Hude Hude hh3024@columbia.edu

Introduction

This series of SQL queries aims to construct a user-friend network and analyze game ownership patterns among Steam users. The process begins with sampling 10,000 users from the Steam dataset and proceeds by creating tables to capture direct and indirect friendships. The network is expanded to second- and third-degree connections, and game ownership information is integrated for each user and their friends.

0. Random Sampling

This part selects a random sample of 10,000 users from the `steam.Player_Summaries` table. The sample includes users who either own games listed in `steam.Games_2` or are friends with other users (present in `steam.Friends`), and the results are randomized and limited to 10,000 entries.

```
SELECT steamid
FROM (
   SELECT steamid
   FROM steam.Player_Summaries
   WHERE steamid IN (SELECT steamid FROM steam.Games_2)
   OR steamid IN (SELECT steamid_a FROM steam.Friends)
) AS filtered_users
ORDER BY RAND()
LIMIT 10000;
```

1. Create User_Friend Table

This query creates the `User_Friend` table by linking users with their friends. The `Friends` table is joined twice with `steamid_a` and `steamid_b` to ensure that friends are captured in both directions, combining the results using a `UNION`.

```
create table Steam_Sample.User_Friend AS
SELECT s.steamid AS user_id, f.steamid_b AS friend_id
FROM Steam_Sample s
JOIN Friends f
   ON s.steamid = f.steamid_a

UNION

SELECT s.steamid AS user_id, f.steamid_a AS friend_id
FROM Steam_Sample s
JOIN Friends f
   ON s.steamid = f.steamid_b;
```

2. Create User_Friend_IndirectFriend

This query constructs the `User_F_IF` table to capture indirect friendships (friends of friends). It joins the `User_Friend`table with the `Friends` table twice to retrieve indirect friends, ensuring no direct friendship or self-references.

```
Create table Steam_Sample.User_F_IF AS
SELECT sf.user_id,
       sf.friend_id,
       f2.steamid_b AS indirect_friend_id
FROM Steam_Sample.User_Friend sf
JOIN steam_data.Friends f2
    ON sf.friend_id = f2.steamid_a
WHERE f2.steamid_b != sf.user_id
  AND f2.steamid_b NOT IN (
      SELECT friend_id
      FROM Steam_Sample.User_Friend sf2
      WHERE sf2.user_id = sf.user_id
  )

UNION

SELECT sf.user_id,
       sf.friend_id,
       f2.steamid_a AS indirect_friend_id
FROM Steam_Sample.User_Friend sf
JOIN steam_data.Friends f2
    ON sf.friend_id = f2.steamid_b
WHERE f2.steamid_a != sf.user_id
  AND f2.steamid_a NOT IN (
      SELECT friend_id
      FROM Steam_Sample.User_Friend sf2
      WHERE sf2.user_id = sf.user_id
  );
```

3. Get Ownership of Each Game of Each User

This query creates the `Sample_User_Own` table, which stores user-game ownership information. It checks if each user (`steamid`) owns a specific game (`appid`) and marks ownership as `1` if a match is found in `steam_data.Games_2`, otherwise `0`.

```
Create Table Steam_Sample.Sample_User_Own
SELECT u.user_id, u.appid, u.Title, u.Price, u.Release_Date, u.Rating,
       CASE
           WHEN g.steamid IS NOT NULL THEN 1
           ELSE 0
       END AS owns_game
FROM Steam_Sample.Sample_User_App u
LEFT JOIN steam_data.Games_2 g
    ON u.user_id = g.steamid
    AND u.appid = g.appid;
```

4. Drop Users with No Game Adoption

4.1:
This query calculates how many games each user owns and stores the total in the `User_Game_Count` table.

```
CREATE TABLE Steam_Sample.User_Game_Count AS
```

```
SELECT user_id, SUM(owns_game) AS total_games_owned
FROM Steam_Sample.Sample_User_Own
GROUP BY user_id;
```

4.2:
This query filters out users who do not own any games by joining the game ownership count with the `Sample_User_Own`table, creating the `User_Game_Base` table for users who own at least one game.

```
CREATE TABLE Steam_Sample.User_Game_Base AS
SELECT u.*
FROM Steam_Sample.Sample_User_Own u
JOIN Steam_Sample.User_Game_Count c
   ON u.user_id = c.user_id
WHERE c.total_games_owned > 0;
```

