CS 121 25wi Final Project Proposal

Due: Friday Feb. 14th 2024, 11:30PM

- *Note: Proposal deadline is an exception to weekly Monday deadlines for sets, between A3 and A4. Try to submit by Thursday, but we will accept by Friday.* ***A4 is due on Tuesday Feb. 19th*** *due to Monday holiday but will be shorter with proposal overlap.*

Notable 25wi updates:

- app.py with function stubs is **required** in proposal this year
  - [02/10] El went through some student project ideas and will provide another version of template since we're simplifying client/admin breakdown this year
  - A summary of minimum requirements is added to this document (show_options() and starts to function stubs)
- Setup-proposal.sql (DDL) **required** in proposal
- Procedural SQL and DB Performance **both removed** from proposal but kept for reference
  - Triggers will be **optional** for the Final Project this year
- Other requirements may be reduced a bit to account for lost week during wildfires

For the final project, you may choose to work with up to one other person. For partner-finding, use the #partner-search Discord channel to share your interests, any ideas, and preferences for working with a partner. There will be optional work time in lecture to find partners and ask about any questions with El around; we encourage you to attend to aim for finalizing a project idea, a partner, and any dataset/scoping questions you have. You are welcome to remove any suggestions/notes from this proposal in your proposal, as long as your required components are clearly in the order specified (we've highlighted text for you to answer in blue for ease).

Note: If working with another student, only one of you needs to submit the required files to CodePost, but ***both of you should have equal contributions; i.e. you should ideally work through this document together.*** *For ease of grading please have the other student submit a simple* **collaboration.txt** *file to their CodePost submission in the following format (failure to do so may result in deductions):*

```
Names: <student1>, <student2>

CodePost emails: <email1>, <email2>
```

For full credit, we are looking for a robust proposal demonstrating careful consideration of the motivation, design process, and implementation plan for your application; you should expect to spend 3-4 hours minimum on your proposal (finding a dataset may take the longest). For each part of this proposal that requires your response, you should have 2-3 sentences (bullet points are encouraged as well) justifying your answers. We strongly encourage you to include as much detail as you can in this proposal (this is your only assignment this week) and you can include any diagrams and SQL brainstorming (e.g. a start to your DDL) with your submission on CodePost. If you want to add any additional planning, you can include a diagram as a PDF/PNG/etc. and/or .sql files, though not required. You can also include a starter Python program with function stubs for a command-line implementation you will complete for the Final Project (without SQL integration).

Summary of Files to Submit (max 3MB, reach out to El if you have trouble with this limit, but usually compressing images and/or the PDF file using your system's viewer or an online compressor will work):

- **proposal.pdf (a copy of this document filled out)**
- ***additional diagrams/sketches/brainstorming not otherwise required; you can include these at the bottom of your proposal document or as additional files in your submission.***
- **setup_proposal.sql (start to your DDL)** - your final project submission will rename this appropriately
- **app.py (or app_client.py/app_admin.py) function stubs**

The advantage of adding these in your proposal is to get you started in advance, and also to get feedback from the staff on your design and implementation so far.

Student name(s): Emeka Nkurumeh

Student email(s): e.nk@caltech.edu

Database/Application Overview

In this proposal, you will be "pitching" your project, in which you have some freedom in choosing the domain of with a structured set of requirements that bring together the course material in a single project, from design to implementation to tuning. Keep in mind that unlike CS 121 assignments, you are free to publish/share your project, which can be useful for internship or job applications. In terms of scope, you should shoot for 8-12 hours on this project, though you are free to go above and beyond if you choose.

First, what type of database application are you planning on designing? Remember that the focus of this project is the database DDL and DML, but there will be **a small Python command-line application** component to motivate its use and give you practice applying everything this term in an interactive program; you can find a template from last year here, which may be adjusted slightly before the Final Project is released, but gives you an idea of the breakdown. Don't worry too much about implementation at this step, and you can jot down a few ideas if you have more than one. Just think about something you would like to build "if you had the data" which could be simulated with a command-line program in Python. Your command-line program will start with a main menu with usage options for different features in your application. **For students who took CS 132, you may alternatively use a web-based application which El can help with.** Staff are here to help you with scoping!

