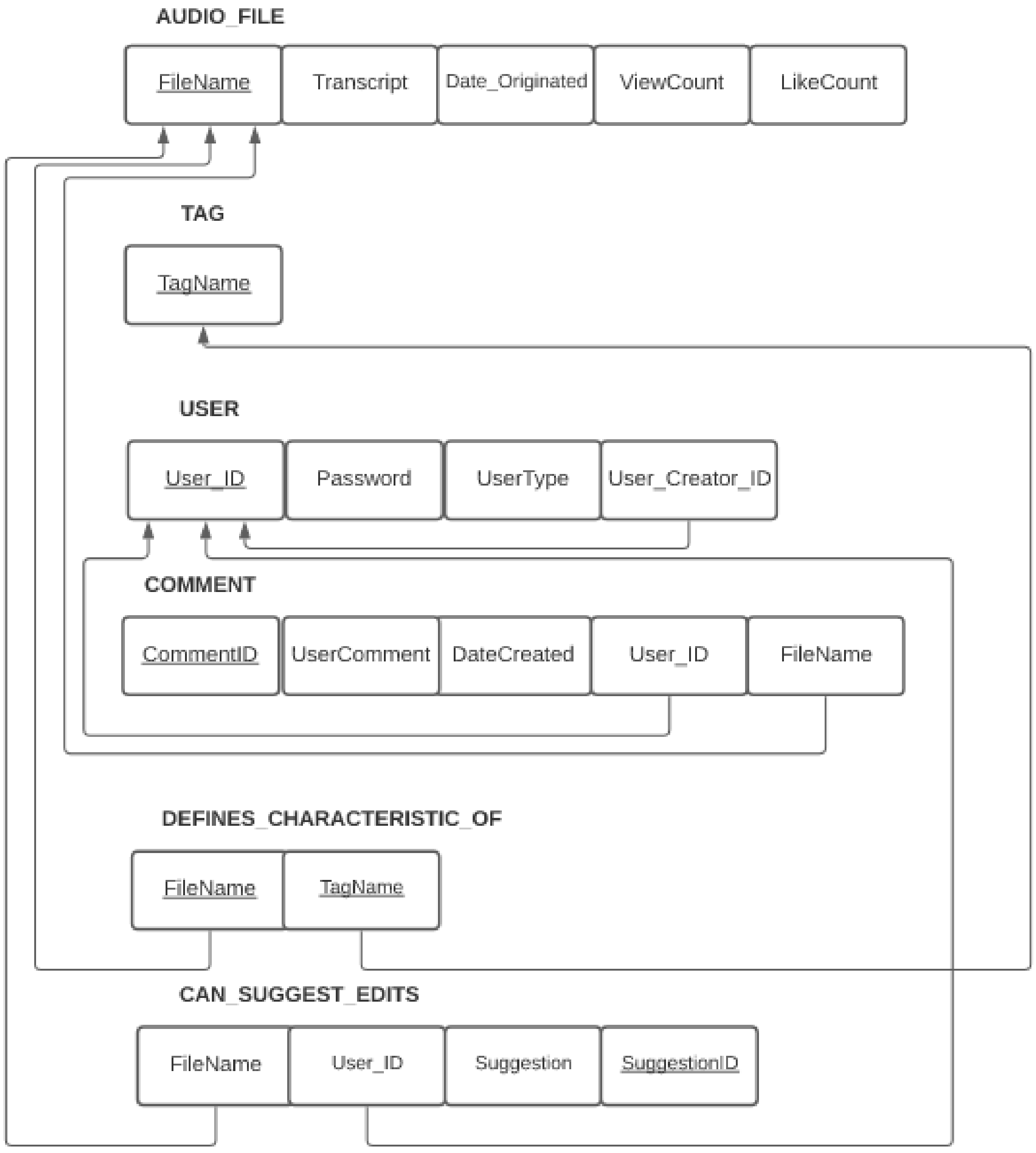
Summer Martin, Lalima Bhola, Jared Schmidt, Kyla Ramos: Group Three
Audio-Transcript Database Implementation
Stage Four: Design

1. Demonstrate that all the relations in the relational schema are normalized to Boyce–Codd normal form (BCNF).

After reviewing it, we made some changes to our original relation schema that we submitted in Stage 3 before starting the normalization. We took out TagID and condensed some USER attributes into one that works the same way. We also realized that we could not use the primary key we had for the CAN_SUGGEST_EDITS relation so we added SuggestionID and made that the primary key. This was all done before we started testing for normalization. The relational schema we used to check for normalization is:



- For **each table**, specify whether it is in BCNF or not, and explain why.
  - **AUDIO_FILE**

-- It is in first-normal form because each column is unique and contains one element. All of the attribute values are indivisible.
-- It is in second-normal form because all of the non-prime attributes are fully functionally dependent on the primary key FileName (which is the only primary key) and there are no partial dependencies. It is also fully functionally dependent on the candidate key transcript.
-- It is third normal for because there are no transitive dependencies.
-- It is in BCNF because for any functional dependencies, the left side is a super key. For example, either FileName, the primary key, or Transcript, the unique key, is a superkey.

  - **TAG**

-- It is in first normal form because each column is unique and contains one element. All of the attribute values are indivisible.
-- It is in second normal form because there is only one attribute which is the primary key TagName. There are no non-primary attributes or other candidate keys.
-- It is in third normal form because TagName is the only attribute so there are no transitive dependencies.
-- It is in BCNF because TagName is the only attribute.

  - **USER**

-- It is in first-normal form because each column is unique and contains one element. All of the attribute values are indivisible.
-- It is in second normal form because there is only one attribute used as the primary key, User_ID and all of the non-prime attributes are fully functionally dependent on User_ID
-- It is in third normal form because there are no transitive dependencies. None of the non primary key attributes could determine any of the other attributes.
-- It is in BCNF because besides the primary key, User_ID, there are no other prime attributes and no functional dependencies that do not have the primary key as the super key.

  - **COMMENT**

-- It is in first-normal form because each column is unique and contains one element. All of the attribute values are indivisible.
-- It is in second normal form because the primary key contains only one attribute, CommentID and all of the other non-prime attributes are fully functionally dependent on Comment ID. UserComment could not be a candidate key because there could potentially be two of the same comment.
-- It is in third normal form because there are no transitive identities. None of the non primary key attributes could determine any of the other attributes.
-- It is in BCNF because there are no functional dependencies that do not have a superkey on the left side. There are no prime attributes other than the primary key and no functional dependencies where a prime attribute is dependent on a non superkey.

  - **DEFINES_CHARACTERISTIC_OF**

-- It is in first-normal form because each column is unique and contains one element. All of the attribute values are indivisible.
-- It is in second normal form because FileName and TagName are the only attributes and they are both part of the primary key. You must have both to determine the instance of

Defines_Characteristic_Of. So, because there are no non-prime attributes, we can say that all of the non-existent non-prime attributes are fully functionally dependent on the primary key.
-- There are no attributes that are not part of the primary key so there are no transitive dependencies and as such, it is in third normal form.
-- It is in BCNF because there are only two attributes, both of which are needed for the primary key, so the relation only has the primary key. There are no functional dependencies where the left side is a non superkey and the right is a prime attribute.

- **CAN_SUGGEST_EDITS**

-- It is in first-normal form because each column is unique and contains one element. All of the attribute values are indivisible.
-- It is in seconds normal form because the non-prime attributes are fully functionally dependent on the primary key SuggestionID.
-- This is in third normal form because there are no transitive dependencies. For example {FileName, User_ID} could not determine Suggestion and so neither of them alone could either. Suggestion also could ont determine {FileName,User_ID} or either of them separately.
-- It is in BCNF because there are no other prime attributes, other than the primary key, because theoretically, a user could leave the same suggestion on an audio file twice or more times. As such, the only prime attribute is SuggestionID and none of the other attributes, together or apart, could determine SuggestionID for the reason stated previously. This means that there are no nontrivial functional dependencies where the left side is not a superkey.

- For each table that is not in BCNF, show the complete process that normalizes it to BCNF.

We did not need to normalize any of our tables because they were already in BCNF.

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

- Admin can: Select, Insert, Update, and Delete from any Table
  - For example, they can insert a new user into the user table, they can change the user type of a user, they can delete an audio file or update the actual audio file or transcript file, and they can select all the users they have created from the user table by selecting the tuples where their user id was the user_creator_id.
- Moderator can: Select, Delete Comments, Delete and manage suggestions, Update some attributes of audio files but not update the transcript or actual audio file, Insert/Update Tags
  - For example, they can delete comments from any file, delete suggestions from user suggestions, insert new tags for audio files,and update tags for audio files to keep the data relevant
- General User can: select and view all audio files, make suggestions, insert comments, update their own comments and suggestions.
  - For example, they can select audio files based on different parameters like view count or a certain tag or attribute, they can insert a new suggestion tuple for

some audio file, they can update or delete a comment or suggestion they have made for some audio file.

