2DV513 Database Theory

Assignment 3

Zacky Kharboutli . (zk222ay)

December 2020

1 Introduction

For the assignment, a book management system was created. The applications user is considered as an administrator who can manage all kind of functionality of the application such as creating users, books, book types and genders. The application has a user-friendly UI where the user can easily navigate between pages and them. There are two views for books and users where the user sees a list of the existing data and filter this list by multiple filter criteria.

The application is developed and written in Javascript and NodeJs. Express is used as framework and Handlebars is the view engine. The database that is used in the app is MySQL.

Source code : `https://github.com/zackkyyy/databasetheoryA3`
Video demonstration : `https://vimeo.com/501440871`

1.1 Application Requirements

- CRUD functions on book and user models
- Search and filtering on books and users
  - Find user by name
  - Find all users by gender
  - Find Book by name
  - Find all books by author
  - Find all book by category

2 Logical Model

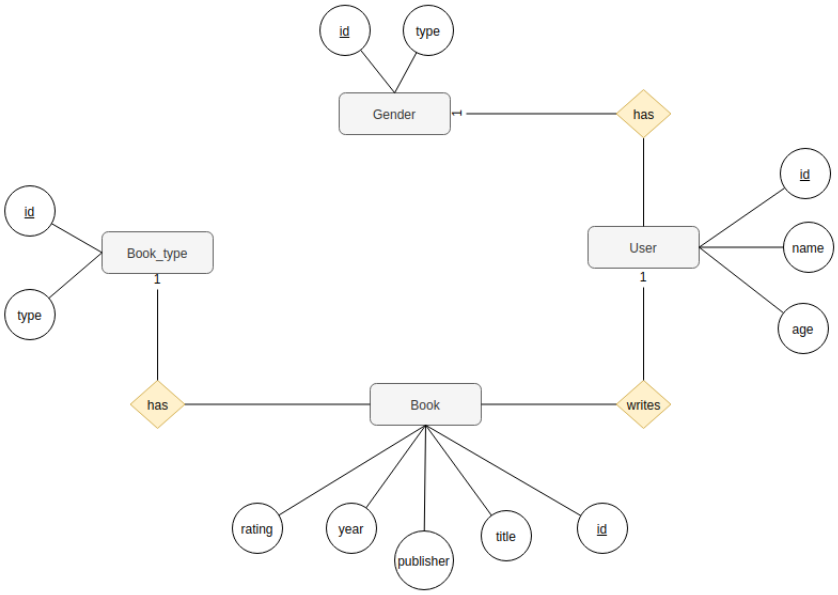

FIGURE 1 – Entity Relation Diagram for the application

2.1 About the design

There are two main entities or models in this application which are the book and the user. Each of them has multiple attributes such as name and age for the user and title, publisher, category, publishing year and rating for the book. In the design, I intended to create the gender of the user and the type of the book as their own entities as this way will help in manipulating the data when updating the main entity. The values of these two entities are countable and therefore having them as an attribute in their perspective tables seemed unnecessary as there will be a lot of repentance in the data.

The user for the book is the author which means one user can have many books. Same works for gender and book type, for example as one gender can belong to many users and a book type belongs to many books ! On the contrary, the book cannot have many users or many book type, and the user cannot have many gender.

3 Database Design

The Entity relationship diagram is translated and developed into the following tables. Each table has a primary key ID and they are auto incremented. All columns are made to be not null. Names for genders and book types are unique so that the type wont be repeated.

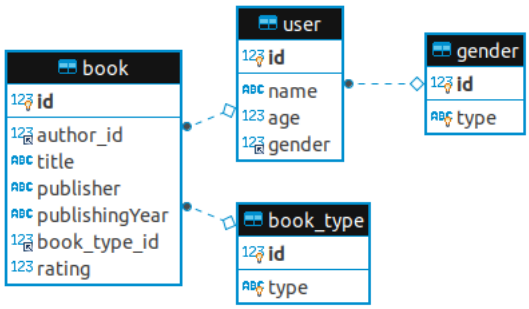

FIGURE 2 – Tables design (Powered by DBeaver)

