Group 10
John O’Brien, Roman Rychkov, Kiera Gill
obriej13@tcnj.edu, rychkor1@tcnj.edu, gillk3@tcnj.edu

Inception: Executive Summary

The existing Trentoniana Room website provides a great service to local historians and those interested in the history of Trenton. Users are able to upload audios, videos, and transcripts for others to view and enjoy. While the site serves its purpose, we believed there are aspects that could be improved on. In particular, we noticed aesthetic inconsistencies in the website, certain redundant features/links, and the lack of organization in the files displayed on the site. We felt that these issues needed to be solved to provide the Trentoniana Room community with truly satisfying experience. To solve these issues, we reconfigured the website to cut down on extraneous pages and consolidate other pages to make the site less cluttered and more straightforward. We also changed the landing page to make the files easier to access, and added options to filter the files for ease of viewing. We used HTML and python to create the front end of the website, and PSQL as the backend database management system. With our site, admins are able to log in with their user ID number and password. From the site, they can update entries, add new ones, and delete existing entries. The benefit of our solution is that the stakeholders will have an easier time using the site. It is much less cluttered than the previous model, and it will be easy for anyone to navigate. This will make users feel comfortable and not overwhelmed, and could potentially recommend it to a colleague or friend. It is possible that we can grow the community this way. The changes that we are proposing also are not a major derivative from the original model, so the existing users will still feel comfortable and will have an easy time adapting to our solution. There is not a major cost associated with our solution. The existing website will have to be taken down, and replaced with ours. Our database also requires hosting and server space to run.

Elaboration: Project Proposal and Specifications

Problem statement

After thoroughly exploring and evaluating the Trentoniana Room website, we have discovered several problems that must be addressed in order to provide the best possible user experience on this site. These problems include poor design choices, unnecessary information, and general inorganization. Each of these is further broken down below:

- ***Problem 1:*** The fonts/color for links on the landing page are inconsistent
    - ***Solution:*** Make consistent, use a red font instead of blue, underline it. Maybe some indication it's the link such as a hover feature that can easily be implemented through modifying the CSS
    - *Edit: In the final solution, we removed all links from the site. They were no longer necessary*
- ***Problem 2:*** The Internet Archive linked in the landing page is unnecessary and redundant
    - ***Solution:*** Simply remove this as a link from the page because you don’t want to confuse the user by providing an abundance of information
- ***Problem 3:*** Landing Page for the Audio Database is a mess, and not user friendly (Oral History)
    - There are files with titles that are just names and numbers, rather than a name.
    - The naming is very inconsistent
    - There are no descriptions about what the file contains
    - There is only a dropdown for sorting by date archived, there should be for others
    - There is no reason to display all of the files randomly on the landing page
    - ***Solution:*** Reorganize the database in a way that makes it much easier for the user to navigate. This will be accomplished by placing each file into more specific groups by content and date.

Objective of the module.

We will address the problems and provide solutions to them to the best of our ability, providing an improved, positive experience to the users of the Trentoniana. The Audio and Visual section will be reformatted to access the appropriate files that are organized, named, and described to offer more context. In order to make this website and database more accessible, we will improve the display of the data, the sorting, and the overall visual presentation of the site. We intend for our system to be attractive to the eye, and easily navigable: data will be displayed so as not to overwhelm the user, and provide them with as much relevant and related information.

Description of the desired end product, and the part you will develop for this class.
Our desired end product is a new and improved The Trentoniana Room website that contains the proper website tools, organized retrieval of audio and visual files found on the site, and providing proper database access for users and creators. We will organize the files into a database with proper access, and rework some issues with the website in general.

Description of the importance and need for the module, and how it addresses the problem.
The current website is very hard to navigate, and there is much to be improved on from the user interface. There are inconsistencies that make it seem unprofessional and unorganized. We will be creating a more user-friendly interface to make the site easier to use. We will be paying close attention to formatting on the site so it looks visually appealing as well.

