CS4416 Database system project

Christoffer Näs, Jules Espinoux, Jeremy Nicolas, David Chuntishvili

December 1, 2024

Contents

1 Contribution

All work was done together and distributed equally.

2 Platform

Platform used for the project was XAMPP on Windows 11.

3 Modifications to schema

This project aimed to improve the existing database to better handle the details of artists, albums, songs, concerts, and fans interactions. In the updated schema, each table has a unique primary key to make sure every record is distinct and can be quickly found. This helps keep the database accurate and organized.

Originally, the schema had direct relationships, like linking artists straight to albums, which made it hard to show multiple artists working on the same album. To fix this, we added new tables like "artist_album_song_link" that allow for many artists to be connected to many albums. This change makes the database more flexible and true to how music albums are actually made.

This table also let us show that multiple artists can work on one song and that many songs can be played at one concert. These updates help the database better reflect the teamwork and dynamics of music production and concerts.

Updating the fan data was another key improvement. Before, fan details were directly put into the

”concert_tickets” table, leading to unnecessary duplication and mistakes. The new schema uses a ”fans” table and a ”fan_tickets_link” to connect fans with their tickets, covering situations where tickets might be shared among several fans.
The ”fan_favorites” table is a new addition that allows fans to have multiple favorite artists, which was not possible with the old schema.
Furthermore, we added a ”notifications” table to support database triggers that might alert fans about new concerts or releases from their favorite artists. This setup prepares the database for future features that improve how fans interact with artists.
Together, these changes enhance the database’s functionality and efficiency, making sure it can handle a wider range of real-life situations.

4 Entity Relationship Diagram

Our ERD shows how the data in our system is organized and connected. It maps out the key parts of the data, like entities, their details, and how they relate to each other. It’s a clear plan for how the data fits together, without getting into how it’s physically stored in the database.

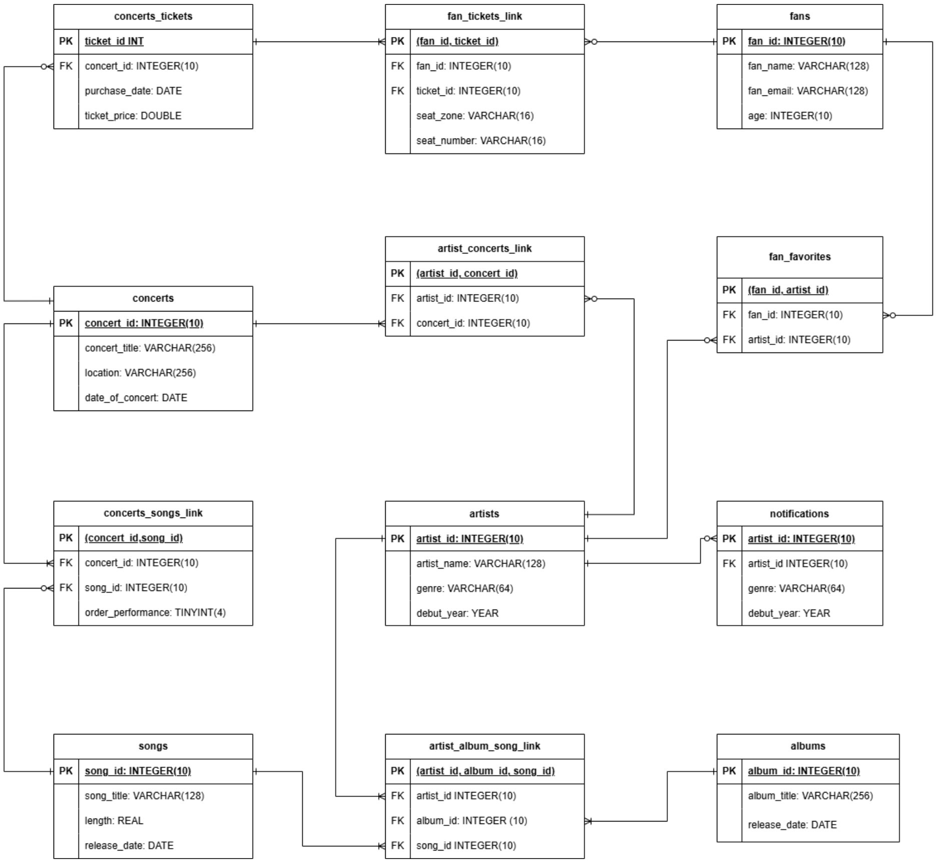

Figure 1: The logical ERD of the modified_conserts database.

5 View and triggers

5.1 View

The view concerts_over_one_place_summary summarizes concerts with more than one ticket sold, showing the concert title, total songs performed, total song duration, and tickets sold. Data comes from several tables: concerts, concerts_songs_link, songs, concerts_tickets, and artist_concerts_link. The query uses JOINs to connect these tables by IDs, groups the data by concert title, and calculates totals for songs, duration, and tickets. It filters out concerts with one or fewer tickets sold and sorts results by tickets sold.

5.2 Triggers

The trigger fan_deletion makes sure so all the links connected to a fan is removed before a record in the fan table is removed making sure that all references to the fan is removed on deletion. The second trigger notify_artist_on_new_concert works in the following way, after signing a new artist to a concert and adding it to the database, an automatic notification record is created in the database that could be used to send out to the artist or to all of his fans.

6 Function and procedures

6.1 total_nr_of_occupied_seats function

This function returns the number of occupied seats for a given concert. To do this, the function takes all the tickets associated to a concert and count the number of fan (and then seat) associated with each ticket.

6.2 add_song_album_link procedure

This procedure checks whether a given song is associated with a given album. To do this, we do the assumption that a song stored in the database will always have an associated value in the table artist_album_song_link. Then we can retrieve the artist associated to the song and add the link if it's not already in the table.

7 Index requirement for efficient execution

Introducing indexes to tables and columns where data retrieval is performed can improve query execution time since data is localized faster. However introducing indexes on tables make modification of the database slower since all the indexes would need to be updated on modification.

7.1 Indexes for the triggers

For trigger fan_deletion, using indexes would make the execution of the trigger slower since it is modifying the database by performing deletion of rows.

For the trigger notify_artist_on_new_concert, using indexes for table concerts and column date_of_concert would optimize our query retrieval time since we are retrieving rows from this column in the trigger. However introducing indexes for the table notifications would slower our deletions on the table notifications and also weaken our trigger execution time since modifications on the database is performed on insertion of rows.

7.2 Indexes for the function

Introducing indexes for table concerts_tickets and column ticket_id would improve the function total_nr_of_occupied_seats execution time since it is retrieving data from this column.

7.3 Indexes for the procedure

To improve execution time for procedure add_song_album_link, indexes could be added to table songs and the column release_date since the columns is only used for data retrieval. For the other columns used the rows are also modified so there is a trade off to think about. The procedure is more read heavy so introducing indexes for column release_date in albums, column song_id, album_id in artist_album_song_link is probably worth it.