BÁO CÁO DỰ ÁN CỦA NHÓM 7

Cửa hàng game

CÁC THÀNH VIÊN CỦA NHÓM:

Đinh Huy Dương 20215020

Nguyễn Thanh Nhật Bảo 20210096

Nguyễn Văn Đăng 20215033

I, Mô tả nghiệp vụ

1, Mô tả chung:

- Cửa hàng để người dùng có thể mua và đánh giá game.

- Các nhà phát hành game có thể đăng tải game lên cửa hàng

- Người quản lý sẽ kiểm soát game và người dùng

2, Phân loại người dùng:

a, NGƯỜI DÙNG (Người tải game - USER):

- Cần có tài khoản đăng kí với hệ thống để tải game. Nếu không đăng kí tài khoản thì chỉ có thể xem game đang được bán trên nền tảng.

- Thông tin tài khoản: Các thông tin cơ bản như: Tên, mã ID định danh, mật khẩu email, số điện thoại, số tiền trong tài khoản, và thanh trạng thái (bị cấm hay không cấm)

- Các thao tác trên cửa hàng

+Để mua game, người dùng có thể tìm kiếm tên game mình muốn mua bằng thanh tìm kiếm, hoặc tìm hiểu những tựa game đang xu hướng, được yêu thích dựa trên điểm số đánh giá game, hoặc những tựa game được đề xuất.

+Khi đã chọn được tựa game ưng ý, người dùng thực hiện mua game bằng cách sử dụng ví điện tử của hệ thống. Nếu tài khoản không đủ số dư thì thực hiện nạp thêm vào tài khoản để thanh toán.

+Họ có thể đánh giá game theo số điểm (sao). Game tính trung bình các đánh giá và đưa ra rating chung cho game đấy

+Còn nếu chưa mua thì chỉ có thể xem đánh giá game của người khác.

b, NHÀ PHÁT TRIỂN GAME (DEVELOPER):

- Bao gồm những thông tin: mã ID định danh, tên, email, website (nếu có), ngày thành lập

- Upload game lên trên hệ thống

- Trong thông tin của người nhà phát triển, hệ thống có thể thống kê được doanh thu của các game đã đăng tải với điều kiện các game phải là game trả phí

- Game khi được đưa lên hệ thống sẽ cần phải được duyệt bởi quản trị viên để có thể được nhìn thấy bởi những người dùng thông thường khác.

- Họ cũng có thể thực hiện gỡ game của mình đăng lên nếu muốn.

c, QUẢN TRỊ VIÊN (ADMINISTRATOR):

- Người trong công ty, có khả năng duyệt game đưa lên hệ thống, duyệt cung cấp quyền cho các nhà phát triển game. Bao gồm những thông tin: Mã ID định danh, tên quản trị

- Chỉnh sửa các trạng thái đối với game và người dùng: Cấm, không cấm,…

- Tất cả hành động của Quản trị viên đều sẽ bị ghi lại để quản lý và xử lý khiếu nại.

3, Trò chơi (Game):

- Phân loại game: Sẽ có nhiều tiêu chí để đánh giá các tựa game, chẳng hạn như theo thể loại game, theo giá game, theo độ phổ biến của game trong cộng đồng...

- Giá bán game: Giá bán là do người đăng tải niêm yết. Tùy theo sự kiện cửa hàng có thể tạo một đợt sale cho game. Trong cửa hàng, có thể có phân loại các game bán chạy, Best seller, Editor’s choice,…

- Đương nhiên các trò chơi vẫn có thể để tải xuống miễn phí (Free). Lúc này trò chơi sẽ không thống kê doanh thu (không thỏa thuận tính theo doanh thu), mà chỉ

thống kê lượt tải/ tương tác để có thể được giới thiệu theo xu hướng hấp dẫn người dùng

- Đánh giá game theo sao, và có thể được "trending" khi đạt lượt sao thống kê kỷ lục theo tháng

- Wish list: Tương đương với Danh sách yêu thích. Người dùng có thể đưa một game vào trong danh sách yêu thích của mình mà không cần phải mua nó.

4, Các chức năng cho ứng dụng:

- Sẽ được xem các tất cả các game với thông tin, được sắp xếp vào các thể loại

- Được xem Xu hướng, bảng xếp hạng các game được cập nhật vào 1 khoảng thời gian nhất định (theo tháng, theo tuần,…)

- Được truy cập và xem thông tin cụ thể của từng game, với các đánh giá của người dùng khác.

- Đăng kí tài khoản: Xác thực thông tin cá nhân bằng các định danh có thật

- Wish list (được đề cập ở trên)

- Tìm kiếm game: Bằng thanh tìm kiếm, bằng các tag thể loại, bằng tab xu hướng hoặc tab đề xuất

- Nạp tiền vào ví tài khoản

- Mua và đánh giá game, với mỗi lần mua chỉ được 1 game

- Trong trang cá nhân nhà phát hành, tra cứu được số liệu về doanh thu của các trò chơi đã đăng tải

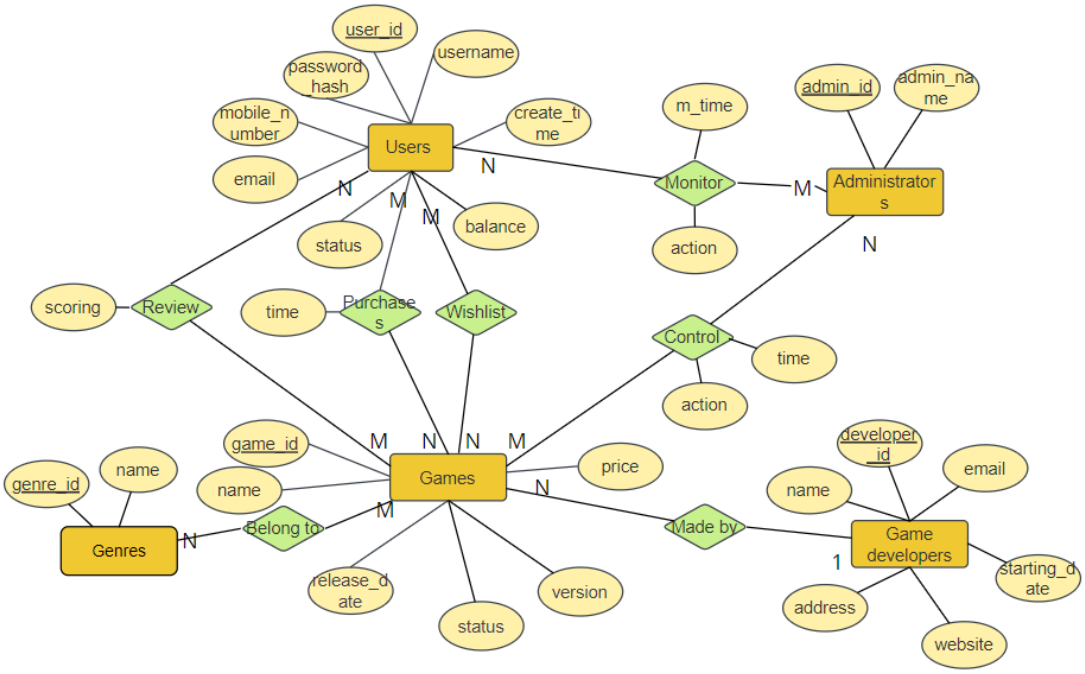
- Có khả năng yêu cầu xóa/ báo cáo người dùng khác (kể cả Quản trị viên)

