

BadgerBibEngine: A Bibliometric Database Web Application

Willie Klein, Peter Bryant, Xiang Zheng

CS 564, Professor Hien Nguyen

[Presentation Slides](#), [Demo Video](#)

Summer 2022

Abstract

This report details the application that we designed and built in CS564: Database Management Systems: Design and Implementation, at the University of Wisconsin-Madison, in the Summer of 2022. The project showcases each step required to design and implement a database, and further, presents an intuitive interface for the user to interact with the database. Additionally, we will provide a high-level description of the backend implementation of the user application, which will include discussion into what design choices were made, what weaknesses our design has, and what we would have changed if we had more time. In this write-up we will first discuss how we picked an application that benefited from a database and interface. Next, we will explain our conceptual design of the database, deciding on what sets our entity-relational model would consist of. Further, we translated this design into a relational schema, and normalized the data so that our relations were dependency preserving and not lossless. After that will describe our choice of database management system, backend web framework and imported libraries. And finally we will describe our backend implementation, including our user interface, how we access user information and pass it to the backend, and how we connect to the database and run stored queries needed to draw insightful information from our data. This report describes each step of the process.

Step 1: Finding an Application

Bibliometric Data

Bibliometrics is a research area that determines the “impact” that scholarly articles and journals have, usually measured through the number of citations. Datalab at UC Davis describes it as “Bibliometrics typically measure research outputs (...) often in terms of publication counts, citation counts, and measurements derived from these” (Datalab at UCDavis, 2020). Our database holds bibliometric data for a large set of scholarly articles, administered by the Web of Science database by Clarivate Analytics (Web of Science, 2022). The Web of Science is one of the world’s largest and most popular publisher-independent global publication and citation databases, covering a wide range of journals in different disciplines. This database is widely applied by research institutes, universities, governments, etc. to support scientific research, research evaluation, and informed science policy decision making. Using high-quality data from the Web of Science, we developed this database to help users search bibliographic information and visualize bibliometric data.

Picking an Application

In scientometrics and Science of Science research, computing various types of bibliometric measures about articles, authors, and journals is a fundamental and complex work for almost every research project concerning massive bibliographic data. For example, a recent work of our group member, Xiang, has compared the field-normalized annual publication numbers of men and women scientists to quantify the widely-known gender disparity in the scientific community. Our group quickly decided that an application that helped return searched results regarding articles, their author’s, the journal’s they are published in and the journal’s

impact factor could be a beneficial application that accelerates the complex computation for scholarly research. Moreover, we realized that not only searching, but being able to make an account and insert and delete articles from the database would make the final application much more dynamic. We also thought that it would provide much higher upside to us as application developers, because our database could be updated and more relevant at the cost of just a few simple SQL queries. Additionally, at the onset we saw that these tables were already normalized, so we figured we would be able to set up our database quickly and otherwise prioritize design and software development. The core purpose of the application is to allow the user to effortlessly search and edit our bibliometric database; we also added some relevant data plots to help visualize our dataset on the home page, but this was mostly a design choice to increase user experience with more visual appeal.

We later organized the data into different tables, the number records for each table ranges from 3,000 - 24,000 records, the exact numbers as so:

```
SELECT COUNT(*) FROM author; -- 15505
SELECT COUNT(*) FROM article; -- 9593
SELECT COUNT(*) FROM authored_by; -- 24913
SELECT COUNT(*) FROM journal_impact_factor; -- 14847
SELECT COUNT(*) FROM journal; -- 3596
```

Figure 1: Number of Records per Table

Here is a sample of the CSV data for each entity:

Journal

Code_Venue	abbrev	journal_name	discipline	specialty
23	ACAD PSYCHI	ACADEMIC PSYCHIATRY	PROFESSIONAL FIELDS	EDUCATION
37	ACCOUNT HOR	ACCOUNTING HORIZONS	PROFESSIONAL FIELDS	MANAGEMENT & BUSINESS
62	ACM T MATH	ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE	MATHEMATICS	APPLIED MATHEMATICS
374	ADM POL M H	ADMINISTRATION AND POLICY IN MENTAL HEALTH	PSYCHOLOGY	MISC PSYCHOLOGY
412	ADV ATMOS S	ADVANCES IN ATMOSPHERIC SCIENCES	EARTH & SPACE	METEOROL & ATMOS SCI
433	ADV COLL IN	ADVANCES IN COLLOID AND INTERFACE SCIENCE	CHEMISTRY	PHYSICAL CHEMISTRY
454	ADV EXP MED	ADVANCES IN EXPERIMENTAL MEDICINE AND BIOLOGY	BIOMEDICAL RESEARCH	BIOCHEM & MOLEC BIOL
476	ADV MATH CO	ADVANCES IN MATHEMATICS OF COMMUNICATIONS	MATHEMATICS	APPLIED MATHEMATICS

Figure 2: Shortened Journal Table

Journal Impact Factor

Code_Venue	jcrYear	issn	totalCites	totalArticles	JIF	quartile
667	2020	0364-0094	161	17		
805	1997	0195-6744	194	7	0.8	Q1
805	1998	0195-6744	217	9	0.833	Q1
805	1999	0195-6744	198	4	0.4	Q3
805	2000	0195-6744	208		0.9	Q1
805	2001	0195-6744	189		0.333	Q3

Figure 3: Shortened JIF Table

Authored By

UID	order	author_id
47253620	1	6568041
47256062	1	49150724
47256062	2	44811639
47256062	3	37240941

Figure 4: Shortened Authored By Table

Author

author_id	Full_Name	First_Name	Last_Name	Suffix
1342667	Christian, Bradley T.	Bradley T.	Christian	
13568871	Betthausen, Tobey J.	Tobey J.	Betthausen	
7076883	Zammit, Matthew D.	Matthew D.	Zammit	
12437291	Cody, Karly A.	Karly A.	Cody	
29528801	Barnhart, Todd E.	Todd E.	Barnhart	
45718442	Converse, Alexander K.	Alexander K.	Converse	

Figure 5: Shortened Author Table

Article

UID	PubYear	Code_Venue	Title	Volume	Issue	Nb_Page	b_Referen	Nb_Author	b_addres	DOI
50083663	2018	20328	Association of Cadm	136	12	9	57	9	2	10.1001/jamaophthalmol.2018.3931
50871632	2018	20329	Comparison Between	144	11	9	42	8	7	10.1001/jamaoto.2018.0309
50442260	2018	20330	Association Between	172	6	9	48	10	7	10.1001/jamapediatrics.2018.0322
70479972	2017	6003	Urban heat island-in	44	2	9	64	4	4	10.1002/2016GL072190
67909872	2018	392	Highly Porous Polym	28	13	9	42	7	8	10.1002/adfm.201706365

Figure 6: Shortened Article Table

Steps 2 and 3: Conceptual Design and Relational Schema

Entity-Relationship Model and Diagram

Before creating any tables, it was important that we decided how to organize all the data. We organized the attributes into relevant entity sets as Article, Author, Journal, and Journal Impact Factor. Article contains the information about the articles like it's: UID (the article's publication identifier assigned by the Web of Science), title, DOI (Digital Object Identifier), Code Venue (the journal's unique identifier assigned by the Web of Science), publication year, volume, issue, number of pages, number of authors, number of references, and number of addresses. Article's UID connects it to the Author relation by way of the Authored-By relation that contains the UID of an article, the order of the authors of the article, and the Author ID of an Author of the article. Not only does the Author relation contain the Author ID attribute, it also contains the Author's first and last names and Suffix.

