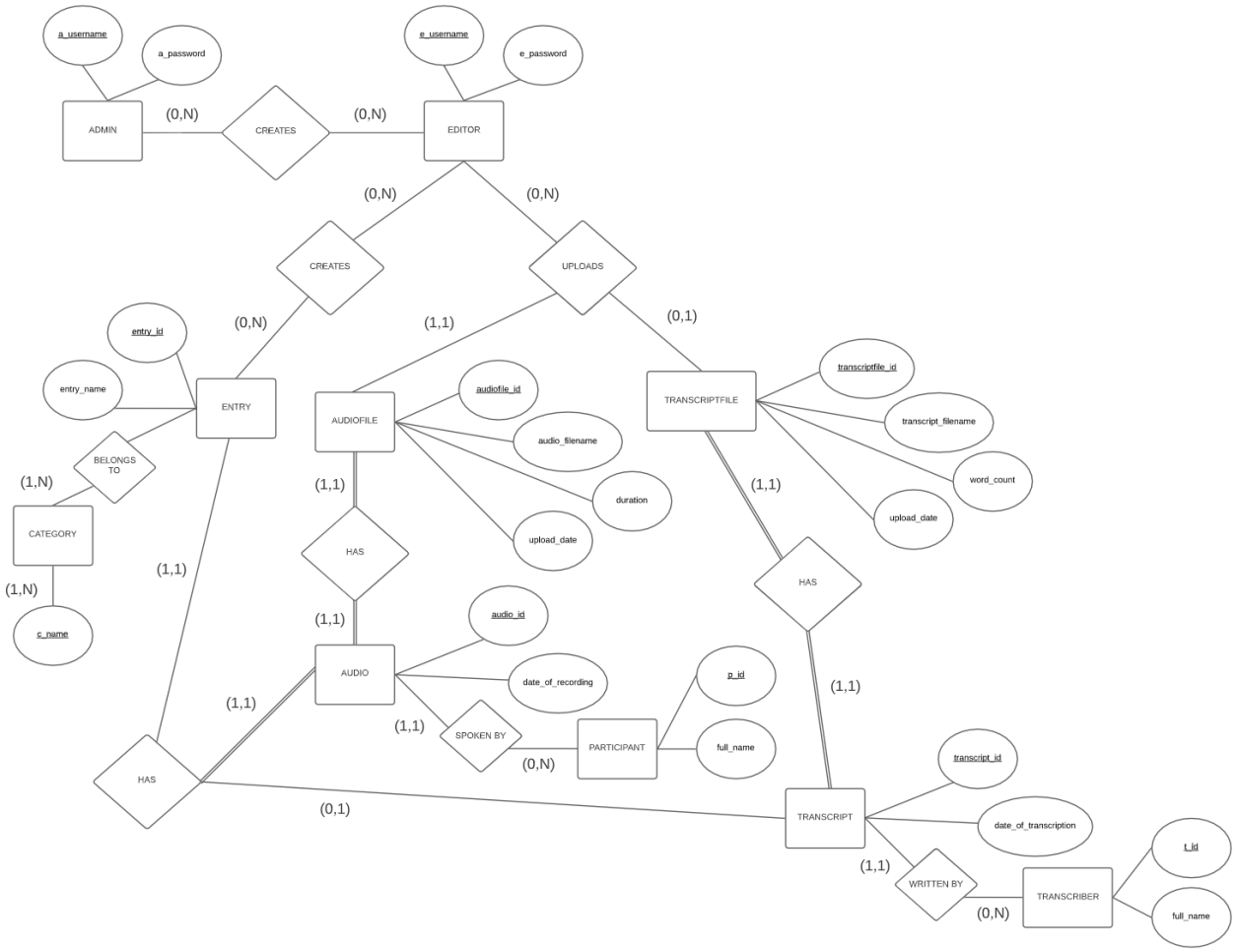
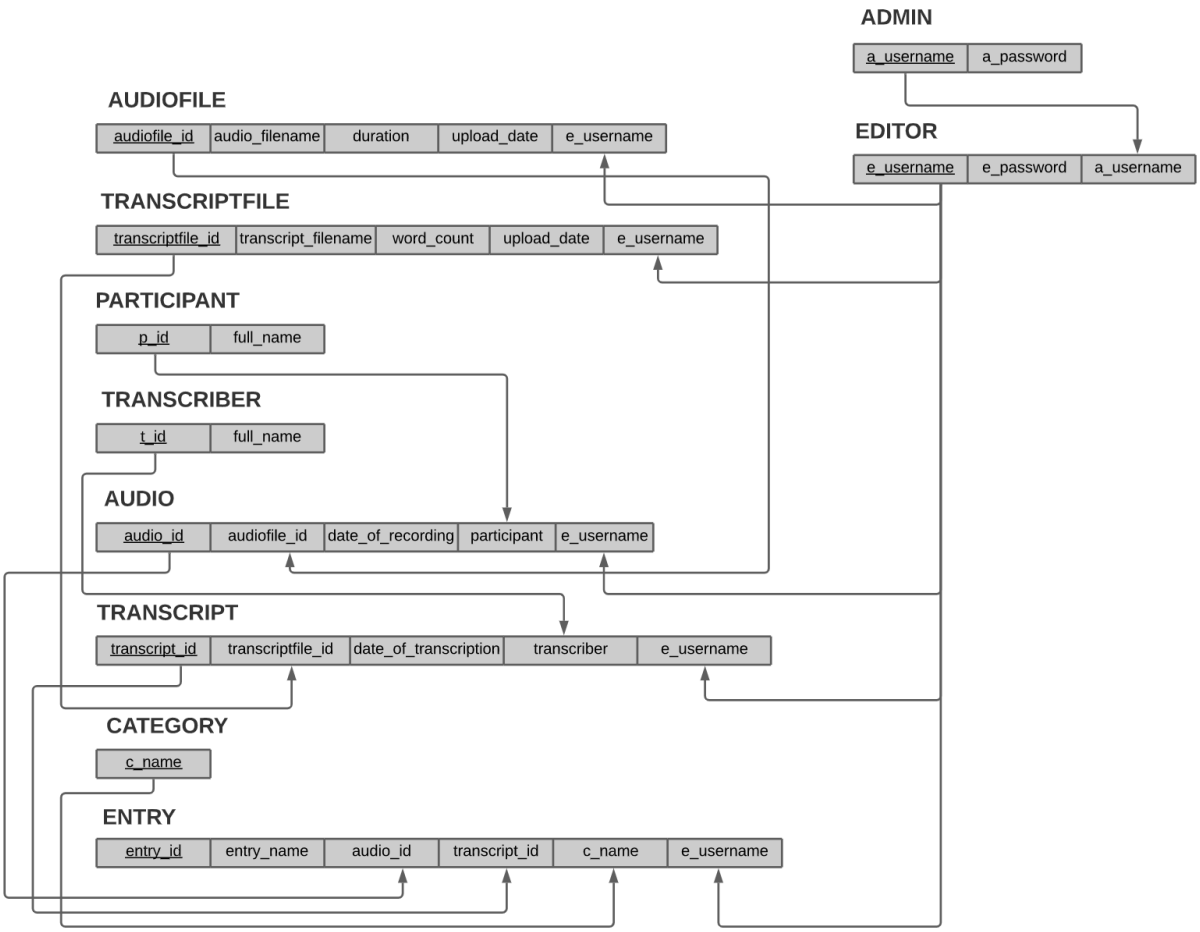
Michael Mongelli , Zachary Rich, Ashley Bennett , Alexandra Rizzo
Dr. DeGood
CSC315