- Với quản trị viên

    + Duyệt game lên hệ thống

    + Quản lí người dùng thông thường: Cấm, xóa tài khoản, đánh giá,…

II, Sơ đồ thực thể liên kết (ER) và các bảng Quan hệ:



Từ đó ta suy ra các bảng quan hệ:

users(<u>user_id</u>, username, password_hash, email, phone_number, balance, status, create_date)

games(<u>game_id</u>, name, release_date, status, *developer_id,* price, ver)

administrators(<u>admin_id</u>, admin_name)

monitor(*<u>admin_id</u>, <u>user_id</u>*, <u>m_time</u>, action)

control(*<u>admin_id</u>, <u>game_id</u>*, <u>c_time</u>, action)

genres(<u>genre_id</u>, genre_name)

game_dev(<u>developer_id</u>, name, email, address, website, starting_date)

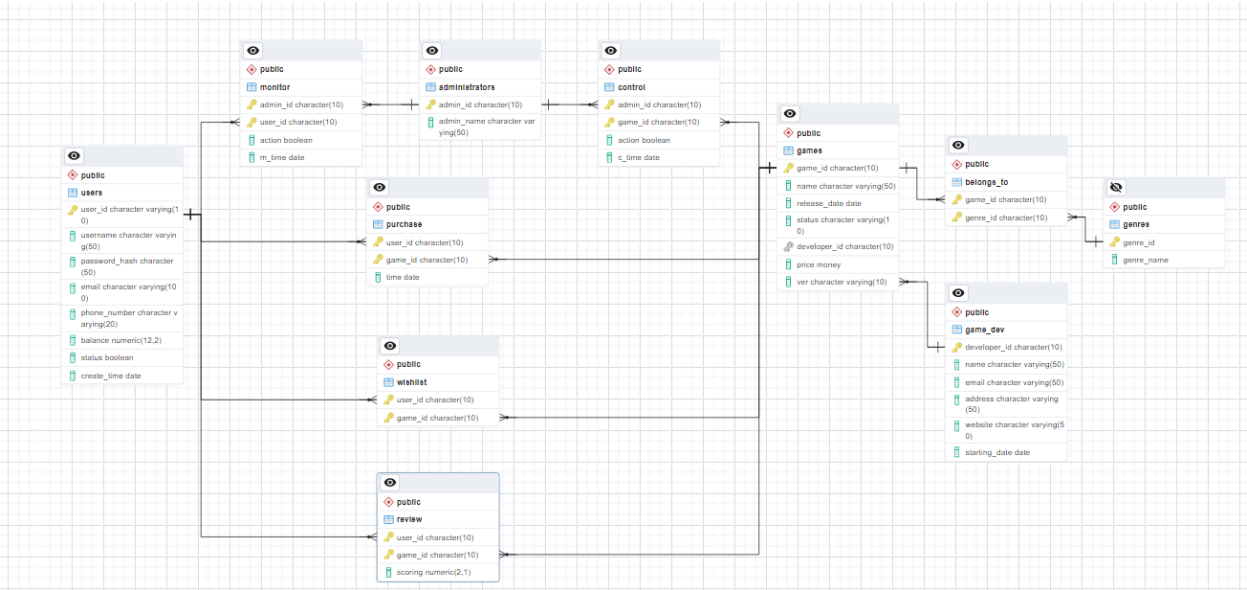
purchase(*user_id, game_id*, time)

wishlist(*<u>user_id, game_id</u>*)

review(*user_id, game_id*, scoring)

belongs_to(*game_id, genre_`id*)

Với các định dạng kiểu thuộc tính và các ràng buộc như trong hình sau và trong file định nghĩa:



III, Các câu lệnh truy vấn

Các câu lệnh truy vấn sau có phụ thuộc vào 2 VIEW: game_rate (số điểm đánh giá của game) và game_revenue (doanh thu của game)

```
CREATE VIEW game_rate AS
SELECT game_id, AVG(scoring) AS rating
FROM review
GROUP BY game_id;
```

```
CREATE VIEW game_revenue AS
SELECT p.game_id, SUM(g.price) AS revenue
FROM purchase p
JOIN games g ON p.game_id = g.game_id
GROUP BY p.game_id;
```

1, Đinh Huy Dương 20215020:

1. In ra những người dùng đã gắn bó hệ thống được 2 năm:

```
-- C1

SELECT *

FROM users

WHERE (SELECT EXTRACT(YEAR FROM AGE(CURRENT_DATE, create_time))) >= 2;
```

Phương án này sử dụng truy vấn con để tính toán khoảng thời gian từ thời điểm hiện tại đến bây giờ đối với TỪNG hàng. Hoàn toàn không hiệu quả và không thể sử dụng Index

```
-- C2

SELECT *

FROM users

WHERE create_time <= CURRENT_DATE - INTERVAL '2 years';
```

Có thể sử dụng Index trên 'create_time' để tìm kiếm theo Index trong 1 thuộc tính 'create_time' với điều kiện nó bé hơn thời gian từ bây giờ đến thời điểm tạo ra là 2 năm (Đây gần như là một hằng số với câu truy vấn này), có thể tối ưu câu truy vấn này

2. In ra những người dùng mua được từ 10 game trở lên

```
--C1

SELECT *

FROM users

WHERE (

    SELECT COUNT(*)

    FROM purchase

    WHERE purchase.user_id = users.user_id

) > 10;
```

Với mỗi 1 User được kiểm tra ở WHERE, lại phải chạy query phụ và duyệt qua xem có đếm đủ số game được mua không. Hiển nhiên là không hiệu quả

```
--C2

WITH purchase_counts AS (

    SELECT user_id, COUNT(*) as game_count
```

```
    FROM purchase
    GROUP BY user_id
    HAVING COUNT(*) > 10
)
SELECT users.*
FROM users
JOIN purchase_counts
ON users.user_id = purchase_counts.user_id;
--C3
SELECT users.*
FROM users
JOIN (
    SELECT user_id, COUNT(*) as game_count
    FROM purchase
    GROUP BY user_id
    HAVING COUNT(*) > 10
) purchase_counts
ON users.user_id = purchase_counts.user_id;
```