Proposed Database and Application Program Answer (3-4 sentences) :

The database will consist of Last.FM listening data. The frontend application will then allow users to query this listening data in a fashion similar to Spotify Wrapped (total listening time, most listened to artists, etc.) Additionally, it will support more advanced queries such as filtering by artist, time

span, genre etc. For an MVP this data will be output in a simple text-based format, but we hope to generate graphics as well.

Next, where will you be getting your data? We will post a list of datasets to get you started, but you can find many here and here (look for ones in CSV file(s), which you will break into schemas similar to the Spotify assignment; we can also help students convert JSON to CSV if needed). You are also welcome to auto-generate your own datasets (e.g. with a Python script or using ChatGPT similar to A2 Part D) and staff are more than happy to help you with the dataset-finding process, which can take longer than the rest of the starting design process.

Data set (general or specific) Answer:
Our data will come from Last.FM and MusicBrainz (Last.FM reports the scrobble information with IDs that reference the MusicBrainz database). Since the rate limits on MusicBrainz are very high (50 requests/s) on data import, the scrobble information will be augmented with data from MusicBrainz that will be cached in the DB.

Who is the intended user base and what is their role? Identify at least one client usertype, and one potential admin. These different users may have different permissions granted for tables or procedural SQL. Consider clients as having the ability to search ("query") data from the command line, submit requests to insert/update data, etc. In past years, some students had a separate client .py file and admin .py file, with admins having admin-related features, such as approving requests, inserting/updating/deleting information, querying, etc. Including app_client.py and app_admin.py is optional for the proposal this year, but would be a recommended exercise for proposing two different applications to interact with your database. app_admin.py would have a separate username/password in get_conn() and an admin-specific set of show_options() related to admin functionality (e.g. managing inventory or user accounts for an e-commerce store).

***Client user(s) Answer*:**

The user-facing CLI would handle performing queries against the database for generating reports, recommendations and listing scrobbled MBIDs. Regular users of the app would be clients.

Admin user(s) Answer:

The admin accounts would be the only ones with the ability to bulk insert data into the database upon account creation. Admin users would mainly be active during account creation and data import. They also will have the ability to delete and view existing accounts.

Required Features

The full specification of the project will outline the requirements of the project, but you remember you should aim to spend 8-12 hours (it replaces the final exam and A8), depending on how far you want to go with it. The time spent on finding or creating a dataset will vary the most. For the proposal, you will need to brainstorm the following (you can change your decisions later if needed).

DDL: What are possible tables you would have for your database? Include at least 4 tables in your response. Remember to think about your application (e.g. command-line functions for user prompts to select, add, update, and delete data, either as a client or an admin user). To make this step easier, thinking about the menu options your application provides, as well as the queries you might want to support, is helpful to narrow down the information you'll want to model.

1. Provide a general breakdown of your database schema and tables below, as well as any design decisions you may run into in your schema breakdown. You can provide any notes/questions that you would like us to be aware of for your proposed project.
2. Start your DDL with CREATE TABLE statements in the **`setup_proposal.sql`** file attached in your submission. We will not grade strictly on DDL, but your final submission should have documentation, etc. and we will prioritize evaluating your table breakdown, choice of attribute domains, and keys/relationships.

Answer:

The tables will be described using a slight extension of the DDL language given in class:

row(column1 : INT, column2 : BLOB?, column3 : CHAR)

- "column1" has type INT and is a primary key
- "column2" has type BLOB and may be NULL
- "column3" has type CHAR may **not** be NULL

Columns called "X_mbid" will be taken from MusicBrainz. These are 36 character UUIDS that are guaranteed to be globally unique (there is no overlap between artists, albums, and tracks).

"mbids.name" is a UTF-8 encoded string given in the native language of the artists unless MusicBrainz or Last.FM indicate they use a different form.