CSC 315 Stage IV - Design

1. Review the Database Model document with stakeholders, and update the model as needed.



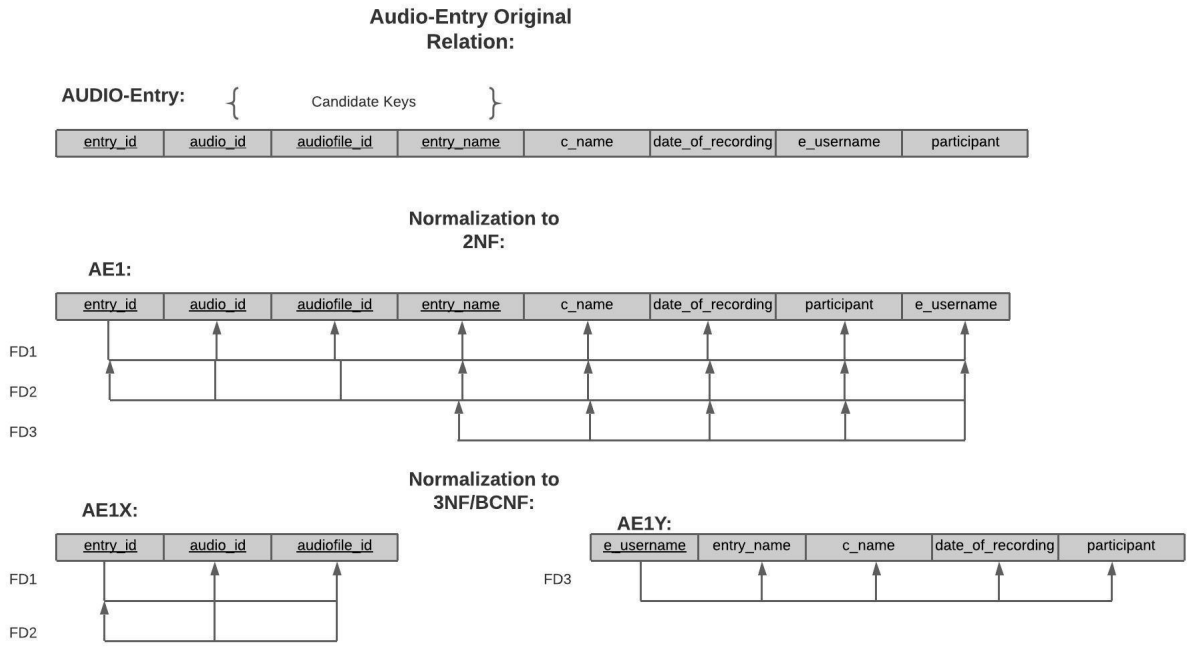


2. Demonstrate that all the relations in the relational schema are normalized to Boyce-Codd Normal Form (BCNF).

- For each table, specify whether it is in BCNF or not, and explain why.
- For each table that is not in BCNF, show the complete process that normalizes it to BCNF.

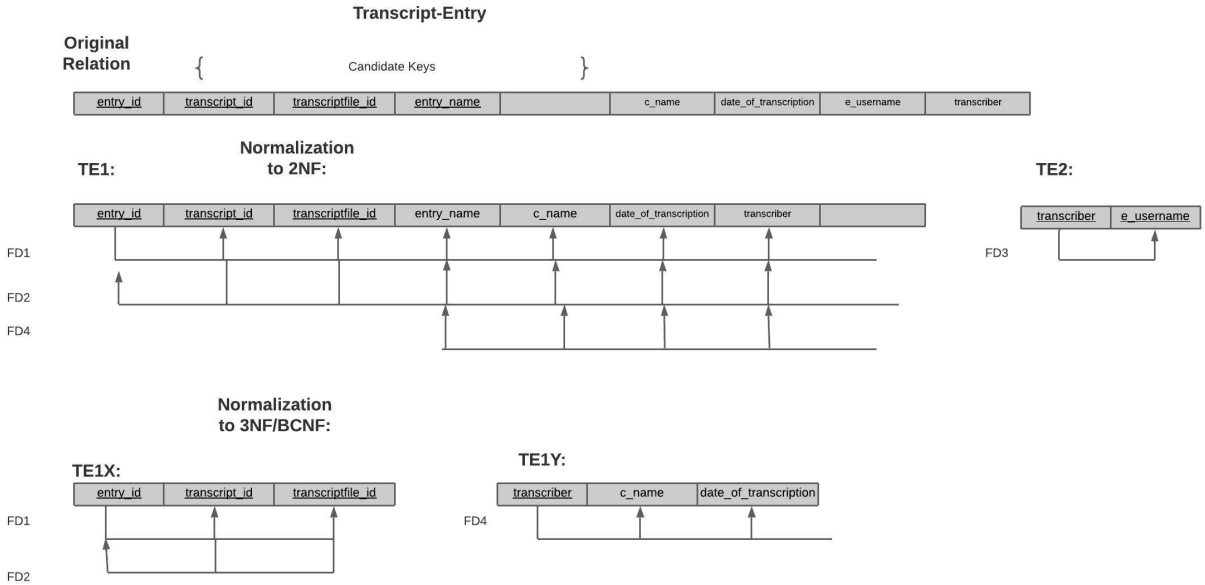
Audio-Entry:

- Currently this relation is not in Boyce-Codd Normal Form. This is because there is a lack of functional dependency on entry_id attribute, which acts as the primary key of this relation. To make this relation satisfy BCNF, this has to be decomposed into two relationships, AE1X and AE1Y:



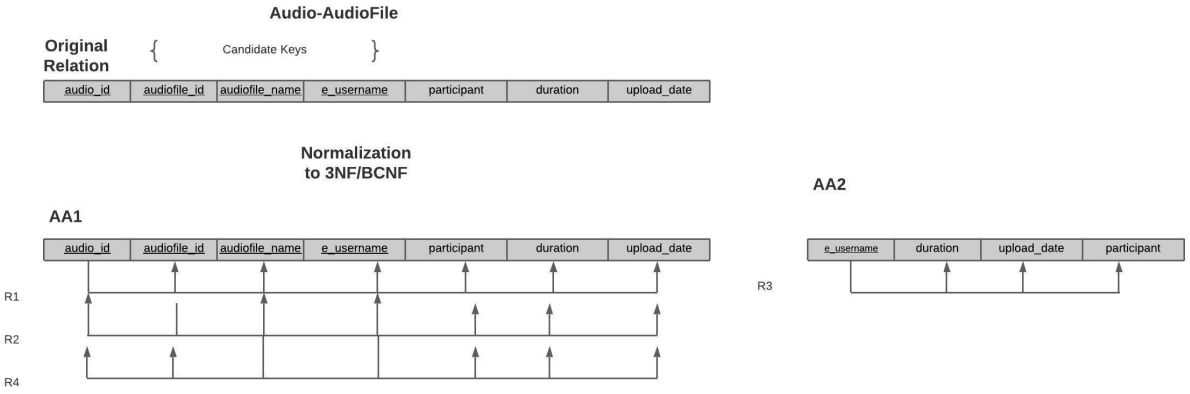
Transcript-Entry:

- Currently this relation is not in Boyce-Codd Normal Form. This is because there is a lack of functional dependency on the entry_id attribute, which acts as the primary key of this relation. Two separate relationships are created, TE1X and TE1Y, to satisfy BCNF.



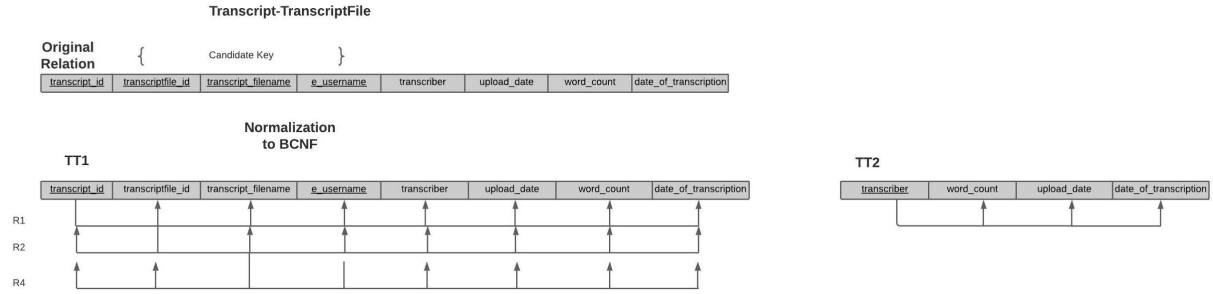
Audio-AudioFile:

- Currently this relation is not in Boyce-Codd Normal Form. This is because the filename and date_of_recording attributes currently lack a functional dependency on any candidate or primary key. To satisfy BCFN, we make sure that no prime attributes are transitively dependent on a key.



