MTQGA

COS221 Assignment 5

MAKE THE QUERY GREAT AGAIN

23.05.2025

Cailin Smith (u24570525)
Jordan Naidoo (u24664155)
Kyle McCalgan (u24648826)
Marcel Stoltz (u24566552)
Johan Coetzer (u24564584)

Task 1: Research . . . . . .

General overview and explanation

Electronic commerce refers to the buying and selling of goods over the internet [7]. This industry has experienced significant growth since its inception, especially with its rise in popularity in the 21st century. Product categories such as electronics, clothing, books, and entertainment remain popular on these online sites. Alongside the growing demand for digital storefronts, beneficial utilities emerged; most notably price comparison tools [8]. These services help customers save money, provide transparency on profit margins, and supply useful reviews from past customers [8]. These tools have considerably impacted the industry by raising price competition between retailers, thus making pricing strategies essential. Pricespy, Pricerunner, and Skinflint are among the growing list of these deal-hunting services [6].

Information on How Content is Rated or Categorised in Online Shopping

In the retail and ecommerce industry, content - specifically product information - is rated and categorised with the intention to improve user experience, enhance product discovery, and support customers in their purchasing decisions [5]. Product categorisation groups products into hierarchical categories and subcategories. For example, books can be categorized into groups such as fiction, academic works, and children's literature [9]. Content rating is another method where reviews and ratings are user-generated based on written feedback and/or star ratings. Examples include the five-star system, verified purchase reviews, and weighted reviews [3]. Books generally have both user ratings as well as third-party ratings like the "NY-Times Bestseller" list, which evaluates the success of newly released books [2]. Metadata and attributes such as ISBN, language, format, and publication dates are also used to support comparison and filtering tools [9].

User Experience Importance (Additional features e.g. Reviews)

User experience plays a crucial role in the success of online shopping platforms [5]. Reviews and ratings on products can greatly influence a customer's purchasing decisions. Additional features, such as user reviews, allow consumers to view feedback from others. Book reviews provide buyers with more information about the story, book tropes, and the overall quality, potentially sparking their interest and convincing them to buy and read the book [9]. Reviews can also influence which store a consumer will purchase from, as stores themselves often receive reviews on quality and experience [4]. Displaying recommendations to users based on their buying and browsing history further personalises the shopping experience and makes it more engaging [5].

Different types or categories of Products and retailers

E-commerce platforms have the ability to sell products both physically and digitally, unlike physical retail stores that are limited to physical forms of products [7]. For example, in the case of books, e-commerce platforms sell hard copy versions as well as electronic books [9]. Within e-commerce spaces, there are various types of retailers, such as general online retailers, niche product stores, and direct-to-consumer stores [6]. In the book industry, general retailers include platforms like Amazon that sell all types of books, niche stores such as BetterWorldBooks that sell books from less popular genres[1], and some publishers that sell their books directly to the public through their own online stores [2]. Products are grouped into categories to improve customer experience when searching for similar products. In the book industry, these categories are referred to as genres. Popular genres include children's books, educational books, religious books, and various types of fiction [9].

Referencing

[1] BetterWorldBooks, 2024. *BetterWorldBooks.com*. [online] Available at: https://www.betterworldbooks.com/ [Accessed 26 April 2025].

[2] Book Website, 2024. *The Top 10 Best Online Bookstores*. [online] Available at: https://book-website.com/the-top-10-best-online-bookstores/ [Accessed 26 April 2025].

[3] Delighted, 2024. *E-commerce Trends*. [online] Available at: https://delighted.com/blog/ecommerce-trends [Accessed 28 April 2025].

[4] First Atlantic Commerce, 2024. *How E-commerce Changed Traditional Retail*. [online] Available at: https://firstatlanticcommerce.com/blog-fac/how-e-commerce-changed-traditional-retail/ [Accessed 26 April 2025].

[5] Go-Globe, 2024. *E-commerce and Online Shopping Stats*. [online] Available at: https://www.go-globe.com/e-commerce-and-online-shopping-stats/ [Accessed 26 April 2025].

[6] Shopify, 2024. *10 Best Comparison Shopping Engines to Increase E-commerce Sales*. [online] Available at: https://www.shopify.com/za/blog/7068398-10-best-comparison-shopping-engines-to-increase-ecommerce-sales [Accessed 26 April 2025].

[7] Statista, 2024. *Online Shopping - Statistics & Facts*. [online] Available at: https://www.statista.com/topics/871/online-shopping/ [Accessed 28 April 2025].

[8] TGN Data, 2024. *Price Comparison for Online Retailers: A Comprehensive Guide*. [online] Available at: https://tgndata.com/price-comparison-for-online-retailers-a-comprehensive-guide/ [Accessed 28 April 2025].

[9] The Urban Writers, 2024. *Popular Book Genres for Your Next Bestseller*. [online] Available at: https://theurbanwriters.com/blogs/publishing/popular-book-genres-for-your-next-bestseller [Accessed 28 April 2025].

Task 2: (E)ER-Diagram

Below is our final EER Diagram. Please see the attached images in our project files for better quality visuals.

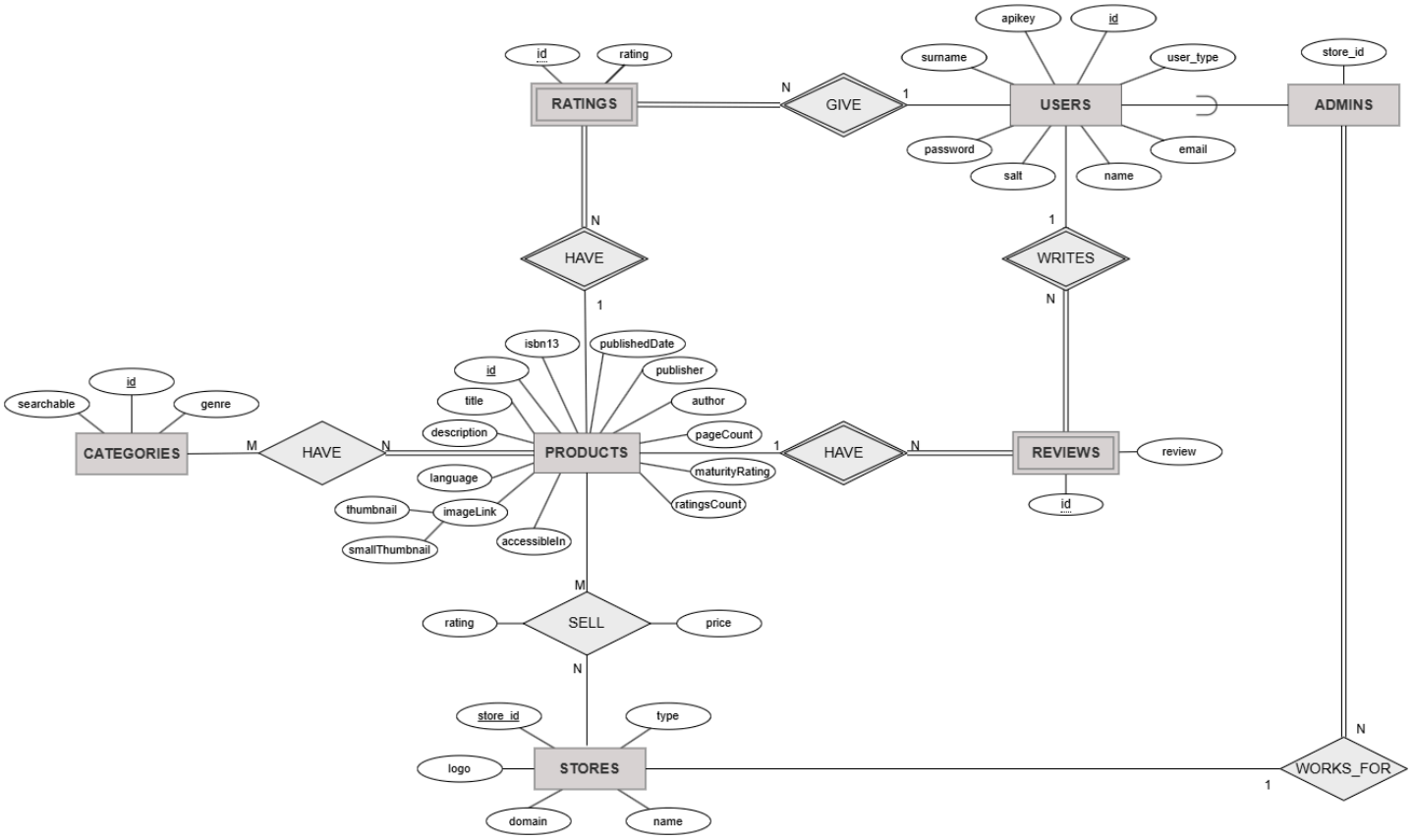
Assumptions and Information

The `SELL` relationship (`PRODUCTS` to `STORES`) is a partial participation on both sides, as not all products loaded into the database will have prices from a store, and not all stores will have prices on products. The reason being, that an admin for a store could remove all the prices on their books, but the store will still be in the database, even though they will have no entries in the `SELL` relationship table (`STORE_INFO`). In addition, a book may be added to the database, without adding a price from a store simultaneously, resulting in the book not being in the `STORE_INFO` table yet.

