E-Book Management System- Database Design and Implementation

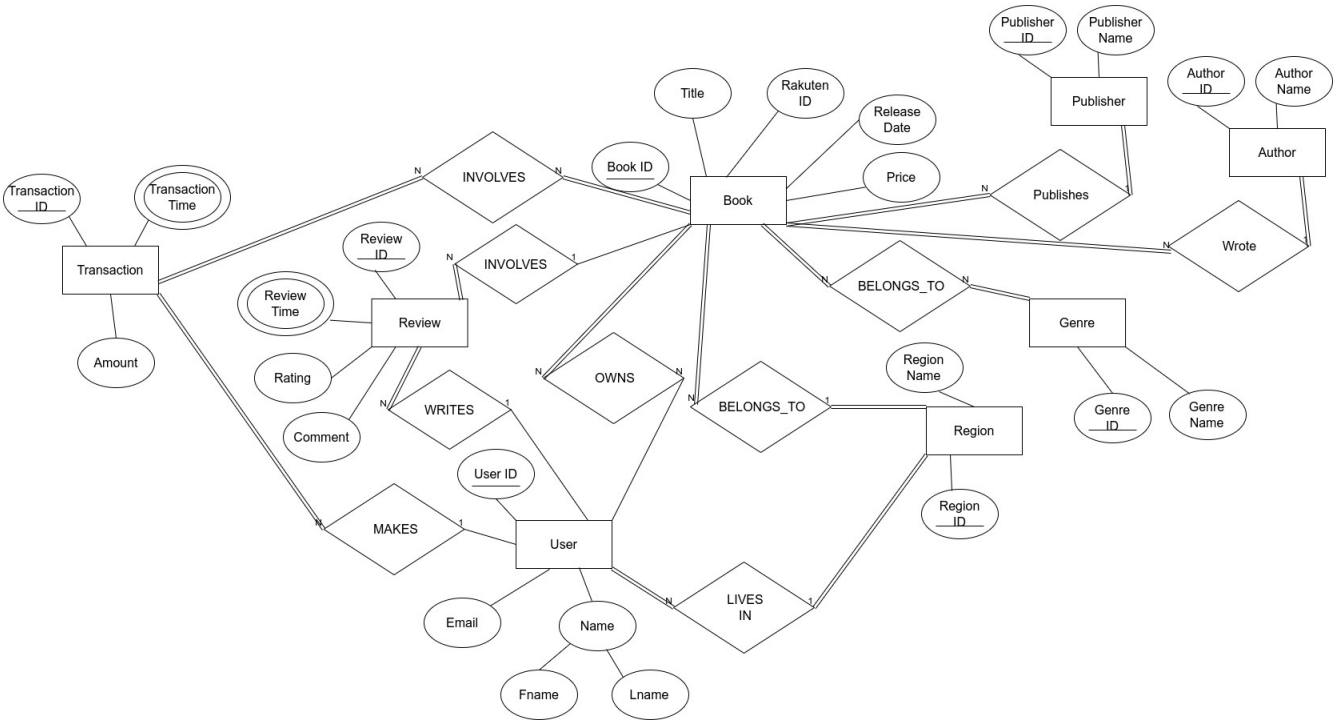
Ruilin Peng(Group 24)

ECE9014 Data Management and Application

1. Introduction

The motivation arose from my personal interest in how Rakuten Kobo, an Ebook selling platform, managed its data- ebooks, users, etc. As the ebook platform that originated in Toronto and has the highest market share in Canada, Rakuten Kobo has one key feature that differentiates it from other famous platforms such as Amazon Kindle. Instead of having different domains to manage services in different regions, Rakuten Kobo only has one which is kobo.com, which means it is also possible that it has been using a centralized database to manage data across all service regions. Given that I am an individual who understands 3 different languages- English, Chinese, and Japanese(corresponding to the major service regions of Rakuten Kobo- North America, Japan, Taiwan, and Hong Kong), the challenge of managing data across regions intrigued me. In order to implement a database management system that is modeled after Rakuten Kobo, I am using draw.io to plot ER diagrams, locally hosted MariaDB to perform as a database, and Python with libraries such as Zenrows and Beautiful Soup to web scrape from Rakuten Kobo's pages, since I believe using real data web scraped would better validate the database design.

