CS307-Database Project 1

Group session: Thursday 3-4

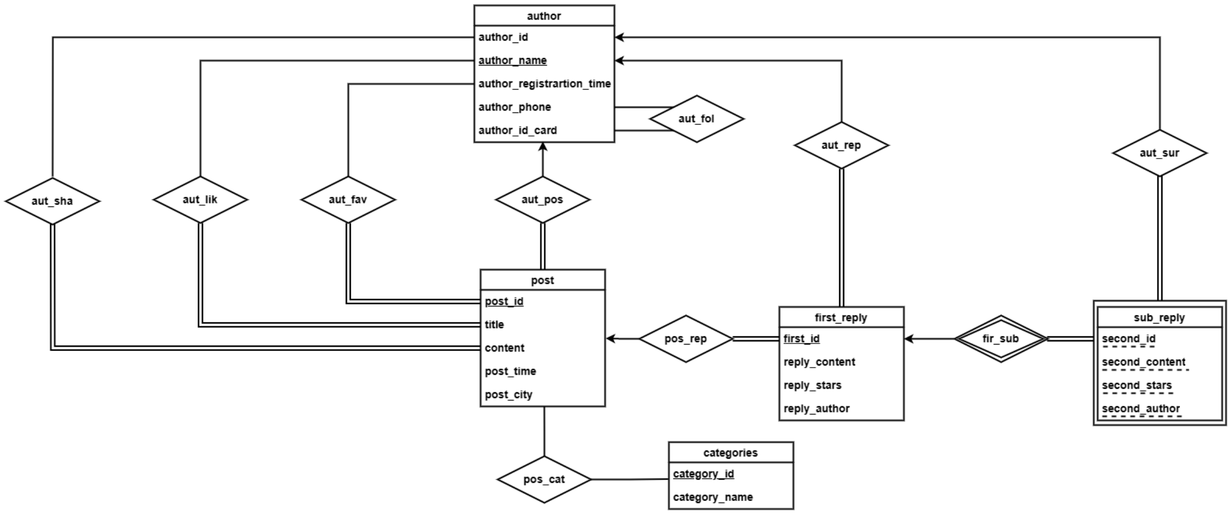
Name(SID): 钟志源(12110517)、刘浩贤(12111515)

