SUSTECH CS307 Fall 2024 Project Part II

12311124 Zhang Zihan(Monday 9-10) 12311624 Lin Pinhao(Wednesday 1-2)

December 28, 2024

Abstract

In this project, in the basic tasks, we create a temporary table to calculate the citations of every article when the program starts. We create a new class CitationManager to manage this temporary table. We also finished the bonus tasks by using bfs.

By exposing APIs and incorporating Role-Based Access Control (RBAC), we developed a GUI frontend using HTML and JavaScript to interact with the backend APIs. This implementation includes user login, registration, and query functionalities, as well as role-based access separation, allowing different users to call different APIs according to their permissions.We adopted Role-Based Access Control (RBAC) to achieve this, restricting users' access to specific resources and functionalities based on their roles.

1 Contributions

Percentages of Contributions: Zhang Zihan 0.5 and Lin Pinhao 0.5

1.1 Task Allocation

Task Name	Responsible Member(s)	Contribution (%)
Database Design (E-R Diagram)	Lin	100
Permissions Management Design	Zhang	100
API Implementation (Basic)	Zhang Lin	40 60
Performance Optimization	Zhang	100
Advanced APIs GUI	Lin	100
Report Writing	Zhang	100

2 Database Design

2.1 E-R Diagram

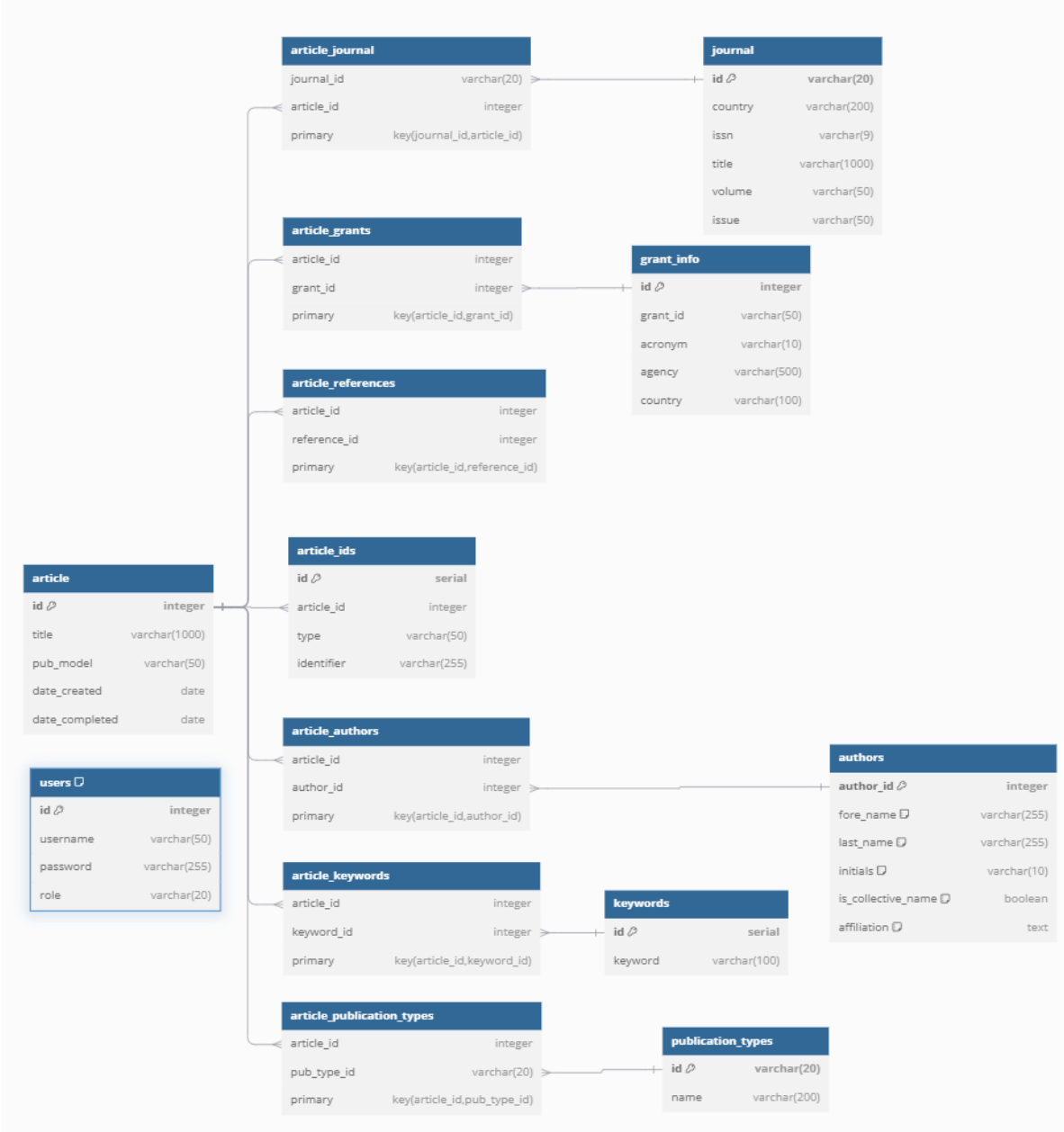

Figure 1: Enter Caption

2.2 Database Visualization

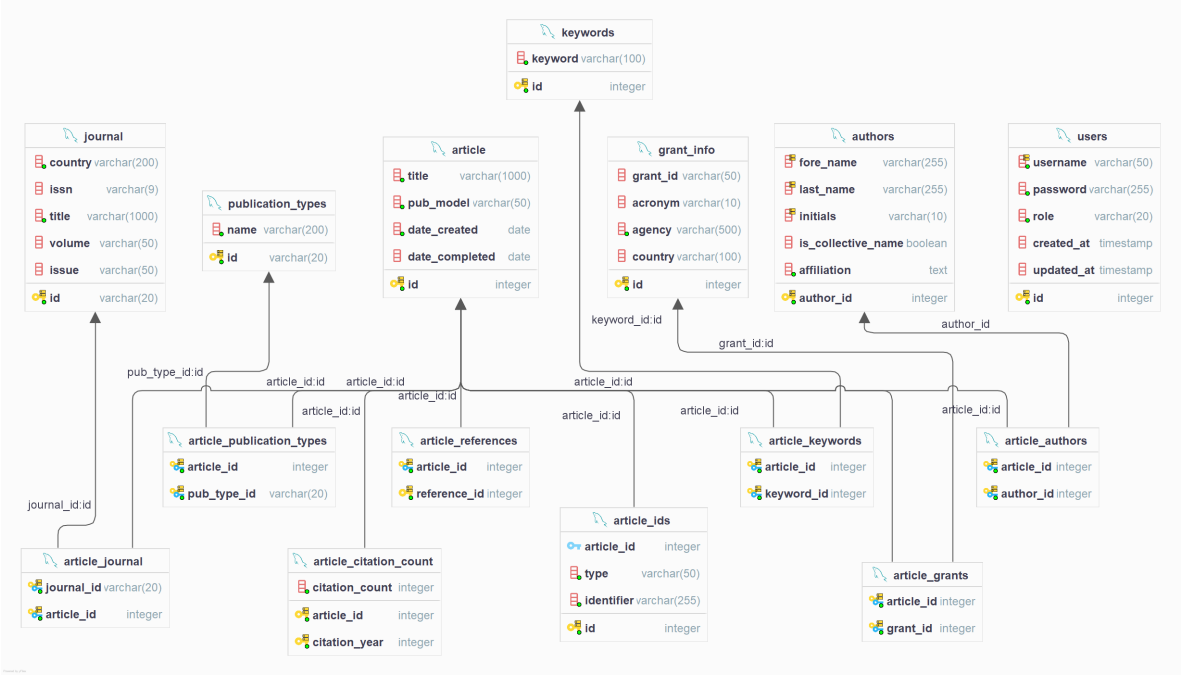

Figure 2: Database Visualization

3 Database Design

3.1 Database Table Design and Column Descriptions

In this project, the database is designed to store information related to scholarly articles, journals, authors, grants, and keywords. Each table is designed with specific fields to ensure efficient storage and retrieval of data while maintaining data integrity.

1. Article Table

The `Article` table stores basic information about each article. The key columns are:

- **id**: A unique identifier for the article, stored as an integer (1 to 99,999,999).
- **title**: The title of the article, stored as a string (maximum 1000 characters).
- **pub_model**: The publication model of the article, such as 'Print', 'Electronic', etc.
- **date_created**: The date on which the record of the article was created.
- **date_completed**: The completion date of the article (optional).

