημερομηνίας έγκρισης δανεισμού (approval_date), η οποία με τη σειρά της είναι προγενέστερη της ημερομηνίας επιστροφής (return_date).

Ένας χρήστης μπορεί να κάνει κράτηση ενός βιβλίου σε περίπτωση που αυτό δεν είναι διαθέσιμο και να μπει έτσι σε μια σειρά αναμονής (queue).
Στην περίπτωση ενός βιβλίου χωρίς διαθέσιμα αντίτυπα, για το οποίο έχει σχηματιστεί μία ουρά αναμονής, άπαξ και προκύψει κάποιο διαθέσιμο αντίτυπο (όταν γίνει κάποιο return) τότε ο χρήστης με το παλαιότερο αίτημα κράτησης μπορεί πλέον να δανειστεί το βιβλίο (μετάβαση από κατάσταση queue σε κατάσταση reserved). Επισημαίνουμε ότι σε περίπτωση που ο χρήστης με το παλαιότερο αίτημα κράτησης έχει φτάσει το ανώτατο όριο δανεισμών αυτή την εβδομάδα, τότε τον προσπερνάμε και εξυπηρετείται ο αμέσως επόμενος χρήστης στην ουρά αναμονής.
Κατά τη διαδικασία αυτή γίνεται update του πεδίου request_date και εξασφαλίζεται ότι:
NEW.request_date>OLD.request_date

Τέλος, έχουμε:

1. κωδικούς χρηστών της μορφής τεσσάρων χαρακτήρων (αριθμοί/γράμματα/σύμβολα κ.ο.κ)
2. τηλέφωνα σχολείων με 10 ψηφία (το δεύτερο τηλέφωνο δεν είναι υποχρεωτικό)
3. πενταψήφιους ταχυδρομικούς κώδικες
4. προαιρετική προσθήκη δεύτερου συγγραφέα και λέξεων-κλειδιών κατά την προσθήκη νέου βιβλίου από τον υπεύθυνο χειριστή.

DDL script

Στα DDL scripts περιλαμβάνεται το **library_schema.sql** (το οποίο βρίσκεται στον φάκελο sql του repository στο Github) στο οποίο δημιουργούνται τα tables, τα events, τα views και τα triggers της βάσης, τα οποία παραθέτουμε:

Tables

```
--- Table 'school'
CREATE TABLE school (
  school_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  school_name VARCHAR(45) NOT NULL,
  city VARCHAR(45) NOT NULL,
  street VARCHAR(45) NOT NULL,
  street_number INT UNSIGNED NOT NULL,
  postal_code INT UNSIGNED NOT NULL,
  email VARCHAR(45) NOT NULL,
  principal_lastname VARCHAR(45) NOT NULL,
  principal_firstname VARCHAR(45) NOT NULL,
  school_admin_lastname VARCHAR(45) NOT NULL,
  school_admin_firstname VARCHAR(45) NOT NULL,
  pending_flag ENUM('pending'),
  last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  CHECK(postal_code > 9999 and postal_code < 100000),
  PRIMARY KEY  (school_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

--- Table 'school_phone'
CREATE TABLE school_phone(
  phone VARCHAR(10) NOT NULL,
  school_id INT UNSIGNED NOT NULL,
  phone_flag ENUM('pending'),
  last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (phone,school_id),
  KEY fk_school_id (school_id),
  CONSTRAINT fk_school_id FOREIGN KEY (school_id) REFERENCES school (school_id) ON DELETE
CASCADE ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Table 'lib_user'
CREATE TABLE lib_user(
    user_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    user_pwd VARCHAR(10) NOT NULL,
    user_name VARCHAR(45) NOT NULL,
    school_id INT UNSIGNED NOT NULL,
    role_name ENUM('student', 'teacher', 'admin', 'super_admin') NOT NULL,
    user_email VARCHAR(45) NOT NULL,
    user_firstname VARCHAR(45) NOT NULL,
    user_lastname VARCHAR(45) NOT NULL,
    user_date_of_birth DATE NOT NULL,
    user_pending_flag ENUM('waiting'),
    last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (user_id),
    KEY fk_user_school_id (school_id),
```

```
    CONSTRAINT fk_user_school_id FOREIGN KEY (school_id) REFERENCES school (school_id) ON
DELETE CASCADE ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Table 'book'
CREATE TABLE book (
  book_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  title VARCHAR(45) NOT NULL,
  publisher VARCHAR(45) NOT NULL,
  pages INT UNSIGNED NOT NULL,
  ISBN BIGINT UNSIGNED NOT NULL,
  summary VARCHAR(400),
  number_of_books INT UNSIGNED NOT NULL,
  number_of_available_books INT UNSIGNED NOT NULL,
  book_image VARCHAR(256) NOT NULL,
  book_language VARCHAR(45),
  school_id INT UNSIGNED NOT NULL,
  last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (book_id),
  KEY fk_book_school_id (school_id),
  CONSTRAINT fk_book_school_id FOREIGN KEY (school_id) REFERENCES school (school_id) ON
DELETE CASCADE ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Table book_status
CREATE TABLE book_status (
  book_status_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  book_id INT UNSIGNED NOT NULL,
  user_id INT UNSIGNED NOT NULL,
  status ENUM('borrowed','reserved','queue') NOT NULL,
  request_date DATE,
  approval_date DATE,
  return_date DATE,
  PRIMARY KEY (book_status_id),
  KEY fk_book_status_book_id (book_id),
  KEY fk_book_status_user_id (user_id),
  CONSTRAINT fk_book_status_book_id FOREIGN KEY (book_id) REFERENCES book (book_id) ON
DELETE CASCADE,
  CONSTRAINT fk_book_status_user_id FOREIGN KEY (user_id) REFERENCES lib_user (user_id) ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;



-- Table 'book_keywords'
CREATE TABLE book_keywords (
  keywords  VARCHAR(50) NOT NULL,
  book_id INT UNSIGNED NOT NULL,
  last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (keywords,book_id),
  KEY fk_book_keywords_book_id (book_id),
  CONSTRAINT fk_book_keywords_book_id FOREIGN KEY(book_id) REFERENCES book (book_id) ON
DELETE CASCADE ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
-- Table 'book_theme'
CREATE TABLE book_theme (
  theme ENUM('Fiction', 'Non-fiction','Dystopia','Gothic','Tragedy','Science Fiction',
'Science','Drama', 'Adventure','Mystery', 'Romance','War', 'Classic','Thriller', 'Horror',
'Fantasy', 'Biography', 'Autobiography', 'History', 'Poetry', 'Comics', 'Cookbooks',
'Travel', 'Religion', 'Self-help', 'Art', 'Music','Coming of Age', 'Sports', 'Humor',
'Children','Reference') NOT NULL,
  book_id INT UNSIGNED NOT NULL,
  last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (theme, book_id),
  KEY fk_book_theme_book_id (book_id),
  CONSTRAINT fk_book_theme_book_id FOREIGN KEY(book_id) REFERENCES book (book_id) ON
DELETE CASCADE ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Table 'book_author'
CREATE TABLE book_author (
  author  VARCHAR(50) NOT NULL,
  book_id INT UNSIGNED NOT NULL,
  last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (author,book_id),
  KEY fk_book_author_book_id (book_id),
  CONSTRAINT fk_book_author_book_id FOREIGN KEY(book_id) REFERENCES book (book_id) ON
DELETE CASCADE ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Table 'review'
CREATE TABLE review (
  user_id INT UNSIGNED NOT NULL,
  book_id INT UNSIGNED NOT NULL,
  review_text VARCHAR(400),
  rev_date DATE NULL,
  rating ENUM('1', '2', '3', '4', '5') NOT NULL,
  review_pending_flag ENUM('pending'),
  PRIMARY KEY (user_id,book_id),
  KEY fk_review_user_id (user_id),
  CONSTRAINT fk_review_user_id FOREIGN KEY (user_id) REFERENCES lib_user (user_id) ON
DELETE CASCADE,
  KEY fk_review_book_id (book_id),
  CONSTRAINT fk_review_book_id FOREIGN KEY (book_id) REFERENCES book (book_id) ON DELETE
CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Events

```
---Every day the db needs to check whether a queue deadline has expired
DELIMITER $$
CREATE EVENT delete_from_queue
ON SCHEDULE EVERY 1 DAY
DO
BEGIN
    -- Delete old queued reservations
    DELETE FROM book_status
```

```
    WHERE status = 'queue'
    AND request_date < DATE_SUB(NOW(), INTERVAL 1 WEEK);
END $$
DELIMITER ;

---Every day the db needs to check whether a reservation deadline has expired
DELIMITER $$

CREATE EVENT delete_old_reservations
ON SCHEDULE EVERY 1 DAY
DO
BEGIN
    -- Declare variables
    DECLARE done INT DEFAULT FALSE;
    DECLARE book_id INT;
    DECLARE function_result BOOLEAN;

    -- Cursor to fetch book_id values
    DECLARE cur CURSOR FOR
        SELECT book_id
        FROM book_status
        WHERE status = 'reserved'
        AND request_date < DATE_SUB(NOW(), INTERVAL 1 WEEK);

