Bilkent University

Department of Computer Engineering

CS353 Term Project Design Report

Social Cataloging Platform for Books: BookCase

Assigned TA

Mustafa Can Çavdar

Group 12

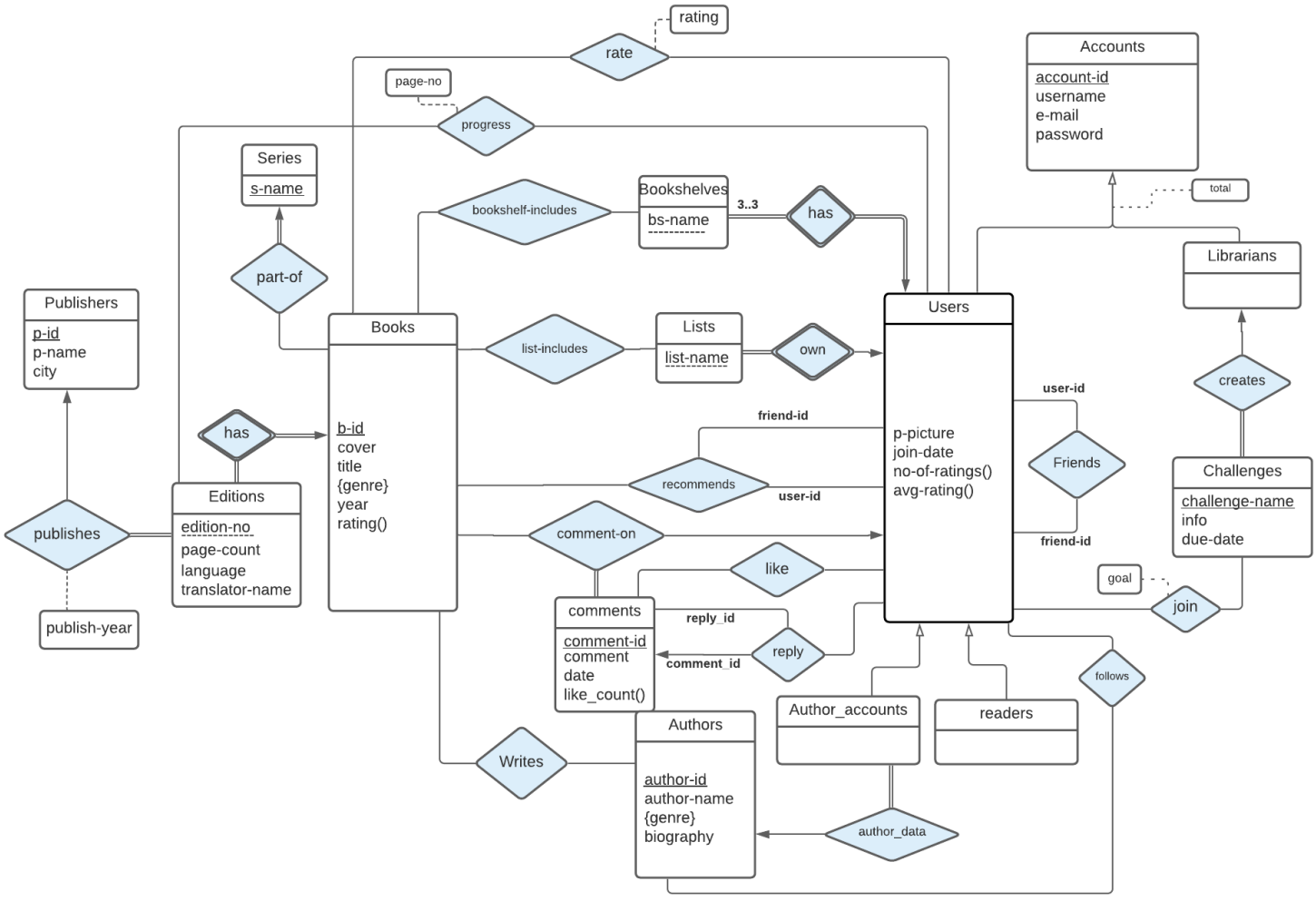
Ahmet Cemal Alıcıoğlu

Beril Canbulan

Kübra Okumuş

Ömer Faruk Kayar

1. Revised E/R Diagram



Changes:

- ➔ Added replying to a comment as an additional feature. Users may reply to a comment or another reply. This may provide a better discussion environment about a book.
- ➔ Removed translations table, and embedded it inside editions. If an edition has no translator, then its translator will have null as its value.
- ➔ Added publish year information to the “publishes” relation.

- ➔ Fixed problems with some weak entities and multiplicities.
- ➔ Changed the bookshelf structure by removing the three tables “read”, “to-read” and “currently-reading.” Instead of this structure, added a table-name attribute to bookshelves to hold this information that distincts the bookshelves.

2. Table Schemas

2.1 Accounts

Relational Model

Accounts(<u>account-id</u>, username, e-mail, password)

Functional Dependencies

account-id → username, e-mail, password

Candidate Keys

{(account-id), (username)}

Normal Form

3NF

Table Definition

create table Accounts(

account-id **int** not null auto_increment,

username **varchar**(15) not null,

e-mail **varchar**(30) not null,

password **varchar**(10) not null,

primary key (account-id)

);

2.2 Users

Relational Model

Users(user-id, p-picture, join-date)

FK: user-id references Accounts(account-id)

Functional Dependencies

user-id → p-picture, join-date

Candidate Keys

{(user-id)}

Normal Form

3NF

Table Definition

create table Users(

user-id **int** not null,

p-picture **image,**

join-date **date** not null,

primary key (user-id),

foreign key(user-id) **references** Accounts(account-id)

);

2.3 Librarians

Relational Model

Librarians(librarian-id)

FK: librarian-id references Accounts(account-id)

Functional Dependencies

None

Candidate Keys

{(librarian-id)}

Normal Form

3NF

Table Definition

create table Librarians(

librarian-id **int** not null,

primary key (librarian-id),

foreign key(librarian-id) **references** Accounts(librarian-id)

);

2.4 Challenges

Relational Model

Challenges(challenge-name, librarian-id, info, due-date)

Functional Dependencies

challenge-name→account-id, info, due-date

Candidate Keys

{(challenge-name)}

Normal Form

3NF

Table Definition

create table Challenges(

challange-name **varchar**(32) not null,

librarian-id **int** not null,

Info **varchar**(100) not null,

due-date **date** not null,

primary key (challange-name),

foreign key(librarian-id) **references** Librarians

);

2.5 join

Relational Model

join(challenge-name, user-id, goal)

FK: challenge-name references Challenges

FK: user-id references Users

Functional Dependencies

challenge-name, user-id→goal

Candidate Keys

{(challenge-name, user-id)}

Normal Form

3NF

Table Definition

create table Challenges(

challange-name **varchar**(32) not null,

user-id **int** not null,

goal **int** not null,

primary key (challange-name, account-id),

foreign key(challange-name) **references** Challenges

foreign key(user-id) **references** Users

);

2.6 Friends

Relational Model

Friends(user-id, friend-id)

FK: user-id references Users

FK: friend-id references Users(user-id)

Functional Dependencies

None

Candidate Keys

{(user-id, friend-id)}

Normal Form

3NF

Table Definition

create table Friends(

    user-id **int** not null,

    friend-id **int** not null,

    primary key (user-id, friend-id),

    foreign key(user-id) **references** Users(user-id)

    foreign key(friend-id) **references** Users(user-id)

);

2.7 Author_accounts

Relational Model

author_accounts(<u>author-id</u>)

FK: author-id references Users(user-id)

Functional Dependencies

None

Candidate Keys

{(author-id)}

Normal Form

3NF

Table Definition

create table Author_accounts(

author-id **int** not null,

primary key (author-id),

foreign key(author-id) **references** Users(user-id),

);

2.8 Authors

Relational Model

Authors(<u>author-id</u>, author-name, biography)

Functional Dependencies

author-id → author-name, biography

Candidate Keys

{(author-id)}

Normal Form

3NF

Table Definition

create table Authors(

author-id **int** not null,

author-name **varchar**(15) not null,

biography **varchar**(1000) not null,

primary key (author-id)

);

2.9 Author-genre

Relational Model

Author-genre(author-id, genre)

FK: author-id references Authors

Functional Dependencies

None

Candidate Keys

{(author-id, genre)}

Normal Form

3NF

Table Definition

create table Author-genre(

author-id **int** not null,

genre **varchar**(20) not null,

foreign key (author-id) **references** Authors(author-id)

primary key (author-id, genre)

