CS307 Database project part2

Name: Wen Yinan
StudentID: 12310841
Class: Lab Mon 9-10

Name: Cui Zixuan
StudentID: 12311007
Class: Lab Mon 9-10

March 10, 2025

Contents

1 Information and Contribution

1.1 Contribution

The basic information and contribution of our members are as follow.
- Cui Zixuan:

- the basic backend implementation
- the implementation of the bonus method
- code optimization
- the completion of a small portion of the report

- Wen Yinan:

- Connect the frontened and the complete backend system
- Construct the frontened web pages
- Design user's privilege. Realised login, logout, and admin management functions.
- ProjectReports

1.2 Language

We applied the following languages in our design
Java 19.0.1 2022-10-18
Java(TM) SE Runtime Environment (build 19.0.1+10-21)
Java HotSpot(TM) 64-Bit Server VM (build 19.0.1+10-21, mixed mode, sharing)

1.3 Test Environment

We deployed two machines in our test
• HUAWEI SOUND(Processor: 13th Gen Intel(R)Core(TM)i5-13500H 2.60 GHz)
• Lenovo Legion R9000x-2021-R(Processor: AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz)

2 Database Design

2.1 Designing the E-R Diagram

2.1.1 ER Diagram Snapshot

Below is a snapshot of the ER diagram we designed for our website:

This ER diagram provides a comprehensive view of the structure of the database and how entities and attributes are interconnected.

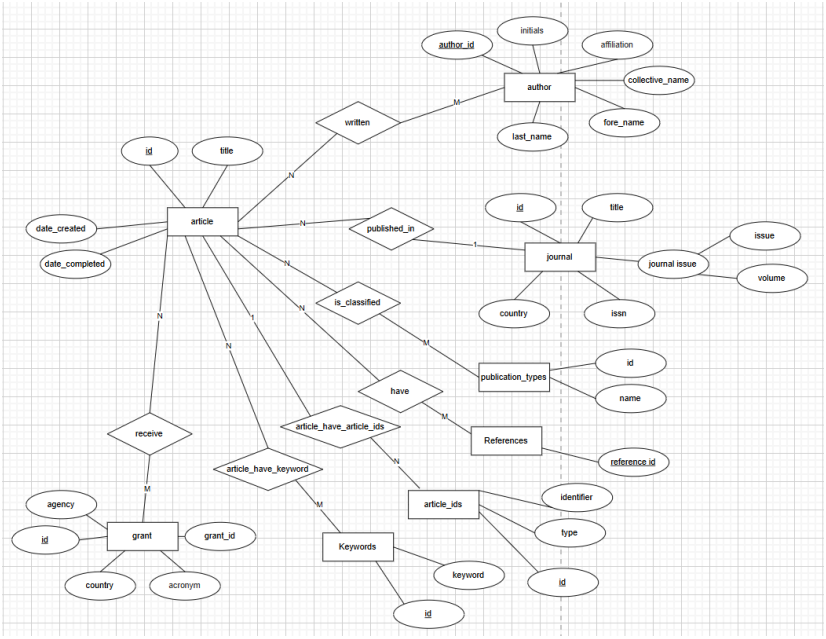

Figure 1: ER Diagram

2.2 Brief description

2.2.1 Tables and Columns

Here is the database diagram for a clearer understanding of our database design.
We adjust the given "Goodloader" and add a user table to construct our database.