Cách 2 và 3 điều tạo ra 1 bảng nhỏ mới từ truy vấn con để nối vào bảng User, không cần phải duyệt và tạo bảng nhỏ theo từng bản ghi ở trong câu lệnh WHERE, nên có thể sử dụng Index cho user_id để tăng hiệu năng cho câu lệnh

3. In ra những người dùng mua nhiều 5 game trở lên thuộc thể loại "RPG".

```
--C1
SELECT u.*
FROM users u
JOIN purchase p ON u.user_id = p.user_id
JOIN belongs_to b ON p.game_id = b.game_id
JOIN genres g ON b.genre_id = g.genre_id
WHERE g.genre_name = 'RPG'
GROUP BY u.user_id
HAVING COUNT(*) > 5;
```

```
--C2
SELECT u.*
FROM users u
JOIN (
    SELECT p.user_id, COUNT(*) as purchase_count
    FROM purchase p
    JOIN belong_to b ON p.game_id = b.game_id
    JOIN genres g ON b.genre_id = g.genre_id
    WHERE g.genre_name = 'RPG'
    GROUP BY p.user_id
    HAVING COUNT(*) > 5
) purchase_counts
ON u.user_id = purchase_counts.user_id;
```

2 phương án trên được đánh giá là hiệu năng khá tương đồng với nhau, do cũng cùng khối lượng nối các bảng với nhau, và nối các bảng với điều kiện trong câu lệnh con trong mệnh đề IN ở phương án cũng không giảm bớt đi số lượng bản ghi

4. Danh sách các người dùng đã mua game nhưng bị cấm (trạng thái = false)

```
-- C1
SELECT DISTINCT users.*
FROM users
JOIN purchase ON users.user_id = purchase.user_id
WHERE users.status = false;
```

-- Sử dụng Distinct sẽ bắt buộc hệ thống phải nhớ lại những bản ghi nào đã có chưa

```
--C2
SELECT *
FROM users
WHERE status = false
AND EXISTS (
    SELECT 1
    FROM purchase
```

```
    WHERE purchase.user_id = users.user_id
);
```

Sử dụng Exist như trên sẽ lọc ra những bản ghi có tồn tại trong Purchase trong mệnh đề WHERE ở trước, hơn thế nữa còn có thể sử dụng Index cho user_id

5. Các game có rating trên 8.0 và doanh thu thấp hơn 10000

```
SELECT g.name, gr.rating, gre.revenue
FROM games g
JOIN game_rate gr ON g.game_id = gr.game_id
JOIN game_revenue gre ON g.game_id = gre.game_id
WHERE gr.rating > 8.0 AND gre.revenue < 10000;
```

6. Game được nhiều người mua nhất vào tháng 3/2023

```
--C1
SELECT g.name, COUNT(*) as purchase_count
FROM games g
JOIN purchase p ON g.game_id = p.game_id
WHERE date_part('year', p.time) = 2023 AND date_part('month', p.time) = 3
GROUP BY g.game_id, g.name
ORDER BY purchase_count DESC;
--C2
SELECT g.name, purchase_counts.purchase_count
FROM games g
JOIN (
    SELECT game_id, COUNT(*) as purchase_count
    FROM purchase
    WHERE date_part('year', time) = 2023 AND date_part('month', time) = 3
    GROUP BY game_id
) purchase_counts
ON g.game_id = purchase_counts.game_id
ORDER BY purchase_counts.purchase_count DESC;
```

2 câu lệnh trên về cơ bản cũng hoạt động tương tự nhau, nối bảng games với purchase, lọc ra điều kiện, và đếm theo nhóm. Tuy nhiên phương án thứ 2 lại nhóm và lọc điều kiện trước rồi mới nối nên có khả năng sẽ lọc ra ít bản ghi hơn. Mặc dù ít bản ghi hơn nhưng khi nối vào vẫn phải duyệt qua các bản ghi ở bảng games nên khó có thể phán đoán được cái nào thực sự hiệu quả hơn. Nhưng kết luận chung, là 2 phương án trên đều có hiệu năng khá tương đương với nhau

7. Game thuộc thể loại "Sport" có doanh thu cao nhất tính đến nay

```
--C1
SELECT g.name, gre.revenue
FROM games g
JOIN belongs_to b ON g.game_id = b.game_id
JOIN genres ge ON b.genre_id = ge.genre_id
JOIN game_revenue gre ON g.game_id = gre.game_id
WHERE ge.genre_name = 'Sport'
ORDER BY gre.revenue DESC
LIMIT 1;
--C2
WITH max_sport_revenue AS (
    SELECT MAX(revenue) as max_revenue
    FROM game_revenue gr
    JOIN games ga ON gr.game_id = ga.game_id
    JOIN belongs_to be ON ga.game_id = be.game_id
    JOIN genres gen ON be.genre_id = gen.genre_id
    WHERE gen.genre_name = 'Sport'
)
SELECT g.name, gre.revenue
FROM games g
JOIN belongto b ON g.game_id = b.game_id
JOIN genres ge ON b.genre_id = ge.genre_id
JOIN game_revenue gre ON g.game_id = gre.game_id
JOIN max_sport_revenue msr ON gre.revenue = msr.max_revenue
```

```
WHERE ge.genre_name = 'Sport';
```

Phương án thứ nhất là sắp xếp các các bảng nối vào nhau để ra được các game có thể loại 'Sport' theo thứ tự của Revenue trong VIEW game_revenue và lấy ra 1 để làm Max. Cách này có hạn chế là nếu như các game có cùng Revenue sẽ không được hiển thị quá 2. Cách thứ 2 lại sử dụng hàm MAX để hiển thị được ra, tuy nhiên lại phải sử dụng truy vấn con trong WITH để lọc ra các game có Revenue lớn nhất rồi nối với các bảng còn lại để hiển thị được tên, như thế sẽ hiển thị được TẤT CẢ các game có cùng giá trị cao nhất của doanh thu. Thực tế là 2 phương án trên cũng có khả năng mà hiệu năng tương tự, phương án 2 tưởng chừng sử dụng câu lệnh con, nhưng lại lọc ra số bản ghi cũng ít như điều kiện phương án đầu.