5. Recreate User_Friend Network Table with Desired User Base

This query creates a new table, `User_F`, by linking the remaining users (from `User_Game_Base`) to their friends in the `User_Friend` table, ensuring the new user base contains only users with valid game ownership.

```
CREATE TABLE Steam_Sample.User_F AS
SELECT DISTINCT b.user_id, f.friend_id
FROM Steam_Sample.User_Game_Base b
LEFT JOIN Steam_Sample.User_Friend f
   ON b.user_id = f.user_id;
```

6. Count Number of Friends for Each User

This query creates the `User_F_Count` table by counting how many friends each user has in the `User_F` table and ordering the results by the number of friends in descending order.

```
CREATE TABLE Steam_Sample.User_F_Count AS
SELECT user_id, COUNT(friend_id) AS num_friend
FROM Steam_Sample.User_F
GROUP BY user_id order by num_friend desc;
```

7. Get Second-Degree Friend Network

This query creates the `User_F_IF` table, which contains second-degree friends (friends of friends). It joins the `User_F`table with `Friends`, ensuring that no direct connections or repeated entries are included.

```
CREATE TABLE Steam_Sample.User_F_IF AS
SELECT sf.user_id,
       sf.friend_id,
       f2.steamid_b AS IF_id
FROM Steam_Sample.User_F sf
JOIN steam_data.Friends f2
   ON sf.friend_id = f2.steamid_a
WHERE f2.steamid_b != sf.user_id
  AND f2.steamid_b NOT IN (
      SELECT friend_id
      FROM Steam_Sample.User_F sf2
      WHERE sf2.user_id = sf.user_id
  )

UNION

SELECT sf.user_id,
       sf.friend_id,
       f2.steamid_a AS IF_id
FROM Steam_Sample.User_F sf
JOIN steam_data.Friends f2
    ON sf.friend_id = f2.steamid_b
WHERE f2.steamid_a != sf.user_id
  AND f2.steamid_a NOT IN (
      SELECT friend_id
      FROM Steam_Sample.User_F sf2
      WHERE sf2.user_id = sf.user_id
  );
```

8. Drop Games that No One Owns

8.1:
This query counts the total number of users owning each game and stores the result in the `App_Userbase` table.

```
Create Table Steam_Sample.App_Userbase AS
SELECT appid, SUM(owns_game) AS total_player
FROM Steam_Sample.SUser_Game_Base
GROUP BY appid;
```

8.2:
This query filters the games, keeping only those that have at least one owner, and creates the `SUser_Game` table by joining the game ownership count with the base table.

```
CREATE TABLE Steam_Sample.SUser_Game AS
SELECT u.*, c.total_player
FROM Steam_Sample.SUser_Game_Base u
JOIN Steam_Sample.App_Userbase c
    ON u.appid = c.appid
WHERE c.total_player > 0;
```

9. Redo User-Friend Network with Smaller Sample

This query creates the `SUser_F` table by redoing the user-friend relationship using the smaller sample, ensuring that the new network is based on the filtered user base.

```
Create Table Steam_Sample.SUser_F AS
Select u.user_id, f.friend_id
FROM Steam_Sample.S_User u
LEFT JOIN Steam_Sample.User_F f
```

```
ON u.user_id = f.user_id order by u.user_id;
```

10. Get Second-Degree Friends Network

This query creates the `SUser_F_IF` table, capturing second-degree friends from the smaller user sample by joining the `SUser_F` table with `Friends`.

```
Create table Steam_Sample.SUser_F_IF AS
SELECT sf.user_id,
       sf.friend_id,
       f2.steamid_b AS IF_id
FROM Steam_Sample.SUser_F sf
Left JOIN steam_data.Friends f2
    ON sf.friend_id = f2.steamid_a
    AND f2.steamid_b != sf.user_id;
```

11. Get Friends' Game Ownership

This query creates the `SF_Own` table, capturing the ownership status of games for friends. It checks if each friend (`friend_id`) owns a game (`appid`), and marks ownership as `1` if the game is owned, otherwise `0`.

```
Create table Steam_Sample.SF_Own AS
SELECT DISTINCT sf.friend_id, app.appid,
       CASE
           WHEN g2.steamid IS NOT NULL THEN 1
           ELSE 0
       END AS owns_game
FROM (SELECT DISTINCT friend_id FROM Steam_Sample.SUser_F) sf
CROSS JOIN Steam_Sample.S_App app
LEFT JOIN steam_data.Games_2 g2
    ON sf.friend_id = g2.steamid
    AND app.appid = g2.appid;
```

