CS4416 - Report

Group member:

Yiming Li 23039248

Monika Wohlfarth 24239526

Kirtika Thakur 23241233

Ryan Pendergast

Sec.1 Contributions

Yiming Li :

1. Generated the report and normalized its frame and format.
2. Completed **Sec.3 Task 1 + Sec.7 Discussion of the Report**
3. Completed **modified_concerts_schema.sql** + **data.sql**

Monika Wohlfarth:

1. Created ERD diagram
2. Wrote ERD diagram overview

Kirtika Thakur

1. Completed task 5 and 6
2. Completed sec.6 of the report

Sec.2 Platform Used

- IntelliJ

Sec.3 Task 1

1. Support for Multiple Artists per Album, Song, and Concert

Modifications:

- **Added `album_artists` table**:
  - Schema: `(album_id, artist_id)`
  - Allows associating multiple artists with a single album. The composite primary key ensures that an artist cannot be linked to the same album more than once.
- **Added `song_artists` table**:
  - Schema: `(song_id, artist_id)`
  - Allows multiple artists to be credited for a single song, even for different recordings/versions (distinct `song_id`s).
- **Added `concert_artists` table**:
  - Schema: `(concert_id, artist_id)`
  - Supports associating multiple artists with a single concert, allowing collaborative performances.

Reason:

- The original schema assumed a one-to-many relationship (e.g., a single `artist_id` in the `albums`, `songs`, and `concerts` tables). By introducing these many-to-many junction tables, the schema now supports more realistic scenarios such as:
  - Collaborative albums (e.g., two or more artists releasing an album together).
  - Joint songs or remixes with multiple contributing artists.
  - Concerts with multiple performers.

2. Support for Multiple Favorite Artists per Fan

Modification:

- **Added `fan_favorite_artists` table**:
  - Schema: `(fan_id, artist_id)`
  - Allows fans to mark multiple artists as their favorites. The composite primary key prevents duplicates.

Reason:

- The original schema assumed each fan had only one favorite artist (`favourite_artist_id` in `concert_tickets`), which was restrictive. Fans often admire multiple artists, so this change makes the system more flexible and realistic.

3. Support for Multiple Fans per Ticket

Modification:

- **Added `ticket_fans` table**:
  - Schema: `(ticket_id, fan_id)`
  - Links multiple fans to a single ticket.

Reason:

- The original schema tied `ticket_id` directly to a single `fan_id`, preventing shared tickets.

4. Optimization of the schema

Modifications:

- Removed redundant fields from the `concert_tickets` table:
  - Removed `fan_name`, `fan_email`, and `age`, and stored in the `fans` table.
- Removed `artist_id` from `albums`, `songs`, and `concerts` tables:
  - These fields were replaced with their respective many-to-many tables (`album_artists`, `song_artists`, `concert_artists`).

Reason:

- Avoids redundancy and potential inconsistencies. For example:
  - If a fan's name or email changes, it should not require updating multiple rows across tables.
  - If multiple artists are linked to an album, storing only a single `artist_id` would lose information about other contributors.

5. Addition of Primary Keys and Foreign Keys for Each Table

Modifications:

- Added primary keys and foreign keys where appropriate for all tables

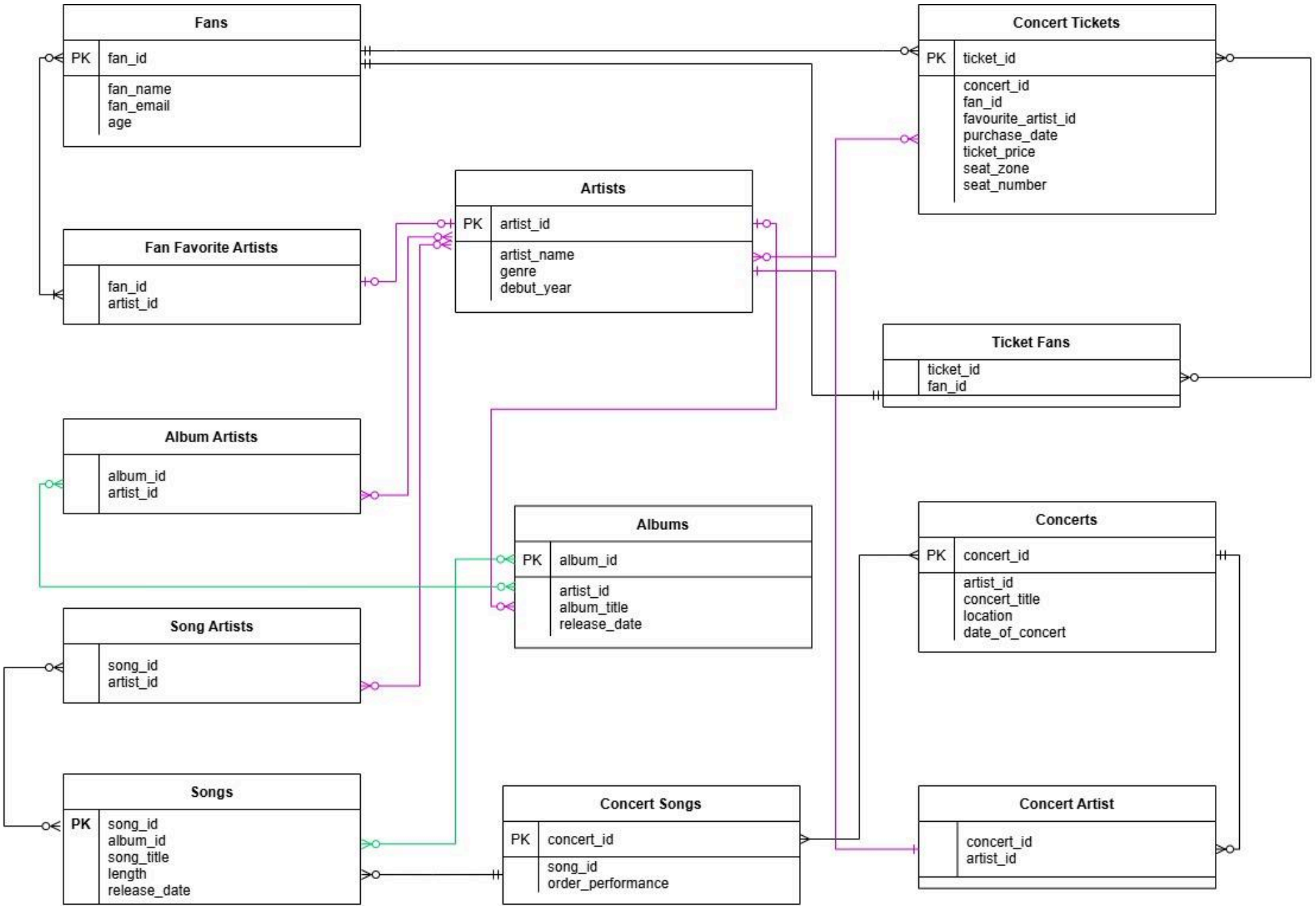
Reason:

- to ensure data integrity and establish clear relationships between entities.

Sec.4 Task 2: ERD diagram

Relationships

1. **Fans and Concert Tickets**
    - Relationship: A fan can purchase multiple concert tickets, and each ticket belongs to one fan.
    - Cardinality: 1:N
2. **Fans and Fan Favorite Artists**
    - Relationship: A fan can have multiple favorite artists, and each artist can be a favorite for multiple fans.
    - Cardinality: M:N
3. **Artists and Albums**
    - Relationship: An artist can create multiple albums, but each album belongs to one artist.
    - Cardinality: 1:N
4. **Albums and Songs**
    - Relationship: An album contains multiple songs, but each song belongs to one album.
    - Cardinality: 1:N
5. **Artists and Songs (via Song Artists)**
    - Relationship: An artist can perform multiple songs, and each song can be performed by multiple artists.
    - Cardinality: M:N
6. **Concerts and Artists (via Concert Artist)**
    - Relationship: A concert can feature multiple artists, and each artist can participate in multiple concerts.
    - Cardinality: M:N
7. **Concerts and Concert Songs**
    - Relationship: A concert can include multiple songs, and each song can be performed at multiple concerts.
    - Cardinality: M:N
8. **Concert Tickets and Concerts**
    - Relationship: A ticket is for one concert, but a concert can have multiple tickets sold.
    - Cardinality: 1:N
9. **Concert Tickets and Fans (via Ticket Fans)**
    - Relationship: A ticket can be purchased by multiple fans, and a fan can buy multiple tickets.
    - Cardinality: M:N


