CSE414 DATABASE SYSTEMS
SEMESTER PROJECT

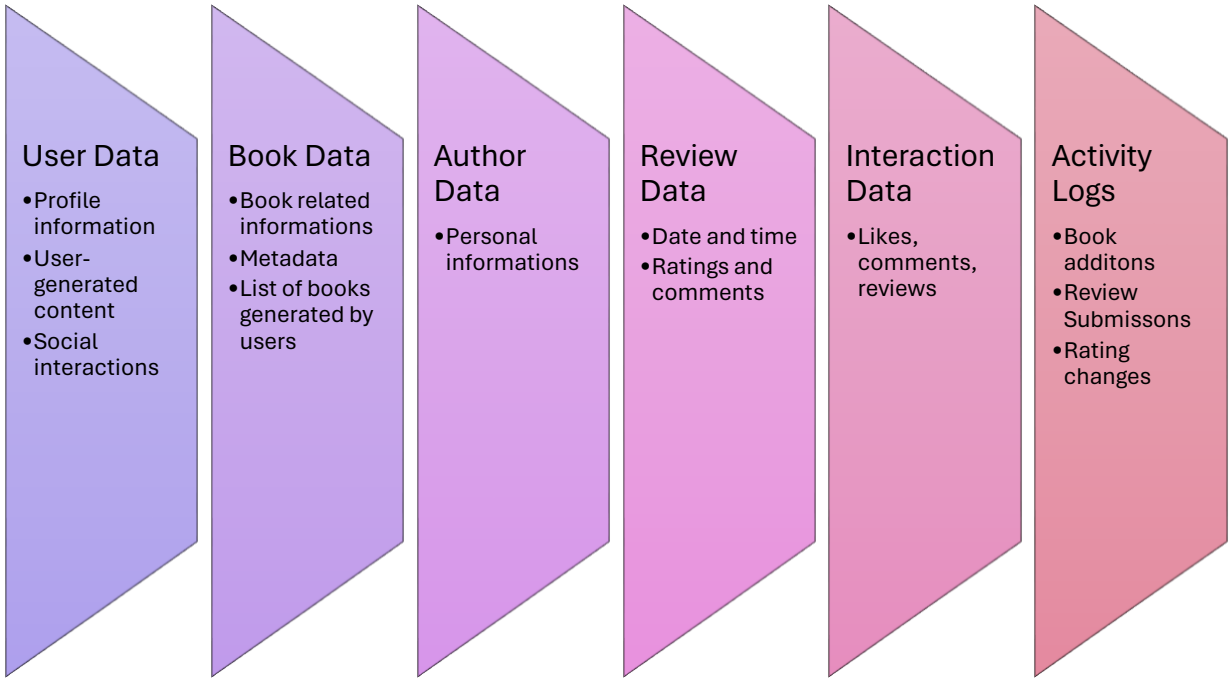
GOODREADS BACKEND

AYŞE GÜL DEMİRBİLEK

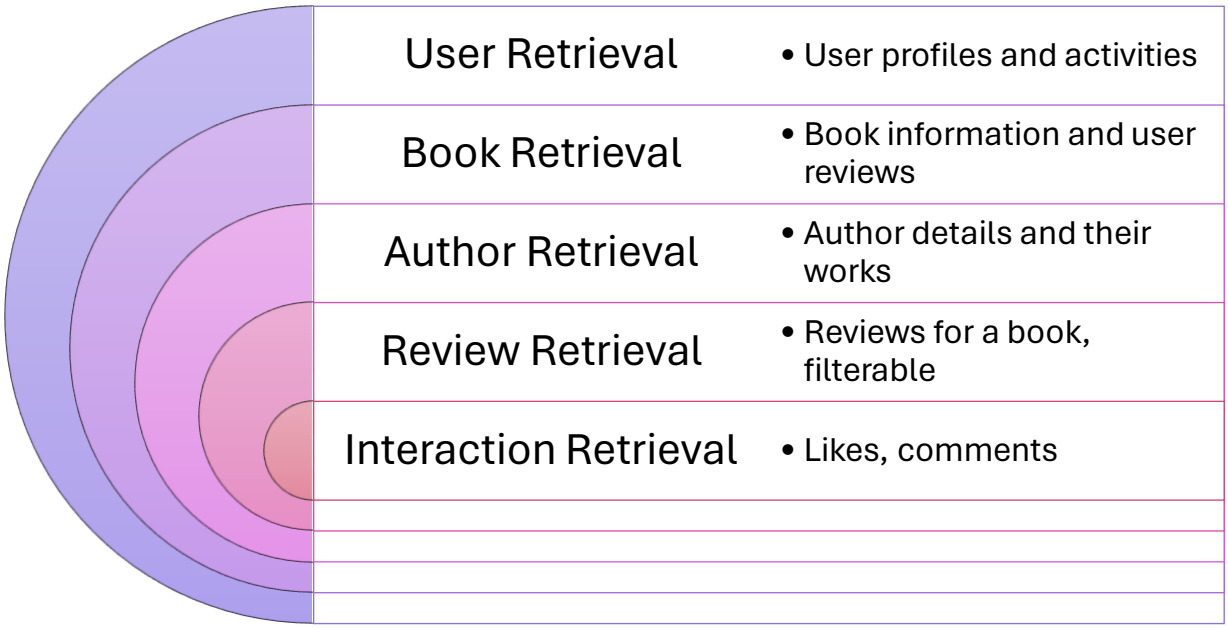
1801042088

USER REQUIREMENTS

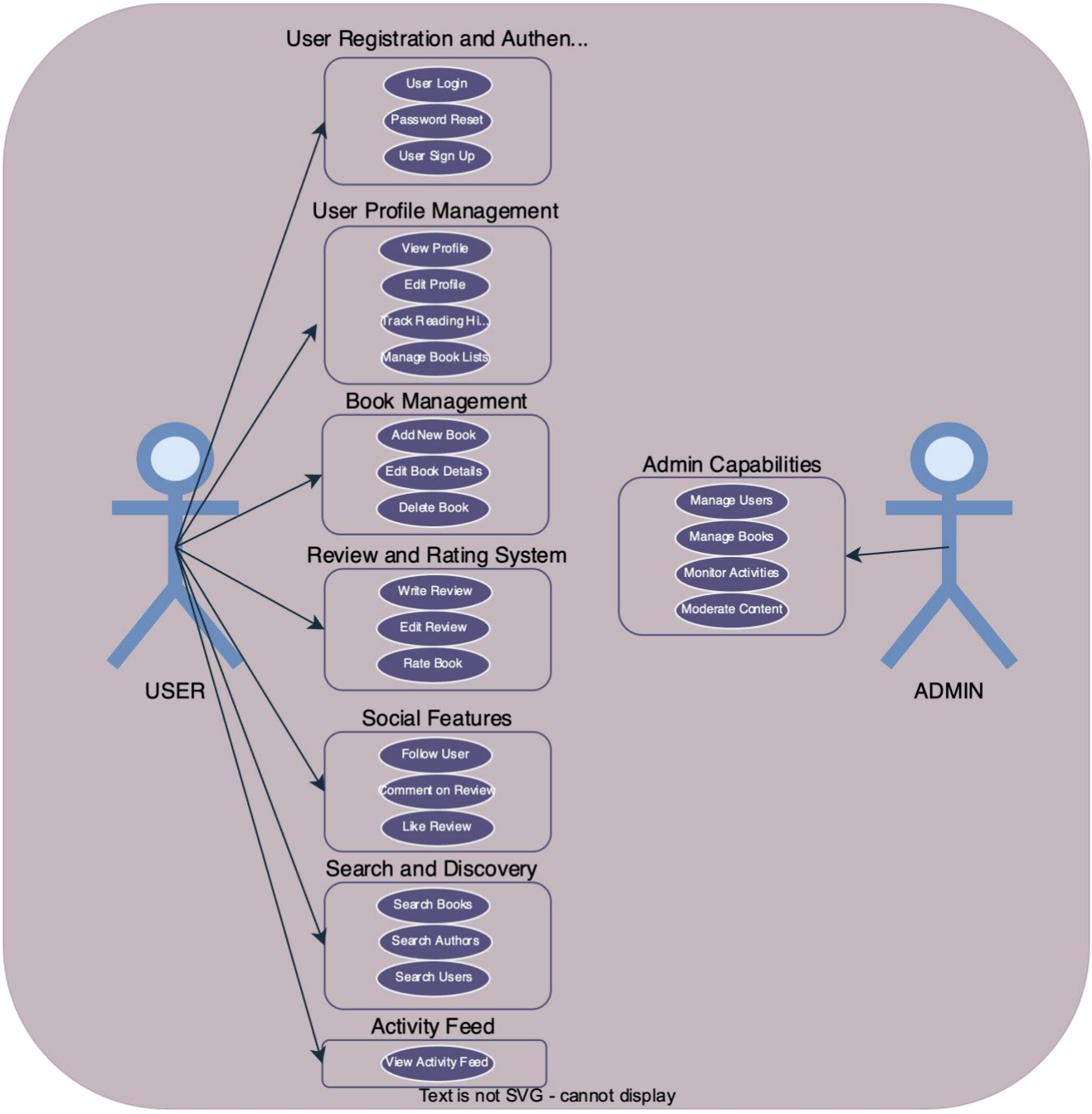
DATA STORAGE



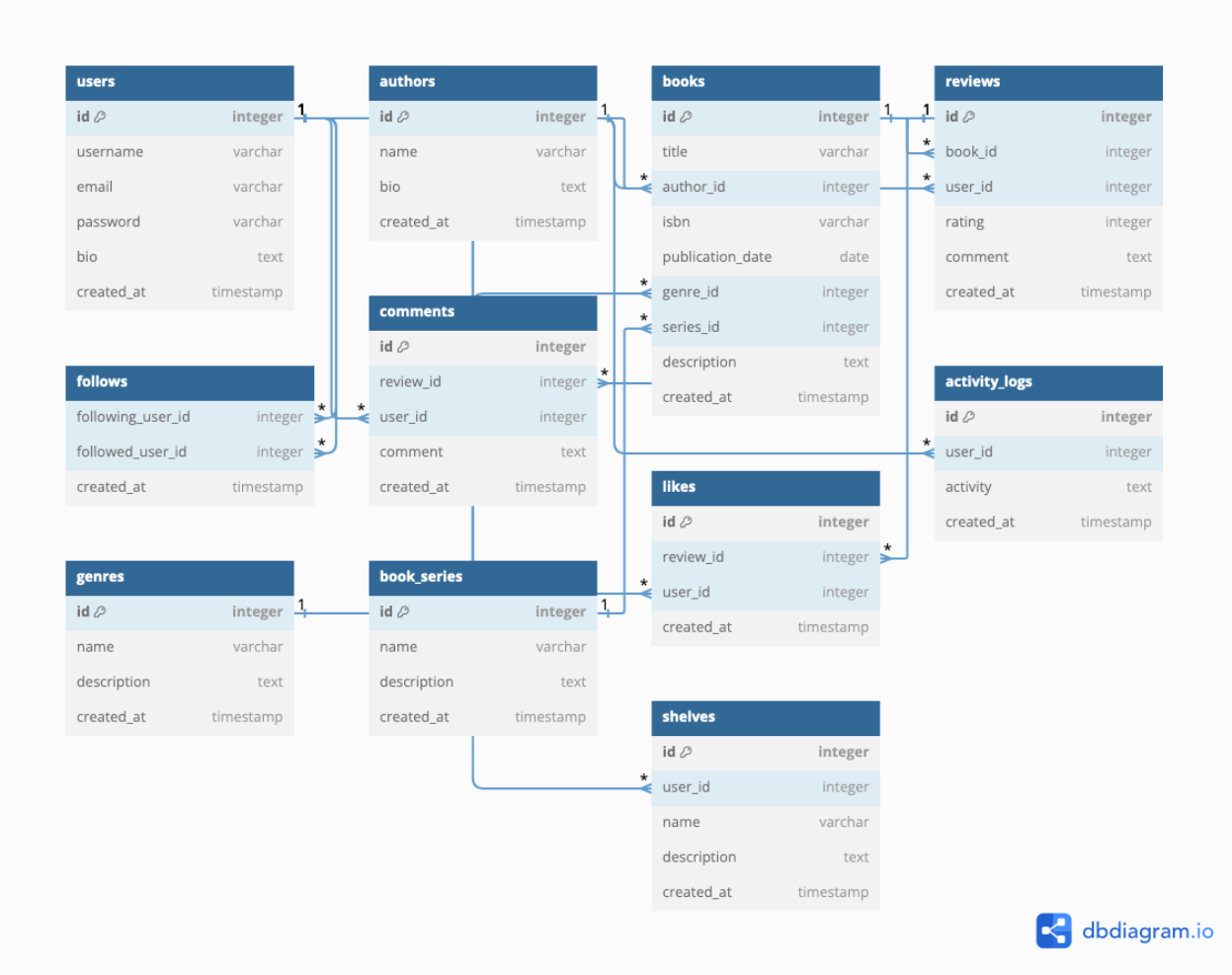
RETRIEVAL CAPACITIES



SPECIFIC FUNCTIONALITIES DESIRED BY USERS



ENTITY-RELATIONSHIP (E-R) DIAGRAM



- Each user (users) can write multiple reviews (reviews), and each review is linked to a specific book (books) and a user.
- Books are authored by authors (authors), creating a many-to-one relationship between books and authors.
- Users can follow other users through the follows table, establishing self-referential relationships within the users table.
- Reviews can receive multiple comments (comments) and likes (likes), both linked to users and reviews, representing many-to-one relationships.
- Additionally, user activities are logged in the activity_logs table, recording various actions performed by users, linked back to the users table.