Frecency (not a typo) is a combined measure of frequency and recency used for the recommendation algorithm. It is described in this paper and this post by Mozilla.

mbids(mbid : CHAR(36), name : VARCHAR(256), frecency : INT?)

genres(mbid CHAR(36), genre_name : VARCHAR(32))

artists(artist_mbid : CHAR(36))

Since "album_mbids" can have multiple artists associated with them, this table serves as a linking table between them.

albums(album_mbid : CHAR(36), artist_mbid: VARCHAR(36))

tracks(track_mbid : CHAR(36), artist_mbid: CHAR(36), album_mbid: CHAR(36)?, track_length : TIME)

users(user_id : INT AUTO_INCREMENT, user_name : VARCHAR(16))

scrobbles(scrobbble_id : INT AUTO_INCREMENT, scrobble_time : DATETIME, track_mbid : CHAR(36), user_id: INT)

This schema was inspired by this SO post and a desire to have a single table for tagging MBIDs with their genre. It's essentially OOP-style inheritance.

The artist, album, and track tables all serve a similar purpose to those in the Spotify schemas in A3.

The mbid_genre tables serve to allow tagging of music, artists, and albums with the genre.

"Scrobbles" are music listening events.

Queries: Part of the application will involve querying data in a way that would be useful for a client (e.g. searching for "Puzzle" games made in the last 5 years, ordered by year and price in a Video Game Store database, or finding all applicants who are pending for an adoption agency).

Identify at least 3 queries that would make sense in a simple command-line application. In your answers, provide a brief description of the query or pseudocode, as well as the purpose in your application. These will likely be implemented as SQL queries within Python functions, wrapping your SQL queries (the methods of which will be taught in class). You are welcome (and encouraged) to add more, though not required.

Answers:

1. Search: Accepts a simple pattern language that allows search by artist/album/track names or tag. When searching for artists, the albums and tracks with scrobble counts are reported for each artist. For albums, just per-track scrobble counts are reported. For tracks, the scrobble count with dates and tags is reported. The tags option is used for filtering results but can be used on its own to filter all available artists/albums/tracks.

2. Report: Accepts an optional time range as well as a search pattern. These are used to generate a Spotify Wrapped-style listening report (top 5 artists/albums/tracks/tags with scrobble counts, total listening minutes, etc.)

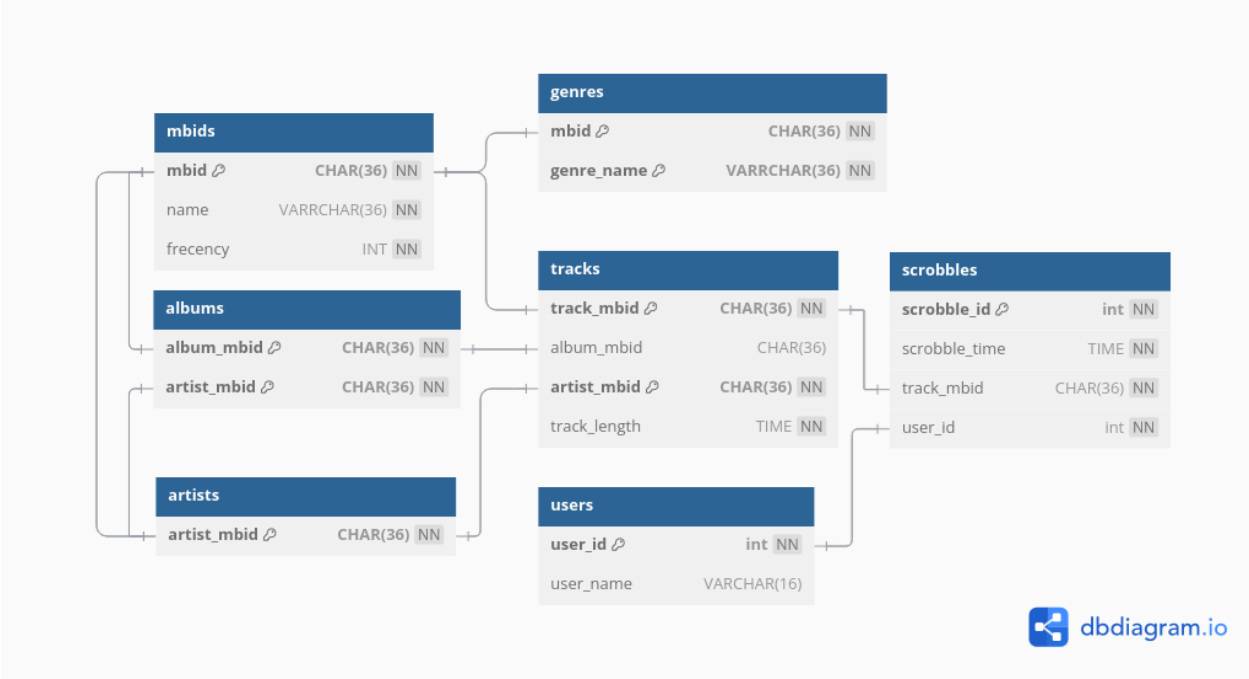
3. Recommend: Accepts an optional search pattern. Given your listening history, recommend a song you haven't listened to frequently/recently with some randomness thrown in to prevent deterministic results.