Similarly, the Article relations code venue attribute creates a relationship with the Journal relation that also contains code venue. Journal also contains fields corresponding to the Journal's abbreviation, name, discipline and specialty. Code venue again connects Journal to Journal Impact Factor which is another relation that contains code venue, the jcr year, the ISSN, the total number of citations for the journal, the total articles for the journal, the journal impact factor, and the quartile.

In total, there are five tables from which our application will draw from, 4 entity tables and 1 relationship table. We only set up a table for the relationship between Article and Author, because that is a many to many relationship. We did not set up a table for the "contains" relationship between Article and Journal because an article can only be published by one journal due to academic ethics, making it a many-to-one relationship. We also did not set up a table for

the “has-a” relationship between Journal and Journal Impact Factor because the latter is a weak entity set. It just needs the Code Venue and the JCR year in order to identify the JIF and Quartile attributes.

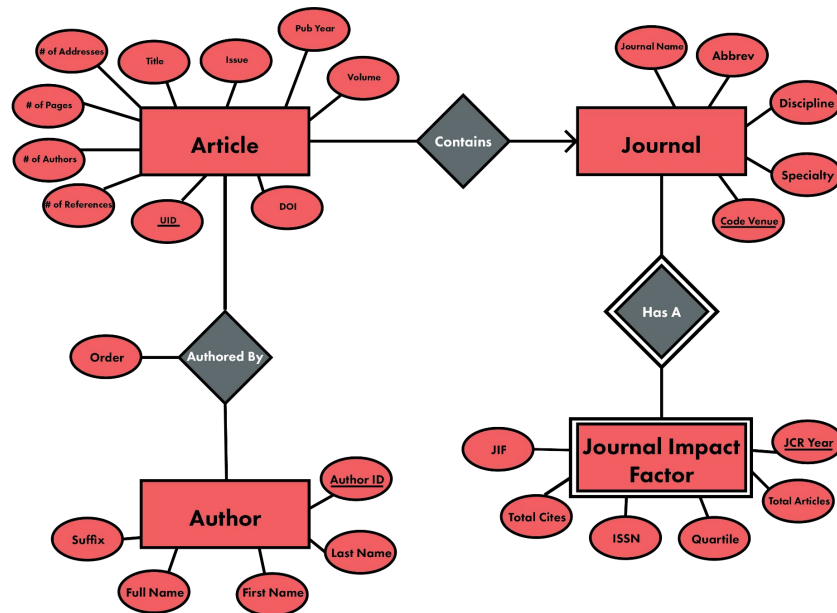


Figure 7: ER Diagram

Relational Schema

Our entity sets were easily translated into a relational schema from our ER Diagram. We did have to decide on the domain for each attribute per relation, as well as determine all the functional dependencies, which in our case was trivial and will be discussed in Step 4 about the normalization process.

Relationships in the Relational Schema

After translating to the relational schema, we kept one relationship table called Authored By that links Article and Author, however instead of making a table for the relation between Journal and Article, and Journal and Journal Impact Factor, we found it sufficient to share a key (code venue) between those three tables. Ultimately, this reduced the complexity of the queries

we had to write, as the process of joining tables was just a matter of joining some combination of UID, Author ID, and code venue for all relations.

Step 4: Normalization Process

Boyce Codd Normal Form (BCNF)

The way that we organized our data and chose our primary keys, made this a fairly simple process. Our tables were already in BCNF and did not require normalization at this step in the process.

Proposition 1 *BadgerBibEngine's DBMS relational schema, relationship set R (PK-Primary Key):*

$$R = \{ \\ \text{Article}(\text{UID (PK), Title, DOI, Issue, Volume, Pub year, \# of Addresses, \# of Pages, (1)} \\ \text{\# of Authors, \# of References, Code Venue})$$

$$\text{Author}(\text{Author ID (PK), Full Name, First Name, Last Name, Suffix}) \quad (2)$$

$$\text{Journal}(\text{Code Venue (PK), Journal Name, Abbreviation, Discipline, Specialty}) \quad (3)$$

$$\text{Journal Impact Factor}(\text{Code Venue (PK), JCR Year (PK), ISSN, JIF, Total Articles, (4)} \\ \text{Total Cities, Quartile})$$

$$\text{AuthoredBy}(\text{UID, Order, Author ID}) \quad (5)$$

Proposition 2 *Non-trivial Funtional Dependencies (FDs):*

$$\text{FDs} = \{ \\ \text{UID} \rightarrow \text{Title, DOI, Issue, Volume, Pub year, \# of Addresses, \# of Pages, (6)} \\ \text{\# of Authors, \# of References, Code Venue}$$

$$\text{Author ID} \rightarrow \text{Full Name, First Name, Last Name, Suffix} \quad (7)$$

$$\text{Code Venue} \rightarrow \text{Journal Name, Abbreviation, Discipline, Specialty} \quad (8)$$

$$\text{Code Venue, JCR Year} \rightarrow \text{ISSN, JIF, Total Articles, Total Cities, Quartile} \quad (9)$$

$$\text{UID, Order} \rightarrow \text{Author ID} \quad (10)$$

Theorem 1 *A relation R is in BCNF if and only if whenever there is a nontrivial FD:*

$$A_1, A_2, A_3, \dots, A_n \rightarrow S B$$

for R, it is the case that,

$$A_1, A_2, A_3, \dots, A_n$$

is a superkey for R.

Proposition 3 *R is already BCNF in our application.*

Showing R is already BCNF is very easy in our case, because no BCNF decomposition is required.

