Application Domain

Group 50
Michael Chang
Jet Li
Justen Jiang
Owen Deng

December 18, 2025

1 Introduction (Phase 1)

Our database project is a video game and user database that stores data on every user and the games they play. As far as languages, we anticipate that we will use java and Datagrip. We are unsure about what framework we will use, if any, at this stage but will continue to look into it. For our tentative roles, they are listed below.
Video game data Manager - Justen Jiang
User data Manager - Michael Chang
Project leader - Jet Li
Tester - Owen Deng

When creating the ER diagram, we decided to take to take some attributes out and make it its own sub entity with a relationship to the Video Games entity. (Specialization). This was done not only to reduce complexity in the diagram but also to increase flexibility. For example, making Genre its own entity allows a video game to have an unlimited number of genres and genres to have multiple video games associated with it, which can also be helpful when selecting a video game based on genre in the future. Making Contributor its own entity allowed us to classify a contributor as publisher and or developer with two relationships. This is because a contributor can be a developer, publisher, or both, so having two relationships connecting to

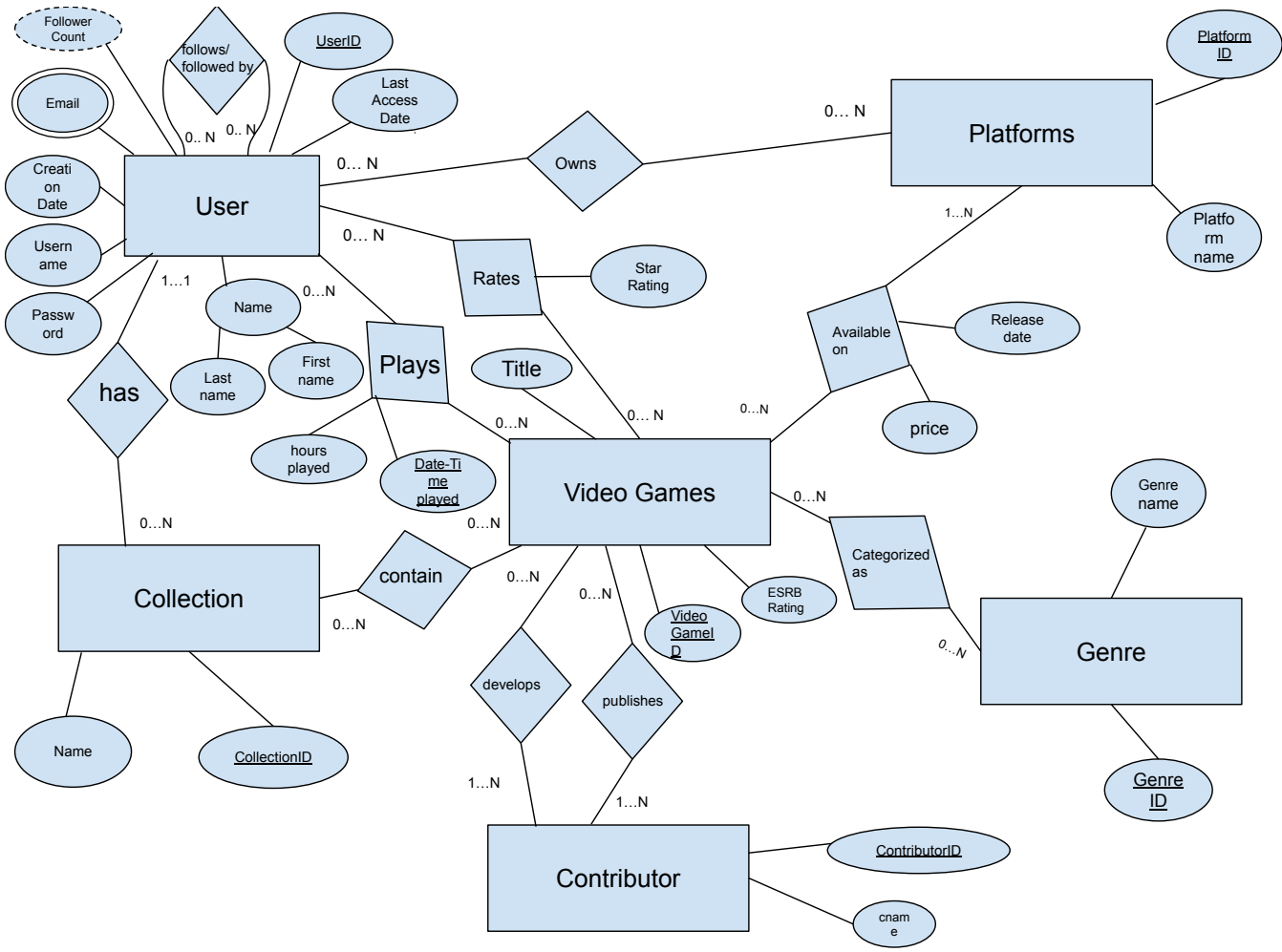

Figure 1: ER Diagram

one entity with an id and name creates a clear and well-organized relation. And finally, we made platforms its own separate entity because platforms has its own attributes and because a video game can have multiple platforms, similar to how they can have multiple genres and contributors.

1.1 Reduction to tables

User(*User*ID, creation_date, username, password, last_access, first_name, last_name)
Follows(*UserID*, *FollowerID*)
PlatformsOwned(*UserID*, *PlatformID*)
UserEmail(*UserID*, *email*)

VideoGame(VideoGameID, title, ESRB)
Platform(PlatformID, platform_name)
Contributor(ContributorID, contributor_name)
Genre(GenreID, genre_name)
Collection(CollectionID, collection_name)

UserHas(*CollectionID*, *UserID)*
VideoGameIsIn(*CollectionID*, *VideoGameID*)
UserPlays(*VideoGameID*, *UserID*, *date_time_played*, hours_played)
UserRates(*VideoGameID*, *UserID*, rating)
ContributorDevelops(*ContributorID*, *VideoGameID*)
ContributorPublishes(*ContributorID*, *VideoGameID*)
