Group members: Zhang Feifei, Goh Gek Chuah Kenneth, Michael Y Abay

Database Systems Design HW3

ABSTRACT

The project is based on a book database system pertaining to various needs of the user. The basic interface involves querying books according to genre, title, author, edition, ISBN. We support services for the issuing of books. We build a personal profile page which is used for handling the transactions between the administrator and the various customers. The system gives the status for books available at the time, when that book is not found in the library.

A) The Database application proposal.

We have been tasked by a small community public library in Tacoma, Washington to help them migrate from a paper copy book borrowing system to a database driven book catalog, user and book inventory system. Up to this point, the librarians have been using Microsoft Excel to keep track of their catalog and users.

We have built a back end and front end for this database driven book catalog user and book inventory system.

Database API

We built the project using NodeJS and MySQl. More specifically, the web application, database driven book catalog user and book inventory system is created on Azure Cloud and it interfaces with the backend Azure Database for MySQL.

1. View all customers
    a. Can search customers by data fields, etc
    b. Edit customer
    c. Delete customer
2. Add customer
3. View all books
    a. Can search book by data fields, etc
    b. View requested books
    c. View issued books
4. Add new book
5. View own profile
    a. Change password
6. Edit own profile
7. View statistics

a. Total number of books
b. Total number of members
c. Total books borrowed
d. Total books borrowed in last 30 days
e. Most borrowed book in last 30 days with the number of occurrences
f. Most requested book with number of requests

Implement at least 3 out of these 7 fundamental functions in front end:

1. Adding a patron (a user) **(Done)**
2. Adding a book **(Done)**
3. Searching for a patron **(Done)**
4. Searching for a book **(Done)**
5. Reserving a book **(Done)**
6. Checking out a book
7. Checking in a book

The attributes that the application will contain.

TABLE 'books'

```
CREATE TABLE `books` (
  `book_id` int(100) NOT NULL,
  `user_id` int(100) DEFAULT NULL,
  `genre` varchar(300) NOT NULL,
  `title` varchar(300) NOT NULL,
  `author` varchar(300) NOT NULL,
  `publisher` varchar(300) NOT NULL,
  `edition` int(100) NOT NULL,
  `isbn` varchar(100) NOT NULL,
  `pages` int(100) NOT NULL,
  `date_issued` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

TABLE 'books_request'

```
CREATE TABLE `books_request` (
  `request_id` int(10) NOT NULL,
  `user_id` int(10) NOT NULL,
  `genre` varchar(300) NOT NULL,
  `title` varchar(300) NOT NULL,
  `author` varchar(300) NOT NULL,
  `edition` int(10) NOT NULL,
  `isbn` varchar(100) NOT NULL,
  `date` date NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

TABLE 'issue_date'

```
CREATE TABLE `issue_date` (
  `issue_id` int(10) NOT NULL,
  `book_id` int(10) NOT NULL,
  `user_id` int(10) NOT NULL,
  `date` date NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

TABLE 'users'