Proof First we list all candidate keys for R,

$$\begin{aligned} \text{Candidate Keys } R = \{ \\ &(\text{UID}), \\ &(\text{Author ID}), \\ &(\text{Code Venue}), \\ &(\text{Code Venue, JCR year}) \} \end{aligned}$$

and now we compute the attribute closure for all candidate keys in R,

$$(\text{UID})^* = \text{Title, DOI, Issue, Volume, Pub year, \# of Addresses, \# of Pages, \# of Authors, \# of References, Code Venue}$$

$$(\text{Author ID})^* \rightarrow \text{Full Name, First Name, Last Name, Suffix}$$

$$(\text{Code Venue})^* \rightarrow \text{Journal Name, Abbreviation, Discipline, Specialty}$$

$$(\text{Code Venue, JCR Year})^* \rightarrow \text{ISSN, JIF, Total Articles, Total Cities, Quartile}$$

$$(\text{UID, Order})^* \rightarrow \text{Author ID}$$

It is very clear to see that in this instance, the left side of all the functional dependencies is a candidate key for the relation set R. And the right hand side of each functional dependency is thus equal to the attribute closure of the candidate key. Because of this fact, and the fact that all candidate keys for a relation R are super keys for the R, R in our case is BCNF.

Step 5: Choice of DBMS, Web framework and Libraries

DBMS

We chose to use MySQL as our DBMS because of its accessibility with our chosen web framework, and additionally because it is the DBMS that we were taught during the semester so we could ensure that we were all able to contribute. Implementing the database was a simple process because of how we selected our data. We saved all five of our tables as excel files, not CSV because some of our data contains commas. We then created all the tables in MySQL, set the domains, and imported the data.

Web framework and Libraries

Our application is built on the Flask micro web framework for a number of reasons but mostly because of its simplicity. We installed Flask into a Python virtual environment and imported from it a number of useful library functions, including functions to render HTML pages, redirect URLs and set status codes, and create user request objects. We also imported “[Flask-MySQL](#)” which is a Flask extension that allowed us to configure our database connection. Additionally, we imported the Python library “[wtforms](#)” that allowed us to render different types of web forms for retrieving user data. And lastly, we imported the Python “datetime” module so that we could provide the user with the correct time that elements were modified.

Step 6: Interfaces

Interfaces

Over the lifetime of our project we have designed, implemented, and changed our interfaces. From the outset, we were determined to allow a user to search for entries, create entries, and delete entries in the database. Starting with those three core functionalities, we

needed to design how we wanted the user to communicate with the database, what information we needed from the user, and what information we would return.

Search Interface

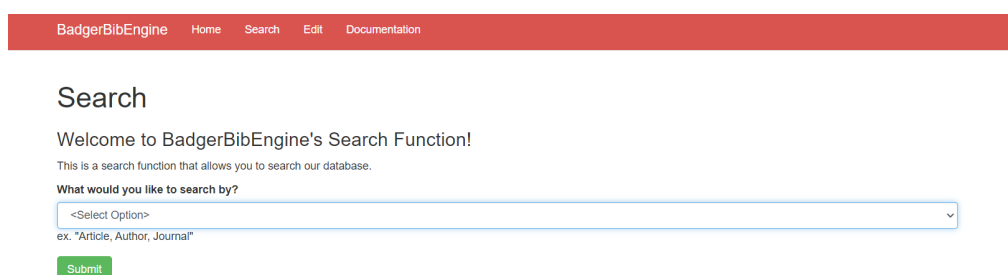


Figure 8: Search Page

Our search interface is arguably the core of our application, as this application serves as a tool to find information on scholarly work. We broke the Search interface down into three subcomponents, regarding the content by which the user would like to search and return information by: article, author, or journal.

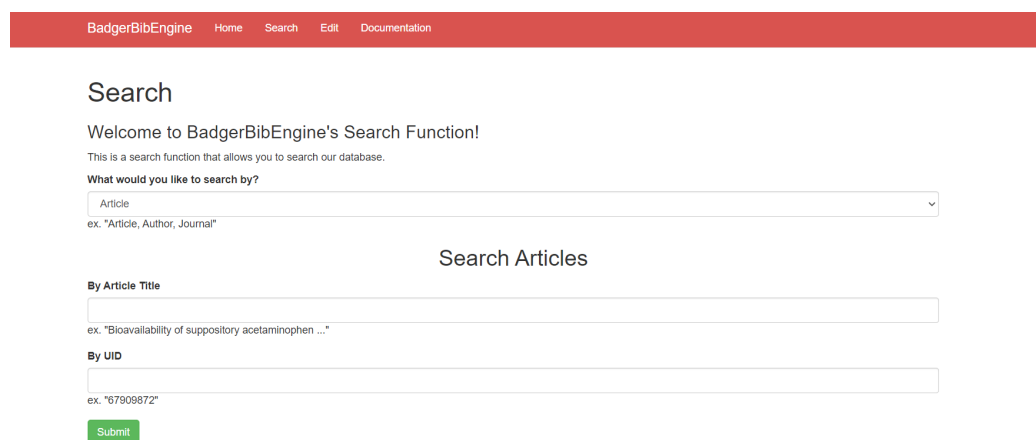


Figure 9: Search By Article Page

If a user is going to search by article, they have the option to return information corresponding to the article's title or the article's UID. Whereas the article's title search will return values for articles that have a substring in their title that is like the user provided string, the article's UID search will be an exact search, meaning that it will only return articles that have

the exact same UID the user provides. Both searches return a table of information about a resulting article - the table contains the article UID, title, authors, DOI, number of pages, issue number, volume number and year published.

Figure 10: Search By Article "Cancer" Result Page

Search Articles

By Article Title

ex. "Bioavailability of suppository acetaminophen ..."

By UID

ex. "67909872"

[Clear Results](#)

~ Found 156 Results ~

UID	Titles	Authors	DOI	# Pages	Issue #	Volume #	Year Published
50863494	4-HPR Is an Endoplasmic Reticulum Stress Aggravator and Sensitizes Breast Cancer Cells Resistant to TRAIL/Apo2L	Anding, Allyson L. Clagett-Dame, Margaret Curley, Robert W., Jr. Newton, Michael A. Jones, James D.	10.21873/anticancerres.12742	14	8	38	2018
50479629	A Ca2+-stimulated exosome release pathway in cancer cells is regulated by Munc13-4	Messenger, Scott W. Woo, Sang Su Martin, Thomas F. J. Sun, Zhongze	10.1083/jcb.201710132	14	8	217	2018

BadgerBibEngine Home Search Edit Documentation

Search

Welcome to BadgerBibEngine's Search Function!

This is a search function that allows you to search our database.

What would you like to search by?

Author

ex. "Article, Author, Journal"

Search Authors

By Author Name

ex. "Betthausen, Tobey J."