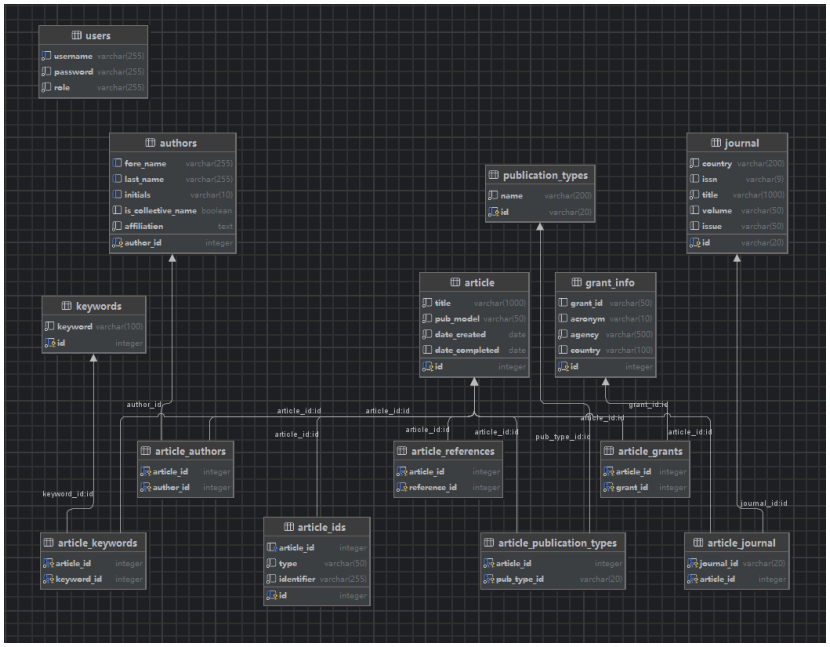

Figure 2: database Diagram

- **Article Main Table**
  1. **id(int, PRIMARY KEY, unique)**
  2. **title(varchar(2550),not null):** We used the varchar(2550) data type for "title" to ensure it can accommodate sufficiently long article titles
  3. **pub model(varchar(30))not null:** We utilized the varchar data type for "pub model" as there are only five possible values for it : "Print" , "Print-Electronic" , "Electronic" , "Electronic Print" , "Electronic-eCollection".
  4. **date created(date, not null):** We use the date data type for the DATE type ensures that date data is stored in a uniform format (YYYY-MM-DD), avoiding errors caused by inconsistent formats. Also by using the DATE type, the stored dates can be limited to a reasonable range, preventing the input of invalid dates such as "2024-13-01". Moreover, Using the DATE type can reduce unnecessary storage space usage, especially when dealing with large amounts of date data.
  5. **date completed(date):** It is very similiar to the colume "date created".
- **Authors Main Table**
  1. **author id(serial PRIMARY KEY):** We choose serial as its datatype to create a unique primary key. It ensures each relationship has a unique identifier, and the identifier increments automatically, making it easy to manage.
  2. **lastname(varchar(255))**
  3. **forename(varchar(255))**
  4. **initials(varchar(10))**
  5. **affiliation(text)**
  6. **is collective name(boolean)**
- **Journal Main Table**
  1. **journal id(varchar(40) PRIMARY KEY**
  2. **title(varchar(2550),not null)**

3. **country(varchar(100))**
4. **issn(varchar(20))**
5. **journal-issue-issue(varchar(200))**
6. **journal-issue-volume(varchar(200))**

- **Grant info Main Table**
  1. **grant id(varchar(50) PRIMARY KEY):**
  2. **agency(varchar(255) not null):** We used the varchar(255) data type for "agency" to ensure it can accommodate sufficiently long agency name.
  3. **country(varchar(70))**
  4. **acronym(varchar(50))**
- **Article Ids Main Table**
  1. **article id(int)**
  2. **id(int PRIMARY KEY)**
  3. **type(varchar(10) PRIMARY KEY)**
  4. **identifier (VARCHAR(255) NOT NULL)**
- **Publication Types Main Table**
  1. **publication type id(varchar(50) PRIMARY KEY):**
  2. **name(varchar(50))**
- **keywords Main Table**
  1. **id(int PRIMARY KEY):**
  2. **keyword(varchar(100))**
- **User Main Table**
  1. **username(varchar(255) PRIMARY KEY):**
  2. **reference id(varchar(255)PRIMARY KEY)**
  3. **role (varchar(255)PRIMARY KEY)**It must be user or admin
- **article-authors Table,article-publication-type Table,article-grants Table,article-references Table,article-journal Table,article-keywords Table:** This six tables share similar structures, aiming to connect the article with author, article with publication type, and article with grant by their id.
  1. **article id(int PRIMARY KEY)**
  2. **author/publication type/grant id(int PRIMARY KEY)**

2.2.2 Privilege

We divided users into "user" and "admin". Users can only reach functions of seaching such as getArticleCitationsByYear, getArticlesByAuthorSortedByCitations, getJournalWithMostArticlesByAuthor, getMinArticlesToLinkAuthors, getCountryFundPapers, getImpactFactor, getArticleCountByKeywordInPastYears

Whenever a account is added, we will add the new information into the table "User". Also, we enables admins to delete users in the table to better manage the database.

3 Basic API Specification

3.1 Raw Data

The data file is pubmed24n.ndjson

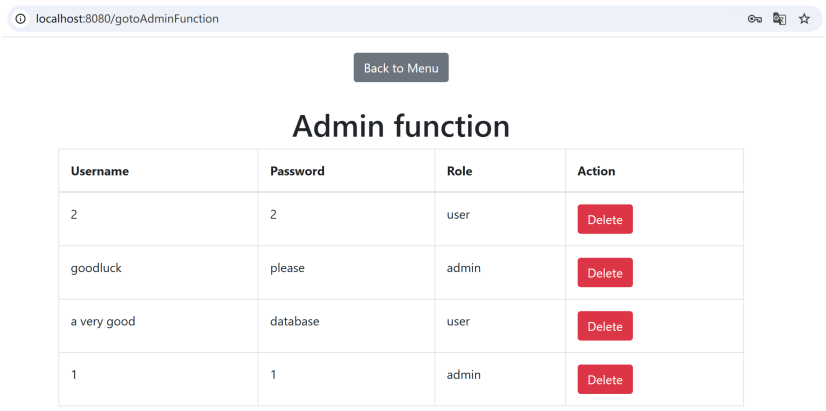

Figure 3: admin priviledge

3.2 Implementation Techniques

In this section, we outline the key techniques used in implementing the back-end functionalities, with a focus on dynamic SQL execution, error handling, and performance optimization.

