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:**

A diagram of a network

Description automatically generated

Từ đó ta suy ra các bảng quan hệ:

users(user\_id, username, password\_hash, email, phone\_number, balance, status, create\_date)

games(game\_id, name, release\_date, status, *developer\_id,* price, ver)

administrators(admin\_id, admin\_name)

monitor(*admin\_id, user\_id*, m\_time, action)

control(*admin\_id, game\_id*, c\_time, action)

genres(genre\_id, genre\_name)

game\_dev(developer\_id, name, email, address, website, starting\_date)

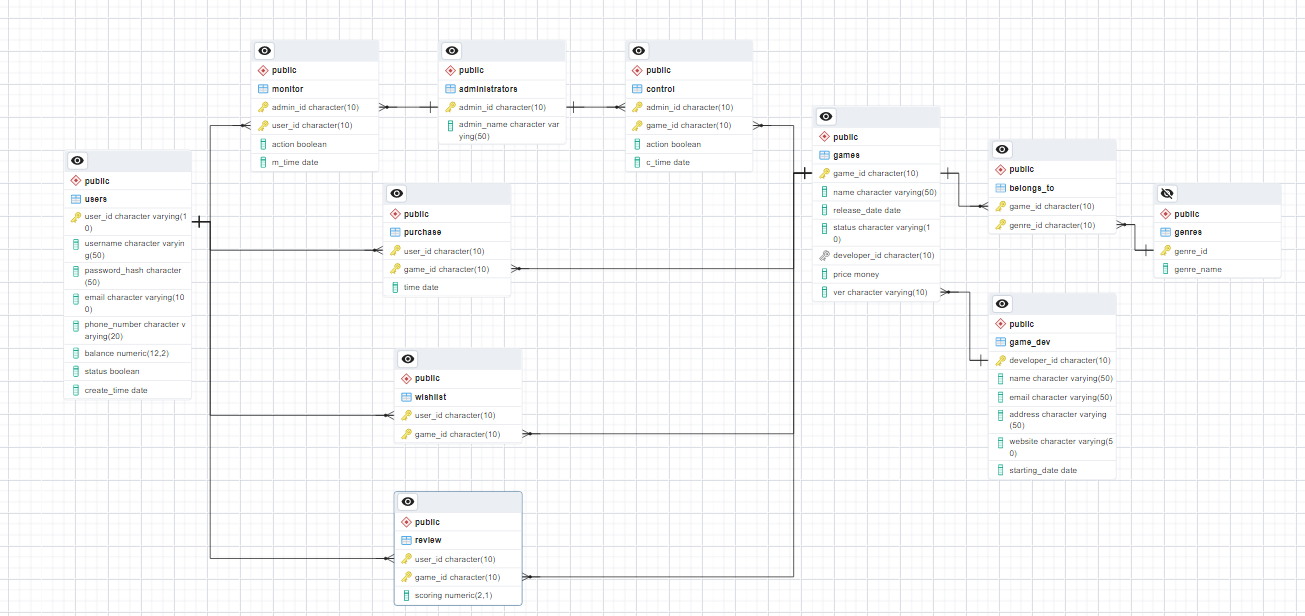
purchase(*user\_id, game\_id*, time)

wishlist(*user\_id, game\_id*)

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 đè 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"**

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**

-- 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**

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

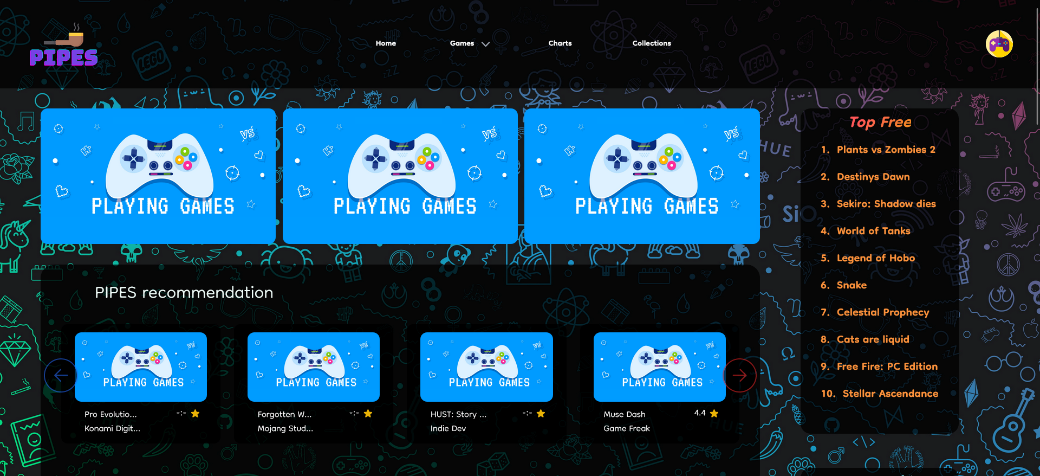
CREATE VIEW top\_paid\_games AS

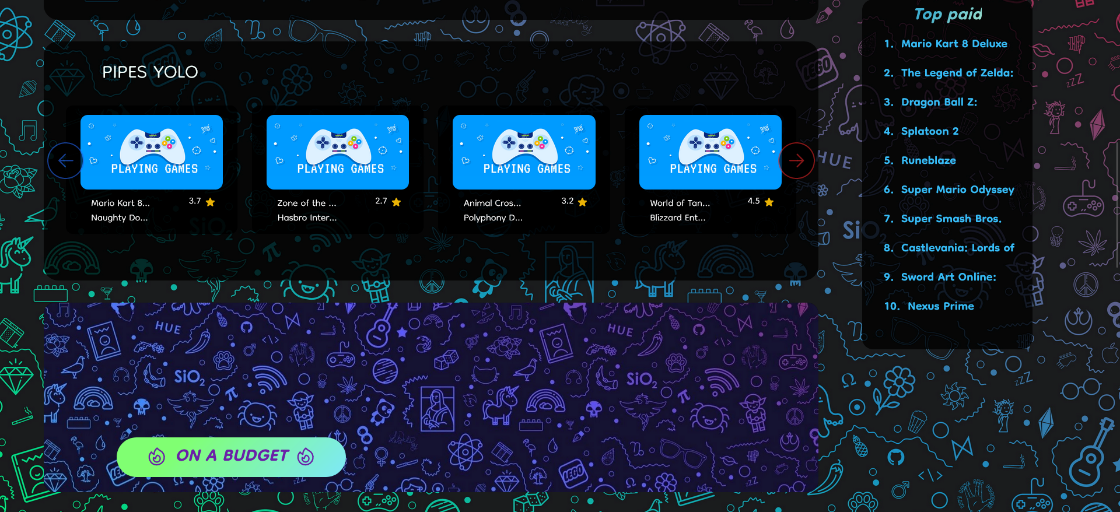
    SELECT name, revenue

    FROM game\_revenue JOIN games USING (game\_id)

    WHERE price > 0

    ORDER BY revenue DESC;





- Hàm thực hiện nạp tiền vào tài khoản:

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