Team 5 – Derek Kneisel, Ethan Kochis, Alec Lopez, Tomer Singal, Marc Kaliroff, Jamie Yoos

CSC 315-01

GitHub Wiki: https://github.com/CSC-315/cab-tag-we-are-it/wiki

Professor DeGood

27 April 2020

Stage Vb- Construction

In this phase, we created the UI. In order to run the UI, run the following commands in the VM environment:

- sudo apt install python3-flask
- pip3 install python3-flask_cors

Then, in a terminal, navigate to the cloned repository folder and enter the "code" folder. Run the commands "export FLASK_APP=backend.py" and "flask run". This will launch the Flask server. Then, in the GUI folder explorer, navigate to the cloned repository folder and enter the "code" folder. Double-click on the "index.html" file and it will launch your web browser to our website.

article.html & article.js → Front-end code to generate the page for a given article

articles.html & articles.js → Front-end code to generate the page for articles

index.html & frontend.js → Front-end code to generate the home page

tags.html & tags.js → Front-end code to generate the page for tags

style.css → Style sheet for the front-end

backend.py → Back-end code to run SQL queries on the database

config.py → Code to connect to the database, given to us by Professor DeGood, uses database.ini

database.ini → File that contains the needed parameters to establish a connection with the database

init_db.sh → Shell script that automatically creates the database, tables, views, and inserts the sample data

Stage Va- Construction

We created the SQL for our queries as part of Stage IV, see the section **Complete Set of Queries** in Stage IV- Design part of document. See create_tables.sql, data.py, insert_data.py, create_view.sql, example_queries.sql files under the "code" folder on the GitHub. Also, on Ethan Kochis's VM these files live at /home/osc/Desktop/cab-tag-we-are-it/code. The database name is project (e.g. run "psql project" to get interactive SQL shell).

create_tables.sql → Creates the tables

data.py → Contains 10 articles copy-pasted from srhub.org as well as our tagging information for our database

insert_data.py → Python script to insert the data from data.py into the project database

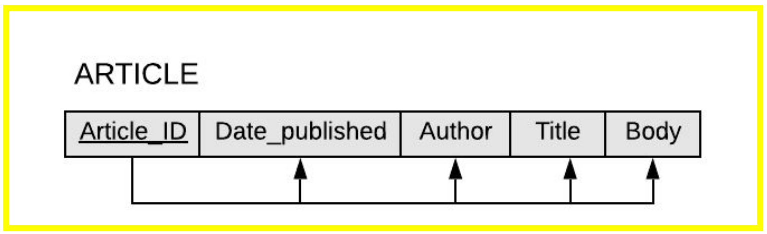
Create_view.sql → Create our ARTICLE_TAGS view as shown in Stage IV under the section **Required Views**

example_queries.sql → Contains all of our SQL queries as described in the Stage IV section **Complete Set of Queries** that are not INSERT, DELETE, or UPDATE related to show that our database works

Stage IV- Design

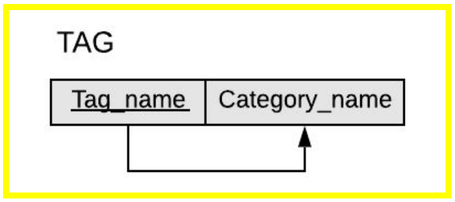
Relations Normalized to BCNF

ARTICLE



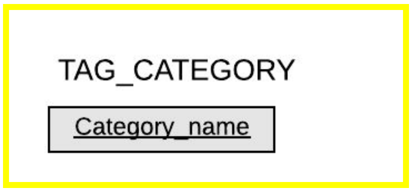
ARTICLE is in **BCNF**. ARTICLE is in **1NF**, because it only has atomic values. ARTICLE is in **2NF**, because each nonprime attribute, namely Date_published, Author, Title, and Body, is fully functionally dependent on the prime attribute, namely Article_ID. ARTICLE is in **3NF**, because it has no transitive dependencies. Finally, ARTICLE is in **BCNF**, because there is only one nontrivial functional dependency in ARTICLE, namely the one shown in the figure above.

