Comp 306- Group Project: OdinLibrary

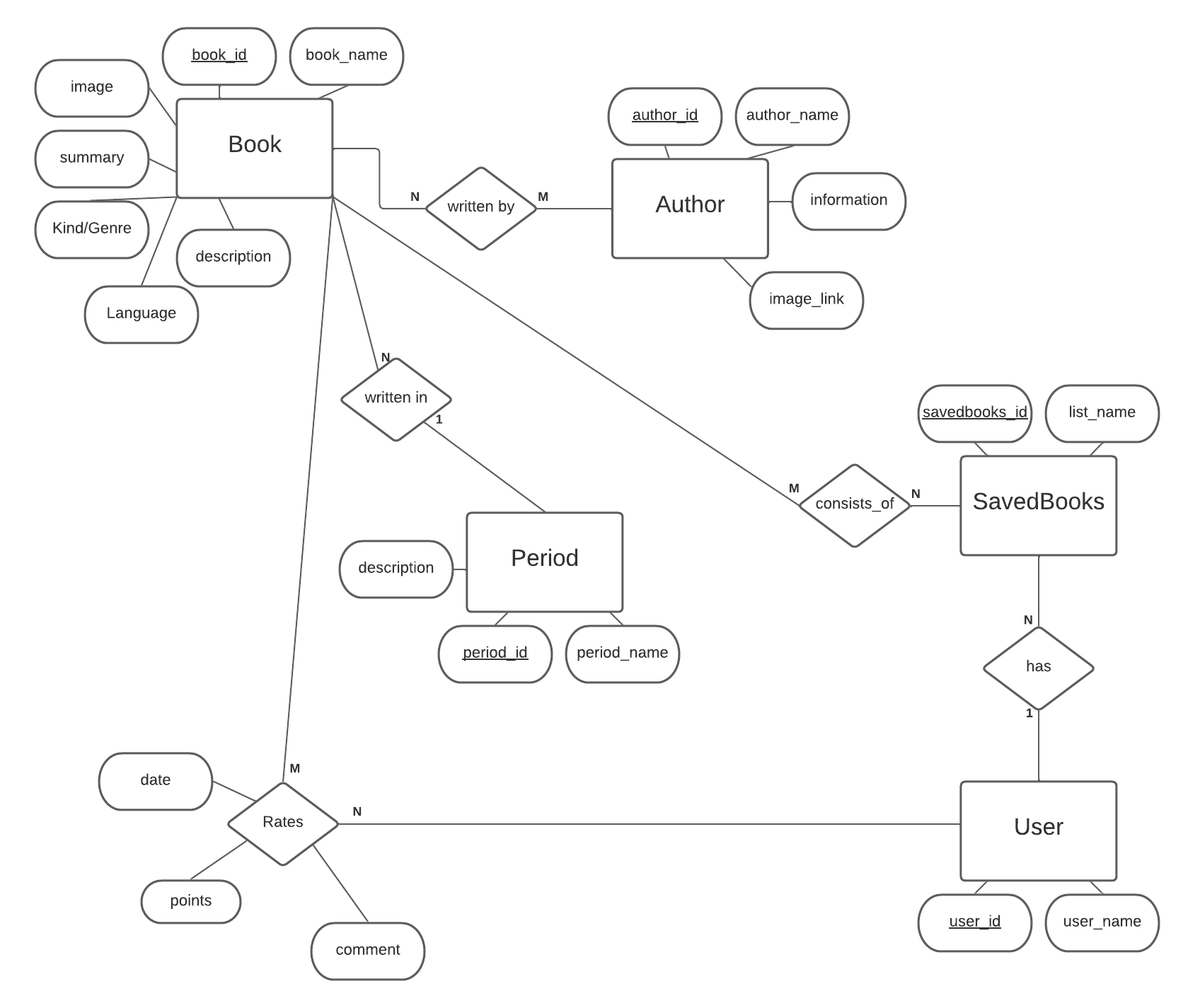
*Alp Özaslan, İrem Demir, Özlem Şerifoğulları*

Project Repository: [OdinLibrary](https://github.com/iremddemir/odinlibrary)

Project Description:

OdinLibraryis a library application that a user can find books, authors, periods and can save their favorite books into lists that created by themselves. Books, authors, and periods have little information that a user can enjoy reading. Moreover, we generated a database that consists of books frequently asked in Turkish University Exam (AYT). One of our aims is to help students to get this exam related information in a structured way rather than mixed up PDFs and blogs that they can find on internet.

Entity-Relationship Diagram:



Relational Database Design:

Following the ER to Relational Model conversion algorithm we have obtained the following relations to create our OdinLibraryDB.