);

2.10 follows

Relational Model

follows(<u>user-id, author-id</u>)

FK: user-id references Users

FK: author-id references Authors

Functional Dependencies

None

Candidate Keys

{(user-id, author-id)}

Normal Form

3NF

Table Definition

create table follows(

    user-id **int** not null,

    author-id **int** not null,

    primary key (user-id, author-id),

    foreign key(user-id) references Users

    foreign key(author-id) references Authors

);

2.11 author_data

Relational Model

author_data(<u>account-id</u>, author-id)

FK: account-id references Author_accounts(author-id)

FK: author-id references Authors(author-id)

Functional Dependencies

account-id → author-id

Candidate Keys

{(account-id)}

Normal Form

3NF

Table Definition

create table author_data(

account-id **int** not null,

author-id **int** not null,

primary key (account-id),

foreign key(account-id) references Author_accounts(author-id),

foreign key(author-id) references Authors(author-id)

);

2.12 Books

Relational Model

Books(b-id, cover, title, year)

Functional Dependencies

b-id → cover, title, year

Candidate Keys

{(b-id)}

Normal Form

3NF

Table Definition

create table Books(

b-id **int** not null auto_increment,

cover **image,**

title **varchar**(100) not null,

year **YEAR** not null**,**

primary key (b-id)

);

2.13 Book-genre

Relational Model

Book-genre(b-id, genre)

FK: b-id references Books

Functional Dependencies

None

Candidate Keys

{(b-id, genre)}

Normal Form

3NF

Table Definition

create table Book-genre(

b-id **int** not null,

genre **varchar**(20) not null,

foreign key (b-id) **references** Books(b-id)

primary key (b-id, genre)

);

2.14 Rate

Relational Model

Rate(b-id, user-id, rating)

FK: b-id references Books

FK: user-id references Users

Functional Dependencies

b-id, user-id→ rating

Candidate Keys

{(b-id, user-id)}

Normal Form

3NF

Table Definition

create table Rate(

b-id **int** not null,

user-id **int** not null,

rating **int** not null,

foreign key (b-id) **references** Books(b-id)

foreign key (user-id) **references** Users(user-id)

primary key (b-id, user-id)

);

2.15 Lists

Relational Model

Lists(list-name,user-id)

FK: user-id references Users(user-id)

Functional Dependencies

None

Candidate Keys

{(list-name, user-id)}

Normal Form

3NF

Table Definition

create table Lists(

list-name **varchar**(20) not null,

user-id **int** not null,

primary key (list-name, user-id),

foreign key(user-id) **references** Users(user-id) **on delete cascade**

);

2.16 list-includes

Relational Model

list-includes(list-name, user-id, b-id)

FK: list-name, user-id references Lists(list-name, user-id)

FK: b-id references Books(b-id)

Functional Dependencies

None

Candidate Keys

{(list-name, user-id, b-id)}

Normal Form

3NF

Table Definition

create table list-includes(

list-name **int** not null,

user-id **int** not null,

b-id **int** not null,

primary key (list-name, user-id, b-id),

foreign key(b-id) **references** Books(b-id),

foreign key(user-id, list-name) **references** Lists(user-id, list-name)

);

2.17 comments

Relational Model

comments(<u>comment-id</u>, comment, date)

Functional Dependencies

comment-id → comment, date

Candidate Keys

{(comment-id)}

Normal Form

3NF

Table Definition

create table comments(

comment-id **int** not null auto_increment,

comment **varchar**(500) not null,

date **date** not null**,**

primary key (comment-id)

);

2.18 comment-on

Relational Model

comment-on(b-id, comment-id, user-id)

FK: b-id references Books(b-id)

FK: comment-id references comments(comment-id)

FK: user-id references Users(user-id)

Functional Dependencies

b-id, comment-id → user-id

Candidate Keys

{(b-id, comment-id)}

Normal Form

3NF

Table Definition

create table comment-on(

b-id **int** not null,

comment-id **int** not null,

user-id **int** not null,

primary key (b-id, comment-id),

foreign key(b-id) **references** Books(b-id),

foreign key(comment-id) **references** comments(comment-id),

foreign key(user-id) **references** Users(user-id)

);

2.19 Like

Relational Model

Like(user-id, comment-id)

FK: user-id references Users(user-id)

FK: comment-id references comments(comment-id)

Functional Dependencies

None

Candidate Keys

{(user-id, comment-id)}

Normal Form

3NF

Table Definition

create table Like(

user-id **int** not null,

comment-id **int** not null,

primary key (user-id, comment-id),

foreign key(user-id) **references** Users(user-id),

foreign key(comment-id) **references** comments(comment-id)

);

2.20 recommends

Relational Model

recommends(user-id, friend-id, b-id)

FK: user-id references Users

FK: friend-id references Users(user-id)

FK: b-id references Books(b-id)

Functional Dependencies

Candidate Keys

{(user-id, friend-id, b-id)}

Normal Form

3NF

Table Definition

create table recommends(

user-id **int** not null,

friend-id **int** not null,

b-id **int** not null,

primary key (user-id, friend-id, b-id),

foreign key(user-id) **references** Users(user-id),

foreign key(friend-id) **references** Users(user-id),

foreign key(b-id) **references** Books(b-id),

);

2.21 Writes

Relational Model

Writes(author-id, b-id)

FK: author-id references Authors(author-id)

FK: b-id references Books(b-id)

Functional Dependencies

None

Candidate Keys

{(author-id, b-id)}

Normal Form

3NF

Table Definition

create table Writes(

author-id **int** not null,

b-id **int** not null,

primary key (author-id, b-id),

foreign key(author-id) **references** Authors(author-id),

foreign key(b-id) **references** Books(b-id),

);

2.22 Reply

Relational Model

Reply(reply-id,user-id, comment-id)

FK: reply-id references comments(comment-id)

FK: comment-id references comments(comment-id)

FK: user-id references Users(user-id)

Functional Dependencies

None

Candidate Keys

{(reply-id, comment-id, user-id)}

Normal Form

3NF

Table Definition

```
create table Reply(
        reply-id            int not null,
        comment-id          int not null,
        user-id             int not null,
        primary key (reply-id, user-id),
        foreign key(reply-id) references comments(comment-id),
        foreign key(comment-id) references comments(comment-id),
        foreign key(user-id) references Users(user-id),
);
```

2.23 Editions

Relational Model

Editions(b-id, edition-no, page-count, language, translator-name)

FK: b-id references Books(b-id)

Functional Dependencies

b-id,edition-no → page-count, language, translator-name

Candidate Keys

{(b-id,edition-no)}

Normal Form

3NF

Table Definition

```
create table Editions(
        b-id                int not null,
        edition-no          int not null,
        page-count          int not null,
        language,           varchar(10) not null,
        translator-name,    varchar(15),
        primary key (b-id, edition-no),
        foreign key(b-id) references Books(b-id) delete on cascade
);
```

2.25 Publishers

Relational Model

Publishers(p-id, p-name, city)

Functional Dependencies

p-id → p-name, city

Candidate Keys

{(p-id)}

Normal Form

3NF

Table Definition

create table Publishers(

p-id **int** not null auto_increment,

p-name **varchar**(30) not null,

city **varchar**(30) not null,

primary key (p-id)

);

2.26 Publishes

Relational Model

Publishes(p-id, b-id, edition-no, publish-year)

FK: p-id references Publishers

FK: {b-id, edition-no} references Editions

Functional Dependencies

p-id, b-id, edition-no → publish-year

Candidate Keys

{(p-id, b-id, edition-no)}

Normal Form

3NF

Table Definition

create table Publishes(

p-id **int** not null,

b-id **int** not null,

edition-no **int** not null,

publish-year **YEAR** not null,

primary key (p-id, b-id, edition-no)

foreign key (p-id) **references** Publishers(p-id)

foreign key (b-id, edition-no) **references** Editions(b-id, edition-no)

);

2.27 progress

Relational Model

progress(user-id,e-id, page-no)

FK: e-id references Editions(e-id)

FK: user-id references Users(user-id)

Functional Dependencies

user-id,e-id → page-no

Candidate Keys

{(user-id,e-id)}

Normal Form

3NF

Table Definition

create table progress(

user-id **int** not null,

e-id **int** not null,

page-no **int** not null,

primary key (user-id, e-id),

foreign key(user-id) **references** Users(user-id),

foreign key(e-id) **references** Editions(e-id),

);