Plan for how you will research the problem domain and obtain the data needed.
Currently, as a group, we went through the audio and visual portion of The Trentoniana Room, and noted down the issues we saw as new users to the website. After identifying the problems, we brainstormed ways of how the website could improve based on these issues. When we've developed an example product showing the new features implemented to The Trentoniana Room, we plan on presenting our product to other groups, gaining qualitative data on how useful, streamlined, and effective our changes have been. If there are points brought up by other students that will help our design, those changes will be implemented.

Other similar systems/ approaches that exist, and how your module is different or will add to the existing system.
For the video aspect, a platform like YouTube is a great example of how to display videos. We can base our approach for the video's on the style/organization YouTube utilizes. As for the other aspects, academic databases like Ebscohost, and JSTOR, provide a straightforward and visually

appealing approach that makes viewing a variety of content (videos, photos, text entries) efficient and easy. The Schomburg Center is another similar system, in terms of the organization style, though our system will be less cluttered and overwhelming to the user.

Possible other applications of the system(how it could be modified and reused.)

This could be modified and reused to sort and organize other websites, like The Trentonian Room, to help streamline other historical collections available online. The layout can be reformatted to organize any collection of historical audio and visual data.

Performance –specify how and to what extent you will address this

The database performance can be defined as the optimization of resource use to increase throughput and minimize contention, enabling the largest possible workload to be processed. We will address the performance through the workload, throughput, optimization, resources, and contention. The workload is the system commands and transactions directed through the system at any given time, and connected with throughput, the necessary hardware and software would be needed to run the database efficiently. For optimization, the purposeful parts of the database would be created (SQL formulation, database parameters, database organization, etc.) to enable the database optimizer to create the most efficient access paths to the data. Meeting the requirements of contention would also lower throughput, by allowing more than one component of the workload access to a single resource. Our database will cover all of these performance tactics to provide the best usage to the users.

Security –specify how and to what extent you will provide security features.

The security of the database will be implemented by providing only the creators and website owners to access the details of the database. Regular users will be able to access the features but not be able to implement any new features. Authentication would verify a user's credentials if they match the ones stored in the database. New data can only be added through the verification of such accounts.

Edit: We used a "user table" and "super user" roles to grant access to certain people who could insert, delete, and modify the database.

Backup and recovery –specify how and to what extent you will implement this.
Having backup data in case the site is corrupted or there is an issue with the database would be useful. We will implement some method of a backup, but we'll need to decide how often it will be backed up. This will depend on the frequency of how entries there are. It may be necessary to only back it up once a week.
Edit: No backup/recovery method is provided with our final solution

Technologies and database concepts the team will need to learn, and a plan for learning these.
We will need to learn how to implement a database into a website, and be able to manipulate the output to fit how we'd like it to appear on the user interface. We also need to learn how to sort entries, and display them based on user input. Backing up the database is another component we need to learn. In our 315 Class, we hope we'll learn how to create the database and fill it with entries, as well as how to sort them and back them up. We may need to do extra research on implementing it into the website, which we will do by watching tutorials/reaching out to professors for assistance if needed.

Elaboration: Design

Entities: Entry, Audio, Video, Transcript (weak)

Entry (super)
Attributes: EntryType, Length, Description, Category, DateUpload, FileSize