```
CREATE TABLE `users` (
  `user_id` int(100) NOT NULL,
  `name` varchar(300) NOT NULL,
  `phone` varchar(11) NOT NULL,
  `email` varchar(300) NOT NULL,
  `is_admin` tinyint(1) NOT NULL,
  `password` varchar(300) NOT NULL,
```

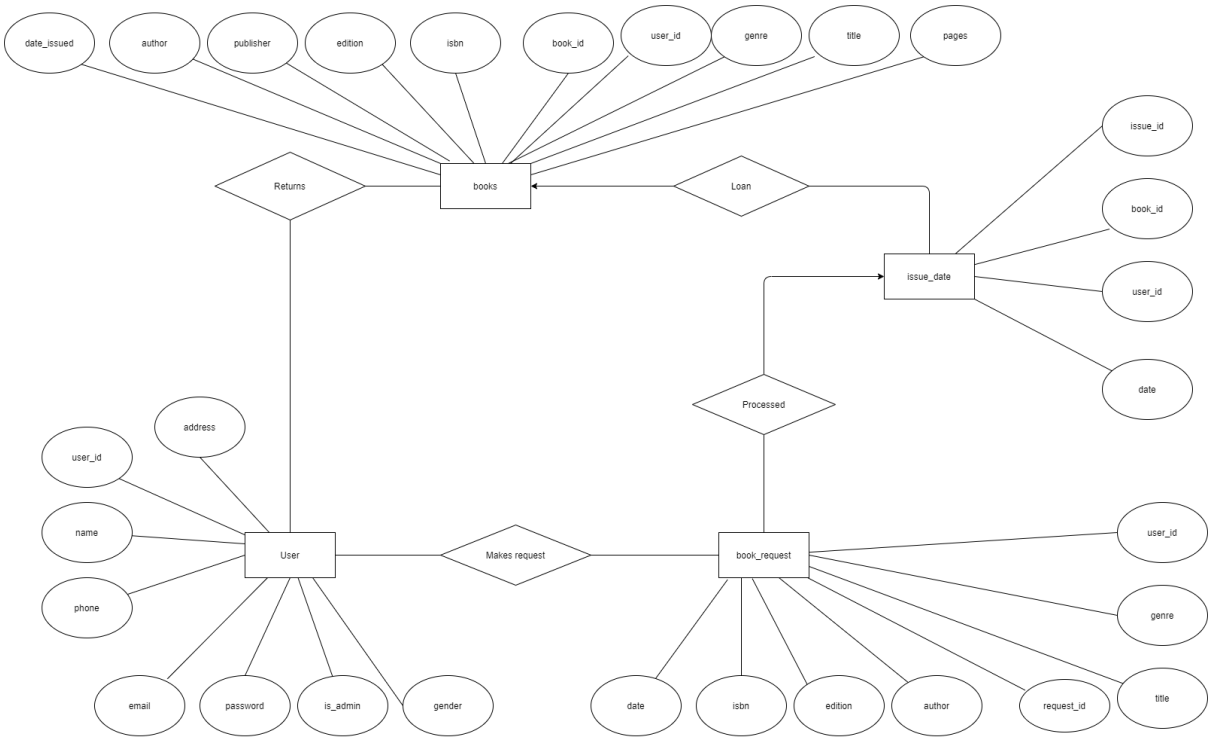
`address` varchar(300) NOT NULL,

`gender` varchar(300) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

The database backend is implemented on the Cloud (Microsoft Azure).

B) An ER Model of your proposed application.



C) The schema of the database that is normalized to 4NF. It should describe in detail each step of normalization.

Tables:

1) books(book_id, user_id, **genre**, title, **author**, **publisher**, **edition**, isbn, **pages, date_issued**)

key = {book_id, user_id}

2) books_request(request_id, user_id, **genre**, title, author, **edition**, isbn, **date**)

key = {request_id}

3) issue_date(issue_id, book_id, user_id, **date**)

key = {issue_id}

4) users(user_id, name, phone, email, is_admin, password, address, gender)

key = {user_id, email}

Non-Trivial Functional Dependencies:

1. books:

    a. book_id, user_id → (**genre**, title, **author**, **publisher**, **edition**, isbn, **pages, date_issued**)
2. books_request:
    a. request_id → (user_id, **genre**, title, author, **edition**, isbn, **date**)
3. issue_date:
    a. issue_id → (book_id, user_id, **date**)
4. users:
    a. user_id, email → (**name**, **phone**, **is_admin**, **password**, **address**, **gender**)

Multi-valued Dependencies:

1. books:
    a. date_issued ->-> title
    b. publisher ->-> author
    c. author ->-> genre
2. books_request:
    a. user_id ->-> request_id
3. issue_date:
    a. date ->-> user_id
4. users:
    a. No MVD's exists.

For the table, users:

Note that the **user_id, email** is the candidate key for users table.

The users table is a unique case in which the two tuples together constitutes the primary key. To ensure BCNF, we need to make sure that no other attribute from the users table appears along with these two tuples in some other table. The given observation holds in our database and hence users table is in BCNF form.

The users table is in 4NF since there does not exists any MVD's.

For the table, issue_date:

Note that the issue_id is the candidate key for issue_date table.

The corresponding set F+ corresponding to the above functional dependency can be divided into two sets (rule $\alpha \rightarrow \beta$):

1) α contains the candidate key, issue_id, then β would contain the whole schema attributes.

2) α does not contain key, issue_id, then $\beta = \alpha$

Now we will check the BCNF condition for each set.

For set-1, α is the superkey key and hence satisfies the condition.

For set-2, $\alpha \rightarrow \beta$ is a trivial functional dependency. Hence, we can see that if the non-trivial functional dependency contains the candidate key only, then the resulting schema would be in BCNF.

4NF Violators

    a. date ->-> user_id

Decomposition

1st Decomposition

Decomposition on date ->-> user_id

issue_date(issue_id, book_id, user_id, **date**)

R_1*(date,* user_id*)*

R_2*(date,* issue_id*,* book_id*)*

New Candidate Keys

R_1*: {date,* user_id*}*

R_2*: {date,* issue_id*,* book_id*}*

For the table, books_request:

Note that the request_id is the candidate key for books_request table.

The corresponding set F+ corresponding to the above functional dependency can be divided into two sets (rule α → β):

1) α contains the candidate key, request_id, then β would contain the whole schema attributes.