2. Journal Table

The `Journal` table contains information about journals where articles are published. The key columns include:

- **id**: Unique identifier for the journal (string, maximum 20 characters).
- **country**: The country where the journal is registered.
- **issn**: The ISSN of the journal (9 characters or empty).
- **title**: The title of the journal (maximum length of 1000 characters).
- **volume** and **issue**: The volume and issue of the journal (optional).

3. Article_Journal Table

The `Article_Journal` table establishes a many-to-many relationship between articles and journals. It includes:

- **journal_id**: Foreign key referencing `Journal(id)`.
- **article_id**: Foreign key referencing `Article(id)`.

4. Authors Table

The `Authors` table stores information about the authors of articles. Key columns include:

- **author_id**: A unique identifier for the author.
- **fore_name**, **last_name**, **initials**: The author's first name, last name, and initials.
- **is_collective_name**: A flag indicating whether the author is a collective name.
- **affiliation**: The author's affiliation (institution or organization).

5. Article_References Table

The `Article_References` table tracks the references between articles. Key columns include:

- **article_id**: Foreign key referencing the `Article` table.
- **reference_id**: Foreign key referencing the referenced article in the `Article` table.

6. Article_Authors Table

The `Article_Authors` table links authors to articles. The key columns are:

- **article_id**: Foreign key referencing `Article(id)`.
- **author_id**: Foreign key referencing `Authors(author_id)`.

7. Publication_Types Table

The `Publication_Types` table stores different types of publication. The key columns are:

- **id**: Unique identifier for the publication type.
- **name**: The name of the publication type (e.g., Journal Article, Conference Paper).

8. Article_Publication_Types Table

The `Article_Publication_Types` table links articles to publication types. The key columns are:

- **article_id**: Foreign key referencing `Article(id)`.
- **pub_type_id**: Foreign key referencing `Publication_Types(id)`.

9. Grant_Info Table

The `Grant_Info` table stores information about grants. Key columns include:

- **id**: Unique identifier for the grant.
- **grant_id**: The grant identifier (optional).
- **acronym**: Acronym for the funding agency (optional).
- **agency**: The name of the grant agency.
- **country**: The country of the funding agency (optional).

10. Article_Grants Table

The `Article_Grants` table links articles to grants. The key columns include:

- **article_id**: Foreign key referencing `Article(id)`.
- **grant_id**: Foreign key referencing `Grant_Info(id)`.

11. Article_Ids Table

The `Article_Ids` table stores alternative identifiers (like PubMed, DOI) for articles. Key columns are:

- **id**: A unique identifier for the record.
- **article_id**: Foreign key referencing `Article(id)`.
- **type**: The type of the ID (e.g., DOI, PubMed).
- **identifier**: The actual identifier value.

12. Keywords Table

The `Keywords` table stores keywords associated with articles. Key columns include:

- **id**: A unique identifier for the keyword.
- **keyword**: The actual keyword (cannot be null).

13. Article_Keywords Table

The `Article_Keywords` table links articles to their keywords. The key columns include:

- **article_id**: Foreign key referencing `Article(id)`.
- **keyword_id**: Foreign key referencing `Keywords(id)`.

—

3.2 User Creation and Privilege Descriptions

We use a new table to storage the user information. This SQL statement creates a `users` table with the following columns:

- **id**: Auto-incremented primary key (`SERIAL`).
- **username**: Unique, non-null `VARCHAR(50)`.
- **password**: Non-null `VARCHAR(255)`.
- **role**: Non-null `VARCHAR(20)` with a `CHECK` constraint, allowing only values: `'Site Admin'`, `'Journal Admin'`, `'Article Admin'`, `'Reader'`.
- **created_at**: Timestamp with the current time as default (`CURRENT_TIMESTAMP`).
- **updated_at**: Timestamp with the current time as default (`CURRENT_TIMESTAMP`).
- **Constraints**: Enforces valid roles with a `CHECK` constraint.

Purpose: To store user details, including username, password, and role, ensuring data consistency and validation.

1. User Registration

The ‘registerUser(User user)‘ method allows for the creation of a new user by providing the username, password, and role information.

- The method first checks if the username already exists in the database. If it does, registration fails.
- If the username is available, the user's data (including role) is inserted into the ‘users‘ table.
- The ‘role‘ field determines the user's permissions, which can be roles like "Journal Admin", "Article Admin", "Reader", or "Site Admin".

