Can KOZ
Comp. Eng.
ckoz20@ku.edu.tr

Kaan DEMİRER
Comp. Eng.
kdemirer18@ku.edu.tr

Ömer Faruk AKSOY
Comp. Eng.
oaksoy18@ku.edu.tr

Serhan TÜRKAN
MSc. Data Sci.
sturkan20@ku.edu.tr

LIBRARY MANAGEMENT DATABASE SYSTEM

We decided to create a Library Management System that would help organize and maintain a university library. The main purpose of the system is to process the daily borrowing & returning of books by the registered students. The database provides some information about books, authors, students, and librarians. It keeps track of a book's borrow date, return date, and penalty according to due date. After analyzing our design and its requirements further, it became clear that the system must perform many functions in three main areas:

(1) Insert **book**, delete **book**, search **book**, borrow **book**, return **book**

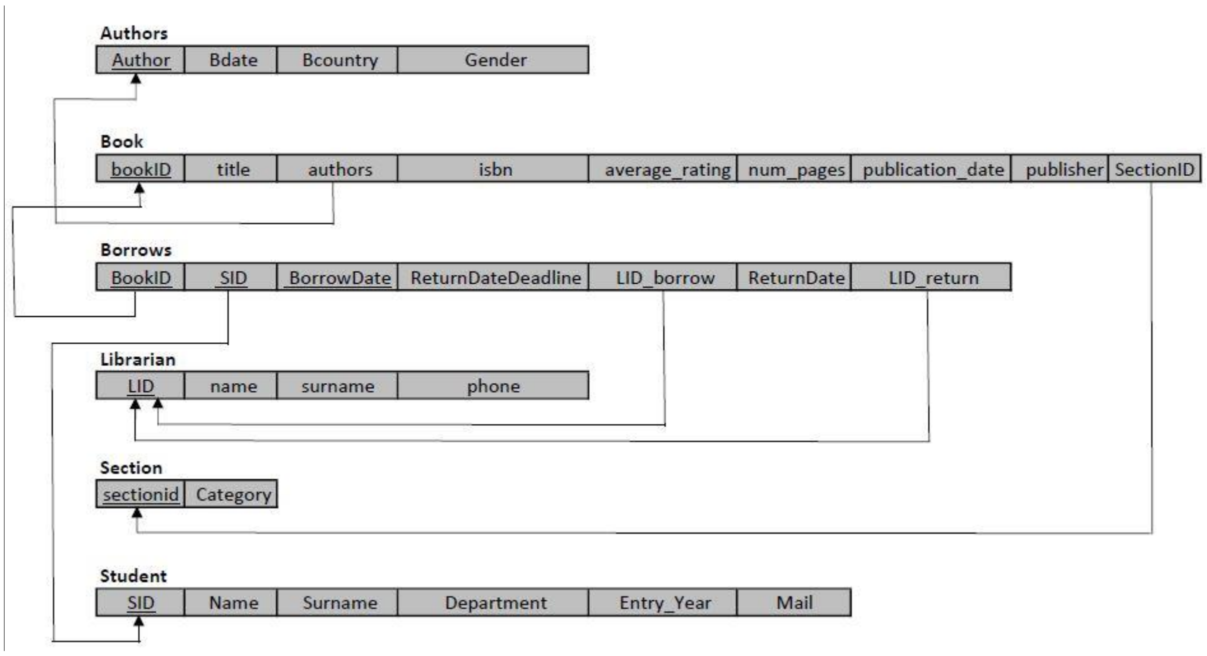
(2) Insert **student**, delete **student**, search **student**

(3) Insert **author**, delete **author**

Furthermore, we wanted to allow the user to be able to easily analyze the data at hand, which we later designed and integrated into the system as part of the **complex queries** requirement of the project.

We chose MYSQL as a Database Management System (DBMS) for this project.

Relational Database Schema


Authors
Author | Bdate | Bcountry | Gender
Book
bookID | title | authors | isbn | average_rating | num_pages | publication_date | publisher | SectionID
Borrows
BookID | SID | BorrowDate | ReturnDateDeadline | LID_borrow | ReturnDate | LID_return
Librarian
LID | name | surname | phone
Section
sectionid | Category
Student
SID | Name | Surname | Department | Entry_Year | Mail

Create Table Statements

Authors table:

```
CREATE TABLE authors                                    (
Author varchar(100) PRIMARY KEY NOT NULL,
Bdate INTEGER,
Bcountry varchar(100),
Gender VARCHAR(10)                                      )
```

Books table:

```
CREATE TABLE books                                      (
bookID INT PRIMARY KEY NOT NULL,
title varchar(160),
authors varchar(100),
isbn varchar(15),
average_rating DECIMAL(3,2),
num_pages INT,
publication_date INT,
publisher varchar(100),
SectionID varchar(2),
FOREIGN KEY(authors) REFERENCES authors(Author),
FOREIGN KEY(SectionID) REFERENCES section(sectionid)    )
```

Section table:

```
CREATE TABLE section                                    (
sectionid varchar(2) PRIMARY KEY NOT NULL,
Category varchar(50)                                    )
```

Borrow table:

```
CREATE TABLE borrow                                          (
bookID int PRIMARY KEY NOT NULL,
SID int PRIMARY KEY NOT NULL,
BorrowDate PRIMARY KEY datetime NOT NULL,
ReturnDateDeadline datetime,
LID_borrow int,
ReturnDate datetime,
LID_return int,
FOREIGN KEY (bookID) REFERENCES books(bookID),
FOREIGN KEY (SID) REFERENCES students(SID),
FOREIGN KEY (LID_borrow) REFERENCES librarian(LID),
FOREIGN KEY (LID_return) REFERENCES librarian(LID)           )
```

Librarian table:

```
CREATE TABLE librarian                                       (
LID int PRIMARY KEY NOT NULL,
name varchar(50),
surname varchar(50),
phone bigint                                                 )
```

Student table:

```
CREATE TABLE students                                        (
SID int PRIMARY KEY NOT NULL,
Name varchar(50),
Surname varchar(50),
Department varchar(100),
Entry_Year int,
mail varchar(50)                                             )
```

Populating The Database

Authors table (28291 rows):

Author names: Taken from the books table.