FUNCTIONAL DEPENDENCIES & DATABASE SCHEMA

Tables	Fields	Explanation
Users	**user_id**	user_id uniquely determines the username, email, password, and created_at attributes for each user.
	username	
	email	
	password	
	created_at	
Authors	**author_id**	author_id uniquely determines the name, bio, and created_at attributes for each author.
	name	
	bio	
	created_at	
Books	**book_id**	book_id uniquely determines the title, author_id, isbn, publication_date, genre_id, series_id, description, and created_at attributes for each book.
	title	
	author_id	
	publication_date,	
	isbn	
	genre_id	
	series_id	
	description	
	created_at	
Reviews	**review_id**	review_id uniquely determines the book_id, user_id, rating, comment, and created_at attributes for each review.
	book_id	
	user_id	
	rating	
	comment	
	created_at	
Follows	**following_user_id**	The combination of following_user_id and followed_user_id uniquely determines the created_at attribute for each follow relationship.
	followed_user_id	
	created_at	
Comments	**comment_id**	comment_id uniquely determines the review_id, user_id, comment, and created_at attributes for each comment.
	review_id	
	user_id	
	comment	
	created_at	
Likes	**like_id**	like_id uniquely determines the review_id, user_id, and created_at attributes for each like.
	review_id	
	user_id	
	created_at	
Activity Logs	**activity_log_id**	activity_log_id uniquely determines the user_id, activity, and created_at attributes for each activity log.
	user_id	
	activity	
	created_at	
Genres	**genre_id**	genre_id uniquely determines the name, description, and created_at attributes for each genre.
	name	
	description	
	created_at	
Book_Series	**series_id**	series_id uniquely determines the name, description, and created_at attributes for each book series.
	name	
	description	
	created_at	
Shelves	**shelf_id**	shelf_id uniquely determines the user_id, name, description, and created_at attributes for each shelf.
	user_id	
	name	
	description	
	created_at	