Audio (sub)
Attributes: EntryType (foreign key), AudioName, DateCreated

Video (sub)
Attributes: EntryType (foreign key), VideoName, DateCreated

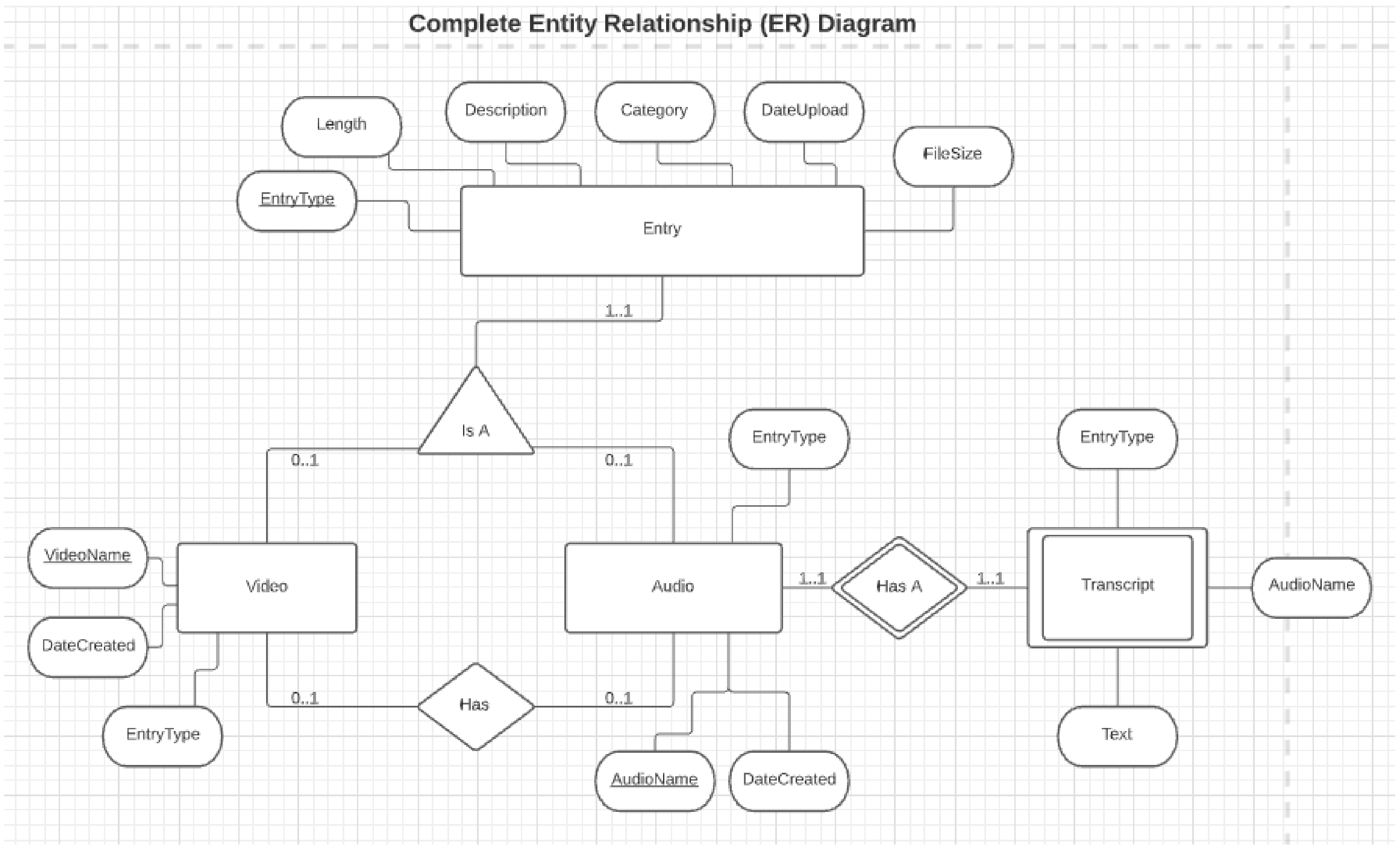
Transcript (Weak Entity)
Attributes: EntryType (foreign Key), Text, AudioName

Relationships:

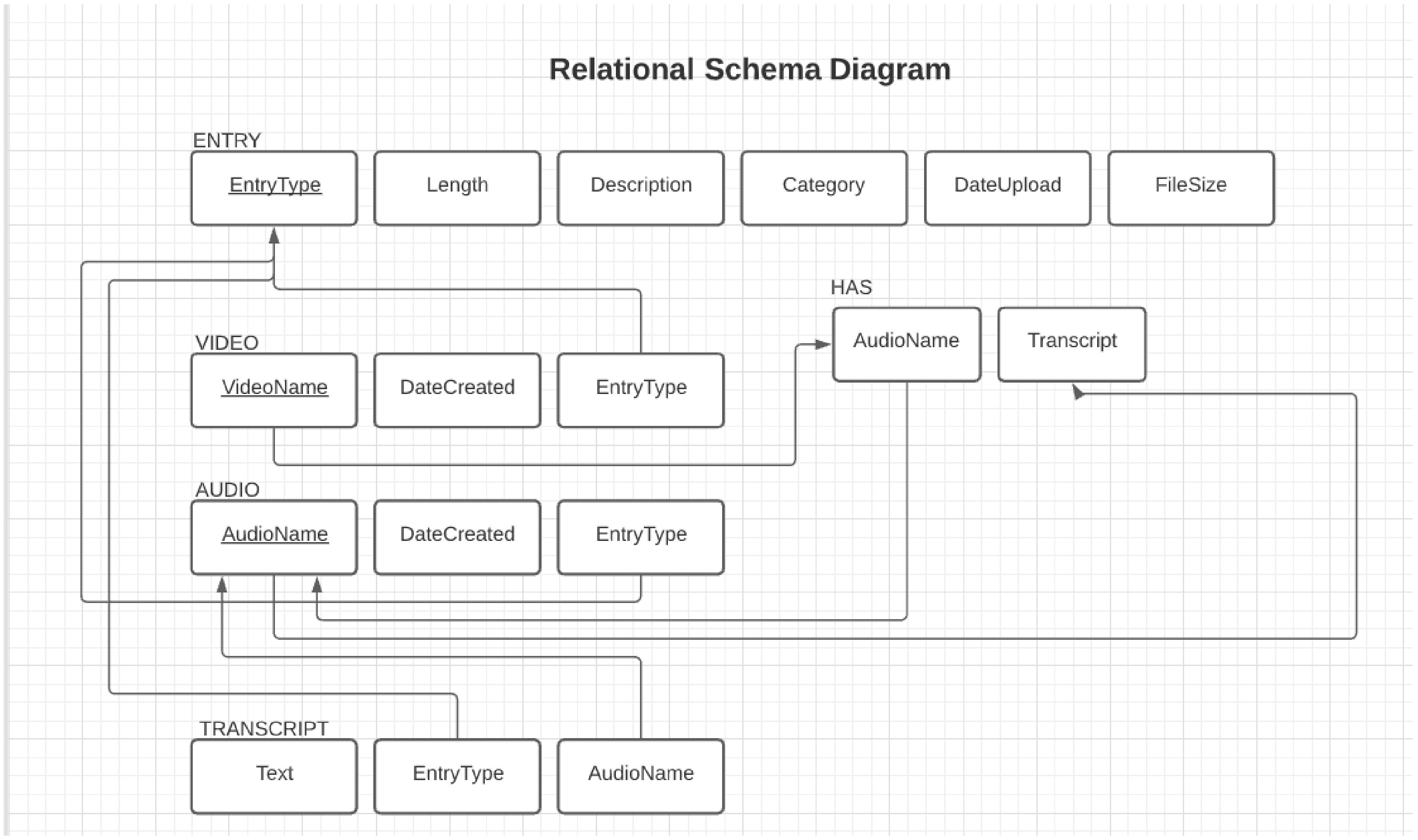
Audio IS A File	(generalization)
Video IS A File	(generalization)
Audio HAS A Transcript	(aggregation)
Video HAS Audio	(aggregation)

Multiplicities:
Video Is An Entry -> 1..1
Audio Is An Entry -> 1..1
Entry Is A Video -> 0..1
Entry Is An Audio -> 0..1
Audio Has A Transcript -> 1..1
Transcript Has An Audio -> 1..1
Video Has Audio -> 0..1
Audio Has Video -> 0..1

ER Model:


Complete Entity Relationship (ER) Diagram
Length
Description
Category
DateUpload
FileSize
EntryType
Entry
1..1
Is A
0..1
0..1
EntryType
EntryType
VideoName
Video
Audio
1..1
Has A
1..1
Transcript
AudioName
DateCreated
EntryType
0..1
Has
0..1
AudioName
DateCreated
Text

Relational Schema:



Normalizing relations to BCNF

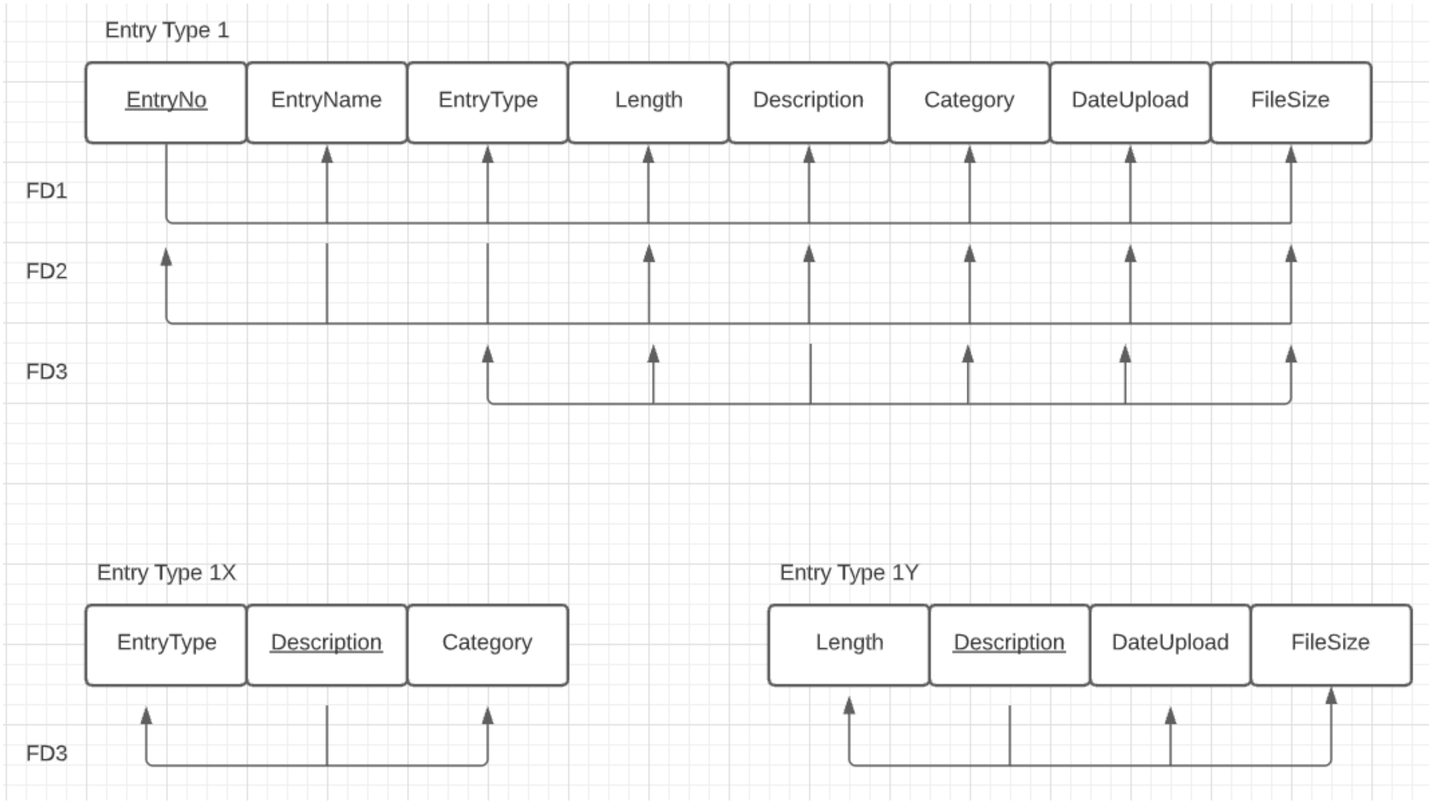
1.

<u>EntryType</u>	Length	Description	Category	DateUpload	FileSize

a. This entry is not in BCNF. We have modified the table entry to put it in BCNF, as shown below.

