Library Management System Database Project

Julia Chen - CSPB 3287

Project Description

This project will create a simple library management database system which includes basic functions needed by a library user. The project will focus on the following functions:

1. Search the library for a book by title or author
2. Check out a book from the library
3. Display currently checked out books for a user
4. Allow users to give books a rating
5. Display related books when returning a book
6. Report on "Top Books" of the library
7. View previously borrowed books for users
8. Allow user to delete account

Additional details:

- Database will be populated with an initial store of various books and tags. The csv files were downloaded from a Github project, goodbooks-10k (https://github.com/zygmuntz/goodbooks-10k/tree/master), where the data was originally sourced from GoodReads. I have cleaned and reduced the data to a more manageable size for this project.
- There will only be one copy of each book available to borrow.
- Database will be populated with an initial list of users.
- Users are allowed to check out multiple books if their account is in good standing.
- On return of a book, if late, the user account will be updated to only allow 1 book out at a time until user account is in better standing. Better standing may be earned after 5 non-late borrowed books.
- On return, user may give a rating of the book 1-5.
- Search function will use a simple string search on title and author attributes.
- Related books will use the current books tags to search for books with similar tags.
- When user deletes their account the system will verify they have no books currently checked out before deleting.

Project Video Walkthrough

Link to YouTube Video: https://youtu.be/8OmMKeU2d1A

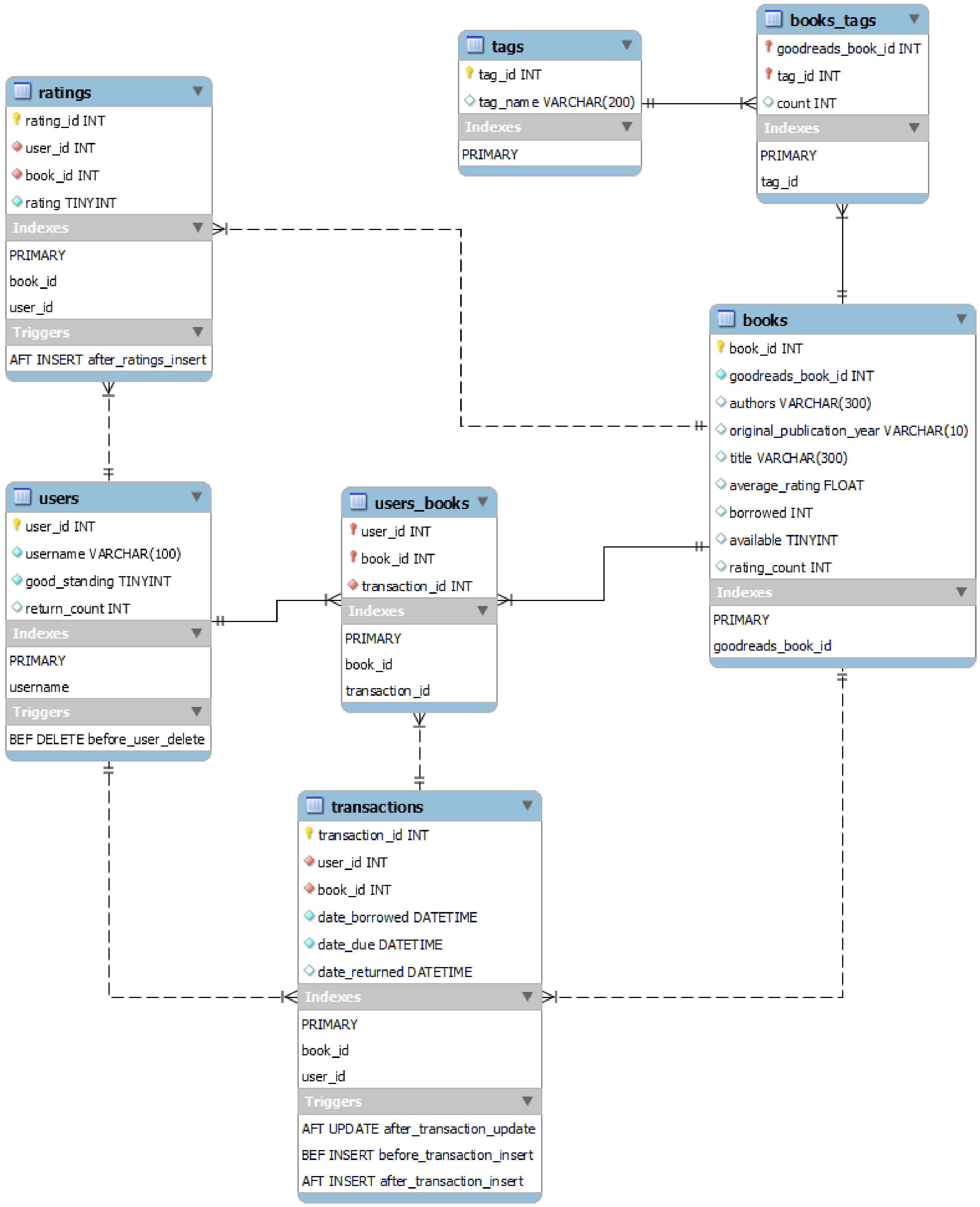
Project GitHub Link

GitHub Repository: https://github.com/juliahchen/library_db

Design

Data sourced from the goodbooks-10k project will populate the books table with 9,800 rows, the tags table with approximately 32,000 rows, and the books_tags table with approximately 117,000 rows. The users table will be populated with 5 users for testing. The ratings and transactions tables will start empty and be populated by the simulated user interactions. Each book that is checked out will have its own transaction, where the information about date borrowed, date due, and returned date can be found. There is a one to many relationship between users and transactions, a one to many relationship between books to transactions, and a one to many relationship between books and ratings. There are two junction tables, users_books and books_tags, as users and books have a many to many relationship, and books and tags also have a many to many relationship.

