DATABASES

SEMESTER PROJECT

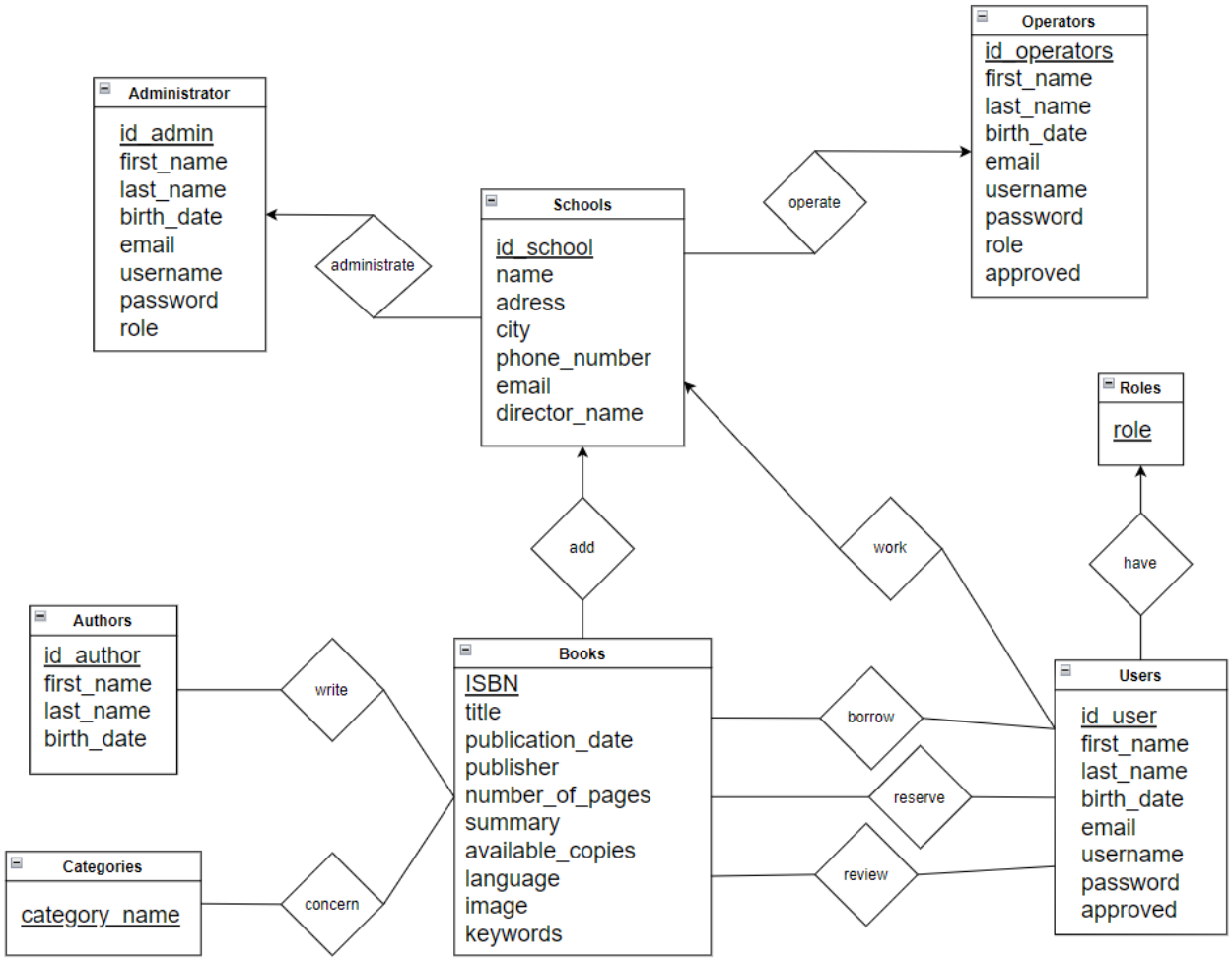
REPORT

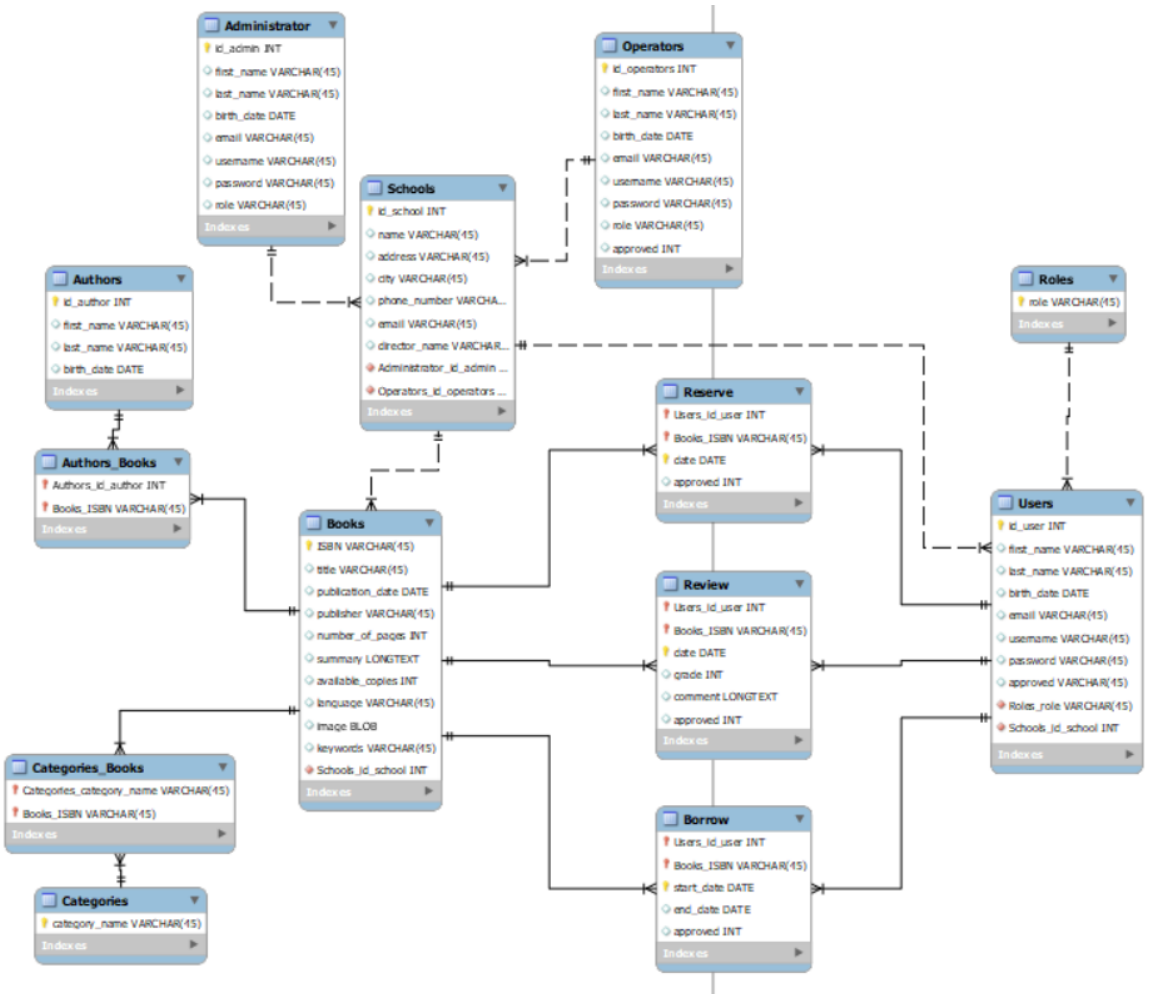
DUPUY Théo

BOULNOIS Boris

In this report, you will find a detailed presentation of our semester project from the databases course, 6th semester of the Electrical and computing engineering school of NTUA.

