Group 11: TAP++ Dev Team

TAP++

(Trentoniana Archive Project++)

Alexander Benasutti benasua1@tcnj.edu

Alexander Viola violaa1@tcnj.edu

Sam Haseloff haselos1@tcnj.edu

Raymond Chow chowr2@tcnj.edu

Inception: “Executive Summary”

The Trentoniana archive of audio recordings is disorganized, presenting an overwhelming amount of information to the average user that they will have no understanding of how to navigate. Many audio files are incorrectly tagged as well, likely due to the difficulty in using the archive.org website, which is designed to be a very general solution to many problems. In addition, transcripts are not currently available to the general public after having been removed from the main Trentoniana website, so it is difficult to listen and view both at the same time. In addition, some of the audio files are very difficult to understand, so transcripts are necessary to provide accessibility to the user.

We have developed an easy-to-use user interface called Trentoniana Archive Project++ (TAP++), which allows users to simultaneously listen to audio files and view a transcript of the associated audio file. As the transcripts are generally not perfect, the TAP++ application has features that allow for crowdsourced transcript revisions, as well as administrative functionality to prevent abuse of these crowdsourced features. These features allow for a better user experience, should they want to listen to oral histories from the Trentoniana project.

Compared to the status quo Trentoniana setup that relies heavily on links to external sites -- Google Drive for transcripts, archive.org for audio files -- our implementation provides unified hosting of both of these resources. With the transcripts and the audio in the same location, we can provide useful features like transcript edit requests, transcript edit histories, and robust search and tagging functionality. This allows for greater community engagement and thus the potential for higher quality transcripts; without these faculties in place, it is difficult for an interested member of the community to access audio files and transcripts to begin with, much less offer revisions and feedback on them.

TAP++ is a separate new website with its own authentication system, so it would not make sense to integrate this into the existing Trentoniana content management system (WordPress). In addition, the project is not ready for production use, as it is missing features like proper audit logging viewing and the ability to add tags to an audio file. Adding these features would require development time by someone who is familiar with modern web development technologies. However, after development is complete, it is safe to say that the project can be run on a $5/month server or even an on-premise Raspberry Pi. Renting a server would provide the network infrastructure required, but using an on-premise server would require a dedicated “business” network line to be provisioned and maintained in order to expose the website to the Internet.

Elaboration: Project Proposal and Specifications

Problem statement.

The Trentoniana archive of audio recordings is disorganized, presenting an overwhelming amount of information to the average user that they will have no understanding how to navigate. Limited or incorrect filtering options (two of the five provided filters are “New Jersey”), limited synopses of the recordings, and general information overload are some of the issues with the current way they are presented. In addition, with both audio recordings and their respective transcripts being located on completely different websites than Trentoniana’s, it is difficult to listen and view both at the same time for comparison and transcript revision purposes. The creation of an all-in-one interface for the recordings and transcripts on the Trentoniana website would be preferable. This would be an interface that allows a user to freely navigate between oral histories by applying meaningful parameters (such as filtering by topic or date) to accurately find what they are looking for.

Objective of the module.

The objective of our module is to provide an easy-to-use interface accessible to users who may not have the highest proficiency in technology-usage to view and filter through Trenton residents’ oral histories and their transcripts simultaneously. In other words, our module’s objective is to create an alternate, improved version of the archive.org website for the Trentoniana audio collections.

Description of the desired end product, and the part you will develop for this class.

Our team's development goal is to group Trenton residents' oral histories (audio files) and their transcripts together onto the same webpage for easy comprehension of said audio. We will develop a database using PostgreSQL that allows the average user to easily filter these audio files based on parameters they provide (e.g. date, location, topic, date added, etc.), in addition to a clean interface that allows a user hear a recording and see its respective transcript at the same time. In terms of user access, privileged users/administrators should have the ability to edit and revise transcripts at will for better workflow. If they find an error, unprivileged users should also be able to suggest edits by commenting on transcripts and have those edits be officially added upon administrator approval.

Description of the importance and need for the module, and how it addresses the problem.