Contribution: 钟志源（Database Design、Data Import）刘浩贤（E-R Diagram、 Database Design）

percentages of contributions： 50% : 50%

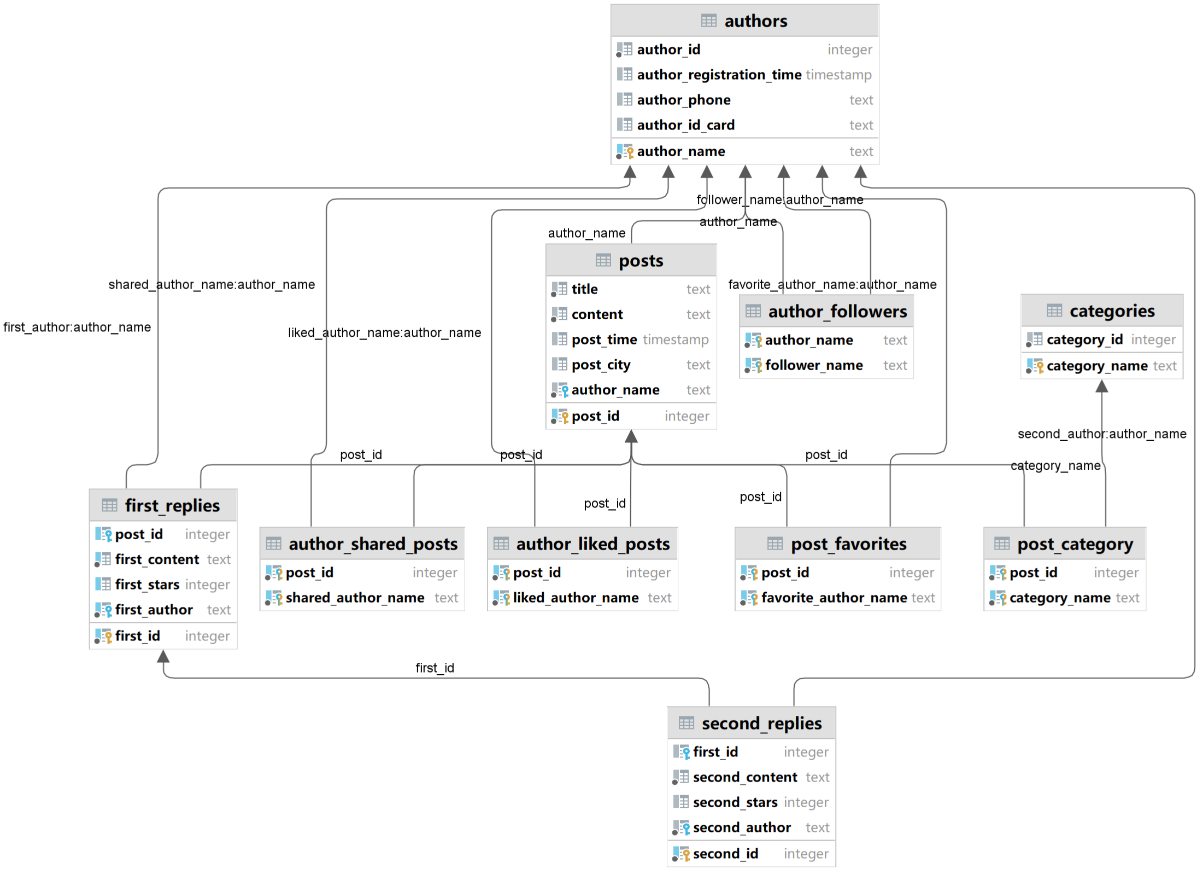
Task 1: E-R Diagram

- The E-R Diagram is draw with diagrams.net



Task 2: Relational Database Design

- The E-R diagram generated by DataGrip



- **The create table statements is attached with `CreateTable.sql`**
- **Briefly describe the table designs and the meanings of each table and column.**

1. **Table Name: authors (Entity set)**
    - Description: This table stores information about authors, including their ID, name, registration time, phone number, and ID card.
    - Columns:
        - author_id: a unique identifier for each author, automatically incremented by the system (SERIAL type)
        - author_name: the author's name (text type, not null and unique, **primary key**)
        - author_registration_time: the date and time when the author registered (TIMESTAMP type)
        - author_phone: the author's phone number (text type)
        - author_id_card: the author's identification card number (text type)
2. Table Name: posts (Entity set)
    - Description: This table stores information about posts made by authors, including the post's title, content, posting time, location, and the author who made the post.
    - Columns:
        - post_id: a unique identifier for each post, automatically incremented by the system (SERIAL type, **primary key**)
        - title: the title of the post (text type, not null)

        - content: the content of the post (text type, not null)
        - post_time: the date and time when the post was published (TIMESTAMP type)
        - post_city: the city where the post was published (text type)
        - author_name: the name of the author who published the post (text type, foreign key to authors.author_name, not null)

3. **Table Name: categories (Entity set)**
    - Description: This table stores information about post categories, including the category's ID and name.
    - Columns:
        - category_id: a unique identifier for each category, automatically incremented by the system (SERIAL type)
        - category_name: the name of the category (text type, not null and unique, **primary key**)
4. **Table Name: post_category (Relationship set)**
    - Description: This table represents the many-to-many relationship between posts and categories. Each row represents a post and its associated category.
    - Columns:
        - post_id: the id of the post (integer type, foreign key to posts.post_id)
        - category_name: the name of the category (text type, foreign key to categories.category_name)
    - **Primary Key**: (post_id, category_name)
5. **Table Name: author_followers (Relationship set)**
    - Description: This table represents the many-to-many relationship between authors and their followers. Each row represents an author and their follower.
    - Columns:
        - author_name: the name of the author being followed (text type, foreign key to authors.author_name)
        - follower_name: the name of the follower (text type, foreign key to authors.author_name, not null)
    - **Primary Key**: (author_name, follower_name)
6. **Table Name: post_favorites (Relationship set)**
    - Description: This table represents the many-to-many relationship between posts and authors who have marked them as favorites. Each row represents a post and the author who marked it as a favorite.
    - Columns:
        - post_id: the id of the post being favorited (integer type, foreign key to posts.post_id)
        - favorite_author_name: the name of the author who favorited the post (text type, foreign key to authors.author_name, not null)
    - **Primary Key**: (post_id, favorite_author_name)