Other attributes were created with the python script using the logic below:

Birth date: To be somewhat realistic, we assumed that the authors published books between the ages of 18-70. The dates are obtained by subtracting a random value between 18-70 from the year the author published any of her/his book.

Birth Country: 100 country names randomly distributed.

Gender: Evenly and randomly distributed.

Books table (56718 rows):

All the data in the table, except foreign key, consists of the real-world data of Goodreads and is obtained from the data named "Goodreads Book Datasets with User Rating 10M" from Kaggle[1].

Borrows table (1000 rows):

All the data in the table, except foreign keys, were created with the python script using the logic below:

Half of the students did not borrow a book, 500 students borrowed books 1-4 times.

Borrow date: Randomly generated dates between 01.01.2021-15.05.2021.

Return date: Borrow date + randomly generated number between 1-15 (Also there are some unreturned books)

Librarian table (50 rows):

All the data in the table were created with the python script, randomly.

Section table (30 rows):

All the data in the table were created with the python script, randomly.

Student table (1000 rows):

All the data in the table were created with the python script, randomly.

To keep the data realistic, departments were matched with departments at Koç University and e-mail addresses were created as the “first letter of the name + surname + entry year @ ku.edu.tr”.

[1] *See* https://www.kaggle.com/bahramjannesarr/goodreads-book-datasets-10m.

Complex SQL Queries

All complex queries listed below were integrated in the prototype under the "advanced queries" tab. We chose to place them under a distinct part of our prototype because as opposed to working on a single table like most other (simple) queries in our system, these queries access a variety of tables within the database. Also, they are used as observational queries to analyze the large dataset at hand and they would not be required to run and/or maintain the system. They have a special use case and are used less frequently. The fields described below with letters and / symbols are customizable via the web interface of the system.

Name, number of books and average ratings (2 decimal places) of authors with at least N books.

- This query is useful for the administrator/data analyst to see if more books necessarily mean a better author, since it allows the user to see all authors, the number of books they have in the system, and the average "average-rating" of those books.

```
SELECT B.authors as Author, count(*) as NumberOfBooks, round(avg(B.average_rating), 2) as
AverageRating

        FROM books as B, authors as A

        WHERE B.authors=A.author

        GROUP BY A.author

        HAVING count(*) > $numBook

        ORDER BY $orderCol $order
```

The average age of the authors of the books with a rating higher/lower than R, by book categories.

- This is a useful query since it allows the user to analyze the age of authors within a certain category and rating range: thereby allowing claims like "A Romance book is more likely to be written by younger author compared to a Law book" to be made.

```
SELECT Category, AVG(publication_date - Bdate) as Average_Age

        FROM books B,authors A, section S

        WHERE B.authors=A.Author and S.sectionid=B.SectionID and average_rating ".$comparison."
$rating

        GROUP BY Category

        ORDER BY $orderCol $order
```

Authors with more than N books published before the age of M.

- This is a useful query to see which of the authors within the database managed to get published earlier and more frequently, which is a measure of success for the author. This could therefore be used to see which authors performed better/worse.

```
SELECT authors as Author, count(*) as NumberOfBooks

        FROM books B,authors A

        WHERE B.authors=A.Author AND (B.publication_date-Bdate)<$age

        AND A.Author in (   SELECT authors

                     FROM books B

                     GROUP BY authors

                     HAVING count(*)>$numBook)

        GROUP BY authors

        HAVING count(*)>$numBook

        ORDER BY $orderCol $order
```

Among the borrowed books, the birth countries of the authors of books whose author with gender G and rating higher/lower than R.

- This is a useful query to see which countries produced male/female authors with a certain rating floor or ceiling. This allows the user to analyze distribution of the authors' birth countries in relation with both their average rating and their gender.

```
SELECT BCountry, count(*) as numAuthors

        FROM authors A, books B, section S

        WHERE A.Author=B.authors AND B.SectionID=S.sectionID AND A.Gender='$gender'

                AND B.average_rating $comparison $rating AND bookID in (   SELECT bookID

                                                    FROM borrow bo)

        GROUP BY BCountry

        HAVING count(*)>0

        ORDER BY $orderCol $order
```

Most rented book category by month.

- Perhaps the one that would most frequently be used among the advanced queries, this query displays the category of books which was borrowed most often each month. The prototype has data for the year 2021, up to and including May, so the user is able to see which category was most popular for January-May, and how many books were borrowed each month as well.

```
SELECT Month as Month, category, max(C) as NumOfBorrow

        FROM (SELECT category, count(*) as C, MONTH(BorrowDate) as Month

             FROM borrow bo, Books B, section S

             WHERE bo.bookID=B.bookID and B.SectionID=S.sectionid

             GROUP BY MONTH(BorrowDate), Category) as e3

        GROUP BY Month

        ORDER BY $orderCol $order
```