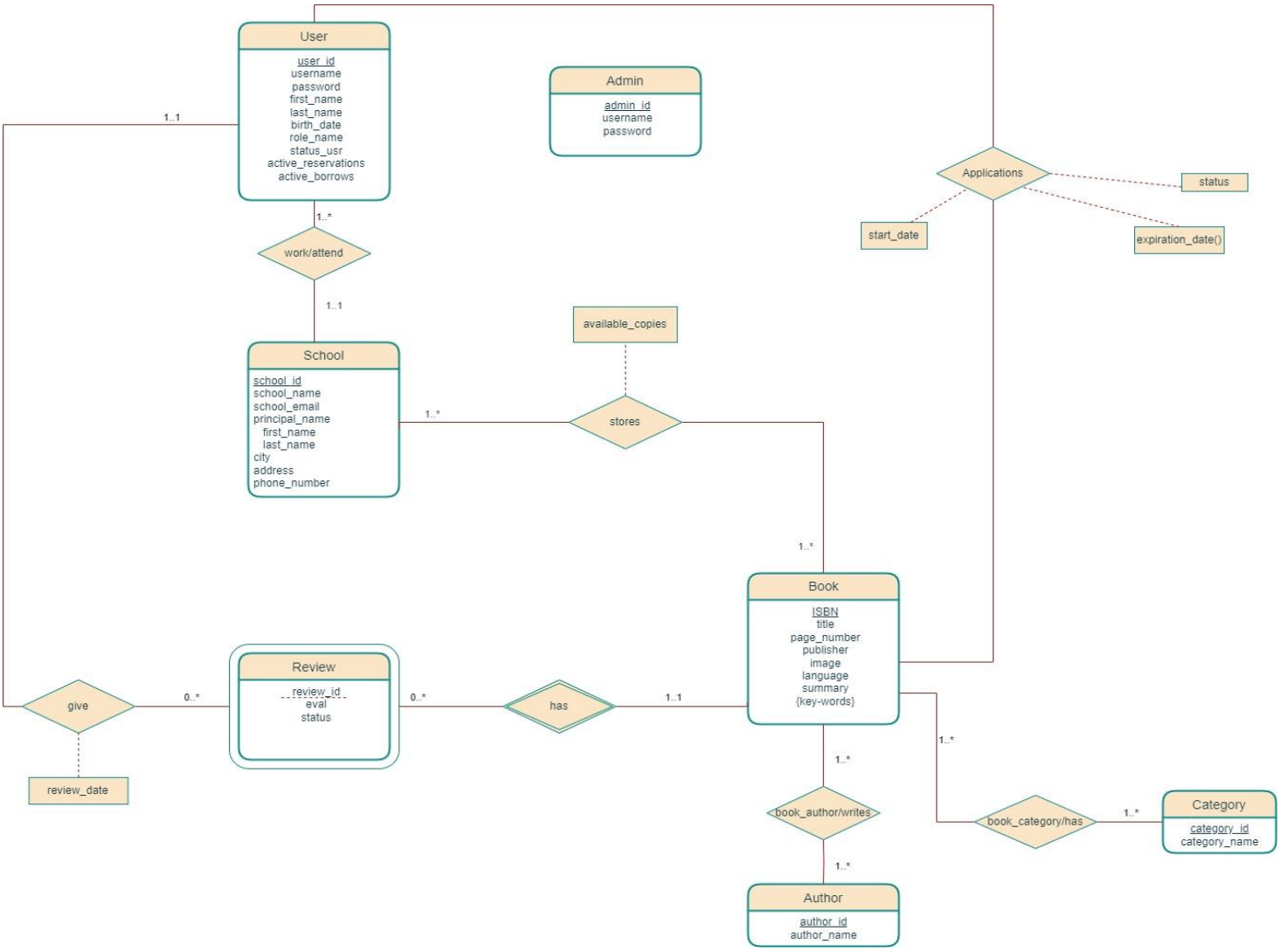
Βάσεις Δεδομένων
Εξαμηνιαία Εργασία

ΣΧΟΛΙΚΕΣ ΒΙΒΛΙΟΘΗΚΕΣ

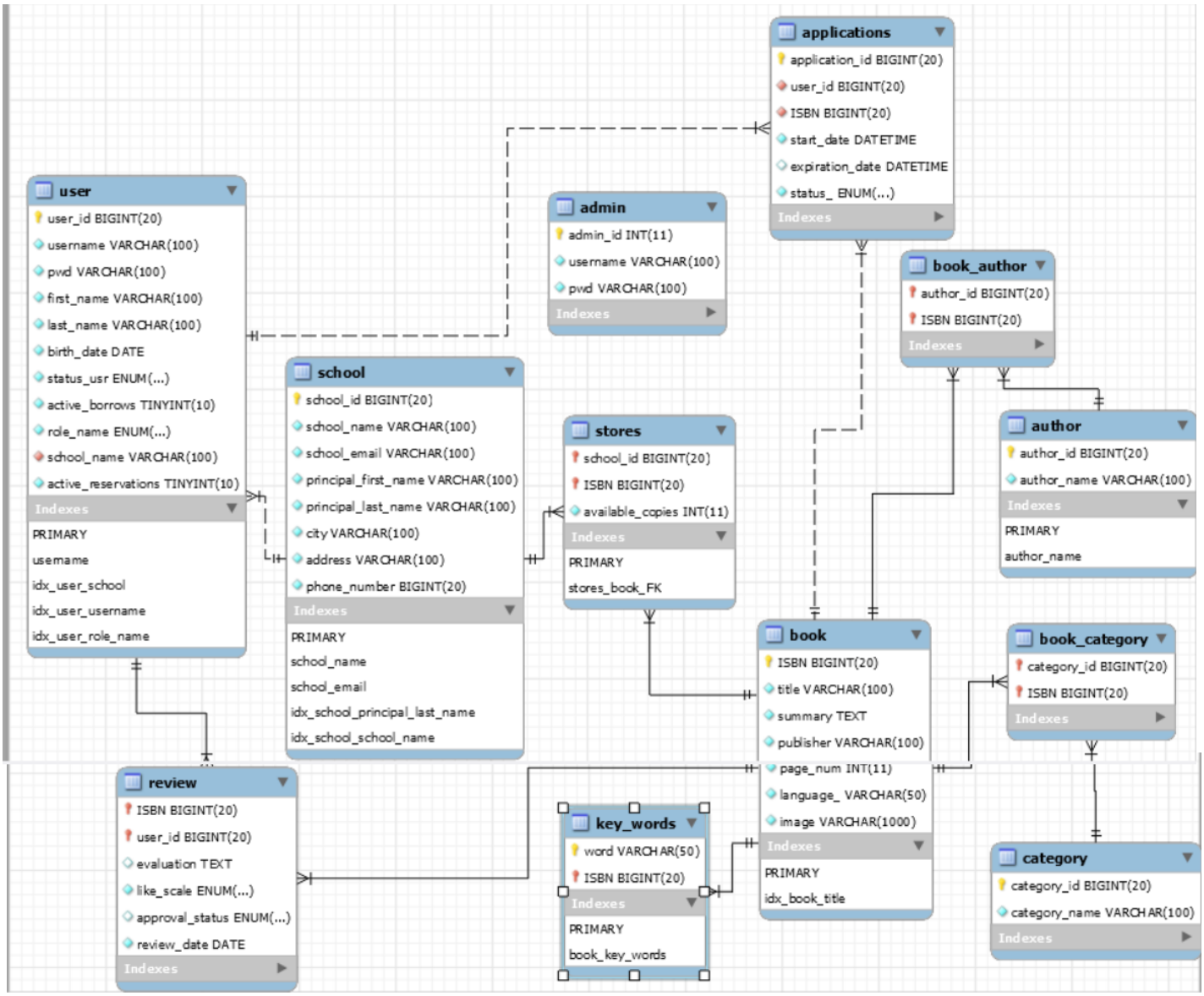
Ομάδα Project 119

Ονοματεπώνυμο	Αριθμός Μητρώου
Αδαμόπουλος Διονύσης	el20061
Καμπουγέρης Χαράλαμπος	el20098
Κουστένης Χρίστος	el20227

1.1
ER δίαγραμμα /Διάγραμμα Οντοτήτων-Συσχετίσεων



Σχεσιακό Μοντέλο/Relational Model



Ο πίνακας admin είναι ανεξάρτητος από όλους του άλλους αφού ο admin αφού δεν αποτελεί άμεσος χρήστης των βιβλιοθηκών.

Πέρα από αυτήν την οντότητα στην βάση μας περιέχονται 5 κύριες οντότητες : School, Book, User, Author, Category και ένα αδύναμο σύνολο οντοτήτων Review που εξαρτάται άμεσα από το Book. Οι υπόλοιποι πίνακες του σχεσιακού αποτελούν σύνολα συσχετίσεων με τα κατάλληλα foreign keys αναλόγως με τις πληθικότητες συμμετοχής των βασικών συνόλων οντοτήτων.

Ο πίνακας key_words αποτελεί ένα attribute πολλαπλών τιμών του πίνακα Book.