Transcript-TranscriptFile:

- Currently this relation is not in Boyce-Codd Normal Form. This is because the transcriber is not considered a primary key but has its own functional dependencies, meaning a nonprime attribute has a functional dependency.



Audio-Participant:

- This relation is in Boyce-Codd Normal Form. This is because values that would normally be nonprime have a functional dependency in which they are a primary key. This applies to the participant attribute, which is a nonprime attribute in Audio, is the primary key in the Participant entity.

Transcript-Transcriber:

- This relation is in Boyce-Codd Normal Form. This is because the nonprime attribute that has its own functional dependency (transcriber) is a primary key in the entity of the same name, and thus also acts as the candidate key for the First and Second normal forms of the Transcript entity.

Entry-Category:

- This relationship is in Boyce-Codd Normal Form. This is because with the category name attribute being its own primary key, it no longer needs a functional dependency on the e_username class, the only nonprime attribute in this relation.

Editor-Audio:

- This relationship is in Boyce-Codd Normal Form. This is because the e_username attribute is now a candidate key and a superkey, meaning all its functional dependencies can now use it as a primary key.

Editor-Transcript:

- This relationship is in Boyce-Codd Normal Form. Just like the Editor-Audio relationship, e_username is now a candidate key and superkey, meaning it is a primary key in all its functional dependencies.

Editor-AudioFile:

- This relationship is in Boyce-Codd Normal Form. This is because the e_username attribute is now a candidate key and superkey, meaning it is a primary key in all its functional dependencies.

Editor-TranscriptFile:

- This relationship is now in Boyce-Codd Normal Form. Just like Editor-AudioFile, e_username is now a candidate key and superkey, meaning it can act as the primary key in its functional dependencies.

Admin-Editor:

- This relation is in Boyce-Codd Normal Form. This is because the two nonprime attributes, a_password and e_password, have functional dependencies on the a_username and e_username parameters, respectively. There are also two functional dependencies for each parameter, as a_username and e_username also depend on the other group's password information, respectively. (i.e. for having an administrator reset their password).

3. Define the different views (virtual tables) required. For each view list the data and transaction requirements. Give a few examples of queries, to illustrate.

SQL	**Description**

`CREATE VIEW ENTRIES_BY_KEYWORD AS`
`SELECT`
`  entry_name,`
`  audio_filename,`
`  transcript_filename,`
`  PARTICIPANT.full_name AS participant,`
`  TRANSCRIBER.full_name AS transcriber`
`FROM`
`  ENTRY,`
`  AUDIO,`
	This table will be useful for storing information we would like to be available to search by the user using keywords. These will include attributes such as the name of the entry, the name of the audio or transcript file, and the name of the participant or