TAG



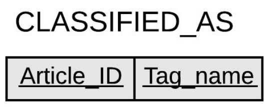
TAG is in **BCNF**. TAG is in **1NF**, because it only has atomic values. TAG is in **2NF**, because the nonprime attribute - Category_name - is fully functionally dependent on the prime attribute - Tag_name. TAG is in **3NF**, because it has no transitive dependencies. Finally, TAG is in **BCNF**, because there is only one nontrivial functional dependency in TAG, namely the one shown in the figure above.

TAG_CATEGORY



TAG_CATEGORY is in **BCNF**. TAG_CATEGORY is in **1NF**, because it only has atomic values. TAG_CATEGORY is in **2NF**, because there are no nonprime attributes. TAG_CATEGORY is in **3NF**, because it has no transitive dependencies. Finally, TAG is in **BCNF**, because there are no nontrivial functional dependencies.

CLASSIFIED_AS



CLASSIFIED_AS is in **BCNF**. CLASSIFIED_AS is in **1NF**, because it only has atomic values. CLASSIFIED_AS is in **2NF**, because there are no nonprime attributes. CLASSIFIED_AS is in **3NF**, because it has no transitive dependencies. Finally, CLASSIFIED_AS is in **BCNF**, because there are no nontrivial functional dependencies.

Required Views

Define the different views required. For each view list the data and transaction requirements.

Give a few examples of queries, in English, to illustrate.

```
CREATE VIEW ARTICLE_TAGS AS
        SELECT * FROM ARTICLE
                NATURAL JOIN CLASSIFIED_AS
                NATURAL JOIN TAG
```

Data: Article_ID, Date_published, Author, Title, Body, Tag_name, Category_name

Transaction Requirements: Whenever we need to get info for articles and their tags

Example Queries: Get article info for a given tag, Get article info for multiple different tags, Get article info for articles with multiple specific tags

Complete Set of Queries

Design a complete set of queries to satisfy the transaction requirements identified in the previous stages.

```
# Get article info for a given tag
SELECT *
FROM ARTICLE_TAGS
WHERE Tag_name = 'example tag';
```