By Author ID

ex. "7076883"

[Submit](#)

Figure 11: Search By Author Page

If a user chooses to search by author, they have the option to search by the author's name or by their author ID. Like article search, searching by author name will return data about all author's that contain a substring in their full name that is like the user provided string, and the author ID search is an exact search that only returns information for one author that has the exact author ID that the user provides (e.g. if an author named 'Peter' in the DB has author ID = '7076883', searching by author ID with '7076' will not return Peter's information). If the author information is successfully provided, two tables will be returned, the first will be a table containing the author's ID, the author's full name, the author's number of publications, and the author's number of references. The second table will contain the author ID, the author's full name, and the titles of the articles written by the author. We use two tables here because if you provide an author name that is a substring for more than one author's name in the table, there will be multiple entries returned. Thus, with our current implementation, the first table returns all author results, and the second table provides a list of all titles from those authors, so some entries on the second table could have the same author listed.

Search Authors

By Author Name

ex. "Bethauser, Tobey J."

By Author ID

ex. "7078883"

[Clear Results](#)

~ Found 1 Results ~

Author ID	Author's Name	Number of Publications	Number of References
13568871	Bethauser, Tobey J.	6	195

~ Found 6 Results ~

Author ID	Full Name	Titles
13568871	Bethauser, Tobey J.	In Vivo Comparison of Tau Radioligands F-18-THK-5351 and F-18-THK-5317
13568871	Bethauser, Tobey J.	[F-18]Nifene test-retest reproducibility in first-in-human imaging of 42"nicotinic acetylcholine receptors
13568871	Bethauser, Tobey J.	Human biodistribution and dosimetry of [F-18]nifene, an alpha 4 beta 2"nicotinic acetylcholine receptor PET tracer
13568871	Bethauser, Tobey J.	In Vivo Characterization and Quantification of Neurofibrillary Tau PET Radioligand F-18-MK-6240 in Humans from Alzheimer Disease Dementia to Young Controls

Figure 12: Search By Author Name Page

If a user chooses to search by journal, they have the option to search by the journal's name or by the journal's code venue. As with the other search options, searching by code venue is the only option that requires an exact match. The output of searching by either field will resemble the figure below.

BadgerBibEngine Home Search Edit Documentation

Search

Welcome to BadgerBibEngine's Search Function!

This is a search function that allows you to search our database.

What would you like to search by?

ex. "Article, Author, Journal"

Search Journals

Find a Journal Name

ex. "Academic"

By Code Venue

ex. "7957"

[Clear Results](#)

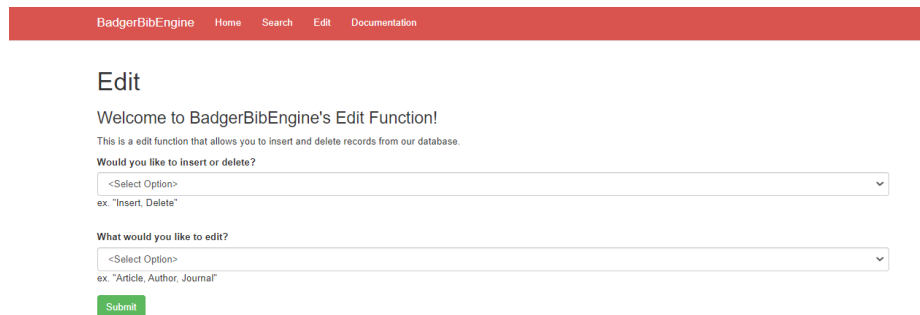
~ Found 6 Results ~

Code Venue	Abbrev	Journal Name	Discipline	Specialty
18985	ACAD PEDIAT	ACADEMIC PEDIATRICS	CLINICAL MEDICINE	PEDIATRICS
21	ACAD EM MED	ACADEMIC EMERGENCY MEDICINE	CLINICAL MEDICINE	GENRL & INTERNAL MED
22	ACAD MED	ACADEMIC MEDICINE	CLINICAL MEDICINE	MISC CLINICAL MED

Figure 13: Search By Journal Name Result Page

Edit Interface

Our edit interface is the other large function of our application, as it allows the user the ability to modify the database. We broke the edit interface down into two subcomponents, regarding the content by which the user would like to edit entries by: insert and delete.

The image shows a web browser window with a red header bar containing the text "BadgerBibEngine Home Search Edit Documentation". Below the header, the page title is "Edit". The main content area has a heading "Welcome to BadgerBibEngine's Edit Function!" followed by a subtext "This is a edit function that allows you to insert and delete records from our database." Below this, there are two dropdown menus. The first dropdown is labeled "Would you like to insert or delete?" and has a placeholder "<Select Option>" and an example "ex: 'Insert, Delete'". The second dropdown is labeled "What would you like to edit?" and has a placeholder "<Select Option>" and an example "ex: 'Article, Author, Journal'". At the bottom of the form is a green "Submit" button.*Figure 14: Edit Page*

If the user chooses to insert, they will have three options of types of entries to insert. If they are looking to insert an article, they will be prompted to enter all attributes stored in the database per article, as well as the author information for the article. For simplicity, we only allow articles to be inserted with a max of three authors.

Insert an Article

Article Information

Article UID

Article Title

Article DOI

Article Issue

Article Volume

Code Venue

Number of Addresses

Number of References

Number of Pages

Number of Authors

Publication Year

Article Author's Information

Format must match: "AuthorID FirstName LastName Suffix"

Author 1

Author 2

Author 3

Figure 15: Insert Article Page

Similarly, if the user wants to insert an author, they will be prompted to insert all fields stored per author in the database.

Insert an Author

Author ID

ex: "123456"

Author First Name

ex: "Peter"

Author Last Name

ex: "Bryant"

Author Suffix

ex: "Mr."

Figure 16: Insert Author Page

The same procedure is followed for inserting a journal.

Insert a Journal

Code Venue
ex: "1234"

Journal Name
ex: "GoodJournal"

Discipline
ex: "Computer Science"

Specialty
ex: "Database Management System Design"

Figure 17: Insert Journal Page

If the user chooses to delete, they will have three options of types of entries to delete, but each type only requires one field. If they are looking to delete an article they will be prompted to provide just the UID, to delete an author they must provide just the author ID, and to delete a journal they just provide the code venue. For brevity, all images of each page besides “delete article” will be omitted, to see all pages and their outputs, please reference our [demo video](#).

BadgerBibEngine Home Search Edit Documentation

Edit

Welcome to BadgerBibEngine's Edit Function!

This is a edit function that allows you to insert and delete records from our database.

Would you like to insert or delete?

Delete

ex: "Insert, Delete"

What would you like to edit?