ER Diagram & Relational Schema Diagram




Administrator
id_admin INT
first_name VARCHAR(45)
last_name VARCHAR(45)
birth_date DATE
email VARCHAR(45)
username VARCHAR(45)
password VARCHAR(45)
role VARCHAR(45)
Indexes
Operators
id_operators INT
first_name VARCHAR(45)
last_name VARCHAR(45)
birth_date DATE
email VARCHAR(45)
username VARCHAR(45)
password VARCHAR(45)
role VARCHAR(45)
approved INT
Indexes
Schools
id_school INT
name VARCHAR(45)
address VARCHAR(45)
city VARCHAR(45)
phone_number VARCHA...
email VARCHAR(45)
director_name VARCHAR...
Administrator_id_admin ...
Operators_id_operators ...
Indexes
Authors
id_author INT
first_name VARCHAR(45)
last_name VARCHAR(45)
birth_date DATE
Indexes
Roles
role VARCHAR(45)
Indexes
Reserve
Users_id_user INT
Books_ISBN VARCHAR(45)
date DATE
approved INT
Indexes
Authors_Books
Authors_id_author INT
Books_ISBN VARCHAR(45)
Indexes
Books
ISBN VARCHAR(45)
title VARCHAR(45)
publication_date DATE
publisher VARCHAR(45)
number_of_pages INT
summary LONGTEXT
available_copies INT
language VARCHAR(45)
image BLOB
keywords VARCHAR(45)
Schools_id_school INT
Indexes
Users
id_user INT
first_name VARCHAR(45)
last_name VARCHAR(45)
birth_date DATE
email VARCHAR(45)
username VARCHAR(45)
password VARCHAR(45)
approved VARCHAR(45)
Roles_role VARCHAR(45)
Schools_id_school INT
Indexes
Review
Users_id_user INT
Books_ISBN VARCHAR(45)
date DATE
grade INT
comment LONGTEXT
approved INT
Indexes
Categories_Books
Categories_category_name VARCHAR(45)
Books_ISBN VARCHAR(45)
Indexes
Borrow
Users_id_user INT
Books_ISBN VARCHAR(45)
start_date DATE
end_date DATE
approved INT
Indexes
Categories
category_name VARCHAR(45)
Indexes

Our database is structured as follows:

➔ We find **8 entity sets**:

- **Schools:** List of all schools of the library network with their information.

- **Books:** List of all books of the library network with their information.
  → A book belongs to only one school

  Let's note that, in the way we structured our database, the attribute "available_copies" is an integer which states, for a given book, how many book there is on the network (whether they are borrowed or not). It does not represent the number of available books for a given instant t.

- **Authors:** List of all Authors of the library network with their personal information.
  → An author can write several books
  → A book is written by one or several authors

- **Categories:** List of categories to which the book belongs.
  → A book has one or several categories

- **Administrator:** Personal information of the only administrator of the library network.
  → Every school is administered by the administrator

- **Operators:** List of the different operators of the library network with their personal information.
  → Every school is administered by one operator

  In Operators, we find the attribute "approved" which is an integer whose value is equal to 1 to state that an operator is allowed to use the library network and whose value is equal to 0 to state that he can't use it. These rights are managed by the administrator.

- **Users:** List of all users (students and teachers) of the library network with their personal information.
  → Each user belongs to only one school
  → Users can borrow several books and books can be borrowed by several users
  → Users can reserve several books and books can be reserved by several users
  → Users can review several books and books can be reviewed by several users

  In user, we find the attribute "approved" which is an integer whose value is equal to 1 to state that a user is allowed to use the library network and whose value is equal to 0 to sate that he can't use it. When a user has his "approved" value equal to 0, he will

not be able to connect to the application when he enters his login and password. These rights are managed by the operators.

- **Roles:** List of users' roles (student or teacher).
  → Each user has only one role

  ➔ And we find **5 relationship sets**:

- **Authors_Books:** Associates every author with a book he has written and every book with an author who wrote it.

- **Categories_Books:** Associates every category with a book and every book with a category.

- **Borrow:** List of books which have been borrowed by users since the creation of the database. A book currently borrowed by a user is identified by its "end_date" attribute, which is null. On the opposite, if a book from Borrow has its "end_date" attribute which is not NULL, this means that the book has been returned.
  Let us note that the attribute "approved" is automatically set to 0 when a user asks to borrow a book. It is set to 1 after the approving of an operator.

- **Reserve:** List of books which have been reserved by users because no copies were available. Books from "Reserve" are automatically deleted after 1 week in the table or if the book returned from a loan. In the last case, the book is inserted into the "Borrow" table with an "approved" attribute automatically set to 1.
  The attribute "approved" on "Reserve" is automatically set to 0 when a user asks to reserve a book. It is set to 1 after the approving of an operator.

- **Review:** List of reviewed written by users. When a user writes a review, the attribute "approved" is automatically set to 0. It is set to 1 after the approving of an operator.

DDL & DML Script

The DDL & DML scripts are presented on the other files.

VIEWS

1. Registered_people

```
DROP VIEW IF EXISTS Registered_people ;
CREATE VIEW Registered_people AS
    SELECT id_admin id, username, password, role FROM Administrator
    UNION
    SELECT id_operators id, username, password, role FROM Operators WHERE approved = "1"
    UNION
    SELECT id_user id, username, password, Roles_role role FROM Users WHERE approved = "1";
```

The View “Registered_people” is used to check every people approved in the database. It is especially useful to allow the access to the database only to people who have been approved by administrator/operators and block the access for people who are not registered into the database or who do not have the permission to access to it.