Fans
PK
fan_id
fan_name
fan_email
age
Concert Tickets
PK
ticket_id
concert_id
fan_id
favourite_artist_id
purchase_date
ticket_price
seat_zone
seat_number
Artists
PK
artist_id
artist_name
genre
debut_year
Fan Favorite Artists
fan_id
artist_id
Ticket Fans
ticket_id
fan_id
Album Artists
album_id
artist_id
Albums
PK
album_id
artist_id
album_title
release_date
Concerts
PK
concert_id
artist_id
concert_title
location
date_of_concert
Song Artists
song_id
artist_id
Songs
PK
song_id
album_id
song_title
length
release_date
Concert Songs
PK
concert_id
song_id
order_performance
Concert Artist
concert_id
artist_id

Sec.5 Task 3 & 4

Task 3 Assumptions

Table Structure - All of concerts, concert_artists, and tickets tables currently exist with specific columns by which include artist_id, the price, and concert_id.

Relationship Joins - Through foreign keys, it ensures data integrity and the joins being accurate.

Calculation of Revenue - The price column in the tickets table represents the cost of a single ticket, this helps calculate concert revenue for the equation.

Logic Filter - The HAVING command gives a threshold of greater than 900 becoming a filter for the data.

Task 4 Assumptions

Table Structure - The price column inside the tickets table exists so that ticket_logs can store logs with object such as message, log_date, and ticket_id

Price Enforcement - The BEFORE trigger is meant to make up and assumes that the ticket values to not be less than 10 for its pricing.

Logging Logic - The AFTER trigger makes sure that a new ticket should be logged every single time with a timestamp.

Data Integrity - Makes sure that all the tickets are valid, not null, and properly put to be linked towards concerts so there will be no errors.

Sec.6 Task 5 & 6

Task 5 Assumptions:

Table Structure: It is assumed that the concert_tickets table exists and has the column concert_id.

Data Integrity: It is assumed that the concert_id input to the function exists in the concert_tickets table. If it does not, the function will return 0, which is considered a valid output.

Single Concert Context: The function is designed to count tickets for a single concert.

Deterministic Behavior: The function is marked as DETERMINISTIC, indicating that it will consistently produce the same output for a given input, which aligns with its intended design.

Task 6 Assumptions:

Table Structure: It is assumed that both the songs and albums tables exist.

Single Update Context: The procedure assumes that each song can be associated with only one album at a time. If the song is already linked to the specified album, no changes will be made.

Release Date Logic: The procedure assumes that a song's release date must not exceed the release date of its associated album, reflecting a business rule that songs should be released on or before the album's release date.

Sec.7 Discussion

1. Index on Foreign Keys

Artists-Songs-Album Table:

```
CREATE INDEX idx_artist_id ON artists_songs(album_id, song_id);
CREATE INDEX idx_album_id ON album_songs(album_id);
CREATE INDEX idx_song_id ON songs(song_id);
```

Use Case:

- Querying songs by artist or album.
- Joining `songs`, `albums`, and `artists` tables to retrieve songs by a specific artist or album.

Disadvantage:

- **Write Operations (Insert, Update, Delete)**: Foreign key indexes can slow down writes because the system needs to update the index when the underlying data changes.
- **Disk Space Usage**: Indexing foreign key columns on multiple tables could increase disk space usage, especially for large datasets.

2. Index for Many-to-Many Relationships (Concerts and Tickets)

Concerts and Tickets:

```
CREATE INDEX idx_concert_id ON tickets(concert_id);
CREATE INDEX idx_ticket_id ON tickets(ticket_id);
```

Use Case:

- Efficient querying of tickets purchased for a specific concert (`WHERE concert_id = X`).
- Quick lookups of tickets purchased by fans (`WHERE ticket_id = X`).

Disadvantage:

- **Insertion Overhead**: When new tickets are sold, the index will need to be updated. This can slow down inserts, especially if many tickets are being sold simultaneously.