The Trentoniana Collection aims to preserve a local history pertaining to the lives of Trenton, New Jersey residents. Their pursuit aims to enrich the quality of life in the City and pass on the stories of its people's heritage and diversity. At its current state, the archive leaves much room for improvement in its organization and its ease-of-access to the public. All current oral histories of Trenton residents are located on a third-party website that presents the average user an overwhelming amount of data that they will have no idea how to traverse. Our module aims to address these ambiguities with the current system by presenting an easily accessible interface for Trenton residents' oral histories that is sortable through a variety of different filters pertaining to user specifications. For example, the user should be able to filter

oral histories by their date and topic of conversation. In addition, a transcript will have the option to be provided with each audio file to keep resident accounts understandable.

Plan for how you will research the problem domain and obtain the data needed.

- Download audio files from archive.org
- Collect transcripts from associated data storage

Other similar systems / approaches that exist, and how your module is different or will add to the existing system.

Phonemica (https://phonemica.net/) is an audio collection of Chinese topolects and dialects for archival and preservation purposes. This is not open source and is highly specialized. Instead, we will build a more generic system.

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

Our database has the potential to also be used for other general-purpose audio database applications, such as a database that holds audio for songs (instead of Trenton resident oral histories) and their lyrics (transcripts). A variety of filters and parameters should be able to be specified by the user in order to find the song they're looking for. For example, if they only know the year and the artist of a song, they should be able to search for the song using these filters.

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

- Writing good database queries
- Using database indexes intelligently
- Hosting audio files and transcripts directly on our own server

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

Security features of our module include the prevention of unauthorized editing of transcripts or any other data, in addition to providing administrative roles for authorized users (i.e. users who are allowed to edit the database).

Backup and Recovery – specify how and to what extent you will implement this.

- Nightly dumps of audio data to an S3 server
- Nightly incremental backups of the database

Technologies

- Programming Language (e.g. Rust, PHP, C#, etc.)
- Backend Web Framework (e.g. warp/thruster, ASP.NET Core)
- Frontend Web Frameworks (e.g. React/Vue/vanilla JavaScript)

Database Concepts

- PostgreSQL/Relational Databases
- Database modelling and design
- Database security
- SQL queries
- UML in data modelling

A diagrammatic representation of the system boundary that specifies what data you will model and which queries you will implement.

