SOEN 363

Data Systems For Software Engineers

Database Project Report

Phase I

Hrag Bankian (40245363)

Craig Kogan (40175780)

Jack Spiratos (40210312)

Nicholas Werugia (40131956)

Concordia University

Winter 2024

April 4, 2024

The topic that our team has decided to work on is ‘songs and podcasts’. MySQL is the database implementation platform that we decided to use.

**Overview of the System**

The SQL code provided in ‘medias.sql’ outlines the database schema for a system handling media of type ‘**songs**’ and ‘**podcasts**’. Here is an analysis of each component and their functionality.

1. **Parent Table - media\_types:**
   * This table serves as the parent table for all media types in the system.
   * It contains a unique **media\_id** as the primary key and a **media\_type** field to describe the type of media (e.g., song, podcast).
2. **Child Table - songs:**
   * Represents the child table for songs, inheriting from the **media\_types** table (ISA relationship).
   * Contains the following attributes: **song\_id**, **song\_name**, **artist\_name**, **album**, and **duration**.
   * Utilizes referential integrity with a foreign key constraint referencing **media\_types(media\_id)**.
3. **Child Table - podcasts:**
   * Represents the child table for podcasts, also inheriting from the **media\_types** table.
   * Contains the following attributes: **podcast\_id**, **podcast\_name**, **publisher**, **overview**, and **total\_episodes**.
   * Maintains referential integrity with a foreign key constraint referencing **media\_types(media\_id)**.
4. **Weak Entity Table - available\_markets:**
   * Represents a weak entity related to podcasts, storing available markets for each podcast.
   * Composite primary key (**podcast\_id**, **market\_name**) ensures uniqueness for each market per podcast.
   * Includes a foreign key constraint referencing **podcasts(podcast\_id)** for referential integrity.
5. **Triggers for Referential Integrity:**
   * Two triggers (**songs\_before\_insert\_trigger** and **podcasts\_before\_insert\_trigger**) ensure referential integrity during insert operations.
   * They validate that the **song\_id** or **podcast\_id** being inserted in their respective tables exists in the **media\_types** table, preventing invalid references.

A diagram of a company

Description automatically generated

**Analysis:**

* **Normalization:** The database design follows normalization principles by separating media types into distinct tables (**songs** and **podcasts**) while maintaining a central repository in **media\_types**.
* **Data Integrity:** Foreign key constraints and triggers enforce referential integrity, ensuring that only valid **media\_id** values can be inserted into child tables.
* **Flexibility:** The use of inheritance (ISA relationship) allows for extensibility, making it easy to add new media types in the future.
* **Weak Entity Handling:** The inclusion of a weak entity table (**available\_markets**) demonstrates consideration for complex relationships within the system.

As can be seen in the “import\_data.py” Python file, in order to populate the database, Spotify’s and YouTube Music’s APIs have been used.

**Approach For Populating The Database:**

1. **API Selection:** We chose to use Spotify's and YouTube Music's APIs for data population due to their extensive music and podcast libraries, along with robust developer tools.
2. **Data Retrieval:** Utilized endpoints provided by the APIs to retrieve relevant data such as song titles, artists, album information, genres, release dates, popularity metrics, and podcast details like titles, hosts, episode titles, descriptions, and publication dates.
3. **Data Processing:** Implemented data processing techniques to clean and format the retrieved data for consistency and usability within our database schema.
4. **Database Population:** Developed a script to efficiently populate our database with the processed data, ensuring data integrity and relational coherence within the database schema.

**Challenges Faced In Populating The Database:**

1. **API Rate Limits:** Both Spotify and YouTube Music APIs impose rate limits on API calls, requiring us to implement efficient data retrieval strategies.
2. **Missing Data and Errors:** Dealing with missing or incomplete data entries from the APIs and handling errors gracefully to prevent data loss or corruption in the database.