2.28 Series

Relational Model

Series(s-name)

Functional Dependencies

None

Candidate Keys

{(s-name)}

Normal Form

3NF

Table Definition

create table Series(

s-name **varchar(30)** not null,

primary key (s-name)

);

2.29 Bookshelves

Relational Model

Bookshelves(bs-name, user-id)

FK: user-id references Users(user-id)

Functional Dependencies

Candidate Keys

{(bs-name, user-id)}

Normal Form

3NF

Table Definition

create table Bookshelves(

bs-name **varchar**(20) not null,

user-id **int** not null,

primary key (bs-name, user-id),

foreign key(user-id) **references** Users(user-id) **on delete cascade**

);

2.30 bookshelf-includes

Relational Model

bookshelf-includes(bs-name, user-id, b-id)

FK: bs-name, user-id references Bookshelves(bs-name, user-id)

FK: b-id references Books(b-id)

Functional Dependencies

None

Candidate Keys

{(bs-name, user-id, b-id)}

Normal Form

3NF

Table Definition

create table bookshelf-includes(

bs-name **int** not null,

user-id **int** not null,

b-id **int** not null,

primary key (bs-, b-id),

foreign key(b-id) **references** Books(b-id),

foreign key(user-id, bs-name) **references** Bookshelves(user-id, bs-name)

);

2.31 part-of-series

Relational Model

part-of-series(s-name, b-id)

FK: s-name references Series(s-name)

FK: b-id references Books(b-id)

Functional Dependencies

None

Candidate Keys

{(b-id, s-name)}

Normal Form

3NF

Table Definition

create table part-of-series(

s-name **varchar**(20) not null,

b-id **int** not null,

primary key (list-id, b-id),

foreign key(s-name) **references** Series(s-name),

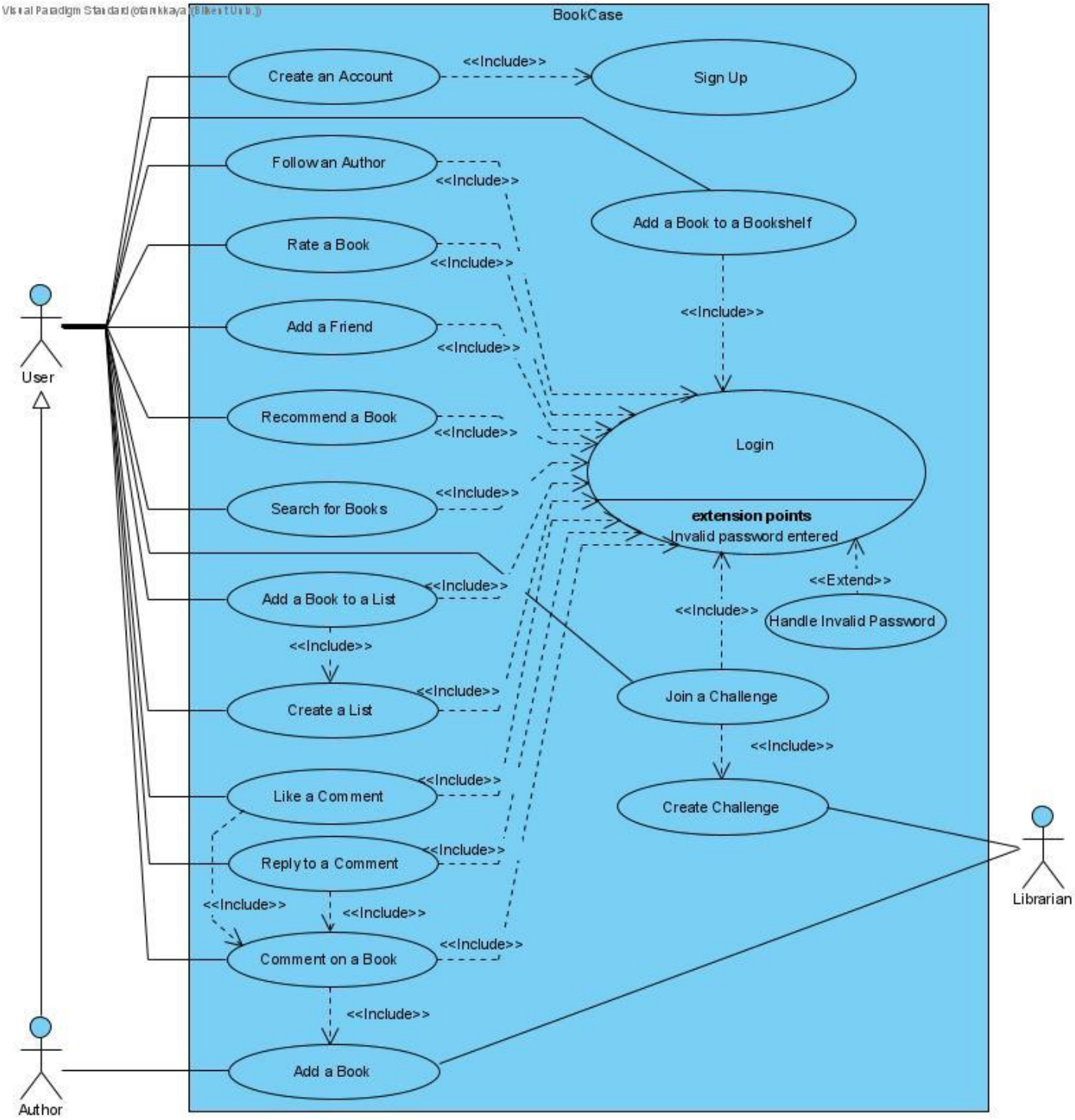
foreign key(b-id) **references** Books(b-id)

);

3. Functional Components

We discuss the functional components of our project in this section.

3.1 Use-Case Model



3.2 Use Cases

Sign Up: Any user can sign in to the system with an email address, a username, and a password. In order to create an account signing in is required.

Create an Account: A user will be able to create an account that binds to the previously given email. This account will consist of some information related to the user. Moreover, user's book lists, following authors, friends list, and bookshelves will be available through the account page.

Login: Any user can log in to an existing account with a corresponding username and required password. Invalid password entry will be handled in an extension point. Log in is required for the following use cases: add a book to a bookshelf, follow an author, rate a book, add a friend, recommend a book, search for books, join a challenge, add a book to a list, create a list, like a comment, reply to a comment, comment on a book.

Follow an Author: Following an author is an option to display the author in the user's profile. In order to follow an author, logging in is required.

Rate a Book: Users will be able to rate a book on a scale from 5 to 0. In order to rate a book, logging in is required.

Add a Friend: A user will be able to add any user as a friend. All the users' usernames that are added by the user as a friend will be displayed on the account page of that user. In order to add a friend, logging in is required.

Recommend a Book: A user can recommend a book from the books page to a friend of him/her. This recommendation will include the name of the book. In order to recommend a book, logging in is required.

Search for Books: A search motor bar will be used to enable users to search through the database for a book via a typed keyword. In order to search for books, logging in is required.

Join a Challenge: The user can join any existing reading challenge. In order to compete in a challenge, the user must log in and there needs to be an existing challenge.

Create Challenge: Any librarian can create a challenge with a description and a deadline date.

Create a List: Any user can create a list of a book for any purpose and any given name. In order to create a list, logging in is required.

Add a Book to a List: The user can add any book to any list s/he created. For adding a book to a list, creating a list and logging in is required.

Comment on a Book: The user can comment on an existing book. In order to comment on a book, the user should log in to the system and there has to be an existing book that is added previously in the system. Eventually, add a book, and login is required.

Like a Comment: The user can like any comment on the system. The like count will increment after the action. In order to like a comment, the user should be logged in and there needs to be a comment on a book.

Reply to a Comment: The user can reply to any comment on the system. In order to reply to a comment, the user should be logged in to the system and there has to be a comment in the system to reply.

Add a Book: Any author, which is a specialized user, can add a book of his/her own to the system. A librarian should confirm the book in order for the book to be added.

3.3 Algorithms

3.3.1 Most Popular Challenges Algorithm

For finding the top 5 most popular challenges right now, a sorting algorithm will be used. Here, the sorting algorithm that is used in the SQL keyword **order by** is used. Lastly, the top 5 rows will be selected from the resulting table.

