CMPT 354 Mini Project

Emmanuel de Caiman, Alex Menzies

April 2nd, 2025

Overview:

A simple command line interface (CLI) that allows users to manage library items, participate in events, and apply for volunteer opportunities. This application is written using Python and SQLite3.

```
Main Menu
1. Find an Item
2. Borrow an Item
3. Return an Item
4. Donate an Item
5. Find an Event
6. Request Event Recommendations
7. Volunteering Opportunities
8. Ask a librarian for help
9. Exit
Enter [1-9]:
```

Features

As a user, you can perform the following actions:

- Find an Item: Search for books, online books, magazines, scientific journals, CDs, records, etc.
- Borrow an Item: Check out an available item.
- Return an Item: Return an item to the library and clear it from your account.
- Donate an Item: Donate new or used items.
- Find an Event: Sign up for library-hosted events.
- Request Event Recommendations: Search upcoming events based on your own favorite genre.
- Volunteer Opportunities: Sign up as a volunteer to help with library operations or events.
- Ask a librarian for help: Receive information about system use, library hours, or general inquiries.

Installation

Ensure you have the following installed on your system:

- Python ($\geq$ 3.12)
- SQLite3

Run application `main.py` with `python3 main.py`. The application will prompt to enter your user ID. Available user IDs are numbers 1-5. The database is already populated with sample data, which you can view in `create_tables.ipynb`.

Step 2: Project Specifications

To break down the needs and requirements of our library database, we will take a look at what functionalities our database will need to satisfy for the library, and then break this down into what entities and relationships it needs.

Functionality and Requirements:

- The database should allow the library to manage its inventory of items, including adding new items and updating their availability status.
- The database should store information about all library users, and track the borrowing and returning of items from each user, including calculating fines for late returns.
- The database should store details about library events, and handle recommending events for specific audiences based on their genre preferences.
- The database should manage employee records and their role within the library, including volunteers.
- The database should automatically update a users favorite genre based on the most popular genre types borrowed.
- The database should allow storing items that are not currently in the libraries inventory, to keep track of which items the library may add in the future.
- The database should allow users to donate items, which should update the current libraries inventory to include the item.

Based on the functionality we need for the library database, we can define the following entities and relationships to construct this.

Entities and Relationships

- **Library:** Contains information about the library itself, such as its name, address, and contact details.
- **Item:** Represents the various items available in the library, such as books, magazines, CDs, etc. Each item will have a publisher, location (either physical location in library, or URL if online book), genre, and status (indicating if it is borrowed, available, or to be added).
- **Category:** Holds the various categories that an item can be, such as book, journal, online book, CD, vinyl, etc.
- **Reading (inherits from Item):** Contains details specific to reading items, such as its international standard book number (ISBN) and author.
- **Music (inherits from Item):** Contains details specific to music items, such as the artist name, and number or songs.
- **User:** Represents the library users, including their name, birth date, phone number, address, email, and favorite genre.
- **Employee (inherits from User):** Represents employees of the library. They contain the same information as Users, but have additional attributes of a job title and salary.
- **Events:** Contains information about various library events, including the date, time, room number, and a description of the event. It also contains the ID of the library the event is at.
- **Audience:** Defines the target audience for events, storing the type (age range) of audience, and the genre.
- **Borrows:** Stores the borrowing of items by users. This includes the borrowed date, due date, returned date, and a fine for not returning the book. This also contains the books item ID, and the users ID.
- **Attending:** Relationship to track which users are attending which events.
- **RecommendedFor:** Relationship to link events to their recommended audiences.