2. Database Design



The Entity-relationship Diagram above is the full design I came up with and the explanation is as follows:

Entities:

1. **Book**
    - **Attributes**: *Book_ID* as primary key, *Title*, *Rakuten_ID*, *Release_Date*, *Price*.
    - Represents ebooks in the system.
    - The assumption is each ebook is associated with one region, one author, and one publisher.
    - An ebook can be in multiple genres.
2. **Author**
    - **Attributes**: *Author_ID* as primary key, *Author_Name*.
    - Represents an author who can write multiple books
3. **Publisher**
    - **Attributes**: *Publisher_ID* as primary key, *Publisher_Name*.
    - Represents a publisher that can publish multiple ebooks
4. **Region**
    - **Attributes**: *Region_ID* as primary key, *Region_Name*.

- Represents a region that can have multiple ebooks

5. **Genre**
    - **Attributes**: *Genre_ID* as primary key, *Genre_Name*.
    - Represents a genre that can have multiple ebooks.
6. **User**
    - **Attributes**: *User_ID* as primary key, *Email*. *Fname*, *Lname*.
    - Represents a user(buyer only) who can own ebooks, buy ebooks, and write reviews.
    - Each user can only belong to one region.
7. **Transaction**
    - **Attributes**: *Transaction_ID* as primary key, *Transaction_Time*. *Amount*.
    - Represents a purchase conducted by a user and one or more ebooks.
8. **Review**
    - **Attributes**: *Review_ID* as primary key, *Review_Time*. *Rating*, *Comment*
    - Represents a review written by a user of an ebook.

Relationships:

1. **WROTE(Author → Book)**
    - **Type**: One-to-Many
    - An Author can write multiple books but a book can only be written by one author.
2. **Publish(Publisher → Book)**
    - **Type**: One-to-Many
    - A Publisher can publish multiple books but a book can only be published by one publisher.
3. **BELONGS_TO(Book → Region)**
    - **Type**: Many-to-One
    - Each book is associated with one region, but a region can include multiple books.
4. **BELONGS_TO(Book → Genre)**
    - **Type**: Many-to-Many
    - Each book can be in multiple genres, and so is each genre can have multiple books.
5. **LIVES_IN(User → Region)**

- **Type**: Many-to-One
- Each user can only be in one region and each region can have multiple users.

6. **OWNS(User → Book)**
    - **Type**: Many-to-Many
    - A user can own multiple books, and a book can be owned by multiple users.
7. **MAKES(User → Transaction)**
    - **Type**: One-to-Many
    - A user can make multiple transactions, but each transaction is linked to one user.
8. **INVOLVES(Transaction → Book)**
    - **Type**: Many-to-Many
    - A user can make multiple transactions, but each transaction is linked to one user.
9. **WRITES(User → Review)**
    - **Type**: One-to-Many
    - A user can write multiple reviews, but each review is linked to one user.
10. **INVOLVES(Review → Book)**
    - **Type**: Many-to-One
    - A review is written for one book, but a book can have multiple reviews.

3. Implementation in MariaDB

To start with, the database is created using the command line in the bash terminal:

```
MariaDB [(none)]> CREATE DATABASE ebook1;
Query OK, 1 row affected (0.007 sec)
MariaDB [(none)]> GRANT ALL PRIVILEGES ON ebook1.* TO 'ruilin'@'localhost' identified by 'passwd';
Query OK, 0 rows affected (0.010 sec)
MariaDB [(none)]> FLUSH PRIVILEGES ;
Query OK, 0 rows affected (0.012 sec)
```