3.3.2 Book Rating Algorithm

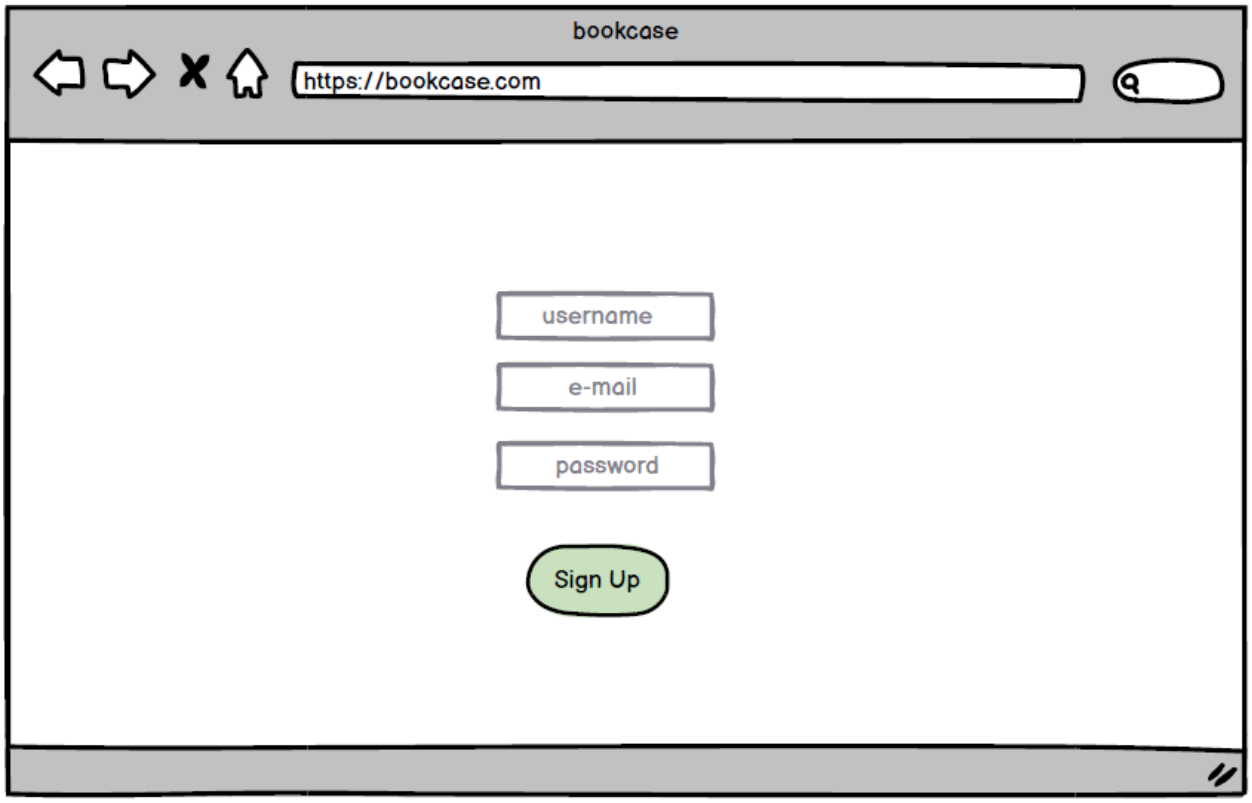
Book rating should be calculated immediately after each new rating or rating change. The algorithm for calculating the rating of a book is calculated from the individual rates that are stored from the users. For each rating of a book, the sum of all ratings is calculated and divided by the number of ratings.

3.4 Data Structures

In the relation schemas we used the SQL data types **int** for numeric values like unique id of rows or ratings; **varchar** and **char** for string values like names; **date** and **year** for storing the dates.

4. User Interface Design

4.1 Sign-up Page

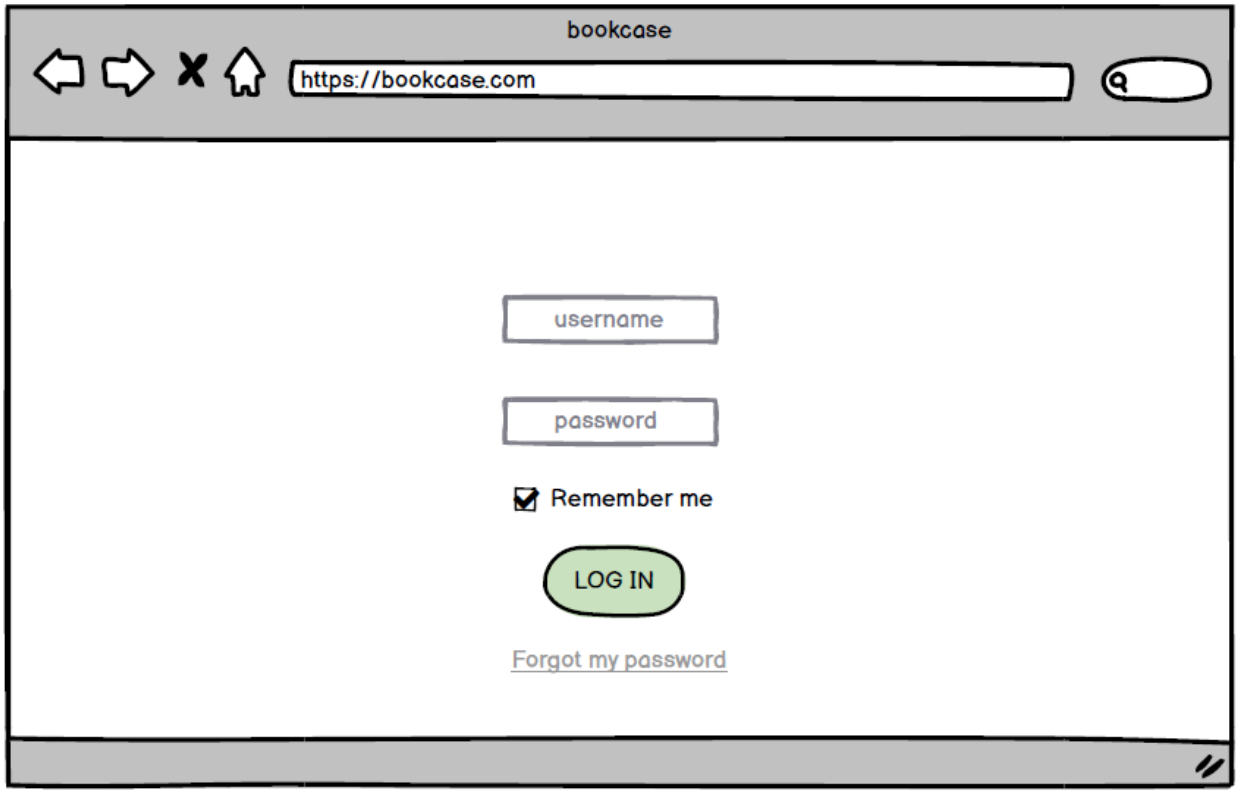


➔ **Signing up:**

insert into Accounts(0, @username, @email, @password);

insert into Users(account-id, null, curdate());