DDL

```
/* Create the Database */

CREATE DATABASE db2023;

USE db2023;

SET GLOBAL event_scheduler = ON;

CREATE TABLE IF NOT EXISTS admin
(
    admin_id INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    username VARCHAR(100) DEFAULT 'admin'   NOT NULL,
    pwd      VARCHAR(100)                   NOT NULL
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS school
(
    school_id            BIGINT AUTO_INCREMENT NOT NULL PRIMARY KEY,
    school_name          VARCHAR(100)          NOT NULL UNIQUE,
    school_email         VARCHAR(100)          NOT NULL UNIQUE,
    principal_first_name VARCHAR(100)          NOT NULL,
    principal_last_name  VARCHAR(100)          NOT NULL,
    city                 VARCHAR(100)          NOT NULL,
    address              VARCHAR(100)          NOT NULL,
    phone_number         BIGINT                NOT NULL
) ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS user
(
    user_id BIGINT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    username VARCHAR(100) UNIQUE NOT NULL,
    pwd VARCHAR(100) NOT NULL
        CHECK (CHAR_LENGTH(pwd) > 3),
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    birth_date DATE NOT NULL,
    status_usr ENUM ('active','pending','removed') DEFAULT 'pending' NOT NULL,
    active_borrows TINYINT(10) DEFAULT 0 NOT NULL
        CHECK ((active_borrows < 3 AND active_borrows >= 0 AND role_name = 'student')
        OR(active_borrows < 2 AND active_borrows >=0 AND role_name = 'teacher')
        OR (active_borrows =0 AND role_name = 'handler')),
    role_name ENUM ('student', 'teacher', 'handler') NOT NULL,
    school_name VARCHAR(100) NOT NULL,
    active_reservations TINYINT(10) DEFAULT 0 NOT NULL
        CHECK ((active_reservations < 3 AND active_reservations >= 0 AND role_name =
'student')
        OR(active_reservations < 2 AND active_reservations >=0 AND role_name = 'teacher')
        OR (active_reservations =0 AND role_name = 'handler')),
    CONSTRAINT FK_school_name FOREIGN KEY (school_name)
        REFERENCES school (school_name)
        ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE = InnoDB;



CREATE TABLE IF NOT EXISTS book
```

```
(
    ISBN      BIGINT         NOT NULL PRIMARY KEY,
    title     VARCHAR(100)   NOT NULL,
    summary   TEXT           NOT NULL,
    publisher VARCHAR(100)   NOT NULL,
    page_num  INT            NOT NULL CHECK (page_num > 0),
    language_ VARCHAR(50)    NOT NULL,
    image     VARCHAR(1000) NOT NULL
) ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS category
(
    category_id   BIGINT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    category_name VARCHAR(100) UNIQUE               NOT NULL
) ENGINE = InnoDB;

CREATE TABLE book_category
(
    category_id BIGINT NOT NULL,
    ISBN        BIGINT NOT NULL,
    CONSTRAINT FK_category FOREIGN KEY (category_id)
        REFERENCES category (category_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT FK_book FOREIGN KEY (ISBN)
        REFERENCES book (ISBN)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT compound_PK_category_book PRIMARY KEY (ISBN, category_id)
) ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS author
(
    author_id   BIGINT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    author_name VARCHAR(100) UNIQUE               NOT NULL
) ENGINE = InnoDB;

CREATE TABLE book_author
(
    author_id BIGINT NOT NULL,
    ISBN      BIGINT NOT NULL,
    CONSTRAINT FK_author_writes FOREIGN KEY (author_id)
        REFERENCES author (author_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT FK_book_writes FOREIGN KEY (ISBN)
        REFERENCES book (ISBN)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT compound_PK_writes PRIMARY KEY (ISBN, author_id)
)ENGINE = InnoDB;
CREATE TABLE IF NOT EXISTS key_words
(
    word VARCHAR(50) NOT NULL,
    ISBN BIGINT      NOT NULL,
    CONSTRAINT book_key_words FOREIGN KEY (ISBN)
        REFERENCES book (ISBN)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT compound_PK_key_words PRIMARY KEY (word, ISBN)
) ENGINE = InnoDB;
```

```sql
CREATE TABLE IF NOT EXISTS stores
(
    school_id        BIGINT NOT NULL,
    ISBN             BIGINT NOT NULL,
    available_copies INT    NOT NULL CHECK (available_copies >= 0),
    CONSTRAINT school_stores_FK FOREIGN KEY (school_id)
        REFERENCES school (school_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT stores_book_FK FOREIGN KEY (ISBN)
        REFERENCES book (ISBN)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT compound_PK_stores PRIMARY KEY (school_id, ISBN)
) ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS applications
(
    application_id  BIGINT AUTO_INCREMENT
NOT NULL PRIMARY KEY,
    user_id         BIGINT
NOT NULL,
    ISBN            BIGINT
NOT NULL,
    start_date      DATETIME
NOT NULL,
    expiration_date DATETIME CHECK (expiration_date >= start_date),
    status_         ENUM ('applied','borrowed','expired_borrowing','completed','queued')
DEFAULT ('queued') NOT NULL,
    CONSTRAINT FK_application_user FOREIGN KEY (user_id)
        REFERENCES user (user_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT FK_application_ISBN FOREIGN KEY (ISBN)
        REFERENCES book (ISBN)
        ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE = InnoDB;

CREATE TABLE review
(
    ISBN            BIGINT                          NOT NULL,
    user_id         BIGINT                          NOT NULL,
    evaluation      TEXT,
    like_scale      ENUM ('1', '2', '3', '4', '5') NOT NULL,
    approval_status ENUM ('approved','pending') DEFAULT 'pending',
    review_date     DATE                            NOT NULL,
    CONSTRAINT FK_book_review FOREIGN KEY (ISBN)
        REFERENCES book (ISBN)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT FK_user_review FOREIGN KEY (user_id)
        REFERENCES user (user_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT compound_PK_review PRIMARY KEY (user_id, ISBN)
) ENGINE = InnoDB;

/* If a teacher uploads review then automatically make it active */
CREATE TRIGGER review_upload
    BEFORE INSERT
    ON review
    FOR EACH ROW
BEGIN
```

```
    IF NEW.approval_status = 'pending' AND (SELECT user.role_name
                                            FROM user
                                            WHERE user.user_id = NEW.user_id) = 'teacher'
    THEN
        SET NEW.approval_status = 'approved';
    END IF;
END;


CREATE TRIGGER trigger_update_active_borrows
    BEFORE UPDATE
    ON applications
    FOR EACH ROW
BEGIN

    IF NEW.status_ = 'borrowed' AND OLD.status_ = 'applied' THEN
          SET NEW.start_date = NOW();
        SET NEW.expiration_date = DATE_ADD(NOW(), INTERVAL 1 WEEK);
        UPDATE user
        SET active_borrows      = active_borrows + 1,
            active_reservations = active_reservations - 1