<u>EntryNo</u>	EntryName	EntryType	Length	Description	Category	DateUpload	FileSize

Fixed below



We normalized the Entry Type table by breaking down the individual functional dependencies of the attributes. FD1 shows how EntryNo points to the access of all other attributes of the entry. FD2 shows how EntryName and EntryType points to the access of all other attributes of the entry as well. Finally, in FD3, the Description points to the EntryType, Length, Category, DateUpload, and FileSize, which will all be somewhere in the Description. We decided to break down this Functional dependency into two groups in order to normalize with a specific category in mind. The Entry Type 1X list of attributes will be the "letters" descriptions, where the descriptions have words containing each attribute. The Entry Type 1Y has attributes that will be “number” descriptions, where the descriptions have numbers containing each attribute. By decomposing, we were able to achieve 3NF normalization of our relational diagram and make it functionally determine other non-key attributes.

Similarly, we believe that video, audio, and transcript are already in the BCNF form because they cannot be further broken down. All three of them rely on the Entry Type to determine other distinguishing features. This is because each different entry type has the same additional attributes, including EntryName, length, description, category, DateUploaded and FileSize. Essentially, the thing that distinguishes between all of the sources in the database is the entry type.

Defining different views

1. *Views and roles*
    a. <u>User</u>
        i. A user can search for entries and access entries in the database
    b. <u>Admin / Super User</u>
        i. An admin can post, search, delete, and modify the entries
2. *Transactions that the views will have*
    a. Admin only transactions:
        i. Upload Entry
        ii. Delete Entry
        iii. Modify Entry
        iv. **Virtual Tables Required**:
            1. entry(EntryNo, EntryName, EntryType, DateUpload, Description), audio(AudioName), video(VideoName), transcript(Text)
    b. General user transactions:
        i. Pause/Play the Audio
        ii. Pause/Play the Video
        iii. Read Transcript
        iv. Search Entry by Name
        v. Search Entry by Date
        vi. Search Entry by Type
        vii. Search by Author
        viii. Filter by Date
        ix. Filter by File Size
        x. Virtual Tables Required:
            1. entry(Name, DateUpload, Author, Description, Length)

Designing SQL queries to satisfy transaction requirements

CREATE TABLE entries (entryNo SERIAL PRIMARY KEY, entryName text, length interval, description text, category text, dateUpload date, fileSize varchar)

1. *PSQL commands for an admin/superuser*
    a. INSERT INTO entries VALUES (‘Big Speech’, ‘00:02:15’, ‘A big speech given by an important person’, ‘Audio’, ‘12-04-1886’, ‘125 MB’);
        i. Inserting a 125mb file called “Big Speech” into the entry database.
    b. DELETE FROM entries WHERE (entryNo = ‘35’)
        i. Deleting an entry whose entry number is ‘35’.
    c. UPDATE entries SET dateUpload = ‘1-16-1948’ WHERE entryNo = ‘35’
        i. Modifying an entry whose number is ‘35’.
2. *PSQL commands for general user*
    a. SELECT audioFile FROM Audio WHERE audioTitle = title;
        i. Retrieving an audio file to be played.
    b. SELECT videoFile FROM Video WHERE videoTitle = title;
        i. Retrieving a video file to be played.
    c. SELECT text FROM Transcript WHERE audioName = title;
        i. Retrieving a transcript to be read.
    d. CREATE VIEW searchName AS SELECT * FROM entries WHERE entryName = name;
        i. Creating a view for the user, who searched for the entry by name.
    e. CREATE VIEW searchDate AS SELECT * FROM entries WHERE uploadDate = date;
        i. Creating a view for the user, who searched by date.
    f. CREATE VIEW searchType AS SELECT * FROM entries WHERE fileType = type;
        i. Creating a view for the user, who searched by file type.
    g. CREATE VIEW searchAuthor AS SELECT * FROM entries WHERE fileAuthor = author;
        i. Creating a view for the user, who searched by author.
    h. SELECT dateUpload FROM **the current view** ORDER BY dateUpload ASC;
        i. Displaying the results by date order.
    i. SELECT fileSize FROM **the current view** ORDER BY fileSize ASC;
        i. Displaying the results by file size.