7. **Table Name: author_shared_posts (Relationship set)**
    - Description: This table represents the many-to-many relationship between authors and posts that they have shared. Each row represents a post and the author who shared it.
    - Columns:
        - post_id: the id of the post being shared (integer type, foreign key to posts.post_id)
        - shared_author_name: the name of the author who shared the post (text type, foreign key to authors.author_name, not null)
    - **Primary Key**: (post_id, shared_author_name)
8. **Table Name: author_liked_posts (Relationship set)**
    - Description: This table represents the many-to-many relationship between authors and posts that they have liked. Each row represents a post and the author who liked it.
    - Columns:
        - post_id: the id of the post being liked (integer type, foreign key to posts.post_id)
        - liked_author_name: the name of the author who liked the post (text type, foreign key to authors.author_name, not null)
    - **Primary Key**: (post_id, liked_author_name)
9. **Table Name: first_replies (Entity set)**
    - Description: This table stores information about the first reply to a post, including the reply's ID, content, rating, and author.
    - Columns:
        - post_id: the id of the post being replied to (integer type, foreign key to posts.post_id)
        - first_id: a unique identifier for each first reply, automatically incremented by the system (SERIAL type, **primary key**)
        - first_content: the content of the first reply (text type, not null)
        - first_stars: the number of stars received by the first reply (integer type)
        - first_author: the name of the author who wrote the first reply (text type, foreign key to authors.author_name, not null)
10. **Table Name: second_replies (Entity set)**
    - Description: This table stores information about the second reply to a post, including the reply's ID, content, rating, and author, as well as the ID of the first reply that it is associated with.
    - Columns:
        - second_id: a unique identifier for each second reply, automatically incremented by the system (SERIAL type, **primary key**)
        - first_id: the id of the first reply being replied to (integer type, foreign key to first_replies.first_id)
        - second_content: the content of the second reply (text type, not null)
        - second_stars: the number of stars received by the second reply (integer type)

- second_author: the name of the author who wrote the second reply (text type, foreign key to authors.author_name, not null)

Task3

Task3.1 Data Import

The script consists of 4 files: `dbUser.properties`,`Main`,`Post`,`Replies`.

`dbUser.properties` contains the information of database and its user, including `host`,`database`,`user`,`password`,`port`, in order to connect to the database.

`Post` is a java class to create corresponding java object from the json data. Similarly for class `Replies`.

`Main` file is used to import data. **The basic steps are as follows**:

1. Load database user information from `dbUser.properties`.
2. Connect to database using `postgresql.Driver`.
3. Clear data in relevant tables(and create relevant empty tables).
4. Load data from `posts.json` to a `List<Post> posts`.
5. Load data from `replies.json` to a `List<Replies> replies`.
6. Start the timer.
7. Prepare insert statements.
8. Traverse `posts` and `replies`, extract attributes out, set statements' parameters, add to batch.
9. Execute batch. `con.commit()` to commit changes to database.
10. Close database connection.
11. Stop the timer.

Prerequisites: Make sure the `dbUser.properties` is in a directory called `resources` under the project, make sure the directory `lib` contains `fastjson.jar` and `postgresql.jar` and add `lib` **as library**.

Cautions: Make sure that `posts.json` and `replies.json` are in the same directory as `Main`. Make sure that there are **NO space** in the attributes name in the json file. **Before** executing `Main`, please create the corresponding tables in advance!

For the script, please refer to the attachments.

Task3.2 Efficiency Comparison

In the `Main` file, we use **`PreparedStatement`**,**`Transaction`** and **`Batch`** to improve performance and security.

In `loader1NoPrepare`, we use normal `Statement` to execute sql inserts. Since there could be `'` in an English sentence, SQL injection problem happened and data import failed.

In `loader2Prepare`, we use `PreparedStatement` to **precompile** the SQL statement once and then execute it multiple times with different parameter values. It helps to prevent SQL injection attacks by automatically escaping special characters in user input. Additionally, `PreparedStatement` can improve performance by **caching** the compiled SQL statement, reducing the overhead of repeatedly parsing and optimizing the statement.
On average: 4400 ms

```
53308 records successfully inserted.
Insertion speed: 11573 insertions/s
Time spent: 4406 ms
```

In `loader3Transaction`, we added `Transaction`. We start a `transaction` by disabling auto-commit mode, and then perform the database operations. If all the operations are successful, we commit the transaction. By grouping multiple operations into a single transaction, it reduces the number of round-trips between the Java application and the database.
On average: 1550 ms.

```
53308 records successfully inserted.
Insertion speed: 33151 insertions/s
Time spent: 1608 ms
```

In `Main`, we added `Batch`. It allows multiple SQL statements to be executed as a single batch, reducing the number of round-trips between the Java application and the database.
On average: 600 ms.

```
53308 records successfully inserted.
Insertion speed: 93359 insertions/s
Time spent: 571 ms
```

Test environment: Apple MacBook Pro 2021 (M1 pro,8 cores) 16GB RAM, macOS 12.6.3. To summarize, `Batch` inserts can be useful for inserting large amounts of data, `PreparedStatement` can be useful for executing similar SQL statements multiple times, and `transactions` can be useful for ensuring data consistency.