        WHERE user.user_id = NEW.user_id;



    ELSEIF NEW.status_ = 'completed' AND OLD.status_ = 'borrowed' THEN
        UPDATE user
        SET active_borrows = active_borrows - 1
        WHERE user_id = NEW.user_id;

        UPDATE stores
        SET available_copies = available_copies + 1
        WHERE stores.ISBN = NEW.ISBN
          AND school_id = (SELECT s.school_id
                           FROM (SELECT user.school_name FROM user WHERE user_id =
NEW.user_id) u
                           INNER JOIN school s ON s.school_name = u.school_name);


    ELSEIF NEW.status_ = 'completed' AND OLD.status_ = 'expired_borrowing' THEN
        UPDATE user
        SET active_borrows = active_borrows - 1
        WHERE user.user_id = NEW.user_id;
        UPDATE stores
        SET available_copies = available_copies + 1
        WHERE stores.ISBN = NEW.ISBN
          AND stores.school_id = (SELECT s.school_id
                                  FROM (SELECT user.school_name FROM user WHERE
user.user_id = NEW.user_id) u
                                           INNER JOIN school s ON s.school_name =
u.school_name);

    END IF;

END;

CREATE TRIGGER trigger_update_dates_available_copies_on_applying
    BEFORE INSERT
    ON applications
```

```
    FOR EACH ROW
BEGIN
    IF (SELECT available_copies
        FROM stores
        WHERE ISBN = NEW.ISBN
          AND school_id = (SELECT s.school_id
                           FROM (SELECT user.school_name FROM user WHERE user.user_id =
NEW.user_id) u
                                    INNER JOIN school s ON s.school_name =
u.school_name)) > 0 THEN
        SET NEW.status_ = 'applied';
        SET NEW.start_date = NOW();
        SET NEW.expiration_date = DATE_ADD(NOW(), INTERVAL 3 MINUTE);
    END IF;

    IF NEW.status_ = 'applied' THEN
        SET NEW.start_date = NOW();
        SET NEW.expiration_date = DATE_ADD(NOW(), INTERVAL 3 MINUTE);
        UPDATE stores
        SET stores.available_copies = stores.available_copies - 1
        WHERE stores.ISBN = NEW.ISBN
          AND stores.school_id = (SELECT s.school_id
                                  FROM (SELECT user.school_name FROM user WHERE
user.user_id = NEW.user_id) u
                                           INNER JOIN school s ON s.school_name =
u.school_name);
        UPDATE user
        SET active_reservations = active_reservations + 1
        WHERE user_id = NEW.user_id;
    END IF;

    IF NEW.status_ = 'queued' THEN
        UPDATE user
        SET active_reservations = active_reservations + 1 WHERE user_id = NEW.user_id;
        SET NEW.start_date = NOW();
        SET NEW.expiration_date = DATE_ADD(NOW(), INTERVAL 3 MINUTE);
    END IF;

    IF (NEW.status_ != 'applied' AND NEW.status_ != 'queued') THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You have to apply first';
    END IF;

    IF (SELECT u.role_name FROM user u WHERE u.user_id = NEW.user_id) = 'teacher' AND
       (SELECT u.active_reservations FROM user u WHERE u.user_id = NEW.user_id) = 2 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Teacher can only borrow 1 book';
    END IF;

    IF (SELECT COUNT(*)
        FROM applications
        WHERE status_ != 'completed'
          AND NEW.user_id = user_id
          AND NEW.ISBN = ISBN) > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You have already applied/borrowed
this book';
    END IF;

    IF (SELECT COUNT(*)
        FROM applications
        WHERE status_ = 'expired_borrowing'
```

```
          AND NEW.user_id = user_id
          AND NEW.ISBN = ISBN) > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You cannot apply for borrow when u
have an expired borrowing';
    END IF;

    IF (SELECT u.role_name FROM user u WHERE NEW.user_id = u.user_id) = 'handler' THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Handlers cannot borrow books!';
    END IF;
END;


CREATE TRIGGER trigger_update_active_reservations_on_delete
    AFTER DELETE
    ON applications
    FOR EACH ROW
BEGIN
    IF OLD.status_ = 'applied' OR OLD.status_ = 'queued' THEN
        UPDATE user
        SET active_reservations = active_reservations - 1
        WHERE user.user_id = OLD.user_id;
    END IF;
    IF OLD.status_ = 'applied' THEN
        UPDATE stores
        SET available_copies = available_copies + 1
        WHERE ISBN = OLD.ISBN
          AND school_id = (SELECT school_id FROM school WHERE school_name = (SELECT
school_name FROM user WHERE user_id = OLD.user_id));
    END IF;
END;


CREATE EVENT check_not_returned_books
    ON SCHEDULE
        EVERY 30 SECOND
            STARTS NOW()
    DO
    UPDATE applications
    SET status_ = 'expired_borrowing'
    WHERE expiration_date < NOW()
      AND status_ = 'borrowed';

CREATE EVENT event_check_book_availability
    ON SCHEDULE
        EVERY 30 SECOND
            STARTS NOW()
    DO
    BEGIN
        UPDATE applications a
            INNER JOIN stores s ON a.ISBN = s.ISBN
        SET a.status_ = 'applied',s.available_copies = s.available_copies - 1
        WHERE a.status_ = 'queued'
          AND s.available_copies > 0
         ORDER BY a.start_date ASC
        LIMIT 1;
    END;

CREATE TRIGGER expired_borrow_penalty
    BEFORE INSERT
```

```
    ON applications
    FOR EACH ROW
BEGIN
    IF (SELECT COUNT(*)
        FROM applications
        WHERE status_ = 'expired_borrowing'
          AND NEW.user_id = user_id) > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You cannot apply for borrow when u
have an expired borrowing';
    END IF;
END;



CREATE TRIGGER duplicate_apply_borrow
    BEFORE INSERT
    ON applications
    FOR EACH ROW
BEGIN
    IF (SELECT COUNT(*)
        FROM applications
        WHERE status_ != 'completed'
          AND NEW.user_id = user_id
          AND NEW.ISBN = ISBN) > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You have already applied/borrowed
this book';
    END IF;
END;

CREATE TRIGGER trigger_check_handler_school
BEFORE INSERT ON user
FOR EACH ROW
BEGIN
    IF (SELECT COUNT(*) FROM user WHERE school_name = NEW.school_name AND NEW.role_name =
'handler') > 0  THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Only one handler can be assigned to
this school';
    END IF;
END;

CREATE INDEX idx_user_school ON user(school_name);/*FK*/
CREATE INDEX idx_user_username ON user(username);
CREATE INDEX idx_user_role_name ON user(role_name);

CREATE INDEX idx_applications_ISBN ON applications(ISBN);/*FK*/
CREATE INDEX idx_applications_user ON applications(user_id);/*FK*/
CREATE INDEX idx_applications_status_ ON applications(status_);

CREATE INDEX idx_school_principal_last_name ON school(principal_last_name);
CREATE INDEX idx_school_school_name ON school(school_name);

CREATE INDEX idx_book_title ON book(title);

CREATE INDEX idx_review_like_scale ON review(like_scale);
```