E/R Diagram



Tools

A locally hosted MySQL database was used. MySQL is a logical choice as we have covered the database extensively in our class. The Python programming language was used in a Jupyter notebook for database creation, population, and display of logic and user actions. SQLAlchemy was used to connect with the remote database as well as SQL magic for queries in the Jupyter notebook.

Learning Outcomes

E/R Diagrams: Practice designing E/R diagrams as part of the project design.

Triggers: Multiple triggers were used in this project. In order to determine if the triggers have worked properly I will display before/after table information to show updates that have taken place, for example if the user is in good standing after recent late return.

Other concepts covered in the course which will be demonstrated:

- Indexes and constraints.
- Common table expressions (CTE).
- Combine and aggregate data from multiple tables. The "Top Books" report will join, group, and aggregate data from 3 tables (books, tags, books_tags).
- Use LIKE operator to search book titles and authors.

Load Libraries and Connect to Database

MySQL Database is hosted locally on my personal computer. An SQL Alchemy connection will be created and SQL magic will be available for queries.

In [ ]:
```
import os
import configparser
from sqlalchemy import create_engine
import pandas as pd
```

In [ ]:
```
mysqlcfg = configparser.ConfigParser()
mysqlcfg.read("./mysql.cfg")
user, passwd = mysqlcfg['mysql']['user'], mysqlcfg['mysql']['passwd']
dburl = f"mysql+pymysql://{user}:{passwd}@localhost:3306/cspb_library"

os.environ['DATABASE_URL'] = dburl  # define this env. var for sqlmagic
eng = create_engine(dburl)
con = eng.connect()

%reload_ext sql
print ("get version...")
%sql SELECT version()
```

```
get version...
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

version()
8.0.34

Create Tables with Indexes

Seven tables will be created.

Composite primary key indexes are used in junction tables:

- books_tags, users_books

Foreign key indexes are in the following tables:

- ratings, transactions, books_tags, users_books

Unique key indexes are in the following tables:

- users, books

Additional Constraints:

- NOT NULL
- CHECK(rating > 0 and rating < 6)

In [ ]:

```
%%sql
# Drop tables if already in database
drop table if exists ratings;
drop table if exists users_books;
drop table if exists transactions;
drop table if exists users;
drop table if exists books_tags;
drop table if exists books;
drop table if exists tags;

# Users Table
# Primary Key / Index: user_id
# Unique Key Index: username
CREATE TABLE users (
  user_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  username VARCHAR(100) NOT NULL,
  good_standing TINYINT NOT NULL DEFAULT '1',
  return_count INT UNSIGNED NULL DEFAULT '0',
  PRIMARY KEY (user_id),
  UNIQUE KEY (username)
);

# Books Table
# Primary Key / Index: book_id
# Unique Key Index: goodreads_book_id
CREATE TABLE books (
  book_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  goodreads_book_id INT UNSIGNED NOT NULL,
  authors VARCHAR(300) NULL DEFAULT NULL,
  original_publication_year VARCHAR(10) NULL DEFAULT NULL,
  title VARCHAR(300) NULL DEFAULT NULL,
  average_rating FLOAT NULL DEFAULT NULL,
  borrowed INT UNSIGNED NULL DEFAULT '0',
  available TINYINT NULL DEFAULT '1',
  rating_count INT UNSIGNED NULL DEFAULT '0',
  PRIMARY KEY (book_id),
  UNIQUE KEY (goodreads_book_id)
);
```

```
# Ratings Table
# Primary Key / Index: rating_id
# Foreign Key / Index: book_id
# Foreign Key / Index: user_id
# Attribute Constraint: rating integer between 1-5 inclusive
CREATE TABLE ratings (
  rating_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  user_id INT UNSIGNED NOT NULL,
  book_id INT UNSIGNED NOT NULL,
  rating TINYINT NOT NULL CHECK(rating > 0 and rating < 6),
  PRIMARY KEY (rating_id),
  FOREIGN KEY (book_id) REFERENCES books (book_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (user_id) REFERENCES users (user_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

# Transactions Table
# Primary Key / Index: transaction_id
# Foreign Key / Index: book_id
# Foreign Key / Index: user_id
CREATE TABLE transactions (
  transaction_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  user_id INT UNSIGNED NOT NULL,
  book_id INT UNSIGNED NOT NULL,
  date_borrowed DATETIME NOT NULL,
  date_due DATETIME NOT NULL,
  date_returned DATETIME NULL DEFAULT NULL,
  PRIMARY KEY (transaction_id),
  FOREIGN KEY (book_id) REFERENCES books (book_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (user_id) REFERENCES users (user_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