VideoGameOnPlatform(*PlatformID*, *VideoGameID*, release_date, price)
VideoGameInGenre(*GenreID*, *VideoGameID*)

To reduce the ERD to tables, each data entity should be it's own table and any relationship that the entity is in should have its primary key as a foreign key. For a unary relation, we renamed one of the attributes to "FollowerID" for clarity instead of UserID. If a relation has attributes, it should also be in its corresponding table, and for Plays specifically, date-time played should be a key so each time a user plays, it can be recorded. Multi-valued attributes, like Email for user, should be its own relation with both UserID and email as a super key, so that a single user can have multiple emails.

1.2 Data Requirements/Constraints

User:
- UserID: integer type and primary key, required
- last access date: date type
- email: variable character type of length 50, required
- Creation date: date type, required
- Username: variable character type of length 30, required
- Password: variable character type of length 50, required
- last name: variable character type of length 20, required
- first name: variable character type of length 20, required

Collection:
- CollectionID: integer type and primary key, required
- Name: variable character type of length 20, required

Video Games:
- CollectionID: integer type and primary key, required
- Title: variable character type of length 30, required
- ERSB Rating: variable character type, required

Genre:
- GenreID: integer type and primary key, required
- Name: variable character type of length 20, required

Platforms:
-PlatformID: integer type and primary key, required
- Name: variable character type of length 20, required

Contributor:
- ContributorID: integer type and primary key, required
- Name: variable character type of length 30, required

User/Video Games (Rates relationship):
- Star Rating: integer type
- Rates is unique
User/Video Games (Plays relationship):
- Date played: date type, required

- Time played: time type, required

User/Platforms (Owns relationship):

Video Games/Platforms (Available On relationship):
- Release date: date type, required
- Price: float, required

User/Collection (Has relationship):

Contributor/Video Games (Develops relationship):

Contributor/Video Games (Publishes relationship):