2. User Login

The ‘login(String username, String password)‘ method validates user credentials during login.

- The method retrieves the stored password from the database for the given username.
- It then compares the entered password with the stored one. If they match, the login is successful.
- The user's role, retrieved upon successful login, determines the resources they are permitted to access.

3. Find User by Username

The ‘findByUsername(String username)‘ method is used to retrieve the user's details based on their username.

- This method queries the ‘users‘ table and returns a user object containing the username, password, and role.
- The role field dictates the user's access rights to the system's resources.

4. Update User Role

The ‘updateUserRole(Long userId, String newRole)‘ method allows system administrators to change a user's role.

- The method updates the ‘role‘ field in the ‘users‘ table for the specified user ID.
- This role update affects the user's access permissions, such as allowing or restricting access to specific resources (e.g., a "Journal Admin" can access more resources compared to a "Reader").

5. Delete User

The ‘deleteUser(Long userId)‘ method deletes a user from the system.

- The method removes the user record from the ‘users‘ table using the provided user ID.
- If the deletion is successful, all associated information tied to the user is also removed from the system.

3.3 User Permission Control

In this system, we utilize **Role-Based Access Control (RBAC)**. User roles are used to control access to different resources and functionalities. Each user is assigned a role (e.g., "Journal Admin", "Article Admin", "Reader", "Site Admin"), and these roles determine which APIs the user can access. For instance, a "Journal Admin" has access to all journal, article, and author data, while a "Reader" can only access article-related APIs.

The user role is stored in the ‘role‘ field in the database and is checked during login and when accessing restricted endpoints. This ensures that only authorized users can access specific resources.

For more details on the role-based access control implementation, please refer to *Section 5.4*.

Role	Accessible APIs
JOURNAL_ADMIN	Can access all journal, article, and author-related APIs and all interfaces
ARTICLE_ADMIN	Can access article and author-related APIs
READER	Can only access article APIs
SITE_ADMIN	Can access all APIs

—

4 Basic API Specification

4.1 Managing Citation Count with Temporary Table

In this project, to optimize frequent citation count queries, we have implemented a temporary table called `Article_Citation_Count` that holds the citation counts for each article. This table is initialized once at the start of the program, and it significantly improves query performance by reducing the need for repetitive and computationally expensive counting operations.

4.1.1 Principle of the Implementation

The main goal of this implementation is to avoid recalculating the citation count for articles on every query. Instead, we maintain a temporary table that stores the citation counts, and we update this table when new citations are added or removed. The process is as follows:

1. **Initialization of the Temporary Table**: - When the application starts,`initializeTempTable()` method is invoked asynchronously using the `@Async` annotation. This method performs the following actions: - It creates the `Article_Citation_Count` table if it does not already exist. - It initializes the citation counts by querying the `article_references` table, which stores the relationships between articles and their references. The count of references is inserted into the `Article_Citation_Count` table using an `INSERT` statement.

2. **Increment and Decrement Citation Counts**: - Whenever a new citation is added or removed, the citation count in the `Article_Citation_Count` table is updated using `UPDATE` statements. - The methods `incrementCitationCount()` and `decrementCitationCount()` handle this by either incrementing or decrementing the citation count for a specific article. - If the article does not already exist in the table, the citation count is inserted with the `INSERT` statement.

3. **Querying the Citation Count**: - When the citation count for a specific article is needed, the `getCitationCount()` method queries the `Article_Citation_Count` table, returning the current citation count. This avoids the need for repeated expensive calculations by directly accessing the precomputed data.

4. **Asynchronous Operation**: - The table initialization happens asynchronously, ensuring that the application can continue running without waiting for the table to be created. The `@Async` annotation allows this operation to be performed in the background.

5. **Cleanup**: - Upon application shutdown, the temporary table is removed by the `cleanup()` method, which drops the table from the database, ensuring that no unused data remains.

4.1.2 Benefits of the Approach

- **Performance**: By maintaining a temporary table that stores precomputed citation counts, we avoid the need for costly joins and aggregate operations each time we need the citation data. This approach significantly speeds up query responses.

- **Scalability**: As the number of articles and references grows, the performance of querying citation counts remains stable because the complex counting operation is done only once during initialization.