# Tags Table
# Primary Key / Index: tag_id
CREATE TABLE tags (
  tag_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  tag_name VARCHAR(200) NULL DEFAULT NULL,
  PRIMARY KEY (tag_id)
);

# Books_Tags Table
# Primary Key / Index: goodreads_book_id, tag_id
# Foreign Key / Index: goodreads_book_id
# Foreign Key / Index: tag_id
CREATE TABLE books_tags (
  goodreads_book_id INT UNSIGNED NOT NULL,
  tag_id INT UNSIGNED NOT NULL,
  count INT UNSIGNED NULL DEFAULT 0,
  PRIMARY KEY (goodreads_book_id, tag_id),
  FOREIGN KEY (goodreads_book_id) REFERENCES books (goodreads_book_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (tag_id) REFERENCES tags (tag_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

# Users_Books Table
# Primary Key / Index: user_id, book_id
# Foreign Key / Index: user_id
# Foreign Key / Index: book_id
# Foreign Key / Index: transaction_id
CREATE TABLE users_books (
  user_id INT UNSIGNED NOT NULL,
  book_id INT UNSIGNED NOT NULL,
  transaction_id INT UNSIGNED NOT NULL,
  PRIMARY KEY (user_id, book_id),
  FOREIGN KEY (user_id) REFERENCES users (user_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (book_id) REFERENCES books (book_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (transaction_id) REFERENCES transactions (transaction_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
```

Out[ ]:
```
[]
```

Load Data Into Tables from CSV Files

In [ ]:
```
# Import all data into tables sourced from
# https://github.com/zygmuntz/goodbooks-10k/tree/master

# Import Books Data (9831 rows)
df1 = pd.read_csv("books_table.csv", sep=',', quotechar='"', encoding='utf8')
df1.to_sql('books', con=eng, index=False, if_exists='append')

# Import Tags Data (31816)
df2 = pd.read_csv("tags_table.csv", sep=',', quotechar='"', encoding='utf8',
    encoding_errors='ignore')
df2.to_sql('tags', con=eng, index=False, if_exists='append')
```

```
# Import Books_Tags Data (116780 rows)
df3 = pd.read_csv("books_tags_sm_table.csv", sep=',',quotechar='"', encoding='utf8')
df3.to_sql('books_tags', con=eng, index=False, if_exists='append');
```

In [ ]:
```
%%sql
# Sample of books data
select book_id, goodreads_book_id as goodreads_id, authors,
    original_publication_year as year, title, average_rating,
    borrowed, available, rating_count
from books
limit 5
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
5 rows affected.
```

Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
1	2767052	Suzanne Collins	2008	The Hunger Games (The Hunger Games, #1)	4.34	0	1	1
2	3	J.K. Rowling, Mary GrandPr	1997	Harry Potter and the Sorcerer's Stone (Harry Potter, #1)	4.44	0	1	1
3	41865	Stephenie Meyer	2005	Twilight (Twilight, #1)	3.57	0	1	1
4	2657	Harper Lee	1960	To Kill a Mockingbird	4.25	0	1	1
5	4671	F. Scott Fitzgerald	1925	The Great Gatsby	3.89	0	1	1

In [ ]:
```
%%sql
# Sample of tags data
select * from tags limit 5
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
5 rows affected.
```

Out[ ]:

tag_id	tag_name
1346	a-man-named-dave
1347	a-manette-ansay
1348	a-mango-shaped-space
1349	a-mano
1350	a-mccall-smith

In [ ]:
```
%%sql
# Sample of books_tags data
select * from books_tags limit 5
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
5 rows affected.
```

Out[ ]:

goodreads_book_id	tag_id	count
1	1691	1742
1	2104	1022
1	2106	305
1	3371	433
1	3389	836

Add Users into Database

In [ ]:
```
%%sql
insert into users(username) values ('julia');
insert into users(username) values ('phil');
insert into users(username) values ('amy');
insert into users(username) values ('rose');
insert into users(username) values ('steve');
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

Out[ ]:
```
[]
```

In [ ]:
```
%%sql
select * from users
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
5 rows affected.
```

Out[ ]:

user_id	username	good_standing	return_count
1	julia	1	0
2	phil	1	0
3	amy	1	0
4	rose	1	0
5	steve	1	0

Create Triggers

#1 When checking out a book (insert a transaction)

Before insert

- Check book is available to check out
- Validate rules on user good standing

After insert

- Update users_books with new row
- Update book to unavailable and increment borrowed

In [ ]:
```
%%sql
drop trigger if exists before_transaction_insert;
create trigger before_transaction_insert
before insert on transactions
for each row
begin
    if exists (
        select *
        from books
        where books.book_id=new.book_id and books.available=0
    ) then
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT =
        'The book is not available to borrow.',
        MYSQL_ERRNO = 1001;
    end if;
    if exists (
        select *
        from users, users_books
        where users.user_id=new.user_id and users.good_standing=0
            and (users.return_count < 5 and
            exists (select * from users_books where users_books.user_id=new.user_id))
    ) then
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT =
        'The user is not allowed to check out more books.',
        MYSQL_ERRNO = 1001;
    end if;
end;

drop trigger if exists after_transaction_insert;
create trigger after_transaction_insert
after insert on transactions
for each row
begin
    insert into users_books values (new.user_id, new.book_id, new.transaction_id);
    update books
        set borrowed=borrowed+1, available=0
        where book_id=new.book_id;
end;
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
```