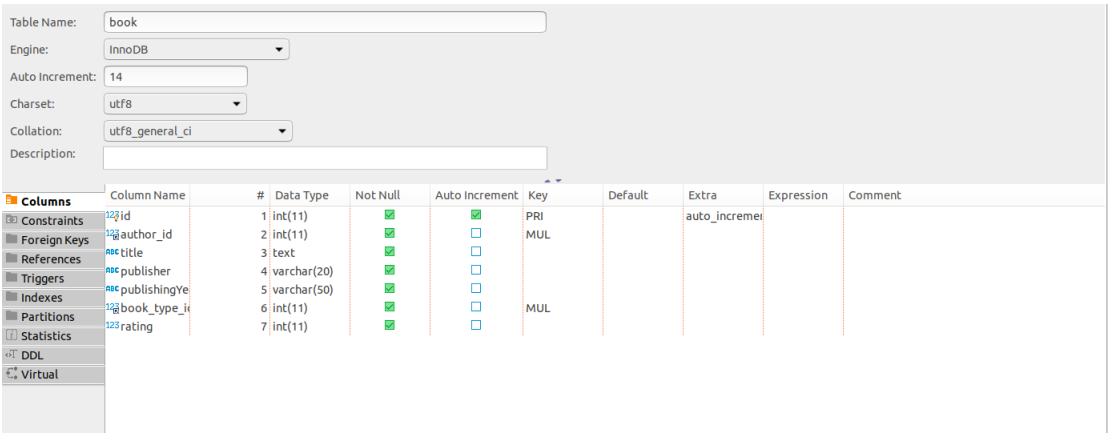

FIGURE 3 – Book table proprieties

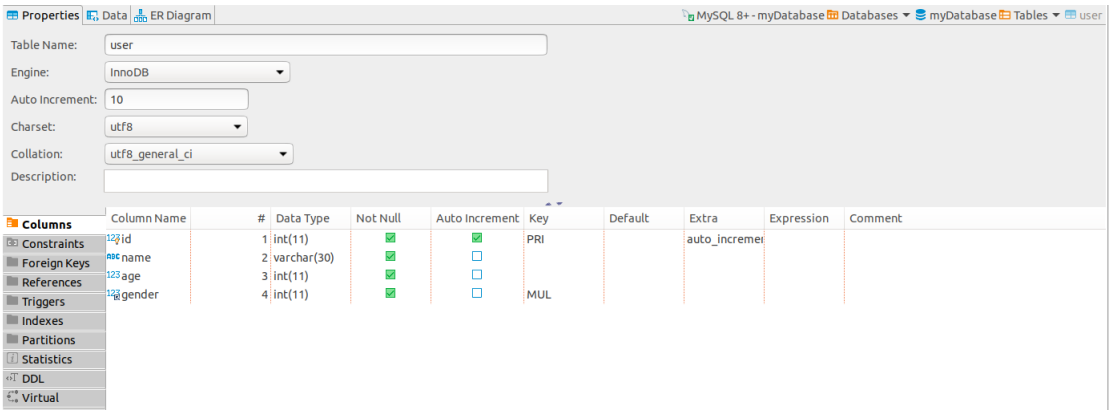

FIGURE 4 – User table proprieties

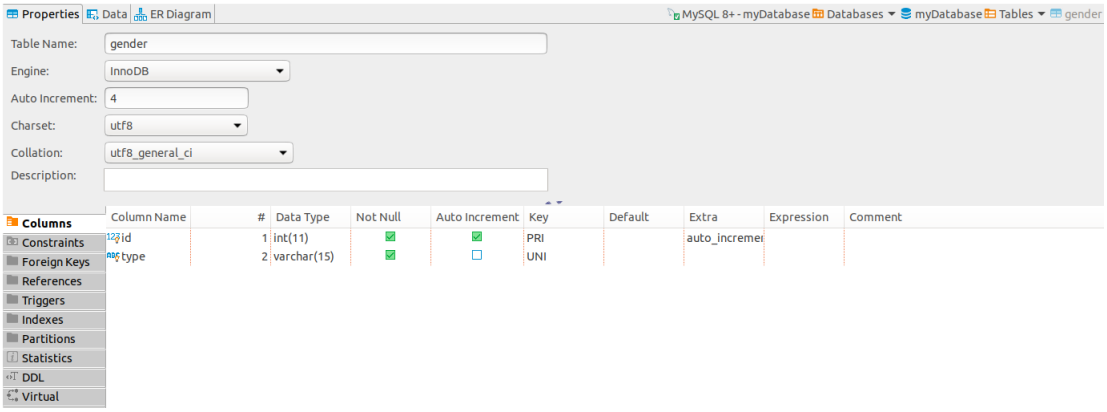

FIGURE 5 – Gender table proprieties

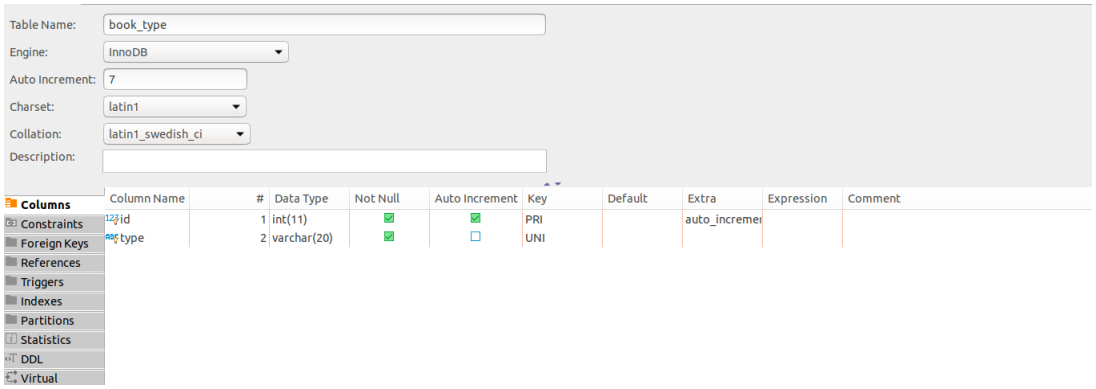

FIGURE 6 – Book type table proprieties

4 Queries

4.1 Book Queries

```
SELECT b.id, u.name AS author, b.author_id, title, publisher,
    publishingYear, rating, book_type.type FROM book b
    LEFT JOIN book_type ON book_type_id = book_type.id
    LEFT JOIN user u ON u.id = b.author_id
```

The query above is used to fetch all books the exists in the database. It joins both users and book_table to import their actual values as the values stored in the book table are their ids.