- **Simplicity**: Using a temporary table simplifies the logic of updating and retrieving citation counts, making the codebase cleaner and more maintainable.

4.1.3 Design of the Temporary Table

The `Article_Citation_Count` temporary table is designed to store each article's citation count efficiently. The table includes three main columns:

- **article_id**: This column stores the ID of the article. It is a foreign key that references the `Article` table, ensuring that only valid article IDs are stored. The table is indexed by this column, making lookups efficient.

- **citation_count**: This column stores the citation count for each article. The default value is set to 0, and it is updated whenever a citation is added or removed. The column is of integer type, which efficiently handles large counts.

- **citation_year**: This column stores the year in which the citation occurred. It helps in organizing citations by their respective years.

To ensure data consistency and avoid duplicate entries, the table uses a composite primary key constraint on `article_id` and `citation_year`. Additionally, an `INSERT ON CONFLICT DO NOTHING` strategy is used during initialization to prevent overwriting existing counts, which ensures that the initial insertion only adds new records without modifying existing data.

By using this temporary table, the system can quickly and efficiently retrieve citation counts for any article without performing costly joins or aggregations on the `article_references` table for each query.

4.2 Article Service Implementation

The 'ArticleServiceImpl' class implements the 'ArticleService' interface and provides methods for managing articles, journals, and citations. It interacts with the database using JDBC and executes various SQL queries to perform tasks such as retrieving citation counts, adding articles, and calculating journal impact factors.

1. Get Article Citations by Year

The 'getArticleCitationsByYear(int id, int year)' method retrieves the number of times an article with a given ID has been cited in a specific year.

- The method constructs an SQL query that joins the 'article_references' and 'article' tables.
- It uses 'EXTRACT(YEAR FROM a.date_created)' to filter the references by the specified year.
- The method returns the citation count if found, or '0' if no citations are recorded.

2. Add Article and Update Impact Factor

The 'addArticleAndUpdateIF(Article article)' method adds a new article to the database and calculates the journal's impact factor based on articles published in the last two years.

- The article is inserted into the 'Article' table, and the article-journal relationship is recorded in the 'Article_Journal' table.
- If the article has references, the citation count for each referenced article is incremented using 'CitationCountManager'.
- Two SQL queries retrieve articles published in the last two years and their citation counts.
- The impact factor is calculated as the ratio of total citations to total articles.
- The method returns the calculated impact factor and deletes the article and associated data if necessary.

3. Insert Article and Journal Relationship

The ‘insertArticleAndJournal(Article article)‘ method inserts a new article and its corresponding journal into the database.

- The article is inserted into the ‘Article‘ table.
- The method checks if the journal already exists in the ‘Journal‘ table. If not, a new journal is added.
- It then creates an entry in the ‘Article_Journal‘ table to associate the article with the journal.

4. Delete Article and Associated Data

The ‘deleteArticleAndAssociatedData(Article article)‘ method deletes an article and its associated data from the database.

- The method first deletes the entry in the ‘Article_Journal‘ table that links the article to its journal.
- The article is then removed from the ‘Article‘ table.
- If no other articles are associated with the journal, the journal is also deleted.

5. Insert Article-Journal Relationship

The ‘insertArticleJournal(Article article)‘ method inserts the relationship between the article and its journal into the ‘Article_Journal‘ table.

- The method checks if the journal already exists. If it does, the journal’s ‘id‘ is used.
- If the journal does not exist, it is inserted into the ‘Journal‘ table, and then the article-journal relationship is added to ‘Article_Journal‘.

—

This class efficiently handles article and journal management, citation counting, and impact factor calculation using SQL queries and the ‘CitationCountManager‘ class for citation updates. By leveraging ‘PreparedStatement‘ and managing database connections properly, it ensures both performance and maintainability.

4.3 Author Service Implementation

The ‘AuthorServiceImpl‘ class implements the ‘AuthorService‘ interface and provides methods for managing authors, their articles, citation counts, and the journals they publish in. It interacts with the database using JDBC to execute various SQL queries.

1. Get Articles by Author Sorted by Citations

The ‘getArticlesByAuthorSortedByCitations(Author author)‘ method retrieves all articles written by a given author and sorts them by citation count in descending order.