Video Games/Genre (Categorized as relationship):

1.3 Sample instance data

Entities

User

```
(User_ID, username, email, first_name, last_name, creation_date, last_access_date,
1, "nova", "nova@gmail.com", "Ava", "Nguyen", 2025-01-05, 2025-09-20, {PC, PS5}
2, "pixelpete", "pete@yahoo.com", "Peter", "Lopez", 2025-01-10, 2025-09-21, {Switc
3, "z3n", "zhen.li@hotmail.com", "Zhen", "Li", 2025-01-12, 2025-09-18, {PC}
4, "sammyG", "sam.gonzalez@gmail.com", "Sam", "Gonzalez", 2025-01-15, 2025-09-22,
5, "mika", "mika.khan@outlook.com", "Mika", "Khan", 2025-01-18, 2025-09-22, {PS5}
```

Video Games

```
(Video_Game_ID, title, ESRB_rating)
1, "Elden Ring", "M"
2, "Stardew Valley", "E10+"
3, "Mario Kart 8 Deluxe", "E"
4, "Hades", "T"
5, "Celeste", "E10+"
```

Platforms

```
(Platform_ID, name)
1, "PC"
2, "PS5"
3, "Xbox Series"
4, "Nintendo Switch"
5, "Mac"
```

Genre

```
(Genre_ID, name)
1, "Action RPG"
2, "Farming Sim"
3, "Racing"
4, "Rogue-like"
5, "Platformer"
```

Contributor

```
(Contributor_ID, name)
1, "FromSoftware"
2, "ConcernedApe"
3, "Nintendo"
4, "Supergiant Games"
5, "Matt Makes Games"
```

Collection

```
(Collection_ID, name)
1, "Backlog"
2, "Favorites"
3, "Co-op Night"
4, "Indies"
5, "Racing Only"
```

Relationship Types

Plays (User - Video Games)

```
1, 1, 2025-09-15, 120
2, 3, 2025-09-14, 45
3, 2, 2025-09-12, 90
4, 4, 2025-09-10, 75
5, 5, 2025-09-11, 60
```

Rates (User - Video Games)

```
1, 1, 5
2, 3, 3
3, 2, 5
4, 4, 4
5, 5, 5
```

Has (User - Collection)

```
1, 1
1, 2
2, 3
4, 4
5, 5
```

Follow (User - User)

```
3, 1
5, 2
10, 3
0, 2
2, 2
```

Is in (Collection - Video Games)

```
1, 4
1, 5
2, 1
3, 3
4, 2
```

Develops (Contributor - Video Games)

```
1, 1
2, 2
3, 3
4, 4
5, 5
```

Publishes (Contributor - Video Games)

```
1, 1
2, 2
3, 3
4, 4
5, 5
```

Is (Video Games - Genre)

```
1, 1
2, 2
3, 3
4, 4
5, 5
```

On (Video Games - Platforms)

```
1, 1, 2022-02-25, 59.99
2, 4, 2017-10-05, 14.99
3, 4, 2017-04-28, 59.99
4, 1, 2020-09-17, 24.99
5, 1, 2018-01-25, 19.99
```

2 Implementation (Phase 2)

We decided to use Java for our DML, and PostgreSQL was given as our DDL. Initially, we populated the tables with .csv files generated from Chat-GPT. Occasionally, there would be inconsistencies and some constraints that were not met, so we did a lot of inserting data manually via our application.