AUDIOFILE, PARTICIPANT, TRANSCRIPT, TRANSCRIPTFILE, TRANSCRIBER WHERE ENTRY.audio_id=AUDIO.audio_id, AUDIO.audiofile_id=AUDIOFILE.audiofile_id, AUDIO.participant=PARTICIPANT.p_id, ENTRY.transcript_id=TRANSCRIPT.transcript_id, TRANSCRIPT.transcriptfile_id=TRANSCRIPTFILE.transcriptfile_id, TRANSCRIPT.transcriber=TRANSCRIPT.t_id ;	transcriber in the entry. This way, the user can enter a keyword they are looking for in the search bar and it will search all these items for that keyword to look for relevant results. This could also be helpful because names for example could show up in the name of the entry and as a participant listed in the entry.
CREATE VIEW ENTRIES_BY_CATEGORY AS SELECT entry_name, c_name AS category FROM ENTRY ;	This is a useful table because the user should be able to search entries based one what category they belong to.
CREATE VIEW ENTRIES_BY_DATE AS SELECT entry_name, AUDIO.date_of_recording AS audio_date, AUDIOFILE.upload_date AS audiofile_date, TRANSCRIPT.date_of_transcription AS transcript_date, TRANSCRIPTFILE.upload_date AS transcriptfile_date, FROM ENTRY, AUDIO, AUDIOFILE, TRANSCRIPT, TRANSCRIPTFILE WHERE ENTRY.audio_id=AUDIO.audio_id, AUDIO.audiofile_id=AUDIOFILE.audiofile_id, ENTRY.transcript_id=TRANSCRIPT.transcript_id, TRANSCRIPT.transcriptfile_id=TRANSCRIPTFILE.transcriptfile_id ;	This table will be useful when the user tries to search the entries based on a specific date or range of time. The user can enter a date or time period and this action will return the names of all the entries which were either recorded, transcribed, or whose files were uploaded at that time frame. This will be helpful if the user wants to see which entry took place the longest ago, for example, or conversely, which ones were added the most recently. The user could also sort the entries in chronological order, which would also be done using the data in this table.
CREATE VIEW ENTRIES_BY_DURATION AS SELECT entry_name, AUDIOFILE.duration, TRANSCRIPTION.word_count FROM ENTRY, AUDIO, AUDIOFILE, TRANSCRIPT, TRANSCRIPTFILE WHERE ENTRY.audio_id=AUDIO.audio_id, AUDIO.audiofile_id=AUDIOFILE.audiofile_id, ENTRY.transcript_id=TRANSCRIPT.transcript_id,	It may be desirable for the user to be able to search the entries based on the length of the recording or the length of the transcription. For example, users looking to peruse the

`    TRANSCRIPT.transcriptfile_id=TRANSCRIPTFILE.transcriptfile_id` `;`	entries may want to look through shorter entries.

`CREATE VIEW ENTRIES_BY_EDITOR AS`
`SELECT`
`    entry_name,`
`    ENTRY.e_username AS entry_creator,`
`    AUDIO.e_username AS audio_info_editor,`
`    AUDIOFILE.e_username AS audiofile_uploader,`
`    TRANSCRIPT.e_username AS transcript_info_editor,`
`    TRANSCRIPTFILE.e_username AS transcriptfile_uploader`
`WHERE`
`    ENTRY.audio_id=AUDIO.audio_id,`
`    AUDIO.audiofile_id=AUDIOFILE.audiofile_id,`
`    ENTRY.transcript_id=TRANSCRIPT.transcript_id,`
`    TRANSCRIPT.transcriptfile_id=TRANSCRIPTFILE.transcriptfile_id`
`;`
	This view will be useful if the user wishes to search based on which editor created an entry, uploaded an audio or transcript file, or who entered the accompanying audio or transcript information. This may be useful to the linguistics department if they wish to search for entries edited/uploaded by a specific student or faculty member.

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

Query entries by keyword in entry_name, filename, or participant:
SELECT entry_name
FROM ENTRIES_BY_KEYWORD
WHERE COALESCE(entry_name + audio_filename + transcript_filename + participant + transcriber) LIKE user_input;

Query entries by category:
SELECT entry_name
FROM ENTRIES_BY_CATEGORY
WHERE category LIKE user_input;

Query entries by date:
SELECT entry_name
FROM ENTRIES_BY_DATE
WHERE COALESCE(audio_date + audiofile_date + transcript_date + transcriptfile_date) LIKE user_date;

Query entries by recording duration or transcript length:
SELECT entry_name
FROM ENTRIES_BY_DURATION
WHERE COALESCE(duration + word_count) LIKE user_input;

Query entries by editor username:
SELECT entry_name

```
FROM ENTRIES_BY_EDITOR
WHERE COALESCE(entry_creator + audio_info_editor + audiofile_uploader +
transcript_info_editor + transcriptfile_uploader) LIKE user_input;
```

Add audio file:

```
INSERT INTO Audio
VALUES (filename, duration, date_of_recording, date_archived, participants);
```

Add transcript:

```
INSERT INTO Transcript
VALUES (filename, date_archived, length);
```