2. Registration_demand

```
DROP VIEW IF EXISTS Registration_demand ;
CREATE VIEW Registration_demand AS
    SELECT * FROM Users WHERE approved = "0";
```

The View “Register_demand” is used to check for every users which don’t have access to the applciation. This is useful for opperators to check the new registration demand.

3. Review_demand

```
DROP VIEW IF EXISTS Review_demand ;
CREATE VIEW Review_demand AS
    SELECT * FROM Review WHERE approved = "0";
```

The View “Review_demand” is used to check for every review which has been written by a user. It is useful for the operators to approved any new review.

4. Borrow_demand

```
DROP VIEW IF EXISTS Borrow_demand ;
CREATE VIEW Borrow_demand AS
    SELECT * FROM Borrow WHERE approved = "0";
```

The View “Borrow_demand” is used to check on every demand of loan made by users. It is useful for the operators to approved any loan after checking that the user follows the loan criteria.

5. Books_borrowed

```
DROP VIEW IF EXISTS Books_borrowed ;
CREATE VIEW Books_borrowed AS
    SELECT br.Users_id_user id_user, br.Books_ISBN ISBN, br.start_date start_date, s.id_school id_school, u.Roles_role role
    FROM Borrow br
    INNER JOIN Users u ON u.id_user = br.Users_id_user
    INNER JOIN Schools s ON s.id_school = u.Schools_id_school
    WHERE br.end_date IS NULL AND br.approved = 1;
```

The view “Books_borrowed” returns all books which are currently borrowed by users. This is useful for many examples and especially to count how many books have a user borrowed so operators can check if he follows the loan criteria.

6. Available_books

```
DROP VIEW IF EXISTS Available_books;
CREATE VIEW Available_books AS
    SELECT b.ISBN, b.title, b.number_of_pages, b.language, b.summary, b.keywords, s.id_school FROM Books b
    INNER JOIN Schools s ON s.id_school = b.Schools_id_school
    WHERE (b.ISBN NOT IN (SELECT b.ISBN FROM Books b
        INNER JOIN (SELECT bb.ISBN, COUNT(*) compt FROM Books_borrowed bb GROUP BY bb.ISBN) T
        ON T.ISBN = b.ISBN WHERE b.available_copies = T.compt))
    GROUP BY ISBN;
```

The view “Available_books” returns all books which are currently available. This is useful to know, when a user wants to borrow a book, if he can borrow it or if a reservation has to be done.

PROCEDURES

1. Borrow_or_reserve

```
DROP PROCEDURE IF EXISTS Borrow_Or_Reserve;
DELIMITER //

CREATE PROCEDURE Borrow_Or_Reserve(IN book_ISBN VARCHAR(255), IN user INT, OUT location VARCHAR(45))
BEGIN
    DECLARE Count INT;

    SELECT COUNT(*) INTO Count
    FROM available_books
    WHERE ISBN = book_ISBN AND id_school = (SELECT Schools_id_school FROM Users WHERE id_user = user);

    IF Count > 0 THEN
        INSERT INTO Borrow (Books_ISBN,Users_id_user,start_date,end_date,approved) VALUES (book_ISBN, user, DATE(NOW()), NULL, 0);
        SET location = "Borrow";
    ELSE
        INSERT INTO Reserve (Books_ISBN,Users_id_user,date,approved) VALUES (book_ISBN, user, DATE(NOW()), 0);
        SET location = "Reserve";
    END IF;
END //

DELIMITER ;
```

The procedure “Borrow_Or_Reserve” is used to check, when a user wants to borrow a book, whether he can borrow it or he has to reserve it. This procedure is called when the user clicks on the button to get the book on the application. If the book is available, a borrow request is sent (which will have to be approved or denied by an operator) and if not, a reservation request is sent (which will also have to be approved or denied by an operator).

