Inter-Institutional Library System

Deliverable 3: Application

COMP421 - Group 80

Boqi (Percy) Chen, Tianzhu Fu, ZiQi Li, Dalin Saheb

Notice: The database must be connectable before start the application (e.g. VPN enabled)
Our database password: COMP421G80IIL. (The database is reset to the status before any modification query runs, so everything below is reproducible.)

1. Stored Procedure

1.1 Description

This stored procedure takes three parameters iname, lname and default length of the loan. It searches all the requests to find the requests associated with book copy in the library lname belongs to institution iname. Then it checks whether the request is approved, if it is, it inserts a new record to Loans with current date as start date, current date + default length as required return date, and delete the original request. Last, it returns how many loans have been created.

1.2 SQL

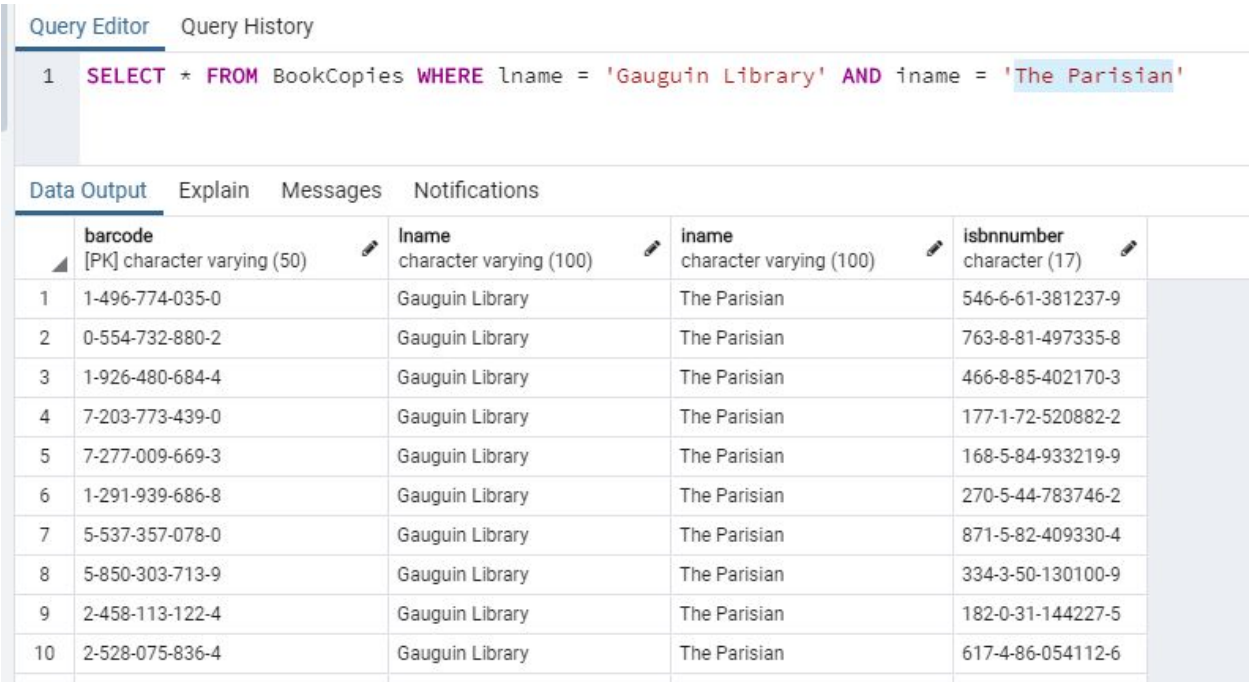
```
CREATE OR REPLACE FUNCTION createLoansFromApprovedRequest(selectedIname VARCHAR(100), selectedLname
VARCHAR(100), loanLength INTEGER) RETURNS INTEGER AS $$
  DECLARE rec RECORD;
      createdLoanCount INTEGER;
      currentDate Date;
      patronBookPairs CURSOR FOR
          (WITH BookRequests AS
          (SELECT R.email, R.barcode, R.status FROM requests R, Bookcopies B
          WHERE R.barcode = B.barcode AND B.lname = selectedLname AND B.iname = selectedIname)
          SELECT br.email, br.barcode
          FROM BookRequests br
          WHERE br.status = 'approved');
  BEGIN
    createdLoanCount := 0;
    currentDate := (SELECT CURRENT_DATE);

    FOR rec IN patronBookPairs LOOP
      createdLoanCount := createdLoanCount + 1;
      INSERT INTO Loans(startdate, requiredreturndate, actualreturndate, fine, hasrequest, email, barcode) VALUES
      (currentDate,currentDate + loanLength,NULL, 0,true,rec.email,rec.barcode);
```

```
        DELETE FROM Requests r WHERE r.email = rec.email AND r.barcode = rec.barcode;
    END LOOP;
    RETURN createdLoanCount;
  END;
$$ LANGUAGE plpgsql;
```