8. Game được phân vào 3 thể loại trở lên

```
--C1

WITH genre_counts AS (

    SELECT game_id, COUNT(*) as genre_count

    FROM belongs_to

    GROUP BY game_id

)

SELECT g.name

FROM games g

JOIN genre_counts b ON g.game_id = b.game_id

WHERE b.genre_count > 3;
```

Sử dụng được Index cho các thuộc tính của bảng belongs_to để tăng hiệu năng

```
--C2

SELECT g.name

FROM games g

JOIN belongs_to b ON g.game_id = b.game_id

GROUP BY g.game_id, g.name

HAVING COUNT(DISTINCT b.genre_id) > 3;
```

Cần phải sử dụng Distinct để có thể đếm được các thể loại khác nhau, nhưng sử dụng nó lại phải tốn kém chi phí lưu trữ, tuy nhiên vẫn có thể sử dụng Index trên thuộc tính bảng belongs_to để tìm kiếm

9. Game được ưa thích nhất thuộc thể loại "Action" (không tính theo doanh thu mà tính theo lượt mua và wishlist)

```
WITH action_games AS (
```

```
    SELECT g.game_id, g.name
    FROM games g
    JOIN belongs_to b ON g.game_id = b.game_id
    JOIN genres ge ON b.genre_id = ge.genre_id
    WHERE ge.genre_name = 'Action'
),
popularity AS (
    SELECT game_id, COUNT(*) as popularity_count
    FROM (
        SELECT game_id FROM purchase
        UNION ALL
        SELECT game_id FROM wishlist
    ) p
    GROUP BY game_id
),
max_popularity AS (
    SELECT MAX(popularity_count) as max_popularity_count
    FROM popularity p
    JOIN action_games a ON p.game_id = a.game_id
)
SELECT a.name, p.popularity_count
FROM action_games a
JOIN popularity p ON a.game_id = p.game_id
JOIN max_popularity m ON p.popularity_count = m.max_popularity_count;
```

Ta sử dụng 3 bảng phụ trong mệnh đề WITH để lọc theo các tiêu chí là theo thể loại 'Action', bảng hợp của purchase và wishlist, và max số bản ghi của bảng hợp đó. Tuy nhiên phương án này sẽ có 1 phần khá nặng là hợp UNION ALL giữa 2 bảng purchase với wishlist. Ta có thể sử dụng phương án dưới đây là tách 2 bảng ra và tính theo công thức:

```
--C2
WITH action_games AS (
    SELECT g.game_id, g.name
```

```
    FROM games g
    JOIN belongs_to b ON g.game_id = b.game_id
    JOIN genres ge ON b.genre_id = ge.genre_id
    WHERE ge.genre_name = 'Action'
),
purchase_counts AS (
    SELECT game_id, COUNT(*) as purchase_count
    FROM purchase
    GROUP BY game_id
),
wishlist_counts AS (
    SELECT game_id, COUNT(*) as wishlist_count
    FROM wishlist
    GROUP BY game_id
),
popularity AS (
    SELECT a.game_id, COALESCE(p.purchase_count, 0) + COALESCE(w.wishlist_count,
0) as popularity_count
    FROM action_games a
    LEFT JOIN purchase_counts p ON a.game_id = p.game_id
    LEFT JOIN wishlist_counts w ON a.game_id = w.game_id
),
max_popularity AS (
    SELECT MAX(popularity_count) as max_popularity_count
    FROM popularity
)
SELECT a.name, p.popularity_count
FROM action_games a
JOIN popularity p ON a.game_id = p.game_id
JOIN max_popularity m ON p.popularity_count = m.max_popularity_count;
```

Ở phương án này ta tách các tiêu chí đếm các bản ghi từ wishlist và purchase thành các bảng khác nhau và với bảng tính độ nổi tiếng: Đếm số bản ghi của 2 bảng trên với hàm COALESCE trong trường hợp mà game không có Purchase nhưng có Wishlist/ Có Purchase nhưng không có Wishlist khi nối ngoài ở các bảng. Sử dụng bảng này để tính được độ nổi tiếng của game và tìm ra game nào có độ nổi tiếng nhất theo Purchase và Wishlist. Phương án này sẽ bỏ được việc hợp 2 bảng trong phương án 1, có khả năng tối ưu hơn.

10. VIEW hiển thị thể loại được yêu thích nhất (trung bình các đánh giá của game trong thể loại này là cao)

```
CREATE VIEW popular_genres AS
    SELECT g.genre_name, AVG(ga.rating) AS avg_rating
    FROM genres g
    JOIN games ga USING (game_id)
    JOIN game_rate USING (game_id)
    GROUP BY g.genre_name
    ORDER BY avg_rating DESC;
```

2, Nguyễn Thanh Nhật Bảo 20210096:

1. 10 game mới được xuất bản gần đây nhất

```
SELECT *
FROM games
WHERE release_date < current_date
ORDER BY release_date DESC
LIMIT 10;
 Phương án này có thể sử dụng được Index trên cột release_date để tăng hiệu năng
```

2.Thể loại game được ưa thích nhất (phụ thuộc vào việc chứa nhiều bản game được tải nhất)

```
-- c1
WITH table_temp as (
    SELECT genre_id, genre_name, count(user_id) as downloaded
    FROM (genres left join belongs_to using (genre_id)) join purchase using
(game_id)
    GROUP BY genre_id, genre_name
)
SELECT genre_id, genre_name  FROM table_temp WHERE downloaded >= ALL (SELECT
downloaded FROM table_temp);
```

```
       Sử dụng được Index trên các thuộc tính của belongs_to
-- C2
SELECT g.genre_id, g.genre_name
FROM genres g
JOIN belongs_to b ON g.genre_id = b.genre_id
JOIN purchase p ON b.game_id = p.game_id
GROUP BY g.genre_id, g.genre_name
HAVING COUNT(p.user_id) = (
    SELECT MAX(downloaded)
    FROM (
        SELECT COUNT(p2.user_id) as downloaded
        FROM genres g2
        JOIN belongs_to b2 ON g2.genre_id = b2.genre_id
        JOIN purchase p2 ON b2.game_id = p2.game_id
        GROUP BY g2.genre_id, g2.genre_name
    ) t
);
      Phương án này sử dụng điều kiện của nhóm HAVING và trong đó có một câu
lệnh truy vấn con. Khi đó, chương trình sẽ phải chạy qua từng bản ghi trong nhóm,
thực hiện câu truy vấn con này. Khiến cho phương án này không hiệu quả
3. Loại Rating nào mà được nhiều người chơi đánh giá nhất cho "Celestial
Odyssey"?
WITH table1 AS (
    SELECT user_id, scoring,
           CASE
                WHEN scoring < 2
                     THEN 'Bad'
                WHEN scoring >= 2
                    AND scoring < 4 THEN 'Average'
                WHEN scoring > 4 THEN 'Good'
           END type_rating
    FROM review JOIN (select game_id from games where name = 'Celestial
Odyssey' ) as temp USING (game_id)
)
SELECT type_rating, count(user_id) number_rating
FROM table1
GROUP BY type_rating
HAVING count(user_id) >= ALL (select count(user_id) from table1 group by
type_rating);
       Ta có thể sử dụng Index cho cột game_id hoặc scoring, tuy nhiên cột
scoring có thể có các giá trị lặp nên chỉ cần game_id cũng có thể tăng hiệu năng

4.Game được mong chờ nhất (chưa hoàn thiện/ nhiều người wishlist)
select  game_id, count(user_id)
```

```
from (select * from games where release_date > current_date) as temp join
wishlist using (game_id)
group by game_id
order by count(user_id) DESC
limit 1;
```