Users can only have a single rating and review on a book. However, they can have one *or* the other, hence why the `RATINGS` and `REVIEW` entities are separate.

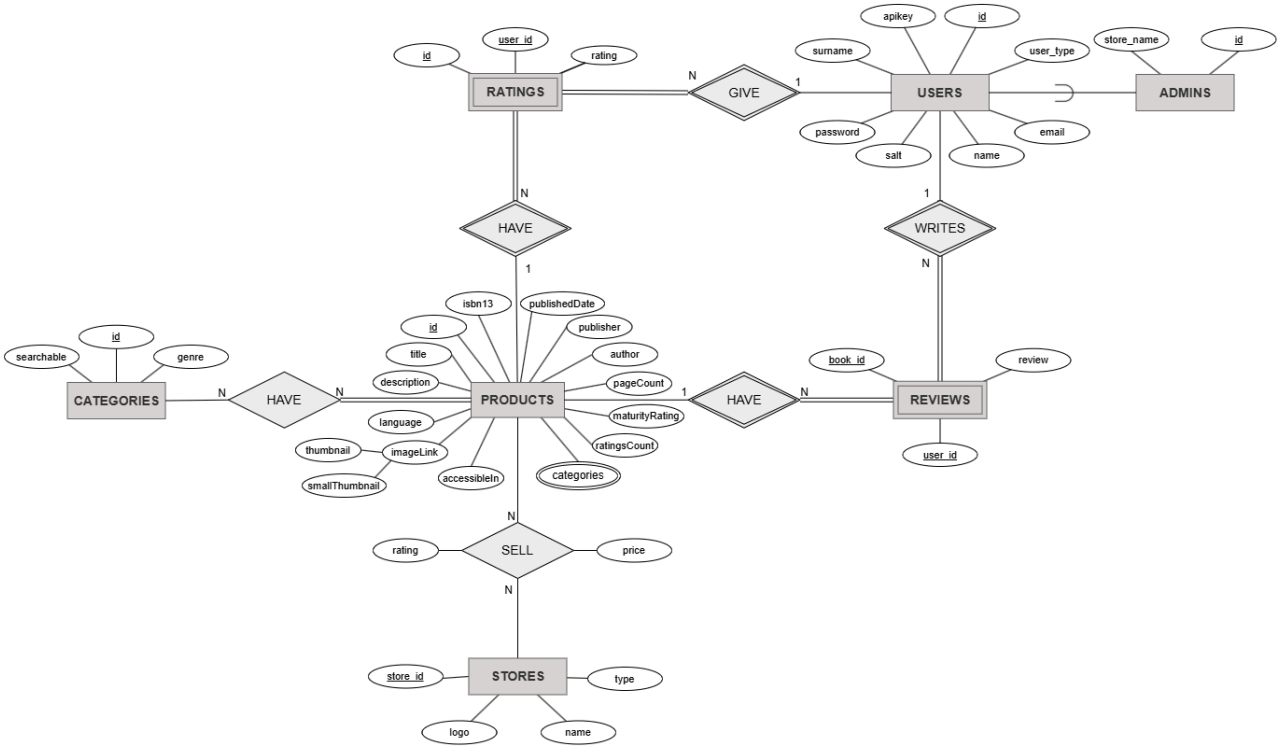
`CATEGORIES` was made a separate entity instead of a multivalued attribute of `PRODUCTS`. This was done because categories will have it's own attribute '`searchable`'. This attribute is changed by our superadmin and determines what categories user's can filter the books by.

Final EER Diagram

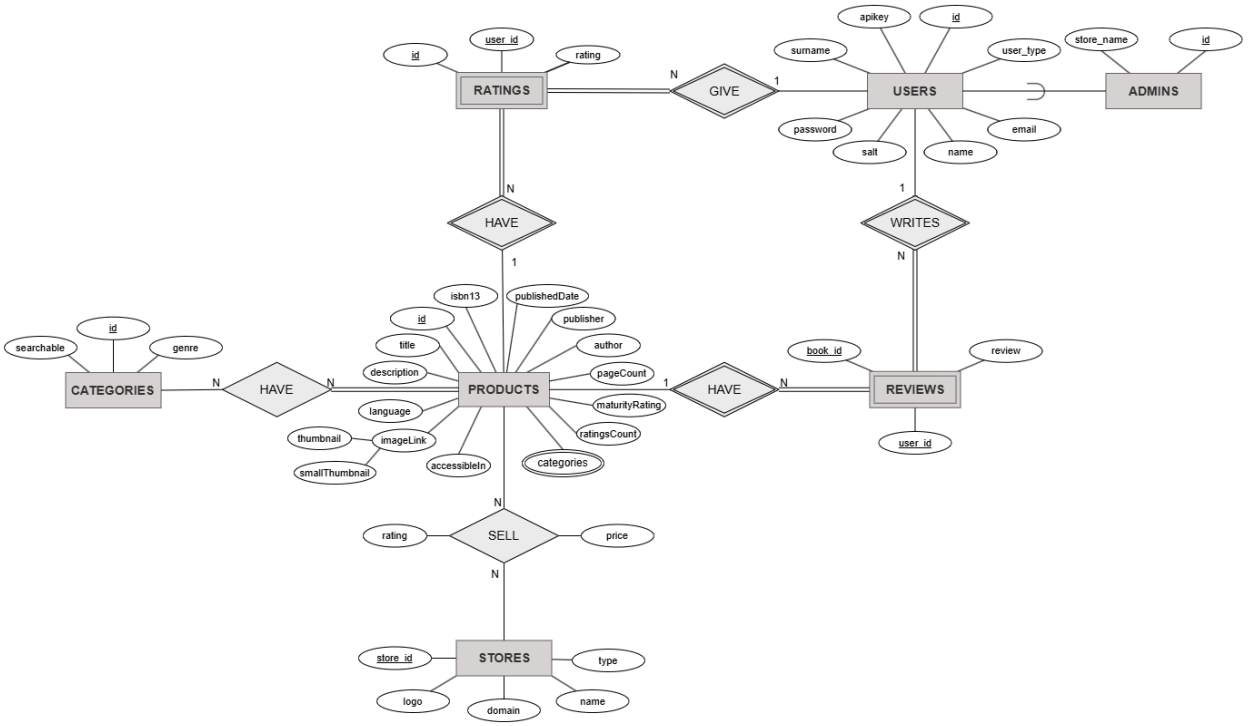


Iterations

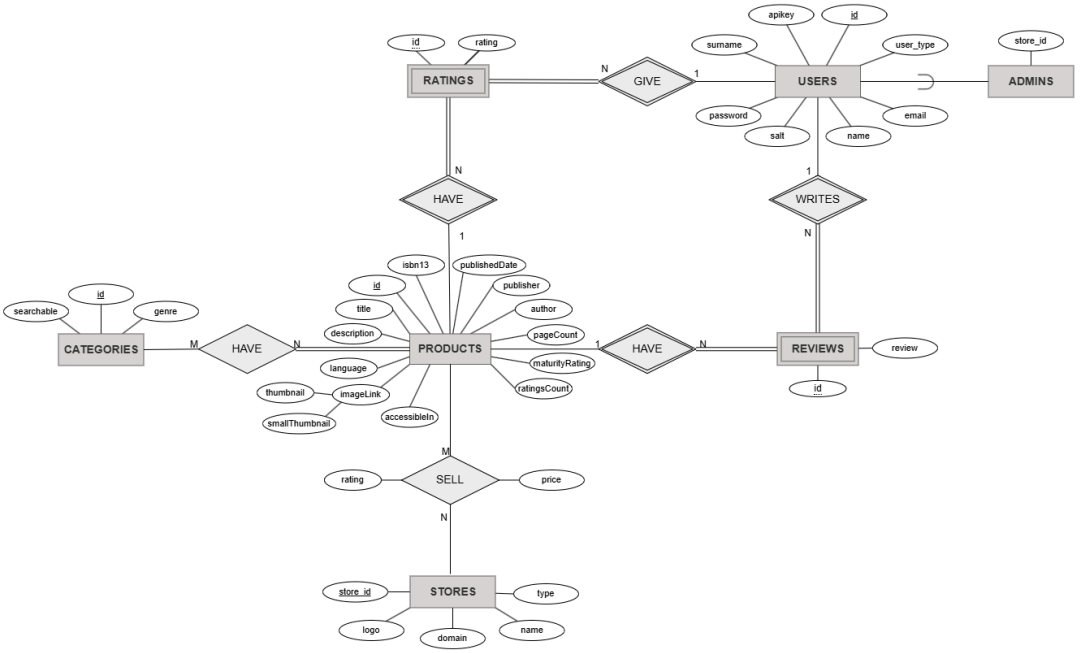
1. Below is our initial iteration.