Article

ex: "Article, Author, Journal"

Delete an Article

Article UID

ex: "60140911"

Figure 18: Delete Article Page

Step 7: Back End Overview (Brief)

MySQL Database Connection

Our Flask application starts by creating an instance of a Flask object, and hard coding the appropriate local MySQL database configuration directly into the Flask object's [config attribute](#). In step 5 we described how we created a MySQL database and populated it would our excel

tables containing information from the Web of Science. In order to connect this database to our application, we used a popular Flask extension called “[Flask-MySQLdb](#)”. Using this extension, we were able to create a [MySQL instance](#) in our application, from which we connected to our database, and made cursor objects for query execution.

```
# Flask Object
app = Flask(__name__)
app.debug = True

#Config MySQL
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = 'Orion007'
app.config['MYSQL_DB'] = 'bibliometric_db'
app.config['MYSQL_CURSORCLASS'] = 'DictCursor'

#Create MySQL Instance
mysql = MySQL(app)
```

Figure 19: MySQL Application Configuration

General Structure

In brief, the general structure and backend design for our application interfaces involves creating “[WTForms](#)” for user input, rendering the correct forms for each combination of search/edit options, collecting user input from the forms, creating a cursor to execute a query on our DB with the user input, and rendering the HTML page with the output of the query in a format that is useful to the user. Below we walk through an example of article search, we omit a walkthrough of edit, but the only difference in execution is that edit is either using INSERT INTO or DELETE FROM in their executed SQL queries to alter records in the database - the output will be the values of the records inserted or deleted.

Example Search Query

Suppose a user is searching for data on an article with “Cancer” in the title. On the search page, the user will insert “Cancer” into our search form, which will set `select_field` to “Article” and `name_exact` to “Cancer”.

```
# Form for Search
class SearchForm(Form):
    select_options = ['<Select Option>', 'Article', 'Author', 'Journal']
    select_field = SelectField('What would you like to search by?', choices=select_options)
    name_exact = StringField('By Article Title', [validators.Length(min=0, max=50)])
    uid = StringField('By UID', [validators.Length(min=0, max=50)])
    journal = StringField('Find a Journal Name', [validators.Length(min=0, max=50)])
    authors_exact = StringField('By Author Name', [validators.Length(min=0, max=50)])
    author_id = StringField('By Author ID', [validators.Length(min=0, max=50)])
    code_venue = StringField('By Code Venue', [validators.Length(min=0, max=50)])
```

Figure 20: Search WTForm in app.py

In our application the search_flag will be set to 1,

```
# SEARCH PAGE
@app.route('/search', methods=['GET', 'POST'])
def search():
    form = SearchForm(request.form)
    if request.method == 'POST' and form.validate():
        # Init variable from user input in WTF form
        select_field = form.select_field.data
        uid = form.uid.data
        name_exact = form.name_exact.data
        journal = form.journal.data
        authors_exact = form.authors_exact.data
        author_id = form.author_id.data
        code_venue = form.code_venue.data

        # Create a MySQL connection and cursor
        cur = mysql.connection.cursor()

        # Search Flag is 1 if user selects Article, 2 if Author, or 3 if Journal
        search_flag = -1
        # Flag is 1-6 corresponding to the search options
        flag = -1

        # Set search flag based on Search By option
        if select_field == "Article":
            search_flag = 1
        elif select_field == "Author":
            search_flag = 2
        elif select_field == "Journal":
            search_flag = 3
        else:
            search_flag = -1
```

Figure 21: Search Function in app.py

and the following section will execute,

```

# Begin Searching
if search_flag == 1 and uid != None and name_exact != None:
    # BY ARTICLE UID
    if uid != '':
        cur.execute("SELECT Distinct a.UID, a.Title, GROUP_CONCAT(c.Full_Name SEPARATOR ' ') as Authors, a.DOI, a.Nb_Page, a.Issue, a.Volume, a.PubYear FROM Article a, Authored_by b, Author c WHERE a.UID = %s and a.UID = %s")
        fetchdata = cur.fetchall()
        print(fetchdata)
        flag = 1
        cur.close()
        return render_template('search.html', form = form, data = fetchdata, flag = flag, search_flag = search_flag)
    # BY ARTICLE NAME
    if name_exact != '':
        cur.execute("SELECT a.UID as UID, a.Title as Title, a.DOI as DOI, a.issue as Issue, a.Volume as Volume, a.PubYear as YearPublished, GROUP_CONCAT(au.Full_Name SEPARATOR ' ') as Authors, a.Nb_Page as NumberofPages, a.issue as Issue FROM Article a JOIN authored_by ab ON a.UID = ab.UID JOIN author au ON au.author_Id = ab.author_Id WHERE a.title LIKE '%" + name_exact + "%' GROUP BY a.title;")
        fetchdata = cur.fetchall()
        flag = 5
        cur.close()
        return render_template('search.html', form = form, data = fetchdata, flag = flag, search_flag = search_flag)

```

Figure 22: Search By Article Execution in app.py

Since name_exact is non-empty, the “if name_exact” condition will execute, meaning the following function call will be executed:

```

cur.execute("SELECT a.UID as UID, a.Title as Title, a.DOI as DOI, a.issue as Issue, a.Volume as Volume,
a.PubYear as YearPublished, GROUP_CONCAT(au.Full_Name SEPARATOR ' ') as Authors, a.Nb_Page
as NumberofPages, a.issue as Issue FROM Article a JOIN authored_by ab ON a.UID = ab.UID JOIN
author au ON au.author_Id = ab.author_Id WHERE a.title LIKE '%" + name_exact + "%' GROUP BY
a.title;")

```

This function will execute the following SQL query on the database:

```

SELECT a.UID as UID, a.Title as Title, a.DOI as DOI, a.issue as Issue, a.Volume as Volume,
a.PubYear as YearPublished, GROUP_CONCAT(au.Full_Name SEPARATOR ' ') as Authors, a.Nb_Page
as NumberofPages, a.issue as Issue
FROM Article a JOIN authored_by ab ON a.UID = ab.UID JOIN author au ON au.author_Id =
ab.author_Id
WHERE a.title LIKE '%Cancer%' GROUP BY a.title;