2. Check_books_available

```
DROP PROCEDURE IF EXISTS Check_books_available;
DELIMITER //

CREATE PROCEDURE Check_books_available()
    BEGIN
        INSERT INTO Borrow (Books_ISBN,Users_id_user,start_date,end_date,approved)
        SELECT r.Books_ISBN, r.Users_id_user, DATE(NOW()), NULL, 1
        FROM Reserve r
        INNER JOIN available_books ab ON ab.ISBN = r.Books_ISBN
        WHERE (r.Books_ISBN = ab.ISBN) AND r.approved = 1 AND
        (EXISTS (SELECT * FROM Reserve rr INNER JOIN available_books ab ON ab.ISBN = r.Books_ISBN WHERE rr.Books_ISBN = ab.ISBN));

        DELETE FROM Reserve WHERE (Books_ISBN = (SELECT bb.ISBN FROM books_borrowed bb))
        AND (Users_id_user = (SELECT bb.id_user FROM books_borrowed bb)) AND approved = 1;
    END//

DELIMITER ;
```

The procedure “Check_books_available” is called on the event “Check_availability” (see below) so it can be executed every 10s. We will see in the next paragraph a trigger called “books_return” which checks, when a book returns from a loan, if there is such a same book which has been reserved from a user and can therefore be borrowed.

But imagine now that a user wants a book, that the book is not available because all copies have already been borrowed, then the user has to send a reservation request for the book (which is done automatically when he clicks on the button as we’ve seen previously). An operator has to approve this reservation request. If he does it before the return of the book, then when the book will be returned it will activate the trigger “books_return” and the user will receive the book. But if the book is returned before that the operator approved the reservation, then when the reservation will be approved by the operator the trigger would have already been activated and the book will stay in the table “reserve” although it shouldn’t because the book is available. Therefore, by executing the procedure “Check_books_available” every 10s on the event “Check_availability” every time a book which have been reserved from a user and returned from the loan of another user, then it will be borrowed by the next user.

TRIGGERS

1. Books_return

```
DROP TABLE IF EXISTS Return_books;
CREATE TABLE Return_books (
  id_user INT NOT NULL,
  ISBN VARCHAR(45) NOT NULL,
  end DATE NULL
);

DROP TRIGGER IF EXISTS books_return;
DELIMITER //

CREATE TRIGGER books_return
AFTER INSERT ON Return_books
FOR EACH ROW
BEGIN

    UPDATE Borrow SET end_date = DATE(NOW())
    WHERE end_date IS NULL AND approved = 1 AND Books_ISBN = NEW.ISBN AND Users_id_user = NEW.id_user AND DATE(NOW()) = NEW.end;

    INSERT INTO Borrow (Books_ISBN,Users_id_user,start_date,end_date,approved)
    SELECT r.Books_ISBN, r.Users_id_user, DATE(NOW()), NULL, 1
    FROM Reserve r
    WHERE (r.Books_ISBN = NEW.ISBN) AND r.approved = 1 AND
    (EXISTS (SELECT * FROM Reserve rr INNER JOIN Return_books rb ON rr.Books_ISBN = rb.ISBN WHERE rr.Books_ISBN = rb.ISBN));

    DELETE FROM Reserve WHERE (Books_ISBN = NEW.ISBN) AND approved = 1;
END //

DELIMITER ;
```

We've already talked briefly about the trigger "books_return" in the previous paragraph. It is automatically activated when an operator adds to the database that a book is back from a loan to check if there is any such book in the "reserve" table that must be insert into the "borrow" table (and also delete from the "reserve" table).

Let us note that we need to use a table "Return_books" that is used only with this trigger. Indeed, as we've talked about when we described previously the relationship set "Borrow", a book from this relationship set that is currently borrowed has an attribute "end_date" sets to NULL. When such book is returned from a loan, we must find an automatic process which set the "end_date" from NULL to the date when the operator handles this return. At the same time, the trigger must activate to decide whether there is a book from the "reserve" relationship set who can be insert into "borrow". Therefore the easier way to do this would be to follow the following process :

- When a book is returned, the operator clicks on a button of the application which update the end_date of the book from NULL to the daily date.
- For every updates on the "borrow" set, if a returned book is reserved by another user, we insert into "borrow" the book with the new user id and we delete it from the "reserve set"

But in SQL, we can't create a trigger which is activated by on update on a given table and which effects are on the same table. That is why we've created a table "Return_books" which is used to add every book that is returned. The trigger modified therefore the sets "Borrow" and "Reserve" for every insert into "Return_books".

EVENTS

1. delete_expired_reservations

```
DROP EVENT IF EXISTS delete_expired_reservations;
CREATE EVENT delete_expired_reservations
ON SCHEDULE EVERY 1 DAY
DO
    DELETE FROM Reserve WHERE (DATE_ADD(date, INTERVAL 7 DAY) < DATE(NOW())) AND (approved = 1);
```

This event is used to delete every approved reservation which remained more than one week.