Out[ ]:
```
[]
```

#2 Returning a book (update transaction)

After update

- Remove correct row from users_books
- Update book to be available
- If book is past due, update users good standing
- If not past due, increment users return count
- If user is not in good standing and return count is five, update to good standing

```
In [ ]: %%sql
drop trigger if exists after_transaction_update;
create trigger after_transaction_update
after update on transactions
for each row
begin
    delete from users_books where transaction_id=new.transaction_id;
    update books
        set available=1
        where book_id=new.book_id;
    if (new.date_returned > new.date_due) then
        update users
            set good_standing=0, return_count=0
            where user_id=new.user_id;
    else
        update users
            set return_count=return_count+1
            where user_id=new.user_id;
    end if;
    if exists (
        select *
        from users
        where user_id=new.user_id and good_standing=0 and return_count >= 5
    )
    then
        update users
        set good_standing=1
        where user_id=new.user_id;
    end if;
end;
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
0 rows affected.
0 rows affected.
Out[ ]: []
```

#3 Rating a book (insert rating)

After insert

- Update book record to calculate new average rating and increase rating count

```
In [ ]: %%sql
drop trigger if exists after_ratings_insert;
create trigger after_ratings_insert
after insert on ratings
for each row
begin
```

```
    update books
        set average_rating=((average_rating+new.rating)/(rating_count+1)),
            rating_count=rating_count+1
        where book_id=new.book_id;
end;
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
0 rows affected.
0 rows affected.
```

Out[ ]:
```
[]
```

#4 Deleting a user (delete user)

Before delete

- Validate user does not have any books checked out or throw error.

In [ ]:
```
%%sql
drop trigger if exists before_user_delete;
create trigger before_user_delete
before delete on users
for each row
begin
if exists (
        select *
        from users_books
        where user_id=old.user_id
    ) then
        SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT =
          'A user may not be deleted if they have books currently checked out.',
          MYSQL_ERRNO = 1001;
    end if;
end;
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
0 rows affected.
0 rows affected.
```

Out[ ]:
```
[]
```

Library use cases demonstration

Search the library for a book by title or author

User can select book (book_id) and validate if book is available for check out.

In [ ]:
```
%%sql
select book_id, goodreads_book_id as goodreads_id, authors,
    original_publication_year as year, title, average_rating,
    borrowed, available, rating_count
from books
where title like '%dune%' or authors like '%dune%'
limit 10
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
10 rows affected.
```

Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
126	234225	Frank Herbert	1965	Dune (Dune Chronicles #1)	4.19	0	1	1
1105	106	Frank Herbert	1969	Dune Messiah (Dune Chronicles #2)	3.86	0	1	1
1262	112	Frank Herbert	1976	Children of Dune (Dune Chronicles #3)	3.9	0	1	1
1687	53764	Frank Herbert	1977	The Great Dune Trilogy	4.35	0	1	1
2045	42432	Frank Herbert	1981	God Emperor of Dune (Dune Chronicles #4)	3.81	0	1	1
2491	117	Frank Herbert	1984	Heretics of Dune (Dune Chronicles #5)	3.83	0	1	1
2817	105	Frank Herbert	1985	Chapterhouse: Dune (Dune Chronicles #6)	3.89	0	1	1
6188	761575	Brian Herbert, Kevin J. Anderson	1999	House Atreides (Prelude to Dune #1)	3.69	0	1	1
6440	99219	Brian Herbert, Kevin J. Anderson	2002	The Butlerian Jihad (Legends of Dune, #1)	3.57	0	1	1
8049	20253	Brian Herbert, Kevin J. Anderson	2000	House Harkonnen (Prelude to Dune #2)	3.63	0	1	1

Check out a book from the library

User is in good standing and the book is available

In [ ]:

```
%%sql
# Verify user is in good standing and return_count is zero
select * from users
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
5 rows affected.
```

Out[ ]:

user_id	username	good_standing	return_count
1	julia	1	0
2	phil	1	0
3	amy	1	0
4	rose	1	0
5	steve	1	0

In [ ]:

```
%%sql
# Verify there are no transactions
select * from transactions
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
0 rows affected.
```

Out[ ]:

transaction_id	user_id	book_id	date_borrowed	date_due	date_returned

In [ ]:

```
%%sql
# There should be no books currently checked out
select * from users_books
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
0 rows affected.
```

Out[ ]:

user_id	book_id	transaction_id

In [ ]:

```
%%sql
# User julia will check out book_id=126, Dune (Dune Chronicles #1)
# Book is due in 2 weeks
insert into transactions(user_id, book_id, date_borrowed, date_due)
values (1, 126, CURDATE(), DATE(date_add(now(),interval 2 week)))
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

```
[]
```

In [ ]:

```
%%sql
# Validate transaction table update
select * from transactions
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

transaction_id	user_id	book_id	date_borrowed	date_due	date_returned
1	1	126	2023-12-10 00:00:00	2023-12-24 00:00:00	None

In [ ]:

```
%%sql
# Validate books table changes (borrowed and available have been updated)
select book_id, goodreads_book_id as goodreads_id, authors,
    original_publication_year as year, title, average_rating,
    borrowed, available, rating_count
from books
where book_id=126
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
126	234225	Frank Herbert	1965	Dune (Dune Chronicles #1)	4.19	1	0	1

In [ ]:

```
%%sql
# Validate users_books table has been updated (new row added)
select * from users_books
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

user_id	book_id	transaction_id
1	126	1

Show books currently checked out for a user

In [ ]:

```
%%sql
# Current books checked out for user 1
select books.title, transactions.date_due
from users_books, books, transactions
where users_books.user_id=1 and
        books.book_id=users_books.book_id and
        transactions.transaction_id=users_books.transaction_id
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

title	date_due
Dune (Dune Chronicles #1)	2023-12-24 00:00:00

Return a book on time

In [ ]:

```
%%sql
# Update the transaction with return date equal to today
update transactions
set date_returned=CURDATE()
where transaction_id=1
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

```
[]
```

In [ ]:

```
%%sql
# Verify book is now available
select book_id, goodreads_book_id as goodreads_id, authors,
    original_publication_year as year, title, average_rating,
    borrowed, available, rating_count
from books
where book_id=126
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
126	234225	Frank Herbert	1965	Dune (Dune Chronicles #1)	4.19	1	1	1

In [ ]:

```
%%sql
# Verify transaction has the correct return date
select * from transactions
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

transaction_id	user_id	book_id	date_borrowed	date_due	date_returned
1	1	126	2023-12-10 00:00:00	2023-12-24 00:00:00	2023-12-10 00:00:00

In [ ]:

```
%%sql
# Verify return_count has been increased by one for user 1
select * from users
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
5 rows affected.
```

Out[ ]:

user_id	username	good_standing	return_count
1	julia	1	1
2	phil	1	0
3	amy	1	0
4	rose	1	0
5	steve	1	0

In [ ]:

```
%%sql
# Verify users_books table has no rows
select * from users_books
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
0 rows affected.
```

Out[ ]:

user_id	book_id	transaction_id

User gives book a valid rating

In [ ]:

```
%%sql
# Book record before rating
select book_id, goodreads_book_id as goodreads_id, authors,
    original_publication_year as year, title, average_rating,
    borrowed, available, rating_count
from books
where book_id=126
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
126	234225	Frank Herbert	1965	Dune (Dune Chronicles #1)	4.19	1	1	1

In [ ]:

```
%%sql
# julia will give Dune a rating of 5
# This should update the average rating of dune to 4.595
# and update the rating_count to 2
insert into ratings(user_id, book_id, rating)
values(1,126,5)
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

```
[]
```

In [ ]:

```
%%sql
# Display updated book entry
select book_id, goodreads_book_id as goodreads_id, authors,
    original_publication_year as year, title, average_rating,
    borrowed, available, rating_count
from books
where book_id=126
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
126	234225	Frank Herbert	1965	Dune (Dune Chronicles #1)	4.595	1	1	2

In [ ]:

```
%%sql
# Validate ratings table includes the new rating
select * from ratings
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

rating_id	user_id	book_id	rating
1	1	126	5

Display related books

This would be included in a GUI to encourage users to check out more books. We know the book_id and goodreads_book_id as the user has just returned the book.

Concepts:

- Common table expressions (CTE)
- Join between three tables (books_tags, books, tags)

In [ ]:

```
%%sql
# Find the tags which are associated with the goodreads_book_id
# Order tags by the most popular tags (highest count)
# Display books having these tags ordered by the average rating
# Limit to 10 books
with
    cte1 AS (select books_tags.tag_id as a
        from books_tags
        where books_tags.goodreads_book_id=234225
        order by books_tags.count desc
    )
select distinct b.book_id, b.title, b.average_rating, b.available
from books as b, cte1, books_tags, tags
where cte1.a=books_tags.tag_id and
        b.goodreads_book_id=books_tags.goodreads_book_id and
        b.goodreads_book_id!=234225 and
        tags.tag_id=books_tags.tag_id
order by b.average_rating desc
limit 10;
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
10 rows affected.
```

Out[ ]:

book_id	title	average_rating	available
3628	The Complete Calvin and Hobbes	4.82	1
3275	Harry Potter Boxed Set, Books 1-5 (Harry Potter, #1-5)	4.77	1
862	Words of Radiance (The Stormlight Archive, #2)	4.77	1
8854	Mark of the Lion Trilogy	4.76	1
7947	ESV Study Bible	4.76	1
4483	It's a Magical World: A Calvin and Hobbes Collection	4.75	1
6361	There's Treasure Everywhere: A Calvin and Hobbes Collection	4.74	1
422	Harry Potter Boxset (Harry Potter, #1-7)	4.74	1
3753	Harry Potter Collection (Harry Potter, #1-6)	4.73	1
6590	The Authoritative Calvin and Hobbes: A Calvin and Hobbes Treasury	4.73	1

Report on "Top Books"

Report is generated by querying the top 10 highest rated books, then pulling the tags for each book. Each book includes a list of all the tags and the total number of times the book has been tagged is calculated by summing the individual tag counts.