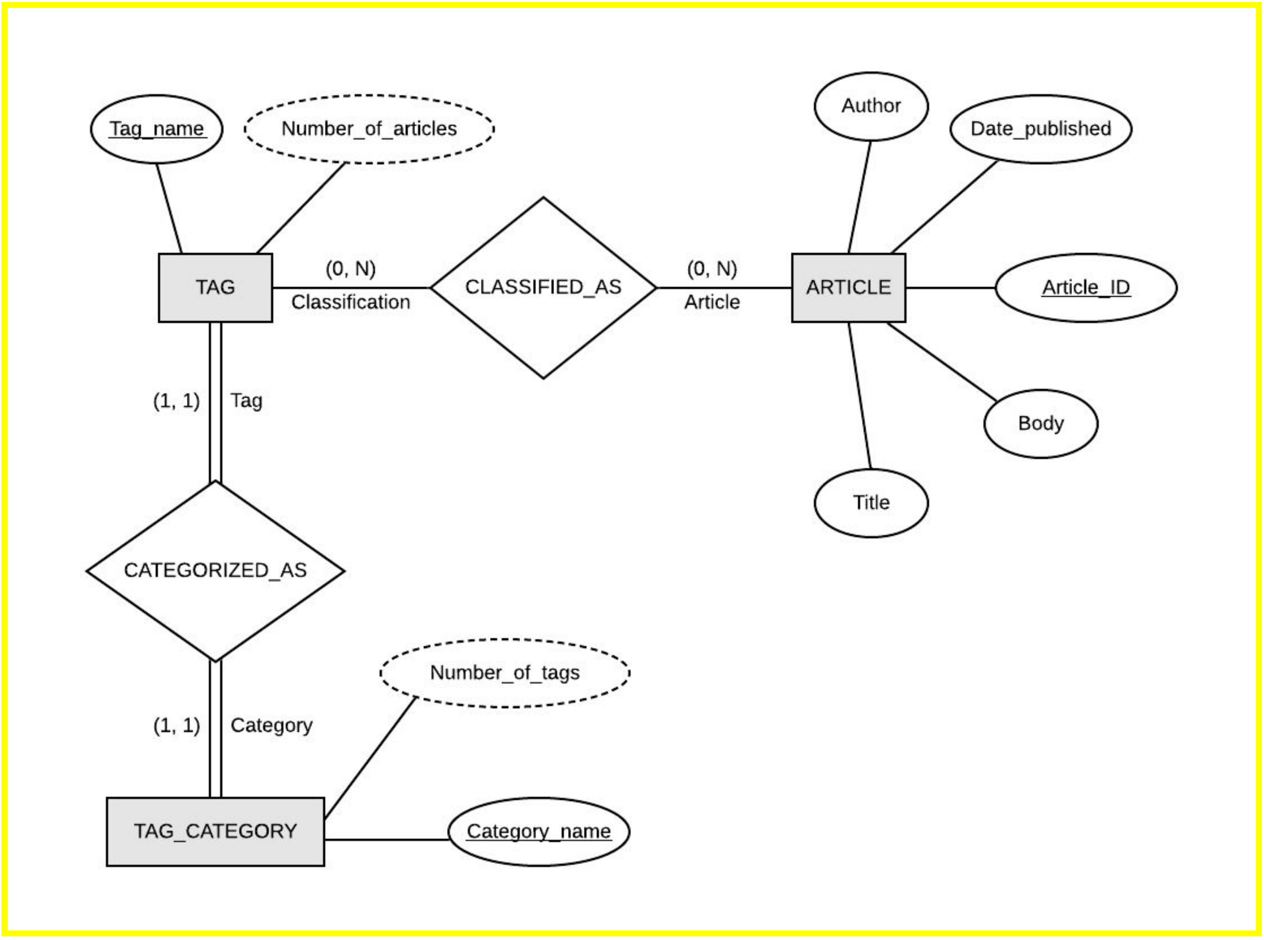
```
# Get number of tags for all categories

SELECT COUNT(Tag_name)

FROM TAG_CATEGORY NATURAL JOIN TAG

GROUP BY Category_name;


# Get all article info for multiple different tags

SELECT *

FROM ARTICLE_TAGS

WHERE Tag_name = 'example tag' OR Tag_name = 'example tag 2' OR Tag_name ='example

tag 3';


# Get article info for articles with multiple specific tags

SELECT *

FROM ARTICLE

WHERE Article_id=

(SELECT Article_id

FROM ARTICLE_TAGS

WHERE Tag_name = 'example tag'

INTERSECT

SELECT Article_id

FROM ARTICLE_TAGS
WHERE Tag_name = 'example tag 2');

# Get all tag names for a given article

SELECT Tag_name

FROM CLASSIFIED_AS

WHERE Article_ID=1;


# Select ARTICLE based on ID

SELECT *

FROM ARTICLE

WHERE Article_ID=1;


# Insert into ARTICLE

INSERT INTO ARTICLE (Date_published, Author, Title, Body)

        VALUES ('2017-03-19', 'Georgina Smith', 'Organization does Good Thing', 'Yesterday

morning an organization...');


# Insert into TAG

INSERT INTO TAG (Tag_name, Category_name)

        VALUES ("beach cleanup", "climate action");
```

```
# Insert into TAG_CATEGORY

INSERT INTO TAG_CATEGORY (Category_name)

        VALUES ("climate action");


# Insert into CLASSIFIED_AS

INSERT INTO CLASSIFIED_AS (Article_ID, Tag_name)

        VALUES(1, 'beach cleanup');


# Update ARTICLE

UPDATE ARTICLE SET

        "Date_published"='2018-06-02',

        "Author"='Michael Bloomberg',

        "Title"='New Article Title',

        "Body"='New Article Body…'

WHERE Article_ID = 3;


# Update TAG

UPDATE TAG SET

        Tag_name='new name'

        Category_name=' new category name'

WHERE Name='old name'
```