    -- Declare handlers
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Delete old reservations
    DELETE FROM book_status
    WHERE status = 'reserved'
    AND request_date < DATE_SUB(NOW(), INTERVAL 1 WEEK);

    -- Open cursor
    OPEN cur;

    -- Fetch book_id values and call function/procedure
    read_loop: LOOP
        FETCH cur INTO book_id;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Call your function with book_id as a parameter
        SET function_result = check_book_update(book_id);

        IF function_result = 0 THEN
            -- Call your procedure with book_id as a parameter
            CALL increase_available_books(book_id);
        END IF;
    END LOOP;

    -- Close cursor
    CLOSE cur;
END$$
```

```
DELIMITER ;
```

Views

```
---All schools with their names
CREATE VIEW all_schools AS
SELECT school_id, school_name
FROM school;

-- School_application
CREATE VIEW school_applications AS
SELECT s.school_id, s.school_name, s.city, s.street, s.postal_code, s.email,
       s.principal_lastname, s.principal_firstname, s.school_admin_lastname,
s.school_admin_firstname,
       u.user_id, u.user_name, u.user_email, u.user_firstname, u.user_lastname,
u.user_date_of_birth
FROM school s
JOIN lib_user u ON s.school_id = u.school_id
WHERE s.pending_flag = 'pending' AND u.user_pending_flag = 'waiting' AND u.role_name =
'admin';

--- User applications
CREATE VIEW new_user_application AS
SELECT *
FROM lib_user
WHERE user_pending_flag = 'waiting';
```

Procedures

```
DELIMITER //

CREATE PROCEDURE decrease_available_books(IN _book_id INT)
BEGIN
    UPDATE book
    SET number_of_available_books = number_of_available_books - 1
    WHERE book_id = _book_id;
END //

DELIMITER ;

DELIMITER //

CREATE PROCEDURE increase_available_books(IN _book_id INT)
BEGIN
    UPDATE book
    SET number_of_available_books = number_of_available_books + 1
    WHERE book_id = _book_id;
END //

DELIMITER ;
```

Triggers

```
DELIMITER $$
CREATE TRIGGER check_borrow_limit
BEFORE INSERT ON book_status
FOR EACH ROW
BEGIN
    DECLARE borrow_count INT;
    DECLARE queue_count INT;
    DECLARE reserved_count INT;
    IF (NEW.status = 'reserved' OR NEW.status = 'borrowed') THEN
      IF NEW.user_id IN (SELECT user_id FROM lib_user WHERE role_name='student') THEN
          SET borrow_count = (
              SELECT COUNT(*) AS count
              FROM book_status
              WHERE user_id = NEW.user_id
                AND status IN ('borrowed')
                AND approval_date >= DATE_SUB(NOW(), INTERVAL 7 DAY)
          );
          SET reserved_count = (
              SELECT COUNT(*) AS count
              FROM book_status
              WHERE user_id = NEW.user_id
                AND status IN ('reserved')
                AND request_date >= DATE_SUB(NOW(), INTERVAL 7 DAY)
          );

          IF (borrow_count+reserved_count) >= 2 THEN
              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You have exceeded the limit on
the number of books you can borrow or reserve in the last seven days.';
          END IF;
      ELSEIF NEW.user_id IN (SELECT user_id FROM lib_user WHERE role_name='teacher' OR
role_name='admin') THEN
          SET borrow_count = (
              SELECT COUNT(*) AS count
              FROM book_status
              WHERE user_id = NEW.user_id
                AND status IN ('borrowed')
                AND approval_date >= DATE_SUB(NOW(), INTERVAL 7 DAY)
          );
          SET reserved_count = (
              SELECT COUNT(*) AS count
              FROM book_status
              WHERE user_id = NEW.user_id
                AND status IN ('reserved')
                AND request_date >= DATE_SUB(NOW(), INTERVAL 7 DAY)
          );
          IF (borrow_count+reserved_count) >= 1 THEN
              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You have exceeded the limit on
the number of books you can borrow or reserve in the last seven days.';
          END IF;
      END IF;
    END IF;
END$$
```

```
DELIMITER ;

DELIMITER $$
CREATE TRIGGER check_queue_limit
BEFORE INSERT ON book_status
FOR EACH ROW
BEGIN
    DECLARE borrow_count INT;
    DECLARE queue_count INT;
    DECLARE reserved_count INT;
    IF (NEW.status = 'queue') THEN
      IF NEW.user_id IN (SELECT user_id FROM lib_user WHERE role_name='student') THEN
          SET queue_count = (
              SELECT COUNT(*) AS count
              FROM book_status
              WHERE user_id = NEW.user_id
                AND status IN ('queue')
                AND request_date >= DATE_SUB(NOW(), INTERVAL 7 DAY)
          );

          IF (queue_count) >= 2 THEN
              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You have exceeded the limit on
the number of books you can borrow or reserve in the last seven days.';
          END IF;
      ELSEIF NEW.user_id IN (SELECT user_id FROM lib_user WHERE role_name='teacher' OR
role_name='admin') THEN
          SET queue_count = (
              SELECT COUNT(*) AS count
              FROM book_status
              WHERE user_id = NEW.user_id
                AND status IN ('queue')
                AND request_date >= DATE_SUB(NOW(), INTERVAL 7 DAY)
          );
          IF (queue_count) >= 1 THEN
              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You have exceeded the limit on
the number of books you can borrow or reserve in the last seven days.';
          END IF;
      END IF;
    END IF;
END$$
DELIMITER ;

-- Phones must have 10 digits
DELIMITER //
CREATE TRIGGER phone_length_trigger BEFORE INSERT ON school_phone
FOR EACH ROW
BEGIN
  IF CHAR_LENGTH(NEW.phone) != 10 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Phone number must be 10 digits.';
  END IF;
END //
DELIMITER ;

-- Passwords must have 4 digits
```

```
DELIMITER //
CREATE TRIGGER user_pwd_format_trigger BEFORE INSERT ON lib_user
FOR EACH ROW
BEGIN
  IF CHAR_LENGTH(NEW.user_pwd) != 4 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Phone number must be 10 digits.';
  END IF;
END //
DELIMITER ;

--Trigger για ηλικία καθηγητών:
DELIMITER //

CREATE TRIGGER check_teacher_admin_age
BEFORE INSERT ON lib_user
FOR EACH ROW
BEGIN
  IF (NEW.role_name = 'teacher' OR NEW.role_name = 'admin') AND
     (TIMESTAMPDIFF(YEAR, NEW.user_date_of_birth, CURDATE()) < 23) THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Teachers and admins must be at least 23 years old.';
  END IF;
END//

DELIMITER ;

--Trigger για ηλικία μαθητών:
DELIMITER //

CREATE TRIGGER check_student_age
BEFORE INSERT ON lib_user
FOR EACH ROW
BEGIN
  IF (NEW.role_name = 'student') AND
     (TIMESTAMPDIFF(YEAR, NEW.user_date_of_birth, CURDATE()) < 7) THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Teachers and admins must be at least 7 years old.';
  END IF;
END//

DELIMITER ;

-- Ενας διευθυντής μπορεί να υπάρχει μόνο σε 1 σχολείο
DELIMITER //

CREATE TRIGGER unique_principal_trigger BEFORE INSERT ON school
FOR EACH ROW
BEGIN
  DECLARE principal_count INT;

  SET principal_count = (
    SELECT COUNT(*) FROM school
    WHERE principal_lastname = NEW.principal_lastname
    AND principal_firstname = NEW.principal_firstname
```

```
  );

  IF principal_count > 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'The same principal cannot be assigned to
multiple schools.';
  END IF;
END //

DELIMITER ;

-- Δεν γίνεται εισαγωγή 2ου super admin
DELIMITER //

CREATE TRIGGER trigger_super_admin_check
BEFORE INSERT ON lib_user
FOR EACH ROW
BEGIN
  DECLARE super_admin_count INT;
  SET super_admin_count = (
    SELECT COUNT(*)
    FROM lib_user
    WHERE role_name = 'super_admin'
  );
  IF super_admin_count > 0 AND NEW.role_name = 'super_admin' THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Only one super admin is allowed.';
  END IF;
END //

DELIMITER ;

-- Δεν γίνεται εισαγωγή 2ου admin για ένα σχολείο
DELIMITER //

CREATE TRIGGER trigger_school_admin_check
BEFORE INSERT ON lib_user
FOR EACH ROW
BEGIN
  DECLARE admin_count INT;
  SET admin_count = (
    SELECT COUNT(*)
    FROM lib_user
    WHERE school_id = NEW.school_id AND role_name = 'admin'
  );
  IF admin_count > 0 AND NEW.role_name = 'admin' THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'A school can have only one admin.';
  END IF;
END //

DELIMITER ;
```

Functions

```
--- Function ωστε μια κράτηση να περνάει αυτόματα σε επίπεδο 'reserved' οταν ενα βιβλίο αποκτήσει διαθεσιμο αντιτυπο και
--- εφόσον ο χρήστης που εχει αιτηθεί κράτηση δεν υπερβαίνει τα όρια του δανεισμού. Σε αυτή την περίπτωση επιλέγεται ο επόμενος στην σειρά
--- προτεραιότητας χρηστης.
--- Ο χρήστης ενημερώνεται για αυτή την αλλαγή απο την καρτέλα MyBooks στο προφιλ του.