2.1 Updated EER Diagram

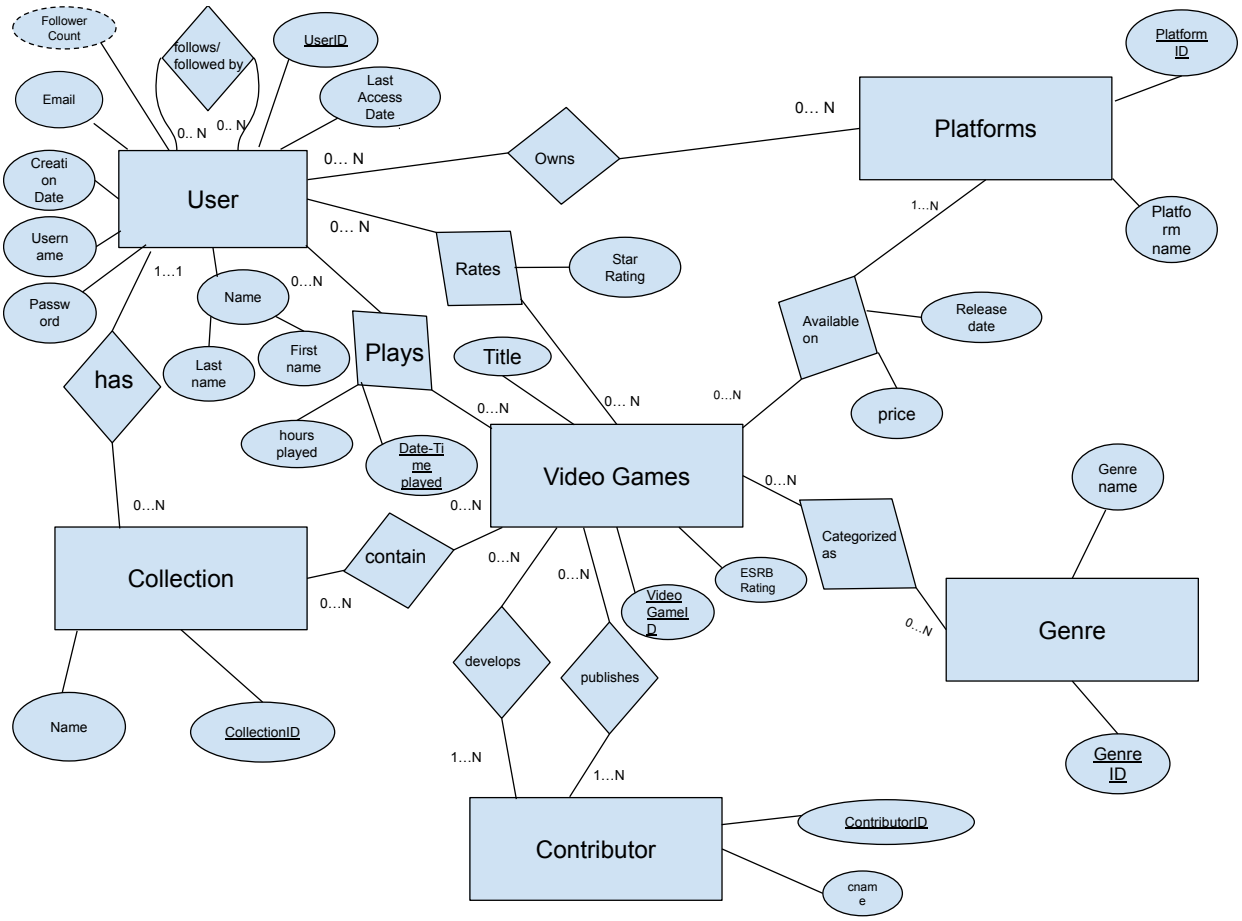

Figure 2: ER Diagram

2.2 Updated Reduction to tables

User(*User*ID, creation_date, email, username, password, last_access, first_name, last_name)
Follows(*UserID*, *FollowerID*)
PlatformsOwned(*UserID*, *PlatformID*)

VideoGame(VideoGameID, title, ESRB)

Platform(PlatformID, platform_name)
Contributor(ContributorID, contributor_name)
Genre(GenreID, genre_name)
Collection(CollectionID, collection_name)

UserHas(*CollectionID*, *UserID)*
VideoGameIsIn(*CollectionID*, *VideoGameID*)
UserPlays(*VideoGameID*, *UserID*, *date_time_played*, hours_played)
UserRates(*VideoGameID*, *UserID*, rating)
ContributorDevelops(*ContributorID*, *VideoGameID*)
ContributorPublishes(*ContributorID*, *VideoGameID*)
VideoGameOnPlatform(*PlatformID*, *VideoGameID*, release_date, price)
VideoGameInGenre(*GenreID*, *VideoGameID*)

