Homework 3: StackExchange

Name: Chitra Mukherjee

The 13 problems that I completed are: 1, 2, 3, 4, 5, 6, 7, 8, 12, 13, 14, 15, 16 Questions that I partially completed and showed all my work for but not quite done: 9, 10, 18

We need to install the necessary libraries and packages.

```
if("DBI" %in% rownames(installed.packages()) == FALSE) {
  install.packages("DBI")
}
if("RSQLite" %in% rownames(installed.packages()) == FALSE) {
  install.packages("RSQLite")
}
library(DBI)
library(RSQLite)
```

We will now create the database connection so we can see the format of data in our StackExchange database.

```
cur_dir = './' #current directory is STA141B folder
print(getwd())
```

```
## [1] "/Users/chitramac/Desktop/STA 141B"
```

```
db_filepath = paste0(cur_dir, 'stats.stackexchange.db')
db = dbConnect(SQLite(), db_filepath)
print("List of all the table names in our database:")
```

```
## [1] "List of all the table names in our database:"
```

```
dbListTables(db)
```

```
##  [1] "BadgeClassMap"     "Badges"            "CloseReasonMap"
##  [4] "Comments"          "LinkTypeMap"       "PostHistory"
##  [7] "PostHistoryTypeId" "PostLinks"         "PostTypeIdMap"
## [10] "Posts"             "TagPosts"          "Users"
## [13] "VoteTypeMap"       "Votes"
```

Questions

Overall Notes on the Tables

In the Posts table: - UserId: Id of the user who posted a comment, question, answer, etc. - ParentId: Each answer has a ParentId that identifies the question to which it is an answer (questions don't have a parent id since they are the parent so that field is empty) - Id: Id of the post itself

1. How many users are there? This question will require reading in the table called Users.

I found the distinct Id values since that would associate with the unique Ids rather than depending on unique names. This is because the Id values correlate to each row in the table, and these are unique.

I am considering the number of Users to be the unique number of IDs that there are in the Users table.

I didn't want to count the number of unique DisplayNames since I thought that there could be more than one name for accounts (for spam accounts, for example).

Although the DisplayName would most likely need to also be unique as well since stack overflow wouldn't want duplicate names for users, it felt safer to rely on Ids for unique values since each user would be associated with 1 ID.

```
users_db = dbReadTable(db, "Users")
print("Column names in the Users table: ")
```

```
## [1] "Column names in the Users table: "
```

```
print(colnames(users_db))
```

```
##  [1] "Id"             "Reputation"     "CreationDate"   "DisplayName"
##  [5] "LastAccessDate" "WebsiteUrl"     "Location"       "AboutMe"
##  [9] "Views"          "UpVotes"        "DownVotes"      "AccountId"
```

```
head(users_db)
```

```
##   Id Reputation           CreationDate  DisplayName          LastAccessDate
## 1 -1          1 2010-07-19T06:55:26.860    Community 2010-07-19T06:55:26.860
## 2  2        101 2010-07-19T14:01:36.697 Geoff Dalgas 2019-02-07T22:01:05.890
## 3  3        101 2010-07-19T15:34:50.507 Jarrod Dixon 2019-02-07T16:22:42.717
## 4  4        101 2010-07-19T19:03:27.400       Emmett 2016-11-24T19:37:25.313
## 5  5      12131 2010-07-19T19:03:57.227        Shane 2022-12-07T19:30:33.150
## 6  6        832 2010-07-19T19:04:07.647       Harlan 2022-07-14T17:07:11.783
##                       WebsiteUrl             Location
## 1 http://meta.stackexchange.com/    on the server farm
## 2       http://stackoverflow.com         Corvallis, OR
## 3         http://jarroddixon.com Johnson City, TN, USA
## 4   http://minesweeperonline.com     San Francisco, CA
## 5        http://www.statalgo.com          New York, NY
## 6  http://www.harlan.harris.name     Brooklyn, NY, USA
##
## 1
## 2
## 3
## 4
## 5 <p>Quantitative researcher focusing on statistics and machine learning methods in finance. Primaril
## 6
##   Views UpVotes DownVotes AccountId
## 1  2713   17774      9322        -1
## 2    47       3         0         2
## 3    47      23         0         3
## 4    23       0         0      1998
## 5  2089     684         5     54503
## 6   214      65         0     46050
```

```
print("Number of rows in the Users table:")
```

```
## [1] "Number of rows in the Users table:"
```

```
print(length(users_db$Id))
```

```
## [1] 321677
```

```
q = "SELECT COUNT(DISTINCT Id) FROM Users"
dbGetQuery(db, q)
```

```
##   COUNT(DISTINCT Id)
## 1             321677
```