DELIMITER $$
CREATE FUNCTION check_book_update(_book_id INT) RETURNS BOOLEAN
BEGIN
    DECLARE borrow_count INT;
    DECLARE reserved_count INT;
    DECLARE _user_id INT UNSIGNED;
    DECLARE done INT DEFAULT FALSE;
    DECLARE update_occurred BOOLEAN DEFAULT FALSE;

    DECLARE cur CURSOR FOR
      SELECT user_id AS _user_id FROM book_status
      WHERE book_id = _book_id AND status = 'queue'
      ORDER BY request_date;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    IF (SELECT number_of_available_books FROM book WHERE book_id = _book_id) = 0 THEN

        OPEN cur;

        read_loop: LOOP
          FETCH cur INTO _user_id;
          IF done THEN
            LEAVE read_loop;
          END IF;

          SET borrow_count = (
            SELECT COUNT(*) AS count
            FROM book_status
            WHERE user_id = _user_id
              AND status IN ('borrowed')
              AND approval_date >= DATE_SUB(NOW(), INTERVAL 7 DAY)
          );

          SET reserved_count = (
            SELECT COUNT(*) AS count
            FROM book_status
            WHERE user_id = _user_id
              AND status IN ('reserved')
              AND request_date >= DATE_SUB(NOW(), INTERVAL 7 DAY)
              LIMIT 1
          );

          IF _user_id IN (SELECT user_id FROM lib_user WHERE role_name='student') THEN
              IF borrow_count + reserved_count < 2 THEN
```

```
                UPDATE book_status
                SET status = 'reserved', request_date = CURRENT_DATE
                WHERE book_id = _book_id AND status = 'queue' AND user_id = _user_id
                ORDER BY request_date;
                SET done = TRUE;
                SET update_occurred = TRUE;
              END IF;
          ELSEIF _user_id IN (SELECT user_id FROM lib_user WHERE role_name='teacher' OR
role_name = 'admin') THEN
            IF borrow_count + reserved_count < 1 THEN
                UPDATE book_status
                SET status = 'reserved', request_date = CURRENT_DATE
                WHERE book_id = _book_id AND status = 'queue' AND user_id = _user_id
                ORDER BY request_date;
                SET done = TRUE;
                SET update_occurred = TRUE;
              END IF;
          END IF;
        END LOOP;

        CLOSE cur;
    END IF;

    RETURN update_occurred;
END$$

DELIMITER ;


DELIMITER $$
CREATE FUNCTION check_book_update_after_deny(_book_id INT) RETURNS BOOLEAN
BEGIN
    DECLARE borrow_count INT;
    DECLARE reserved_count INT;
    DECLARE _user_id INT UNSIGNED;
    DECLARE done INT DEFAULT FALSE;
    DECLARE update_occurred BOOLEAN DEFAULT FALSE;

    DECLARE cur CURSOR FOR
      SELECT user_id AS _user_id FROM book_status
      WHERE book_id = _book_id AND status = 'queue'
      ORDER BY request_date;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    IF (SELECT number_of_available_books FROM book WHERE book_id = _book_id) > 0 THEN

        OPEN cur;

        read_loop: LOOP
          FETCH cur INTO _user_id;
          IF done THEN
            LEAVE read_loop;
          END IF;
```

```
          SET borrow_count = (
            SELECT COUNT(*) AS count
            FROM book_status
            WHERE user_id = _user_id
              AND status IN ('borrowed')
              AND approval_date >= DATE_SUB(NOW(), INTERVAL 7 DAY)
          );

          SET reserved_count = (
            SELECT COUNT(*) AS count
            FROM book_status
            WHERE user_id = _user_id
              AND status IN ('reserved')
              AND request_date >= DATE_SUB(NOW(), INTERVAL 7 DAY)
              LIMIT 1
          );

          IF _user_id IN (SELECT user_id FROM lib_user WHERE role_name='student') THEN
              IF borrow_count + reserved_count < 2 THEN
                UPDATE book_status
                SET status = 'reserved', request_date = CURRENT_DATE
                WHERE book_id = _book_id AND status = 'queue' AND user_id = _user_id
                ORDER BY request_date;
                SET done = TRUE;
                SET update_occurred = TRUE;
              END IF;
          ELSEIF _user_id IN (SELECT user_id FROM lib_user WHERE role_name='teacher' OR
role_name = 'admin') THEN
            IF borrow_count + reserved_count < 1 THEN
                UPDATE book_status
                SET status = 'reserved', request_date = CURRENT_DATE
                WHERE book_id = _book_id AND status = 'queue' AND user_id = _user_id
                ORDER BY request_date;
                SET done = TRUE;
                SET update_occurred = TRUE;
              END IF;
            END IF;
        END LOOP;

        CLOSE cur;
    END IF;

    RETURN update_occurred;
END$$

DELIMITER ;
```

DML

Τα DML scripts περιλαμβάνουν το αρχείο **insert_data.sql** μέσα στο οποίο γίνεται η εισαγωγή των αρχικών δεδομένων στη βάση, καθώς και τα views και τα queries. Το αρχείο **insert_data.sql** βρίσκεται στον φάκελο sql του repository στο Github.

Queries

```
---3.1 Superadmin Queries
---3.1.1 List of the total number of borrowings per school  //Κριτήρια Αναζήτησης
SELECT
  school.school_name,
  COUNT(*) AS borrow_count
FROM
  book_status
  INNER JOIN book ON book_status.book_id = book.book_id
  INNER JOIN school ON book.school_id = school.school_id
WHERE
  book_status.status = 'borrowed' AND
  YEAR(book_status.approval_date) = <year> AND
  MONTH(book_status.approval_date) = <month>
GROUP BY
  school.school_name
ORDER BY
  borrow_count DESC;

---3.1.2(a) Authors who belong to a given book theme
SELECT DISTINCT book_author.author
FROM book_theme
INNER JOIN book_author ON book_theme.book_id = book_author.book_id
WHERE book_theme.theme = <your_book_theme>;
---3.1.2(b) Teachers who have borrowed books of a given book theme in the last year
SELECT DISTINCT lib_user.user_name
FROM book_theme
INNER JOIN book ON book_theme.book_id = book.book_id
INNER JOIN book_status ON book.book_id = book_status.book_id
INNER JOIN lib_user ON book_status.user_id = lib_user.user_id
WHERE book_theme.theme = <your_book_theme>
  AND book_status.status = 'borrowed'
  AND book_status.approval_date BETWEEN DATE_SUB(NOW(), INTERVAL 1 YEAR) AND NOW()
  AND (lib_user.role_name = 'teacher'OR lib_user.role_name = 'admin');

---3.1.3 Find the teachers who are younger than 40 years old and have borrowed the most
books as well as the number of the books.
SELECT u.user_id, u.user_firstname, u.user_lastname,u.user_date_of_birth, s.school_name,
COUNT(*) as num_books_borrowed
FROM lib_user u
JOIN school s ON u.school_id = s.school_id
JOIN book_status bs ON u.user_id = bs.user_id
WHERE (u.role_name = 'teacher' OR u.role_name = 'admin' ) AND u.user_date_of_birth >
DATE_SUB(CURDATE(), INTERVAL 40 YEAR) AND bs.status = 'borrowed'
GROUP BY u.user_id
```

```sql
ORDER BY num_books_borrowed DESC;

---3.1.4 List of book authors whose books have not been borrowed
SELECT DISTINCT book_author.author
FROM book_author
LEFT JOIN book_status ON book_author.book_id = book_status.book_id
WHERE book_status.book_id IS NULL;

---3.1.5 Which administrators have registered the same number of more than 20 borrowings
within a year
SELECT lib_user.user_name
FROM book_status
INNER JOIN lib_user ON book_status.user_id = lib_user.user_id
WHERE lib_user.role_name = 'admin'
  AND book_status.approval_date BETWEEN DATE_SUB(NOW(), INTERVAL 1 YEAR) AND NOW()
GROUP BY lib_user.user_id
HAVING COUNT(*) >= 20;
--Το query 3.1.5 διορθωμένο(το ερμήνευσα ως φθίνουσα σειρά αριθμού δανεισμών ανά σχολείο
(>20) με όνομα σχολείου και admin):
SELECT A.school_id, B.school_id, A.borrowed_count, ua1.user_lastname AS admin_lastname1,
ua2.user_lastname AS admin_lastname2
FROM (
    SELECT b.school_id, COUNT(*) AS borrowed_count
    FROM book_status bs
    INNER JOIN book b ON bs.book_id = b.book_id
    WHERE bs.status = 'borrowed' AND YEAR(bs.approval_date) = 2023
    GROUP BY b.school_id
    HAVING borrowed_count > 10
) A
JOIN (
    SELECT b.school_id, COUNT(*) AS borrowed_count
    FROM book_status bs
    INNER JOIN book b ON bs.book_id = b.book_id
    WHERE bs.status = 'borrowed' AND YEAR(bs.approval_date) = 2023
    GROUP BY b.school_id
    HAVING borrowed_count > 10
) B ON A.borrowed_count = B.borrowed_count AND A.school_id <> B.school_id
JOIN lib_user ua1 ON A.school_id = ua1.school_id AND ua1.role_name = 'admin'
JOIN lib_user ua2 ON B.school_id = ua2.school_id AND ua2.role_name = 'admin'
WHERE A.school_id < B.school_id
ORDER BY A.borrowed_count DESC;