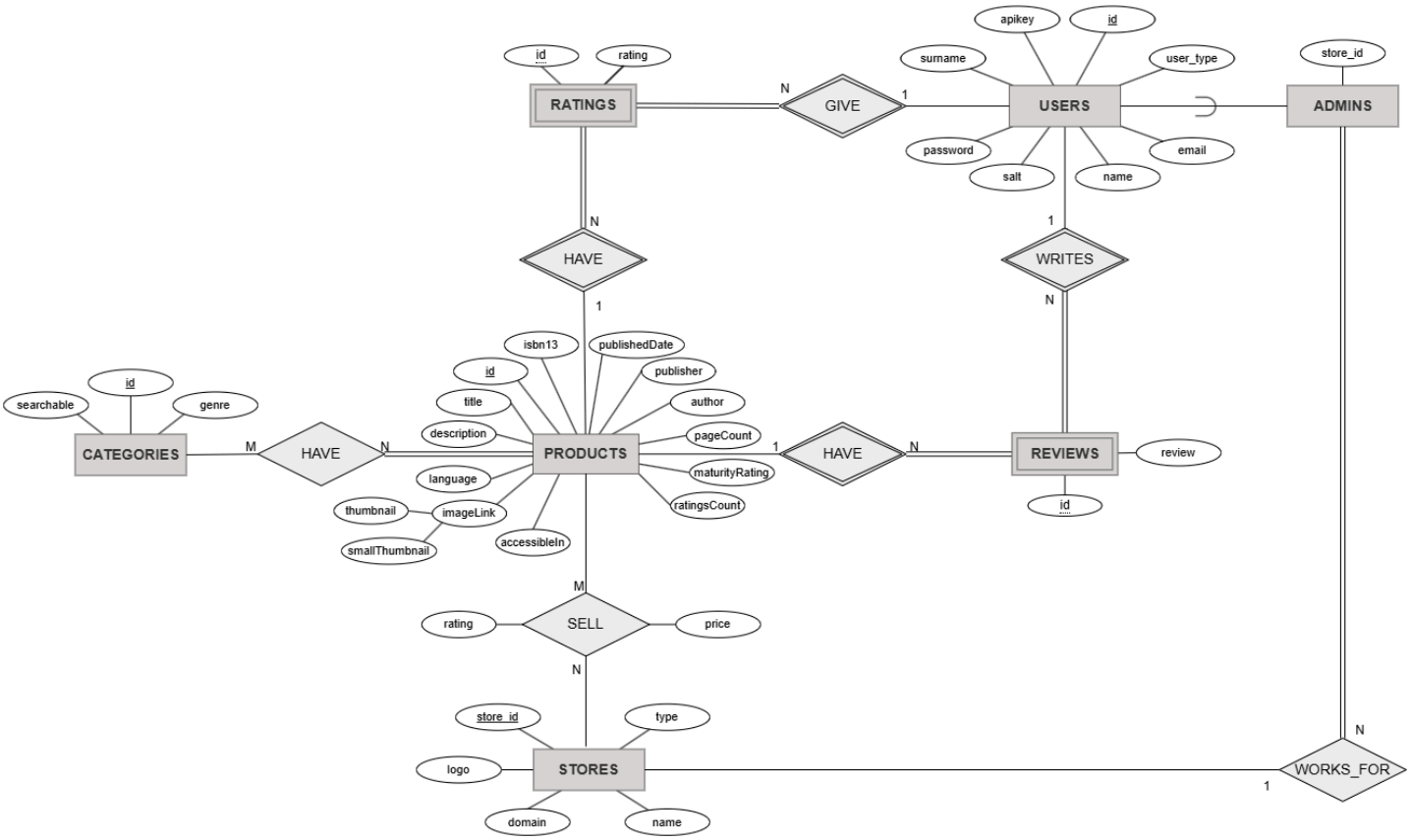
2. Our second iteration (below) added an additional attribute to `STORES`, called '`domain`'. This attribute contains the links to the stores' websites.



3. The third iteration removed the '`book_id`' and '`user_id`' attributes from the `REVIEWS` and `RATINGS` entities, as these are the attributes of these weak entities' owners, and thus are meant to be added as part of the relational mapping steps. Partial keys called '`id`' were added instead.
   The final iteration also removed the '`categories`' multivalued attribute, as the relationship to the entity `CATEGORIES` accounts for this attribute and its connection to `PRODUCTS`. Having the multivalued attribute, as well as the relationship to the entity was unnecessary.



4. The final iteration, which was our final EER Diagram, added the relationship between `STORES` and `ADMINS`. Admins are users that work for a specific store and can add prices and ratings to books for that store.



Task 3: (E)ER-diagram to Relational Mapping . . . . . .

All of the steps are included in the project files, for better quality viewing.

Step 1: Mapping of regular (strong) entity types

For each regular entity, create a relation, add the simple attributes to this relation and choose a primary key.

USERS

id	apikey	name	surname	email	password	salt	user_type

PRODUCTS

id	title	isbn13	description	language	accessibleIn	thumbnail	smallThumbnail	ratingsCount	maturityRating	pageCount	author	publisher	publishedDate

STORES

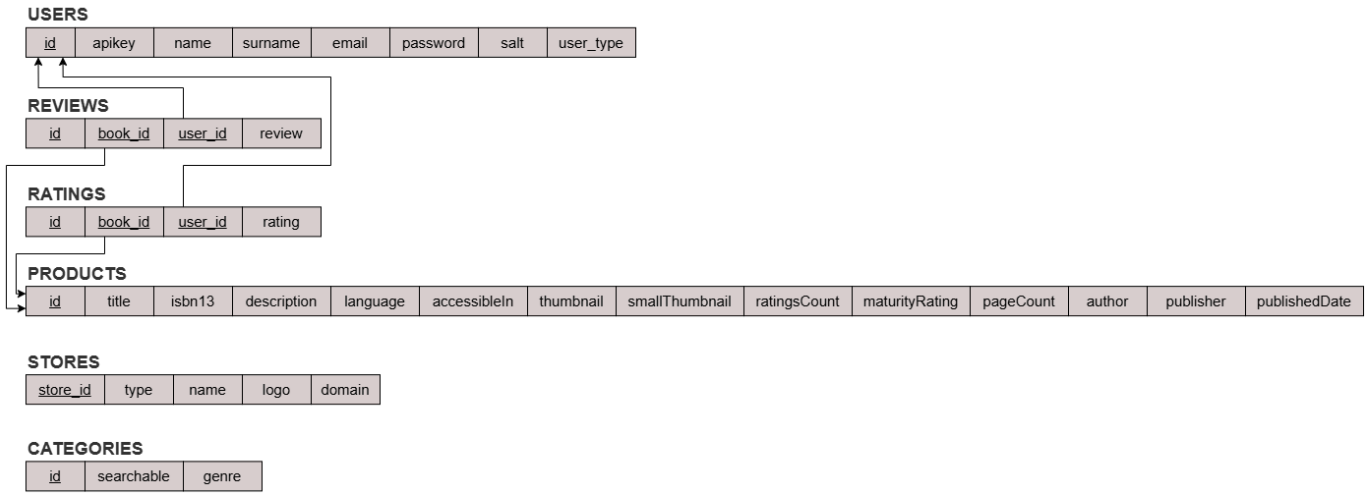
store_id	type	name	logo	domain

CATEGORIES

id	searchable	genre

Step 2: Mapping of weak entity types

For each weak entity type, create a relation, add the simple attributes, include the owner entity types' primary keys as foreign key attributes, and create the primary key from the partial key of the weak entity, and the primary keys of the owners.
`REVIEWS` and `RATINGS` are the weak entity types, and their owners are `USERS` and `PRODUCTS`.