There are **321,677** users.

2. How many users joined since 2020? (Convert the CreationDate to a year) This question will require reading in the Users table. I used this website to help me convert to a year: https://www.neonscience.org/resources/learning-hub/tutorials/dc-convert-date-time-posix-r.

We will specifically get data from the CreationDate column and convert it into POSIXct class so we can extract data which is in a certain year range.

```
dateData = dbGetQuery(db, "SELECT CreationDate FROM Users")
dateData$CreationDate = as.POSIXct(strptime(dateData$CreationDate, format = "%Y"))
head(dateData)
```

```
##   CreationDate
## 1   2010-05-22
## 2   2010-05-22
## 3   2010-05-22
## 4   2010-05-22
## 5   2010-05-22
## 6   2010-05-22
```

```
years = data.frame(as.numeric(strftime(dateData$CreationDate, format="%Y")))
print("Number of users who joined since 2020:")
```

```
## [1] "Number of users who joined since 2020:"
```

```
print(length(which(years >= 2020)))
```

```
## [1] 100796
```

There are 100,796 users that joined since 2020.

We can also do this in sequel: I used this website to help me filter based on a date. https://stackoverflow.com/questions/9532668/list-rows-after-specific-date https://stackoverflow.com/questions/658395/find-the-number-of-columns-in-a-table#:~:text=Query%20to%20count%20the%20number,of%20columns%20you%20want%20returne

```
query = "SELECT COUNT(*) FROM Users WHERE CreationDate >= '2020-01-01'"
dbGetQuery(db, query)
```