CREATE TABLE book

(

book\_id INT,

book\_name VARCHAR(500),

description VARCHAR(500),

summary VARCHAR(500),

image VARCHAR(500),

kind\_genre VARCHAR(100),

language VARCHAR(100),

period\_id INT,

PRIMARY KEY (book\_id),

FOREIGN KEY (period\_id) REFERENCES period(period\_id)

);

CREATE TABLE author

(

author\_id INT,

author\_name VARCHAR(100),

information VARCHAR(500),

image\_link VARCHAR(500),

PRIMARY KEY (author\_id)

);

CREATE TABLE period

(

period\_id INT,

period\_name VARCHAR(100),

description VARCHAR(500),

PRIMARY KEY (period\_id)

);

CREATE TABLE savedBooks

(

savedbooks\_id INT,

list\_name VARCHAR(100),

user\_id INT,

PRIMARY KEY (savedbooks\_id),

FOREIGN KEY (user\_id) REFERENCES user(user\_id)

);

CREATE TABLE user

(

user\_id INT,

user\_name VARCHAR(100),

PRIMARY KEY (user\_id)

);

CREATE TABLE author\_book

(

book\_id INT,

author\_id INT,

FOREIGN KEY (book\_id) REFERENCES book(book\_id),

FOREIGN KEY (author\_id) REFERENCES author(author\_id),

PRIMARY KEY (book\_id, author\_id)

);

CREATE TABLE user\_book

(

book\_id INT,

user\_id INT,

date timestamp,

points INT,

comment VARCHAR(100),

FOREIGN KEY (book\_id) REFERENCES book(book\_id),

FOREIGN KEY (user\_id) REFERENCES user(user\_id),

PRIMARY KEY (book\_id, user\_id)

);

CREATE TABLE book\_savedBooks

(

book\_id INT,

savedbooks\_id INT,

FOREIGN KEY (book\_id) REFERENCES book(book\_id),

FOREIGN KEY (savedbooks\_id) REFERENCES savedBooks(savedbooks\_id),

PRIMARY KEY (book\_id, savedbooks\_id)

);

Data Sources:

Our first aim was to populate a database that will be a good representative of university exam (AYT) questions so that students can benefit from a more structured, organized data rather than long websites. In the light of this, AYT related information is gathered from online resources and using various tools it is formatted in the MySQL insert into statements. We created a database that fits our aim. As a future expansion of our application, we are planning to add books and authors from different literatures and make our platform usable with various purposes.

Complex SQL Queries:

1) highest rating books by their period

select b.book\_name, a.author\_name, p.period\_name, mt.total\_points

from book b

join (select book\_id, sum(points) as total\_points

from user\_book

group by book\_id) bp on bp.book\_id = b.book\_id

join author\_book ab on ab.book\_id = b.book\_id

join author a on ab.author\_id = a.author\_id

join period p on p.period\_id = b.period\_id

join (select p.period\_id, max(total\_points) as total\_points

from book b

join (select book\_id, sum(points) as total\_points

from user\_book

group by book\_id) bp on bp.book\_id = b.book\_id

join author\_book ab on ab.book\_id = b.book\_id

join author a on ab.author\_id = a.author\_id

join period p on p.period\_id = b.period\_id

group by p.period\_id) mt on mt.total\_points = bp.total\_points;

2) authors that wrote in more than one period

select a.\*

from author\_book ab

join author a on a.author\_id = ab.author\_id

join book b on b.book\_id = ab.book\_id

join period p on p.period\_id = b.period\_id

group by a.author\_id

having count(distinct(p.period\_id)) >1;

3) fuzzy search on book\_name with wanted book display information & this can be also applied to other searches

select ban.book\_name, ban.description, ban.author\_name, abr.average\_bookrating from

(select avg(user\_book.points) as average\_bookrating, user\_book.book\_id from user\_book group by user\_book.book\_id)abr right join

(select author.author\_name, baid.book\_name, baid.description, baid.book\_id from author

join (select book.book\_name, book.description, author\_book.author\_id, book.book\_id from book

join author\_book on book.book\_id = author\_book.book\_id) baid on author.author\_id = baid.author\_id) ban

on ban.book\_id = abr.book\_id

where ban.book\_name like '%hay%'

4) users with highest activity is in the last 3 month in order

-- can give limit function to limit number of records that will be shown

select u.user\_name, COUNT(book\_id) as book\_count

from (select \*

from user\_book

where date > now() - interval 3 month) ub

join user u on u.user\_id = ub.user\_id

group by ub.user\_id

order by book\_count DESC;

5) most saved books in order

-- can give limit function to limit number of records that will be shown

select b.book\_id, b.book\_name, p.period\_name, count(\*) as book\_saved\_count

from savedBooks sb

join book\_savedBooks bsb on sb.savedbooks\_id = bsb.savedbooks\_id

join book b on b.book\_id = bsb.book\_id

join period p on p.period\_id = b.period\_id

group by bsb.book\_id

order by book\_saved\_count DESC;

6) most popular authors in order

select a.name, bp.total\_points

from book b

join (select book\_id, sum(points) as total\_points

from user\_book

group by book\_id) bp on bp.book\_id = b.book\_id

join author\_book ab on ab.book\_id = b.book\_id

join author a on a.author\_id = ab.author\_id

order by bp.total\_points DESC

limit 5;

***We also added limit & offset function for our queries to implement pagination.***

Screenshots:

Ad