Το παραπάνω αρχείο αποτελεί συγχώνευση των myschema.sql, triggers_and_events.sql, indexes.sql

DML

Τα DML αρχεία για το populate του database βρίσκονται στα αρχεία book_insertions.sql, school_insertions.sql, stores.sql, user_insert.sql, key_words.sql, Authors&Categories.sql, applications_insert.sql

Constraint checks

Για τον πίνακα user:

Κωδικός πρέπει να έχει μέγεθος τουλάχιστον 4 χαρακτήρες

Active Reservations: Βοηθητικός μετρητής attribute για γρήγορους ελέγχους που κοιτάμε πόσες ενεργές κρατήσεις έχει ένας χρήστης. Γίνεται έλεγχος για τις εξής 3 περιπτώσεις, ανάλογα με το attribute role_name του user, είτε το constraint 0<=active reservations<3 για role_name=student, είτε το constraint 0<=active reservations<2 για role_name=teacher και active reservations=0 για role_name=handler (θεωρήσαμε ότι ο χειριστής δεν μπορεί να κάνει κράτηση/δανεισμό για τον ίδιο)

Active Borrows: Αντίστοιχος μετρητής attribute για τους ενεργούς δανεισμούς, όπου ισχύουν ίδια constraints.

Για τον πίνακα book:

Αριθμός Σελίδων, attribute, όπου έχει constraint να είναι θετικός αριθμός.

Για τον πίνακα stores:

Πίνακας που σχετίζει βιβλίο και σχολείο που «αποθηκεύει» το βιβλίο, έχουμε το attribute available copies που είναι πάντα μη αρνητικός ακέραιος.

Για τον πίνακα applications:

Attribute expiration_date πρέπει να είναι μεγαλύτερο ή ίσο του start_date

Queries

3.1. (Διαχειριστής)

3.1.1. Παρουσίαση λίστας με συνολικό αριθμό δανεισμών ανά σχολείο (Κριτήρια αναζήτησης: έτος, ημερολογιακός μήνας πχ Ιανουάριος):

```
SELECT school.school_name, COUNT(a.application_id) AS total_borrows FROM school
LEFT JOIN user ON school.school_name = user.school_name
LEFT JOIN applications a ON user.user_id = a.user_id AND a.status_!='applied' AND
MONTH(a.start_date) = '{dt.month}' AND YEAR(a.start_date) = "{dt.year}"
GROUP BY school.school_name
```

3.1.2. Για δεδομένη κατηγορία βιβλίων (επιλέγει ο χρήστης), ποιοι συγγραφείς ανήκουν σε αυτήν και ποιοι εκπαιδευτικοί έχουν δανειστεί βιβλία αυτής της κατηγορίας το τελευταίο έτος;

```
SELECT DISTINCT user.first_name, user.last_name, user.school_name
FROM user
INNER JOIN applications ON user.user_id = applications.user_id
INNER JOIN book ON applications.ISBN = book.ISBN
INNER JOIN book_category ON book.ISBN = book_category.ISBN
INNER JOIN category ON book_category.category_id = category.category_id
WHERE category.category_name = '{category}'
AND applications.start_date >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
AND user.role_name = 'teacher
```

3.1.3. Βρείτε τους νέους εκπαιδευτικούς (ηλικία < 40 ετών) που έχουν δανειστεί τα περισσότερα βιβλία και των αριθμό των βιβλίων.

```
SELECT u.birth_date, u.first_name, u.last_name, num_books
FROM user u
INNER JOIN (SELECT a.user_id, COUNT(*) AS num_books
FROM applications a
INNER JOIN user u ON u.user_id = a.user_id
WHERE u.role_name = 'teacher'
AND TIMESTAMPDIFF(YEAR, u.birth_date, CURRENT_DATE()) < 40
AND a.status_ IN ('borrowed', 'completed','expired_borrowing')
GROUP BY a.user_id
HAVING COUNT(*) = (SELECT COUNT(*) AS max_books
```

```
FROM applications a
INNER JOIN user u ON u.user_id = a.user_id
WHERE u.role_name = 'teacher'
AND TIMESTAMPDIFF(YEAR, u.birth_date, CURRENT_DATE()) < 40
AND a.status_ IN ('borrowed', 'completed', 'expired_borrowing')
GROUP BY a.user_id
ORDER BY max_books DESC
LIMIT 1)) counts_the_most ON u.user_id = counts_the_most.user_id
```