- The method constructs an SQL query that joins the ‘Article‘, ‘Article_Authors‘, and ‘Authors‘ tables to retrieve articles by the author, identified by first and last name.
- For each article, the method calls ‘CitationCountManager.getCitationCount()‘ to get the citation count.
- After collecting the citation counts, the method sorts them in descending order.
- It then returns an array of citation counts sorted in descending order.

2. Get Journal with Most Articles by Author

The ‘getJournalWithMostArticlesByAuthor(Author author)‘ method retrieves the journal with the most articles published by the specified author.

- The method constructs an SQL query that joins the ‘Journal‘, ‘Article_Journal‘, ‘Article_Authors‘, and ‘Authors‘ tables.
- It groups the results by journal title and orders them by the count of articles published in each journal.
- The query limits the result to the journal with the most articles.
- The method returns the title of the journal with the most articles. If the author has no articles, it returns ‘null‘.

3. Get Minimum Articles to Link Two Authors

The ‘getMinArticlesToLinkAuthors(Author A, Author E)‘ method calculates the minimum number of articles needed to link two authors through citations using a breadth-first search (BFS) algorithm.

- The method first retrieves the articles of both authors using ‘getAuthorArticles()‘.
- It initializes a BFS queue and starts by adding all articles from author A with depth 0.
- The BFS algorithm explores the references of each article until it finds an article written by author E.
- If a connection is found, the method returns the number of articles (depth) needed to link the authors. If no connection exists, it returns ‘-1‘.

4. Helper Method: Get Articles by Author

The ‘getAuthorArticles(Author author)‘ method retrieves all articles written by the specified author.

- The method constructs an SQL query that joins the ‘Article‘, ‘Article_Authors‘, and ‘Authors‘ tables to get all articles associated with the author based on their first name, last name, and initials.
- The method executes the query and collects the article IDs in a ‘Set‘ to avoid duplicates.

5. Helper Method: Check if Article is Written by Author

The ‘isArticleWrittenByAuthor(int articleId, Author author)‘ method checks if a given article is written by the specified author.

- The method constructs an SQL query to check if an article is written by the author using the author’s name, last name, and initials.
- It returns ‘true‘ if the article is written by the author, otherwise ‘false‘.

6. Helper Method: Get Article References

The ‘getArticleReferences(int articleId)‘ method retrieves all references for a given article.

- The method constructs an SQL query that selects all reference IDs from the ‘article_references‘ table for the specified article.
- It collects all referenced article IDs in a ‘Set‘ to ensure uniqueness.

—

The ‘AuthorServiceImpl‘ class effectively manages author-related operations such as retrieving articles, calculating citation counts, and determining the journal with the most articles. It utilizes efficient SQL queries and helper methods to manage articles, references, and relationships between authors and journals.

4.4 Grant Service Implementation

The ‘GrantServiceImpl‘ class implements the ‘GrantService‘ interface and provides methods for managing grants and associated articles. Specifically, it includes functionality to retrieve articles funded by a specific country.

1. Get Funded Papers by Country

The ‘getCountryFundPapers(String country)‘ method retrieves the PMIDs of all articles funded by a given country.

- Step 1: The method constructs an SQL query that joins the ‘Article‘, ‘Article_Grants‘, and ‘Grant_info‘ tables. The core query is:

```
SELECT a.id AS pmid
FROM Article a
JOIN Article_Grants ag ON a.id = ag.article_id
JOIN Grant_info g ON ag.grant_id = g.id
WHERE g.country = ?
```

  The ‘Article‘ table stores the article's basic information, the ‘Article_Grants‘ table records the relationship between articles and grants, and the ‘Grant_info‘ table stores detailed information about grants. The query uses ‘JOIN‘ to link these tables based on the article ID (‘article_id‘) and grant ID (‘grant_id‘), filtering the results by the ‘country‘ of the grant.

- Step 2: The method uses ‘PreparedStatement‘ to set the country parameter for the query.
- Step 3: It executes the query and processes the result set, adding each article's ‘pmid‘ (article ID) to a list.
- Step 4: Finally, the list of PMIDs is converted into an ‘int[]‘ array and returned.

Summary

The method efficiently retrieves the ‘pmid‘ (article ID) of all articles funded by a specific country by joining the ‘Article‘, ‘Article_Grants‘, and ‘Grant_info‘ tables. The query leverages the ‘country‘ field in the ‘Grant_info‘ table to filter the results based on the given country.