Concepts:

- Common table expressions (CTE)
- Join between three tables (books, tags, books_tags)
- Group By
- Aggregation

In [ ]:

```
%%sql
with
    cte1 AS (
        select books.goodreads_book_id as gid, books.title as ti, books.borrowed as bo,
            books.average_rating as ar, books.rating_count as rc
        from books
        order by books.average_rating desc
        limit 10
    ),
    cte2 as (
        select gid, ti, bo, ar, rc, books_tags.tag_id as tid, tags.tag_name as tname,
            books_tags.count as btc
        from books_tags, tags, cte1
        where cte1.gid=books_tags.goodreads_book_id and tags.tag_id=books_tags.tag_id
        order by books_tags.count desc
    )
select cte2.ti as Title, cte2.ar as 'Average Rating', group_concat(distinct cte2.tname
        order by cte2.tname
        separator ', ') as Tags, sum(cte2.btc) as 'Total number of times tagged'
from cte2
group by cte2.ti, cte2.bo, cte2.ar
order by cte2.ar desc, cte2.bo desc;
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
10 rows affected.
```

Out[ ]:

Title	Average Rating	Tags	Total number of times tagged
The Complete Calvin and Hobbes	4.82	comics, currently-reading, favorites, fiction, graphic-novels, humor, owned, to-read	13354
Harry Potter Boxed Set, Books 1-5 (Harry Potter, #1-5)	4.77	fantasy, favorites, to-read	3071
Words of Radiance (The Stormlight Archive, #2)	4.77	audible, audiobook, audiobooks, books-i-own, brandon-sanderson, cosmere, currently-reading, epic, epic-fantasy, fantasy, favorites, favourites, fiction, high-fantasy, kindle, magic, owned, read-in-2014, sanderson, sci-fi-fantasy, series, to-read	20234
ESV Study Bible	4.76	currently-reading, favorites, to-read	463
Mark of the Lion Trilogy	4.76	christian-fiction, to-read	356
It's a Magical World: A Calvin and Hobbes Collection	4.75	comics, currently-reading, favorites, fiction, graphic-novels, humor, to-read	4786
Harry Potter Boxset (Harry Potter, #1-7)	4.74	adventure, all-time-favorites, books-i-own, childhood, children, children-s, childrens, classics, currently-reading, fantasy, favorite, favorite-books, favorites, favourites, fiction, owned, owned-books, re-read, series, shelfari-favorites, to-read, ya, young-adult	13871
There's Treasure Everywhere: A Calvin and Hobbes Collection	4.74	comics, favorites, humor, to-read	2494
Harry Potter Collection (Harry Potter, #1-6)	4.73	currently-reading, fantasy, favorites, to-read	3936
The Authoritative Calvin and Hobbes: A Calvin and Hobbes Treasury	4.73	comic, comics, currently-reading, favorites, fiction, graphic-novels, humor, owned, to-read	4487

View previously borrowed books for users with ratings

As we only have one rating added to the library so far, we should see one row.

Concepts:

- Join between 4 tables (transactions, books, users, ratings)

In [ ]:
```
%%sql
select users.username, books.title, transactions.date_borrowed,
        transactions.date_returned, ratings.rating
from transactions, books, users, ratings
where transactions.user_id=users.user_id and
        transactions.book_id=books.book_id and
        transactions.book_id=ratings.book_id and
        transactions.user_id=ratings.user_id
order by users.user_id
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

username	title	date_borrowed	date_returned	rating
julia	Dune (Dune Chronicles #1)	2023-12-10 00:00:00	2023-12-10 00:00:00	5

Delete a user

Because of foreign key relationships all transactions and ratings will be removed as well.

In [ ]:
```
%%sql
# User table before delete
select * from users
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
5 rows affected.
```

Out[ ]:

user_id	username	good_standing	return_count
1	julia	1	1
2	phil	1	0
3	amy	1	0
4	rose	1	0
5	steve	1	0

In [ ]:
```
%%sql
# Transaction table before delete
select * from transactions
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

transaction_id	user_id	book_id	date_borrowed	date_due	date_returned
1	1	126	2023-12-10 00:00:00	2023-12-24 00:00:00	2023-12-10 00:00:00

In [ ]:
```
%%sql
# Ratings table before delete
select * from ratings
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

rating_id	user_id	book_id	rating
1	1	126	5

In [ ]:
```
%%sql
# Delete user julia (user_id=1)
delete from users where user_id=1
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:
```
[]
```

```
In [ ]: %%sql
        # Users table now
        select * from users
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
4 rows affected.
```

Out[ ]:

user_id	username	good_standing	return_count
2	phil	1	0
3	amy	1	0
4	rose	1	0
5	steve	1	0

```
In [ ]: %%sql
        # Transactions table now
        select * from transactions
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
0 rows affected.
```

Out[ ]:

transaction_id	user_id	book_id	date_borrowed	date_due	date_returned

```
In [ ]: %%sql
        # Ratings table now
        select * from ratings
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
0 rows affected.
```

Out[ ]:

rating_id	user_id	book_id	rating

Detailed tests on triggers

Here we will test each trigger in depth

#1 When checking out a book (insert a transaction)

Before insert

- Check book is available to check out
- Validate rules on user good standing

After insert

- Update users_books with new row
- Update book to unavailable and increment borrowed

Book must be available to check out for insert a transaction to be successful.