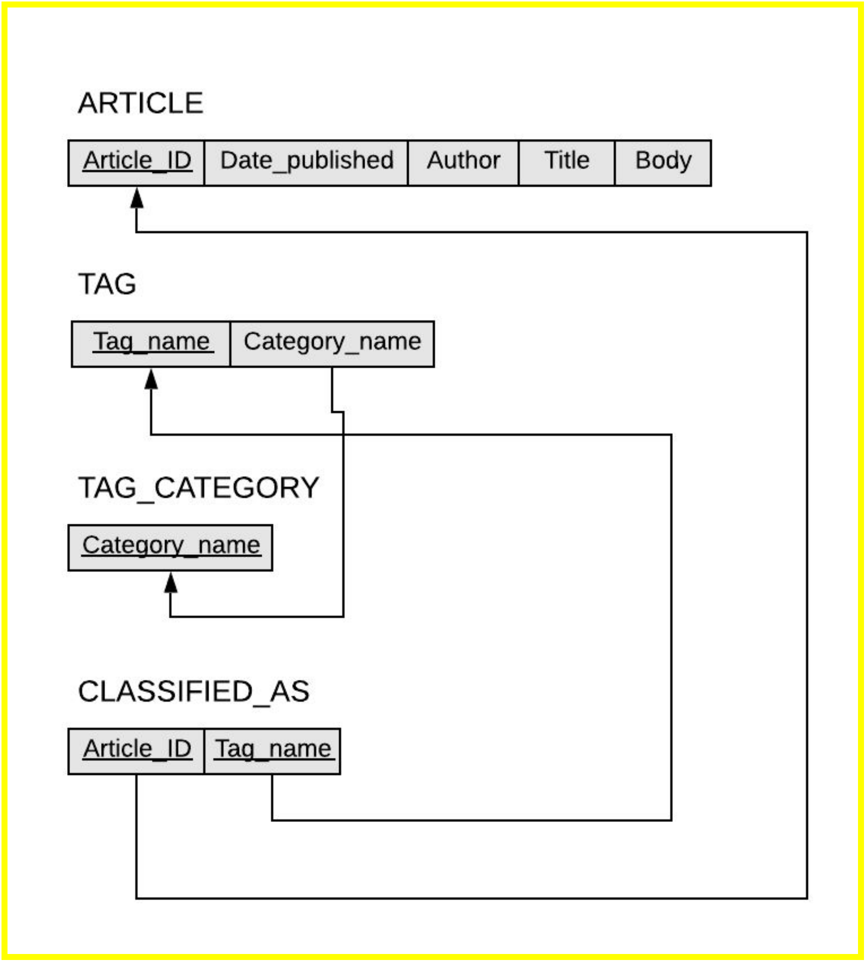
```
#Update CLASSIFIED_AS

UPDATE CLASSIFIED_AS SET

        Article_ID=1

        Tag_name='example tag'

WHERE Article_ID=2;


# Update TAG_CATEGORY

UPDATE TAG_CATEGORY SET

        "Category_name"='new name'

WHERE Category_name = 'old name';


# Delete from ARTICLE

DELETE FROM ARTICLE WHERE Article_ID=1;


# Delete TAG

DELETE FROM TAG WHERE Tag_name = 'name';


# Delete TAG_CATEGORY

DELETE FROM TAG_CATEGORY WHERE Category_name = 'name';


#Delete CLASSIFIED_AS

DELETE FROM CLASSIFIED_AS WHERE Article_ID=1 AND Tag_name='name';
```

Stage III - Database Model

Complete Entity-Relationship (ER) Diagram:



Relational Schema:



Estimation of:

- **Initial database size (approximate number of records):**
  - ~50 articles, ~10 tags, ~3 tag categories
- **Types and average number of searches**
  - Find a specific article's tags: 300/day
  - Find all articles for a given tag: 75/day
  - Find all tag categories and their tags: 200/day

Stage II - Project Proposal and Specifications

- **Problem statement.**
  - Without organization it is difficult to keep people's attention. Currently, SRHub provides little organization for the articles, showing articles in order of recency. People interested in specific kinds of articles, such as climate change articles, may not be willing to scroll through the website to find them. The website lacks the organization that would help it attract frequent visitors.
- **Objective of the module.**
  - We aim to solve the problem by creating a tags module that would enable users to find articles that they are interested in more easily. For example, a visitor may wish to see articles related to trash cleanups. The tag module will allow them to do that without having to look through unrelated articles. Many different tag categories could be created. Some examples are: Renewable Energy, News Source, Sustainable, Ecofriendly, Climate Change, Recycle, Nature, etc. Under the Renewable Energy category, we could have the tags Solar Energy, Wind Energy, etc.

- **Description of the desired end product, and the part you will develop for this class.**
  - The desired end product would be an SRHub website that contains the ability to sort articles by their tags and click on different tags to be able to view articles that pertain to that tag. It would also have the ability to also search for articles based on keywords provided by the admin. The maintainers of the SRHub website

    - would also be able to apply tags to articles as they see fit based on the subject matter of the article.
    - In this class we will develop the database backend for the tags, involving the table structure, as well as necessary queries to create and update tags. We will also provide the structure that would allow querying articles by a selected tag. This will all be presented via a frontend that will display the tags and show relevant articles when a tag is selected.
- **Description of the importance and need for the module, and how it addresses the problem.**
    - This module will allow users to filter articles to be able to access the ones that they are most interested in. This addresses the problem of a lack of organization that deters users by allowing them to organize articles into the topics that they want to see. This will make navigating the website easier. Additionally, it will keep users' attention for longer. When a user finds interest in an article, they will be able to continue exploring that interest by selecting a tag belonging to the article and see more similar articles.

- **Plan for how you will research the problem domain and obtain the data needed.**
    - The journalism students should have experience with articles, so we will discuss which tags they believe would best represent the available articles. A good selection of tags is important.

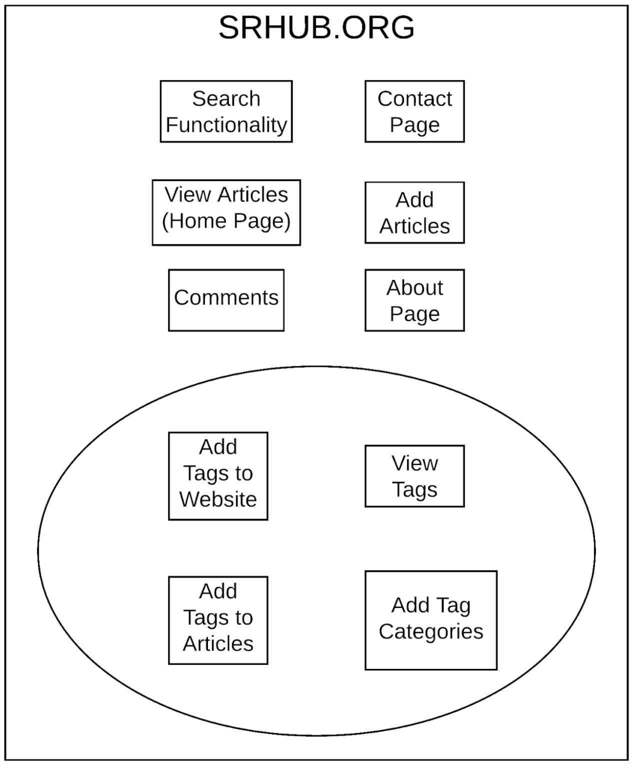
- We will have to populate the database with article data, for which we will use existing articles in SRHub's website. This will allow the tags to be representative of SRHub's actual content.
- We will also look at other news websites and study their categorization system and tag selection.