2.3 Sample SQL Statements

```
Sample Table Population Queries
CREATE Table rates_table(
    User_ID INT,
    VIDEO_GAME_ID INT,
    STAR_RATING INT CHECK (STAR_RATING IN (1,2,3,4,5)),
    PRIMARY KEY (User_ID, VIDEO_GAME_ID, STAR_RATING),
    FOREIGN KEY (User_ID) REFERENCES user_table(User_ID),
    FOREIGN KEY (VIDEO_GAME_ID) REFERENCES video_games_table(VIDEO_GAME_ID)
);

Create Table plays_table(
    User_ID INT NOT NULL,
    Video_Game_ID INT NOT NULL,
    Start_Time DATE NOT NULL,
    Session_Minutes float(4) NOT NULL,
    PRIMARY KEY (User_ID, Video_Game_ID, Start_Time),
    FOREIGN KEY (User_ID) REFERENCES user_table(User_ID),
    FOREIGN KEY (Video_Game_ID) REFERENCES video_games_table(Video_Game_ID)
);
```

Sample Insert Queries

User Table:
```
INSERT INTO user_table(user_id, username, email, first_name, last_name, creation_d
INSERT INTO user_table(user_id, username, email, first_name, last_name, creation_d
```

Video Game Table:
```
INSERT INTO video_games_table(video_game_id, title, esrb_rating) VALUES (1, 'Scott
INSERT INTO video_games_table(video_game_id, title, esrb_rating) VALUES (2, 'Valor
```

Rates table:
```
INSERT INTO rates_table(user_id. video_game_id, star_rating) VALUES (1, 1, 1)
INSERT INTO rates_table(user_id. video_game_id, star_rating) VALUES (1, 2, 5)
```

Publishes Table:
```
INSERT INTO publishes_table(contributor_id, video_game_id) VALUES (1, 2)
INSERT INTO publishes_table(contributor_id, video_game_id) VALUES (1, 1)
```

Plays Table:
```
INSERT INTO plays_table(user_id, video_game_id, start_time, session_minutes) VALUE
INSERT INTO plays_table(user_id, video_game_id, start_time, session_minutes) VALUE
```

Platform Table:
```
INSERT INTO platform_table(platform_id, platform_name) VALUES (1020, 'XBox 360')
INSERT INTO platform_table(platform_id, platform_name) VALUES (1021, 'PC')
```

Owns Table:
```
INSERT INTO owns_table(user_id, platform_id) VALUES (1, 2)
INSERT INTO owns_table(user_id, platform_id) VALUES (1, 1234)
```

Has table:
```
INSERT INTO has_table(user_id, collection_id) VALUES (5001, 1)
INSERT INTO has_table(user_id, collection_id) VALUES (5006, 5010)
```

Genre Table:
```
INSERT INTO genre_table(genre_id, name) VALUES (4561, 'hooray')
INSERT INTO genre_table(genre_id, name) VALUES (3874, 'Horror Vol. 2')
```

```
Available On Table:
INSERT INTO available_on_table(video_game_id, platform_id) VALUES (132, 219)
INSERT INTO available_on_table(video_game_id, platform_id) VALUES (1, 2)

Collection Table:
INSERT INTO collection_table(collection_id, name) VALUES (5, 'Jett from Valorant')
INSERT INTO collection_table(collection_id, name) VALUES (12, 'Greatest Visual Nov

Categorized Table:
INSERT INTO categorized_as_table(video_game_id, genre_id) VALUES (17, 67)
INSERT INTO categorized_as_table(video_game_id, genre_id) VALUES (67, 67)

Contains Table:
INSERT INTO contains_table(collection_id, video_game_id) VALUES (2, 69)
INSERT INTO contains_table(collection_id, video_game_id) VALUES (903, 999)

Contributor Table:
INSERT INTO contributor_table(contributor_id, name) VALUES (1, 'Mojang')
INSERT INTO contributor_table(contributor_id, name) VALUES (10, 'Bandai Namco')

Develops Table:
INSERT INTO develops_table(contributor_id, video_game_id) VALUES (2849, 173)
INSERT INTO develops_table(contributor_id, video_game_id) VALUES (2114, 2526)

Follows Table:
INSERT INTO follows_table(user_id, follower_id) VALUES (67, 70)
INSERT INTO follows_table(user_id, follower_id) VALUES (137, 4339)
```