E-R Model: There will be an ER model component, which we will cover in a few weeks. ER models are an essential part of database projects, and can vary in conventions. For the proposal, you will not need to complete formal ER diagrams, but you will need to provide 1) a visual representation of your database (a very useful strategy for the design and planning phase, especially when proposing a database model to collaborators with varying technical backgrounds) and 2) some visualization of one possible program flow for a client user from start to finish. For 2), you are encouraged to also

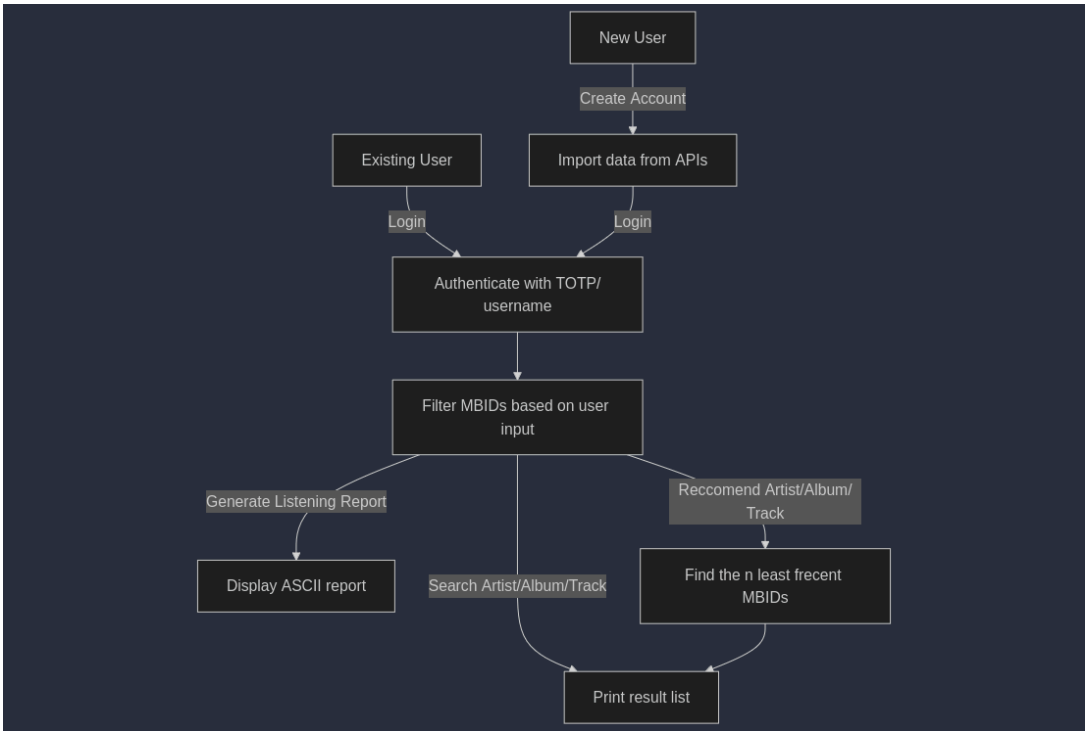
include an example program flow for an admin user, though are not required for the proposal. You have flexibility in how you represent your database application here and both diagrams do not necessarily need to be separate, but you must:

1) Have some representation of (all) proposed schemas and relationships from your DDL (with arrows for foreign keys between tables) that is interpretable.
    - Use annotations (or equivalent) for your proposed queries/client interaction such that someone (e.g. a reviewer) could identify which tables/attributes would be relevant to clients/users. Consider color-coding of keys, etc. to improve your proposed schema organization. Students have also previously added questions for design/implementation decisions they are uncertain about (e.g. "Should this *menu* table be broken down into *drinks* and *foods* or kept as *menu* with the category attribute?")
2) Include at least one visualization (e.g. flow-chart) of a runthrough of your client program from start to finish (it may help to think of your command-line client program as a simulation of a website that interacts with your database). The following is an example (a text-based version, though a visual flow chart is an improvement at the proposal stage to show the different possible edge cases):
    - User starts program (or visits website) -> creates an account after being prompted to login/create account (inserts into loyal customers table with inputted data, such as email) -> logs into created account if success (validation is done on database side) -> is presented a list of options -> inputs ('menu') for show all cafe products listed in '<n>: <product name> <price>' format, where <n> starts with 1 -> inputs 3 to add a Black Coffee to their cart -> confirms 'y' to finish adding to cart and submit order -> order is added to orders table with customer's id and product id -> gets a success message and estimated wait time ('simulated') and is directed back to the main menu -> chooses to log out and finish the order

Schema Diagram (may also alternatively as diagrams.png or diagrams.pdf)



Example User Flowchart(s):



app.py Client Program

Now that you have brainstormed your DDL and UI flowchart, you have a good start to draft your client application for the project. For the scope of the proposal, we want students to get started early with app.py, prioritizing "proof of concept" and modeling the possible UI flow. app.py will be the client program interfacing with your MySQL database, and we have provided a template to get you started.

For the proposal, your app.py should at minimum have:

- A list of 3 or more possible options for your program in show_options()
    - These will likely correspond to your diagrams and queries above
- Function stubs for each option/feature in your app.py
    - Use docstrings/comments to specify any arguments/input, and returns/output
    - You can use comments to show your current ideas for implementing the functionality
    - For input prompts, you are welcome to "mock" examples; this will make it easier to understand the intended flow for your application, and also easily translate to SQL when implementing the FInal Project requirements
    - Think about possible helper functions as you work through these
- If you choose to include both a client and admin application for your proposal, you can submit app_client.py and app_admin.py (there may be overlap with get_conn() and show_options(), with get_conn() having a separate username/password for admin privileges, covered in Wed. 02/12 slides)

You aren't expected to spend more than 30 minutes or so on this part when starting with the template, but should make as much progress early on before the Final Project later. Students can see some past demos in lectures/OH this week as well. Include a comment at the top of the program with any questions you have about your proposed client program and feedback you would like us to prioritize (we won't give in-depth feedback to this otherwise, knowing it's a draft subject to change).

Procedural SQL and Database Performance answers are not required for 25wi proposal, but left in for a preview of the corresponding Final Project component)

Procedural SQL *(complete requirements are left in for a preview of the corresponding Final Project component with slides posted in advance):* You will need to implement at least 1 each of a UDF (user-defined function), procedure, and optionally, a trigger in your project. Identify at least one UDF and/or one procedure here. For each:

1. What is the name of the function/procedure?
2. Why is it motivated in your database? Consider the differences discussed in lecture for UDFs, procedures, queries, temporary tables, and views. In your Final Project, we'll be looking for

you to demonstrate an understanding of appropriate design decisions here (and we're happy to help discuss any trade-offs)

Consider some examples in class/HW, such as logging DML queries, adding an extra layer of constraint-handling (e.g. the overdraft example from lecture), etc. For procedures, these can be called in an application program written in a language like Python or Node.js, so these are especially useful to avoid writing queries in such application programs, *especially* SQL that performs **`INSERT/DELETE/UPDATE`** which you do not want to leave to an application user. Remember that you can also set permissions for different users to access tables and procedures. In your Python program, you can connect to your database as different users defined in your database (similar to the Node.js program El will demo).