1.3 Execution Demo

1. To show the demo of the stored procedure, we will need to take use of elements inside the database. Let's use the library Gauguin Library in the institution The Parisian. Let's first see what book copies this library has:



Query Editor | Query History

```
SELECT * FROM BookCopies WHERE lname = 'Gauguin Library' AND iname = 'The Parisian'
```

Data Output | Explain | Messages | Notifications

	barcode [PK] character varying (50)	lname character varying (100)	iname character varying (100)	isbnnumber character (17)
1	1-496-774-035-0	Gauguin Library	The Parisian	546-6-61-381237-9
2	0-554-732-880-2	Gauguin Library	The Parisian	763-8-81-497335-8
3	1-926-480-684-4	Gauguin Library	The Parisian	466-8-85-402170-3
4	7-203-773-439-0	Gauguin Library	The Parisian	177-1-72-520882-2
5	7-277-009-669-3	Gauguin Library	The Parisian	168-5-84-933219-9
6	1-291-939-686-8	Gauguin Library	The Parisian	270-5-44-783746-2
7	5-537-357-078-0	Gauguin Library	The Parisian	871-5-82-409330-4
8	5-850-303-713-9	Gauguin Library	The Parisian	334-3-50-130100-9
9	2-458-113-122-4	Gauguin Library	The Parisian	182-0-31-144227-5
10	2-528-075-836-4	Gauguin Library	The Parisian	617-4-86-054112-6

2. Next we make sure there is no requests related to this library

Query Editor | Query History

```
DELETE FROM Requests WHERE barcode IN (SELECT barcode FROM BookCopies WHERE lname = 'Gauguin Library' AND iname = 'The Parisian');
SELECT * FROM Requests WHERE barcode IN (SELECT barcode FROM BookCopies WHERE lname = 'Gauguin Library' AND iname = 'The Parisian');
```

Data Output | Explain | Messages | Notifications

email [PK] character varying (100)	barcode [PK] character varying (50)	date date	status character varying (20)

3. We manually add some requests associated with the book copies belonging to this library. For convenience, we just initialized them as 'approved' and insert three of them

```
INSERT INTO Requests VALUES ('jerold.leffler77@mail.com', '1-496-774-035-0', (SELECT * FROM CURRENT_DATE), 'approved');
INSERT INTO Requests VALUES ('shayna.treutel65@mail.com', '0-554-732-880-2', (SELECT * FROM CURRENT_DATE), 'approved');
INSERT INTO Requests VALUES ('gretchen.mckenzie39@gmail.com', '1-926-480-684-4', (SELECT * FROM CURRENT_DATE), 'approved');
SELECT * FROM Requests WHERE barcode IN (SELECT barcode FROM BookCopies WHERE lname = 'Gauguin Library' AND iname = 'The Parisian');
```

	email [PK] character varying (100)	barcode [PK] character varying (50)	date date	status character varying (20)
1	jerold.leffler77@mail.com	1-496-774-035-0	2020-04-11	approved
2	shayna.treutel65@mail.com	0-554-732-880-2	2020-04-11	approved
3	gretchen.mckenzie39@gmail.com	1-926-480-684-4	2020-04-11	approved