Views:

- We will need the view Audio_File_Tags with the data from the tables Audio_File joined with Defines_Characteristic_Of to do the transaction to search for/select all of the audio files and their information for a particular tag.

    -
      ```
      CREATE VIEW Audio_File_Tags
      AS SELECT *
          FROM AUDIO_FILE AS AF * DEFINES CHARACTERISTIC_OF  AS DC
          WHERE AF.FileName = DC.FileName
      ```
    - For example, they could use the view to search for all of the audio files with a date originated before 2000 that have the tag education.
- We will need the view User_Comments for the data from the tables User joined with Comment if we want to do the transaction to see all the comments a user has made and see the users type or other information at the same time.

    -
      ```
      CREATE VIEW User_Comments
      AS SELECT *
          FROM USER AS U * COMMENT AS C
          WHERE U.User_ID = C.User_ID
      ```
    - For example, they could use the view to see all of the comments made by moderator users.
- We can create the view Audio_File_Comments for the data from the tables Comment and Audio_File to do a transaction to select the tuples with all the comments made on a particular file and we want to see all the information for that file as well.

    -
      ```
      CREATE VIEW Audio_File_Comments
      AS SELECT *
          FROM AUDIO_FILE AS AF * COMMENT AS C
          WHERE AF.FileName = C.FileName
      ```
    - For example, you can use this view to Search all the comments made on Audio_File with more than 100 likes.
- We can have a view Creator for the data from the tables User joined with User where User_Creator_ID = User_ID to do a transaction to select all the tuples with users that were been created by another specific user (Admin) if we want to check the information of both the creator and the users they are creating.

    -
      ```
      CREATE VIEW Creator
      AS SELECT *
          FROM USER AS U  JOIN USER AS U2
          WHERE U.User_ID = U2.User_Creator_ID
      ```
    - For example, you can use this view to see how often a certain admin makes other admin users, or how many moderators they have made.
- We can have a view Suggest_File_Edit for the data from the tables from Can_Suggest_Edits joined with Audio_File to do a transaction to select all the

suggestions made for Audio_File and the Audio_File information so we can edit/update the file based on suggestions.

- ○ CREATE VIEW Suggest_File_Edit
  AS SELECT *
      FROM AUDIO_FILE AS AF * CAN_SUGGEST_EDITS AS CSE
      WHERE AF.FileName = CSE.FileName
- ● For example, you could use this view to see all the suggestions made on files with view counts > 100 so that you can work on those first because they get the most traffic

3. Design a complete set of SQL queries to satisfy the transaction requirements identified in the previous stages, using the relational schema and views defined in tasks 2and 3 above.

Select Queries:

SELECT * FROM Audio_FIle_Tag WHERE Date_Originated < 2000 AND TagName = ‘Education’

Select audio file by tag

- SELECT FileName FROM Audio_File_Tag WHERE TagName = ‘Education’;

/* Education is an example. You can replace the word education with any (relevant) word*/

Find all the tags an audio file has

- SELECT FileName, Tag FROM Audio_File_Tag WHERE FileName = <file name>;

Select audio file by view count

- SELECT FileName FROM AUDIO_FILES WHERE ViewCount < 100;

/* ViewCount < 100 is an example. You could replace the where condition to be any relevant amount of views, such as > 100 as another example. */

Select audio file by like count

- SELECT FileName FROM AUDIO_FILES WHERE LikeCount < 100;

/* LikeCount < 100 is an example. You could replace the where condition to be any relevant amount of likes, such as > 100 as another example. */

Select audio file by date originated

- SELECT FileName FROM AUDIO_FILES WHERE Date_Originated = ‘1/12/1950’;

/* Date_Originated = ‘1950’ is an example. You could replace the where condition to be any relevant origin date such as ‘February 1920’ as another example. */

Find the number of comments an audio file has had.