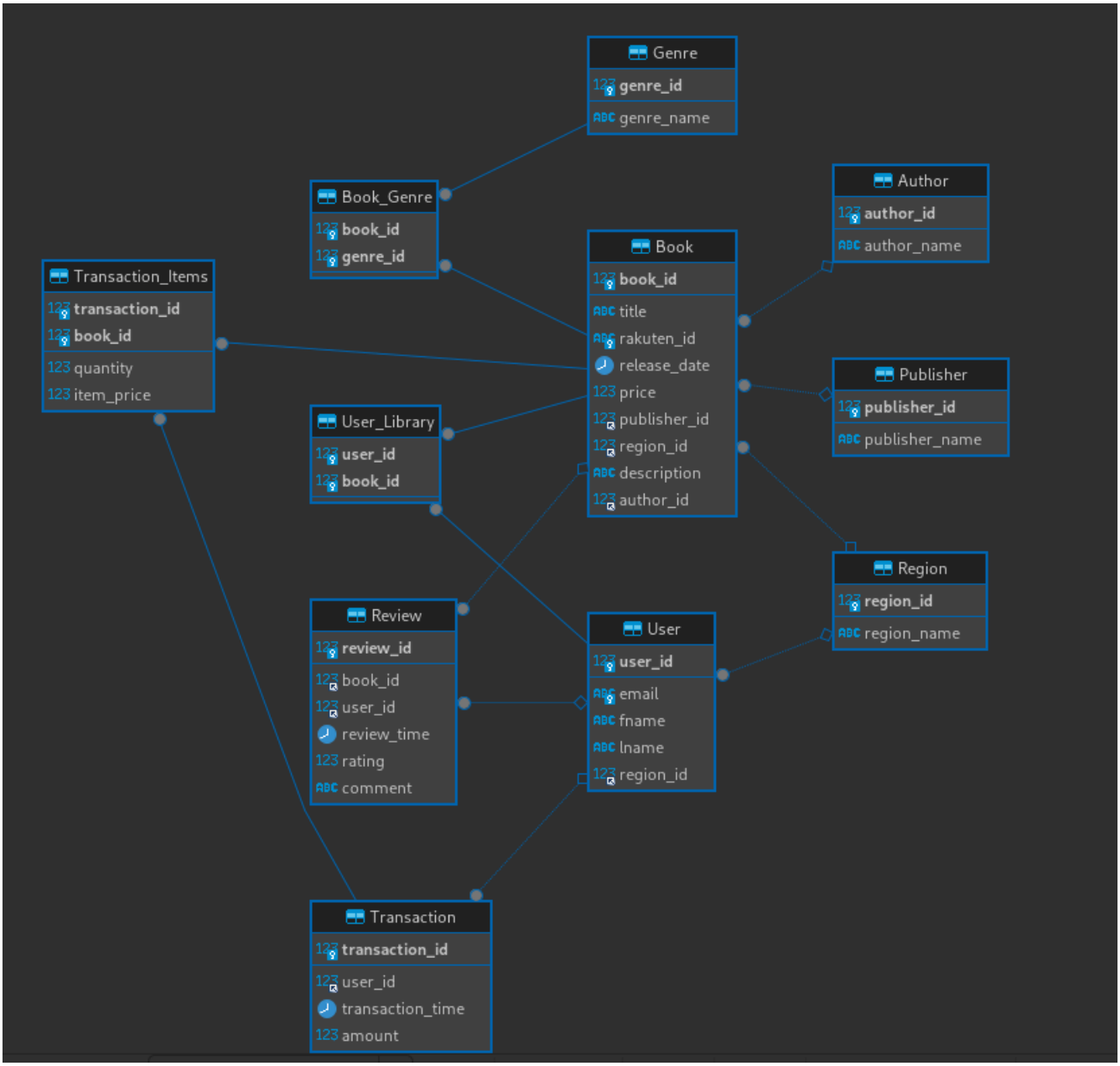
Note that software such as DBeaver doesn’t allow connection to MariaDB using the root user, so a flush of privileges to another user is necessary and connection can then be done using that user.