5. Game nào được nhiều người review nhất

```
-- C1
select *
from games
where game_id = (
    select game_id
    from review
    group by game_id
    having count(user_id) >= all (select count(user_id) from review group by
game_id)
);
```

Sử dụng HAVING sẽ phải lọc điều kiện nhóm, mà nhóm cũng là một công đoạn khá tốn kém nên ta có phương án sử dụng mệnh đề con ở WITH như sau:

```
--C2
WITH max_ratings AS (
    SELECT MAX(rating_count) as max_rating_count
    FROM (
        SELECT game_id, COUNT(user_id) as rating_count
        FROM review
        GROUP BY game_id
    ) t
)
SELECT *
FROM games g
JOIN (
    SELECT game_id, COUNT(user_id) as rating_count
    FROM review
    GROUP BY game_id
) r ON g.game_id = r.game_id
JOIN max_ratings m ON r.rating_count = m.max_rating_count;
```

Lúc này mệnh đề con ở WIth sẽ lọc và tính ra được MAX của số lượng rating sử dụng để nối với bảng game, sẽ không phải nhóm và kiểm tra liên tục như phương án đầu

6.Nhà phát triển có lượt mua cao nhất vào năm 2020

```
-- C1
with table_temp1 as (
    select developer_id, count(user_id) as number_downloaded
```

```
     from (select game_id, user_id from purchase where time between '2020-01-01'
and '2020-12-01' ) as temp1
          join
          (select game_id, developer_id from games ) as temp2
     using (game_id)
     group by developer_id
)
select * from table_temp1 where number_downloaded >= all (select
number_downloaded from table_temp1);
      Phương án này sẽ khá tốt nhưng với điều kiện ở cuối >= ALL, ta sẽ không
thể sử dụng được Index, thay vào đó, ta có thể dùng MAX từ một bảng phụ khác:

-- C2
WITH purchase_counts AS (
    SELECT g.developer_id, COUNT(p.user_id) as purchase_count
    FROM purchase p
    JOIN games g ON p.game_id = g.game_id
    WHERE p.time BETWEEN '2020-01-01' AND '2020-12-01'
    GROUP BY g.developer_id
),
max_purchase AS (
    SELECT MAX(purchase_count) as max_purchase_count
    FROM purchase_counts
)
SELECT pc.developer_id, pc.purchase_count
FROM purchase_counts pc
JOIN max_purchase mp ON pc.purchase_count = mp.max_purchase_count;

7.Nhà phát triển có doanh thu cao nhất (trong thời gian nào đó)
with table_temp1 as (
    select developer_id, sum(price) as total_revenue
     from (select game_id, user_id from purchase where time between '2020-01-01'
and '2020-12-01' ) as temp1
          join
          (select game_id, developer_id, price from games ) as temp2
     using (game_id)
     group by developer_id
)
select * from table_temp1 where total_revenue >= all (select total_revenue from
table_temp1);

      Tương tự như câu trên, để tránh sử dụng ALL, ta có thể thêm một truy vấn
con trong WITH:
WITH revenue AS (
    SELECT g.developer_id, SUM(g.price) as total_revenue
```

```
    FROM purchase p
    JOIN games g ON p.game_id = g.game_id
    WHERE p.time BETWEEN '2020-01-01' AND '2020-12-01'
    GROUP BY g.developer_id
),
max_revenue AS (
    SELECT MAX(total_revenue) as max_total_revenue
    FROM revenue
)
SELECT r.developer_id, r.total_revenue
FROM revenue r
JOIN max_revenue m ON r.total_revenue = m.max_total_revenue;
```

8.In ra các nhà phát triển đã phát triển nhiều nhất 1 game thuộc thể loại "Visual Novel"

```
--C1
SELECT d.name
FROM game_dev d
LEFT JOIN (
    SELECT g.developer_id, COUNT(*) as game_count
    FROM games g
    JOIN belongs_to b ON g.game_id = b.game_id
    JOIN genres ge ON b.genre_id = ge.genre_id
    WHERE ge.genre_name = 'Visual Novel'
    GROUP BY g.developer_id
) v ON d.developer_id = v.developer_id
WHERE v.game_count IS NULL OR v.game_count <= 1;

--C2
WITH vn_games AS (
    SELECT g.developer_id, COUNT(*) as game_count
    FROM games g
    JOIN belongs_to b ON g.game_id = b.game_id
    JOIN genres ge ON b.genre_id = ge.genre_id
    WHERE ge.genre_name = 'Visual Novel'
    GROUP BY g.developer_id
)
SELECT d.name
FROM game_dev d
LEFT JOIN vn_games v ON d.developer_id = v.developer_id
WHERE v.game_count IS NULL OR v.game_count <= 1;
```

Cả 2 cách đều phải sử dụng JOIN ngoài để có thể xem được xem mình đã có được bao nhiêu game chứa 0 hoặc 1 thể loại như đề bài. Điều kiện WHERE không thay đổi nhiều nên hiệu năng tương đương

9.Thể loại game chủ đạo của nhà phát triển "Nintendo"

```sql
SELECT genre_id, genre_name, count(game_id)
FROM (SELECT * FROM game_dev WHERE name = 'Nintendo' ) AS temp
       JOIN games USING (developer_id)
       JOIN belongs_to USING (game_id)
       JOIN genres USING (genre_id)
GROUP BY genre_id, genre_name
ORDER BY count(game_id) DESC
LIMIT 1;
```

10.Thể loại Game mang lại doanh thu lớn nhất cho Nintendo

```sql
-- C1
WITH table1 as(
SELECT genre_id, genre_name, count(game_id) AS number_of_games
FROM ((SELECT * FROM game_dev WHERE name = 'Nintendo' ) AS temp
       JOIN games USING (developer_id)
       JOIN belongs_to USING (game_id)
       JOIN genres USING (genre_id)) as table1
GROUP BY genre_id, genre_name)
SELECT genre_id, genre_name, number_of_games
FROM table1
WHERE number_of_games >= ALL  (SELECT number_of_games FROM table1);
--C2
SELECT genres.genre_id, genres.genre_name, COUNT(games.game_id) AS
number_of_games
FROM game_dev
JOIN games ON game_dev.developer_id = games.developer_id
JOIN belongs_to ON games.game_id = belongs_to.game_id
JOIN genres ON belongs_to.genre_id = genres.genre_id
WHERE game_dev.name = 'Nintendo'
GROUP BY genres.genre_id, genres.genre_name
ORDER BY number_of_games DESC
LIMIT 1;
```

Không phải WITH nào cũng là tối ưu, trường hợp này truy vấn con có khả năng chậm hơn nối JOIN thông thường, và chưa kể ALL nghĩa là phải so sánh với tất cả bản ghi, kém hiệu quả hơn ORDER BY