2) α does not contain key, request_id, then β = α

Now we will check the BCNF condition for each set.

For set-1, α is the superkey key and hence satisfies the condition.

For set-2, α → β is a trivial functional dependency. Hence, we can see that if the non-trivial functional dependency contains the candidate key only, then the resulting schema would be in BCNF.

4NF Violators

a. user_id ->-> request_id

Decomposition

1st Decomposition

Decomposition on user_id ->-> request_id

books_request(request_id, user_id, **genre**, title, author, **edition**, isbn, **date**)

R_1*(*user_id, request_id*)*

R_2*(*user_id*,* **genre**, title, author, **edition**, isbn, **date***)*

New Candidate Keys

R_1*: {*user_id, request_id*}*

R_2*: {*user_id*,* **genre**, title, author, **edition**, isbn, **date***}*

For the table, books:

Note that the **book_id, user_id** is the candidate key for **books** table.

The bookstable is a unique case in which the two tuples together constitutes the primary key. To ensure BCNF, we need to make sure that no other attribute from the users table appears along with these two tuples in some other table. The given observation holds in our database and hence users table is in BCNF form.

4NF Violators

a. date_issued ->-> title
b. publisher ->-> author
c. author ->-> genre

Decomposition

1st Decomposition

Decomposition on date_issued ->-> title

books(book_id, user_id, **genre**, title, **author**, **publisher**, **edition**, isbn, **pages, date_issued**)

R_1*(*date_issued*,* title*)*

R_2*(*date_issued*,* book_id, user_id, **genre**, **author**, **publisher**, **edition**, isbn, **pages***)*

New Candidate Keys

R_1*: {*date_issued*,* title*}*

R_2*: {*date_issued*,* book_id, user_id, **genre**, **author**, **publisher**, **edition**, isbn, **pages***}*

Remaining 4NF Violators

R_1*: None*

R_2*:* publisher ->-> author, author ->-> genre

2nd Decomposition

Decomposition on publisher ->-> author

*We have that {*publisher*, author} & {*publisher*,* date_issued*,* book_id, user_id, **genre**, **edition**, isbn, **pages***} and that:*

R_1 *(*date_issued*,* title*)*

$R_{2.1}$ *(*publisher*, author)*

$R_{2.2}$ *(*publisher*,* date_issued*,* book_id, user_id, **genre**, **edition**, isbn, **pages***)*

New Candidate Keys

$R_{2.1}$: {publisher*, author}*

$R_{2.2}$: {publisher*,* date_issued*,* book_id, user_id, **genre**, **edition**, isbn, **pages***}*

Remaining 4NF Violators

$R_{2.1}$*: None*

$R_{2.2}$*: None*

Test Cases:

1. View all customers
    a. Can search customers by data fields, etc
    b. Edit customer
    c. Delete customer
2. Add customer
3. View all books
    a. Can search book by data fields, etc
    b. View requested books
    c. View issued books
4. Add new book
5. View own profile
    a. Change password
6. Edit own profile
7. View statistics
    a. Total number of books
    b. Total number of members
    c. Total books borrowed
    d. Total books borrowed in last 30 days
    e. Most borrowed book in last 30 days with the number of occurrences
    f. Most requested book with number of requests

Individual Contribution

- **Zhang Feifei: 43%**
- **Goh Gek Chuah Kenneth: 39%**
- **Michael Y Abay: 18%**
- **Zhang Feifei: Improved upon first draft of the project in Nodejs framework and added several new functionalities with SQL script**
- **Goh Gek Chuah Kenneth: Finalized powerpoint slides, SQL script, wrote test cases, readme for installation/running of program and final pdf**
- **Michael Y Abay: SQL script draft, Powerpoint slides draft**