---3.1.6 top 3 book theme pairs that appear in borrowings
SELECT
  CONCAT(bt1.theme,',',bt2.theme) AS theme_pair,
  COUNT(*) AS borrow_count
FROM
  book_theme bt1
  INNER JOIN book_theme bt2 ON bt1.book_id = bt2.book_id AND bt1.theme < bt2.theme
  INNER JOIN book_status ON bt2.book_id = book_status.book_id
WHERE
  book_status.status = 'borrowed'
GROUP BY
  theme_pair
```

```sql
ORDER BY
  borrow_count DESC
LIMIT 3;

---3.1.7 Authors who have written at least 5 books less than the author who has written
the most books
SELECT ba.author, COUNT(DISTINCT b.book_id) AS book_count
FROM book_author ba
JOIN book b ON ba.book_id = b.book_id
GROUP BY ba.author
HAVING book_count <= (SELECT MAX(author_book_count) - 5 FROM
  (SELECT COUNT(DISTINCT b2.book_id) AS author_book_count
   FROM book_author ba2
   JOIN book b2 ON ba2.book_id = b2.book_id
   GROUP BY ba2.author) AS author_book_counts)
ORDER BY book_count DESC;

---3.2 Admin Queries
---3.2.1 Present all books by title, author (search criteria: book title/book theme/book
author/number of books)
SELECT b.title, GROUP_CONCAT(ba.author) AS authors
FROM book AS b
JOIN book_author AS ba ON b.book_id = ba.book_id
WHERE b.school_id = <school_id>
GROUP BY b.book_id;
/*Example for search by title
SELECT b.title, GROUP_CONCAT(ba.author) AS authors
FROM book AS b
JOIN book_author AS ba ON b.book_id = ba.book_id
WHERE b.school_id = <school_id> AND b.title = <book_title>
GROUP BY b.book_id
*/

---3.2.2 Find all borrowers who have at their pocession at least one book and they have
delayed the return
SELECT u.user_id, u.user_firstname, u.user_lastname,DATEDIFF(NOW(), bs.approval_date)
FROM lib_user u
JOIN book_status bs ON u.user_id = bs.user_id
WHERE bs.status = 'borrowed' AND bs.return_date IS NULL AND u.school_id = <school_id> AND
(DATEDIFF(NOW(), bs.approval_date) > 7)
/*Example for search by first_name
SELECT u.user_id, u.user_firstname, u.user_lastname,DATEDIFF(NOW(), bs.approval_date)
FROM lib_user u
JOIN book_status bs ON u.user_id = bs.user_id
WHERE bs.status = 'borrowed' AND bs.return_date IS NULL AND u.school_id = <school_id> AND
(DATEDIFF(NOW(), bs.approval_date) > 7) AND u.user_firstname LIKE <user_firstname>
*/

---3.2.3 Average of reviews by borrower and book theme(search criteria: user/book theme)
---Average by user
SELECT lib_user.user_id, CONCAT(lib_user.user_firstname, ' ', lib_user.user_lastname) as
full_name, AVG(review.rating) as avg_rating
FROM lib_user
JOIN review ON lib_user.user_id = review.user_id
```

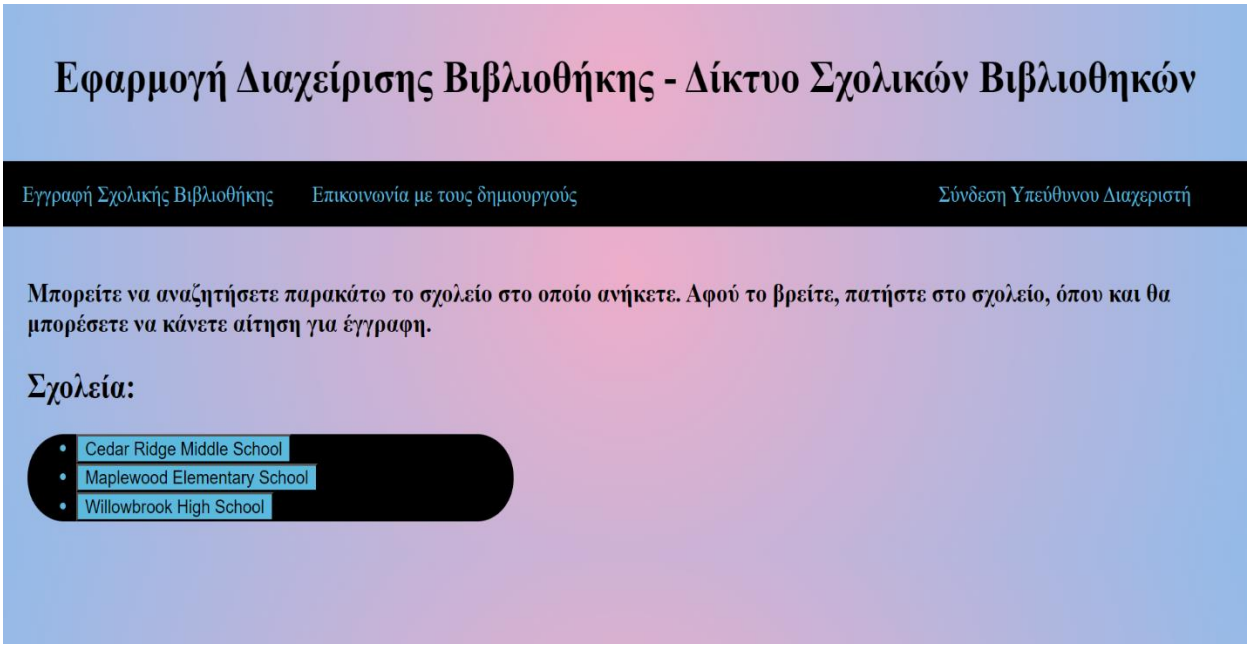
```sql
GROUP BY lib_user.user_id;
---Average by theme
SELECT bt.theme, AVG(r.rating) as avg_rating
FROM book_theme bt
JOIN book b ON bt.book_id = b.book_id
JOIN review r ON b.book_id = r.book_id
GROUP BY bt.theme;
---Average by user and theme
SELECT CONCAT(u.user_firstname, ' ', u.user_lastname) AS borrower_name, bt.theme,
AVG(r.rating) AS avg_rating
FROM lib_user u
INNER JOIN review r ON u.user_id = r.user_id
INNER JOIN book b ON r.book_id = b.book_id
INNER JOIN book_theme bt ON b.book_id = bt.book_id
WHERE u.school_id = %s
GROUP BY u.user_id, bt.theme;


---3.3 User Queries
---3.3.1 All registered books(search criteria: title/book theme/author)
SELECT
  book.book_id,
  book.title,
  book.publisher,
  book.pages,
  book.ISBN,
  book.summary,
  book.number_of_books,
  book.number_of_available_books,
  book.book_image,
  book.book_language,
  GROUP_CONCAT(DISTINCT book_theme.theme SEPARATOR ', ') AS themes,
  GROUP_CONCAT(DISTINCT book_author.author SEPARATOR ', ') AS authors
FROM
  book
  INNER JOIN book_theme ON book.book_id = book_theme.book_id
  INNER JOIN book_author ON book.book_id = book_author.book_id
WHERE
  book.title LIKE '%Title_search%' -- specify the book title
  OR book_theme.theme = 'Theme_search' -- specify the book theme
  OR book_author.author LIKE '%Jane Austen%' -- specify the book author
GROUP BY
  book.book_id;

---3.3.2 List of books that each user has borrowed (we have put user_id,user full
name,book_id,book_title)
SELECT
  lib_user.user_id,
  CONCAT(lib_user.user_firstname, ' ', lib_user.user_lastname) AS user_full_name,
  book.book_id,
  book.title AS book_title
FROM
  book_status
  INNER JOIN lib_user ON book_status.user_id = lib_user.user_id
```