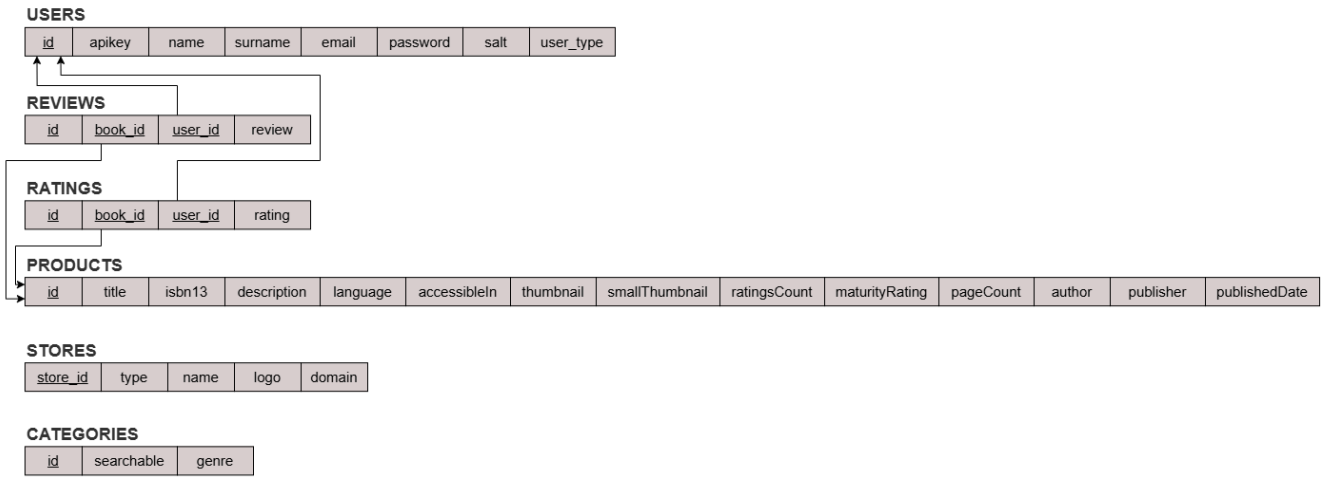


Step 3: Mapping of binary 1:1 relationships

We have no binary 1:1 relationships in our diagram. Thus, this step is not included.

Step 4: Mapping of binary 1:N relationships

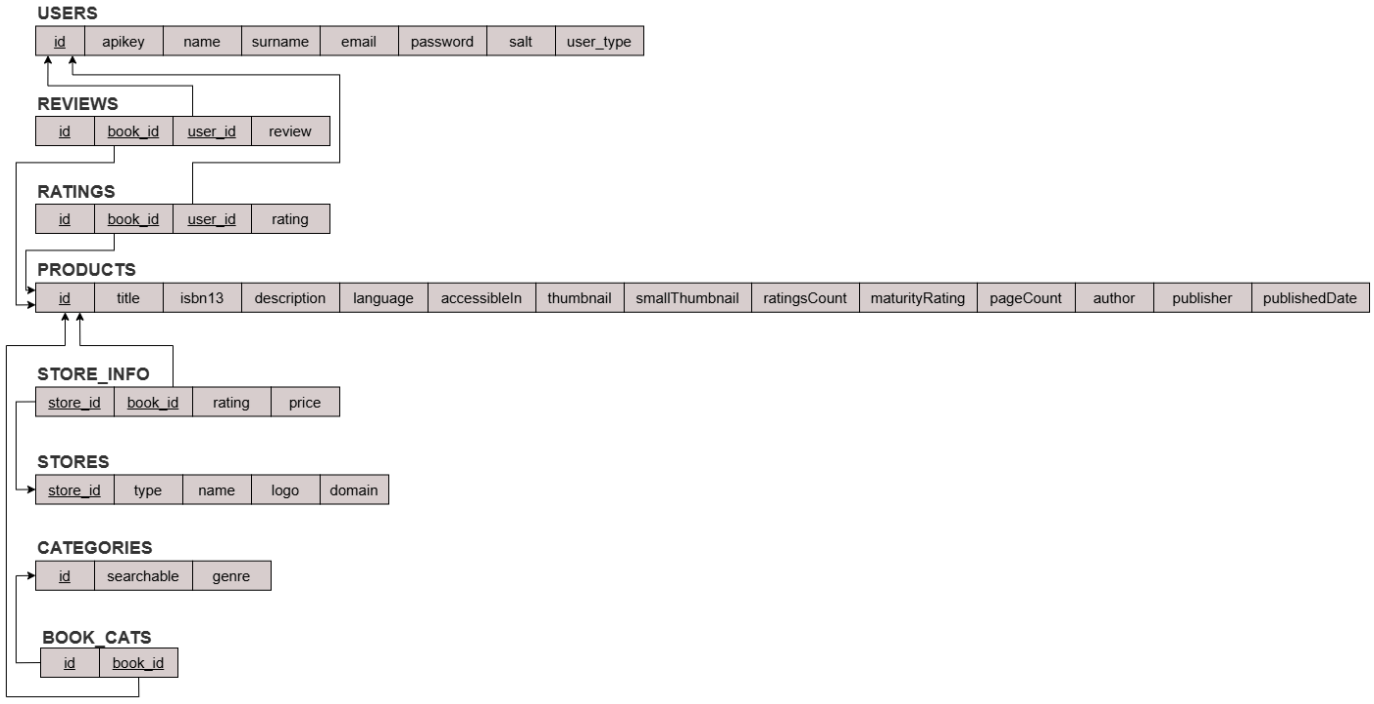
We have four binary 1:N relationships in our diagram between `USERS` and `RATINGS`, `PRODUCTS` and `RATINGS`, `USERS` and `REVIEWS`, and `PRODUCTS` and `REVIEWS`. However, these relationships were mapped in step 2, as `REVIEWS` and `RATINGS` are weak entity types. Thus, no changes were needed for this step.



Step 5: Mapping of binary M:N relationships

We have two binary M:N relationships in our EER Diagram: `CATEGORIES` and `PRODUCTS`, and `PRODUCTS` and `STORES`.

To map these relationships, we created a new relation to represent each relationship - we created `BOOK_CATS` and `STORE_INFO`. The primary keys of the participating entities in the relationship are included as foreign keys in the new relation, and make up the primary key. Any simple attributes of the relationship, including the attributes '`rating`' and '`price`' of the relationship `SELL` between `PRODUCTS` and `STORES`, are included in the new relation.

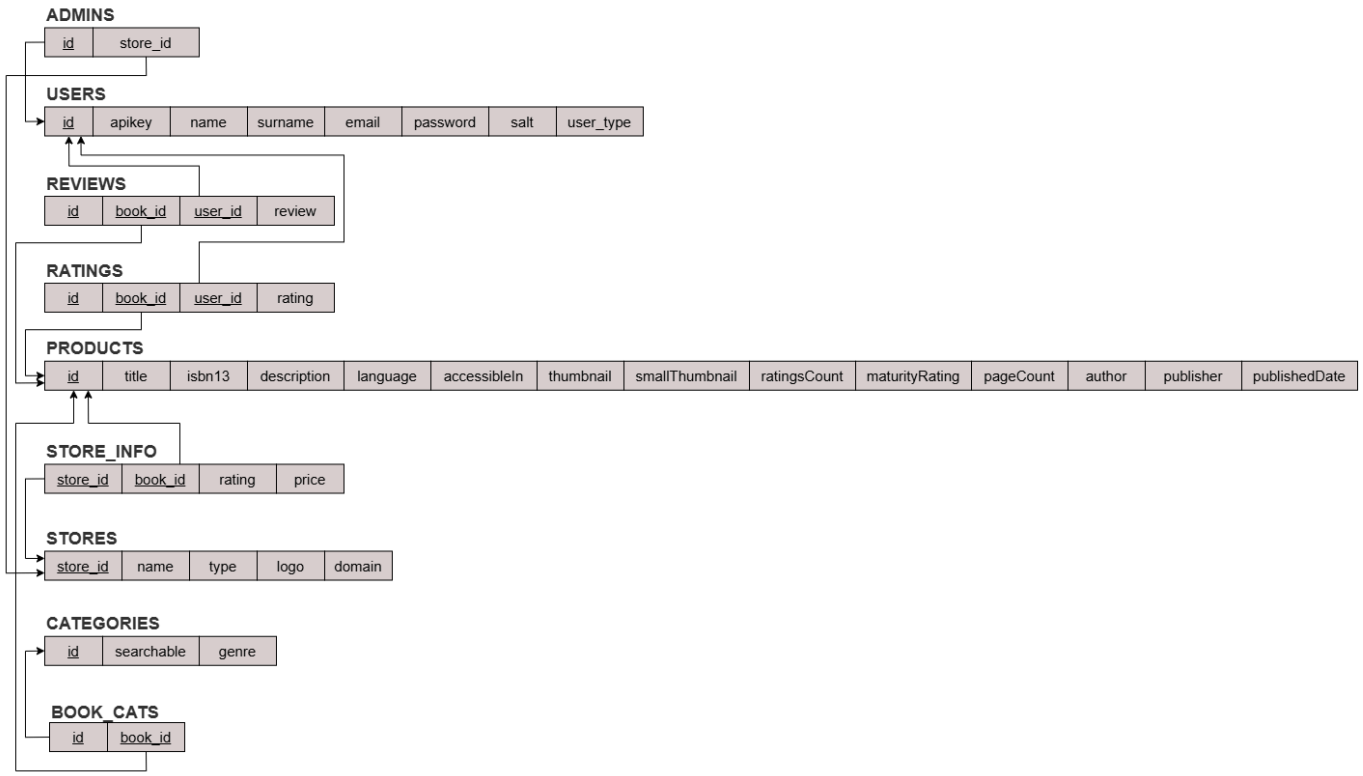


Step 6: Mapping of multivalued attributes

We have no multivalued attributes in our EER Diagram. Please see the assumptions in Task 2: (E)ER-Diagram for an explanation regarding why it was unnecessary to include multivalued attributes, such as a '`categories`' attribute on the entity `PRODUCTS`.