4.5 Journal Service Implementation

The ‘JournalServiceImpl‘ class implements the ‘JournalService‘ interface and provides functionality for calculating the journal impact factor and updating journal information.

1. Calculate Journal Impact Factor for a Given Year

The ‘getImpactFactor(String journal_title, int year)‘ method calculates the impact factor (IF) for a given journal in the specified year.

- **Step 1**: The method constructs an SQL query to retrieve the IDs of articles published in the last two years by the specified journal, identified by ‘journal_title‘.

```
SELECT a.id FROM Article a
JOIN Article_Journal aj ON a.id = aj.article_id
JOIN Journal j ON aj.journal_id = j.id
WHERE j.id = ? AND EXTRACT(YEAR FROM a.date_completed) IN (?, ?)
```

- **Step 2**: The method uses ‘CitationCountManager.getCitationsInYear()‘ to retrieve and accumulate the citation counts for these articles.
- **Step 3**: It calculates the total number of articles published in the last two years by executing another SQL query:

```
SELECT COUNT(*) AS total_articles FROM Article a
JOIN Article_Journal aj ON a.id = aj.article_id
JOIN Journal j ON aj.journal_id = j.id
WHERE j.title = ? AND EXTRACT(YEAR FROM a.date_completed) IN (?, ?)
```

- **Step 4**: The method calculates the impact factor using the formula:

$$\text{Impact Factor} = \frac{\text{Total Citations}}{\text{Total Articles}}$$

If no articles are found, it logs a warning and returns 0.

2. Update Journal Name and ID

The ‘updateJournalName(Journal journal, int year, String new_name, String new_id)‘ method updates the journal name and ID, and updates all articles published from the specified year onward with the new journal information.

- **Step 1**: The method inserts the new journal information into the ‘Journal‘ table with the following SQL query:

```
INSERT INTO Journal (id, title, country, issn, volume, issue)
VALUES (?, ?, '', '', '', '')
```

- **Step 2**: It updates the ‘Article_Journal‘ table by changing the ‘journal_id‘ for articles published from the specified year onward:

```
UPDATE Article_Journal
SET journal_id = ?
WHERE journal_id = ?
AND article_id IN (
    SELECT a.id
    FROM Article a
    WHERE EXTRACT(YEAR FROM a.date_completed) >= ?
)
```

- **Step 3**: The method executes both operations within a transaction to ensure data consistency. If any error occurs, the transaction is rolled back.

4.6 Keyword Service Implementation

The ‘KeywordServiceImpl‘ class implements the ‘KeywordService‘ interface and provides functionality for querying article counts related to a specific keyword.

1. Get Article Count by Keyword in Past Years

The ‘getArticleCountByKeywordInPastYears(String keyword)‘ method retrieves the number of articles published in the past years that contain the specified keyword, sorted by year in descending order.

- Step 1: The method constructs an SQL query that joins the ‘Article‘, ‘Article_Keywords‘, and ‘Keywords‘ tables to retrieve the number of articles associated with the given keyword.

```
SELECT EXTRACT(YEAR FROM a.date_completed) AS year, COUNT(*) AS article_count
FROM Article a
JOIN Article_Keywords ak ON a.id = ak.article_id
JOIN Keywords k ON ak.keyword_id = k.id
WHERE k.keyword = ?
GROUP BY year
ORDER BY year DESC
```

The query extracts the year from ‘article.date_completed‘, counts the articles for each year, and orders the results by year in descending order.

- Step 2: The method uses ‘PreparedStatement‘ to set the keyword parameter for the query.
- Step 3: It executes the query and processes the result set, adding the article counts for each year to a list.
- Step 4: The list of article counts is converted into an ‘int[]‘ array and returned.

5 Advanced APIs - GUI

In this project, we have exposed several backend APIs through RESTful endpoints to facilitate interaction between the front-end and back-end systems. These APIs provide access to various services, including managing journals, articles, authors, grants, and user management.

5.1 User Management APIs

The ‘UserController‘ is responsible for managing user-related requests. It exposes three main API endpoints:

- **Login**: The ‘/api/users/login‘ endpoint allows users to log in by providing their username and password. The controller calls the ‘login‘ method in the ‘UserService‘ to validate the credentials and returns a boolean indicating whether the login was successful.
- **Register**: The ‘/api/users/register‘ endpoint enables users to register by sending their username, password, and role information. The controller invokes the ‘registerUser‘ method in the ‘UserService‘ class to register the new user.
- **Get User Information**: The ‘/api/users/username/username‘ endpoint retrieves the details of a user based on the provided username. The ‘getUserByUsername‘ method in the controller calls ‘findByUsername‘ in the ‘UserService‘ class to fetch the user information from the database.