NORMALIZATION

All the tables are created aligning with 3NF rules, their id's are primary keys and all other attributes are fully functionally dependent on it. So I represent example of compositions which are not in any normalization form:

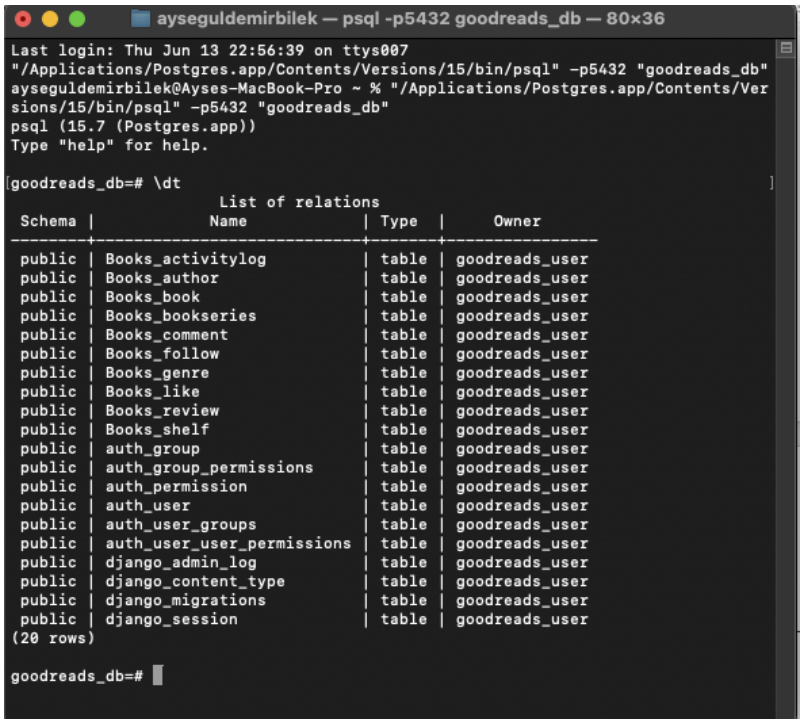
book_id	book_title	author_name	author_bio
1	Book 1	Author 1	Bio 1
2	Book 2	Author 1	Bio 1
3	Book 3	Author 2	Bio 2

2NF/3NF (Eliminate Partial Dependency):

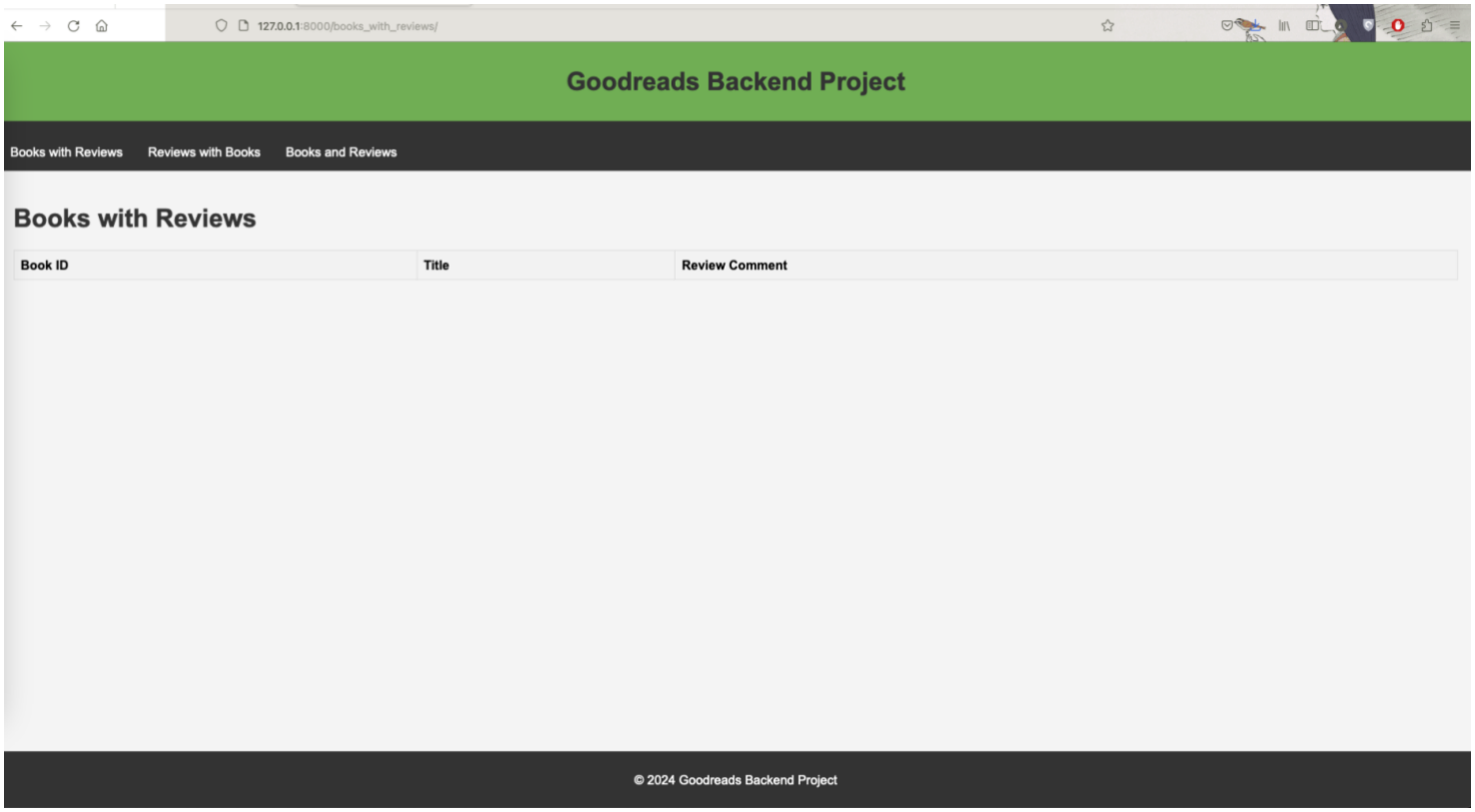
book_id	book_title	author_id
1	Book 1	1
2	Book 2	1
3	Book 3	2