(Optional Answers)

UDF(s):

1.
2.

Procedure(s):

1.
2.

Trigger(s):

1.

Database Performance: *At this point, you should have a rough feel for the shape and use of your queries and schemas. In the final project, you will need to add at least one index and show that it makes a performance benefit for some query(s). You don't need to identify what index(es) you choose right now (that comes with tuning) but you will need to briefly describe how and when you would go about choosing an index(es). Refer to the material on indexes if needed.*

Performance Tuning Brainstorming:

From what I understand, I think I would have to index the scrobble table due to its sheer size. I think track_mbid is probably going to be a good index for searching, but for reports, the time the scrobble happened seems much better. Since I think searching will be more frequent than generating reports, I think it would be best to optimize for that. Or, I could just run a bunch of test queries and check

what is slow. It could be the case that everything is already fast enough and I don't need an index for the scrobble table.

However, the "genres" tables would all probably benefit from an index. All the rows are highly regular and the queries against them will be mostly aggregate queries where we group by MBID.

"Stretch goals"

If you are particularly eager for a certain application (have your own start-up in mind?), it is easy to over-scope a final project, especially one that isn't a term-long project. You can list any stretch goals you might have "if you had the time" which staff can help give feedback on prioritizing.

Answer:

- Either a web-based UI for viewing results or an SVG-based one similar to FlameGraph.
- Scrobbling support (detect playing music and report it Last.FM while also updating the local database) or Last.FM sync on user login.

Potential roadblocks

List any problems (at least one) you think could come up in the design and implementation of your database/application.

Answer:

- Names. Last.FM is crowdsourced and people submit names in different languages to Last.FM. Scrobble sources might also not agree on name for a given artist
- Database Size. I'm not sure how good MySQL is at storing and retrieving data on consumer hardware. My own scrobble data is roughly 10MB of CSV and that's not to mention the extra data we will be getting from MusicBrainz. There might be too much data for me to generate reports over long periods of time or provide too much information (my current plan for displaying tags consists of me performing a lot of NATURAL JOINs).
- Time as reported by Last.FM is in UTC. Is there a way to specify this in MySQL so queries can be made in the local timezone?
  A: Just handle translation at the application boundary and use UTC internally.
- Edge Cases:
  - Tracks with multiple artists (collaborations).
  - Albums with multiple artists (splits).
  - Tracks with no album (singles).

Review

This year, we are adding a small component to get students feedback from their peers and to practice giving feedback on proposed projects. You do not need to spend a lot of time on feedback (but may earn up to 3 additional points for above-and-beyond effort, such as reviewing multiple student posts with thoughtful feedback), but for for full credit you must:

- Post your overview followed by one part of your proposal (e.g. diagrams or DDL) on #final-project-review and include at least one question you have you'd like feedback on
- Review one other posted project and provide 1-3 sentences of feedback that demonstrates at least 5 minutes of reflection of trade-offs and concepts you've learned so far. Feel free to bring in your own user experiences! For example, you may have experience inputting a long last name that a website has a size limit on when creating an account, and could note this when suggesting a modification to another student's DDL for representing names.

Link to your Discord post:
N/A :(

Link(s) to your Discord feedback:
https://discord.com/channels/1325070896913846344/1339359506836230230/1340194669362942063

Open Questions

Is there something you would like to learn how to implement in lecture? Any other questions or concerns? Is there anything the course staff can do to help accommodate these concerns?

Answer:
Password management/user authentication. How are we supposed to store passwords securely for user login?

Have fun!

Optional Brainstorming/Sketches/Other Notes

This is an optional section where you can provide any other brainstorming notes here that may help in the design phase of your database project (you may find it helpful to refer to the early design phase in your Final Project Reflection).