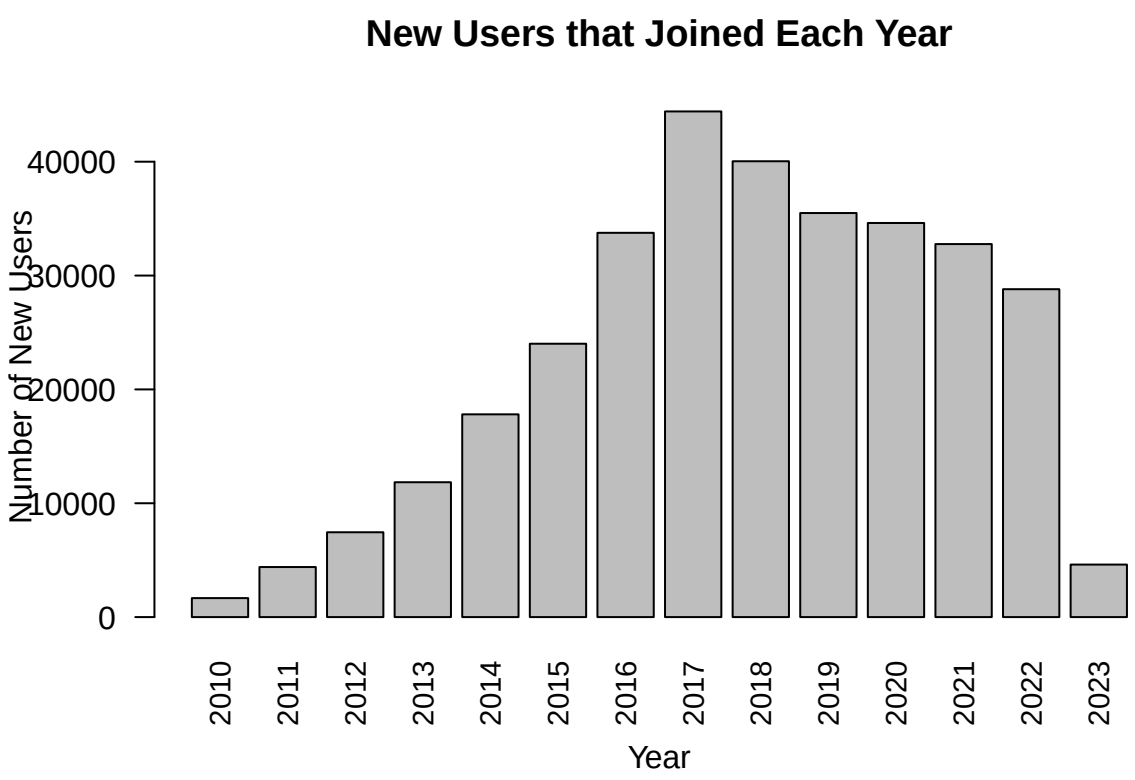
```
##   COUNT(*)
## 1   100796
```

3. How many users joined each year? Describe this with a plot, commenting on any anomalies?
We will use the same dataframe that we used in the previous problem to count the frequency of values for each year.

```
users_each_year = data.frame(table(years))
colnames(users_each_year) = c("Year", "Users Each Year")
users_each_year
```

```
##    Year Users Each Year
## 1  2010            1668
## 2  2011            4396
## 3  