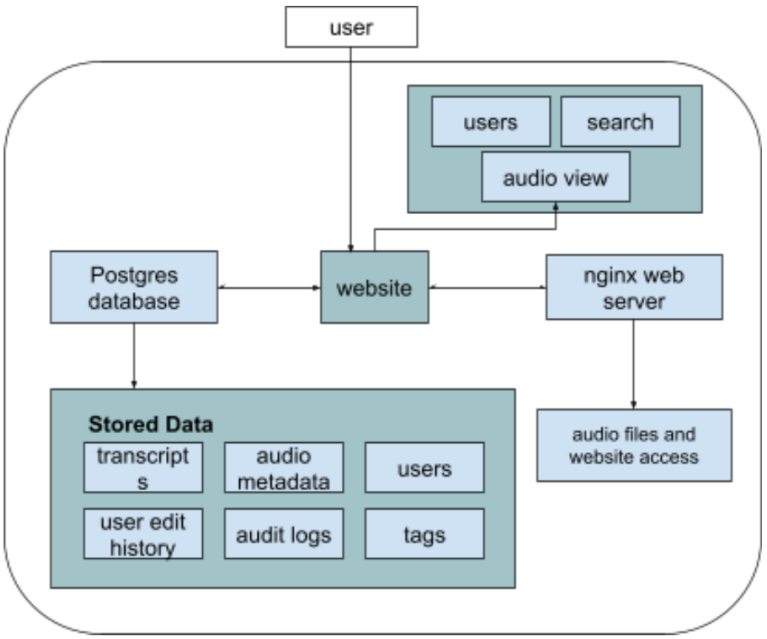

Figure 1. System Boundary Diagram

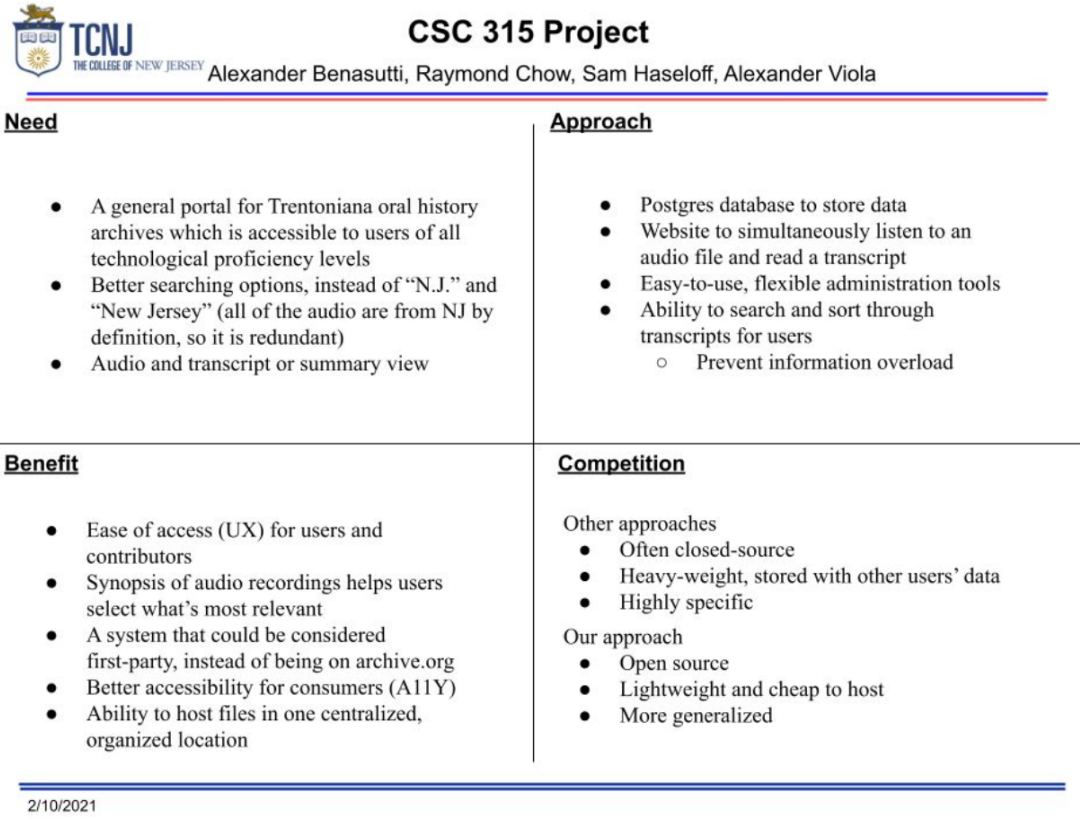

Figure 2. 1-Page Quad Chart

Elaboration: Design

ER Diagram

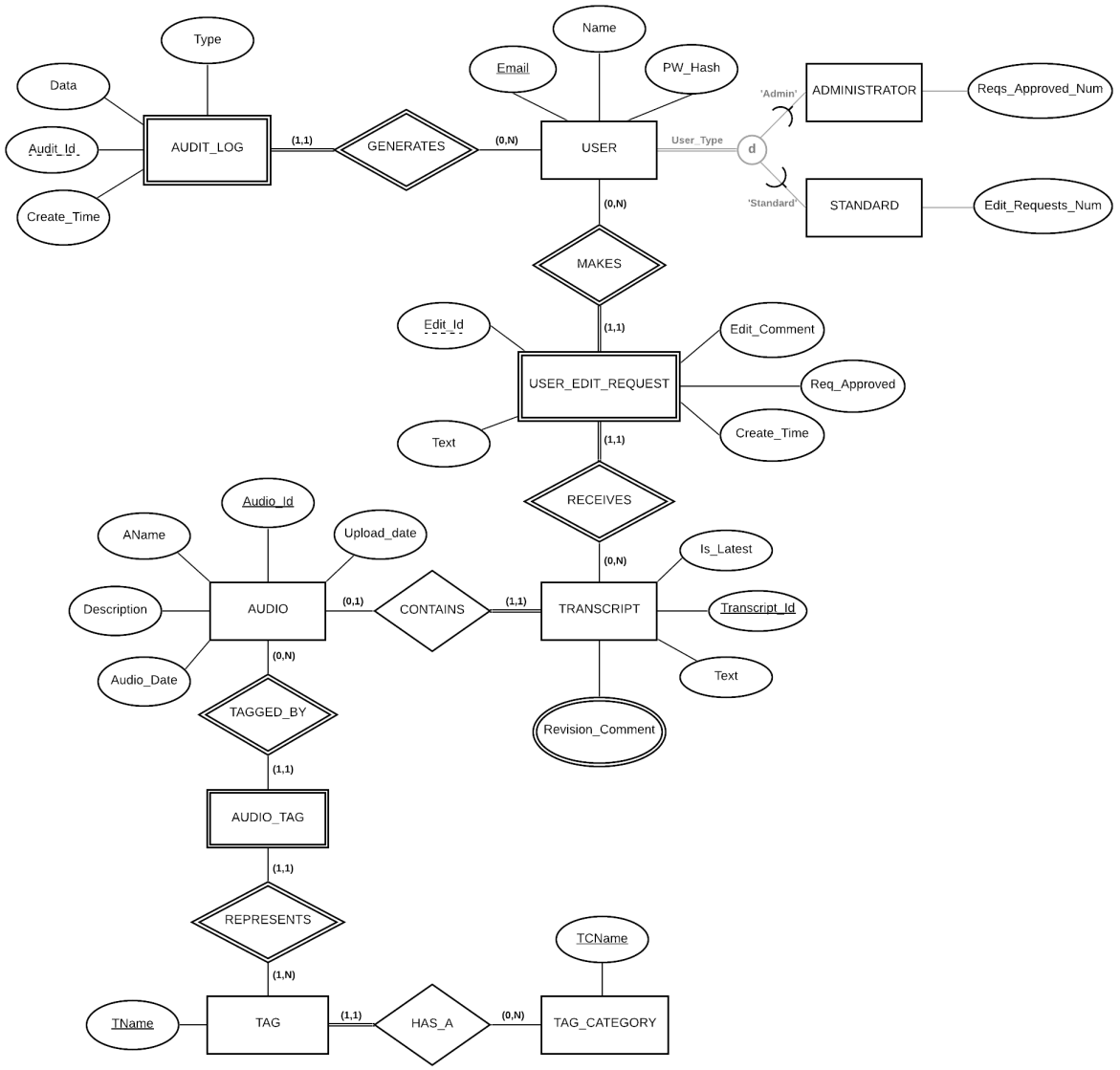

Figure 3. ER Diagram

Relational Schema

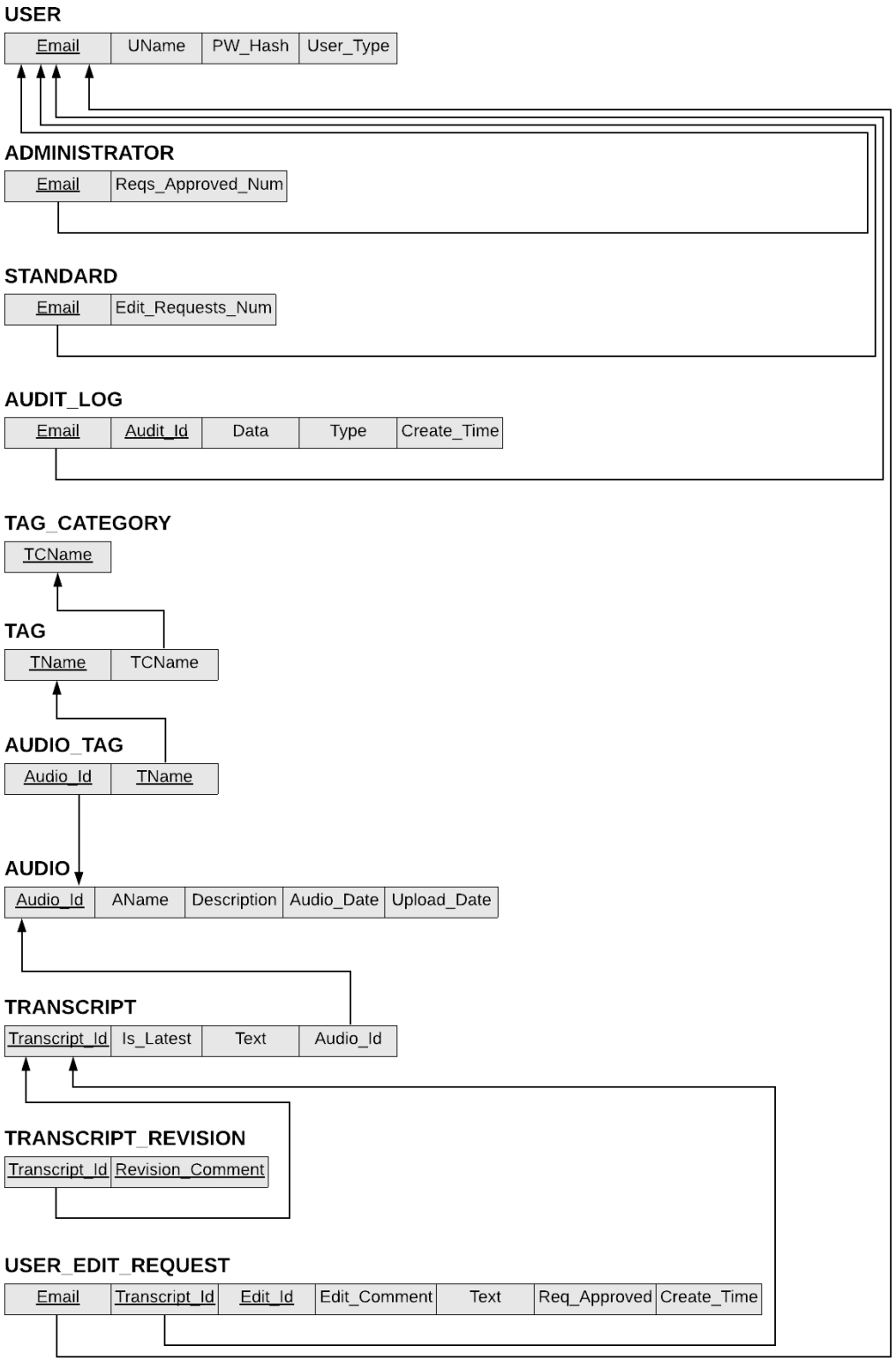

Figure 4. Relational Schema

Other

Database Estimations

- Initial Database Size
    - At the current moment, there will be at most 74 records (for the 74 audio files currently loaded on https://archive.org/details/trentoniana).
- Types and Average Number of Searches
    - 74 metadata entries (audio files)
    - On average 3 tags per metadata entry
    - On average 5 tag categories
    - On average 4 tags per category
    - On average ½ of metadata entries will have a transcript (74/2 = 37)
    - $(4 * 5) + (3 * 74) + 74 + 37 = 353\ searches$

Tables

Users:

- int id: primary key (identity)
- varchar name
- varchar email (unique)