3.1.4. Βρείτε τους συγγραφείς των οποίων κανένα βιβλίο δεν έχει τύχει δανεισμού.

```
SELECT DISTINCT a.author_name
FROM author a
LEFT JOIN book_author ba ON a.author_id = ba.author_id
LEFT JOIN applications app ON ba.ISBN = app.ISBN
WHERE app.ISBN IS NULL
UNION
SELECT DISTINCT a.author_name
FROM author a
JOIN book_author ba ON a.author_id = ba.author_id
JOIN applications app ON ba.ISBN = app.ISBN
WHERE app.status_ NOT IN ('borrowed', 'completed', 'expired_borrowing')
```

3.1.5. Ποιοι χειριστές έχουν δανείσει τον ίδιο αριθμό βιβλίων σε διάστημα ενός έτους με περισσότερους από 20 δανεισμούς;

```
SELECT s.school_name,
COUNT(*) AS books_borrowed,
u_handlers.first_name AS handlers
FROM school s
INNER JOIN
user u_handlers ON s.school_name = u_handlers.school_name AND u_handlers.role_name =
'handler'
INNER JOIN
user u_students ON s.school_name = u_students.school_name AND u_students.role_name IN
('student', 'teacher')
INNER JOIN
applications a ON u_students.user_id = a.user_id
WHERE a.status_ IN ('borrowed', 'completed', 'expired_borrowing')
```

```
AND a.start_date >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
GROUP BY s.school_name
HAVING COUNT(*) > 20
```

3.1.6. Πολλά βιβλία καλύπτουν περισσότερες από μια κατηγορίες. Ανάμεσα σε ζεύγη πεδίων (π.χ. ιστορία και ποίηση) που είναι κοινά στα βιβλία, βρείτε τα 3 κορυφαία (top-3) ζεύγη που εμφανίστηκαν σε δανεισμούς.

```
SELECT c1.category_name AS category_name_1, c2.category_name AS category_name_2,
COUNT(app.ISBN) AS pair_count
FROM book_category bc1
JOIN book_category bc2 ON bc1.ISBN = bc2.ISBN AND bc1.category_id < bc2.category_id
JOIN category c1 ON bc1.category_id = c1.category_id
JOIN category c2 ON bc2.category_id = c2.category_id
JOIN applications app ON bc1.ISBN = app.ISBN
WHERE app.status_ IN ('borrowed','expired_borrowing','completed')
GROUP BY c1.category_name, c2.category_name
ORDER BY pair_count DESC
LIMIT 3
```

3.1.7. Βρείτε όλους τους συγγραφείς που έχουν γράψει τουλάχιστον 5 βιβλία λιγότερα από τον συγγραφέα με τα περισσότερα βιβλία.

```
SELECT a.author_name, COUNT(*) AS book_count
FROM book_author ba
INNER JOIN author a on ba.author_id = a.author_id
GROUP BY author_name
HAVING book_count <= (
SELECT COUNT(*)-5
FROM book_author ba
INNER JOIN author a ON ba.author_id = a.author_id
GROUP BY a.author_id
ORDER BY COUNT(*) DESC
LIMIT 1
)
ORDER BY book_count
```

3.2. (Χειριστής)

3.2.1. Παρουσίαση όλων των βιβλίων κατά Τίτλο, Συγγραφέα (Κριτήρια αναζήτησης: τίτλος/ κατηγορία/ συγγραφέας/ αντίτυπα).

```
SELECT b.*, q.available_copies,
        GROUP_CONCAT(DISTINCT a.author_name ORDER BY a.author_name SEPARATOR ',') AS
author_names,
        GROUP_CONCAT(DISTINCT c.category_name ORDER BY c.category_name SEPARATOR ',') AS
book_categories
        FROM (SELECT stores.ISBN, stores.available_copies FROM stores WHERE
stores.school_id = '{school_id}') q
        INNER JOIN book b ON b.ISBN = q.ISBN
        INNER JOIN book_category bc ON q.ISBN = bc.ISBN
        INNER JOIN category c ON bc.category_id = c.category_id
        INNER JOIN book_author ba ON q.ISBN = ba.ISBN
        INNER JOIN author a ON a.author_id = ba.author_id
        WHERE a.author_name = '{author_name}' AND c.category_name = '{category_name}' AND
q.available_copies = '{available_copies}'
        GROUP BY b.ISBN
```

3.2.2. Εύρεση όλων των δανειζόμενων που έχουν στην κατοχή τους τουλάχιστον ένα βιβλίο και έχουν καθυστερήσει την επιστροφή του. (Κριτήρια αναζήτησης: Όνομα, Επώνυμο, Ημέρες Καθυστέρησης).

```
SELECT u.username, u.first_name, u.last_name, u.role_name, expired_applications.ISBN,
expired_applications.start_date, expired_applications.expiration_date,
expired_applications.application_id
FROM (SELECT *
FROM applications WHERE status_ IN ('expired_borrowing', 'borrowed'))
expired_applications
INNER JOIN user u
ON u.user_id = expired_applications.user_id AND u.school_name = '{school_name}'
WHERE u.first_name = '{first_name}' AND u.last_name = '{last_name}' AND AND
DATEDIFF(NOW(), expired_applications.expiration_date) >= '{number}'
```

3.2.3. Μέσος Όρος Αξιολογήσεων ανά δανειζόμενο και κατηγορία (Κριτήρια αναζήτησης: χρήστης/ κατηγορία)

```
SELECT u.username, c.category_name, AVG(r.like_scale) AS average_rating
                        FROM user u
                        INNER JOIN (SELECT a.user_id, a.ISBN FROM applications a GROUP BY
a.user_id, a.ISBN) AS distinct_borrowings
                            ON u.user_id = distinct_borrowings.user_id
                        INNER JOIN review r ON distinct_borrowings.ISBN = r.ISBN AND
u.user_id = r.user_id
                        INNER JOIN book_category bc ON distinct_borrowings.ISBN = bc.ISBN
```

```
                    INNER JOIN category c ON bc.category_id = c.category_id
                    WHERE u.username = '{username}' AND c.category_name =
'category_name'
                     GROUP BY u.username, c.category_name
```