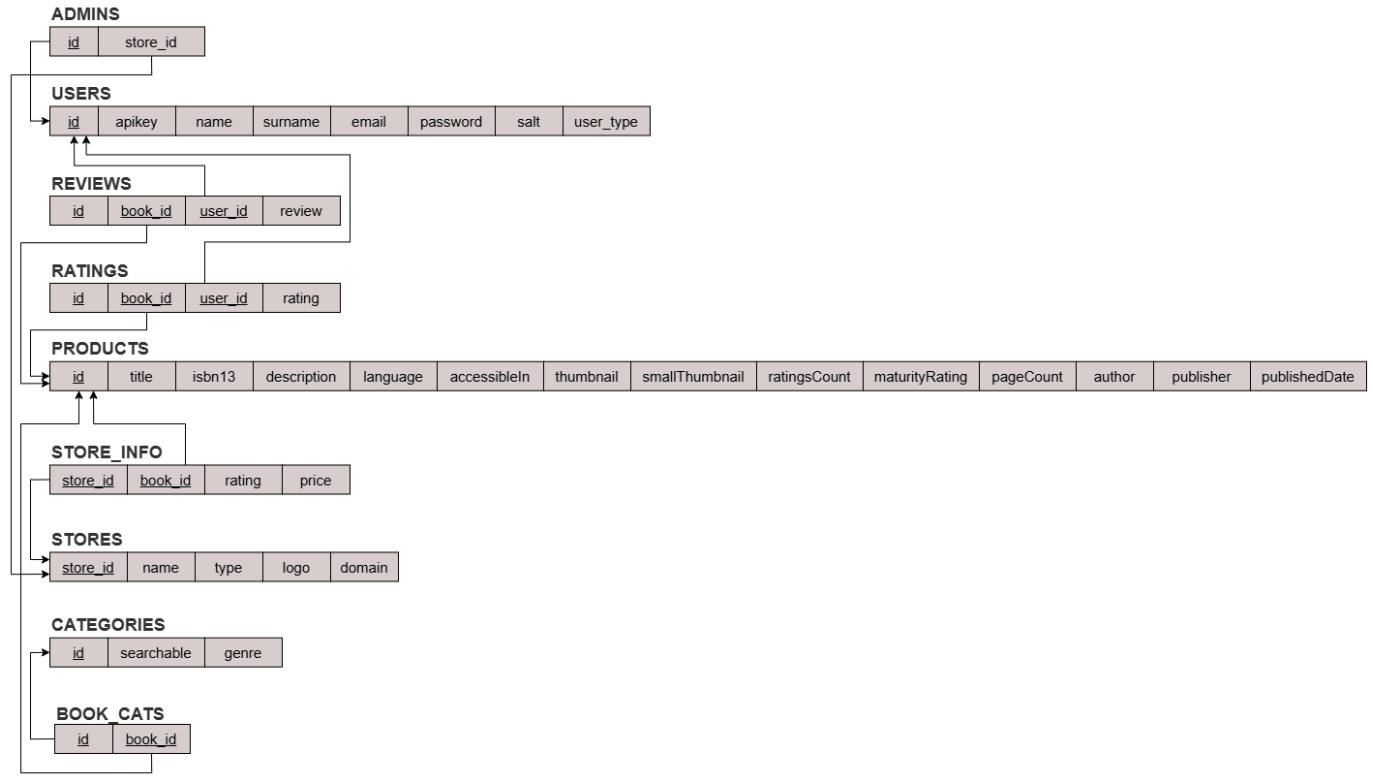
Step 7: Mapping of N-ary relationship
We have no N-ary relationships in our EER Diagram, as they were deemed unnecessary during the diagram creation and therefore were not mapped.

Step 8: Mapping specialisation and generalization
We have one specialisation in our EER Diagram: `ADMINS` is a subclass of the superclass `USERS`. Admins are users that manage a specific store's book prices and ratings. We chose to map it in the following way: created a new relation `ADMINS` for the subclass, and included the attributes of this entity in the relation. We then took the primary key '`id`' of the superclass and made it the primary key of the new relation. We also mapped the relation to the STORES entity by including its primary key as a foreign key in ADMINS.



Step 9: Mapping unions
We have no unions in our EER Diagram as they were deemed unnecessary during the diagram creation and therefore were not mapped.

Final Relational Mapping



Task 4: Relational Schema . . . . . .

Table creation SQL statements:

Includes data types, length constraints and checks.

```
--
-- Database: `u24566552_MQGA`
--
-- --------------------------------------------------------
--
-- Table structure for table `ADMINS`
--
CREATE TABLE `ADMINS` (
 `id` int(11) NOT NULL,
 `store_id` int(11) DEFAULT NULL
);
-- --------------------------------------------------------
--
-- Table structure for table `BOOK_CATS`
--
CREATE TABLE `BOOK_CATS` (
 `book_id` int(11) NOT NULL,
 `category_id` int(11) NOT NULL
);
-- --------------------------------------------------------
--
-- Table structure for table `CATEGORIES`
--
CREATE TABLE `CATEGORIES` (
 `category_id` int(11) NOT NULL,
```

```
 `searchable` tinyint(1) DEFAULT NULL,
 `genre` varchar(100) NOT NULL
);
-- ------------------------------------------------------
--
-- Table structure for table `PRODUCTS`
--
CREATE TABLE `PRODUCTS` (
 `id` int(11) NOT NULL,
 `tempID` varchar(50) NOT NULL,
 `title` text NOT NULL,
 `description` text DEFAULT NULL,
 `isbn13` varchar(13) DEFAULT NULL,
 `publishedDate` date DEFAULT NULL,
 `publisher` varchar(255) DEFAULT NULL,
 `author` varchar(255) DEFAULT NULL,
 `pageCount` int(11) DEFAULT NULL,
 `maturityRating` varchar(50) DEFAULT NULL,
 `language` varchar(50) DEFAULT NULL,
 `smallThumbnail` varchar(512) DEFAULT NULL,
 `thumbnail` varchar(512) DEFAULT NULL,
 `accessibleIn` varchar(100) DEFAULT NULL,
 `ratingsCount` int(11) DEFAULT 0
) ;
-- ------------------------------------------------------
--
-- Table structure for table `RATINGS`
--
CREATE TABLE `RATINGS` (
 `id` int(11) NOT NULL,
 `book_id` int(11) NOT NULL,
 `user_id` int(11) NOT NULL,
 `rating` smallint(6) NOT NULL CHECK (`rating` >= 1 and `rating` <= 5)
);
-- ------------------------------------------------------
--
-- Table structure for table `REVIEWS`
--
CREATE TABLE `REVIEWS` (
 `id` int(11) NOT NULL,
 `book_id` int(11) NOT NULL,
 `user_id` int(11) NOT NULL,
 `review` text DEFAULT NULL
);
-- ------------------------------------------------------
--
-- Table structure for table `STORES`
--
CREATE TABLE `STORES` (
 `store_id` int(11) NOT NULL,
 `name` varchar(150) NOT NULL,
 `logo` varchar(512) DEFAULT NULL,
 `domain` varchar(255) DEFAULT NULL,
 `type` varchar(100) DEFAULT NULL
);
-- ------------------------------------------------------
```

```
--
-- Table structure for table `STORE_INFO`
--
CREATE TABLE `STORE_INFO` (
 `book_id` int(11) NOT NULL,
 `store_id` int(11) NOT NULL,
 `price` decimal(10,2) DEFAULT NULL,
 `rating` decimal(3,2) DEFAULT NULL CHECK (`rating` is null or `rating` >= 1.00 and
`rating` <= 5.00)
);
-- --------------------------------------------------------
--
-- Table structure for table `USERS`
--
CREATE TABLE `USERS` (
 `id` int(11) NOT NULL,
 `apikey` varchar(64) DEFAULT NULL,
 `name` varchar(100) NOT NULL,
 `surname` varchar(100) NOT NULL,
 `email` varchar(255) NOT NULL,
 `password` varchar(255) NOT NULL,
 `salt` varchar(64) NOT NULL,
 `user_type` enum('super','regular','admin') NOT NULL
);
```

Table Constraints SQL statements:

Includes primary, secondary, foreign keys as well as indexes.

