Consider the following ....

For each user, we want

- the number of questions (PostTypeId = 1),
- the number of answers
- the number of badges
- the total number of posts by that person

As we discussed in class on Tuesday, we can do this with 3 separate queries and the JOIN these. That is fine, but one has to verify that it gets the correct answer and doesn't double count any tuples or omit any and that we get answers for all users.

However, here we will do it with a single query in which we combine the User information, the Badges, the Questions from the Posts table and then the Answers from the Posts tables as 4 different tables. So this shows subsetting Posts when we JOIN it for the Questions and again for the Answers.

The question focuses on users so we want to end up with tuples corresponding to each user and therefore GROUP BY user.

Since we also want to include Users who do not have badges as well as those with badges we can do a left join on Users. That should come first.

There is a sample database in this directory with Users, Badges and Posts tables. These are very small and you can check what the results should be manually (or in R.) The information is in the README.md file

To get the tuples, we might use

```
SELECT *
FROM Users
LEFT JOIN Badges
       ON Users.Id = Badges.UserId
LEFT JOIN Posts AS Q
      ON Users.Id = Q.UserId AND Badges.UserId = Q.UserId AND Q.PostTypeId = 1
LEFT JOIN Posts AS A
      ON Users.Id = A.UserId AND Badges.UserId = A.UserId AND A.PostTypeId = 2
```

Note that we are

- doing LEFT JOINs
- matching the Users' Id to the Badges' UserId
- combining the Posts as Questions by
  - matching Users' Id to Posts.Id and
  - the Badges' and Posts' UserId and
  - restricting the PostTypeId to equal 1
- similarly for matching the Posts as Answers

Do we need the match on Badges.UserId = Q.UserId?

The following version does not have the constraint on Badges.UserId = Q.UserId and returns one more row.

```
SELECT *
FROM Users
LEFT JOIN Badges
       ON Users.Id = Badges.UserId
LEFT JOIN Posts AS Q
      ON Users.Id = Q.UserId AND Q.PostTypeId = 1
LEFT JOIN Posts AS A
      ON Users.Id = A.UserId AND A.PostTypeId = 2
```

Let's simplify this to only join Users, Badges and Posts as Questions and perform the two queries

```
db = dbConnect(SQLite(), "test")
do = function(q) dbGetQuery(db, q)
```

```
a = do("SELECT *
FROM Users
LEFT JOIN Badges
       ON Users.Id = Badges.UserId
LEFT JOIN Posts AS Q
      ON Users.Id = Q.UserId AND Badges.UserId = Q.UserId AND Q.PostTypeId = 1")
```

```
a2 = do("SELECT *
FROM Users
LEFT JOIN Badges
       ON Users.Id = Badges.UserId
LEFT JOIN Posts AS Q
      ON Users.Id = Q.UserId AND Q.PostTypeId = 1")
```

```
a:
   Id UserId Badge   Id UserId PostTypeId ParentId
1  u1     u1     A   p4     u1          1
2  u1     u1     A   p8     u1          1
3  u1     u1     B   p4     u1          1
4  u1     u1     B   p8     u1          1
5  u2     u2     A   p6     u2          1
6  u2     u2     C   p6     u2          1
7  u2     u2     D   p6     u2          1
8  u3   <NA>  <NA> <NA>   <NA>         NA     <NA>
9  u4   <NA>  <NA> <NA>   <NA>         NA     <NA>
10 u5     u5     A <NA>   <NA>         NA     <NA>
11 u5     u5     C <NA>   <NA>         NA     <NA>
```

```
12 u5     u5     W <NA>   <NA>          NA     <NA>
13 u5     u5     X <NA>   <NA>          NA     <NA>
14 u5     u5     Z <NA>   <NA>          NA     <NA>
```

a2:

```
   Id UserId Badge   Id UserId PostTypeId ParentId
1  u1     u1     A   p4     u1          1
2  u1     u1     A   p8     u1          1
3  u1     u1     B   p4     u1          1
4  u1     u1     B   p8     u1          1
5  u2     u2     A   p6     u2          1
6  u2     u2     C   p6     u2          1
7  u2     u2     D   p6     u2          1
8  u3   <NA>  <NA>   p1     u3          1
9  u3   <NA>  <NA>   p2     u3          1
10 u4   <NA>  <NA> <NA>   <NA>          NA     <NA>
11 u5     u5     A <NA>   <NA>          NA     <NA>
12 u5     u5     C <NA>   <NA>          NA     <NA>
13 u5     u5     W <NA>   <NA>          NA     <NA>
14 u5     u5     X <NA>   <NA>          NA     <NA>
15 u5     u5     Z <NA>   <NA>          NA     <NA>
```

The second version has an additional row for u3. However, it also includes the details for the questions from User u3, while the first version has NAs for these. This is because in the first query's table (a), the Badges.UserId is NULL/NA and so doesn't match the Posts.UserId values.