3.3. (Χρήστης)

3.3.1. Όλα τα βιβλία που έχουν καταχωριστεί (Κριτήρια αναζήτησης: τίτλος/ κατηγορία/ συγγραφέας), δυνατότητα επιλογής βιβλίου και δημιουργία αιτήματος κράτησης

```
SELECT b.*, q.available_copies,
        GROUP_CONCAT(DISTINCT a.author_name ORDER BY a.author_name SEPARATOR ',') AS
author_names,
        GROUP_CONCAT(DISTINCT c.category_name ORDER BY c.category_name SEPARATOR ',') AS
book_categories
        FROM (SELECT stores.ISBN, stores.available_copies FROM stores WHERE
stores.school_id = '{school_id}') q
        INNER JOIN book b ON b.ISBN = q.ISBN
        INNER JOIN book_category bc ON q.ISBN = bc.ISBN
        INNER JOIN category c ON bc.category_id = c.category_id
        INNER JOIN book_author ba ON q.ISBN = ba.ISBN
        INNER JOIN author a ON a.author_id = ba.author_id
        WHERE a.author_name = '{author_name}' AND c.category_name = '{category_name}'
```

3.3.2. Λίστα όλων των βιβλίων που έχει δανειστεί ο συγκεκριμένος χρήστης.

```
SELECT a.application_id, b.ISBN,b.title
FROM applications a
INNER JOIN user u ON a.user_id = u.user_id
INNER JOIN book b ON a.ISBN = b.ISBN
WHERE a.status_ = 'completed' AND u.user_id = {user_id}
```

Indices/Ευρετήρια

Για να ορίσουμε τα indices της βάσης αρχικά λαμβάνουμε υπόψην ότι στη MySQL/InnoDB Engine indices δημιουργούνται αυτόματα για τις στήλες που αποτελούν primary keys για κάθε table και στα σύνθετα(compound) primary keys δημιουργεί indices για όλες της στήλες που το αποτελούν. Αυτό είναι θεμελιώδες για την ταχεία και αποδοτική λειτουργία της βάσεις εφόσον τα primary keys γενικά αποτελούν IDs που χρησιμοποιούνται συνεχώς τόσο στα διάφορα queries (WHERE clauses, JOIN ON statements), όσο και στα triggers/δείκτες που χρησιμοποιούνται για να υλοποιήσουν περιορισμούς που αφορούν πολλαπλά tables. Επομένως είναι απαραίτητο η πρόσβαση σε κάθε primary key να είναι γρήγορη, κάτι που επιτυγχάνεται με τη χρήση ευρετηρίων. Επιπλέον χρησιμοποιούνται ευρετήρια σε όποια foreign keys ορίζονται και δεν είναι primary keys (οπότε έχουν ήδη ευρετήριο), γιατί είναι απαραίτητο κατά την ενημέρωση/διαγραφή κάποιου στοιχείου το οποίο είναι referenced από άλλο στοιχείο πρέπει να είναι όσο το δυνατόν γρηγορότερη η προσπέλαση του foreign key.

1.3.User Manual

Το home page του ιστότοπου μας είναι κοινό για όλους του users και φαίνεται στην παρακάτω εικόνα.

Από αυτό το σημείο κάθε χρήστης ακολουθεί διαφορετικά βήματα για να συνδεθεί.

Admin

Αρχικά, ο admin/διαχειριστής όλων σχολικών βιβλιοθηκών θα κλικάρει στο από το navigation bar στο στοιχείο Admin και αφού συμπληρώσει τα στοιχεία του θα μπορεί να συνδεθεί στη βάση μέσω του δικού του προφίλ. Σημειώνεται ότι ο admin είναι ο μόνος χρήστης που η είσοδος του λογαριασμού του απαιτείται να έχει προηγηθεί από τους σχεδιαστές της βάσης. Αφου το δοθεί ο κωδικός του και το username που θα είναι πάντα admin ο ίδιος μπορεί να τροποποιήσει μόνο τον κωδικό του.

Στη συνέχεια, ο admin δύναται από το navigation bar που του παρέχεται να κάνει τα εξής:

- Αρχική(3.1.1-3.1.7) : Στατιστικά των βιβλιοθηκών που αφορούν τα queries της εκφώνησης.
- Handlers: Να αποδέχεται αιτήματα εγγραφής από Handlers/Χειριστές βιβλιοθηκών και να μπορεί να βλέπει και τους active handlers και να τους διαγράφει.
- Schools: Να προσθέτει σχολεία, να τροποποιεί τα στοιχεία τους και να τα αφαιρεί από το δίκτυο
- Backup & Restore: Μπορεί να δημιουργήσει αντίγραφο ασφαλείας για όλη τη βάση (backup) και να επαναφέρει το σύστημα από αυτό (restore).
- Change Password: Αλλαγή κωδικού admin.
- Logout: Έξοδος από τον λογαριασμό του.

Handlers/Χειριστές

Οι handlers αφού συμπληρώσουν τα στοιχεία τους στο pop window που εμφανίζεται αφού πατήσουν Apply for school handler στην αρχική στέλνουν ένα αίτημα στον admin. Αφού, ο admin αποδεχτεί το αίτημα ο λογαριασμός του handler γίνεται ενεργός και μπορεί στη συνέχεια αφού διαλέξει το σχολείο του από το drop down list στο κουμπί της αρχικής choose you school να συνδεθεί (Login) στον λογαριασμό του.

