Overview of the System:

1. Data Sources:

   - The system utilizes two external APIs, namely RAWG API and IGDB API, to fetch information about games.
   - The RAWG API is used to obtain general information about games, such as name, platforms, release date, and genre.
   - The IGDB API is used to retrieve additional data, including ratings and summaries.

2. Data Processing:

   - The system consists of three main Python scripts: ***Rawg.py***, ***IGDB.py***, and ***main.py***.
   - ***Rawg.py*** fetches game data from the RAWG API and stores it in a JSON file (***games_data.json***).
   - ***IGDB.py*** uses the obtained RAWG data to make requests to the IGDB API, fetches additional data, and merges it with the RAWG data. The merged data is then stored in another JSON file (***merged_data.json***).
   - ***main.py*** reads the merged data and inserts it into a MySQL database.

3. Database Design:

   - The MySQL database is named **GAMES** and consists of several tables: **Games**, **ActionGames**, **Platforms**, **Genres**, **GamePlatforms**, and **GameGenres**.
   - Views (**ActionGamesView** and **GameDetailsView**) provide different perspectives on the game data.
   - The database also includes triggers to enforce constraints, such as not allowing the insertion of games with a release date more than two years in the future.

4. Queries and Views:

   - The system includes various types of SQL queries, including basic selects, group by, joins, set operations, correlated subqueries, and views.
   - Views, such as ActionGamesView and GameDetailsView, simplify the retrieval of specific information from the database.

5. Constraints:

- The database includes constraints to ensure data integrity, such as primary key constraints, foreign key relationships, and custom triggers for specific conditions.

Data Model: Using MySQL

The data model involves several entities, including:

Games: Basic information about each game.
Genres: Different genres that games can belong to.
Platforms: Various gaming platforms.
GameGenres: Associative table linking games to genres.
GamePlatforms: Associative table linking games to platforms.
ActionGames: A specialized table indicating games classified as "Action."
Views: Provide simplified and specific perspectives on the data.

Approach and Challenges:

1. **API Integration:**

- Fetching data from two different APIs (RAWG and IGDB) required coordinating requests and merging responses.
- Handling rate limits and potential API changes was a consideration.

2. **API Authentication and Authorization:**

- Managing authentication tokens, API keys, and OAuth tokens posed challenges during the interaction with external APIs.
- Understanding and correctly implementing the authentication and authorization processes for both RAWG and IGDB APIs was crucial for secure API interactions.
- As first-time users, overcoming challenges related to API authentication, authorization, and token management required additional effort and learning.

3. **Data Merging:**

    - Merging data from two different sources required careful mapping and handling of potential missing or conflicting information.

4. **Database Population:**

    - The Python script main.py reads the merged data and inserts it into the MySQL database.
    - Handling relationships and ensuring data consistency posed challenges, especially with the insertion of genres, platforms, and associated tables.

5. **Database Design and Constraints:**

    - Designing an effective database structure to represent the relationships between games, genres, and platforms.
    - Implementing constraints, such as the trigger to prevent inserting games with release dates more than two years in the future.

6. **Data Integrity:**

    - Ensuring data integrity through proper use of foreign keys, unique constraints, and avoiding duplicate entries.

The process of populating the game database faced challenges related to data migration, validation, handling duplicate entries, and managing API authentication. These challenges provided valuable learning opportunities, emphasizing the importance of robust data management practices, clear documentation, and continuous training for the development team. By addressing these issues, the system can maintain data accuracy, integrity, and security throughout its lifecycle.