4. We check that there is no loans associated with the patrons and the book copies we just created above:

```
SELECT * FROM Loans WHERE (email, barcode) IN (('jerold.leffler77@mail.com', '1-496-774-035-0'), ('shayna.treutel65@mail.com', '0-554-732-880-2'), ('gretchen.mckenzie39@gmail.com', '1-926-480-684-4'));
```

loanid [PK] integer	startdate date	requiredreturndate date	actualreturndate date	fine double precision	hasrequest boolean	email character varying (100)	barcode character varying (50)

5. Then, we run the stored procedure, with 100 days as the default length.

```
SELECT createLoansFromApprovedRequest('The Parisian', 'Gauguin Library', 100);
```

createloansfromapprovedrequest integer
3

The 3 indicates that three of such requests have been found and three loans have been selected.

6. Check the impact of the stored procedure on the database:

Check that the original requests has been removed:

```
SELECT * FROM Requests WHERE barcode IN (SELECT barcode FROM BookCopies WHERE lname = 'Gauguin Library' AND iname = 'The Parisian');
```

email [PK] character varying (100)	barcode [PK] character varying (50)	date date	status character varying (20)

We see that all the approved requests of the librarys is removed. Let's check the impact on the Loans relation

```
SELECT * FROM Loans WHERE (email, barcode) IN (('jerold.leffler77@mail.com', '1-496-774-035-0'), ('shayna.treutel65@mail.com', '0-554-732-880-2'), ('gretchen.mckenzie39@gmail.com', '1-926-480-684-4'));
```

	loanid [PK] integer	startdate date	requiredreturndate date	actualreturndate date	fine double precision	hasrequest boolean	email character varying (100)	barcode character varying (50)
1	10	2020-04-11	2020-07-20	[null]	0	true	jerold.leffler77@mail.com	1-496-774-035-0
2	11	2020-04-11	2020-07-20	[null]	0	true	shayna.treutel65@mail.com	0-554-732-880-2
3	12	2020-04-11	2020-07-20	[null]	0	true	gretchen.mckenzie39@gmail.com	1-926-480-684-4

Indeed three new loans have been created for the patrons and the book copies, the start date is today, and the required return date is in 100 days, all of which are indicated to have a request.

P.s. the complete script can be found in the folder *p3/storedProcedure.sql*

2. Application

2.1 Option Description

2.1.1 Find Book Location

Input:

a book name
Ex: 'Tiger! Tiger!'

Output:

Find the name of the institutions and the libraries that belong to them which have the requested book. Show all the libraries and their institutions that satisfy this query.

2.1.2 Make Request for a Book

Input:

email, book, isbnNumber
Ex: 'ronny.mayer22@mail.com', '005-9-72-540567-1'

Output:

Ask the patron to enter his/her email and book isbnNumber, if any libraries that do not belong to the patron's institution have such a book, make a request on one of the book copies of this Book. Otherwise, prompt "NO available bookcopy" message to the user. If the barcode list is not empty, make a request on one of the bookCopies in the list for the patron using the current date.

2.1.3 Add New Patron

Input:

email, uName, phoneNumber, uAddress, iname
Ex: ‘boqi.chen@mail.mcgill.ca’, ‘Percy Chen’, ‘421487632’, ‘3213 SOME STREET’, ‘McGill University’

Output:

It first finds the institution with iname and then adds a user to the Database. After that, it adds a patron which has foreign key references to the institution iname and the user just created.When all of them succeed, it returns a string indicating that the insertion is successful. In the case that the institution with iname is not found in the database, it returns an error. When any SQL related error happens, it returns that error as well.

2.1.4 Get All Loans For a Patron

Input :

email

Ex: ‘ronny.mayer22@mail.com’

Output:

It gets all loans for a patron using the email and returns it as a table.