Step 3: E/R Diagram

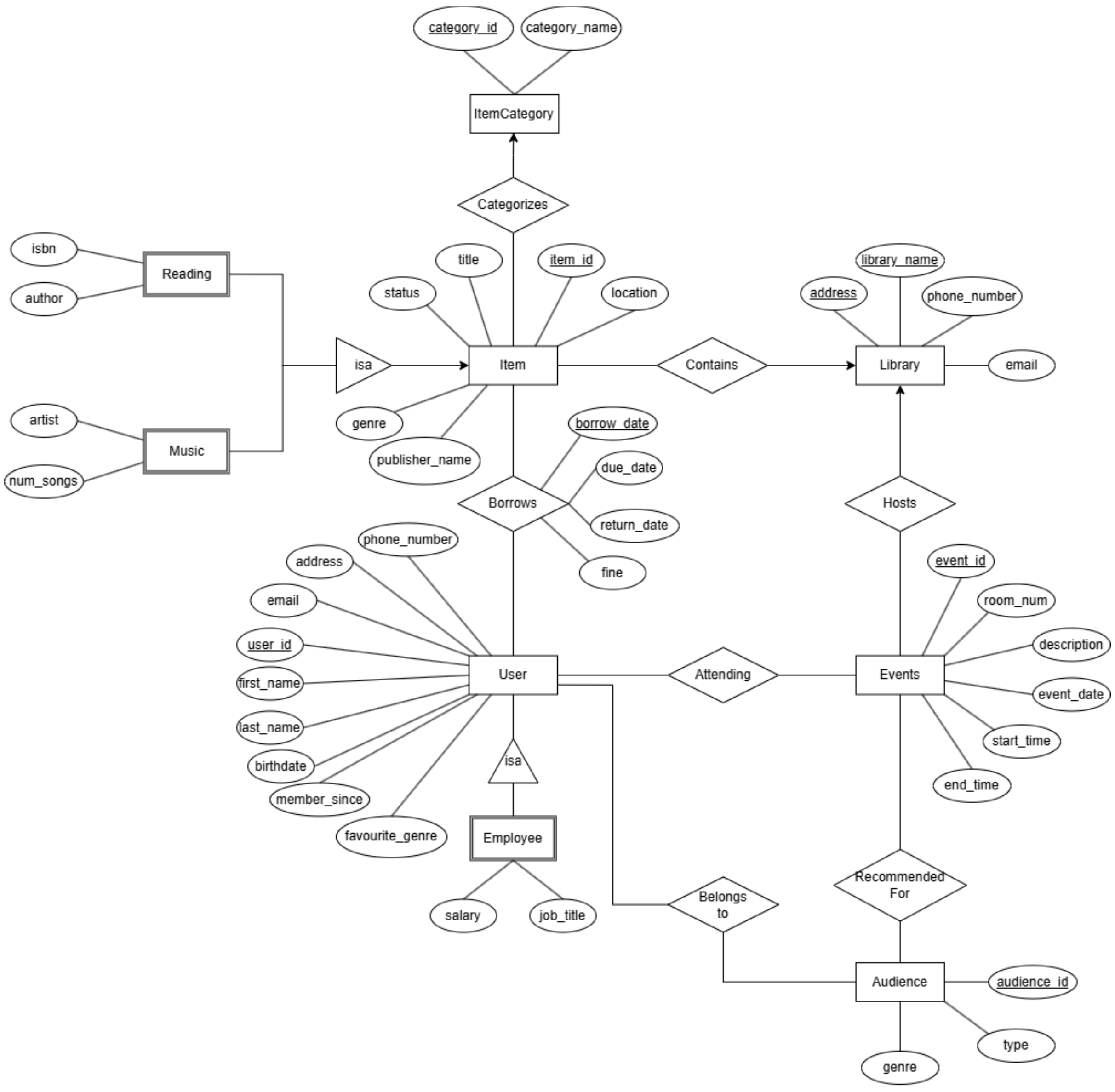

Figure 1: E/R Diagram

Step 4: Does your design allow anomalies?

Interpreting our E/R diagram, we get the following schemas for our library database. In each relation, primary key attributes are underlined, and foreign keys are noted in superscript.

- **Library** = {library_name, address, phone_number, email}
- **Item** = {item_id, category_id$^{\text{FK}}$, library_name$^{\text{FK}}$, address$^{\text{FK}}$, title, status, genre, location, publisher_name}
- **ItemCategory** = {category_id, category_name}
- **Reading** = {item_id$^{\text{FK}}$, isbn, author}
- **Music** = {item_id$^{\text{FK}}$, artist, num_songs}
- **User** = {user_id, first_name, last_name, birthdate, phone_number, address, email, member_since, favourite_genre}
- **Employee** = {user_id$^{\text{FK}}$, job_title, salary}
- **Events** = {event_id, library_name$^{\text{FK}}$, address$^{\text{FK}}$, room_number, description, event_date, start_time, end_time}
- **Audience** = {audience_id, type, genre}
- **Borrows** = {user_id$^{\text{FK}}$, item_id$^{\text{FK}}$, borrow_date, due_date, return_date, fine}
- **Attending** = {user_id$^{\text{FK}}$, event_id$^{\text{FK}}$}
- **RecommendedFor** = {event_id$^{\text{FK}}$, audience_id$^{\text{FK}}$}
- **BelongsTo** = {user_id$^{\text{FK}}$, audience_id$^{\text{FK}}$}

To ensure our design does not allow any anomalies we need to determine if the database is in BCNF. By definition, for all relations in the database, the relation with FDs F is in BCNF if for every $X \to Y$ in F: (i) $Y \subseteq X$ (the FD is trivial), or (ii) X is a key. The following are all functional dependencies with our design.

- **Library:**
  library_name, address → phoneNumber, email
- **Item:**
  item_id → category_id, library_name, address, title, status, genre, location, publisher_name
- **ItemCategory:**
  category_id → category_name
- **Reading:**
  item_id → isbn, author
- **Music:**
  item_id → artist, num_songs
- **User:**
  user_id → first_name, last_name, birthdate, phone_number, address, email, member_since, favourite_genre
- **Employee:**
  user_id → job_title, salary
- **Events:**
  event_id → library_name, address, room_number, description, even_date, start_time, end_time
- **Audience:**
  audience_id → type, genre
- **Borrows:**
  user_id, item_id, borrow_date → due_date, return_date, fine
- **Attending, RecommendedFor, BelongsTo:**
  Relations has trivial functional dependencies

Given the functional dependencies of all relations above, our schema is in BCNF because for every FD we have a candidate key on the left-hand side. Additionally, since our design is already in BCNF, no decomposition was required. Meaning that losslessness and dependency preservation was maintained.