```

The function “fetchall()” will store the output of the SQL query into a variable name, which we pass along with both flags into the render_template() function which will render the search page again in order to format the search results. In the search.html page, since flag was set to 5, the following condition will execute,

```

<!-- PRINT: "BY ARTICLE EXACT TITLE RESULT" -->
{% if flag == 5 %}
<table class="center" border="1">
  <tr>
    <br>
    <h3 style="padding:10px ; text-align:center; font-size:medium font-"> <i>~ Found {{data.length}} Results ~</i> </h3>
    <th style="padding:10px; text-align:center">UID</th>
    <th style="padding:10px; text-align:center">Titles</th>
    <th style="padding:10px; text-align:center">Authors</th>
    <th style="padding:10px; text-align:center">DOI</th>
    <th style="padding:10px; text-align:center"># Pages</th>
    <th style="padding:10px; text-align:center">Issue #</th>
    <th style="padding:10px; text-align:center">Volume #</th>
    <th style="padding:10px; text-align:center">Year Published</th>
  </tr>
  {% for entry in data %}
  <tr>
    <td style="padding:10px">{{entry.UID}}</td>
    <td style="padding:10px">{{entry.Title}}</td>
    <td style="padding:10px">{{entry.Authors}}</td>
    <td style="padding:10px">{{entry.DOI}}</td>
    <td style="padding:10px; text-align:center">{{entry.NumberofPages}}</td>
    <td style="padding:10px; text-align:center">{{entry.Issue}}</td>
    <td style="padding:10px; text-align:center">{{entry.Volume}}</td>
    <td style="padding:10px; text-align:center">{{entry.YearPublished}}</td>
  </tr>
  {% endfor %}
</tr>
</table>
{% endif %}

```

Figure 23: By Article Title Table in search.html

Which will print a table with article attributes as table headers and all article attributes returned from the SQL query as entries. The output of the application will look like,

~ Found 156 Results ~

UID	Titles	Authors	DOI	# Pages	Issue #	Volume #	Year Published
50863494	4-HPR Is an Endoplasmic Reticulum Stress Aggravator and Sensitizes Breast Cancer Cells Resistant to TRAIL/Apo2L	Anding, Allyson L. Clagett-Dame, Margaret Curley, Robert W. Jr. Newton, Michael A. Jones, James D.	10.21873/anticancer.12742	14	8	38	2018
50479629	A Ca2+-stimulated exosome release pathway in cancer cells is regulated by Munc13-4	Messenger, Scott W. Woo, Sang Su Martin, Thomas F. J. Sun, Zhongze	10.1083/jcb.201710132	14	8	217	2018
48365939	A Comparison of Disease Characteristics and Treatment Outcomes for Early Stage Squamous Cell Versus Adenocarcinoma of the Cervix: A National Cancer Database Analysis	Bradley, Kristin A. Rosenberg, Stephen A. Wojcieszynski, A. P., Jr. Hullett, C. R.	10.1016/j.ijrobp.2017.06.1308	2	2	99	2017
54975466	A Human Ribonuclease Variant and ERK-Pathway Inhibition Exhibit Ulnar	Tanrikulu, I. Caglar Raines, Ronald T. Lippman, Thomas M.	10.1158/1078-0432.CCR-17-0774	14	12	17	2018

Figure 24: By Article Title Table Result

Step 8: Test Cases

Testing Overview

In order to make sure that our application was returning correct results, we decided to run a number of evaluations to test our application against queries run directly from MySQL workbench on the same database. The following five evaluations contain a sentence of the

functionality we are testing, a screenshot of our application after doing the operation, the MySQL query, and a screenshot of the output from running the query in MySQL workbench.

Evaluation 1

Functionality Test: This application is able to search for the article name “*A Human Ribonuclease Variant and ERK-Pathway Inhib...*” and return the exact article title, along with all its associated attributes.

By Article Title

A Human Ribonuclease Variant and ERK-Pathway Inhib
ex. "Bioavailability of suppository acetaminophen ..."

By UID

I
ex. "67909872"

Clear Results

~ Found 1 Results ~

UID	Titles	Authors	DOI	# Pages	Issue #	Volume #	Year Published
51325466	A Human Ribonuclease Variant and ERK-Pathway Inhibitors Exhibit Highly Synergistic Toxicity for Cancer Cells	Hoang, Trish T. Tanrikulu, I. Caglar Vatland, Quinn A. Hoang, Trieu M. Raines, Ronald T.	10.1158/1535-7163.MCT-18-0724	11	12	17	2018

Figure 25: Screenshot of BadgerBibEngine Search by Article Title, “*A Human Ribonuclease Variant and ERK-Pathway Inhib*” Result

The corresponding MySQL query,

```
SELECT a.UID as UID, a.Title as Title, a.DOI as DOI, a.issue as Issue, a.Volume as Volume,
a.PubYear as YearPublished, GROUP_CONCAT(au.Full_Name SEPARATOR ' ') as Authors, a.Nb_Page
as NumberofPages, a.issue as Issue
FROM Article a JOIN authored_by ab ON a.UID = ab.UID JOIN author au ON au.author_Id =
ab.author_Id
WHERE a.title LIKE '%A Human Ribonuclease Variant and ERK-Pathway Inhibitors Exhibit
Highly Synergistic Toxicity for Cancer Cells%' GROUP BY a.title;
```

Results of query as run through MySQL workbench (runtime = 0.078 seconds),

UID	Title	DOI	Issue	Volume	YearPublished	Authors	NumberofPages	Issue
51325466	A Human Ribonuclease Variant and ERK-Pathwa...	10.1158/1535-7163.MCT-18-0724	12	17	2018	Hoang, Trish T. Tanrikulu, I. Caglar Vatland, Qu...	11	12

Figure 26: Screenshot of MySQL Workbench Result for Eval #1

Evaluation 2

This application is able to search for an author name by the author ID “35882022” and return the exact author name, author ID, number of publications and references, and all of their published article titles.

By Author ID

35882022

ex. "7076883"

Clear Results

~ Found 1 Results ~

Author ID	Author's Name	Number of Publications	Number of References
35882022	Vavilov, Maxim G.	3	138

~ Found 3 Results ~

Titles
Phonon-mediated quasiparticle poisoning of superconducting microwave resonators
Effects of charge noise on a pulse-gated singlet-triplet S-T_qubit
Weyl nodes in Andreev spectra of multiterminal Josephson junctions: Chern numbers, conductances, and supercurrents

Figure 27: Screenshot of BadgerBibEngine Search by Author ID, “35882022” Result

The corresponding MySQL queries, the first for author information,

```
SELECT b.author_id, b.Full_Name, COUNT(a.UID) as count, CAST(SUM(a.Nb_Reference) as
SIGNED) as sum
```

```
FROM article a, author b, authored_by c
```

```
WHERE b.author_id = "35882022" AND b.author_id = c.author_id AND c.UID = a.UID;
```

and the second for all article titles,

```
SELECT a.Title
```

```
FROM article a, author b, authored_by c
```

```
WHERE b.author_id = "35882022" AND b.author_id = c.author_id AND c.UID = a.UID;
```

Results of query as run through MySQL workbench (runtime = 0.015 seconds),

				Title
author_id	Full_Name	count	sum	
35882022	Vavilov, Maxim G.	3	138	Phonon-mediated quasiparticle poisoning of superconducting microwave resonators
				Effects of charge noise on a pulse-gated singlet-triplet S-T _q ubit
				Weyl nodes in Andreev spectra of multiterminal Josephson junctions: Chern numbers, conductances, and supercurrents

Figure 27: Screenshot of MySQL Workbench Result for Eval #2

Evaluation 3

Functionality Test: This application is able to search for a journal by the name “ACADEMIC” and return all of the journal's attributes.