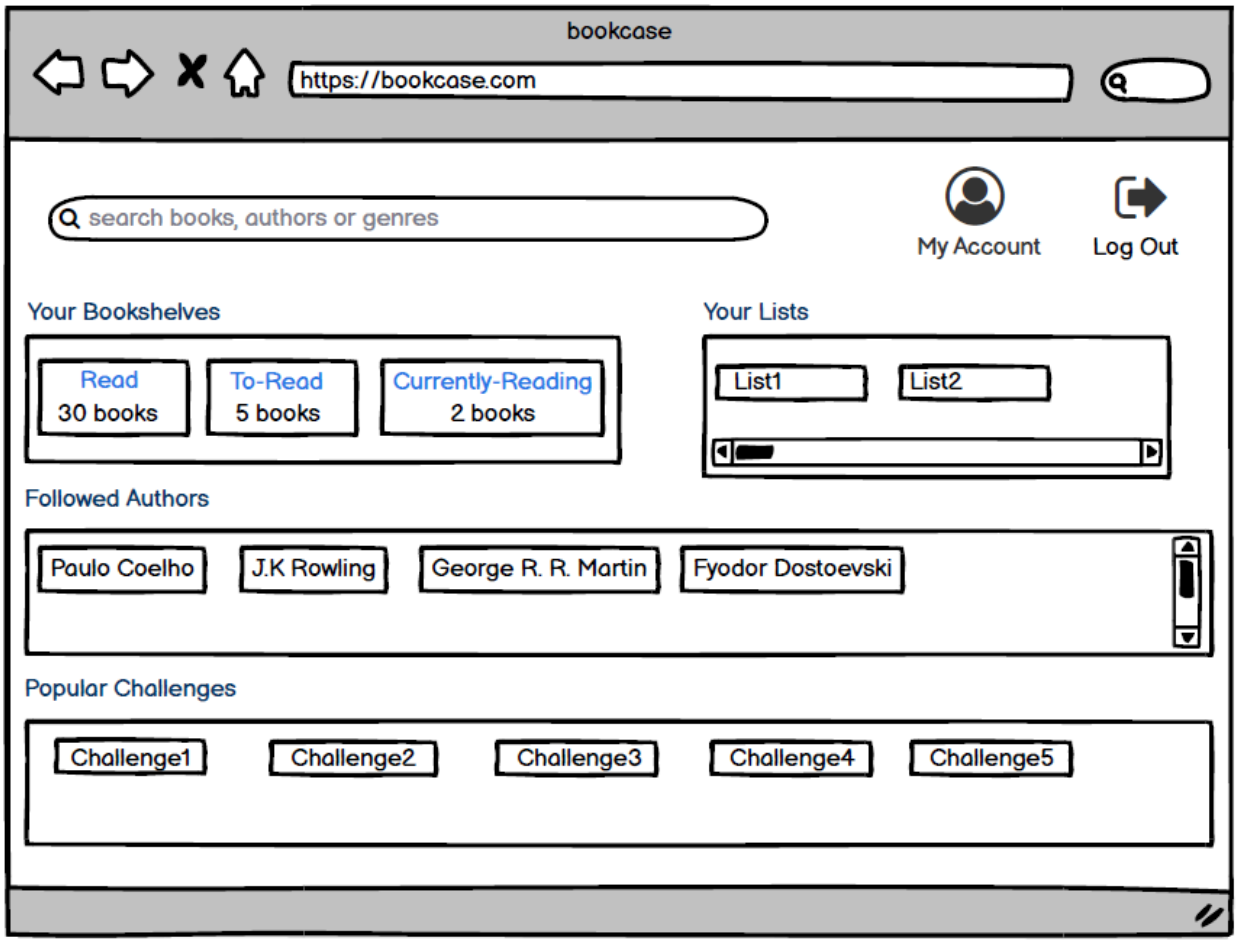
4.2 Login Page



➔ **Login:**

select * **from** Accounts **where** username = @username **and** password = @password

4.3 Home Page



➔ **Search books:**

select *

from Books

where book-name **like** '%' + @search + '%'

➔ **Search authors:**

select *

from Authors

where author-name **like** '%' + @search + '%'

➔ **Search genre:**

select *

from author-genre, book-genre

where author-genre.genre **like** ‘%’ + @search + ’%’ **or** book-genre.genre **like** ‘%’ + @search + ‘%’

➔ **Display bookshelves:**

select (bs-name, count(*))

from Bookshelves, bookshelf-include

where user-id = @user-id

group by bs-name;

➔ **Display lists:**

select list-name

from Lists

where user-id = @user-id;

➔ **Display authors:**

select A.author-name

from Authors **as** A, follows **as** F

where F.user-id = @user-id **and** F.author-id = A.author-id;

➔ **Display the top 5 most popular challenges:**

select challenge-name

from (**select** challenge-name, count(*) **as** cnt

from Challenges **natural join** join

where due-date > curdate()

group by challenge-name

order by cnt

limit 5);

➔ **Display account:**

select * **from** Accounts, Users **where** user-id = account-id and user-id = @user-id;

4.4 Book Page

➔ **Display Book and Edition information:**

select *

from Books **natural join** Editions **natural join** Publishes **natural join** Publishers

where @b-id = b-id **and** @edition-no = edition-no;

➔ **Display rating:**

```
select avg(rating)
from Rate
where @b-id = b-id;
```

➔ **Display Genres:**

```
select genre
from Book-genre
where b-id = @b-id;
```

➔ **Display comments:**

```
with like-count(comment-id, cnt) as (select comment-id, count(*)
                                     from comments natural join Like
                                     group by comment-id)
select comment, cnt
from comments natural join comment-on natural join like-count
where @b-id = b-id;
```

➔ **Rate a book:**

```
insert into Rate values (@b-id, @user-id, @rating);
```

➔ **Recommend book:**

```
insert into recommends values (@user-id, @friend-id, @b-id);
```

➔ **Make a comment:**

```
insert into comments values (0, @comment, curdate());
insert into comment-on values (@b-id, @comment-id, @user-id);
```

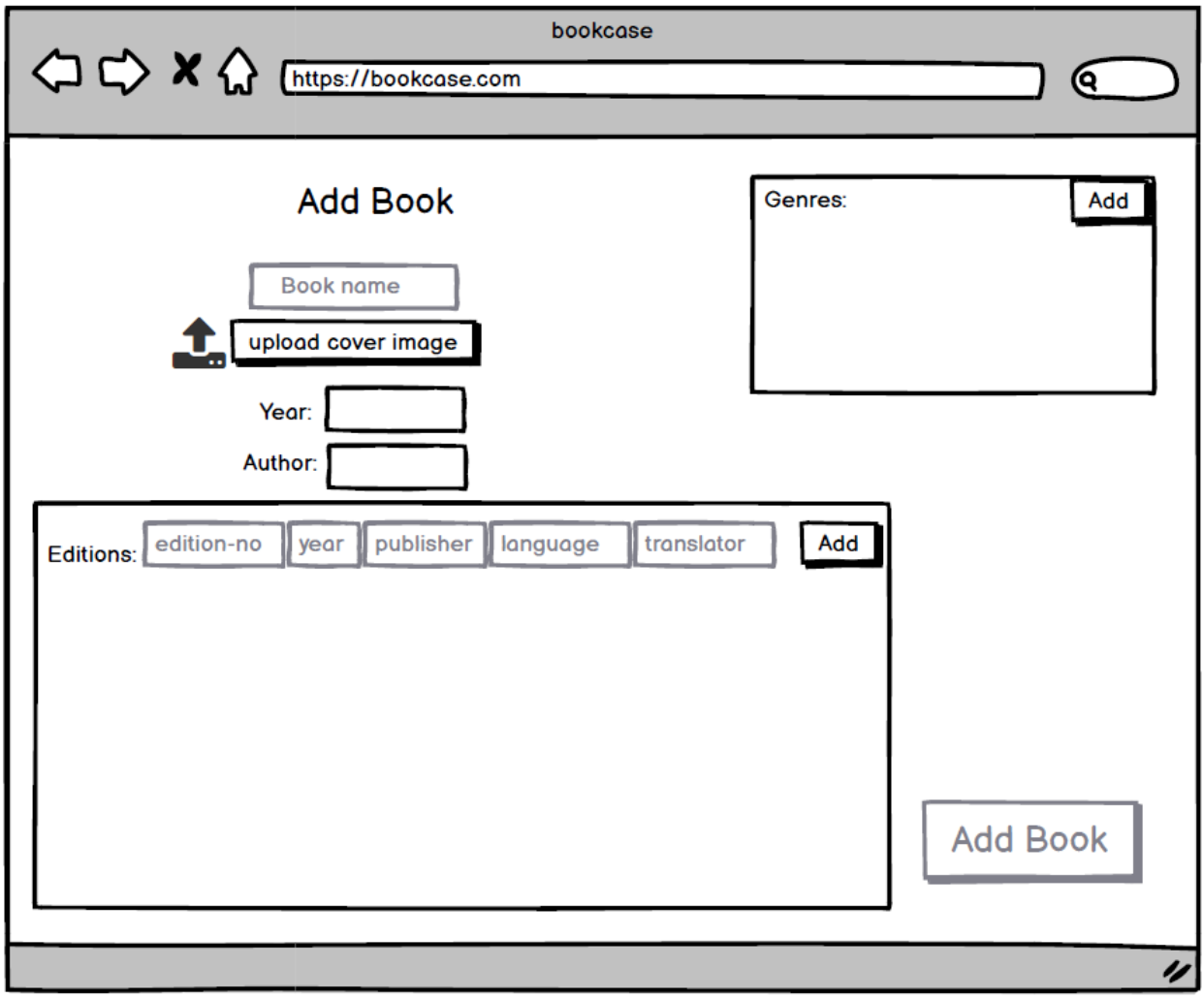
➔ **Reply to a comment:**

```
insert into comments values (0, @comment, curdate());
insert into Reply values(@reply-id, @user-id, @comment-id);
```