While for fetching a specific book by its id so we append to the above a query a where clause to get the desired entry as the following example.

```
const GET_BOOK_BY_ID = GET_ALL_BOOKS + " WHERE b.id = ?"
```

Same applies for almost all queries on this table were we filter by one of its columns.

```
WHERE b.id = ?
WHERE b.author_id = ?
WHERE book_type_id = ?
WHERE b.title LIKE ?
```

In the last query, where we search for a book by its title I used LIKE as this will allow to return values that matches the query parameter that is used to find. Using = will result in fetching only results that fully match the query parameter.

In the book page, I fetch the book with all its information to present to the user. On this page the user can see as well all other books that is written by this author(user) and here a where condition is requested on two columns so we append the following to the query that fetches all books.

```
WHERE b.id != ? AND b.author_id = ?
```

The following queries are used for inserting, updating and deleting a book. For a inserting and updating we send a whole object that represent the book

```
INSERT INTO book SET ?
UPDATE book SET ? WHERE id = ?
DELETE FROM book WHERE id = ?
```

4.2 User Queries

```
SELECT u.id, u.name, age, g.type AS gender, (SELECT COUNT(*) from
    book where author_id = u.id) AS nrOfBooks
    FROM user u
    LEFT JOIN gender g ON u.gender = g.id
```

Similar to book table, this query works for fetching all users from the database by joining the gender table to get what gender each user is.

We append on of these three following lines to fetch the users by gender, name or id.

```
WHERE u.id = ?
WHERE u.name LIKE ?
WHERE g.id = ?
```

The three following queries are used in a simlar way to the book queries to INSERT, UPDATE and DELETE a user.

```
DELETE FROM user WHERE id = ?
INSERT INTO user SET ?
UPDATE user SET ? WHERE id = ?
```

4.2.1 User table View

In the view created for the books I added some filters to filter the list with several criteria. One of them is to filter the books by author. In order to achieve that I created view where I store only users that have created books. I used this view in the drop down button so I only get the needed users. The query to select the view was as following :

```
CREATE VIEW v_users_with_book
AS
SELECT DISTINCT u.id, u.name FROM user u LEFT JOIN book b ON u.id =
     b.author_id
WHERE u.id IN (b.author_id);
```

and then all I needed to do is to select all rows from this view and insure I only get users that have books.

4.3 Gender Queries AND Book type Queries

These two tables have been treated in the same way in almost all cases as they have a similar columns.

In the application design, there is no option to edit data in these tables. The only allowed options on these tables are to INSERT, UPDATE and SELECT. There are separate views for both entities where the user can see list of all data and create new one or delete an existing ones.

5 Implementation

The application is build using Javascript and NodeJS. There has been created a web interface for the client part while a RESTFUL APIS is implemented for the server side.

The application connects to a local mysql database which you may have to change in order to run the application. The database configuration could be changed in the dbParser.js file which is located in the root folder.

Error handling in the app is the not the best as I am working alone on the project so any error will cause a termination of the server and you may need to restart the app.

After connection to your own database, there is script to create the database, the tables and insert data into them. The script is found in the root folder as well and it is called startScript.js.

The program is implemented to run this script if the database is not filled already with data. I used a function where I check if the user table exists and filled with at least one row so the script won't run, otherwise the script will run and drop all tables and create them again with inserting some data. Code used for that is :

```
con.connect(function (err) {
    if (err) {
        console.log('Failed to connect to database')
        throw err
    } else {
        let tableExists = (myTable) => {
            // Return a new promise
            return new Promise(resolve => {
                let sql = "SELECT 1 FROM ??";
                con.query(sql, myTable, function(error, result,
    field){

                    resolve(!!result)
                });
            });
        }
        tableExists("user").then(result => result ?  console.log('
    Database is already filled'): dumpData())
        console.log('Database connected!')
    }
})


function dumpData() {
    const dataSql = fs.readFileSync('./startScript.sql').toString()
    const dataArr = dataSql.toString().split(';');
    dataArr.forEach((query) => {
        if(query) {
            con.query(query, function (err, row) {
                err ? console.log('failed to run script') : console
    .log('Filled the database')
            })
        }
    });
}
```

The codes and how to run it is provided on github
Source code : `https://github.com/zackkyyy/databasetheoryA3`
Video demonstration : `https://vimeo.com/501440871`