Problem 1917: Leetcodify Friends Recommendations

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Table:

Listens

+-------------+---------+ | Column Name | Type | +-------------+---------+ | user_id | int | | song_id | int | | day | date | +-------------+---------+ This table may contain duplicates (In other words, there is no primary key for this table in SQL). Each row of this table indicates that the user user_id listened to the song song_id on the day day.

Table:

Friendship

+---------------+---------+ | Column Name | Type | +---------------+---------+ | user1_id | int | | user2_id | int | +---------------+---------+ In SQL,(user1_id, user2_id) is the primary key for this table. Each row of this table indicates that the users user1_id and user2_id are friends. Note that user1_id < user2_id.

Recommend friends to Leetcodify users. We recommend user

x

to user

y

if:

Users

x

and

y

are not friends, and

Users

x

and

y

listened to the same three or more different songs

on the same day

.

Note that friend recommendations are

unidirectional

, meaning if user

x

and user

y

should be recommended to each other, the result table should have both user

x

recommended to user

y

and user

y

recommended to user

x

. Also, note that the result table should not contain duplicates (i.e., user

y

should not be recommended to user

x

multiple times.).

Return the result table in

any order

.

The result format is in the following example.

Example 1:

Input:

Listens table: +---------+---------+------------+ | user_id | song_id | day | +---------+---------+------------+ | 1 | 10 | 2021-03-15 | | 1 | 11 | 2021-03-15 | | 1 | 12 | 2021-03-15 | | 2 | 10 | 2021-03-15 | | 2 | 11 | 2021-03-15 | | 2 | 12 | 2021-03-15 | | 3 | 10 | 2021-03-15 | | 3 | 11 | 2021-03-15 | | 3 | 12 | 2021-03-15 | | 4 | 10 | 2021-03-15 | | 4 | 11 | 2021-03-15 | | 4 | 13 | 2021-03-15 | | 5 | 10 | 2021-03-16 | | 5 | 11 | 2021-03-16 | | 5 | 12 | 2021-03-16 | +---------+---------+------------+ Friendship table: +----------+----------+ | user1_id | user2_id | +----------+----------+ | 1 | 2 | +----------+----------+

Output:

+---------+----------------+ | user_id | recommended_id | +---------+----------------+ | 1 | 3 | | 2 | 3 | | 3 | 1 | | 3 | 2 | +---------+----------------+

Explanation:

Users 1 and 2 listened to songs 10, 11, and 12 on the same day, but they are already friends. Users 1 and 3 listened to songs 10, 11, and 12 on the same day. Since they are not friends, we recommend them to each other. Users 1 and 4 did not listen to the same three songs. Users 1 and 5 listened to songs 10, 11, and 12, but on different days.

Similarly, we can see that users 2 and 3 listened to songs 10, 11, and 12 on the same day and are not friends, so we recommend them to each other.

Code Snippets

MySQL:

```
# Write your MySQL query statement below
```

MS SQL Server:

```
/* Write your T-SQL query statement below */
```

PostgreSQL:

```
-- Write your PostgreSQL query statement below
```

Oracle:

```
/* Write your PL/SQL query statement below */
```

Pandas:

```
import pandas as pd

def recommend_friends(listens: pd.DataFrame, friendship: pd.DataFrame) ->
pd.DataFrame:
```

Solutions

MySQL Solution:

```
# Write your MySQL query statement below
```

MS SQL Server Solution:

```
/* Write your T-SQL query statement below */
```

PostgreSQL Solution:

```
-- Write your PostgreSQL query statement below
```

Oracle Solution:

```
/* Write your PL/SQL query statement below */
```

Pandas Solution:

```
import pandas as pd

def recommend_friends(listens: pd.DataFrame, friendship: pd.DataFrame) ->
pd.DataFrame:
```