```
  INNER JOIN book ON book_status.book_id = book.book_id
WHERE
  book_status.status = 'borrowed'
ORDER BY
  lib_user.user_id;
```

Πλοήγηση στην εφαρμογή-User Manual

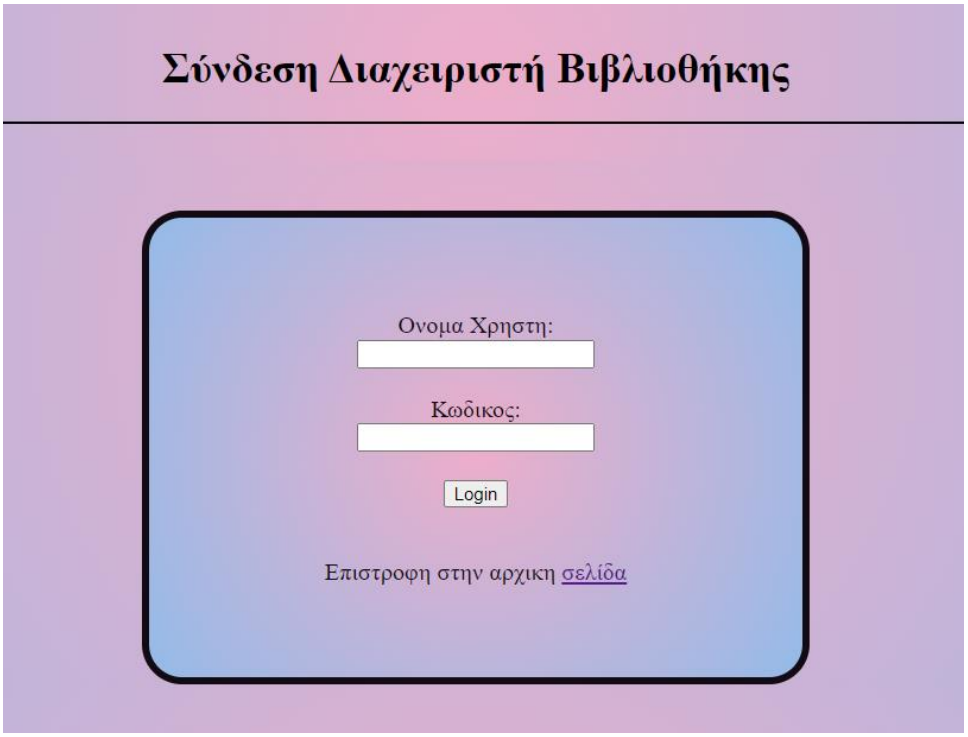
Υπεύθυνος Διαχειριστης:

Παρακάτω Φαίνεται η αρχική Σελίδα:

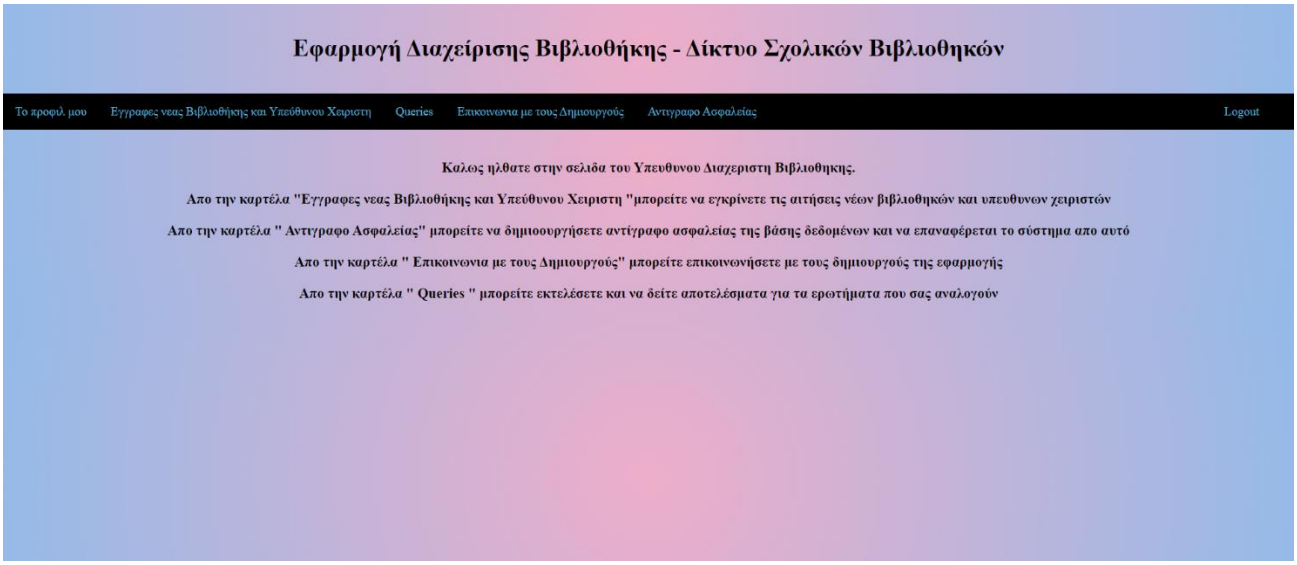


Στην ένδειξη ‘Επικοινωνία’ είναι διαθέσιμα τα στοιχεία των δημιουργών της εφαρμογής.

Ο υπεύθυνος διαχειριστής της βάσης μπορεί να συνδεθεί επιλέγοντας την αντίστοιχη ένδειξη (**‘Σύνδεση Υπεύθυνου Διαχειριστή’**).



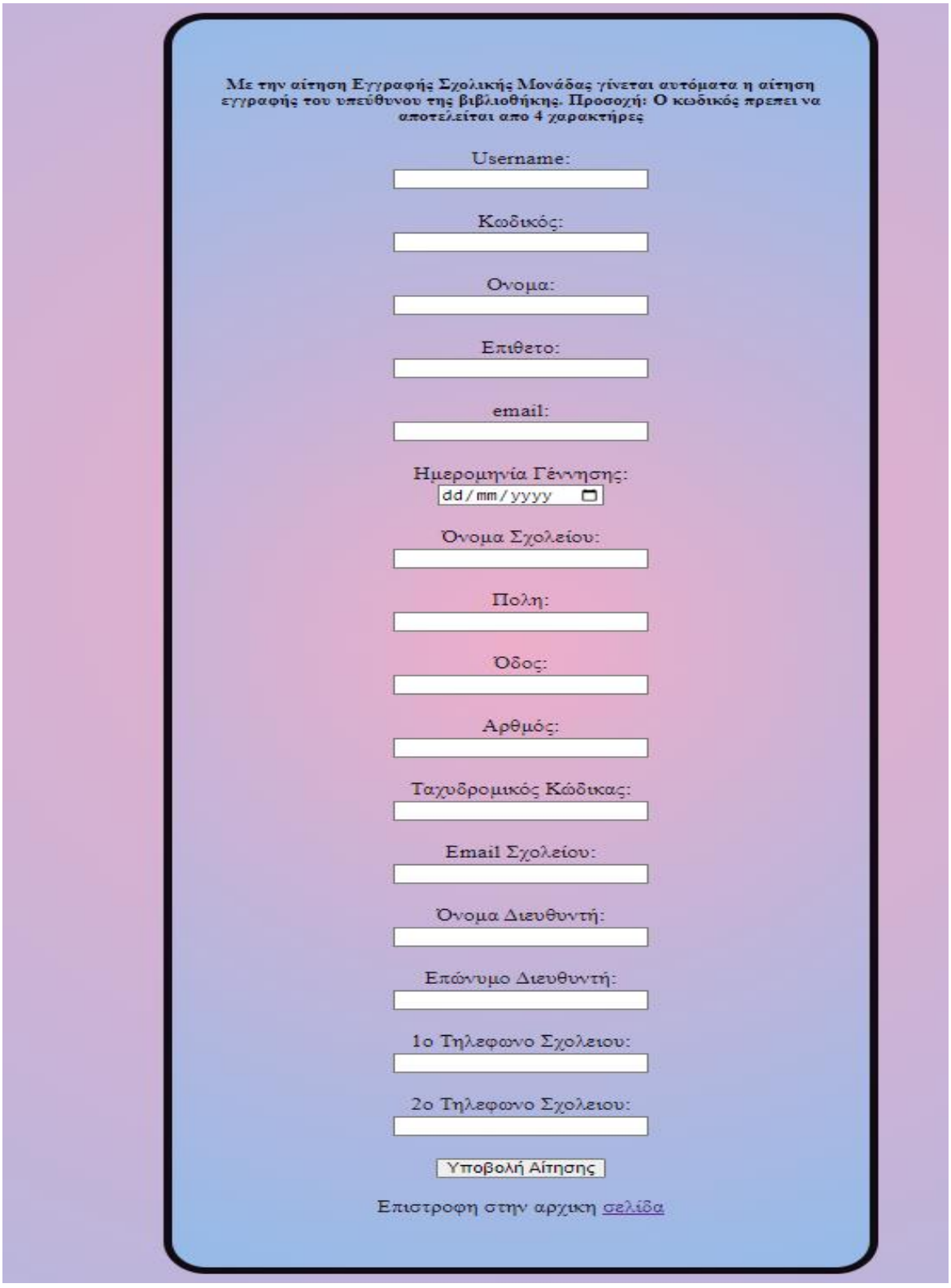
Αφού συνδεθεί, ανακατευθύνεται στην κεντρική σελίδα που του αντιστοιχεί:



Ο υπεύθυνος διαχειριστής μπορεί να δει και να επεξεργαστεί το προφίλ του **(Το προφίλ μου->Edit profile/Edit password)**, να εγκρίνει την εγγραφή μιας σχολικής βιβλιοθήκης και του υπεύθυνου χειριστή της αντίστοιχα **(->Εγγραφές νέας βιβλιοθήκης και Υπεύθυνου Χειριστή)** και να εκτελέσει κάποιο από τα Queries του (**Queries-> Query 1/2/3/4/5/6/7).** Επιπλέον, μπορεί να επικοινωνήσει με τους δημιουργούς της εφαρμογής **(-> επικοινωνία με τους δημιουργούς)**. Τέλος, είναι σε θέση να δημιουργήσει αντίγραφο ασφαλείας για όλη την βάση, και να επαναφέρει το σύστημα σε περίπτωση ατυχούς συμβάντος **(-> αντίγραφο ασφαλείας)**.

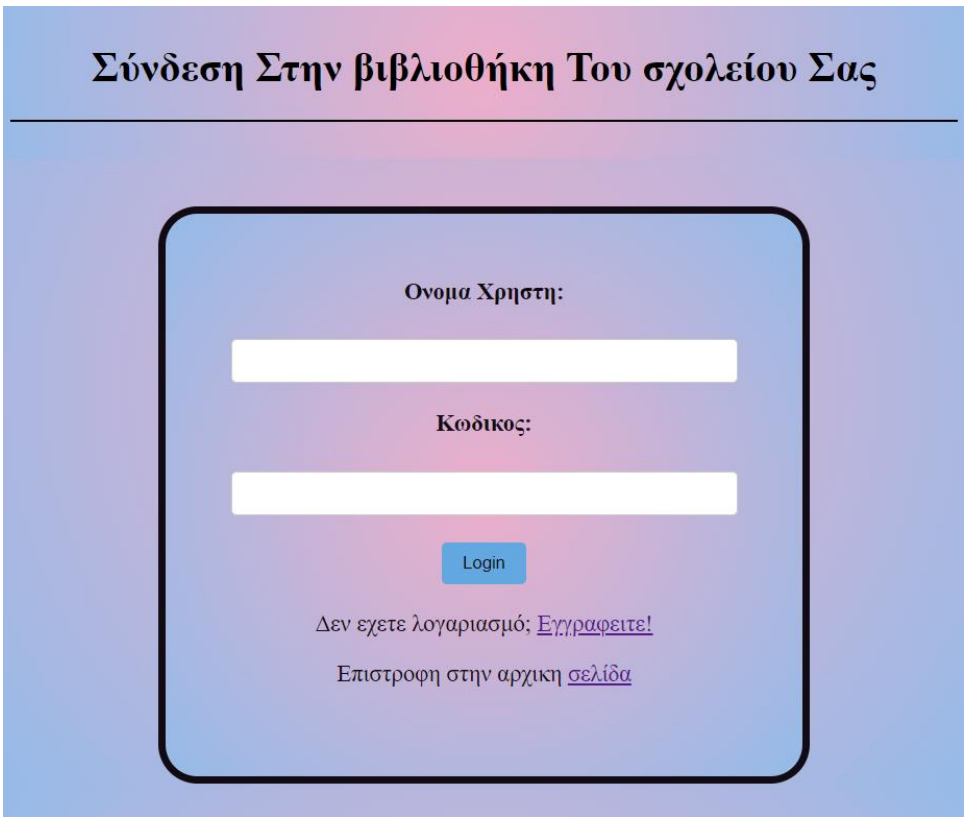
Υπεύθυνος Χειριστής σχολικής βιβλιοθήκης:

Ο επισκέπτης της αρχικής σελίδας έχει την δυνατότητα να κάνει αίτηση εγγραφής μιας σχολικής βιβλιοθήκης, κάνοντας παράλληλα αίτηση εγγραφής για τον εαυτό του ως υπεύθυνο χειριστή της (->Εγγραφή Σχολικής Βιβλιοθήκης). Κατόπιν οδηγείται στην παρακάτω φόρμα:



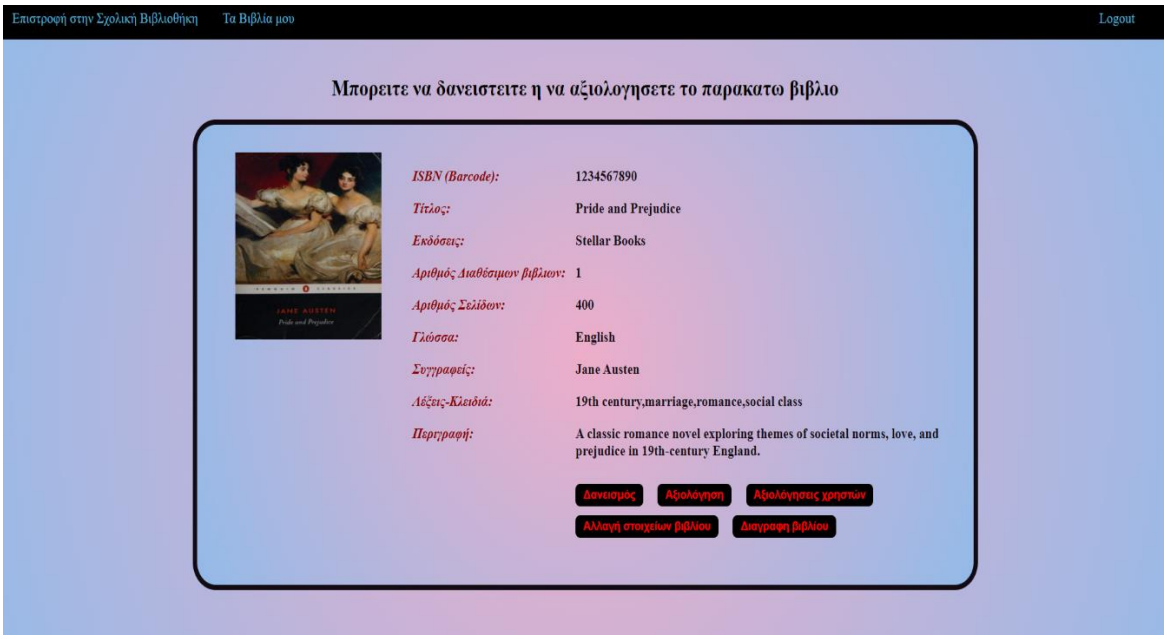
Με την ολοκλήρωση της αίτησης, εμφανίζεται κατάλληλο μήνυμα στον αιτούντα, για έγκριση του αιτήματος του από τον Υπεύθυνο Διαχειριστή.

Άπαξ και εγκριθεί το αίτημα του, συνδέεται από το περιβάλλον της αρχικής σελίδας, επιλέγοντας το σχολείο στο οποίο ανήκει. Αμέσως, γίνεται ανακατεύθυνση του χειριστή στην σελίδα σύνδεσης, όπου πληκτρολογώντας το αντίστοιχο username και password θα οδηγηθεί στο περιβάλλον της σχολικής βιβλιοθήκης στην οποία ανήκει.



Όπως φαίνεται και στην εικόνα, ο υπεύθυνος χειριστής μπορεί να επεξεργαστεί το προφίλ του, να επικοινωνήσει με τον Διαχειριστή Βιβλιοθήκης, να δει το ιστορικό δανεισμών του (MyBooks), τις αιτήσεις αξιολόγησης μαθητών (Αιτήσεις Αξιολόγησης), να προσθέσει βιβλίο (Προσθήκη βιβλίου), να εγκρίνει τις αιτήσεις για δανεισμούς (Κρατήσεις Βιβλίων) και τις αιτήσεις εγγραφής (Αιτήσεις Εγγραφής Χρηστών) που αναμένουν την έγκρισή του. Έχει, επίσης, τη δυνατότητα καταχώρησης δανεισμού (Νέος Δανεισμός) και επιστροφής (Επιστροφή Βιβλίου) βιβλίου εφόσον αυτά πραγματοποιούνται με φυσική παρουσία στο χώρο της βιβλιοθήκης. Τέλος, μπορεί να διαγράψει χρήστες (Διαγραφή χρηστών) και επιλέγοντας την ένδειξη Queries να εκτελέσει τα Queries που του αντιστοιχούν (Queries->query 1/2/3).