- **Other similar systems / approaches that exist, and how your module is different or will add to the existing system.**
  - **https://www.enn.com/ -** This website has tags for each article, however, there is no page that has any way of viewing all the different tags. In our implementation we will display all the tags as well as showing the most popular tags/articles.
  - **http://abcnewsradioonline.com/business-news-tags/** - This website has a page that has all the tags in a given category, however they are just written alphabetically in lines and some have a larger font if a large number of articles have that tag. This makes viewing the information difficult, especially if you just want to explore the tags and do not know what you are looking for.
  - **https://tomeraberba.ch** - This website shows how tags could be depicted beneath articles as well as showing how clicking on a tag will show you articles with that tag.
- **Possible other applications of the system (how it could be modified and reused).**
  - These tags could be integrated into the search functionality and show up as suggestions when the user types in the search field.

    - Articles could display all of their tags and allow users to navigate from article to article in this fashion.

- **Performance – specify how and to what extent you will address this.**
    - Performance is mostly simple for the database system, but we will make sure that our queries and table design are efficient. We will have to ensure that the backend does not take up too much memory when making queries. For the frontend, time and space efficiency will need to be considered, especially if external CSS or Javascript libraries are used.
- **Security – specify how and to what extent you will provide security features.**
    - Write access to the database in the goal product will be restricted to admins only. Website users will only have read access when making their tag selection. This means that attacks such as SQL injections should not be a concern. One strategy that can be used for security is assuming that a user does not have permission until proven otherwise.
- **Backup and recovery – specify how and to what extent you will implement this.**
    - According to the PostgreSQL docs, the following actions are supported: SQL dump, file system level backup, and continuous archiving. We will look into this later and decide which action(s) best support our needs, or if we would rather implement our own backup system.
- **Technologies and database concepts the team will need to learn, and a plan for learning these.**

    - We will have multiple tables, including at least tags and articles. We will need to learn to join these tables in our queries. We will also need to learn PostgreSQL syntax and how to set it up.
    - The team will also need to learn frontend and backend technologies, including JavaScript, HTML, CSS. We will also need to learn database libraries for our backend language. Creating the frontend-backend interaction will also be a learning experience.
- **A diagrammatic representation of the system boundary that specifies what data you will model and which queries you will implement.**



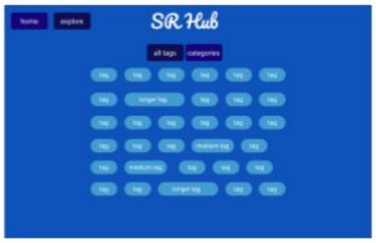
- **1-page quad chart; see: Quad_instructions_template.ppt in the Canvas files section.**

Tag You're It!

Derek Kneisel, Tomer Singal, Alec Lopez, Ethan Kochis, Marc Kaliroff, Jamie Yoos

Objective

- Problem: Current articles are not well organized. Users have to search through articles that they aren't interested in to find those that do interest them.
- Goal: Have each article tagged so website visitors can find articles they are interested in.
  - Have tag categories, like Renewable Energy, that would encompass tags like Solar Energy, Wind Energy, etc.
    - Enable visitors to see articles that have tags with similar meaning



Mockup of the tags page

Approach

1) Have each article tagged, which would show up when the article is being read (either under the by-line or at the end of the article). These tags could be clicked and show you other articles with that tag.
2) Have a webpage designed to explore the different tags and tag categories on SRHub.

Key Milestones

• Specifications	02/24/2020
• Database Model	03/09/2020
• Database Design	03/26/2020
• Table/Queries Finished	04/09/2020
• Implementation/Testing	04/27/2020
• Project Demo	05/04/2020
• Final Report	05/04/2020