2. erase_return_books

```
DROP EVENT IF EXISTS Erase_Return_books;
CREATE EVENT Erase_Return_books
ON SCHEDULE EVERY 1 HOUR
DO
    DELETE FROM Return_books;
```

This event is used to delete every element in Return_books, since it is only used to activate the trigger.

3. Check_availability

```
DROP EVENT IF EXISTS Check_availability;
CREATE EVENT Check_availability
ON SCHEDULE EVERY 10 SECOND
DO
    CALL Check_books_available()
```

As we've talked about in the procedure part, this event is used to call the procedure "Check_books_available".

QUERIES

For every following query, every variable which begins by "@" is a user input.

- Administrator

4.1.1. List with the total number of loans per school (Search criteria: year, calendar month)

```
SELECT s.name school, COUNT(br.Books_ISBN) number_loans
FROM Schools s
INNER JOIN Books b ON b.Schools_id_school = s.id_school
INNER JOIN Borrow br ON br.Books_ISBN = b.ISBN
WHERE
    CASE
        WHEN (@year != '' AND @month != '')
            THEN (YEAR(br.start_date) = @year AND MONTH(br.start_date) = @month)
        WHEN (@year = '' AND @month != '')
            THEN (MONTH(br.start_date) = @month)
        WHEN (@year != '' AND @month = '')
            THEN (YEAR(br.start_date) = @year)
    END
GROUP BY s.name;
```

4.1.2. For a given book category (user-selected), which authors belong to it and which teachers have borrowed books from that category in the last year?

```
SELECT CONCAT(a.first_name, ' ', a.last_name) authors_names
FROM Authors a
INNER JOIN Authors_Books ab ON ab.Authors_id_author = a.id_author
INNER JOIN Books b ON b.ISBN = ab.Books_ISBN
INNER JOIN Categories_Books ca ON ca.Books_ISBN = b.ISBN
WHERE ca.Categories_category_name = @category
GROUP BY a.id_author;

SELECT CONCAT(u.first_name, ' ', u.last_name) teachers_names
FROM Users u
INNER JOIN Borrow bor ON bor.Users_id_user = u.id_user
INNER JOIN Books boo ON boo.ISBN = bor.Books_ISBN
INNER JOIN Categories_Books cb ON cb.Books_ISBN = boo.ISBN
WHERE (u.Roles_role = 'teacher'
    AND cb.Categories_category_name = @category
    AND bor.start_date > DATE_ADD(DATE(NOW()), INTERVAL -1 YEAR));
```

4.1.3. Find young teachers (age < 40 years) who have borrowed the most books and the number of books.

```
SELECT CONCAT(u.first_name, " ", u.last_name) "Teachers names", COUNT(*) "Number of books borrowed"
FROM Users u
INNER JOIN Borrow b ON u.id_user = b.Users_id_user
WHERE (u.birth_date > DATE_ADD(DATE(NOW()), INTERVAL -40 YEAR)) AND (u.Roles_role = "teacher")
GROUP BY u.id_user
ORDER BY "Number of books borrowed" DESC
LIMIT 1;
```

4.1.4. Find authors whose books have not been borrowed.

```
SELECT CONCAT(a.first_name, " ", a.last_name) "Authors names"
FROM Authors a
WHERE a.id_author NOT IN (SELECT distinct(a.id_author)
    FROM Authors a
    INNER JOIN Authors_Books ab ON a.id_author = ab.Authors_id_author
    INNER JOIN Books b ON ab.Books_ISBN = b.ISBN
    INNER JOIN Borrow br ON b.ISBN = br.Books_ISBN);
```

4.1.5. Which operators have loaned the same number of books in a year with more than 20 loans?

```
SET @year = "2021";

SELECT o.id_operators, o.first_name, o.last_name, COUNT(br.Books_ISBN) AS Books_loaned
FROM Operators o
INNER JOIN Schools s ON s.Operators_id_operators = o.id_operators
INNER JOIN Books b ON b.Schools_id_school = s.id_school
INNER JOIN Borrow br ON br.Books_ISBN = b.ISBN
WHERE YEAR(br.start_date) = @year
GROUP BY o.id_operators, o.first_name, o.last_name
HAVING COUNT(br.Books_ISBN) > 20;
```