- varchar password_hash
- int user_type = 0 (user)

Audio:

- int id: primary key (identity)
- varchar name

- (long string type) description (nullable)
- date audio_date (nullable)
- date upload_date

Transcripts:

- int id: primary key (identity)
- int audio_id (Audio: id)
- (long text) text
- text revision_comment (nullable)
- bool is_latest = true

TagCategory:

- int id: primary key (identity)
- varchar name (unique)

Tags:

- int id: primary key (identity)
- int category_id (TagCategory: id)
- varchar name (unique)

AudioTag (+ unique key with both columns):

- int id: primary key (identity) **(remove?)**
- int audio_id (Audio: id)
- int tag_id (Tags: id)

UserEditRequest:

- int id: primary key (identity)
- int user_id (Users: id)
- (long text) text
- int transcript_id (Transcripts: id)

- varchar edit_comment (nullable)
- bool request_approved = false
- timestamp create_time

AuditLogs:

- int id: primary key (identity)
- int user_id (Users: id)
- varchar type (type of log: e.g. approval of edit request, ban user(?), etc.)
- jsonb data
- timestamp create_time

Entity Types

Strong:

- User, Audio, Transcript, TagCategory, Tags

Weak:

- AudioTag, UserEditRequest, AuditLogs

Relationships

Transcripts → Audio (audio_id)

- 1:1
    - one transcript can have at most one audio file associated
    - an audio file can have at most one transcript associated with it

Tags → TagsCategory (category_id)

- N:1
    - a tag category can apply to any number of tags
    - a tag can have at most one tag category

UserEditRequest → User (user_id)

- N:1
    - a user can make any number of user edit requests
    - a user edit request is only made by a single user

UserEditRequest → Transcript (transcript_id)

- N:1
    - a transcript can have any number of user edit requests
    - a user edit request can only be applied to one transcript at a time