author_id	author_name	author_bio
1	Book 1	Bio 1
2	Book 2	Bio 2

DATABASE INTEGRATION



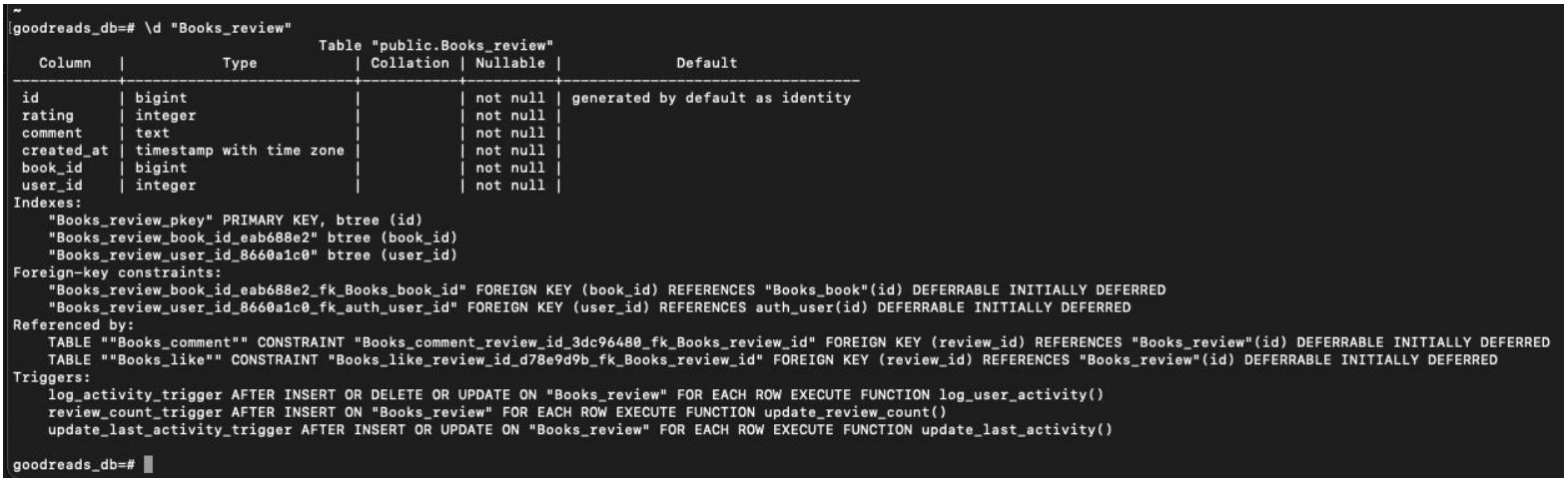
USER INTERFACE



TRIGGERS

TRIGGERS	Trigger Name	Function	Purpose
Update Review Count for a Book	review_count_trigger	update_review_count	Increment the review count for a book whenever a new review is added.
Log User Activities	log_activity_trigger	log_user_activity	Log any activity performed by a user into the Books_activitylog table.
Update Last Activity Date	update_last_activity_trigger	update_last_activity	Update the last activity date of a user whenever they perform any action.
Prevent Duplicate Follows	prevent_duplicate_follows_trigger	prevent_duplicate_follows	Prevent a user from following the same user more than once.
Set Default Shelf for New Users	create_default_shelf_trigger	create_default_shelf	Automatically create a default shelf for a new user when they sign up.

Books_review table and includes a list of triggers defined on the table, such as review_count_trigger.