➔ **Like a comment:**

```
insert into Like values(@user-id, @comment-id);
```

4.5 Add Book Page



➔ **Add Book:**

insert into Books **values**(0, @cover, @title, @year);

insert into Writes **values**(@author-id, @b-id);

➔ **Add edition:**

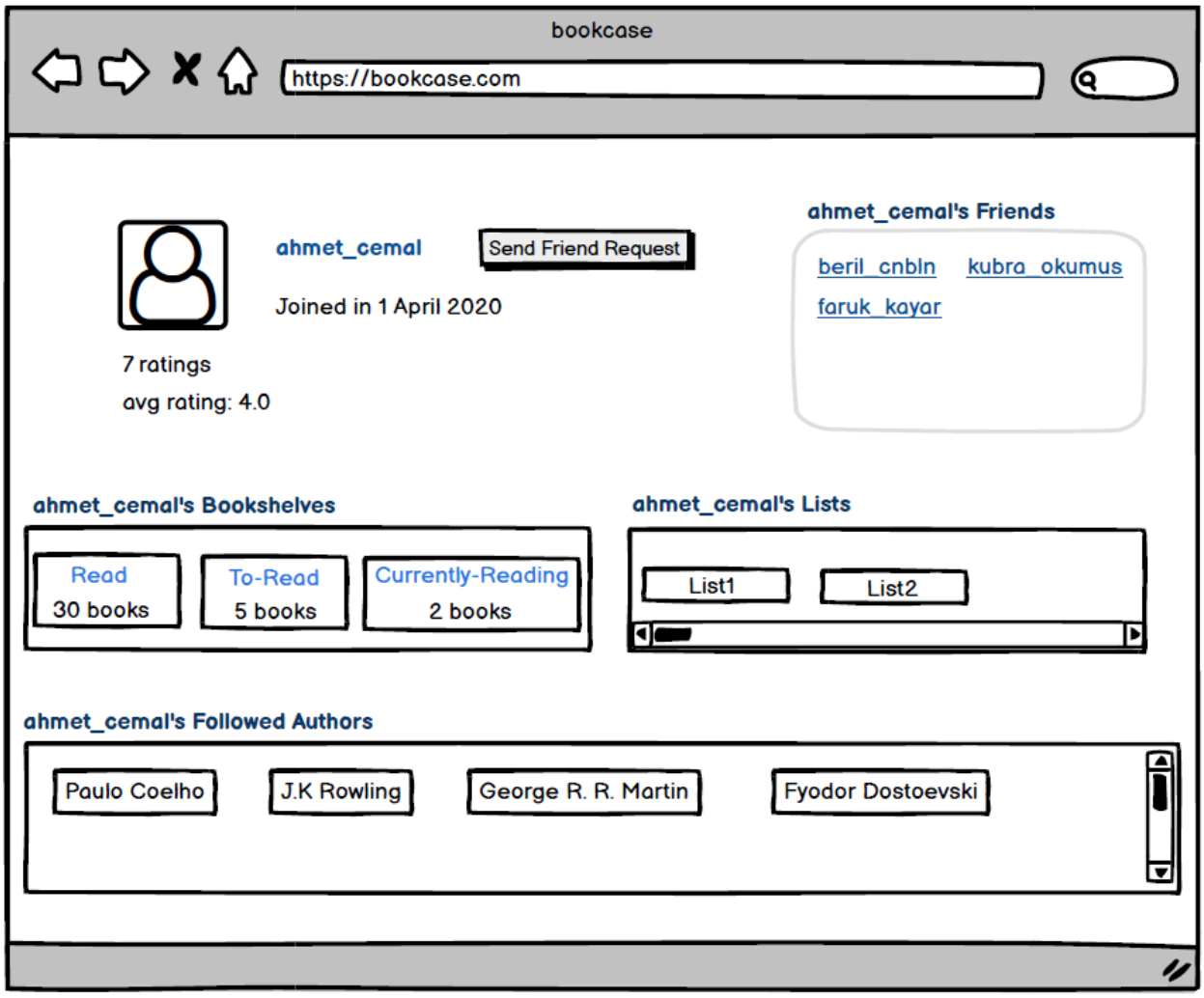
insert into Editions **values**(@b-id, @edition-no, @page-count, @language, @translator-name);

insert into Publishes **values**(@p-id, @edition-no, @publish-year);

➔ **Add genre:**

insert into Book-genre **values**(@b-id, @genre);

4.6 User Profile Page



➔ **Display Rating Information**

select count(*) **as** no-of-ratings, avg(rating) **as** avg-ratings

from Rate

where @user-id = user-id;

➔ **Add friend:**

insert into Friends **values**(@user-id, @friend-id);

➔ **Display Friend List**

```
select Accounts.username
from Friends, Accounts
where (Friends.user-id = @user-id and Accounts.account-id = Friends.friend-id)
      or (Friends.friend-id = @user-id and Accounts.account-id = Friends.user-id)
```

➔ **Display Bookshelves**

select (bs-name, count(*))

from Bookshelves, bookshelf-include

where user-id = @user-id

group by bs-name;

➔ **Display Lists**

select list-name

from Lists

where user-id = @user-id;

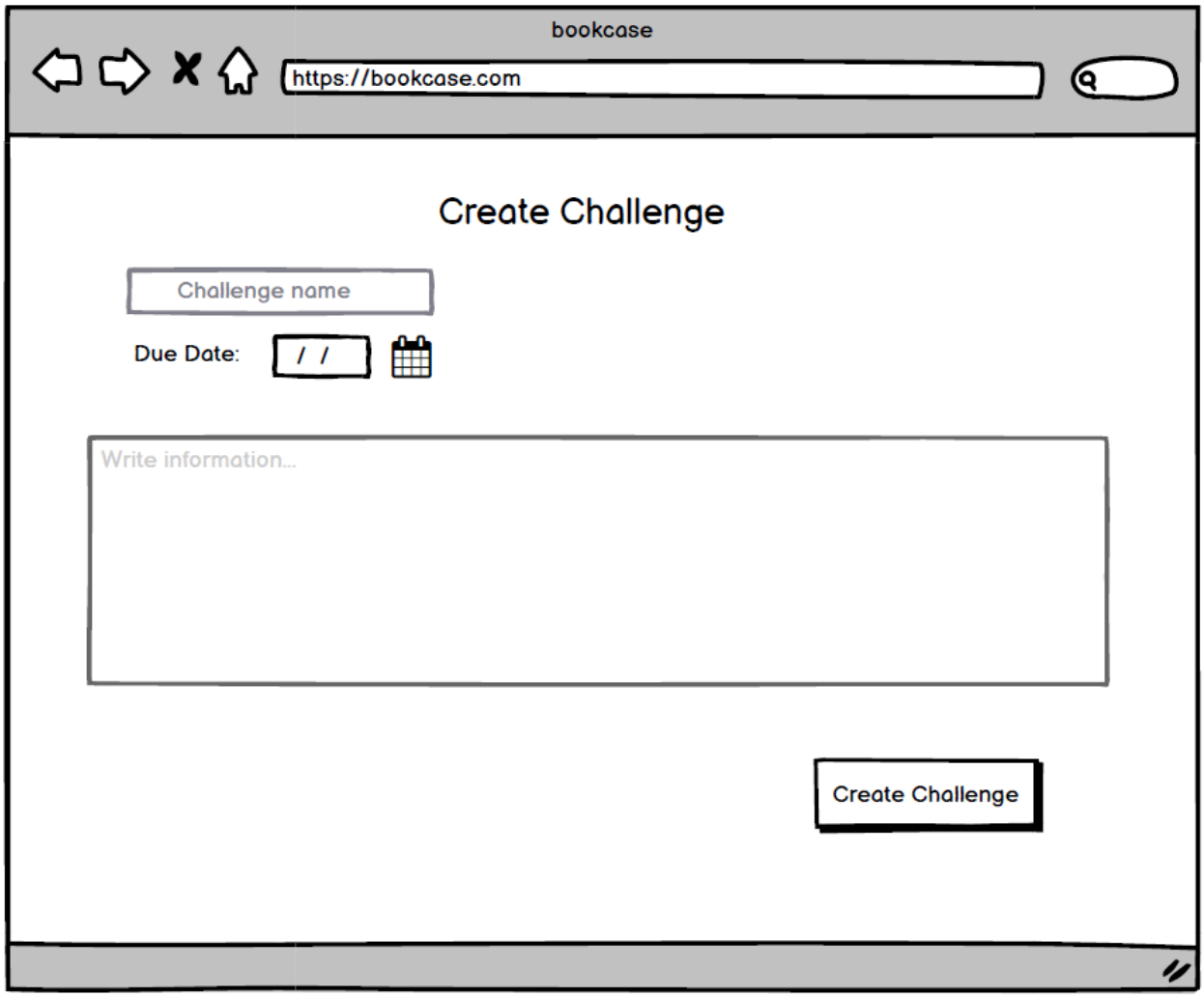
➔ **Display Authors**

select A.author-name

from Authors **as** A, follows **as** F

where F.user-id = @user-id **and** F.author-id = A.author-id;