AudioTag → Audio (audio_id)

- 1:N
    - a single audio file can have many audio tags
    - an audio file tag is unique to the audio file associated with it

AudioTag → Tags (tag_id)

- 1:N
    - an audio tag has a single tag associated with it
    - a tag can be applied to many different audio file tags

AuditLogs → User (user_id)

- 1:N
    - a user could have any number of audit logs associated with them
    - an audit log is usually associated with a single user

Boyce-Codd Normal Form (BCNF)

Normal Form Requirements:

- 1NF
    - Single valued attributes
- 2NF
    - 1NF
    - No partial dependency
        - If the primary key returns (duplicated) information that could be in another relation
- 3NF
    - 2NF
    - No transitive dependency
- BCNF
    - 3NF
    - For every dependency $A \rightarrow B$, A is a superkey

Every table satisfies the above requirements. Therefore, all of the tables in the schema are in BCNF.

Users:

- int id: primary key (identity)
- varchar name
- varchar email (unique)
- varchar password_hash
- int user_type = 0 (user)

There are no multi-valued attributes (1NF), there are no partial dependencies (2NF), there are no transitive dependencies (3NF), and A (id) is a superkey for every non-prime attribute (BCNF). Here, the *email* column is not used as a prime attribute; the unique constraint is only for relational integrity.

Audio:

- int id: primary key (identity)
- varchar name
- (long string type) description (nullable)
- date audio_date (nullable)
- date upload_date

There are no multi-valued attributes (1NF), there are no partial dependencies (2NF), there are no transitive dependencies (3NF), and A (id) is a superkey for every non-prime attribute (BCNF).

Transcripts:

- int id: primary key (identity)
- int audio_id (Audio: id)
- (long text) text
- text revision_comment (nullable)
- bool is_latest = true

There are no multi-valued attributes (1NF), there are no partial dependencies (2NF), there are no transitive dependencies (3NF), and A (id) is a superkey for every non-prime attribute (BCNF). While *audio_id* references another table, it does not act as part of the primary key as the primary key is the identity column *id*.

TagCategory:

- int id: primary key (identity)

- varchar name (unique)

There are no multi-valued attributes (1NF), there are no partial dependencies (2NF), there are no transitive dependencies (3NF), and A (id) is a superkey for every non-prime attribute (BCNF). The column *name* is unique for integrity, but mostly visual integrity. For example, there shouldn't be two categories named "Location" showing up at the search sidebar/panel.

Tags:

- int id: primary key (identity)
- int category_id (TagCategory: id)
- varchar name

There are no multi-valued attributes (1NF), there are no partial dependencies (2NF), there are no transitive dependencies (3NF), and A (id) is a superkey for every non-prime attribute (BCNF). The column *category_id* is not part of a composite key; our primary key is the *id* identity column. The column *name* is unique for integrity purposes, but mostly visual integrity.

AudioTag (+ unique constraint with both columns):

- int audio_id (Audio: id)
- int tag_id (Tags: id)

There are no multi-valued attributes (1NF), there are no partial dependencies (2NF), there are no transitive dependencies (3NF), and there are no non-prime attributes (BCNF).

UserEditRequest:

- int id: primary key (identity)
- int user_id (Users: id)
- int transcript_id (Transcripts: id)
- (long text) text
- varchar edit_comment (nullable)

- bool request_approved = false
- timestamp create_time

There are no multi-valued attributes (1NF), there are no partial dependencies (2NF), there are no transitive dependencies (3NF), and A (id) is a superkey for every non-prime attribute (BCNF). There can be multiple edit requests by the same user for a single transcript. Therefore, each request is uniquely identified by the *id* identity column and not a composite key of *user_id* + *transcript_id*.

AuditLogs:

- int id: primary key (identity)
- int user_id (Users: id)
- varchar type (type of log: e.g. approval of edit request, ban user(?), etc.)
- jsonb data
- timestamp create_time

There are no multi-valued attributes (1NF), there are no partial dependencies (2NF), there are no transitive dependencies (3NF), and A (id) is a superkey for every non-prime attribute (BCNF). While *user_id* references another table, it does not act as part of the primary key as the primary key is the identity column *id*. The *data* column can contain arbitrary data, though it will only ever contain single objects.