In [ ]:
```
%%sql
# Have user phil check out a book
insert into transactions(user_id, book_id, date_borrowed, date_due)
values (2, 7, CURDATE(), DATE(date_add(now(),interval 2 week)))
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:
```
[]
```

In [ ]:
```
%%sql
# Display updated book to show it is not available
select book_id, goodreads_book_id as goodreads_id, authors,
    original_publication_year as year, title, average_rating,
    borrowed, available, rating_count
from books
where book_id=7
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
7	5907	J.R.R. Tolkien	1937	The Hobbit	4.25	1	0	1

In [ ]:
```
%%sql
# User amy tries to checkout the same book
# Error will be shown
insert into transactions(user_id, book_id, date_borrowed, date_due)
values (3, 7, CURDATE(), DATE(date_add(now(),interval 2 week)))
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
(pymysql.err.OperationalError) (1001, 'The book is not available to borrow.')
[SQL: # User amy tries to checkout the same book
# Error will be shown
insert into transactions(user_id, book_id, date_borrowed, date_due)
values (3, 7, CURDATE(), DATE(date_add(now(),interval 2 week)))]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

User phil has one book checked out. If he is not in good standing, he will not be able to check out another book. As he is currently in good standing I will manually set to 0.

In [ ]:
```
%%sql
# Update phil's account to good_standing=0
update users
set good_standing=0
where user_id=2
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:
```
[]
```

In [ ]:
```
%%sql
# Verify account change
select * from users
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
4 rows affected.
```

Out[ ]:

user_id	username	good_standing	return_count
2	phil	0	0
3	amy	1	0
4	rose	1	0
5	steve	1	0

Now phil tries to check out a second book. Error will be thrown because he has one book checked out and his account is not in good standing.

In [ ]:

```
%%sql
insert into transactions(user_id, book_id, date_borrowed, date_due)
values (2, 10, CURDATE(), DATE(date_add(now(),interval 2 week)))
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
(pymysql.err.OperationalError) (1001, 'The user is not allowed to check out more books.')
[SQL: insert into transactions(user_id, book_id, date_borrowed, date_due)
values (2, 10, CURDATE(), DATE(date_add(now(),interval 2 week)))]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

Test after insert trigger. When book is checked out, the users_books table is updated with a new row and book table has been updated to not available and borrowed has been incremented by one.

Amy will check out book_id=2. First let's look at the information for book_id=2.

In [ ]:

```
%%sql
select book_id, goodreads_book_id as goodreads_id, authors,
    original_publication_year as year, title, average_rating,
    borrowed, available, rating_count
from books
where book_id=2
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
2	3	J.K. Rowling, Mary GrandPr	1997	Harry Potter and the Sorcerer's Stone (Harry Potter, #1)	4.44	0	1	1

In [ ]:

```
%%sql
# users_books before the update
select * from users_books
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

user_id	book_id	transaction_id
2	7	2

In [ ]:
```
%%sql
# Check out the book
insert into transactions(user_id, book_id, date_borrowed, date_due)
values (3, 2, CURDATE(), DATE(date_add(now(),interval 2 week)))
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:
```
[]
```

In [ ]:
```
%%sql
# Books table update (not available, borrowed incremented)
select book_id, goodreads_book_id as goodreads_id, authors,
    original_publication_year as year, title, average_rating,
    borrowed, available, rating_count
from books
where book_id=2
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
2	3	J.K. Rowling, Mary GrandPr	1997	Harry Potter and the Sorcerer's Stone (Harry Potter, #1)	4.44	1	0	1

In [ ]:
```
%%sql
# users_books table update
# New row has been added by trigger
select * from users_books
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
2 rows affected.
```

Out[ ]:

user_id	book_id	transaction_id
2	7	2
3	2	3

#2 Returning a book (update transaction)

After update

- Remove correct row from users_books
- Update book to be available
- If book is past due, update users good standing
- If not past due, increment users return count
- If user is not in good standing and return count is five, update to good standing

Test when returning a book late. Start by creating the new transaction.

In [ ]:
```
%%sql
# First need to create another transaction where the return date
# is before today
insert into transactions(user_id, book_id, date_borrowed, date_due)
values (4, 4, CURDATE(), DATE(date_sub(now(),interval 2 week)))
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```
Out[ ]:
```
[]
```

In [ ]:
```
%%sql
# Validate return date is before today
select * from transactions
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
3 rows affected.
```
Out[ ]:

transaction_id	user_id	book_id	date_borrowed	date_due	date_returned
2	2	7	2023-12-10 00:00:00	2023-12-24 00:00:00	None
3	3	2	2023-12-10 00:00:00	2023-12-24 00:00:00	None
4	4	4	2023-12-10 00:00:00	2023-11-26 00:00:00	None

In [ ]:
```
%%sql
# users_books table has a new row from the insert
select * from users_books
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
3 rows affected.
```
Out[ ]:

user_id	book_id	transaction_id
2	7	2
3	2	3
4	4	4

In [ ]:
```
%%sql
# Book is no longer available
select book_id, goodreads_book_id as goodreads_id, authors,
    original_publication_year as year, title, average_rating,
    borrowed, available, rating_count
from books
where book_id=4
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```
Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
4	2657	Harper Lee	1960	To Kill a Mockingbird	4.25	1	0	1

Return the book late. Validate user good_standing has been changed to 0.

