

Live Poets Society

Group 10

Mona Gandhi Nimay Kumar Rohan Saraogi Zhen Huang



Basic Problem & Goals

Create a social cataloging website for poetry books that allows users to

- Search for books, series, authors etc.
- Maintain their own library
- Answer questions like:
 - "Which poetry books should I read next?"
 - "Which authors are most liked/disliked?"

Dataset

Researchers at UCSD prepared datasets by scraping Goodreads in 2017 (later updated in 2019).

- Poetry Books
- Poetry Reviews
- Goodreads Series
- Goodreads Authors
- Poetry User Library Interactions

Preprocessing

- Fake user creation: Username, password, name, email
- **Scraping**: For book cover images
- Format conversion: JSON to CSV
- Subsetting: Series & Author datasets
- **Projection**: Remove duplicate columns, columns with descriptive statistics etc.
- String Processing: Handling quotes/whitespaces, datetime conversions etc.
- Removing/replacing illogical values: eg. negative no. of comments for reviews
- Ensuring foreign key constraints are met



Schema & ER Diagram

Book (<u>id</u>, title, description, language_code, edition, format, is_ebook, isbn, isbn I 3, asin, kindle_asin, publisher, publish_date, num_pages, image_url)

Similar_Books (book id1, book id2)

Series (id, title, description, numbered)

In_Series (book id, series id)

Author (id, name)

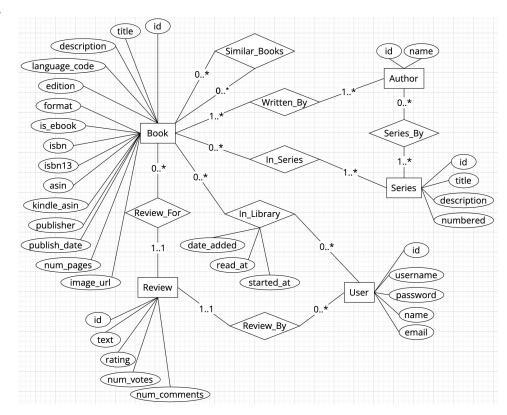
Written_By (book_id, author_id)

Series_By (series_id, author_id)

User (id, username, password, name, email)

In_Library (user_id, book_id, date_added, read_at,
started_at)

Review (<u>id</u>, user_id, book_id, text, rating, num_votes, num_comments)





Complex Queries

Book Recommendations

Returns 16 book recommendations that a user hasn't read, based on the tastes of similar users. If there isn't enough user information to recommend 16 books, also recommends most popular books (based on review count).

Author Statistics

For each author, returns general statistics about the author, such as the no. of books they have written, no. of reviews/ratings, average rating etc.

Author Perceptions

For each author, returns the no. of users that like, are neutral towards, and dislike the author

Top Reviewers

Returns information about users with the highest no. of reviews, such as their username, no. of books in their library, no. of their ratings/reviews, average rating etc.



Performance & Optimization (Part 1)

Observation: Three complex queries have a (common) costly setup

Book Recommendations

```
WITH users_authors AS (

SELECT U.id AS u_id,

U.name as user_name,

A.id as a_id,

A.name as author_name,

AVG(rating) AS rating

FROM Author A

JOIN Written_By WB ON A.id = WB.author_id

JOIN Review R ON WB.book_id = R.book_id

JOIN User U on U.id = R.user_id

GROUP BY A.id, A.name, U.id, U.name

ORDER BY U.name ASC, rating DESC),

top_authors AS (

SELECT a_id FROM users_authors

...
```

Author Perceptions

```
WITH users authors AS (
 SELECT U.id AS u id,
     U.name as user name,
     A.id as a id,
     A.name as author name,
     AVG(rating) AS rating
 FROM Author A
 OIN Written By WB ON A.id = WB.author id
 JOIN Review R ON WB.book id = R.book id
 OIN User U on U.id = R.user id
 GROUP BY A.id, A.name, U.id, U.name
 ORDER BY U.name ASC, rating DESC),
dislikes AS (
 SELECT a id, author name, COUNT(DISTINCT
u id) AS num dislikes
 FROM users authors
 WHERE rating < 2.0
```

Author Statistics



Performance & Optimization (Part 1)

Observation: Three complex queries have a (common) costly setup **Solution**: "Simulate" a materialized view for the setup and adapt queries to use the view

View

```
CREATE TABLE ABRU_View(

SELECT A.id author_id,

A.name author_name,

R.book_id book_id,

R.id review_id,

R.user_id user_id,

R.rating rating

FROM Author A

JOIN Written_By WB on A.id = WB.author_id

JOIN Review R on WB.book id = R.book id);
```

Index

```
CREATE INDEX AU_Index
ON ABRU_View(author_id, user_id);
```

CREATE INDEX Rating_Index ON ABRU View(rating);

Performance & Optimization (Part I)

	Before ABRU_View	After ABRU_View		
Query		Before AU_Index	After AU_Index	After Rating_Index
Recommendation	20.34	2.45	1.53	1.51
Author Perceptions	12.18	3.36	2.16	2.20
Author Statistics	96.42	1.21*	0.3*	0.31*

^{*} After adapting query to use ABRU_View



Performance & Optimization (Part 2)

Observation: Split a complex query into reusable components

Query	Unoptimized	Optimized
Top Reviewers	22.19	2.32



Technical Challenges

- **Dataset**: Finding a dataset large enough to be challenging but still feasible to work with
- Preprocessing: Scraping, fake user creation
- Complex queries: Creating complex queries that meaningfully contribute to the application
- Query optimization:
 - MySQL deprecated query caching
 - No materialized views

Thank you!