After the database is created and connected to DBeaver, the creation of tables, and constraints can be done using SQL scripts like below.

```sql
CREATE TABLE Book (
    book_id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(200) NOT NULL,
    rakuten_id VARCHAR(20) UNIQUE,
    release_date DATE,
    price DECIMAL(10, 2)
);

CREATE TABLE Publisher (
    publisher_id INT PRIMARY KEY AUTO_INCREMENT,
    publisher_name VARCHAR(100) NOT NULL
);

CREATE TABLE Genre (
    genre_id INT PRIMARY KEY AUTO_INCREMENT,
    genre_name VARCHAR(100) NOT NULL
);

CREATE TABLE Region (
    region_id INT PRIMARY KEY AUTO_INCREMENT,
    region_name VARCHAR(100) NOT NULL
);

ALTER TABLE Book
ADD COLUMN publisher_id INT,
ADD COLUMN region_id INT;

ALTER TABLE Book
ADD CONSTRAINT fk_publisher
FOREIGN KEY (publisher_id) REFERENCES Publisher(publisher_id);

ALTER TABLE Book
ADD CONSTRAINT fk_region
FOREIGN KEY (region_id) REFERENCES Region(region_id);
```

The key point to note during this step would be the handling of relationships. For a One-To-Many or Many-To-One relationship, like Book-Region(Many-To-One), a foreign key(the primary key of the one side, Region's region_id) on the many side(Book) would do the work. However, for a Many-To-Many relationship, a "hash table" is needed to store the keys of both entities as foreign keys. For example, Book-Genre(Many-To-Many) would need an additional "hash table" like below:

```sql
CREATE TABLE Book_Genre (
    book_id INT,
    genre_id INT,
    PRIMARY KEY (book_id, genre_id),
    FOREIGN KEY (book_id) REFERENCES Book(book_id),
    FOREIGN KEY (genre_id) REFERENCES Genre(genre_id)
);
```



And the final ER diagram would look like this.

4. Data Collection-Web scraping into CSVs

Some of the data such as Book, Author, and Publisher can actually be web-scraped from kobo.com. How it works is that after fetching the response of a book's page, we can fetch the information such as title, author, publisher, price, etc from the HTML into a CSV and then import the CSV into the database in the next step. In the Python file, we will be using ZenRows, a scraping toolkit, to fetch the response, since otherwise a bare header pretending to be some browser request would not be good enough and blocked as a bot. After we get the response in HTML, we need to use a library called Beautiful Soup to query the information we need.

```
title = soup.find('h1', class_='title product-field').get_text(strip=True)
```

For example, sometimes, the information is in an HTML that has unique class and it would be easy to query out and get text within it.

```
book2ndmeta = soup.find('div', class_="bookitem-secondary-metadata")
table1 = book2ndmeta.find_next('ul')
line = table1.find_all('li')[3]
rakuten_id = line.find_next('span').text.strip()
```

Other times, it involves manually inspecting the HTML response and going from parent HTML elements to child HTML elements layer by layer.

```
if "/jp/ja/" in url:
region_id = 3 # Japan
date_format = "%Y 年%m 月%d 日" # Japanese date format
elif "/tw/zh/" in url:
region_id = 4 # Taiwan
date_format = "%Y 年%m 月%d 日" # Taiwanese date format
else:
region_id = 2 # United States
date_format = "%B %d, %Y" # English date format
```

While I left titles from different regions not translated, date is a column that needs to be processed or otherwise cannot be used in the database. By default, it would be a text no matter which region it it. For example, in English, “November 25, 2024” would be transformed into 2024-11-25, a date format. Similarly, in Chinese or Japanese, an original date text like “2015 年(nian/ねん)5 月(yue/がつ)26 日(ri/にち)” also needs to be transformed into 2015-05-26. Also, in order to tell which region the current book is, the path parameters of the URL(uniform resource locator) would tell us that. For example, https://www.kobo.com/tw/zh/ebook/zIMDg4NARzqLd9vYg8bGXQ

the tw/zh means the current book is in Taiwan and in Chinese. Similarly, the “price” is also converted into numeric from text.

```
books = []


books.append({
"book_id": book_id_counter,
"title": title,
"rakuten_id": rakuten_id,
"release_date": release_date,
"price": price,
"publisher_id": publisher_id,
"region_id": region_id,
"description": description,
"author_id": author_id,
})


book_df = pd.DataFrame(books)
```

```
book_df.to_csv('books_web_scraped.csv', index=False, encoding='utf-8-sig')
```

And in order to output to CSV, for each piece of information, we store them in a list first, then convert them into a pandas data frame, and finally output to CSV files.