Επίσης, ο υπεύθυνος χειριστής έχει πρόσβαση στα βιβλία του σχολείου του και μπορεί να κάνει αναζήτηση κατά τίτλο/κατηγορία/συγγραφέα μέσω του search bar της αρχικής του σελίδας, και κατόπιν να επιλέξει βιβλίο. Για παράδειγμα έστω ότι επέλεξε το Pride and Prejudice:

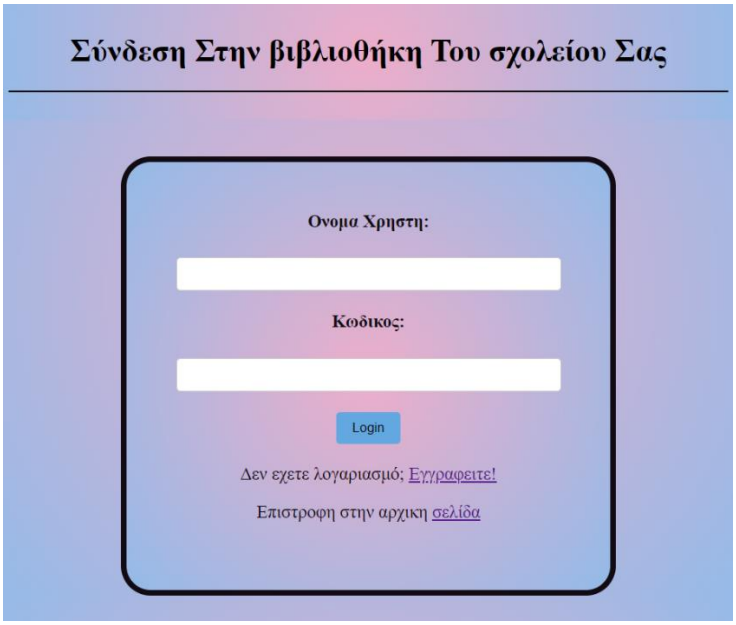


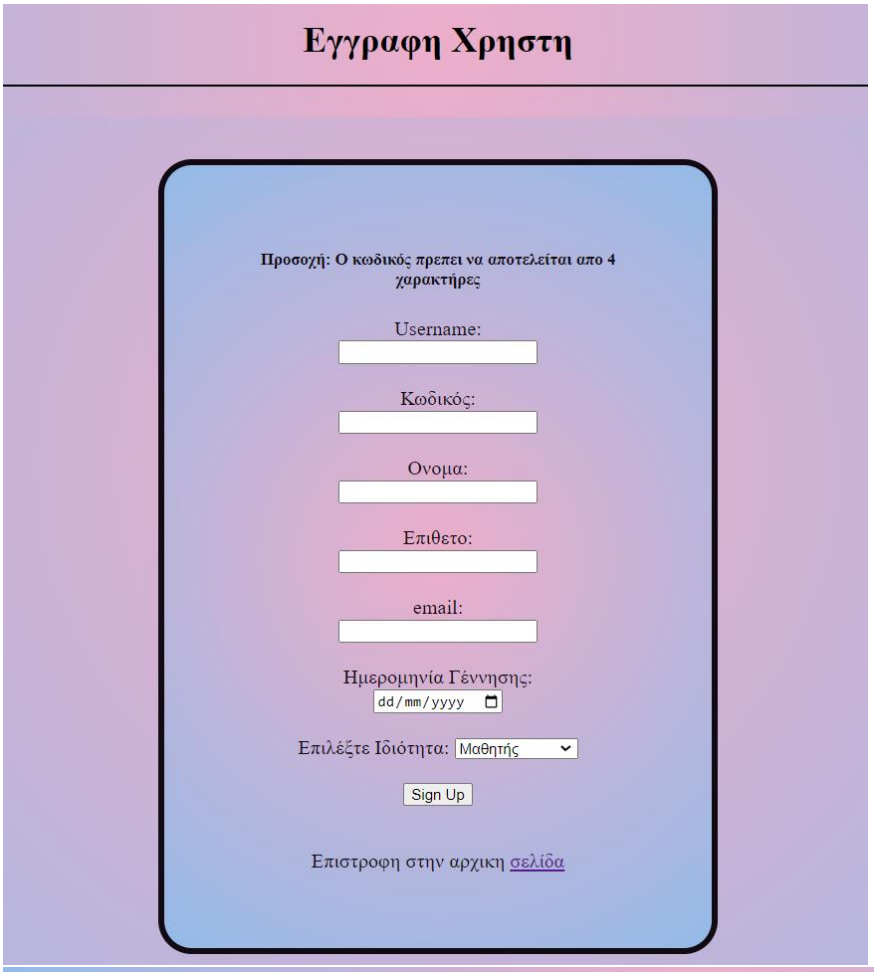
Όπως φαίνεται στην παραπάνω εικόνα, παρατίθενται αναλυτικά όλα τα στοιχεία που αφορούν το συγκεκριμένο βιβλίο, τα οποία μπορεί να επεξεργαστεί. Επιπλέον μπορεί και να διαγράψει το παρόν βιβλίο. Τέλος, έχει τη δυνατότητα να το δανειστεί, καθώς και να το αξιολογήσει αλλά και να δει αναλυτικά όλες τις αξιολογήσεις των άλλων χρηστών.

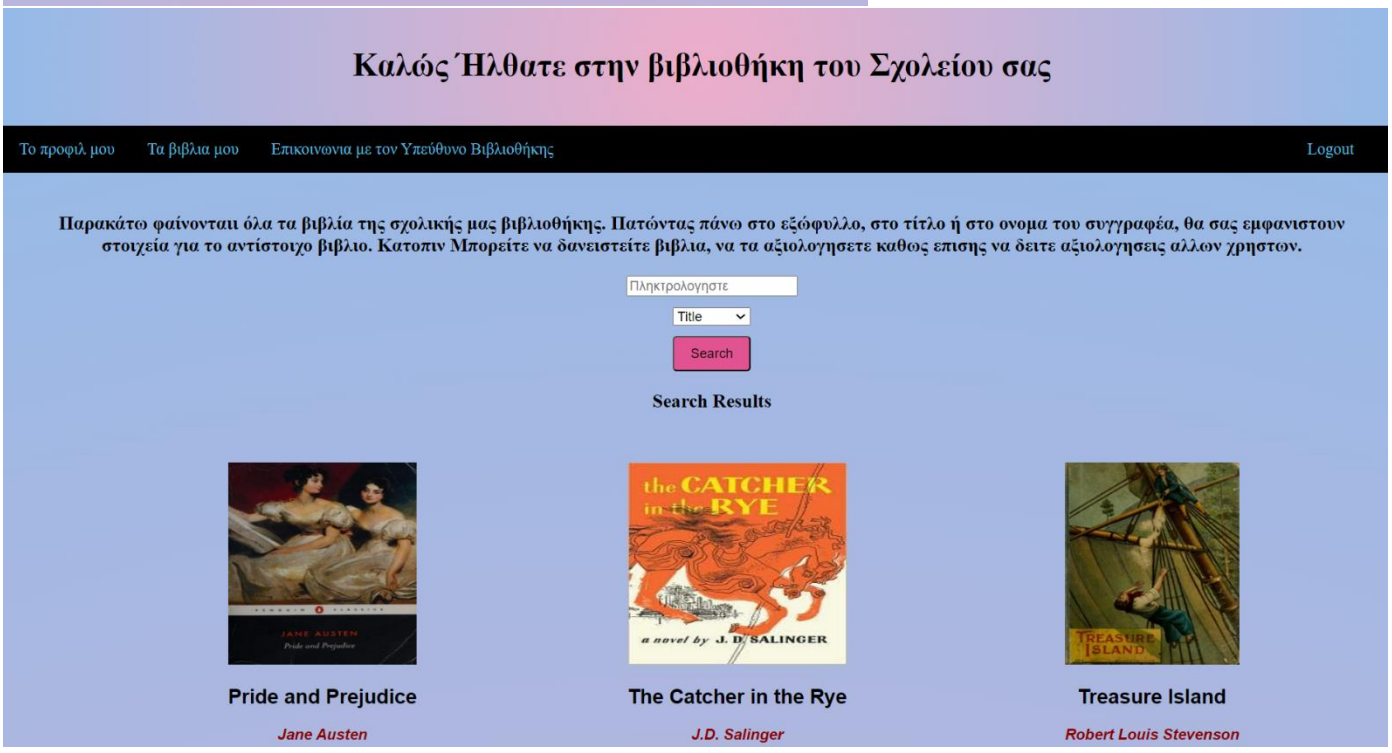
Αντίστοιχη με του υπεύθυνου χειριστή είναι και η αρχική σελίδα του απλού χρήστη, ο οποίος όμως μπορεί απλά να δει το ιστορικό δανεισμού του και τα στοιχεία επικοινωνίας του υπεύθυνου χειριστή. Επιπλέον, δεν μπορεί να παρέμβει σε κανένα στοιχείο του προφίλ του εκτός από το password.

Χρήστης:

Ο χρήστης (καθηγητής/μαθητής) μπορεί να επιλέξει το σχολείο στο οποίο ανήκει από τη λίστα που εμφανίζεται. Αμέσως, γίνεται ανακατεύθυνση του χρήστη στην σελίδα σύνδεσης, όπου πληκτρολογώντας το αντίστοιχο username και password θα οδηγηθεί στο περιβάλλον της σχολικής βιβλιοθήκης στην οποία ανήκει. Σε περίπτωση απλού χρήστη, ήτοι μαθητή ή καθηγητή, που δεν είναι εγγεγραμμένος, του δίνεται η δυνατότητα αίτησης εγγραφής, πατώντας στο αντίστοιχο λινκ **(“Εγγραφείτε”)**. Σε κάθε περίπτωση νέας εγγραφής είναι απαραίτητη η έγκριση του υπεύθυνου διαχειριστή (εμφάνιση μηνύματος αναμονής για έγκριση της αίτησης).







Όπως φαίνεται και στην παραπάνω εικόνα, η αρχική σελίδα του απλού χρήστη είναι αντίστοιχη με αυτή του υπεύθυνου χειριστή. Ο απλός χρήστης, μπορεί να επικοινωνήσει με τον υπεύθυνο χειριστή του σχολείου του (**Επικοινωνία με τον Υπεύθυνο Βιβλιοθήκης**). Πατώντας στα "Βιβλία μου" ανακατευθύνεται στην εξής σελίδα:

Εδώ μπορεί να δει όλα τα βιβλία που έχει δανειστεί στο παρελθόν, τρέχοντες δανεισμούς, δεσμεύσεις βιβλίων **(Reserved Items)** καθώς και τις κρατήσεις του για βιβλία που δεν έχουν διαθέσιμα αντίτυπα. Όπως φαίνεται, μπορεί να διαγράψει κάποιο βιβλίο από τον πίνακα των reserved items **(delete)**, αποδεσμεύοντάς το, καθώς και να αφαιρέσει κάποιο βιβλίο από τις κρατήσεις του, σε περίπτωση που εν τέλει δεν επιθυμεί να το λάβει όταν γίνει διαθέσιμο.

Τέλος, ο χρήστης, επιστρέφοντας στην αρχική έχει πρόσβαση στα βιβλία του σχολείου του και μπορεί να κάνει αναζήτηση κατά τίτλο/κατηγορία/συγγραφέα μέσω του search bar της αρχικής του σελίδας, και κατόπιν να επιλέξει βιβλίο. Για παράδειγμα έστω ότι επέλεξε το The Catcher In the Rye:

Κατόπιν, μπορεί να το αιτηθεί τον δανεισμό του, να το αξιολογήσει σε περίπτωση που το έχει δανειστεί, καθώς και να διαβάσει τις αξιολογήσεις άλλων χρηστών.

Κανονισμός Βιβλιοθήκης

- Οι μαθητές μπορούν να δανειστούν έως δύο βιβλία την εβδομάδα, καθώς και να κάνουν κράτηση για δύο μη διαθέσιμα βιβλία την εβδομάδα.
- Οι καθηγητές (και ο υπεύθυνος χειριστής, καθώς είναι και αυτός καθηγητής) μπορούν να δανειστούν ένα βιβλίο την εβδομάδα, καθώς και να κάνουν κράτηση για ένα μη διαθέσιμο βιβλίο την εβδομάδα.