Construction: Tables, Queries, UI

Create 2 tables containing all the information we need, one for user and one for entry

User: UserNo, firstName, lastName, access
Entry: EntryNo, entryType, fileName, date, link

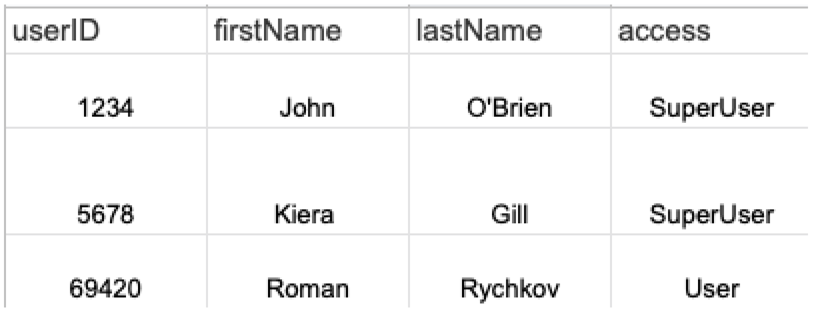
DDL Operations: (Data Definition Language)

```
CREATE TABLE entries (
        entryNo Serial PRIMARY KEY,
        entryType varchar (60)
        entryName varchar (60)
        entryLink varchar (60)
        Date DATE
);

CREATE TABLE users (
        userID serial PRIMARY KEY,
        firstName varchar (30),
        lastName varchar (30),
        edit: password varchar(30),
        access varchar (30)
);
```

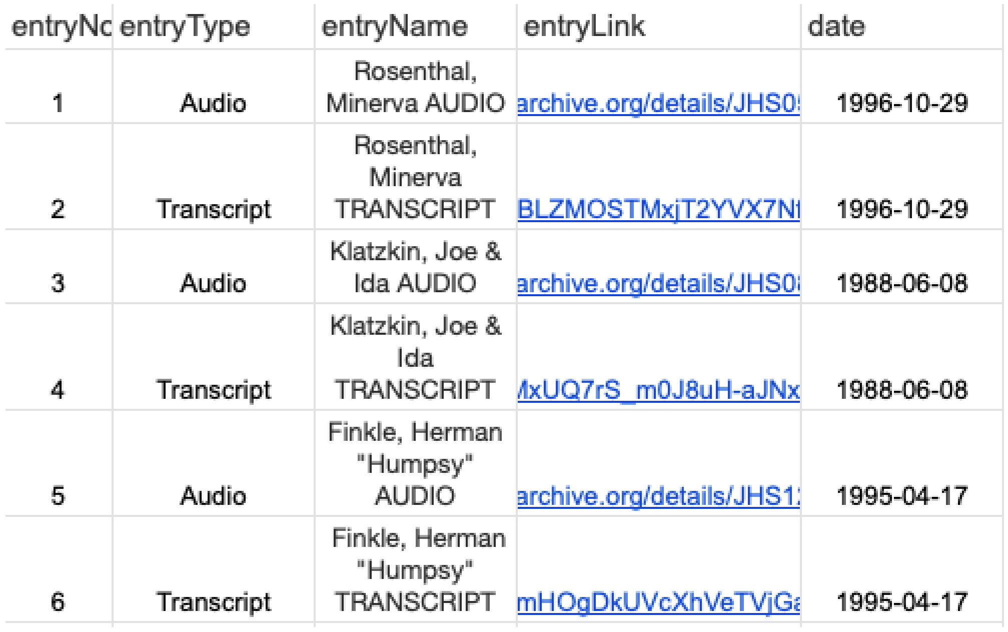
Create a spreadsheet with the information from the website (3 users, 3 audio files, 3 transcripts) - writing the CSV files into TABLE users & TABLE entries:

```
COPY users(
        userID,
        firstName,
        lastName,
        edit: password,
        access)
FROM '/home/lion/UserInfo.csv'
DELIMITER ',' CSV HEADER;
```



userID	firstName	lastName	access
1234	John	O'Brien	SuperUser
5678	Kiera	Gill	SuperUser
69420	Roman	Rychkov	User

```
COPY entries(
        entryNo,
        entryType,
        entryName,
        entryLink,
        date)
FROM '/home/lion/EntryInfo.csv'
DELIMITER ','CSV HEADER;
```