4.1.6. Many books cover more than one category. Among field pairs (e.g., history and poetry) that are common in books, find the top-3 pairs that appeared in borrowings.

```
SELECT T.Categories, COUNT(T.Categories) Borrowing_times
FROM (
    SELECT br.Books_ISBN ISBN, GROUP_CONCAT(cb.Categories_category_name) Categories, br.start_date
    FROM Borrow br
    INNER JOIN Books b ON b.ISBN = br.Books_ISBN
    INNER JOIN Categories_Books cb ON cb.Books_ISBN = b.ISBN
    WHERE br.Books_ISBN IN (SELECT Books_ISBN FROM categories_books GROUP BY Books_ISBN HAVING COUNT(Books_ISBN)=2)
    GROUP BY br.Books_ISBN, br.start_date) T
GROUP BY T.Categories
ORDER BY Borrowing_times DESC
LIMIT 3;
```

4.1.7. Find all authors who have written at least 5 books less than the author with the most books.

```
SELECT CONCAT(a.first_name, " ", a.last_name) "Authors names", T.Nbr_books "Number of books" FROM Authors a
INNER JOIN (SELECT ab.Authors_id_author, COUNT(ab.Authors_id_author) AS Nbr_books
    FROM Authors_Books ab
    GROUP BY ab.Authors_id_author) T
        ON T.Authors_id_author = a.id_author
WHERE T.Nbr_books <= (SELECT MAX(T.Nbr_books) - 5 FROM
    (SELECT ab.Authors_id_author, COUNT(ab.Authors_id_author) AS Nbr_books
    FROM Authors_Books ab
    GROUP BY ab.Authors_id_author) T);
```

- Operator

4.2.1. All books by Title, Author (Search criteria: title/ category/ author/ copies).

```
SELECT b.*, cb.Categories_category_name FROM Books b
INNER JOIN Authors_Books ab ON ab.Books_ISBN = b.ISBN
INNER JOIN Authors a ON a.id_author = ab.Authors_id_author
INNER JOIN Categories_Books cb ON cb.Books_ISBN = b.ISBN
INNER JOIN Schools s ON s.id_school = b.Schools_id_school
WHERE s.Operators_id_operators = @id_op AND
    CASE
        #title, category, author, copies
        WHEN (@title != "" AND @category != "" AND @author != "" AND @copies != "")
            THEN (b.title = @title AND cb.Categories_category_name = @category AND a.last_name = @author AND b.available_copies = @copies)
        #title, category, author
        WHEN (@title != "" AND @category != "" AND @author != "" AND @copies = "")
            THEN (b.title = @title AND cb.Categories_category_name = @category AND a.last_name = @author)
        #title, category, copies
        WHEN (@title != "" AND @category != "" AND @author = "" AND @copies != "")
            THEN (b.title = @title AND cb.Categories_category_name = @category AND b.available_copies = @copies)
        #title, author, copies
        WHEN (@title != "" AND @category = "" AND @author != "" AND @copies != "")
            THEN (b.title = @title AND a.last_name = @author AND b.available_copies = @copies)
        #category, author, copies
        WHEN (@title = "" AND @category != "" AND @author != "" AND @copies != "")
            THEN (cb.Categories_category_name = @category AND a.last_name = @author AND b.available_copies = @copies)
        #category, author
        WHEN (@title = "" AND @category != "" AND @author != "" AND @copies = "")
            THEN (cb.Categories_category_name = @category AND a.last_name = @author)
        #category, copies
        WHEN (@title = "" AND @category != "" AND @author = "" AND @copies != "")
            THEN (cb.Categories_category_name = @category AND b.available_copies = @copies)
        #title, category
        WHEN (@title != "" AND @category != "" AND @author = "" AND @copies = "")
            THEN (b.title = @title AND cb.Categories_category_name = @category)
        #title, author
        WHEN (@title != "" AND @category = "" AND @author != "" AND @copies = "")
            THEN (b.title = @title AND a.last_name = @author)
        #title, copies
        WHEN (@title != "" AND @category = "" AND @author = "" AND @copies != "")
            THEN (b.title = @title AND b.available_copies = @copies)
        #author, copies
        WHEN (@title = "" AND @category = "" AND @author != "" AND @copies != "")
            THEN (a.last_name = @author AND b.available_copies = @copies)
        #title
        WHEN (@title != "" AND @category = "" AND @author = "" AND @copies = "")
            THEN (b.title = @title)
        #category
        WHEN (@title = "" AND @category != "" AND @author = "" AND @copies = "")
            THEN (cb.Categories_category_name = @category)
        #author
        WHEN (@title = "" AND @category = "" AND @author != "" AND @copies = "")
            THEN (a.last_name = @author)
        #copies
        WHEN (@title = "" AND @category = "" AND @author = "" AND @copies != "")
            THEN (b.available_copies = @copies)
    END;
```