Required Views/Virtual Tables

Transaction Requirements = What need to be the possible user actions that query the database?

- User needs to be able to select filters and generate audio recording results
- Users need to be able to create and submit an edit request to a transcript

- Administrators need to be able to view transcript edit requests and approve or deny them

Data Requirements = Data involved in/queried upon user action

- Filter Listing
    - Tag id
    - Tag category_id
    - Tag name

Example Query: A user goes to select certain filters within the Trentoniana oral history archives in order to properly search for what audio file/transcript they are looking for. This view will return the name of the tags with their categories for use by the server.

- Results
    - Audio id
    - Audio name
    - Audio upload_date
    - Transcript text AS transcript_text

Example Query: A user makes their selection of filters on the Trentoniana oral history archives and queries the server database for what results these apply to. This view will return the names of the audio files and their upload date, in addition to their associated transcripts if available.

- Pending User Submitted Transcript Edit Requests
    - Audio name
    - Transcript id
    - Transcript text AS old_text
    - User_Edit_Request text AS new_text
    - User_Edit_Request user_id

    - User_Edit_Request id AS edit_id
    - User_Edit_Request edit_comment
    - User_Edit_Request create_time

Example Query: A user sees some typo or incorrect information within an audio file's associated transcript. In an effort to make a correction, they open a transcript edit request. This view will return the name of the audio file whose transcript is being edited, in addition to the associated text which is being edited. The user can also add an edit comment to leave a reason for their edit.

- Administrator Transcript Edit Request Approval
    - Audio name
    - Transcript id
    - User_Edit_Request edit_comment
    - Transcript text AS old_text
    - User_Edit_Request text AS new_text
    - User_Edit_Request id AS edit_id
    - User_Edit_Request edit_comment
    - User_Edit_Request request_approved AS approval_status

Example Query: An administrator sees that there is an edit request to a transcript which is awaiting approval. This view will return the transcript being edited in addition to the edited text of said transcript. They can view the edit comment left by the user, and also approve or deny the edit request, leaving a revision comment if they choose to approve it.

SQL Queries for Views

- **Filter Listing:**

  SELECT tags.id AS tag_id, tag_category.name AS category_name, tags.name AS tag_name FROM tags JOIN tag_categoryON tag_category.id = tags.category_id;

- **Transcript Requests:**

```
SELECT
        user_edit_request.id AS request_id,
        user_edit_request.user_id,
        transcripts.id AS transcript_id,
        audio.name AS audio_name,
        transcripts.text AS old_text,
        user_edit_request.text AS new_text,
        user_edit_request.edit_comment AS edit_comment,
        user_edit_request.create_time AS request_time
FROM user_edit_request
INNER JOIN transcripts ON user_edit_request.transcript_id = transcripts.id
INNER JOIN audio ON audio.id = transcripts.audio_id
WHERE user_edit_request.request_approved IS NULL;
```

- **Admin Transcript Request View:**

```
SELECT
        user_edit_request.id AS request_id,
        user_edit_request.user_id,
        transcripts.id AS transcript_id,
        audio.name AS audio_name,
        transcripts.text AS old_text,
        user_edit_request.text AS new_text,
        user_edit_request.edit_comment AS edit_comment,
```

```
        user_edit_request.create_time AS request_time,
        user_edit_request.request_approved AS approval_status
    FROM user_edit_request
    INNER JOIN transcripts ON user_edit_request.transcript_id = transcripts.id
    INNER JOIN audio ON audio.id = transcripts.audio_id
    WHERE user_edit_request.request_approved IS NULL;
```

- **Results:**

```
    SELECT audio.id, audio.name, audio.upload_date, transcripts.text AS transcript_text
    FROM audio
    JOIN transcripts ON transcripts.audio_id = audio.id
    WHERE transcripts.is_latest = TRUE;
```

Construction: Tables, Queries, and User Interface

All documents and submissions for Stage Va-Vb of this project can be found on the /db folder of our GitHub repository:

https://github.com/tcnj-violaa/tap-plus-plus/tree/main/db

Transition: Maintenance

All source code is appropriately documented and uploaded onto our GitHub repository:

https://github.com/tcnj-violaa/tap-plus-plus

Transition: Product Hand Over

The project has been forked and made public here:

https://github.com/tcnj-violaa/tap-plus-plus