12. Create IF-Game Panel

This query creates the `SIF_Game` table by generating a cross product of indirect friends (`IF_id`) and games (`appid`), establishing a full panel for further analysis.

```
Create table Steam_Sample.SIF_Game AS
SELECT distinct IF_id, appid
from Steam_Sample.SUser_F_IF
cross join Steam_Sample.S_App;
```

13. Create IF-Game Panel with Ownership Information

This query creates the `SIF_Own` table by adding a column to indicate whether each indirect friend (`IF_id`) owns a game (`appid`), using data from `Games_2`.

```
create table Steam_Sample.SIF_Own
Select u.IF_id, u.appid,
case
when g2.steamid is not null then 1
else 0
```

```
end as owns_game
from Steam_Sample.SIF_Game u
left join steam_data.Games_2 g2
on u.IF_id = g2.steamid
And u.appid = g2.appid;
```

14. Create User-App-Num of Friends Owning the Game Table

This query creates the `SUser_App_NumFOwn` table, which stores the count of direct friends owning each game for each user, by joining the user-friend relationship with the friends' game ownership data.

```
Create Table SUser_App_NumFOwn AS
SELECT sf.user_id AS steamid, fgo.appid, COUNT(distinct fgo.friend_id) AS
num_friends_owning_game
FROM SUser_F sf
JOIN SF_Own fgo
   ON sf.friend_id = fgo.friend_id
   AND fgo.owns_game = 1
GROUP BY sf.user_id, fgo.appid;
```

15. Create User-App-Num of Indirect Friends Owning the Game Table

This query creates the `SUser_App_NumIFOwn` table, which stores the count of indirect friends owning each game for each user, by joining the user-indirect friend relationship with the indirect friends' game ownership data.

```
Create table SUser_App_NumIFOwn AS
SELECT sf.user_id, fgo.appid, COUNT(distinct fgo.IF_id) AS num_IF_own
FROM SUser_F_IF sf
JOIN SIF_Own fgo
   ON sf.IF_id = fgo.IF_id
   AND fgo.owns_game = 1
GROUP BY sf.user_id, fgo.appid;
```

16. Create Big Table with User-App-Num of Friends and Indirect Friends

This query creates the `SUser_App_FIF` table by combining the game ownership information with the counts of friends and indirect friends owning each game for each user.

```
Create Table SUser_App_FIF AS
select u.*,
   IFNULL(nf.num_friends_owning_game, 0) AS num_F_own,
   IFNULL(nif.num_IF_own, 0) AS num_IF_own
FROM SUser_Game u
LEFT JOIN SUser_App_NumFOwn nf
   ON u.user_id = nf.steamid AND u.appid = nf.appid
LEFT JOIN SUSer_App_NumIFOwn num_IF_own
   ON u.user_id = nif.steamid AND u.appid = nif.appid limit 100;
```

17. Append Number of Friends to the Matrix

Hude Hude hh3024@columbia.edu

This step alters the `SUser_App_FIF` table by adding a new column for the total number of friends each user has, joining it with the `SUser_F_Count` table to populate the new column.

```
ALTER TABLE SUser_App_FIF
ADD COLUMN num_friend bigint;

UPDATE SUser_App_FIF AS t1
LEFT JOIN SUser_F_Count AS t2
   ON t1.user_id = t2.user_id
SET t1.num_friend = t2.num_friend;  -- Set num_friend based on the value in SUser_F_Count
```

18. Find IF2 (Third-Degree Friends)

This query creates the `S_IF2` table by retrieving second-degree friends (friends of indirect friends) from the `S_Connected_IF` table, using a `UNION` to capture relationships in both directions.

```
create table Steam_Sample.S_IF2 AS
SELECT s.IF_id, f.steamid_b AS IF_F_id
FROM S_Connected_IF s
JOIN steam_data.Friends f
   ON s.IF_id = f.steamid_a

UNION

SELECT s.IF_id, f.steamid_a AS IF_F_id
FROM S_Connected_IF s
JOIN steam_data.Friends f
   ON s.IF_id = f.steamid_b;
```

…