2012            7450
## 4  2013           11846
## 5  2014           17809
## 6  2015           24012
## 7  2016           33753
## 8  2017           44416
## 9  2018           40040
## 10 2019           35491
## 11 2020           34617
## 12 2021           32765
## 13 2022           28801
## 14 2023            4613
```

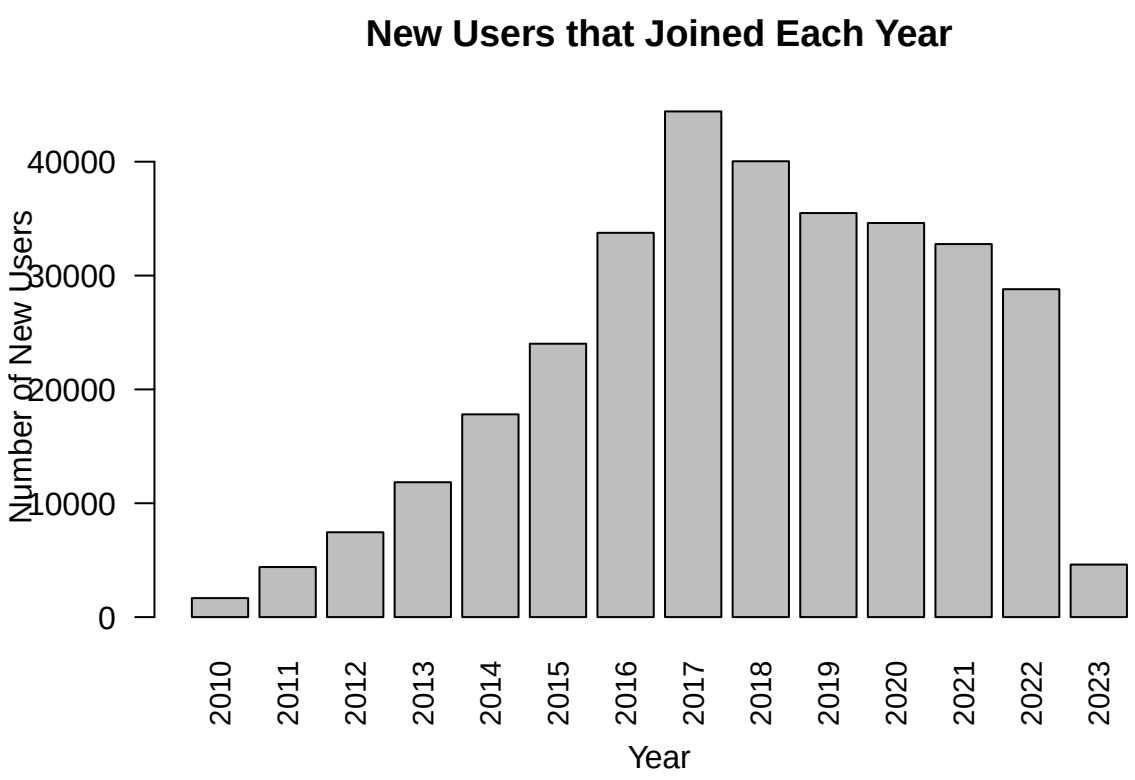
```
barplot(users_each_year$`Users Each Year`, xlab = "Year", ylab = "Number of New Users", names.arg=users
        cex.names = 0.9)
```