- SELECT COUNT (*) FROM Audio_File_Comments WHERE FileName = ‘World War II’;

/* FileName = ‘World War II’ is an example. You could replace the where condition to be any relevant file, such as ‘Economy’ as another example. You could also use a different attribute such as LikeCount to see the comments for files with a specified amount of likes */

Find the total number of views of all audio files

- SELECT SUM(ViewCount) FROM AUDIO_FILE;

Search for by user type to get all the admin users or moderators.

- SELECT * FROM USER WHERE UserType = <admin or moderator>;

Search Comment by User_ID to find all the comments made by some user.

- SELECT * FROM User_Comment WHERE UserType = <desired type>;

Search for a Comment by FileName

- SELECT * FROM COMMENT WHERE FileName = <file name>;

Search the Audio_File_Comments by FileName (a view created in the previous part) to find all the comments for a file and all the audio file information.

- SELECT * FROM Audio_File_Comments WHERE FileName = <file name>;

- SELECT * FROM Audio_File_Comments WHERE LikeCount> 100;

Search Comment by date created to get all the comments from one day.

- SELECT Comment FROM COMMENT WHERE DateCreated = <date>;

Find all the suggestions made on an Audio File with > 100 views

- SELECT FileName, Transcript, Suggestion FROM Suggest_File_Edit WHERE ViewCount > 100;

Find all the users made by some other user

- SELECT User_ID FROM Creator WHERE User_Creator_ID = <user ID>;
- SELECT * FROM Creator WHERE U.UserType = U2.UserType;

Insert Queries:

INSERT INTO TAG (school) VALUES (‘Education’);
/* Education is an example. You can replace the word education with any (relevant) word*/

INSERT INTO AUDIO_FILE VALUES (‘World War II’, ‘<transcript file here>’, ‘9/15/1980’, ‘1000’, ‘999’);

INSERT INTO COMMENT VALUES (‘004567’, ‘<usercomment here>’, ‘4/1/2021’, ‘001234’, ‘World War II’);

```
INSERT INTO USER VALUES (‘000010’, ‘Y$pU1gZdaH’, ‘Moderator’, ‘collegeStudent04’);
/* User_ID would either be default or serial. Only Admin users can make a user admin or a
moderator (see update user)*/

INSERT INTO DEFINES_CHARACTERISTIC_OF VALUES (‘Cold War’, ‘history’);

INSERT INTO CAN_SUGGEST_EDITS VALUES (‘Gubernatorial Election’, ‘000030’,
‘<suggestion here>’);
```

Delete Queries:

```
DELETE FROM AUDIO_FILE WHERE FileName = ‘World War II’ ;
/* FileName = ‘World War II’ is an example. You could replace the where condition to be any
relevant file name, such as ‘Economy’ as another example. This can only be */

DELETE FROM CAN_SUGGEST_EDITS WHERE SuggestionID = <some number>;
/* Delete tuple once suggestion has been handled.*/

DELETE FROM TAG WHERE TagName = ‘Education’;

DELETE FROM USER WHERE User_ID = ‘001234’;
/* Delete a user account in case an account is no longer necessary, for whatever reason.*/

DELETE FROM DEFINES_CHARACTERISTIC_OF WHERE FileName = <name of file> AND
TagName = ‘Education’;
/* removes a tag from an audio file, education is an example*/

DELETE FROM COMMENTS WHERE User_ID = ‘004567’;
/* Delete all comments of a certain user. */

DELETE FROM COMMENTS WHERE FileName = <name of file>;
/* delete all of the comment for some audio file*/

DELETE FROM COMMENTS WHERE CommentID = ‘000123’;
```

Update Queries:

```
UPDATE USER
SET UserType = <Admin, Moderator, General>
WHERE User_ID = <user_ID number>;
/* update usertype for a certain user */
```

```
UPDATE AUDIO_FILE
SET Transcript = <Transcript>
WHERE FileName = <name of file>;
/* update transcript for a certain audio file */

UPDATE COMMENT
SET UserComment = <updated userComment>
WHERE CommentID = <ID of comment>;
/* update UserComment content for a specific comment (users can edit their own comments
after posting) */

UPDATE USER
SET Password = <new password>
WHERE User_ID = <user_ID number>;
/*updates the password of a certain user*/

UPDATE CAN_SUGGEST_EDITS
SET Suggestion = <edited suggestion>
WHERE SuggestionID = <suggestionID number>;
/* updates a user’s suggestion if they want to make changes to something they said*/
```