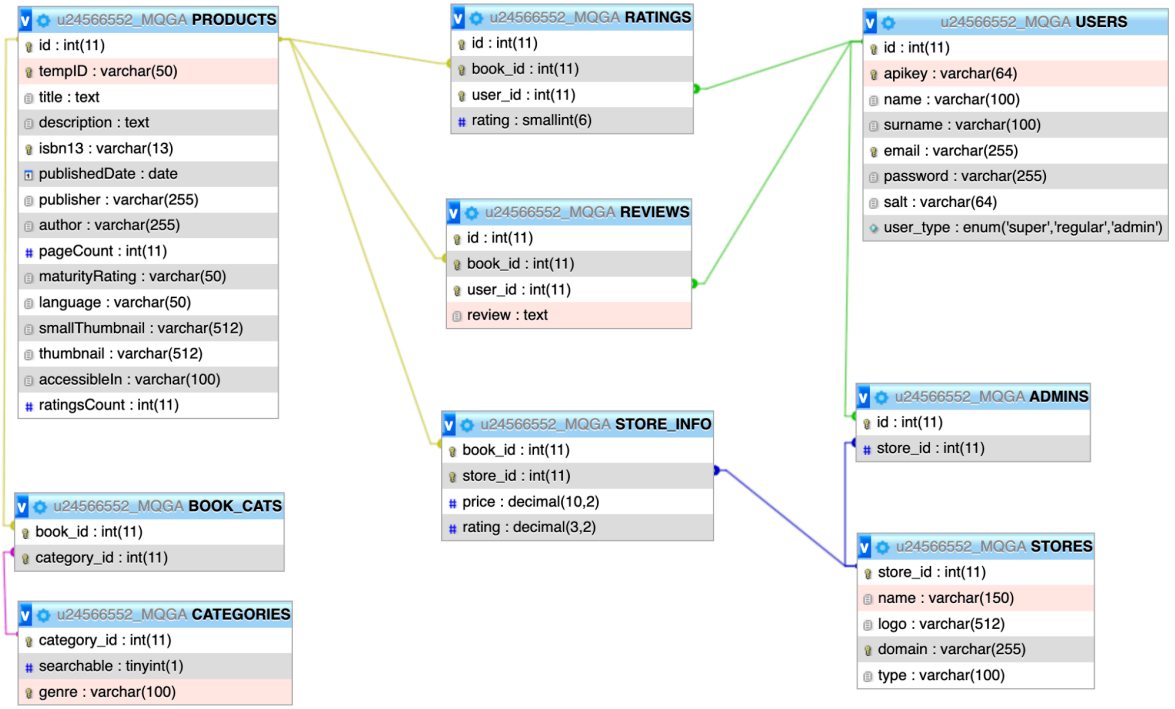
We decided to use database indexes right from the start to make sure everything runs fast, faster is always better. Putting in the main indexes early also means we are set up to handle more data later and can hopefully avoid slowdowns as things get bigger.

```
--
-- Indexes for table `ADMINS`
--
ALTER TABLE `ADMINS`
 ADD PRIMARY KEY (`id`),
 ADD KEY `idx_admins_store_id` (`store_id`);
--
-- Indexes for table `BOOK_CATS`
--
ALTER TABLE `BOOK_CATS`
 ADD PRIMARY KEY (`book_id`,`category_id`),
 ADD KEY `idx_bookcats_category_book` (`category_id`,`book_id`);
--
-- Indexes for table `CATEGORIES`
--
ALTER TABLE `CATEGORIES`
 ADD PRIMARY KEY (`category_id`),
 ADD UNIQUE KEY `genre` (`genre`),
 ADD KEY `idx_categories_searchable` (`searchable`);
--
-- Indexes for table `PRODUCTS`
--
ALTER TABLE `PRODUCTS`
 ADD PRIMARY KEY (`id`),
 ADD UNIQUE KEY `tempID` (`tempID`),
```

```
 ADD UNIQUE KEY `isbn13` (`isbn13`),
 ADD KEY `idx_products_author` (`author`),
 ADD KEY `idx_products_publisher` (`publisher`);
--
-- Indexes for table `RATINGS`
--
ALTER TABLE `RATINGS`
 ADD PRIMARY KEY (`id`,`book_id`,`user_id`),
 ADD UNIQUE KEY `uq_user_book_rating` (`user_id`,`book_id`),
 ADD KEY `idx_ratings_book_user` (`book_id`,`user_id`);
--
-- Indexes for table `REVIEWS`
--
ALTER TABLE `REVIEWS`
 ADD PRIMARY KEY (`id`,`book_id`,`user_id`),
 ADD UNIQUE KEY `uq_user_book_review` (`user_id`,`book_id`),
 ADD KEY `idx_reviews_book_user` (`book_id`,`user_id`);
--
-- Indexes for table `STORES`
--
ALTER TABLE `STORES`
 ADD PRIMARY KEY (`store_id`),
 ADD UNIQUE KEY `domain` (`domain`),
 ADD KEY `idx_stores_name` (`name`);
--
-- Indexes for table `STORE_INFO`
--
ALTER TABLE `STORE_INFO`
 ADD PRIMARY KEY (`store_id`,`book_id`),
 ADD KEY `idx_storeinfo_book_store` (`book_id`,`store_id`),
 ADD KEY `idx_storeinfo_price` (`price`),
 ADD KEY `idx_storeinfo_rating` (`rating`);
--
-- Indexes for table `USERS`
--
ALTER TABLE `USERS`
 ADD PRIMARY KEY (`id`),
 ADD UNIQUE KEY `email` (`email`),
 ADD UNIQUE KEY `apikey` (`apikey`);
--
-- AUTO_INCREMENT for table `CATEGORIES`
--
ALTER TABLE `CATEGORIES`
 MODIFY `category_id` int(11) NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT for table `PRODUCTS`
--
ALTER TABLE `PRODUCTS`
 MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT for table `RATINGS`
--
ALTER TABLE `RATINGS`
 MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT for table `REVIEWS`
```

```
--
ALTER TABLE `REVIEWS`
 MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT for table `STORES`
--
ALTER TABLE `STORES`
 MODIFY `store_id` int(11) NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT for table `USERS`
--
ALTER TABLE `USERS`
 MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
--
-- Constraints for table `ADMINS`
--
ALTER TABLE `ADMINS`
 ADD CONSTRAINT `ADMINS_ibfk_1` FOREIGN KEY (`id`) REFERENCES `USERS` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE,
 ADD CONSTRAINT `ADMINS_ibfk_2` FOREIGN KEY (`store_id`) REFERENCES `STORES`
(`store_id`) ON DELETE CASCADE ON UPDATE CASCADE;
--
-- Constraints for table `BOOK_CATS`
--
ALTER TABLE `BOOK_CATS`
 ADD CONSTRAINT `fk_bookcats_book` FOREIGN KEY (`book_id`) REFERENCES `PRODUCTS` (`id`)
ON DELETE CASCADE ON UPDATE CASCADE,
 ADD CONSTRAINT `fk_bookcats_category` FOREIGN KEY (`category_id`) REFERENCES
`CATEGORIES` (`category_id`) ON DELETE CASCADE ON UPDATE CASCADE;
--
-- Constraints for table `RATINGS`
--
ALTER TABLE `RATINGS`
 ADD CONSTRAINT `fk_rating_book` FOREIGN KEY (`book_id`) REFERENCES `PRODUCTS` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE,
 ADD CONSTRAINT `fk_rating_user` FOREIGN KEY (`user_id`) REFERENCES `USERS` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE;
--
-- Constraints for table `REVIEWS`
--
ALTER TABLE `REVIEWS`
 ADD CONSTRAINT `fk_review_book` FOREIGN KEY (`book_id`) REFERENCES `PRODUCTS` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE,
 ADD CONSTRAINT `fk_review_user` FOREIGN KEY (`user_id`) REFERENCES `USERS` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE;
--
-- Constraints for table `STORE_INFO`
--
ALTER TABLE `STORE_INFO`
 ADD CONSTRAINT `fk_storeinfo_book` FOREIGN KEY (`book_id`) REFERENCES `PRODUCTS` (`id`)
ON DELETE CASCADE ON UPDATE CASCADE,
 ADD CONSTRAINT `fk_storeinfo_store` FOREIGN KEY (`store_id`) REFERENCES `STORES`
(`store_id`) ON DELETE CASCADE ON UPDATE CASCADE;
COMMIT;
```

Schema in visual form:



Task 5: Web-based Application . . . . . .

Pages

Launch Page:

The launch page is the first page the users will see before and after logging in. This page contains useful statistics to all users. A button below the statistics leads users to statistics specific to stores. The header at the top of the page will be used for navigation.

Login:

The login page allows users to enter their email and password and log in to the website and have access to certain webpages depending on their user type.

Register:

The register page allows new users to register an account to gain access to the website and all its features. Only users will make use of this page as store admins need to be registered via the super manager management page.