In our barchart, we see that the trend is increasing for the number of new users joining starting from 2010 all the way up to 2017. However, starting in 2018, there were fewer new users joining in subsequent years, and the number of new users in 2023 dropped significantly. This could be interpreted as an outlier since there were at least 20,00 to 30,000 new users consistently joining in previous years from 2014 to 2022, and in 2023 the number of new users dropped to less than 10,000.

Now I will do the same code in sequel: This resource helped me get times from the database in the structure that I wanted them in. https://www.w3resource.com/sqlite/sqlite-strftime.php This resource helped me get the frequency table. https://www.c-sharpcorner.com/blogs/sql-query-to-find-out-the-frequency-of-each-element-in-a-column1

```
#dateData = dbGetQuery(db, "SELECT strftime('%Y',CreationDate) As Year FROM Users WHERE CreationDate >=

q = "SELECT strftime('%Y',CreationDate) as Year, COUNT(CreationDate) AS Frequency
  FROM Users
  GROUP BY strftime('%Y',CreationDate)"
freqs = dbGetQuery(db, q)
barplot(freqs$Frequency, xlab = "Year", ylab = "Number of New Users", names.arg=freqs$Year, main = "New
```



4. How many different types of posts are there in the Posts table? Get the description of the types from the PostTypeIdMap table. In other words, create a table with the description of each post type and the number of posts of that type, and arrange it from most to least occurrences. First, I will examine the structure of the Posts table.

```
posts_db = dbReadTable(db, "Posts")
posttypeidmap_db = dbReadTable(db, "PostTypeIdMap")
head(posts_db)
```

```
##   Id PostTypeId AcceptedAnswerId            CreationDate Score ViewCount
## 1  1          1               15 2010-07-19T19:12:12.510    49      5364
## 2  2          1               59 2010-07-19T19:12:57.157    34     33588
## 3  3          1                5 2010-07-19T19:13:28.577    71      6622
## 4  4          1              135 2010-07-19T19:13:31.617    23     45393
## 5  5          2                0 2010-07-19T19:14:43.050    90         0
## 6  6          1                0 2010-07-19T19:14:44.080   486    172176
##
## 1
## 2
## 3
## 4
## 5
## 6 <p>Last year, I read a blog post from <a href="http://anyall.org/">Brendan O'Connor</a> entitled <a
##   OwnerUserId        LastActivityDate
## 1           8 2020-11-05T09:44:51.710
```

```
## 2          24 2022-11-23T13:03:42.033
## 3          18 2022-11-27T23:33:13.540
## 4          23 2010-09-08T03:00:19.690
## 5          23 2010-07-19T19:21:15.063
## 6           5 2021-01-19T17:59:15.653
##                                                                Title
## 1                                  Eliciting priors from experts
## 2                                                What is normality?
## 3 What are some valuable Statistical Analysis open source projects?
## 4        Assessing the significance of differences in distributions
## 5
## 6                The Two Cultures: statistics vs. machine learning?
##                                       Tags AnswerCount CommentCount
## 1            <bayesian><prior><elicitation>           6            1
## 2     <distributions><normality-assumption>           7            1
## 3                   <software><open-source>          19            3
## 4 <distributions><statistical-significance>           5            2
## 5                                                     0            3
## 6          <machine-learning><pac-learning>          20           10
##   ContentLicense LastEditorDisplayName            LastEditDate LastEditorUserId
## 1   CC BY-SA 2.5
## 2   CC BY-SA 2.5                user88 2010-08-07T17:56:44.800
## 3   CC BY-SA 2.5                       2011-02-12T05:50:03.667              183
## 4   CC BY-SA 2.5
## 5   CC BY-SA 2.5                       2010-07-19T19:21:15.063               23
## 6   CC BY-SA 3.0                       2017-04-08T17:58:18.247            11887
##       CommunityOwnedDate ParentId OwnerDisplayName ClosedDate FavoriteCount
## 1
## 2
## 3 2010-07-19T19:13:28.577
## 4
## 5 2010-07-19T19:14:43.050        3
## 6 2010-08-09T13:05:50.603
```

```
head(posttypeidmap_db)
```

```
##   id                value
## 1  1             Question
## 2  2               Answer
## 3  3    Orphaned tag wiki
## 4  4     Tag wiki excerpt
## 5  5             Tag wiki
## 6  6 Moderator nomination
```

Although there are 8 types of Post Ids in general as found in the PostTypeMapId table, there are only 7 distinct types of posts present in the Posts table.

```
query = "SELECT COUNT(DISTINCT Posts.PostTypeId), COUNT(DISTINCT PostTypeIdMap.id) FROM Posts, PostTypeI
dbGetQuery(db, query)
```

```
##   COUNT(DISTINCT Posts.PostTypeId) COUNT(DISTINCT PostTypeIdMap.id)
## 1                                7                                8
```

Now we will construct the overall table.

```
query = "SELECT Posts.PostTypeId AS Id, COUNT(Posts.PostTypeId) AS Frequency, PostTypeIdMap.value AS Ty
FROM Posts, PostTypeIdMap
WHERE PostTypeIdMap.id = Posts.PostTypeID
GROUP BY PostTypeId
ORDER BY Frequency DESC"
numTypesOfPosts = dbGetQuery(db, query)
head(numTypesOfPosts)
```

```
##   Id Frequency                Type
## 1  1    204370            Question
## 2  2    197928              Answer
## 3  5      1444            Tag wiki
## 4  4      1444     Tag wiki excerpt
## 5  6        23 Moderator nomination
## 6  3         6    Orphaned tag wiki
```

I didn't use JOIN so I will now use JOIN to implement the table above:

```
query = "SELECT Posts.PostTypeId AS Id, PostTypeIdMap.value AS Type, COUNT(Posts.PostTypeId) AS Freq
FROM Posts
LEFT JOIN PostTypeIdMap
ON Posts.PostTypeId = PostTypeIdMap.id
GROUP BY Posts.PostTypeId
ORDER BY Freq DESC"
head(dbGetQuery(db, query))
```

```
##   Id                Type   Freq
## 1  1            Question 204370
## 2  2              Answer 197928
## 3  5            Tag wiki   1444
## 4  4     Tag wiki excerpt   1444
## 5  6 Moderator nomination     23
## 6  3    Orphaned tag wiki      6
```