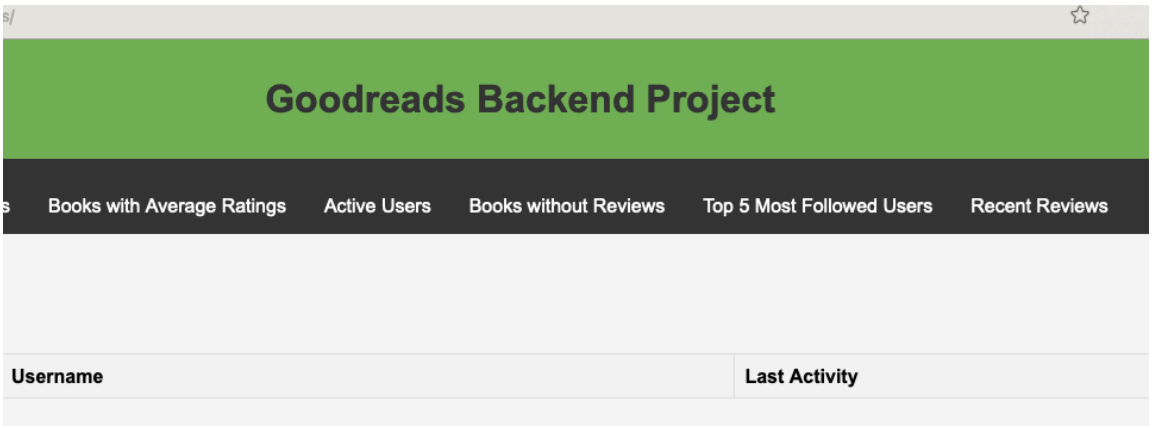
```
~
goodreads_db=# \d "Books_review"
                                  Table "public.Books_review"
   Column   |           Type           | Collation | Nullable |              Default
------------+--------------------------+-----------+----------+-----------------------------------
 id         | bigint                   |           | not null | generated by default as identity
 rating     | integer                  |           | not null |
 comment    | text                     |           | not null |
 created_at | timestamp with time zone |           | not null |
 book_id    | bigint                   |           | not null |
 user_id    | integer                  |           | not null |
Indexes:
    "Books_review_pkey" PRIMARY KEY, btree (id)
    "Books_review_book_id_eab688e2" btree (book_id)
    "Books_review_user_id_8660a1c0" btree (user_id)
Foreign-key constraints:
    "Books_review_book_id_eab688e2_fk_Books_book_id" FOREIGN KEY (book_id) REFERENCES "Books_book"(id) DEFERRABLE INITIALLY DEFERRED
    "Books_review_user_id_8660a1c0_fk_auth_user_id" FOREIGN KEY (user_id) REFERENCES auth_user(id) DEFERRABLE INITIALLY DEFERRED
Referenced by:
    TABLE ""Books_comment"" CONSTRAINT "Books_comment_review_id_3dc96480_fk_Books_review_id" FOREIGN KEY (review_id) REFERENCES "Books_review"(id) DEFERRABLE INITIALLY DEFERRED
    TABLE ""Books_like"" CONSTRAINT "Books_like_review_id_d78e9d9b_fk_Books_review_id" FOREIGN KEY (review_id) REFERENCES "Books_review"(id) DEFERRABLE INITIALLY DEFERRED
Triggers:
    log_activity_trigger AFTER INSERT OR DELETE OR UPDATE ON "Books_review" FOR EACH ROW EXECUTE FUNCTION log_user_activity()
    review_count_trigger AFTER INSERT ON "Books_review" FOR EACH ROW EXECUTE FUNCTION update_review_count()
    update_last_activity_trigger AFTER INSERT OR UPDATE ON "Books_review" FOR EACH ROW EXECUTE FUNCTION update_last_activity()

goodreads_db=#
```