- This extends the database

Header:

The header will act as the navigation bar for all user types. Regular users will be able to see the Home, Dashboard, Products and Profile buttons. Admins and the super admin will be able to see an additional button: Control Panel.

Specific Store Statistics

All user types can access this page. It displays the average price and rating of all books in a store for all stores in the database.

Dashboard:
This page contains featured books and the highest rated books. Currently a few action books are being featured. Clicking on the card will lead users to the view page for that specific product.

Products:
Similar in design to the dashboard, but contains all products. These products can be sorted and filtered by name, author, rating and/or genre. Searching for a book name using the title is also possible. Clicking on the card will lead users to the view page for that specific product.

View:
The view page allows both admins and users to look at a specific book, all its details and the prices at various stores. It displays them in order from cheapest to most expensive. They are also able to view all the ratings and reviews of a book however only the user is able to give a review or rating of a book. They are limited to one review and rating however they are able to replace them.

- This page allows users to rate and/or review products
- This page allows all the different types of users to view a product, and display an image of the product, the list of prices at different retailers (stockists), ratings and reviews

Control Panel:
The control panel is only seen by admins and the super admin. Admins have the option to navigate to their store page, adding a book to their store and editing specific information about a book.

Edit Books:
Accessible by admins and the super admin in the control panel. This page's functionality changes depending on what type of admin is logged in.
The **super admin** has the following privileges:

- Deleting books from all stores.
- Deleting books from specific stores.
- Editing the information of all books from all stores, or simply editing a book's information from a specific store

Admins have the following privileges:

- Deleting books from their store.
- Editing the price and the rating of books in their store.

Management page:
This page allows the super manager to add and delete both stores and admins for specific stores and see the store information of all stores. Mangers are able to view their specific store and edit its information.

- This updates, extends and deletes from the database
- This page meets the criteria to manage stockists

Profile:
This page allows admins, the super admin and users to change their personal information and allows users to review their reviews and ratings and change them.

- This allows users to rate and/or review products
- This would update the database tables

Default Logins

Super admin:

- Email: mqga@gmail.com
- Password: Admin123!

Store admin:

- Email: mike@gmail.com
- Password: mikePass!1

Normal user:

- Email: cailin@gmail.com
- Password: CaiPas123@

Functionality

Customers are able to:

- Sign up and log in
- View the dashboard
- View products and filter and sort them by various filters including categories and more
- Rate and review books and see others' ratings and reviews on various books
- View and edit their profile, past ratings and reviews

Store admins are able to:

- Log in
- View the dashboard and products page
- View and change their profile information
- View and change their store information
- Add new books or add their own store's price for existing ones
- Edit their store's price and rating for a book

Super admin is able to:

- Log in
- View the dashboard and products page
- View and change their profile information
- Add, delete and manage stores and their admins
- View and remove users
- Add and delete books
- Edit book information

Security:

- We protected against common web vulnerabilities by using prepared statements throughout our backend(MySqli statements) to prevent SQL injection and sanitising user inputs(sanitise_util.php) with htmlspecialchars to mitigate XSS(Cross-Site-Scripting) attacks.
- Sensitive user info/credentials are secured through a 2-step-hash-salting method involving unique salts, HMAC-SHA256, and bcrypt via password_hash, while sensitive passwords like database credentials are kept secure using .env files.This was also not uploaded to our github using .gitignore files.

- System access is controlled via randomly generated API keys and role-based authorisation checks(auth_utils.php) to ensure users can only perform permitted actions.

Task 6: Data . . . . . .

Population Methods:

Reasoning:

The data population itself underwent several different iterations before coming to a finalised decision. The initial attempts involved finding existing files with all the data that we would need; however, as a result of the niche of our products, coming across this data with enough relevant fields was difficult. The switch was made to searching for an API that would be able to provide enough data that we could form the results ourselves. Contenders included:

- New York Times API - https://developer.nytimes.com/docs/books-product/1/overview
- Open Library API - https://openlibrary.org/developers/api
- Kaggle dataset: - https://www.kaggle.com/datasets/elvinrustam/books-dataset

All of which had some limiting factor, be it, number of available products, the relevancy of data or paywall access, we had finally decided upon Google Books API, which, while having a maximum response size of just over 40 books, some workarounds were found.

Generation of Product Data:

The google books API was more than sufficient in terms of number of possible fields that could be extracted therefore after careful deliberation regarding what fields we would want to populate our Products table with a python script was written to parse the JSON response of the google books API and return a formatted JSON file with all of our relevant books.

We took this further by incorporating the USE of a map to keep track of all books, genres that we had already seen to force unique categories and an even distribution of books across all genres this was done externally from the requests themselves so that we could make requests to the API 100s of times with varying parameters to get around the 40 book limit. This meant that a file populated with over 500 JSON objects representative of the books themselves, with varying genres, categories, and languages, was all possible in under a minute

Now that a JSON file existed, we needed a way to insert this data into our respective tables.

A separate python script was written to parse the new json file that was created and convert it into a logical set of INSERT statements to populate the Products tables.This file was then provided to AI, namely Claude Sonnet 3.7, to review and adjust any data that would not fit our schema. Finally, the data was manually reviewed after being inserted.

SQL Statements were used to extract all the book ids that had a null value in any of their columns, so that they could be deleted. While our schema allows certain values to be null, we decided that for demo purposes, our database would only contain data that was *complete* and thus would ensure the best presentation of data possible. Thus, after deleting all products with nulls, we had approximately 330 books in our PRODUCTS table.

Generation of Categories Data
Initially, a map was used to keep track of categories, and a php script used to parse and convert the new file into a set of INSERT statements to populate the `CATEGORIES` and `BOOK_CATS` tables. However, this script did not match our final schema, and so it had been changed and that data deleted.

Instead, the following was done. A script was used to extract all of the distinct genres from the initial products JSON, and INSERT statements created to enter these categories into the `CATEGORIES` table. Additional categories were also manually added, to ensure that we had a wide variety in the case of books being added to the `PRODUCTS` table.

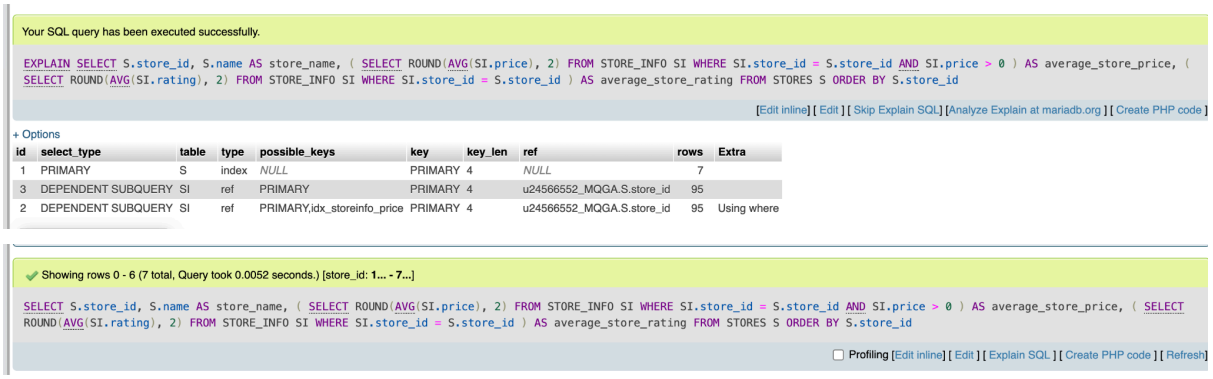
To connect the products to their corresponding categories, another php script was used to extract each products' categories array, and make INSERT statements for the `BOOK_CATS` table. This ensured that relevant categories were added to each book.

Generation of Reviews and Ratings
A php script was used to generate 100 ratings for each of 14 users, on 100 random books. The script generated one INSERT statement that was then used to populate the `REVIEWS` table with 1400 tuples. The reason for choosing to make use of a script, was as it ensured we could have sufficient data, through as little manual entry as possible. Using a script meant we could have hundreds of entries almost immediately, and the most work required was to create the actual script.

Reviews were initially manually inserted by means of navigating the website, and adding reviews and rating through the View page of random books. However, this would have taken too long to create sufficient data. So we decided to also create a php script to insert reviews. This was one of the more difficult scripts to create, as we wanted to ensure that the majority of reviews also had a rating that would *correspond* to the review's level of criticism. The script was created to ensure that a review that would typically come with a 1 star rating, would also insert a rating of 1 into the table. New users were made to do this, as the `RATINGS` table already had ratings for all of the current users, and we wanted to avoid trying to insert ratings for users on books they had already rated. This resulted in the creation of 2700 extra tuples in the `RATINGS` table and 2700 extra tuples in the `REVIEWS` table

Generation of Prices
The `STORE_INFO` table, which contains prices and ratings for specific stores for specific books, needed to be filled with sufficient data to ensure that the price-comparison was useful. A php script was again used to do so. The script would take the 7 store ids, and 330 books, and then do the following for each book: pick 4 stores, and generate a price between R150 and R750 for one of the stores, then generate 3 prices for the other stores. The prices for the other stores are generated by picking a ratio from a price multiplier array, so that the prices for a book are similar, to prevent having unrealistic price gaps. A similar process is done for the ratings for each store. Thus, an INSERT statement was created, with 4 rows for each book.
Once this was done, the query was used to insert the data into the `STORE_INFO` table. This gave us 1320 tuples for this table.