In order to verify that this is correct, I am going to sum up the frequency in my table and make sure that it adds up to the total number of posts there are.

```
query = "SELECT SUM(Freq) FROM (SELECT Posts.PostTypeId AS Id, PostTypeIdMap.value AS Type, COUNT(Posts
FROM Posts
LEFT JOIN PostTypeIdMap
ON Posts.PostTypeId = PostTypeIdMap.id
GROUP BY Posts.PostTypeId
ORDER BY Freq DESC)"
dbGetQuery(db, query)
```

```
##   SUM(Freq)
## 1    405220
```

To verify that this is correct, I am going to count the number of rows that are in the Posts table in R.

```
nrow(posts_db)
```

```
## [1] 405220
```

5. How many posted questions are there? I am interpreting this question as asking us to find how many of the posts are type question. In the table in the previous problem, we found the frequencies of different types of posts, and the category for posted questions was Id 1.

```
#two different ways to query
query1 = "SELECT COUNT(*) FROM Posts WHERE Posts.PostTypeId = 1"
query2 = "SELECT COUNT(*)
FROM Posts, PostTypeIdMap
WHERE Posts.PostTypeId = PostTypeIdMap.Id AND PostTypeIdMap.value = 'Question'"
dbGetQuery(db, query2)
```

```
##   COUNT(*)
## 1   204370
```

There are 204,370 posted questions.

I will now implement the same concept using JOIN.

```
query = "SELECT COUNT(*)
FROM Posts
JOIN PostTypeIdMap
ON Posts.PostTypeId = PostTypeIdMap.Id AND PostTypeIdMap.value = 'Question'"
dbGetQuery(db, query)
```

```
##   COUNT(*)
## 1   204370
```

6. What are the top 50 most common tags on questions? For each of the top 50 tags on questions, how many questions are there for each tag. Through further investigation of the Posts and TagPosts tables, we see that only posts of type question (with PostTypeIdMap.id as 1) have tags. We can verify this in R code.

```
posts_db["Tags"][posts_db["Tags"] == ""] <- NA #set empty tag values to null
question_tags = posts_db[which(!is.na(posts_db$Tags)),]
print(nrow(question_tags))
```

```
## [1] 204370
```

```
head(question_tags)
```

```
##   Id PostTypeId AcceptedAnswerId            CreationDate Score ViewCount
## 1  1          1               15 2010-07-19T19:12:12.510    49      5364
## 2  2          1               59 2010-07-19T19:12:57.157    34     33588
## 3  3          1                5 2010-07-19T19:13:28.577    71      6622
## 4  4          1              135 2010-07-19T19:13:31.617    23     45393
```

```
## 6  6          1                0 2010-07-19T19:14:44.080   486    172176
## 7  7          1               18 2010-07-19T19:15:59.303   103     42426
##
## 1
## 2
## 3
## 4
## 6 <p>Last year, I read a blog post from <a href="http://anyall.org/">Brendan O'Connor</a> entitled <a
## 7
##   OwnerUserId        LastActivityDate
## 1           8 2020-11-05T09:44:51.710
## 2          24 2022-11-23T13:03:42.033
## 3          18 2022-11-27T23:33:13.540
## 4          23 2010-09-08T03:00:19.690
## 6           5 2021-01-19T17:59:15.653
## 7          38 2022-11-30T05:37:29.877
##                                                          Title
## 1                                  Eliciting priors from experts
## 2                                          What is normality?
## 3 What are some valuable Statistical Analysis open source projects?
## 4       Assessing the significance of differences in distributions
## 6              The Two Cultures: statistics vs. machine learning?
## 7                          Locating freely available data samples
##                                      Tags AnswerCount CommentCount
## 1           <bayesian><prior><elicitation>           6            1
## 2    <distributions><normality-assumption>           7            1
## 3                  <software><open-source>          19            3
## 4 <distributions><statistical-significance>           5            2
## 6          <machine-learning><pac-learning>          20           10
## 7   <dataset><sample><population><teaching>          25            2
##   ContentLicense LastEditorDisplayName            LastEditDate LastEditorUserId
## 1   CC BY-SA 2.5
## 2   CC BY-SA 2.5               user88 2010-08-07T17:56:44.800
## 3   CC BY-SA 2.5                      2011-02-12T05:50:03.667              183
## 4   CC BY-SA 2.5
## 6   CC BY-SA 3.0                      2017-04-08T17:58:18.247            11887
## 7   CC BY-SA 2.5                      2013-09-26T21:50:36.963              253
##       CommunityOwnedDate ParentId OwnerDisplayName ClosedDate FavoriteCount
## 1
## 2
## 3 2010-07-19T19:13:28.577
## 4
## 6 2010-08-09T13:05:50.603
## 7 2010-07-20T20:50:48.483
```