3 Data Analysis and Recommendation System (Phase 3)

3.1 Hypothesis

The objectives of our data analysis was to find trends for players. Specifically, our simple hypothesis was people play video games for more hours on weekends than on weekdays. Our reasoning for this is that the regular school and work schedule is from Monday to Friday,

and the weekend days are normally days off. On days off, people would have more time for leisure activities, including video games. As a result, we hypothesized that people would play video games for more hours on weekends.

Then, for our complex hypothesis, we hypothesized that people would play more horror games during the fall and winter months rather than during the spring and summer months. We thought that the longer nights and colder weather during the fall and winter months would not only keep people indoors, but also create a stronger atmosphere for horror themes, thereby encouraging players to spend more time on horror games.

3.2 Data Preprocessing

Since our data was randomly generated, it doesn't really make sense to remove outliers, and there were no missing values to fill in. Our genre names were not the best, so we did have to use a more complex query to get all games that have the word "horror" in its genre name, but that's the extent of the preprocessing we did. We just used one complex query for each hypothesis.

3.3 Data Analytics & Visualization

To analyze the data, we first used the query console to select the relevant data, then using Excel and Google Sheets, we created corresponding visuals. Since our database was not that complex, all of the preprocessing was done in one query (SQL Appendix). Minimal processing was done after for our complex hypothesis only because we wanted more flexible data for our visuals. For the data visualization, we used Excel and Google Sheets, choosing an appropriate graph.

3.4 Indexing

We didn't use any indexes because our dataset was small enough that queries ran quickly without them. If we needed an index, it would have been for our recommendation system, specifically the

recommendation by similar_users feature. That part of the system relies on user_id multiple times to lookup user interactions and compare users. Indexing user_id would speed up those repeated lookups and make the recommendation process more efficient as the dataset grows.

3.5 Conclusions

From our data analysis, we found that, on average, users played around the same amount of hours in any given day of the week. We were expecting players to have more play hours during the weekends, but there is no correlation between the weekday and hours played.

For our complex hypothesis, we expected players to spend more time playing horror games during the colder seasons (Fall and Winter), but we found the opposite. Players played less horror games during the colder seasons, instead opting to play more during the warm seasons.

3.6 Recommendation System

For our recommendation system, we made 3 algorithms; recommend by genre, developer, or similar players.

Our genre recommendations start with selecting your #1 genre by the time played and number of games played with that genre. To give a little more weight to the amount of games with a genre, we raised it to the 1.8 power, as n^2 would grow too fast relative to our database size.
To find the games, we multiply the average rating with ln(play count) in the genre, not in your play history.

For developer, the process is basically the same as above, but games have to be developed by your top developer.

If you are recommending by similar players, we first got the history and time played from your play history, then we found every other user who shared atleast 1 game from your play history, giving them a score based on the amount of time overlap in any given game. Then we summed up all the scores to find the most similar players. Finally we selected

games not owned by you that similar players played.

4 Lessons Learned

Working with a team makes it difficult to work on the same codebase at the same time. During development, we frequently ran into merge conflicts and struggled to resolve them. However, as time went on, we learned to work on different sections at a time and frequently git push and git pull to avoid more merge conflicts.

5 Resources

We did not use any extra resources.