5.2 Journal Management APIs

The ‘JournalController‘ exposes two main API endpoints related to journal management:

- **Get Impact Factor**: The ‘/api/journals/journalId/impact-factor‘ endpoint retrieves the journal impact factor for a specific year. It allows users to query the impact factor based on the journal's published articles.
- **Update Journal Information**: The ‘/api/journals/update‘ endpoint allows users to update the journal's name and ID, and specify the year from which the new details should take effect.

5.3 Article, Author, and other APIs

Similarly, other controllers have been implemented to handle requests related to articles, authors, and grants. These controllers expose relevant endpoints for operations such as:

- Retrieving articles by keyword.
- Managing article references.
- Updating user roles.

5.4 Backend Permission Control

In this project, we have implemented backend permission control to ensure the security of the system and data protection. Frontend-based permission control can be bypassed, so the backend must strictly control user access. We adopted Role-Based Access Control (RBAC) to achieve this, restricting users' access to specific resources and functionalities based on their roles.

5.4.1 User Role Management

Each user is assigned a role during registration (e.g., "Journal Admin", "Article Admin", "Reader", "Site Admin"), which determines the resources and functionalities the user can access. The permissions for each role are as follows:

5.4.2 User Registration and Role Assignment

During user registration, the system first checks if the username already exists. If the username is available, the user's data (including username, password, and role) is inserted into the 'users' table. The 'role' field determines the user's permissions, for example, the "Journal Admin" role allows the user to access all journal, article, and author data.

5.4.3 User Login and Role Validation

During user login, the system verifies the user's identity using JWT tokens. The JWT token contains the user's role information, which is extracted upon successful login and used to grant the user appropriate access to the system's resources.

5.4.4 Role-Based Permission Control

Role-based permission control is implemented using the custom `@RequireRole` annotation and role interceptors. Each method that requires specific roles to access is marked with the `@RequireRole` annotation, which specifies which roles can access the API. The interceptor then checks the user's role before the request reaches the controller to ensure they have the required role to access the resource.

5.4.5 Permission Control Implementation

- **Role Interceptor**: The `RoleInterceptor` intercepts each request and checks the JWT token in the request. It validates whether the user's role matches the required role to access the requested resource. If the role is not authorized, a "Forbidden" response is returned.

- **Role Annotations**: The `@RequireRole` annotation is used on controller methods to specify which roles are allowed to access them. The access to APIs is fine-tuned based on user roles.

- **Database Design**: User information, including roles, is stored in the `users` table. The `role` field directly affects the user's access permissions to different resources.

5.4.6 Role Permission Management

We define different user roles and restrict access based on these roles. The roles and their accessible APIs are as follows:

Role	Accessible APIs
JOURNAL_ADMIN	Can access all journal, article, and author-related APIs and all interfaces
ARTICLE_ADMIN	Can access article and author-related APIs
READER	Can only access article APIs
SITE_ADMIN	Can access all APIs

6 Frontend Architecture and Technical Features

6.1 Frontend Architecture Design

1. Modular Design

- The API service is modularized (api.js).
- Different functionality interfaces are categorized and managed according to business domains.
- Achieved high cohesion and low coupling in the code structure.

2. Layered Architecture

- **Presentation Layer**: HTML pages (index.html, login.html, register.html).
- **Service Layer**: API interface encapsulation (api.js).
- **Data Layer**: Local storage management (localStorage).

3. Permission Management Design

- Role-Based Access Control (RBAC).
- Dynamic UI rendering based on user roles.
- JWT token-based authentication mechanism.

6.2 Technical Features

1. Modern JavaScript Features

- Uses async/await for handling asynchronous operations.
- Promises with chainable calls.
- ES6+ features like destructuring assignment and template literals.

2. Frontend Optimization

- Unified error handling mechanism.
- Request-response interception.
- Dynamic content loading.

3. User Experience

- Responsive design for different devices.
- Real-time feedback mechanism.
- Friendly error messages and prompts.

This architecture design ensures good maintainability and scalability of the system while ensuring a seamless user experience.