Search Journals

Find a Journal Name

ACADEMIC

ex. "Academic"

By Code Venue

ex. "7967"

[Clear Results](#)

~ Found 6 Results ~

Code Venue	Abbrev	Journal Name	Discipline	Specialty
18985	ACAD PEDIAT	ACADEMIC PEDIATRICS	CLINICAL MEDICINE	PEDIATRICS
21	ACAD EM MED	ACADEMIC EMERGENCY MEDICINE	CLINICAL MEDICINE	GENRL & INTERNAL MED
22	ACAD MED	ACADEMIC MEDICINE	CLINICAL MEDICINE	MISC CLINICAL MED
23	ACAD PSYCHI	ACADEMIC PSYCHIATRY	PROFESSIONAL FIELDS	EDUCATION
25	ACAD RADIOL	ACADEMIC RADIOLOGY	CLINICAL MEDICINE	RADIOLOGY & NUCL MED
7967	J ACAD LIBR	JOURNAL OF ACADEMIC LIBRARIANSHIP	PROFESSIONAL FIELDS	INFO & LIBRARY SCI

Figure 28: Screenshot of BadgerBibEngine Search by Journal Name, “ACADEMIC” Result

The corresponding MySQL query,

```
SELECT DISTINCT j.Code_Venue, j.abbrev, j.journal_name, j.discipline, j.specialty
FROM journal j
WHERE j.journal_name LIKE '%ACADEMIC%';
```

Results of query as run through MySQL workbench (runtime = 0.015 seconds),

Code_Venue	abbrev	journal_name	discipline	specialty
18985	ACAD PEDIAT	ACADEMIC PEDIATRICS	CLINICAL MEDICINE	PEDIATRICS
21	ACAD EM MED	ACADEMIC EMERGENCY MEDICINE	CLINICAL MEDICINE	GENRL & INTERNAL MED
22	ACAD MED	ACADEMIC MEDICINE	CLINICAL MEDICINE	MISC CLINICAL MED
23	ACAD PSYCHI	ACADEMIC PSYCHIATRY	PROFESSIONAL FIELDS	EDUCATION
25	ACAD RADIOL	ACADEMIC RADIOLOGY	CLINICAL MEDICINE	RADIOLOGY & NUCL MED
7967	J ACAD LIBR	JOURNAL OF ACADEMIC LIBRARIANSHIP	PROFESSIONAL FIELDS	INFO & LIBRARY SCI

Figure 29: Screenshot of MySQL Workbench Result for Eval #3

Evaluation 4

Functionality Test: This application is able to insert a new value into the journal table and verify that it exists.

Insert a Journal

Code Venue
12345678910
ex. "1234"

Journal Name
GoodJournal
ex. "GoodJournal"

Discipline
COMPUTER SCIENCE
ex. "Computer Science"

Specialty
DATABASE DESIGN
ex. "Database Management System Design"

Journal recieved, thanks for the contribution!

Inserted @ 10:44:25 on 08/02/2022

Code Venue	Journal Name	Discipline	Specialty
12345678910	GoodJournal	COMPUTER SCIENCE	DATABASE DESIGN

Search Journals

Find a Journal Name
ex. "Academic"

By Code Venue
12345678910
ex. "7967"

Clear Results

~ Found 1 Results ~

Code Venue	Abbrev	Journal Name	Discipline	Specialty
12345678910	None	GoodJournal	COMPUTER SCIENCE	DATABASE DESIGN

Figure 30: Screenshot of BadgerBibEngine Insert Journal, "12345678910" Result & Search Journal by Code Venue, "12345678910" Result

The corresponding MySQL query,

```
INSERT INTO journal (Code_Venue, Journal_Name, Discipline, Specialty)
VALUES( '12345678910', 'GoodJournal', 'COMPUTER SCIENCE', 'DATABASE DESIGN');
```

Results of query as run through MySQL workbench (runtime = 0.012 seconds),

```
42 • Select j.code_venue, j.journal_name, j.discipline, j.specialty
43 From Journal j
44 Where j.code_venue = '12345678910';
45
```

code_venue	journal_name	discipline	specialty
12345678910	GoodJournal	COMPUTER SCIENCE	DATABASE DESIGN

```
42 • Select j.code_venue, j.journal_name, j.discipline, j.specialty
43 From Journal j
44 Where j.code_venue = '12345678910';
45
```

code_venue	journal_name	discipline	specialty
12345678910	GoodJournal	COMPUTER SCIENCE	DATABASE DESIGN

Figure 31: Screenshot of MySQL Workbench Result for Eval #1, Left is before insert, Right is after insert

Evaluation 5

Functionality Test: This application is able to delete a journal from the database

Delete a Journal

Code Venue

 ex. "805"

Journal deleted!

Deleted @ 10:49:09 on 08/02/2022

Code Venue	Abbrev	Journal Name	Discipline	Specialty
12345678910	None	GoodJournal	COMPUTER SCIENCE	DATABASE DESIGN

Figure 32: Screenshot of BadgerBibEngine Delete Journal with Code Venue = "12345678910" Result

The corresponding MySQL query,

DELETE FROM journal WHERE code_Venue = '12345678910';

Results of query as run through MySQL workbench (runtime = 0.013 seconds),

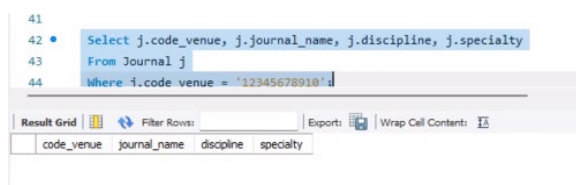


Figure 33: Screenshot of MySQL Workbench Result for Eval #5

Step 9: Room for Improvement

General

At this point in the project, our application still has much room for improvement, despite all main interfaces being fully functional. Due to the time constraints of a Summer course, we had to make strategic decisions to build out our main search and edit interfaces before focusing on other features that would have otherwise made our application more robust. The following sections briefly describe two other interfaces, More and User Login/Registration, that are implemented in our application and are functional for the tasks we wanted to use them for but were not included in the interface section because we feel they need more time to be built out.