VIEWS

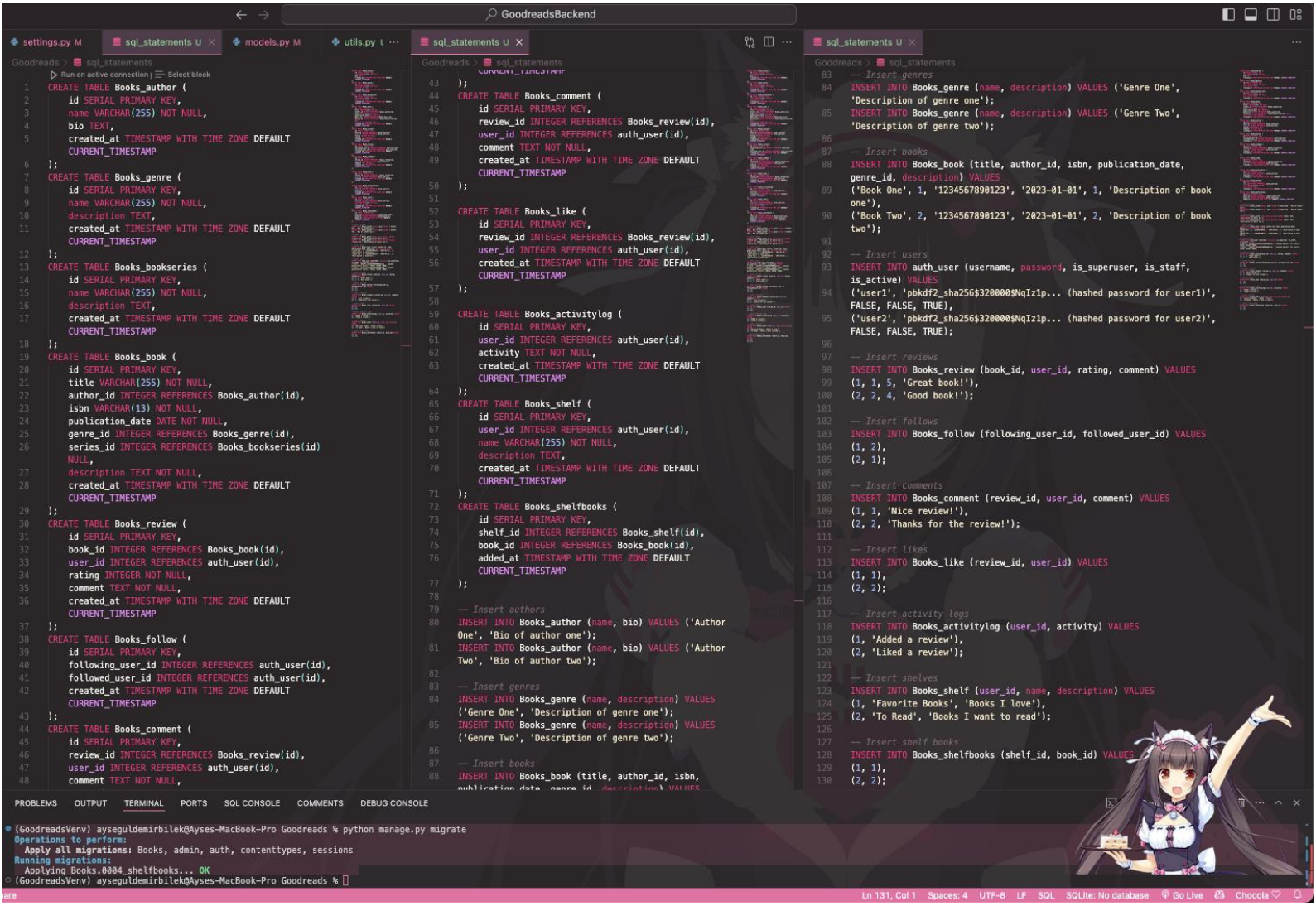
VIEWS	Utility
List Books with Their Average Ratings	This view provides an easy way to see the average rating of each book.
List Active Users	This view helps identify users who are actively engaging with the platform.
List Books Without Reviews	This view highlights books that have not been reviewed yet.
List Top 5 Most Followed Users	This view identifies the most popular users on the platform.
List Recent Reviews	This view provides an up-to-date list of recent reviews.

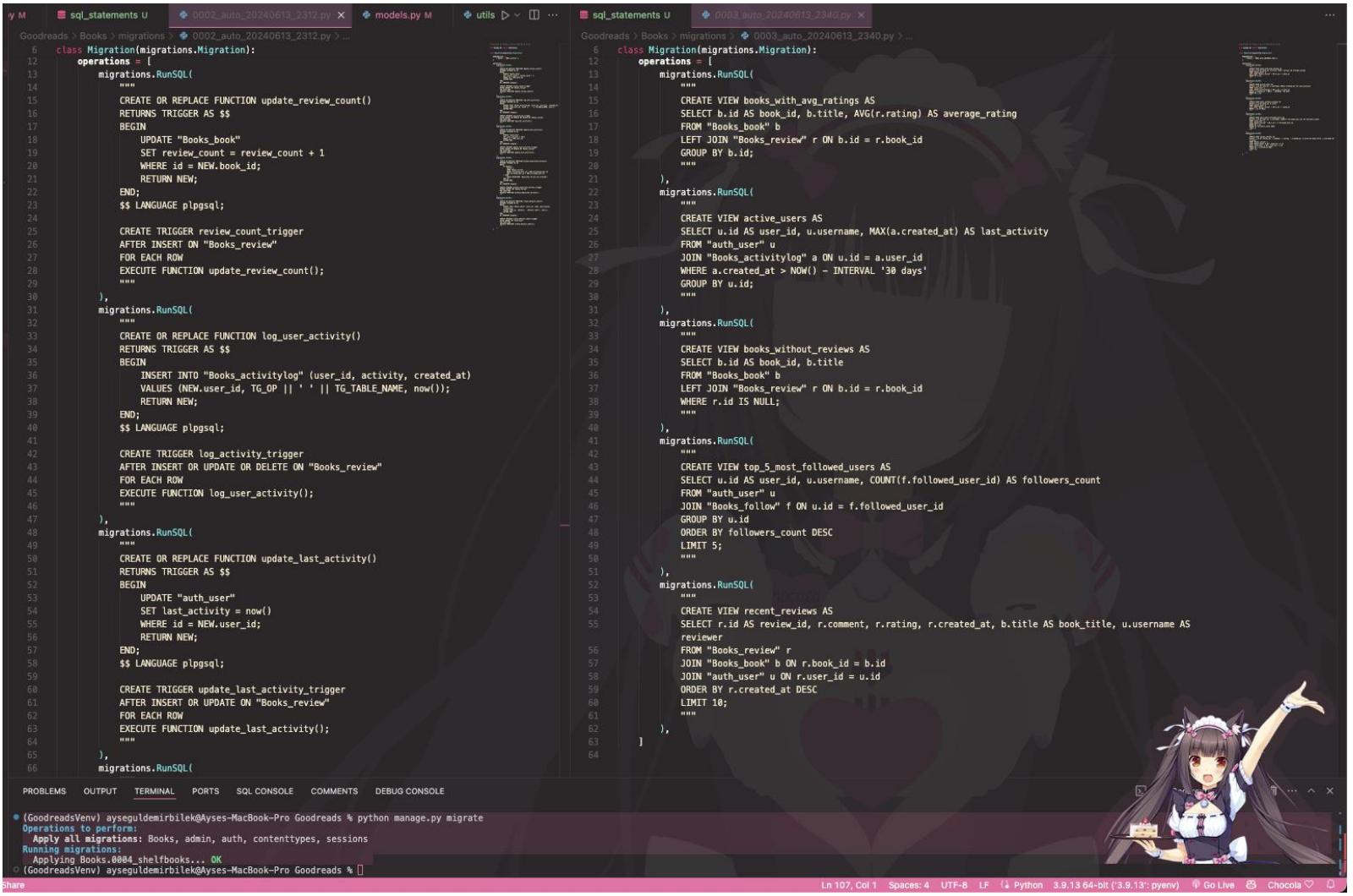
In the user interface, these views are displayed under the corresponding tabs.