6 SQL Appendix

```
Query for simple hypothesis:
SELECT day_of_week, total_hours
FROM (
    SELECT
        TRIM(TO_CHAR(start_time, 'DAY')) AS day_of_week,
        SUM(session_minutes) / 60.0 AS total_hours,
        EXTRACT(DOW FROM start_time) AS dow_number
    FROM plays_table
    GROUP BY day_of_week, dow_number
) AS sub
ORDER BY
    CASE dow_number
        WHEN 1 THEN 1
        WHEN 2 THEN 2
        WHEN 3 THEN 3
        WHEN 4 THEN 4
        WHEN 5 THEN 5
        WHEN 6 THEN 6
        WHEN 0 THEN 7
    END;
```

```
Query for complex hypothesis:
WITH horror_games AS (
    SELECT v.video_game_id
    FROM video_games_table v
             JOIN categorized_as_table c ON c.video_game_id = v.video_game_id
             JOIN genre_table g ON g.genre_id = c.genre_id
    WHERE g.name ILIKE '%horror%'
)
SELECT
    EXTRACT(MONTH FROM p.start_time) AS month_num,
    p.session_minutes
FROM plays_table p
         JOIN horror_games h ON h.video_game_id = p.video_game_id
ORDER BY month_num, session_minutes

Recommendation by similar users:
    WITH target_vector AS (
        SELECT video_game_id,
               session_minutes AS tgt_min
        FROM   plays_table
        WHERE  user_id = ?),
    similar_users AS (
        SELECT p.user_id,
               SUM(LEAST(p.session_minutes, t.tgt_min)) AS sim_weight
        FROM   plays_table      p
        JOIN   target_vector    t USING (video_game_id)
        WHERE  p.user_id <> ?
        GROUP  BY p.user_id
        HAVING SUM(LEAST(p.session_minutes, t.tgt_min)) >= 0),
    recs AS (
        SELECT v.video_game_id,
               v.title,
               SUM(p.session_minutes * s.sim_weight) AS weighted_minutes
        FROM   similar_users      s
        JOIN   plays_table        p ON p.user_id = s.user_id
        JOIN   video_games_table  v ON v.video_game_id = p.video_game_id
```

```
        WHERE  NOT EXISTS (SELECT 1
                           FROM   plays_table pp
                           WHERE  pp.user_id       = ?
                             AND  pp.video_game_id = p.video_game_id)
        GROUP  BY v.video_game_id, v.title)
        SELECT video_game_id, title, weighted_minutes
    FROM recs
    ORDER BY weighted_minutes DESC, video_game_id
    LIMIT 10

Recommendations by genre:
SELECT c.genre_id,
   SUM(p.session_minutes) AS total_time,
   COUNT(DISTINCT p.video_game_id) AS genre_count,
  (SUM(p.session_minutes) * power((COUNT(DISTINCT p.video_game_id)), 1.8) ) AS sco
FROM plays_table AS p
JOIN categorized_as_table AS c ON c.video_game_id = p.video_game_id
WHERE p.user_id = ?
GROUP BY c.genre_id
ORDER BY score DESC
LIMIT 1


SELECT v.video_game_id,
   v.title,
   g.play_count,
   COALESCE(r.avg_stars, 0) AS avg_stars,
   (COALESCE(r.avg_stars, 0) * LN(1 + COALESCE(g.play_count, 0))) AS blend
FROM video_games_table AS v
JOIN categorized_as_table AS c ON c.video_game_id = v.video_game_id
LEFT JOIN (
        SELECT video_game_id, COUNT(DISTINCT user_id) AS play_count
        FROM   plays_table
        GROUP  BY video_game_id
) AS g ON g.video_game_id = v.video_game_id
LEFT JOIN (
        SELECT video_game_id, AVG(star_rating) AS avg_stars
        FROM rates_table
```

```
        GROUP BY video_game_id
) AS r ON r.video_game_id = v.video_game_id
WHERE c.genre_id = ?
AND NOT EXISTS (SELECT 1
                FROM plays_table p
                WHERE p.user_id = ? AND p.video_game_id = v.video_game_id)
ORDER BY blend DESC, v.video_game_id
LIMIT 10
```