5. Data Importing

```
LOAD DATA LOCAL INFILE
'/home/ruilin/Desktop/ECE9014/Rakuten/books_web_scraped.csv'
INTO TABLE Book
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(book_id, title, rakuten_id, release_date, price, publisher_id, region_id,
description, author_id);
```

Assume we have all the data ready in CSV files, the above syntax enables us to import each row from the CSV files to the database tables. The thing that needs to be noticed during this step is the order of importing. If a table has a foreign key from another table, the other table needs to be imported first. For example, a book has author_id from Author as a foreign key, the Author needs to be imported first.

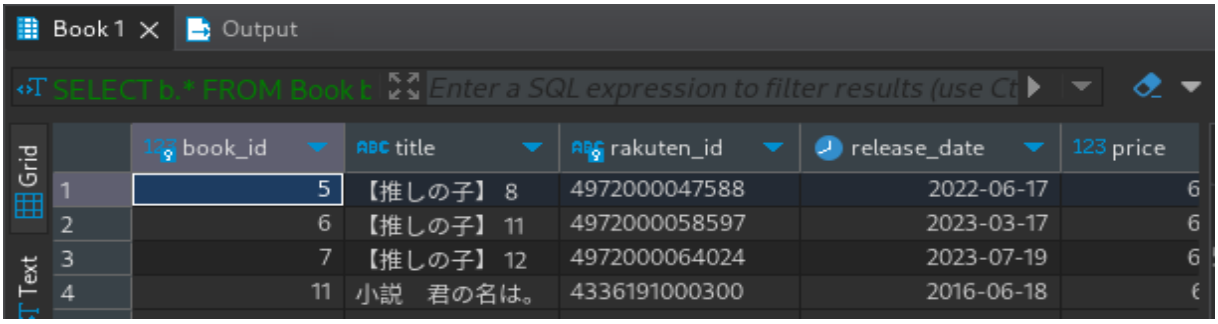
6. Sample Usage

After the population of data into the table, the database is basically completed. Assume in the future we want to build a backend service using this database, and then we need to read and write to the database. Even given that middlewares nowadays can handle all the mapping and translation into SQL syntax, understanding the SQL syntax those middlewares translate into helps us debug. To read, it involves Data Query Language.

- **Filter book by region**

For example, we want all the ebooks that belong to the Japan region. We do the query operation and join,

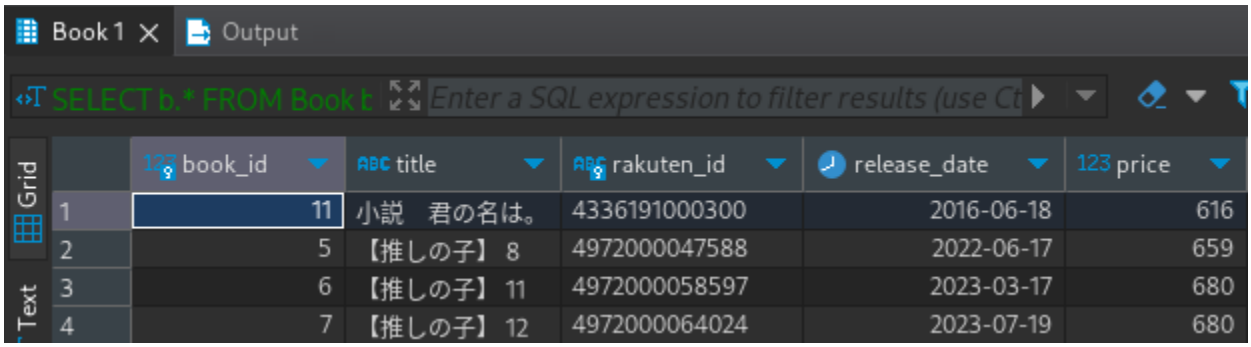
```
SELECT b.*
FROM Book b
JOIN Region r ON b.region_id = r.region_id
WHERE r.region_name = 'Japan';
```



And the operation can give us the filtered result even given that in the table Book, books from different regions are mixed together and not ordered by region in terms of book_id.

```sql
SELECT b.*
  FROM Book b
  JOIN Region r ON b.region_id = r.region_id
 WHERE r.region_name = 'Japan'
 ORDER BY price;
```

Other common features required by backend service apps like sort can also be accomplished using ORDER BY, for example, price from low to high:



- **<u>Filter books by genre</u>**

Since this is a Many-to-Many relationship, we could filter by one or more genres.

```sql
SELECT b.book_id, b.title, g.genre_name
FROM Book_Genre bg
JOIN Book b ON bg.book_id = b.book_id
JOIN Genre g ON bg.genre_id = g.genre_id
WHERE g.genre_name = 'Manga';
```

For example, above we filter all books that can be identified as “Manga”.

```
SELECT b.book_id, b.title, g1.genre_name, g2.genre_name
FROM Book_Genre bg1
JOIN Book_Genre bg2 ON bg1.book_id = bg2.book_id
JOIN Book b ON bg1.book_id = b.book_id
JOIN Genre g1 ON bg1.genre_id = g1.genre_id
JOIN Genre g2 ON bg2.genre_id = g2.genre_id
WHERE g1.genre_name = 'Manga' AND g2.genre_name = 'Romantic';
```

	book_id	title	genre_name	genre_name
1	10	輝夜姫想譲人告白 (9)	Manga	Romantic

To filter using more than one genre tag, the join becomes more complicated, but we need the same number of joins as tags for both the “hash table” and the table containing the tags.

- **<u>Filter book by author or publisher</u>**

```
SELECT b.title, a.author_name
FROM Book b
JOIN Author a ON b.author_id = a.author_id
WHERE a.author_name LIKE '赤坂アカ%';
```

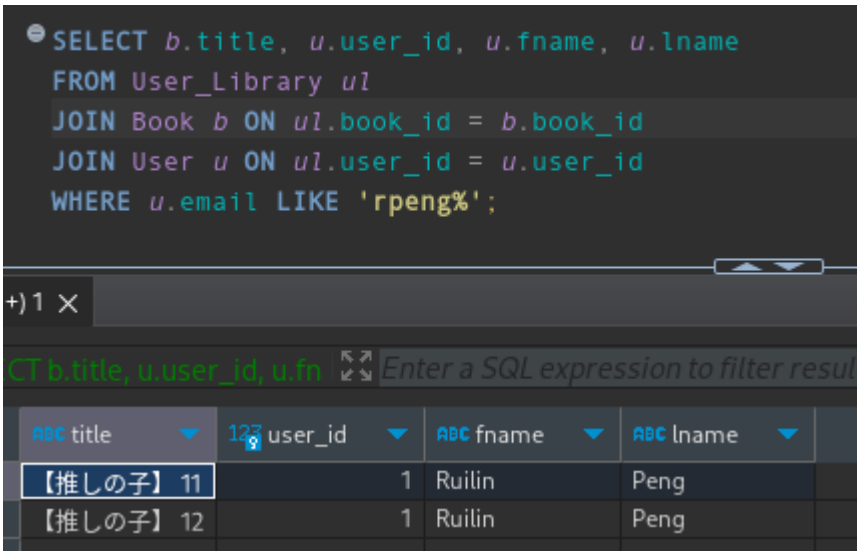
	title	author_name
1	【推しの子】 8	赤坂アカ×横槍メンゴ
2	【推しの子】 11	赤坂アカ×横槍メンゴ
3	【推しの子】 12	赤坂アカ×横槍メンゴ
4	【我推的孩子】(08)	赤坂アカ×横槍メンゴ
5	【我推的孩子】(07)	赤坂アカ×横槍メンゴ
6	輝夜姫想譲人告白 (9)	赤坂アカ

```
SELECT b.title, p.publisher_name
FROM Book b
JOIN Publisher p ON b.publisher_id  = p.publisher_id
WHERE p.publisher_name LIKE '角川%';
```