Images of the results of the operations are omitted, but we again advise you to see the results shown in our [demo video](#).

More

Our More interface allows the user to run one of three extra operations on our database to extract more information. Of the three operations, the user can:

Option 1: Find the authors who have more than a given number of publications.

Option 2: Find the journals and their specialties that have published articles whose titles contain a given word.

Option 3: Find the most productive authors by journal name.

All of these operations are functional, but we felt that the interface doesn't have as many options as we wanted to create and that it was otherwise unfinished.

Figure 34: More Operations Page

Given more time we would have added more options to the More page, and would have returned more data from each option.

User Login/Registration

Our User login/registration interface is a skeleton of what we would like it to be. When a user opens our application they immediately open the Login page where they are prompted to enter their username and password to proceed to our home page and the rest of the application.

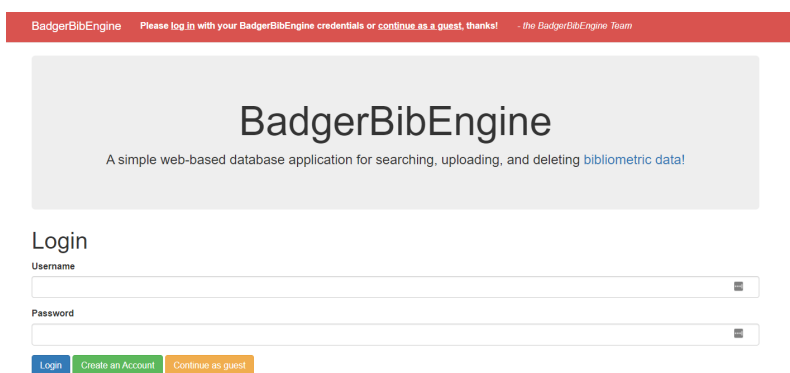


Figure 35: Login Page

Our application, at the moment, uses a trivial and non secure method to store user information. If the user doesn't have an account, they can choose to click create an account, and register an account with us which will store their created username and password in a table in our database.

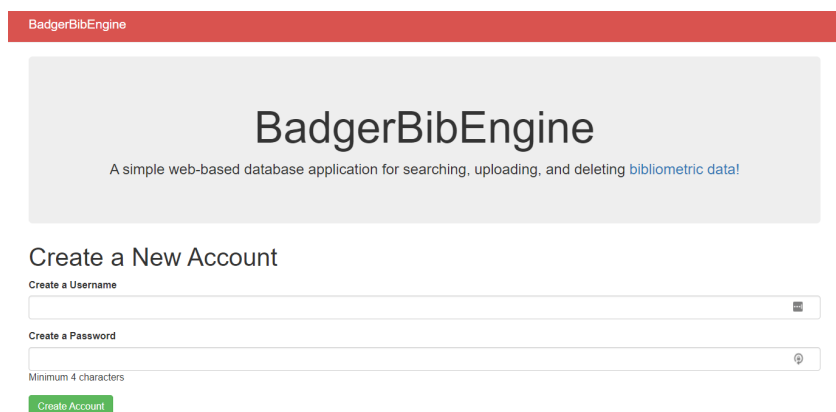


Figure 36: Registration Page

When a user registers an account they will be prompted to sign in where we will query our user table and make sure a user in the database contains the provided username and password. If there exists a copy of the provided username and password, the page will redirect to our homepage, opening our application. Otherwise it will refresh the login page with a warning message. For the sake of demonstrating the application, we currently have a button that says continue as guest, that allows the user to access the home page with full permissions.

If we had more time to work on the project, we would change a number of things to build this interface out more. Currently user information is stored unencrypted in plain-text in our database's user table. The first thing we would do is use other Flask libraries for user information encryption that will allow us to securely store their information. Additionally, we would refactor our ER diagram so that we have information stored per user per entry they have inserted, deleted, or searched. With this information we could better build out user profiles with stored history on what operations they have performed, and create an user account interface so that they can view their user history. Another change we would make is to modify the permissions of someone who is not signed in and who is using the continue as guest option. This user should be able to search or use the More interface but should be required to sign into their account in order to insert or delete entries from the table.

Web Scraping for More Article Data

One small change we would like to make is to go back to our data collection stage and web scrape for the URLs and other relevant links for authors, articles, and journals. For instance, we could return a webpage link to an article when you search for it, or a personal webpage for an author if you searched for them. This would require us to change our ER diagram and relational schema, but we think it would offer the end user much more utility.

Hosting

Lastly, we plan on hosting this application online once we have finished securely storing user information. We will need to host our MySQL database online as well, which will require that we reconfigure our database connection in our application's backend.

Step 10: General Contributions Breakdown

Lastly, we explain our individual contributions. We made these very general because there was a fair amount of overlap between roles and team members.

Xiang Zheng did most of our data collection and cleaning. Once the data was imported, we all set up local MySQL databases in order to write and test queries. We all jointly worked to design our entity-relationship model and Willie Klein designed and produced our final ER diagram. From our finished diagram, we all contributed to and finished our relational model. Our tables were already BCNF so there was no need for any of us to normalize our tables. Peter Bryant designed the general architecture and interfaces for the web application. Peter and Willie did all of our software development, including our front and back end implementation and subsequent iterations of our design and interfaces as the project progressed. We all contributed to writing MySQL queries for the main interfaces, Xiang wrote additional queries serving as the operations on the More page.

References

Bibliometrics. (n.d.). Datalab.ucdavis.edu. Retrieved August 2, 2022, from <https://datalab.ucdavis.edu/bibliometrics/#:~:text=As%20data%2C%20bibliometrics%20typically%20measure>

Clarivate. (n.d.). Access.clarivate.com. <https://www.webofscience.com>

Flask-WTF — Flask-WTF Documentation (1.0.x). (n.d.). Flask-Wtf.readthedocs.io. <https://flask-wtf.readthedocs.io/en/1.0.x/>

Flask-MySQL — flask-mysql 1.4.0 documentation. (n.d.). Flask-Mysql.readthedocs.io. Retrieved August 2, 2022, from <https://flask-mysql.readthedocs.io/en/stable/#:~:text=Flask%2DMySQL%20is%20a%20Flask>

Welcome to Flask-MySQLdb's documentation! — Flask-MySQLdb 0.2.0 documentation. (n.d.). Flask-Mysqldb.readthedocs.io. Retrieved August 1, 2022, from <https://flask-mysqldb.readthedocs.io/en/latest/>