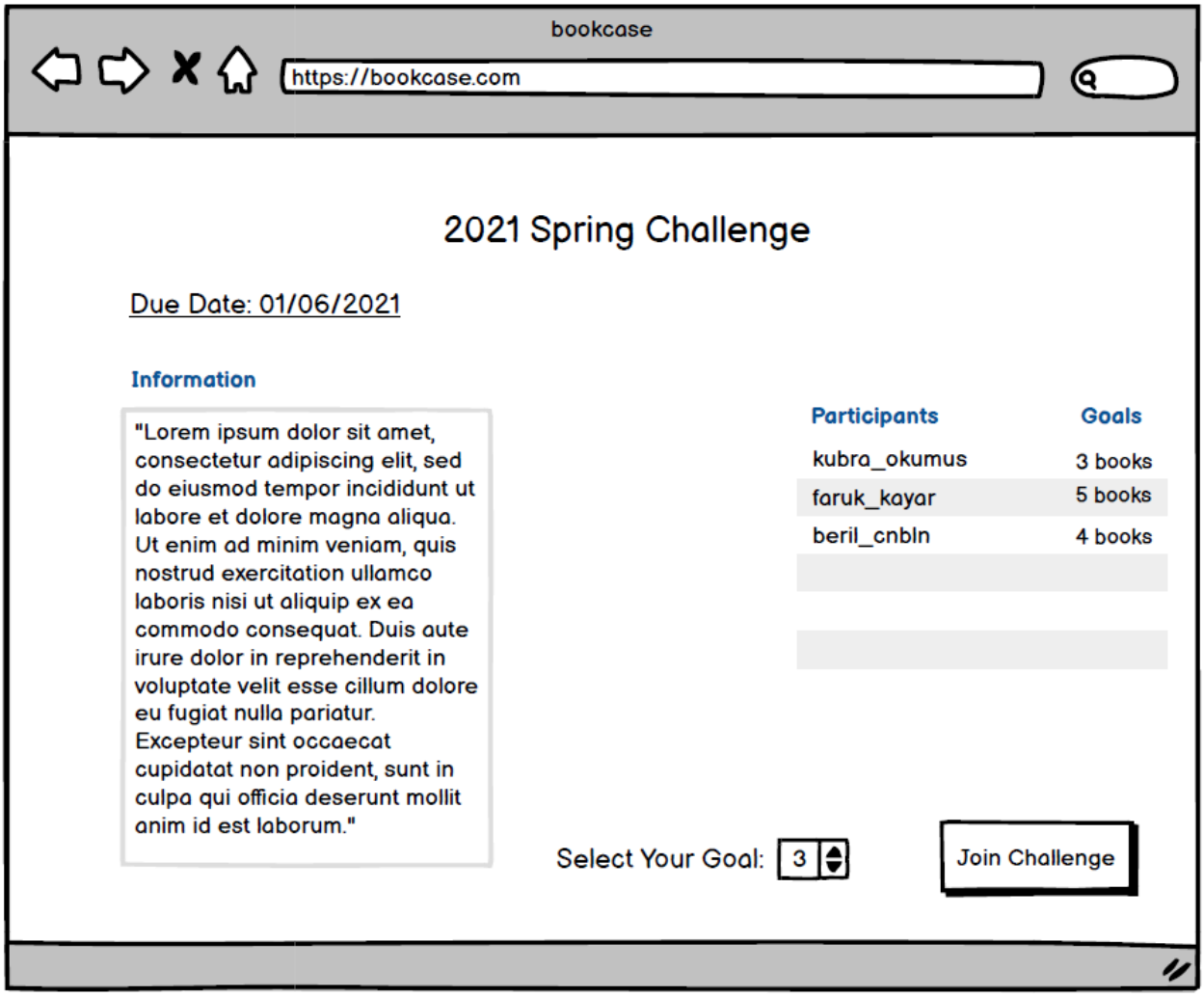
4.7 Create Challenge Page



➔ **Create Challenge:**

insert into Challenges **values**(@challange-name, @librarian-id, @info, @due-date);

4.8 Challenge Page



➔ **Display Challenge information:**

select info
from Challenges
where challange-name = @challenge-name

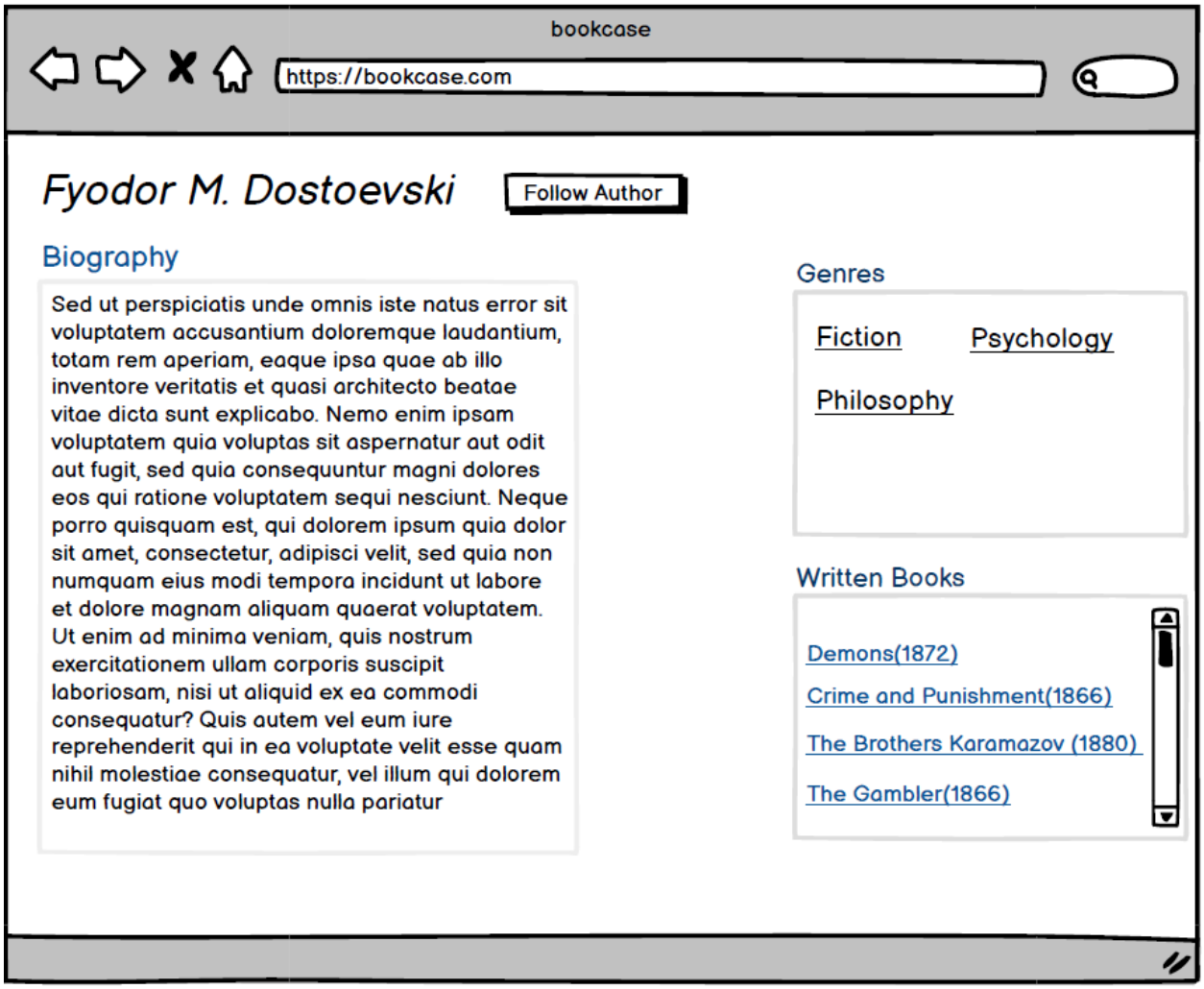
➔ **Display Participants**

select Accounts.username **as** Participants, join.goal **as** Goals
from join, Accounts
where join.user-id = Accounts.account-id and join.challange-name = @challenge-name