	title	publisher_name
1	小説　君の名は。	角川文庫

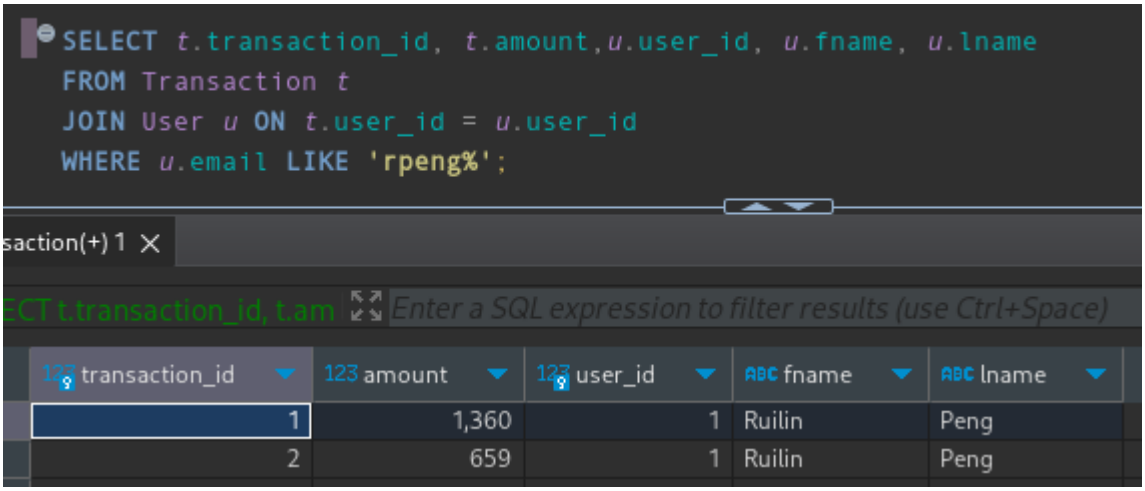
The SQL LIKE Operator is also useful when we don’t know the exact value or there are similar tags.

- **Get the library for a user**



As one of the main features, we would need to check what books are owned by a user. For this Many-To-Many relationship, we join the “hash table” with the 2 tables on each of the “Many side” to filter.

- **Get all transactions(orders) of a user, and details of one of his/her transactions(orders)**



```
SELECT t.transaction_id, b.title, ti.item_price
FROM Transaction_Items ti
JOIN Transaction t ON ti.transaction_id = t.transaction_id
JOIN Book b  ON ti.book_id  = b.book_id
WHERE t.transaction_id = 1;
```

saction(+) 1

ECT t.transaction_id, b.titl Enter a SQL expression to filter results (use Ctrl+Spa

transaction_id	title	item_price
1	【推しの子】 11	680
1	【推しの子】 12	680

In the example above, we retrieve all the orders of this individual called “Ruilin Peng” and the details(including the book name and price of each) of his first order.

- **Get all reviews of a user**

```
SELECT r.review_id, b.title, u.fname, r.rating, r.comment
FROM Review r
JOIN User u ON r.user_id = u.user_id
JOIN Book b ON r.book_id = b.book_id
WHERE u.email LIKE 'rpeng%';
```

v(+) 1

CT r.review_id, b.title, u. Enter a SQL expression to filter results (use Ctrl+Space)

review_id	title	fname	rating	comment
1	【推しの子】 11	Ruilin	5	MEMちょ最高
2	【推しの子】 12	Ruilin	5	ルビーちゃんもかわ

Similarly, we can retrieve all the reviews of that individual.

- **Input new information**

To write to the database, for example, a new publisher, can be added to the database using INSERT operator of Data Manipulation Language.

```
INSERT INTO Publisher(publisher_name)
VALUES
('Ruilin Press')
```

Statistics 1 | Output

Name	Value
Updated Rows	1
Query	INSERT INTO Publisher(publisher_name) VALUES ('Ruilin Press')
Start time	Tue Dec 03 05:10:32 EST 2024
Finish time	Tue Dec 03 05:10:32 EST 2024

And so is any other new information such as book, order, review, etc.

7. Conclusion

In summary, this project demonstrated the feasibility of using a centralized database to manage Ebooks across regions. However, assuming we are building a server app on top of this database management system, heavy service logic is required to perform the right actions(For example, allowing users to only buy ebooks from the region he/she is from, determining what the currency should be for a price).

8. Reference

Beautiful Soup documentation¶. Beautiful Soup Documentation - Beautiful Soup 4.4.0 documentation. (n.d.). https://beautiful-soup-4.readthedocs.io/en/latest/

Web scraping API & data extraction - zenrows. (n.d.). https://www.zenrows.com/