entryNc	entryType	entryName	entryLink	date
1	Audio	Rosenthal, Minerva AUDIO	archive.org/details/JHS0	1996-10-29
2	Transcript	Rosenthal, Minerva TRANSCRIPT	BLZMOSTMxjT2YVX7N	1996-10-29
3	Audio	Klatzkin, Joe & Ida AUDIO	archive.org/details/JHS0	1988-06-08
4	Transcript	Klatzkin, Joe & Ida TRANSCRIPT	xUQ7rS_m0J8uH-aJNx	1988-06-08
5	Audio	Finkle, Herman "Humpsy" AUDIO	archive.org/details/JHS1	1995-04-17
6	Transcript	Finkle, Herman "Humpsy" TRANSCRIPT	mHOgDkUVcXhVeTVjG	1995-04-17

DML Queries on entries: (Data Manipulation Language)

Insert a new file into the system

INSERT INTO entries VALUES (‘*entryName*’, ‘*entryType*’, ‘*entryLink*’, ‘*date*’);

Delete a file from the system

DELETE FROM entries WHERE (entryNo = ‘1”)

Update a file from system

UPDATE entries SET dateUpload = ‘*example dateUpload*’ WHERE entryNo = ‘*example entryNo*’

Regular Queries on entries

Retrieve an entry based on entryName

SELECT * FROM entries WHERE fileName = '*example name*'

Retrieve an entry based on date

SELECT * FROM entries WHERE date = '*example date*';

Display all audio files

SELECT * FROM entries WHERE entryType = 'Audio';

Display all transcripts

SELECT * FROM Transcript;

Search through results by date

SELECT * FROM entries ORDER BY date ASC;

DML Queries on Users: (Data Manipulation Language)

Add a new user to the database

INSERT INTO users (firstname, lastname, password, access)
VALUES (''Sorca', 'MC', '1234', 'user');

Remove a user

DELETE FROM users WHERE userID = '3';

Change the user's status

UPDATE users SET access = admin WHERE userID = '4'

Change the user's name

UPDATE users SET lastName = 'Rychkova' WHERE lastName = 'Gill'

Regular Queries on Users:

Search for a userID by name

SELECT userID FROM users WHERE lastName = '*lastName*' AND firstName =
'firstName';

Search for a access and userID by name

```
SELECT access, userID FROM users WHERE lastName = ‘*lastName*’ AND
firstName = ‘*firstName*’;
```

PSQL CODE

```
createdb proj7

psql proj7

CREATE TABLE users (userID serial PRIMARY KEY, firstName varchar (30), lastName
varchar (30), password varchar(30), access varchar (30));

COPY users(userID,firstName, lastName, password, access)FROM
'/home/lion/app/UserInfo.csv' DELIMITER ',' CSV HEADER;

SELECT * FROM users;

CREATE TABLE entries (entryNo Serial PRIMARY KEY, entryType varchar (60), entryName
varchar, entryLink varchar, date DATE);

COPY entries(entryNo,entryType, entryName, entryLink, date)FROM
'/home/lion/app/EntryInfo.csv' DELIMITER ','CSV HEADER;

SELECT * FROM entries;

DELETE FROM entries WHERE (entryNo = 1);

SELECT * FROM entries;

UPDATE users
SET lastname = 'Rychkova'
WHERE lastname = 'Gill';

SELECT * FROM users;

INSERT INTO users
(firstname, lastname, access)
VALUES ('Emily', 'Carrot', ‘EmilyC’,'user');
```

```
SELECT * FROM users;

DELETE FROM users
WHERE userID = 1;
SELECT * FROM users
WHERE lastname = 'Rychkov';

SELECT * FROM users
WHERE firstname = 'Roman';

UPDATE users
SET access = ‘superUser’
WHERE userID = ‘69420’;

SELECT * FROM entries
WHERE entryType = 'Transcript';


SELECT userID
FROM users
WHERE firstName = 'Roman';

SELECT access
FROM users
WHERE firstName = 'Roman';

SELECT access, userID
FROM users
WHERE (firstName = 'Roman'
AND lastName = 'Rychkov');

SELECT * FROM entries ORDER BY date ASC;

\s work.txt
```