2.1.5 Delete Declined Requests

Output:

This command deletes every request with status ‘approved’ from the database.

2.1.6 Update A Request

Input:

(pEmail, pBarCode, boolean approved)

Ex.: ronny.mayer22@mail.com, 421487632, true

Output:

This command updates the status of a request. Setting the boolean to true approves the request and setting it to false declines the request.

2.1.7 Custom Query

Input:

Any customized SQL query (SELECT ...)

Output:

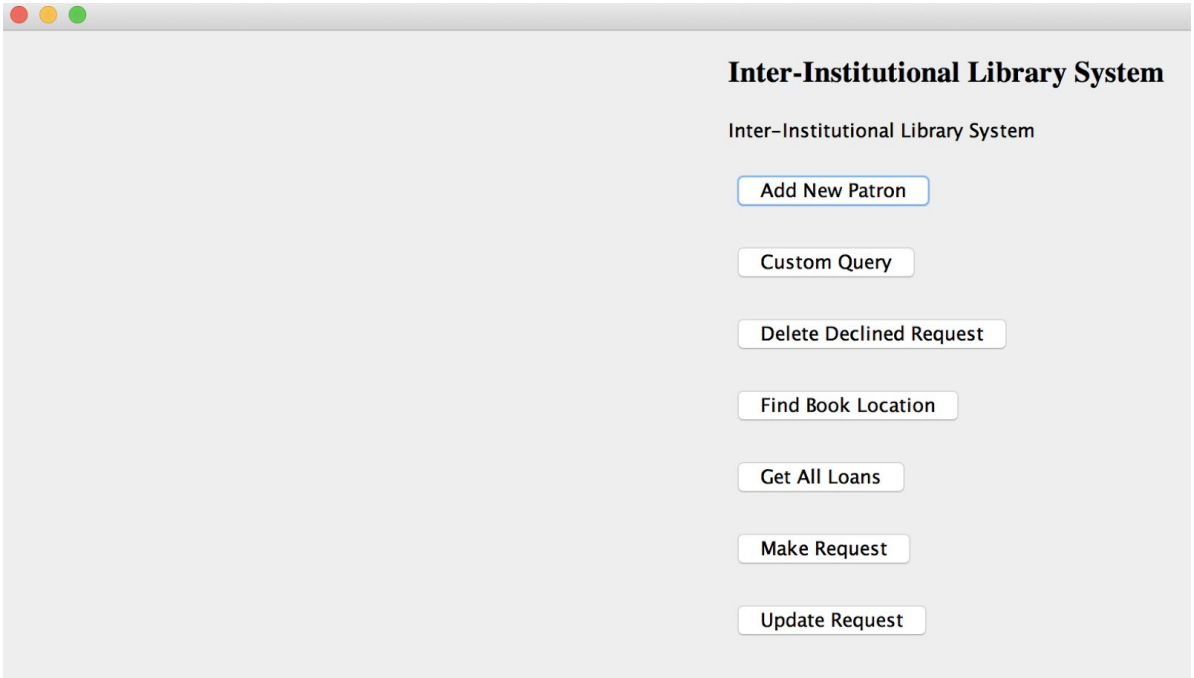
This option allows clients to write any customized SQL query and our application will return the result of the input query to clients. If the input query doesn’t respect the SQL syntax or has some typo on the relation names, our application will return an error message to clients.

2.1.8 Quit

The application can quit by clicking the “x” button on the up left (right on Windows) corner.

2.2 Main Menu

The main menu contains a database system title and 7 buttons. Each button is corresponding to one of the above options. Clients can go to any option panel by clicking these option buttons and close this application by clicking the “X” at the top of the window frame.



2.3 Demo Video Link

The Demo video to execute each option is here:
https://mcgill-my.sharepoint.com/:v:/g/personal/boqi_chen_mail_mcgill_ca/EXBKUL6Db3lEnjGuzC3Pz4oBwsj09DWdmW_O7fnQm1mUGQ?e=8VJ63b

P.s. the application is a Maven Project, you need to install Maven in the system in order to build the project. The source code can be found in p3/IIL-Application/. Run the main mail method in class ill.gui.GUI to execute the project.