Από την αρχική μετά το login του χρήστη μπορεί να αναζητήσει βιβλία με τα κατάλληλα φίλτρα και να πατήσει πάνω τους για να επεξεργαστεί τα στοιχεία τους.

Από το navigation bar έχει τις εξής δυνατότητες:

- Users: Να αποδέχεται αιτήματα χρηστών είτε καθηγητών είτε μαθητών ή να τα απορριπτεί αλλά και να βλέπει του ενεργούς χρήστες και να μπορεί να τους διαγράφει ή να αναστέλλει προσωρινά τη λειτουργία των λογαριασμών τους.
- Reservation Requests: Να αποδέχεται αιτήματα δανείσμου/κράτησης κατόπιν δανεισμού του βιβλίου ή να τα απορρίπτει. Ακόμη και ο ίδιος ο handler μπορεί να δημιουργήσει κρατήσεις με το `+` button.
- Borrowed Books: Να βλέπει τους τρέχοντες δανεισμούς και με τα καταλληλα φίλτρα να εντοπίζει εκπρόθεσμουυς δανεισμούς.
- School history: Το ιστορικό ολοκληρωμένων δανεισμών του σχολείου.
- Review requests: Αποδοχή ή απόρριψη αιτημάτων για δημοσίευση κριτικών του βιβλίων από μαθητές.(Των καθηγητών αναβαίνουν άμεσα).
- School Stats : Μέσος Όρος Αξιολογήσεων ανά δανειζόμενο και κατηγορία (Κριτήρια αναζήτησης: χρήστης/ κατηγορία).
- Change Password: Αλλαγή κωδικού.
- Logout : Έξοδος.

Users/Χρήστες

Οι χρήστες επιλέγουν το σχολείο τους από την αρχική και στη συνέχεια κάνουν Register συμπληρώνοντας τα στοιχεία τους αναμένοντας ενεργοποίηση του λογαριασμού τους από τον Υπεύθυνο Χειριστή της σχολικής του βιβλιοθήκης.

Από την αρχική μετά το login του χρήστη μπορεί να αναζητήσει βιβλία με τα κατάλληλα φίλτρα και να πατήσει πάνω τους για να δει περισσότερες πληροφορίες και να κάνει κράτηση σε αυτά.

Από το navigation bar έχει τις εξής δυνατότητες:

- My Reservations: Ενεργές και σε ουρά αναμονής κρατήσεις.
- My Borrows: Όλους τους ενεργούς δανεισμούς εκπρόθεσμους και μη.
- My History: Ιστορικό ολοκληρωμένων δανεισμών.
- Change password: Αλλαγή κωδικού.
- Profile_details: Αλλαγή στοιχείων μόνο από χρήστες-καθηγητές και πλήθος δανεισμών και κρατήσεων.

1.4.
Αναλυτικά βήματα εγκατάστασης της εφαρμογής και αρχεία τεχνοδιαμόρφωσης (configuration) που χρειάζονται για να εγκατασταθεί από την αρχή η εφαρμογή.

1)Δημιουργία τοπικού directory:

Για να εγκατασταθεί η εφαρμογή μας πρέπει να γίνει clone τοπικά στον υπολογιστή του user το git repo της εφαρμογής. Αυτό μπορεί να γίνει είτε με την εφαρμογή GitHub desktop, είτε μέσω κάποιου terminal με την εντολή git clone https://github.com/koustenischris/Db2023 στο directory που θέλουμε να εγκατασταθεί η εφαρμογή.

2) Δημιουργία της Βάσης δεδομένων(backend):

Για να εγκαταστήσουμε την βάση, πρέπει να υπάρχει ένας sql server. Ο χρήστης μπορεί να εγκαταστήσει το xampp το οποίο εκτός από server εμπεριέχει και ένα DBMS, το οποίο τρέχει σε localhost στο phpMyAdmin. Για να γίνει η δημιουργία του database, θα πρέπει να τρέξουμε καταρχάς το αρχεία στον φάκελο SQL_code με την ακόλουθη σειρά

Για τον σχεδιασμό τη βάσης :

1. myschema.sql
2. index.sql
3. triggers_and_events.sql

και για το populate της βάσης:

4. school_insertions.sql
5. user_insert.sql
6. book_insertions.sql
7. Authors&Categories_Insertions.sql
8. key_words.sql
9. stores.sql

3) Προσθήκη Modules της Python:

Θα χρειαστεί επίσης η εγκατάσταση κάποιων modules της Python, τα οποία βρίσκονται στο αρχείο requirements.txt. Ο χρήστης μπορεί να δημιουργήσει ένα virtual environment της Python αν θέλει, εναλλακτικά μπορεί να τα εγκαταστήσει αυτά τα modules και γενικά στον υπολογιστή του (πρέπει να βεβαιωθεί βέβαια ότι η γλώσσα Python που βρίσκεται στα environment variables συμβαδίζει με αυτή που θα έχει στο terminal, καθώς και με τον Interpreter που θα χρησιμοποιήσει. Εμείς χρησιμοποιήσαμε την γλώσσα Python 3.8 την οποία και συνιστούμε). Για να το κάνει αυτό, θα τρέξει στο terminal την εντολή pip install -r requirements.txt. Έπειτα, με το xaamp ενεργοποιημένο στα modules Apache και mySQL, τρέχουμε το αρχείο run.py, ανοίγουμε ένα browser και μεταβαίνουμε στη διεύθυνση localhost:3000. Έτσι εμφανίζεται η αρχική σελίδα της εφαρμογής.