11.Game chưa có đánh giá dù đã có người tải xuống

```sql
SELECT purchase.game_id,
       count(purchase.user_id) as downloaded,
       count(distinct review.user_id) as number_review
FROM purchase LEFT JOIN review on (purchase.game_id = review.game_id AND
purchase.user_id = review.user_id)
GROUP BY purchase.game_id
HAVING count(distinct review.user_id) = 0;
```

```
12.Hàm tìm kiếm
CREATE OR REPLACE FUNCTION search_specify (something varchar)
returns table(
    game_id char(10),
    game_name varchar,
    dev_id char(10),
    dev_name varchar,
    rating numeric(2,1)
)
AS
$$
DECLARE
    check_number_var boolean;
    check_asign_var boolean;
    string_search varchar;
BEGIN
    select something ilike '#%' into check_number_var;
    select something ilike '@%' into check_asign_var;

    if check_asign_var then
       SELECT SPLIT_PART( something , '@', 2) into string_search;
       SELECT  quote_literal(string_search || '%') into string_search;
       return QUERY EXECUTE 'SELECT  game_id, game_name, developer_id, dev_name,
rating FROM game_small_details WHERE dev_name ilike  ' || string_search;
    end if;

    if check_number_var then
       SELECT SPLIT_PART( something , '#', 2) into string_search;
       SELECT  quote_literal(string_search || '%') into string_search;
       return QUERY EXECUTE 'SELECT  game_id, game_name, developer_id, dev_name,
rating FROM game_small_details JOIN belongs_to USING (game_id) JOIN genres USING
(genre_id) WHERE genre_name ilike  ' || string_search;
    end if;

    SELECT quote_literal(something || '%') into string_search;
    RETURN QUERY EXECUTE 'SELECT  game_id, game_name, developer_id, dev_name,
rating FROM game_small_details WHERE game_name ilike ' || string_search;

END;
$$
LANGUAGE plpgsql;
```

3, Nguyễn Văn Đăng 20215033:

```
1. In ra phiên bản hiện tại của "Hollow Knight: Silksong"
SELECT ver
FROM games
WHERE lower(name) = 'hollow knight: silksong';
2. In ra các giao dịch thanh toán vào tháng 1 năm 2021 với mức thanh toán dưới
10.00
--C1
SELECT *
FROM purchase join games using (game_id)
WHERE extract(month from "time") = 01 and extract (year from "time") = 2021 AND
price < 10.00;

--C2
SELECT *
FROM purchase join games using (game_id)
WHERE "time" > '2020-12-31' AND "time" < '2021-02-01' AND price < 10.00;

      Cách 1 sử dụng extract để lấy giá trị tháng và năm từ cột time, bằng cách
sử dụng AND để nhận được giá trị ngày tháng thuộc tháng 1 năm 2021. Cách làm này
không thể sử dụng index để cải thiện hiệu năng truy vấn do có quá nhiều giá trị
có tháng là 01 và năm là 2021 trong dữ liệu.
      Cách 2 thay vì sử dụng extract, ta dùng toán tử so sánh để lấy các giá trị
thời gian lớn hơn 31/12/2020 và bé hơn 01/02/2021, từ đó có được thời gian mong
muốn. Index có thể được sử dụng để cải thiện hiệu năng truy vấn trong trường hợp
này.

3.Top 3 Nhà phát triển nào bị đánh giá tiêu cực nhất. (Rating trung bình của các
tựa game do họ phát triển là thấp nhất)
SELECT game_dev.name, CAST(AVG(rating) as numeric(3,2)) as scoring
FROM game_dev join games using (developer_id) join game_rate using (game_id)
GROUP BY developer_id
ORDER BY scoring
LIMIT 3;

4.Game có nhiều đánh giá dưới 1.5 nhất
SELECT name, count(user_id) Number_of_low
FROM games join review using (game_id)
WHERE scoring < 1.5
GROUP BY game_id
ORDER BY Number_of_low DESC
LIMIT 1;

5. Ba người dùng khó tính nhất với các tựa game. (Rating đưa ra trung bình là
thấp nhất)
SELECT username, CAST(AVG(scoring) as numeric(3,2)) grading
```

```
FROM users join review using(user_id)
GROUP BY user_id, username
ORDER BY grading ASC
LIMIT 3;

6.Trigger không cho người dùng review game nếu họ chưa sở hữu game hoặc nếu họ
đang bị hạn chếvà đưa ra thông báo

CREATE OR REPLACE FUNCTION review_invalid() RETURNS TRIGGER AS $$
DECLARE
    f_username varchar(50);
    f_gamename varchar(50);
BEGIN
    select into f_username username
    from users
    where user_id = NEW.user_id;
    select into f_gamename name
    from games
    where game_id = NEW.game_id;
    IF (NEW.user_id) in (select user_id from users where status = 'f') THEN
        RAISE NOTICE '% is being banned, thus unable to make reviews',
f_username;
        RETURN NULL;
    END IF;
    IF (NEW.user_id, NEW.game_id) not in (select user_id, game_id from purchase)
THEN
        RAISE NOTICE '% has not owned %, thus unable to leave review.',
f_username, f_gamename;
        RETURN NULL;
    ELSE IF (NEW.user_id, NEW.game_id) in (select user_id, game_id from review)
THEN
        UPDATE review SET scoring = NEW.scoring where (user_id = NEW.user_id and
game_id = NEW.game_id);
        RAISE NOTICE '% has changed their opinion about %', f_username,
f_gamename;
        RETURN NULL;
        ELSE RETURN NEW;
        END IF;
    END IF;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER not_available
BEFORE INSERT ON review
```

```
FOR EACH ROW
EXECUTE PROCEDURE review_invalid();
      Trigger được gọi trước khi insert vào bảng review  để đảm bảo các review
luôn từ những người dùng đã chơi qua, hoặc ít nhất là đã sở hữu tựa game được
review.

7. Thống kê doanh thu mà các thể loại mang lại cho nhà phát triển "Nintendo"
SELECT game_dev.name, genre_name, SUM(revenue)
FROM game_revenue join games using (game_id) join game_dev using (developer_id)
join belongs_to using (game_id) join genres using (genre_id)
WHERE lower(game_dev.name) = 'nintendo'
GROUP BY game_dev.name, genre_name;

8. Thống kê số lượt game được mua và wishlist của các game được xuất bản vào năm
2020
WITH number_purchased AS
(SELECT name, game_id, count(user_id) as no_purchase
FROM games left join purchase using (game_id)
WHERE release_date >= '2020-01-01' AND release_date <= '2020-12-31'
GROUP BY name, game_id)
SELECT name, game_id, no_purchase + count(user_id) as popularity
FROM number_purchased left join wishlist using (game_id)
GROUP BY game_id, name, no_purchase;

9. Thống kê những tựa game có rating trên 4.0 và có giá dưới 20.00 được tạo ra
bởi "PopCaps"
SELECT games.*
FROM games join game_rate using (game_id)
WHERE rating > 4.0 and price < 20.00 and game_id in (
    SELECT game_id
    FROM games join game_dev using (developer_id)
    where lower(game_dev.name) = 'popcap'
)
;

SELECT games.*
FROM games join game_rate using (game_id) join game_dev using (developer_id)
WHERE lower(game_dev.name) = 'popcap' and rating > 4.0 and price < 20.00;


10. Những người dùng tiêu nhiều tiền để mua game nhất.
SELECT username, sum(price) as money_spent
FROM users join purchase using (user_id) join games using (game_id)
GROUP BY username
ORDER BY money_spent DESC
```

```
LIMIT 5;

11. Đưa ra những người dùng đã mua hai game thuộc dòng "Pro Evolution Soccer" trở lên
--C1
WITH a AS
(SELECT user_id
FROM games join purchase using (game_id)
WHERE name LIKE 'Pro Evolution Soccer %'
GROUP BY user_id
HAVING count(game_id) >= 2)
SELECT username
FROM users join a using (user_id)
;

--C2
SELECT username
FROM purchase join users using (user_id)
        join (
        SELECT game_id
        FROM games
        WHERE name LIKE 'Pro Evolution Soccer %'
) AS sub using (game_id)
GROUP BY username
HAVING count(game_id) >= 2;

      Hai cách trên có hiệu suất truy vấn tương tự nhau, tuy nhiên ở cách 1, sub query đã loại bỏ một số bản ghi chứa giá trị user_id không mua tối thiểu 2 game, từ đó giảm thời gian join. Tuy nhiên thời gian tìm ra kết quả của hai cách là tương tự. Việc dùng HAVING ngăn cản sử dụng index trong trường hợp này.
```