4.2.2. Find all borrowers who own at least one book and have delayed its return. (Search criteria: First Name, Last Name, Delay Days).

```
SELECT u.first_name, u.last_name, DATEDIFF(DATE(NOW()), DATE_ADD(br.start_date, INTERVAL 7 DAY)) AS Delayed_days
FROM Users u
INNER JOIN Borrow br ON br.Users_id_user = u.id_user
INNER JOIN Schools s ON s.id_school = u.Schools_id_school
WHERE (br.end_date IS NULL)
    AND (DATE_ADD(br.start_date, INTERVAL 7 DAY) < DATE(NOW()))
    AND br.approved = 1
    AND s.Operators_id_operators = @id_op;
```

4.2.3. Average Ratings per borrower and category (Search criteria: user/category).

```
SELECT AVG(r.grade) AS avg_grade
FROM Review r
INNER JOIN Books b ON b.ISBN = r.Books_ISBN
INNER JOIN Categories_Books cb ON cb.Books_ISBN = b.ISBN
INNER JOIN Schools s ON s.id_school = b.Schools_id_school
WHERE s.Operators_id_operators = @id_op AND
    CASE
        WHEN (@id_user != '' AND @category != '')
            THEN (r.Users_id_user = @id_user AND cb.Categories_category_name = @category)
        WHEN (@id_user != '' AND @category = '')
            THEN (r.Users_id_user = @id_user)
        WHEN (@id_user = '' AND @category != '')
            THEN (cb.Categories_category_name = @category)
    END;
```

- User

4.3.1. List with all books (Search criteria: title/category/author), ability to select a book and create a reservation request.

```
SELECT b.*, a.first_name, a.last_name, cb.Categories_category_name
FROM Books b
INNER JOIN Authors_Books ab ON ab.Books_ISBN = b.ISBN
INNER JOIN Authors a ON a.id_author = ab.Authors_id_author
INNER JOIN Categories_Books cb ON cb.Books_ISBN = b.ISBN
INNER JOIN Schools s ON s.id_school = b.Schools_id_school
WHERE s.id_school = (SELECT Schools_id_school FROM Users WHERE id_user = @id_user) AND
    CASE
        #title, category, author
        WHEN (@title != "" AND @category != "" AND @author != "")
            THEN (b.title = @title AND cb.Categories_category_name = @category AND a.last_name = @author)
        #title, category
        WHEN (@title != "" AND @category != "" AND @author = "")
            THEN (b.title = @title AND cb.Categories_category_name = @category)
        #category, author
        WHEN (@title = "" AND @category != "" AND @author != "")
            THEN (cb.Categories_category_name = @category AND a.last_name = @author)
        #title, author
        WHEN (@title != "" AND @category = "" AND @author != "")
            THEN (b.title = @title AND a.last_name = @author)
        #title
        WHEN (@title != "" AND @category = "" AND @author = "")
            THEN (b.title = @title)
        #category
        WHEN (@title = "" AND @category != "" AND @author = "")
            THEN (cb.Categories_category_name = @category)
        #author
        WHEN (@title = "" AND @category = "" AND @author != "")
            THEN (a.last_name = @author)
    END;
```

4.3.2. List of all books borrowed by this user.

```
SELECT b.ISBN, b.title, a.first_name, a.last_name, cb.Categories_category_name, br.start_date, br.end_date
FROM Books b
INNER JOIN Authors_Books ab ON ab.Books_ISBN = b.ISBN
INNER JOIN Authors a ON a.id_author = ab.Authors_id_author
INNER JOIN (SELECT Books_ISBN, GROUP_CONCAT(Categories_category_name) Categories_category_name
    FROM Categories_Books GROUP BY Books_ISBN) cb ON cb.Books_ISBN = b.ISBN
INNER JOIN Borrow br ON br.Books_ISBN = b.ISBN
WHERE br.Users_id_user = @id_user AND br.approved = 1
ORDER BY br.start_date DESC;
```