3. Indexes

First index:

requeststatus

Purpose:

Creating an index on the ‘requests’ relation on the attribute ‘status’ allows us to retrieve tuples faster when we need to execute queries based on the status of a request. For example, we need this when we delete every ‘approved’ request.

SQL:

```
-- CREATE statement
CREATE INDEX requeststatus ON requests(status);
-- DROP statement
DROP INDEX requeststatus;
```

Second index:

bookname

Purpose:

This index allows us to search for a book faster by creating an index on the book name. Some features of the application need a book name as an input to proceed. For example, a user may need to find the location of a book by giving its name as input.

SQL:

```
-- CREATE statement
CREATE INDEX bookname ON books(title);
-- DROP statement
DROP INDEX bookname;
```

P.s. the source script can be found in *p3/index.sql*

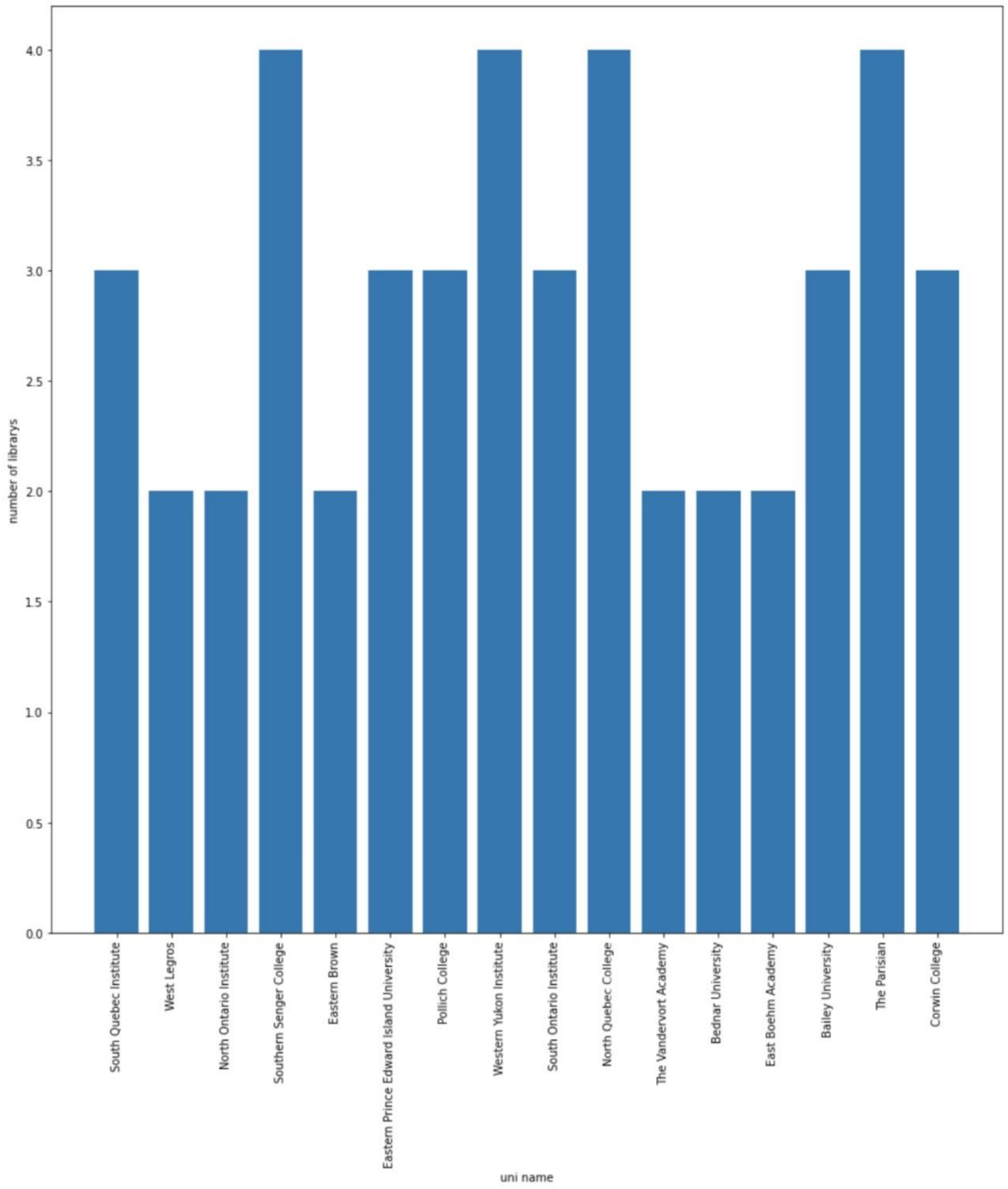
4. Data Visualization

4.1 Visualization Description

1. This query returns the institution name and the number of its libraries if thIS institution has more than one library.

```
SELECT iname, COUNT(*)AS libraryCount FROM libraries
GROUP BY iname
```