The views also can be checked from psql console.

SQL STATEMENTS




sql_statements U
0002_auto_20240613_2312.py
models.py M
utils
Goodreads > Books > migrations > 0002_auto_20240613_2312.py > ...
class Migration(migrations.Migration):
operations = [
migrations.RunSQL(
"""
CREATE OR REPLACE FUNCTION update_review_count()
RETURNS TRIGGER AS $$
BEGIN
UPDATE "Books_book"
SET review_count = review_count + 1
WHERE id = NEW.book_id;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER review_count_trigger
AFTER INSERT ON "Books_review"
FOR EACH ROW
EXECUTE FUNCTION update_review_count();
"""
),
migrations.RunSQL(
"""
CREATE OR REPLACE FUNCTION log_user_activity()
RETURNS TRIGGER AS $$
BEGIN
INSERT INTO "Books_activitylog" (user_id, activity, created_at)
VALUES (NEW.user_id, TG_OP || ' ' || TG_TABLE_NAME, now());
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER log_activity_trigger
AFTER INSERT OR UPDATE OR DELETE ON "Books_review"
FOR EACH ROW
EXECUTE FUNCTION log_user_activity();
"""
),
migrations.RunSQL(
"""
CREATE OR REPLACE FUNCTION update_last_activity()
RETURNS TRIGGER AS $$
BEGIN
UPDATE "auth_user"
SET last_activity = now()
WHERE id = NEW.user_id;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER update_last_activity_trigger
AFTER INSERT OR UPDATE ON "Books_review"
FOR EACH ROW
EXECUTE FUNCTION update_last_activity();
"""
),
migrations.RunSQL(
sql_statements U
0003_auto_20240613_2340.py
Goodreads > Books > migrations > 0003_auto_20240613_2340.py > ...
class Migration(migrations.Migration):
operations = [
migrations.RunSQL(
"""
CREATE VIEW books_with_avg_ratings AS
SELECT b.id AS book_id, b.title, AVG(r.rating) AS average_rating
FROM "Books_book" b
LEFT JOIN "Books_review" r ON b.id = r.book_id
GROUP BY b.id;
"""
),
migrations.RunSQL(
"""
CREATE VIEW active_users AS
SELECT u.id AS user_id, u.username, MAX(a.created_at) AS last_activity
FROM "auth_user" u
JOIN "Books_activitylog" a ON u.id = a.user_id
WHERE a.created_at > NOW() - INTERVAL '30 days'
GROUP BY u.id;
"""
),
migrations.RunSQL(
"""
CREATE VIEW books_without_reviews AS
SELECT b.id AS book_id, b.title
FROM "Books_book" b
LEFT JOIN "Books_review" r ON b.id = r.book_id
WHERE r.id IS NULL;
"""
),
migrations.RunSQL(
"""
CREATE VIEW top_5_most_followed_users AS
SELECT u.id AS user_id, u.username, COUNT(f.followed_user_id) AS followers_count
FROM "auth_user" u
JOIN "Books_follow" f ON u.id = f.followed_user_id
GROUP BY u.id
ORDER BY followers_count DESC
LIMIT 5;
"""
),
migrations.RunSQL(
"""
CREATE VIEW recent_reviews AS
SELECT r.id AS review_id, r.comment, r.rating, r.created_at, b.title AS book_title, u.username AS reviewer
FROM "Books_review" r
JOIN "Books_book" b ON r.book_id = b.id
JOIN "auth_user" u ON r.user_id = u.id
ORDER BY r.created_at DESC
LIMIT 10;
"""
),
]
PROBLEMS
OUTPUT
TERMINAL
PORTS
SQL CONSOLE
COMMENTS
DEBUG CONSOLE
(GoodreadsVenv) ayseguldemirbilek@Ayses-MacBook-Pro Goodreads % python manage.py migrate
Operations to perform:
Apply all migrations: Books, admin, auth, contenttypes, sessions
Running migrations:
Applying Books.0004_shelfbooks... OK
(GoodreadsVenv) ayseguldemirbilek@Ayses-MacBook-Pro Goodreads %
Ln 107, Col 1
Spaces: 4
UTF-8
LF
Python
3.9.13 64-bit ('3.9.13': pyenv)
Go Live
Chocola