```
In [ ]: %%sql
        # Return the book (late)
        update transactions
        set date_returned=CURDATE()
        where transaction_id=4
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
Out[ ]: []
```

```
In [ ]: %%sql
        # Transactions table now with date returned has been updated
        select * from transactions
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
3 rows affected.
```

Out[ ]:

transaction_id	user_id	book_id	date_borrowed	date_due	date_returned
2	2	7	2023-12-10 00:00:00	2023-12-24 00:00:00	None
3	3	2	2023-12-10 00:00:00	2023-12-24 00:00:00	None
4	4	4	2023-12-10 00:00:00	2023-11-26 00:00:00	2023-12-10 00:00:00

```
In [ ]: %%sql
        # User should now have good_standing=0
        # Note also that the return_count is 0
        select * from users where user_id=4
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

user_id	username	good_standing	return_count
4	rose	0	0

```
In [ ]: %%sql
        # users_books table has been updated
        # Row has been removed
        select * from users_books
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
2 rows affected.
```

Out[ ]:

user_id	book_id	transaction_id
2	7	2
3	2	3

```
In [ ]: %%sql
        # Books table has been updated to show available
        select book_id, goodreads_book_id as goodreads_id, authors,
            original_publication_year as year, title, average_rating,
            borrowed, available, rating_count
        from books
        where book_id=4
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
4	2657	Harper Lee	1960	To Kill a Mockingbird	4.25	1	1	1

Next we need to test the case where a user is not in good standing but has had 5 returned books on time so their standing is back to good.

In [ ]:

```
%%sql
# As user Rose is not in good standing we will use that account
# Queries are listed below for 5 checkout/returns on time
insert into transactions(user_id, book_id, date_borrowed, date_due)
values (4, 10, CURDATE(), DATE(date_add(now(),interval 2 week)));
update transactions set date_returned=CURDATE() where transaction_id=5;

insert into transactions(user_id, book_id, date_borrowed, date_due)
values (4, 11, CURDATE(), DATE(date_add(now(),interval 2 week)));
update transactions set date_returned=CURDATE() where transaction_id=6;

insert into transactions(user_id, book_id, date_borrowed, date_due)
values (4, 12, CURDATE(), DATE(date_add(now(),interval 2 week)));
update transactions set date_returned=CURDATE() where transaction_id=7;

insert into transactions(user_id, book_id, date_borrowed, date_due)
values (4, 13, CURDATE(), DATE(date_add(now(),interval 2 week)));
update transactions set date_returned=CURDATE() where transaction_id=8;

insert into transactions(user_id, book_id, date_borrowed, date_due)
values (4, 14, CURDATE(), DATE(date_add(now(),interval 2 week)));
update transactions set date_returned=CURDATE() where transaction_id=9;
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

Out[ ]:

```
[]
```

Rose should now have 5 books returned on time and she is now back to good standing.

In [ ]:

```
%%sql
select * from users
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
4 rows affected.
```

Out[ ]:

user_id	username	good_standing	return_count
2	phil	0	0
3	amy	1	0
4	rose	1	5
5	steve	1	0

#3 Rating a book (insert rating)

After insert

- Update book record to calculate new average rating and increase rating count

Add rating with valid integer value. First let's see what the rating is currently.

In [ ]:
```
%%sql
# View book before rating
select book_id, goodreads_book_id as goodreads_id, authors,
    original_publication_year as year, title, average_rating,
    borrowed, available, rating_count
from books
where book_id=10
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
10	1885	Jane Austen	1813	Pride and Prejudice	4.24	1	1	1

Now add a rating of 5 for book_id=10.

In [ ]:
```
%%sql
# Add rating
insert into ratings(user_id, book_id, rating)
values(5,10,5)
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:
```
[]
```

In [ ]:
```
%%sql
# Rating should now be updated to 4.62 and rating count to 2
select book_id, goodreads_book_id as goodreads_id, authors,
    original_publication_year as year, title, average_rating,
    borrowed, available, rating_count
from books
where book_id=10
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

book_id	goodreads_id	authors	year	title	average_rating	borrowed	available	rating_count
10	1885	Jane Austen	1813	Pride and Prejudice	4.62	1	1	2

In [ ]:
```
%%sql
# Validate rating is in ratings table
select * from ratings
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
1 rows affected.
```

Out[ ]:

rating_id	user_id	book_id	rating
2	5	10	5

#4 Deleting a user (delete user)

Before delete

- Validate user does not have any books checked out or throw error

Let's see who currently has books checked out.

In [ ]:
```
%%sql
select * from users_books
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
2 rows affected.
```

Out[ ]:

user_id	book_id	transaction_id
2	7	2
3	2	3

Phil has a book checked out. Let's try to delete his account. We should see an error.

In [ ]:
```
%%sql
# Try to delete Phil's account
# Will throw error
delete from users where user_id=2
```

```
 * mysql+pymysql://root:***@localhost:3306/cspb_library
(pymysql.err.OperationalError) (1001, 'A user may not be deleted if they have books currently
checked out.')
[SQL: # Try to delete Phil's account
# Will throw error
delete from users where user_id=2]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

Summary

This project has demonstrated many of the concepts covered in our class CSPB 3287.

I have included:

- E/R diagram
- Multiple table relationships
- Constraints
- Indexes
- SQL statements for table creation, insertion of data, updates, and queries
- LIKE operator
- Triggers
- Common table expressions (CTE)
- Joins between 3 and 4 tables
- Grouping of data
- Aggregation
- Deletion of items that have foreign keys