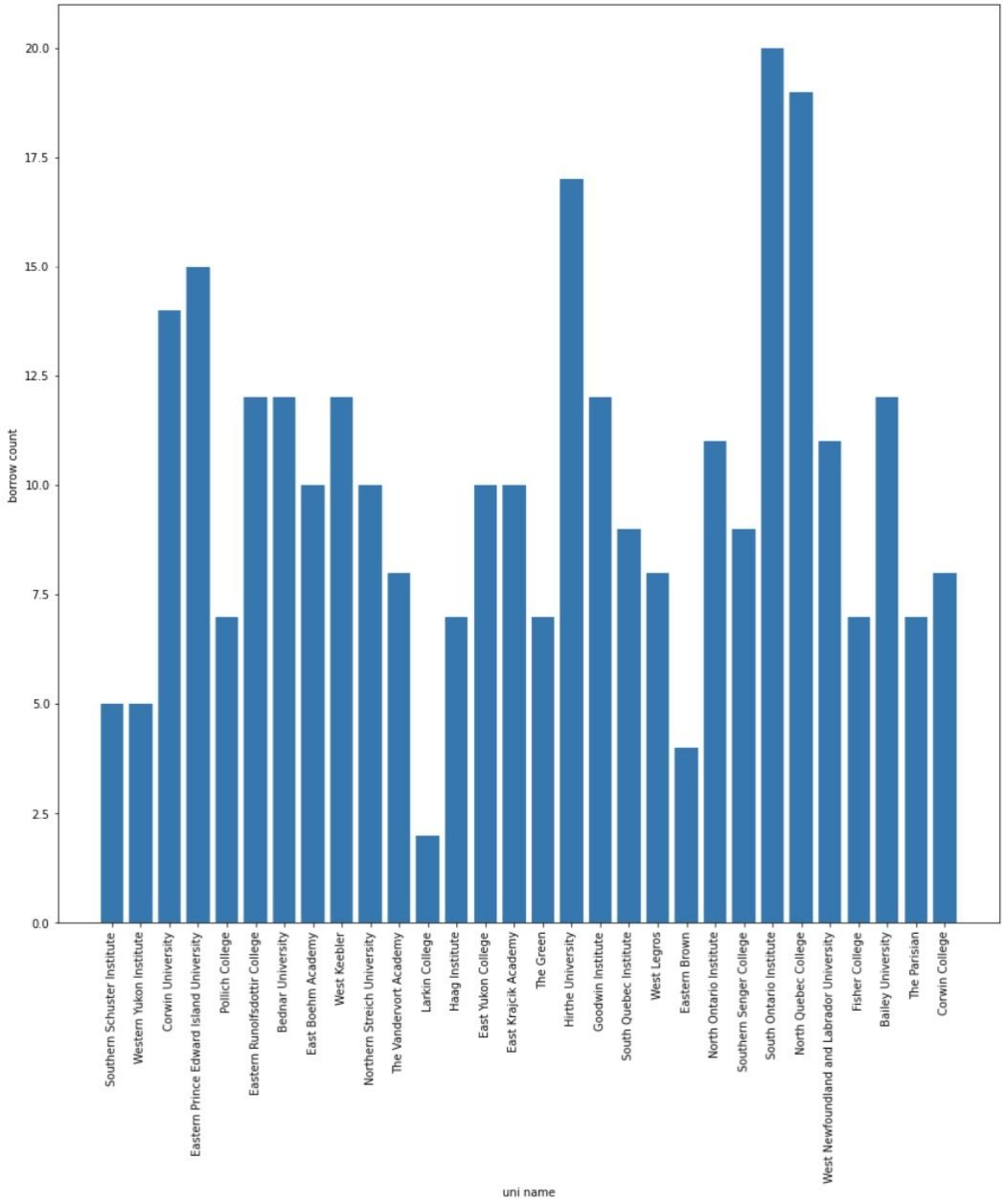
```
HAVING COUNT(*)>1
```



2. This query returns the institution name and the total number of borrowing from its student.

```
WITH borrowcount AS (SELECT COUNT(*)as borrowrecord, email FROM loans
GROUP BY email),
schoolborrowhistory AS (SELECT iname, COALESCE(borrowrecord, 0) AS borrowhistory FROM
borrowcount bc FULL OUTER JOIN patrons p
ON bc.email = p.email)
```

```
SELECT * FROM schoolborrowhistory
SELECT SUM(borrowhistory), iname FROM schoolborrowhistory sbh
GROUP BY iname;
```



4.2 Source Code

The source code for generating these graphs can be found in *p3/p3_datavisual/project3.ipynb*

5. Creativity

For the creativity part, we created the application with GUI, along with an option to let the user perform customer queries to the database. The Demo Video link can be found in section 2.3.

6. Appendix

6.1 Relational Schema

Entities: (for the relationships with both key and participation constraints, we include the relationship set in table of the entity set with the key constraint)

-Users(email, uName, phoneNumber, uAddress)
-Patrons(email, maxBooks, iName)
(iName ref Institutions, email ref Users)
*iName NOT NULL
-Librarians(email, workingDays, lName, iName)
(lName ref Libraries,iName ref Institutions)
*lName, iNameNOT NULL
-Administrators(email, iName)
(iName ref Institutions, email ref Users)
*iName NOT NULL
-Institutions(iName, iAddress)
-Books(isbnNumber, title, category, publicationDate, publisher, description)
-Authors(authorId, aName)
-BookCopies(barCode, lName, iName, isbnNumber)
(lName ref Libraries, iNameref Institutions, isbnNumber ref Books)
*lName, iName, isbnNumber NOT NULL
-Loans(loanId, startDate, requiredReturnDate, actualReturnDate, fine, hasRequest, email, barCode)
(email ref Patrons, barCode ref BookCopies)
*email, barCode NOT NULL

Weak Entity:

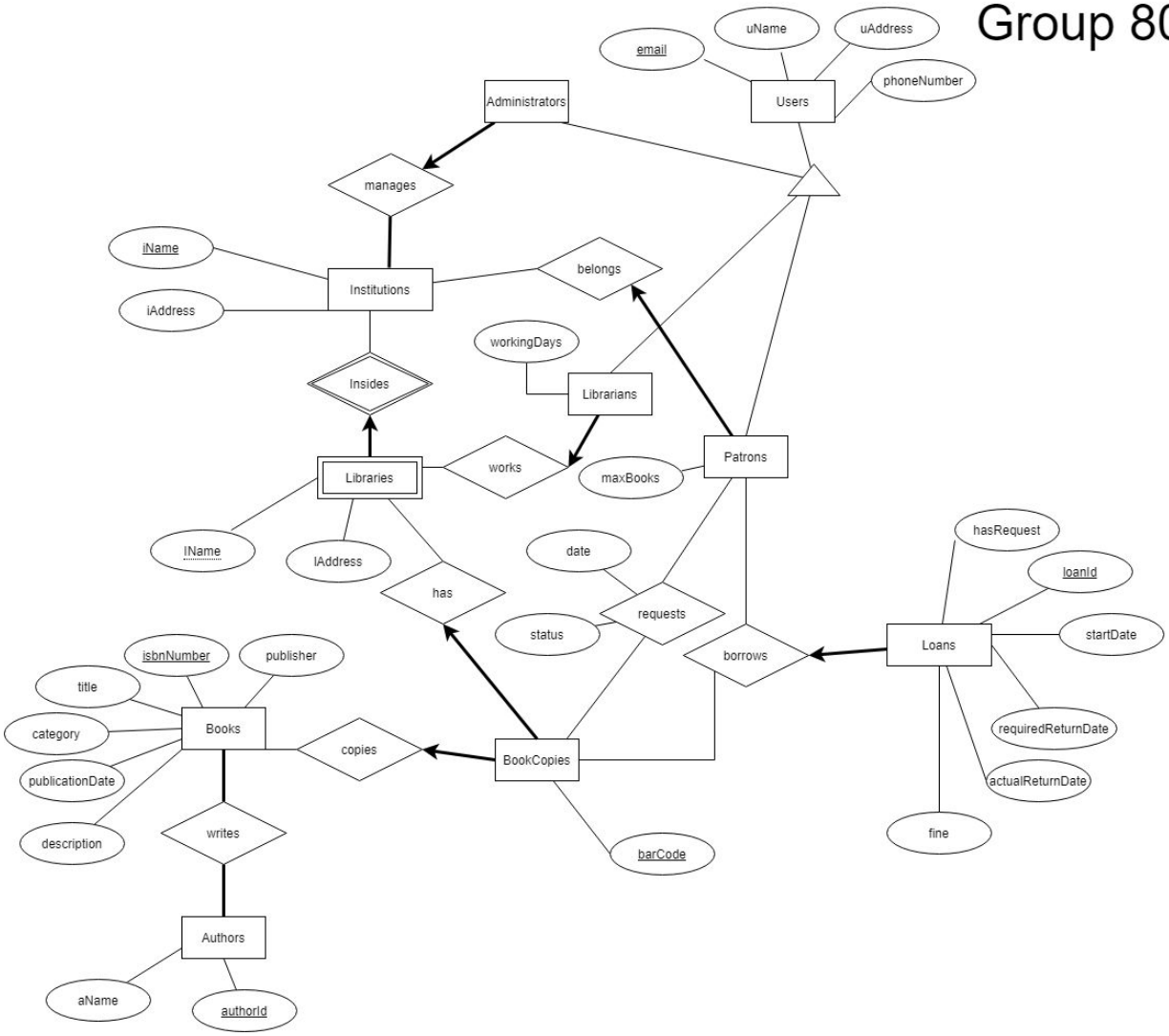
-Libraries(lName, iName, lAddress) (iName ref Institutions)
Tilder at iName

Relationships:

-writes(authorId, isbnNumber)
(authorId ref Authors, isbnNumber ref Books)
-requests(email, barCode, date, status)
(email ref Patrons, barCode ref BookCopies)

Notes: all the relationships with key constraints are contained in the entities table.

Group 80


email
uName
uAddress
phoneNumber
Users
Administrators
manages
iName
Institutions
belongs
iAddress
workingDays
Librarians
Insides
Libraries
works
maxBooks
Patrons
IName
IAddress
has
date
requests
status
hasRequest
loanId
startDate
Loans
borrows
isbnNumber
publisher
title
category
Books
copies
BookCopies
publicationDate
description
writes
barCode
requiredReturnDate
actualReturnDate
fine
Authors
aName
authorId