Task 7: Analyse and Optimise . . . . . .



Your SQL query has been executed successfully.

```
EXPLAIN SELECT S.store_id, S.name AS store_name, ( SELECT ROUND(AVG(SI.price), 2) FROM STORE_INFO SI WHERE SI.store_id = S.store_id AND SI.price > 0 ) AS average_store_price, (
SELECT ROUND(AVG(SI.rating), 2) FROM STORE_INFO SI WHERE SI.store_id = S.store_id ) AS average_store_rating FROM STORES S ORDER BY S.store_id
```

[Edit inline] [Edit] [Skip Explain SQL] [Analyze Explain at mariadb.org] [Create PHP code]

+ Options

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	S	index	*NULL*	PRIMARY	4	*NULL*	7	
3	DEPENDENT SUBQUERY	SI	ref	PRIMARY	PRIMARY	4	u24566552_MQGA.S.store_id	95	
2	DEPENDENT SUBQUERY	SI	ref	PRIMARY,idx_storeinfo_price	PRIMARY	4	u24566552_MQGA.S.store_id	95	Using where

Showing rows 0 - 6 (7 total, Query took 0.0052 seconds.) [store_id: **1... - 7...**]

```
SELECT S.store_id, S.name AS store_name, ( SELECT ROUND(AVG(SI.price), 2) FROM STORE_INFO SI WHERE SI.store_id = S.store_id AND SI.price > 0 ) AS average_store_price, ( SELECT
ROUND(AVG(SI.rating), 2) FROM STORE_INFO SI WHERE SI.store_id = S.store_id ) AS average_store_rating FROM STORES S ORDER BY S.store_id
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Normal query analysis:Initial query analysis reveals a 0.0052-second execution time for retrieving the average price and rating of a book per store. Examining the EXPLAIN output indicates that we used two subqueries, leading to redundant overhead. If we take a closer look at the current structure, it generates 2*n subqueries, where 'n' represents the number of stores, as the subqueries are executed independently before the main query for each store.
Optimisation strategy:Instead of using sub queries for the average and rating for each store,we could use a join function that already joins both together,eliminating the need for repeated sub queries for each store.

Your SQL query has been executed successfully.

```
EXPLAIN SELECT S.store_id, S.name AS store_name, ROUND(AVG(SI.price), 2) AS average_store_price, ROUND(AVG(SI.rating), 2) AS average_store_rating FROM STORES S JOIN STORE_INFO
SI ON S.store_id = SI.store_id GROUP BY S.store_id, S.name ORDER BY S.store_id
```

[Edit inline] [Edit] [Skip Explain SQL] [Analyze Explain at mariadb.org] [Create PHP code]

+ Options

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	S	index	PRIMARY	idx_stores_name	602	*NULL*	7	Using index; Using temporary; Using filesort
1	SIMPLE	SI	ref	PRIMARY	PRIMARY	4	u24566552_MQGA.S.store_id	95	

Showing rows 0 - 6 (7 total, Query took 0.0026 seconds.) [store_id: **1... - 7...**]

```
SELECT S.store_id, S.name AS store_name, ROUND(AVG(SI.price), 2) AS average_store_price, ROUND(AVG(SI.rating), 2) AS average_store_rating FROM STORES S JOIN STORE_INFO SI ON
S.store_id = SI.store_id GROUP BY S.store_id, S.name ORDER BY S.store_id
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

*Optimised query analysis:*We used a join to directly connect STORES with STORE_INFO.We also used a GROUP BY sql clause to partition the resulting dataset by storeusing store_id. Next we used aggregate functions (like AVG()) to compute the average price and rating.This allows the RDBMS to use a more stream-lined,set-based way of computing.
Improvement: We saw a 50% decrease in total execution time(from 0.0052 seconds to 0.0026 seconds) which optimised the query successfully.
Potential reasons for improvement:

- We eliminated the need for iterative subquery execution
- We also leveraged the RDBMS efficiency for set-based processing

Task 8: Development . . . . . .

Github Link: https://github.com/KyleMcCalgan/MQGA-COS221-Practical5

Github Strategy:
Our GitHub strategy uses a stable *main* branch, with *FRONTEND* and *BACKEND* branches for separate development and integration of those components. Cailin, Kyle and Marcel

primarily utilised the *BACKEND* branch while Johan, Jordan and Kyle used the *FRONTEND* branch. The separation of branches ensured development and integration in unison, while working independently from each other. Each of us created temporary branches from these *sub-main* branches which were used when implementing new components/functions/additions. Once the temporary branches were tested and confirmed to be working, they were merged with the parent branch (*BACKEND/FRONTEND*). Finally, the *FRONTEND* and *BACKEND* are merged into the primary *main* branch for a fully integrated application, ensuring only tested, usable code gets merged with *main*.

Task 9: Demo

Cailin Smith (u24570525):
Main Tasks:
1. Backend: API development
2. EER Diagram
3. Relational Mapping

Sub Tasks:
1. Data generation, including ensuring all books have valid data, inserting and assigning categories, and adding prices, ratings and reviews for all books and stores.
2. Project document and file creation and compilation

Jordan Naidoo (u24664155):
Main Tasks:
1. Frontend design
2. Frontend integration

Sub Tasks:
1. E-ER Diagram Design
2. Assisted in designing the Relational Diagram

Kyle McCalgan (u24648826):
Main Tasks:
1. Data Generation, including manipulating a dataset to retrieve and insert product data
2. Backend API development

Sub Tasks:
1. Auto upload to wheatley upon commits to main
2. Github creation and management
3. Frontend integration

Marcel Stoltz (u24566552):
Main Tasks:
1. Backend: API development
2. Relational Schema
3. Data creation/Optimisation of query
4. Designed Demo Presentation/flow diagrams

Sub Tasks:
1. Assisted in compiling the project document.
2. Helped the team with understanding and managing the github strategy.
3. Created the API structure and strategy,assisted with E-ER diagram and relational mapping.

Johan Coetzer (u24564584):
Main Tasks:

1. Frontend Design
2. Frontend Integration

Sub Tasks:

1. E-ER Diagram Design
2. Assisted in designing the Relational Diagram

Contributions

Task 1: Research
- ➢ Completed by every member of the group

Task 2: EER Diagram
- ➢ Initial database design was completed by Jordan, Johan, and Cailin
- ➢ EER diagram was completed by Marcel and Cailin

Task 3: Relational Mapping
- ➢ Completed by Marcel and Cailin

Task 4: Relational Schema
- ➢ Completed by Marcel

Task 5: Web-based application
- ➢ Design of website completed by Johan and Jordan
- ➢ API creation completed by Kyle, Marcel, and Cailin
- ➢ Integration completed by Johan, Jordan, and assisted by Kyle

Task 6: Data
- ➢ Data creation and compilation completed by Kyle, Marcel and Cailin, and assisted by everyone to ensure sufficient data

Task 7: Analyse and Optimisation
- ➢ Completed by Marcel

Task 8: Development
- ➢ Github setup and management completed by Kyle and Marcel

Task 10: Bonus Task
- ➢ Deployment setup completed by Kyle

Task 10: Bonus Task - Push the Boundaries . . . . . .

For our bonus task, we implemented an automated Continuous Deployment pipeline using GitHub Actions to streamline our development workflow and ensure reliable deployments.

What We Implemented

We created an automated deployment system that triggers whenever code is pushed to the main branch of our repository. The pipeline automatically deploys our backend PHP application to a remote server using FTP.

Technical Implementation

Our GitHub Actions workflow includes a trigger that activates on every push to the main branch, runs on Ubuntu latest, and follows two main steps: checking out the source code

using actions/checkout@v4 to fetch the latest code, and automated deployment using SamKirkland/FTP-Deploy-Action@v4.3.5 to sync files to our production server.

Key Features

The system uses GitHub Secrets to securely store FTP credentials (FTP_USERNAME and FTP_PASSWORD), only deploys the BACKEND/ directory to the appropriate server location, has debug mode enabled for transparency and troubleshooting, and provides automatic synchronization to keep the server updated with minimal interruption.

Benefits Achieved

This implementation provides an automated workflow that eliminates manual file transfers, ensures consistency since every deployment follows the same reliable process, offers instant deployment upon code changes, reduces errors by eliminating human error in manual deployments, and improves team collaboration as all team members benefit from the same deployment process.