IV, Các tính năng được cài đặt trong CSDL

- Chức năng cơ bản nhất: Mua và đánh giá game: Thực hiện INSERT vào trong bảng “purchase” và “review” với “user_id” và “game_id” tương ứng.

+Với việc mua game, sẽ có 1 Trigger thực hiện quá trình kiểm tra xem tài khoản tiền (balance) của người dùng còn đủ tiền không, nếu đủ sẽ trừ số tiền trong

tài khoản. Nếu không đủ tiền để mua game hoặc game không ở trạng thái đang bán (selling) hay người dùng trong trạng thái cấm (f) sẽ không thực hiện được INSERT

```
CREATE OR REPLACE FUNCTION update_balance()
RETURNS TRIGGER AS $$
DECLARE
    game_price NUMERIC(12,2);
    user_balance NUMERIC(12,2);
    f_username varchar(50);
    f_gamename varchar(50);
BEGIN
    SELECT INTO f_username username FROM users WHERE user_id = NEW.user_id;
    SELECT INTO f_gamename name FROM games WHERE game_id = NEW.game_id;
    SELECT INTO game_price price FROM games WHERE game_id = NEW.game_id;
    SELECT INTO user_balance balance FROM users WHERE user_id = NEW.user_id;
    IF NEW.user_id in (SELECT user_id FROM users WHERE status = 'f') THEN
    RAISE NOTICE '% is being restricted, thus cannot purchase %', f_username,
f_gamename;
    RETURN NULL;
    END IF;
    IF NEW.game_id in (SELECT game_id FROM games WHERE status <> 'selling') THEN
    RAISE NOTICE '% is unavailable for purchase now.', f_gamename;
    RETURN NULL;
    END IF;
    IF user_balance >= game_price THEN
        UPDATE users SET balance = balance - game_price WHERE user_id =
NEW.user_id;
    RETURN NEW;
    ELSE
        RAISE NOTICE '% cannot buy % due to insufficent funds.', f_username,
f_gamename;
    RETURN NULL;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER update_balance_trigger
BEFORE INSERT ON purchase
FOR EACH ROW
EXECUTE FUNCTION update_balance();
```

+Với việc đánh giá game, chỉ có người dùng đã mua game hoặc không trong trạng thái cấm (f) mới có thể đánh giá được game. Điều này được kiểm soát qua Trigger:

```
CREATE OR REPLACE FUNCTION review_invalid() RETURNS TRIGGER AS $$
```

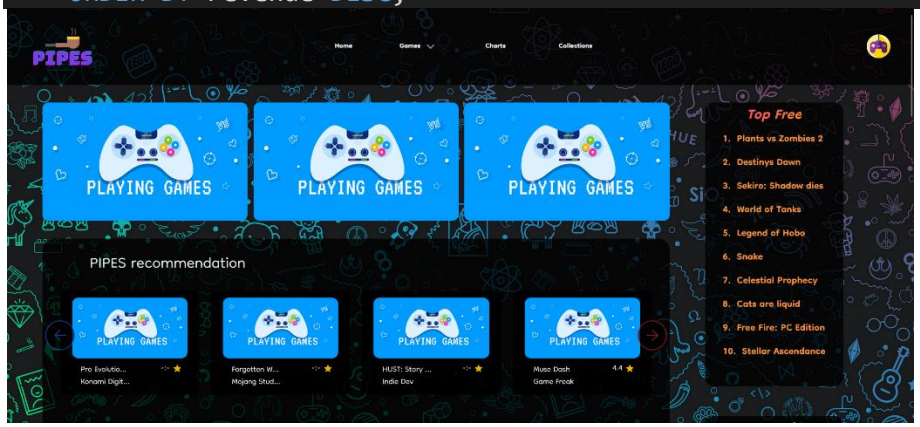
```
DECLARE
        f_username varchar(50);
        f_gamename varchar(50);
BEGIN
        select into f_username username
        from users
        where user_id = NEW.user_id;
        select into f_gamename name
        from games
        where game_id = NEW.game_id;
        IF (NEW.user_id) in (select user_id from users where status = 'f') THEN
                RAISE NOTICE '% is being banned, thus unable to make reviews',
f_username;
                RETURN NULL;
        END IF;
        IF (NEW.user_id, NEW.game_id) not in (select user_id, game_id from
purchase) THEN
                RAISE NOTICE '% has not owned %, thus unable to leave review.',
f_username, f_gamename;
                RETURN NULL;
        ELSE IF (NEW.user_id, NEW.game_id) in (select user_id, game_id from
review) THEN
                UPDATE review SET scoring = NEW.scoring where (user_id =
NEW.user_id and game_id = NEW.game_id);
                RAISE NOTICE '% has changed their opinion about %', f_username,
f_gamename;
                RETURN NULL;
                ELSE RETURN NEW;
                END IF;
        END IF;
END;
$$
LANGUAGE plpgsql;
```