➔ **Join Challenge**

insert into join **values**(@challange-name, @user-id, @goal);

4.9 Author Page



➔ **Display Author Information:**

select author-name, biography

from Authors

where author-id = @author-id;

➔ **Display Genres:**

select genre

from Author-genre

where author-id = @author-id;

➔ **Display Books:**

select Books.title, Books.year

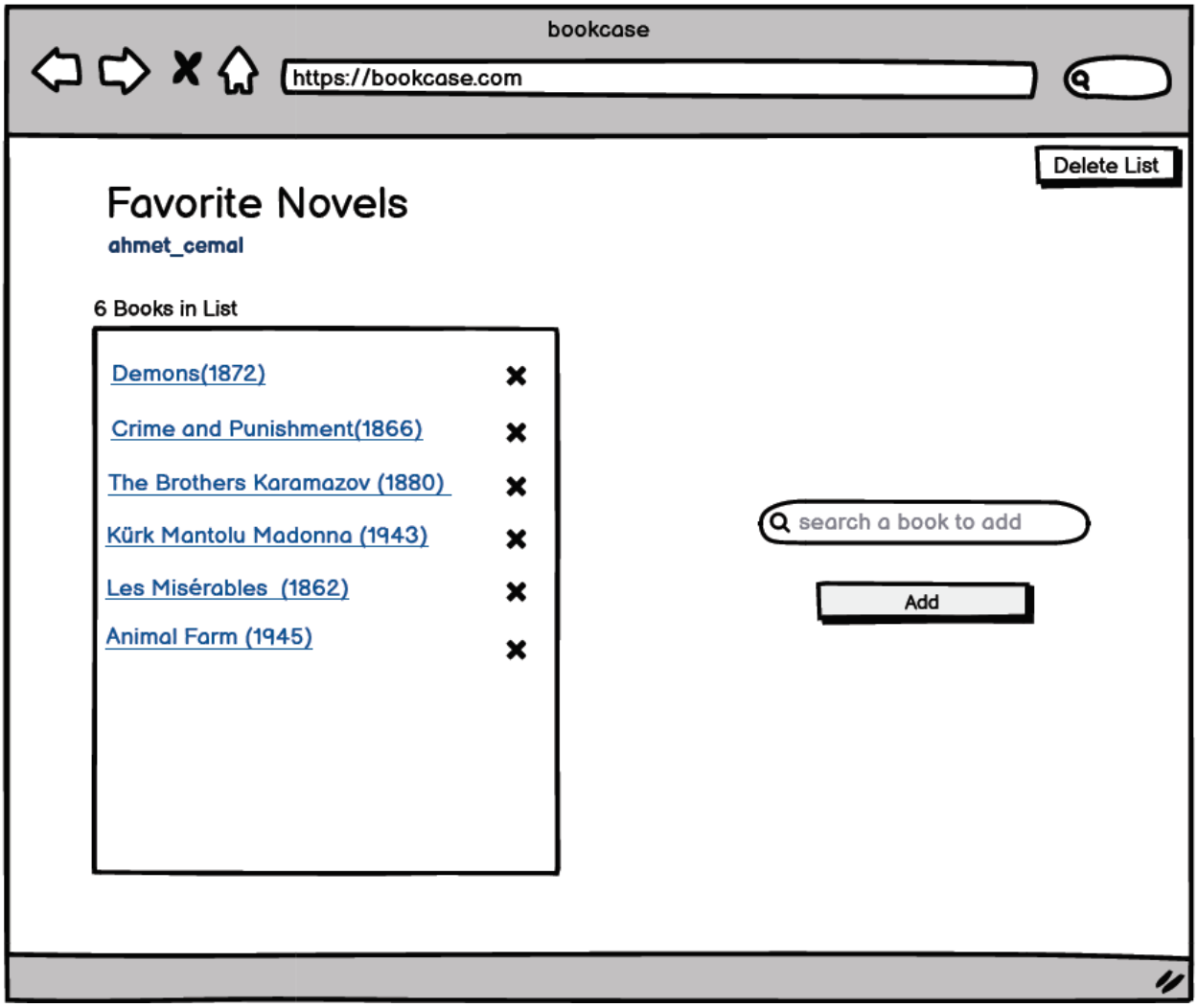
from Books **natural join** Writes

where Writes.author-id = @author-id;

➔ **Follow Author:**

insert into follows **values**(@user-id, @author-id);

4.10 List Page



➔ **Display List Information:**

select list-name

from Lists

where list-name= @list-name **and** user-id = @user-id

➔ **Display Books:**

select Books.title, Books.year

from Books **natural join** list-includes

where list-includes.list-name = @list-name **and** list-includes.user-id = @user-id

➔ **Remove Book From List:**

```
delete from list-includes
where b-id = @b-id and list-name = @list-name and user-id = @user-id;
```

➔ **Delete List:**

```
delete from Lists
where user-id = @user-id and list-name = @list-name;
```

➔ **Add a book:**

```
insert into list-includes values(@list-id, @user-id, @b-id);
```

5. Advanced Database Components

5.1 Views

5.1.1 Most Popular Challenges View

```
create view most_popular_challenges as
        select challenge-name
        from ( select challenge-name, count(*) as cnt
                from Challenges natural join join
                where due-date > curdate()
                group by challenge-name
                order by cnt
                limit 5);
```

5.1.2 Book Comments View

```
with like-count(comment-id, cnt) as (select comment-id, count(*)
                                     from comments natural join Like
                                     group by comment-id)
create view book_comments as
        select comment, cnt
```

from comments **natural join** comment-on **natural join** like-count

where @b-id = b-id;

5.1.3 Book Rating View

create view book_rating **as**

select avg(rating)

from Rate

where @b-id = b-id;

5.1.4 User Rating Information View

create view user-rating-info **as**

select count(*) **as** no-of-ratings, avg(rating) **as** avg-ratings

from Rate

where @user-id = user-id;

5.1.5 Friend List View

create view friends_list **as**

select Accounts.username

from Friends, Accounts

where Friends.user-id = @user-id **and** Accounts.account-id = Friends.friend-id;

5.1.6 Challenge Participant List View

create view challenge_participants **as**

select username, goal

from join, Accounts

where join.user-id = Accounts.account-id **and** join.challenge-name = @challenge-name;

5.1.7 Author Book List View

create view author_books **as**

select Books.title, Books.year

from Writes **natural join** Books

where @author-id = author-id;

5.2 Reports

5.2.1 Total Number of Participants in a Challenge

select count(*)

from join

where @challenge_name = challenge_name;

5.2.2 Total Number of Books in a List

select count(*)

from list-include

where @list-name = list-name **and** @user-id = user-id;

5.3 Triggers

➔ **Create Bookshelves for a new User**

When a new user is created, three new bookshelves of that user will be created automatically through a trigger. The names of the bookshelves are “read”, “to-read” and “currently-reading.”

➔ **Giving a Rating or Changing a Rating to a Book**

When a user rates a book, the average rating is recalculated and updated accordingly.

➔ **Liking a Comment or a Reply**

When a comment or reply is liked, the like count of that reply is updated.

➔ **Joining a Challenge**

When a new user joins a challenge the top 5 most popular challenges in the home page will be updated accordingly.

5.4 Constraints

- ➔ A user must be logged-in to use the website
- ➔ Librarians cannot create challenges with the same name.
- ➔ Users can only give one rating to a book.
- ➔ Users cannot join a challenge after its due date.
- ➔ A book can be in at most one bookshelf of a user.
- ➔ A book cannot be a part of in two different series
- ➔ Users cannot create lists with the same name

5.5 Stored Procedures

- ➔ Since user's lists, bookshelves and followed authors are displayed both in the home page and in a user's profile, this procedure can be a stored procedure.
- ➔ Also, we plan on using stored procedures to implement searching for multiple objects like books, authors, genres and users.

6. Implementation Details

For the implementation of this project, we are planning to use PHP for back-end development; HTML, CSS and Javascript for front-end development. For the database, we are going to use a MySQL server for database management and InnoDB as the database engine.

7. Website and References

Project website:

https://cemalahmet.github.io/book-catalog-project-cs-353/

[1] O. Chandler, "About Goodreads," *Goodreads*, 2007. [Online]. Available: https://www.goodreads.com/about/us. [Accessed: 24-Feb-2021].