I Temp Table for Citation Count

To improve performance in handling APIs related to article citations, we created a temporary table to store the citation count for each article. This optimization eliminates the need for repeated computations during API requests. Key steps include:

- Populating the temp table during the initial data import phase.
- Maintaining data correctness during article updates and deletions.
- Leveraging indexing on the temp table for faster query execution.

II Dynamic SQL Statements

We utilized dynamic SQL statements, dynamically adjusting the update SQL queries based on data received from the front-end. This approach ensures flexibility and efficiency, allowing different SQL queries to be constructed based on varying data conditions.

III Error Handling and Logging

Robust error handling mechanisms were employed to ensure API stability:

- Catching and gracefully handling all exceptions without propagating them to the API layer.
- Logging errors with sufficient details for debugging using the SLF4J logging framework.
- We created a dedicated `LogManager` class to manage error handling and logging, making it easier to view and manage log information.

IV Bonus Methods

We adopted a BFS approach combined with a materialized view. The materialized view serves as an intermediate table, enabling us to efficiently query information about articles, including their references and the articles that reference them. This setup significantly accelerates the query process to some extent. Additionally, the core algorithm involves a

standard BFS procedure. Moreover, we implemented a class named MaterializedViewRefresher, which updates the materialized view in response to changes in the underlying tables, ensuring the view remains up-to-date through a refresh mechanism.

This systematic approach ensures a robust, maintainable, and high-performance backend system.

4 Advanced APIs and Other Requirements

In this part, we developed a web application using Spring Boot for the backend and Thymeleaf for the frontend to handle user interactions and manage functionalities such as login, logout, permission control,and the above 9 APIs.

4.1 Backend (Spring Boot)

4.1.1 Controller Layer

- In Spring Boot, We created Controller classes annotated with @Controller to handle HTTP requests. Each controller method corresponds to an endpoint that performs specific actions like user registration, login, and logout. For example, the UserController handles user-related operations.
- The methods in the controller use annotations like @GetMapping and @PostMapping to map specific HTTP methods (GET, POST) to the corresponding actions.
- For example, the /login endpoint accepts the username and password from the frontend, checks the credentials via the service layer, and then decides the response (either login failure or redirect to a homepage).

4.1.2 Service Layer

1. Open your Datagrip and click the File option and you will find a new option, make your mouse hanging on the New option and you will will find a DataSource option in the new line showned.
2. Click the DataSource option and choose the PostgresSQL option and click.
3. In the new window named DataSource and divers, edit the username and your password to finish the last step.
4. Click the Test Connection button to ensure you have down load the necessary diver.
5. Open our IDEA java codes and enter the Main class, change the file path into yours. And enter the Datamanager class just change the URL, Username, and the password.
6. Till now, you can cilick the Main button and run the codes to import Data!!!

4.1.3 Model and Data Binding

- The data from the controller is passed to the frontend via the Model object, which stores the attributes that will be rendered by Thymeleaf. For instance, in the login process, user data or error messages are added to the model (model.addAttribute("error", "Invalid username or password")).
- Spring Boot uses ModelAndView to send the data and specify the view name (i.e., the Thymeleaf template) to be rendered.

4.2 Frontend (Thymeleaf)

4.2.1 Thymeleaf Templates

- Thymeleaf templates (HTML files) are stored in the src/main/resources/templates/ directory. These templates contain the HTML structure of the page along with Thymeleaf-specific syntax for data binding.
- We use Thymeleaf expressions (e.g., *error*,username) to insert dynamic data from the controller into the HTML.

4.2.2 Dynamic Content Rendering

Thymeleaf tags are used for rendering dynamic content. For example, th:text is used to insert text, and th:if is used to conditionally display elements. This is particularly useful when displaying messages like login errors or the user's name.

4.2.3 Form Handling

We used the ¡form¿ tag in Thymeleaf to handle user inputs and bind them to the backend. For instance, the login form in login.html binds the username and password fields to the corresponding parameters in the controller method using th:field.

5 Conclusion

In this project, we implemented a robust and scalable backend system using Spring Boot, PostgreSQL, and Thymeleaf to meet the requirements for handling article data, user interactions, and permissions. By incorporating optimization techniques such as the use of temporary tables for citation counts and dynamic SQL statements, we enhanced the performance and flexibility of the system, ensuring that it could handle complex queries and data updates efficiently.

The implementation of dynamic SQL allowed us to handle varying data conditions dynamically, making the system adaptable to different use cases. Meanwhile, the use of materialized views and recursive SQL queries provided significant performance improvements by storing precomputed results and refreshing them periodically through scheduled tasks.

Error handling and logging were carefully integrated into the system, ensuring that all exceptions were caught and logged for easy debugging. The integration of SLF4J logging and a dedicated LogManager class further streamlined the process of managing and analyzing logs.

On the frontend, Thymeleaf templates were employed to create dynamic, user-friendly interfaces that bind seamlessly with the backend. Through careful use of dynamic content rendering and form handling, we ensured that the user experience was smooth and interactive, with clear messages and form data binding for actions like login and registration.

Overall, this approach resulted in a robust, maintainable, and high-performance backend system that meets the needs of both data handling and user interaction, while also being flexible enough to accommodate future growth and improvements.