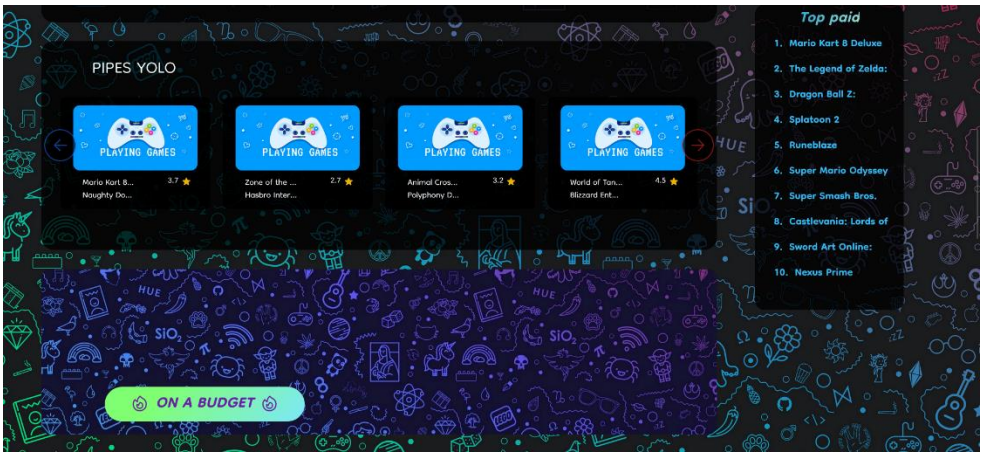
- Hiển thị bảng xếp hạng Top game trả tiền và Top game miễn phí qua 2 VIEW:

```
-- View Top Free Games
-- Ranked by Rating
CREATE VIEW top_free_games AS
    SELECT name, rating
    FROM games join game_rate using (game_id)
    WHERE price = 0
    ORDER BY rating DESC;

-- VIEW Top Paid game
-- Ranked by the Revenue
```

```sql
CREATE VIEW top_paid_games AS
    SELECT name, revenue
    FROM game_revenue JOIN games USING (game_id)
    WHERE price > 0
    ORDER BY revenue DESC;
```





- Hàm thực hiện nạp tiền vào tài khoản:

```sql
CREATE OR REPLACE FUNCTION increase_balance(userid character, money numeric)
RETURNS VOID
AS $$
BEGIN
    UPDATE users
    SET balance = balance + money
    WHERE users.user_id = userid;
END;
$$
LANGUAGE plpgsql;
```

V, Hạn chế và cách khắc phục:

- Ở giai đoạn thiết kế ER, nhóm có phân vân một số điều:

+ Liệu có cần phải có một thực thể "Thư viện" cho từng người dùng

Khắc phục: Không cần thiết, vì chúng ta có thể trích xuất thông tin các game đã mua để tạo ra một "Thư viện" chứa các game đã có lịch sử giao dịch

+ 1 Game sẽ có 1 thuộc tính đa trị là "Genre" (Thể loại)

Khắc phục: Thuộc tính đa trị về mặt logic là đúng, tuy nhiên khi càng đi sâu, ta nhận ra các Thể loại cũng đáng cho vào làm 1 thực thể mới, liên kết n-n với Games. Khi đó ta có thể trích xuất nhiều game cùng thể loại hay sửa các thông tin chỉ trong bảng "belongs_to" là liên kết giữa "games" và "genres".

- Trong bảng "purchase" (Giao dịch), liệu có cần thuộc tính "cost"

Khắc phục: Thuộc tính "cost" về bản chất cũng là "price" của game. Trước đây nhóm đã có viết một Trigger để gán Price của một game vào trước khi một lệnh INSERT vào trong bảng Purchase. Tuy nhiên vì vấn đề game còn có các trạng thái khác "selling", nghĩa là không bán, nên nhóm đã quyết định bỏ hẳn thuộc tính này đi. Thông tin về game đã được mua sẽ nối với bảng game để truy vấn ra được số tiền mà người dùng phải mất

- Trong bảng "games" có 2 thuộc tính "revenue" (Doanh thu), và "rating" (Đánh giá) là thông tin chung của games. Liệu có được không?

Khắc phục: Không. Nhà phát hành games có thể tự ý chỉnh sửa doanh thu hay đánh giá trong game mà không ai kiểm soát. Doanh thu/ Đánh giá của game phụ thuộc vào việc người dùng đã Mua (purchase) hay đã Đánh giá (review); nên để trích xuất thông tin này, ta đơn giản tạo 2 VIEW: game_revenue, game_rate

VI, Đánh giá chung cuối cùng:

- Với một cơ sở dữ liệu ứng dụng cho một cửa hàng game, cơ sở dữ liệu này đã thỏa mãn được những tiêu chí cơ bản: Mua, đánh giá, nạp tiền,…

- Đã phân loại được những nhóm người dùng cụ thể để làm cho việc quản lý và sắp xếp dữ liệu có tổ chức: Người dùng mua, đánh giá; nhà phát triển đăng tải game; người quản lý kiểm soát hoạt động,..

- Đã tạo được trang web thử nghiệm để hiển thị những thông tin từ cơ sở dữ liệu.

- Áp dụng được nhiều câu truy vấn để có thể làm các chức năng cho ứng dụng.

NHỮNG TIÊU CHÍ SAU CHƯA THỎA MÃN ĐƯỢC MỘT ỨNG DỤNG CỬA HÀNG GAME HIỆN ĐẠI

- Trên trang web chạy thử nghiệm, chưa tạo được giao diện của người quản lý và nhà phát hành.

- Chưa ứng dụng được tính năng “giảm giá” (Sales) trong CSDL

- Còn thiếu những chức năng cộng đồng: Chưa thể lưu lại bình luận bằng text

- Bộ dữ liệu chưa đa dạng khiến cho việc kiểm tra truy vấn còn thiếu sót

VII, Phân công công việc:

Các thành viên chịu trách nhiệm cho từng phần câu lệnh truy vấn đã nêu ở trên

1, Đinh Huy Dương 20215020 (Trưởng nhóm):

- Đưa ra ý tưởng về đề tài của dự án

- Thiết kế các Trigger, Function

- Tổng hợp và viết báo cáo

- Kiểm tra lại các câu lệnh truy vấn

2, Nguyễn Thanh Nhật Bảo 20210096:

- Góp ý về việc thiết kế nói chung
- Đưa ra các VIEW cần thiết
- Thiết kế trang web Demo
- Chỉnh sửa các thuộc tính trong các thực thể

3, Nguyễn Văn Đăng 20215033:

- Thiết kế mô hình ER, chuyển đổi qua bảng Quan hệ
- Viết các câu lệnh tạo bảng, và nhập dữ liệu vào trong cơ sở dữ liệu
- Kiểm tra lại CSDL sau mỗi lần chỉnh sửa trong file tạo CSDL