- Ένας χρήστης μπορεί να δεσμεύσει ηλεκτρονικά ένα διαθέσιμο βιβλίο (το βιβλίο προστίθεται στον πίνακα 'Reserved Items') και να το δανειστεί επισήμως παραλαμβάνοντάς το από τη βιβλιοθήκη (το βιβλίο μεταφέρεται από τον πίνακα Reserved Items στον πίνακα 'Borrowed').
- Ένας δανεισμός μπορεί να γίνει άμεσα(χωρίς να έχει προηγηθεί δέσμευση) με φυσική παρουσία στην βιβλιοθήκη (το βιβλίο προστίθεται απευθείας στον πίνακα 'Borrowed').
- Σε περίπτωση που ένα βιβλίο δεν είναι διαθέσιμο και ένας χρήστης κάνει κράτηση για αυτό τότε μπαίνει σε σειρά αναμονής (queue).
- Άπαξ και προκύψει κάποιο διαθέσιμο αντίτυπο (όταν γίνει κάποιο return) τότε ο χρήστης με το παλαιότερο αίτημα κράτησης μπορεί πλέον να δανειστεί το βιβλίο (μετάβαση από κατάσταση queue σε κατάσταση reserved, σε επίπεδο εφαρμογής: μετάβαση από τον πίνακα 'Κρατήσεις βιβλίων που δεν είναι διαθέσιμα' στον πίνακα 'Reserved Items'). Όταν ο χρήστης παραλάβει το βιβλίο δια ζώσης από τη βιβλιοθήκη (δανεισμός) τότε αυτό μεταφέρεται σε επίπεδο εφαρμογής από τον πίνακα 'Reserved Items' στον πίνακα 'Borrowed'.
- Σε περίπτωση που ο χρήστης με το παλαιότερο αίτημα κράτησης έχει φτάσει το ανώτατο όριο δανεισμών αυτή την εβδομάδα, τότε παραβλέπεται και εξυπηρετείται, αν υπάρχει, ο αμέσως επόμενος χρήστης στην ουρά αναμονής σύμφωνα με τη διαδικασία που περιγράφεται στο προηγούμενο bullet. Ο χρήστης που παραλείφθηκε παραμένει στην σειρά αναμονής. Αν εξ αρχής δεν υπήρχε άλλος χρήστης στην ουρά αναμονής, το βιβλίο δεν δεσμεύεται από κανέναν και καθίσταται διαθέσιμο.
- Όλοι οι δανεισμοί θεωρούνται εκπρόθεσμοι μετά το πέρας μιας εβδομάδας και ο χρήστης δεν δικαιούται κανένα δανεισμό και καμία κράτηση μέχρι την επιστροφή του αντίστοιχου βιβλίου.
- Όλες οι δεσμεύσεις διαθέσιμου αντιτύπου (Reserved Items) διαγράφονται μετά από μία εβδομάδα.
- Όλες οι κρατήσεις μη διαθέσιμου βιβλίου (Κρατήσεις βιβλίων που δεν είναι διαθέσιμα) διαγράφονται μετά από μία εβδομάδα, δηλαδή ο χρήστης διαγράφεται από τη σειρά αναμονής του βιβλίου αυτού.
- Ο υπεύθυνος χειριστής έχει το δικαίωμα να διαγράψει κάποιον χρήστη που ανήκει στο σχολείο του εφόσον το κρίνει απαραίτητο.

Οδηγίες Εγκατάστασης - Requirements

Για την εγκατάσταση της βάσης χρειάζεται:

1. Εγκατάσταση DBMS

Απαιτείται η εγκατάσταση ενός DBMS, εμείς χρησιμοποιήσαμε MySQL

2. Εγκατάσταση SQL Server

Απαιτείται η εγκατάσταση ενός SQL Server
Εμείς χρησιμοποιήσαμε:
Linux: LAMPP
Windows: XAMPP

3. Εγκατάσταση Python – Pip

Πρέπει να εγκαταστήσετε την Python (την νεότερη κατά προτίμηση έκδοση).
Προσοχή: πρέπει κατά την εγκατάσταση να εγκαταστήσετε και το pip.

4. Κατέβασμα repository από Github

Το repository της βάσης δεδομένων βρίσκεται στο github:
https://github.com/georgegeo248/Project_Databases_Team_44
Το κατέβασμα των αρχείων μπορεί να γίνει είτε μέσω της ιστοσελίδας του Github είτε τρέχοντας την ακόλουθη εντολή σε ένα terminal το directory που θέλετε να αποθηκεύσετε την εφαρμογή:

```
git clone https://github.com/georgegeo248/Project_Databases_Team_44
```

5. Εγκατάσταση απαραίτητων βιβλιοθηκών

Πρέπει να εγκαταστήσετε όλες τις απαραίτητες βιβλιοθήκες που αναφέρονται στο αρχείο requirements.txt μέσω της εντολής `pip install -r requirements.txt` στο terminal.

Δημιουργία της Βάσης και Εκτέλεση της εφαρμογής

1. Δημιουργία Βάσης Δεδομένων

Πρέπει να φορτώσετε τα αρχεία **library_schema.sql** και **insert_data.sql** μέσω σύνδεσης με mySQL server. Τα αρχεία αυτά βρίσκονται στον φάκελο sql μέσα στο repository που κατεβάσατε.
Αυτό μπορείτε να το κάνετε αυτόματα εκτελώντας ένα script το οποίο βρίσκεται μέσα repository.
Εφόσον έχετε κατεβάσει το repository στο directory που επιθυμείτε, μέσω του terminal πηγαίντε μέσα σε αυτό το directory. Έπειτα:
Linux:
Τρέξτε τις εντολές:

```
chmod +x create_library_linux.sh
./create_library_linux.sh
```

Windows:
Τρέξτε την εντολή:

```
.\create_library_windows.bat
```

Σε περίπτωση σφάλματος βεβαιωθείτε ότι στο directory που βρίσκεστε βρίσκεται και το αρχείο που θέλετε να τρέξετε.
Σε οποιαδήποτε περίπτωση μπορείτε πάντα να φορτώσετε τα αρχεία **library_schema.sql**

και **insert_data.sql** χειροκίνητα μέσω του DBMS.

2. Εκτέλεση της Εφαρμογής

Τρέξτε το αρχείο run.py μέσω της εντολής python run.py ή python3 run.py
Τέλος ανοίξτε έναν browser και μπείτε στην διεύθυνση localhost:3000 (ή αλλιώς http://127.0.0.1:3000)