So we want to remove that extra constraint as it fails to handle the cases where there are no Badges.

So let's look at the tuples we get back from our "correct" query:

```
d = do("SELECT *
FROM Users
LEFT JOIN Badges
       ON Users.Id = Badges.UserId
LEFT JOIN Posts AS Q
      ON Users.Id = Q.UserId AND Q.PostTypeId = 1
LEFT JOIN Posts AS A
      ON Users.Id = A.UserId AND A.PostTypeId = 2")
```

```
Users   Badges             Posts: Questions              Posts: Answers
----- -------------  ------------------------------   ----------------------------
   Id UserId Badge   Id UserId PostTypeId ParentId   Id UserId PostTypeId ParentId
1  u1     u1     A   p4     u1          1            p3     u1          2       p2
2  u1     u1     A   p8     u1          1            p3     u1          2       p2
```

```
3  u1     u1     B   p4     u1          1           p3     u1          2       p2
4  u1     u1     B   p8     u1          1           p3     u1          2       p2
5  u2     u2     A   p6     u2          1           p5     u2          2       p1
6  u2     u2     C   p6     u2          1           p5     u2          2       p1
7  u2     u2     D   p6     u2          1           p5     u2          2       p1
8  u3   <NA>   <NA>   p1     u3          1         <NA>   <NA>         NA     <NA>
9  u3   <NA>   <NA>   p2     u3          1         <NA>   <NA>         NA     <NA>
10 u4   <NA>   <NA> <NA>   <NA>         NA     <NA> <NA>   <NA>         NA     <NA>
11 u5     u5     A <NA>   <NA>         NA     <NA>   p7     u5          2       p6
12 u5     u5     C <NA>   <NA>         NA     <NA>   p7     u5          2       p6
13 u5     u5     W <NA>   <NA>         NA     <NA>   p7     u5          2       p6
14 u5     u5     X <NA>   <NA>         NA     <NA>   p7     u5          2       p6
15 u5     u5     Z <NA>   <NA>         NA     <NA>   p7     u5          2       p6
```

We'll compute the answers we expect first in R:

How many badges does each user have? We count the number of unique Badge names (or their unique identifier if two badges can have the same name.)

```
nb = tapply(d$Badge, d$Id, function(x) length(unique(x[!is.na(x)])))
u1 u2 u3 u4 u5
 2  3  0  0  5
```

To compute how many questions, we count the number of unique Id values from the Posts:Questions table:

```
nq = tapply(d[[4]], d$Id, function(x) length(unique(x[!is.na(x)])))
u1 u2 u3 u4 u5
 2  1  2  0  0
```

Will these already be unique? or will some be duplicated because of the JOIN.

To compute how many questions, we count the number of unique Id values from the Posts:Questions table:

```
na = tapply(d[[8]], d$Id, function(x) length(unique(x[!is.na(x)])))
u1 u2 u3 u4 u5
 1  1  0  0  1
```

The combined results are

```
   nq na nb
u1  2  1  2
u2  1  1  3
u3  2  0  0
u4  0  0  0
u5  0  1  5
```

(questions, answers and badges.)

Let's compute these in SQL

```
ans = do("
SELECT COUNT(DISTINCT Q.Id) AS NumQuestions,
       COUNT(DISTINCT A.Id) AS NumAnswers,
       COUNT(DISTINCT Badge) AS NumBadges
FROM Users
LEFT JOIN Badges
       ON Users.Id = Badges.UserId
LEFT JOIN Posts AS Q
      ON Users.Id = Q.UserId AND Q.PostTypeId = 1
LEFT JOIN Posts AS A
      ON Users.Id = A.UserId AND A.PostTypeId = 2
GROUP By Users.Id;
")
```

```
  NumQuestions NumAnswers NumBadges
1            2          1         2
2            1          1         3
3            2          0         0
4            0          0         0
5            0          1         5
```