As found in the previous question, there are 204,370 posts of type question and there are also 204,370 posts with tags which allows us to conclude that only posts which are questions have tags. This information is in the TagPosts table, so we can create a frequency table of the occurrence of those tags.

```
query = "SELECT Posts.Tags As Tags
FROM Posts
WHERE Posts.PostTypeId = 1"
questionTagGroups = dbGetQuery(db, query)
```

```
tagposts_db = dbReadTable(db, "TagPosts")
head(tagposts_db)
```

```
##   Id                  Tag
## 1  1             bayesian
## 2  1                prior
## 3  1          elicitation
## 4  2        distributions
## 5  2 normality-assumption
## 6  3             software
```

```
head(questionTagGroups)
```

```
##                                       Tags
## 1            <bayesian><prior><elicitation>
## 2     <distributions><normality-assumption>
## 3                   <software><open-source>
## 4 <distributions><statistical-significance>
## 5          <machine-learning><pac-learning>
## 6   <dataset><sample><population><teaching>
```

Saisha Hongal and I worked together to write this frequency table. These are the top 50 most common tags on questions. The associated frequency column gives us the number of questions there are for each tag.

```
query = "SELECT DISTINCT Tag, COUNT(Tag) AS Number_of_Posts
FROM TagPosts
GROUP BY Tag
ORDER BY Number_of_Posts DESC
LIMIT 50"
dbGetQuery(db, query)
```

```
##                         Tag Number_of_Posts
## 1                         r           28495
## 2                regression           28146
## 3          machine-learning           19355
## 4               time-series           13745
## 5               probability           11894
## 6         hypothesis-testing           10091
## 7              distributions            9147
## 8                 self-study            7985
## 9            neural-networks            7793
## 10                  bayesian            7628
## 11                  logistic            7507
## 12  mathematical-statistics            7455
## 13           classification            6654
## 14              correlation            6074
## 15 statistical-significance            6038
## 16      normal-distribution            5877
## 17              mixed-model            5837
## 18      multiple-regression            5265
## 19                    anova            5100
```

```
## 20                   python           4605
## 21      confidence-interval           4367
## 22 generalized-linear-model           4276
## 23                 variance           4042
## 24               clustering           3932
## 25             forecasting            3726
## 26                   t-test           3486
## 27         categorical-data           3468
## 28         cross-validation           3385
## 29                      pca           3333
## 30       maximum-likelihood           3209
## 31               estimation           3159
## 32                lme4-nlme           3156
## 33                 sampling           3104
## 34        predictive-models           2971
## 35                 survival           2960
## 36       data-visualization           2949
## 37                inference           2899
## 38                    arima           2818
## 39                  p-value           2709
## 40                     mean           2705
## 41             optimization           2688
## 42            least-squares           2682
## 43       repeated-measures            2612
## 44         chi-squared-test           2559
## 45                 modeling           2501
## 46               references           2451
## 47    multivariate-analysis           2430
## 48             econometrics           2422
## 49              interaction           2421
## 50             linear-model           2412
```

7. How many tags are in most questions? In the TagPosts table, each post is categorized by an Id number, and each tag is represented by that same Id number.

For example, the first question post has 3 tags: bayesian, prior, and elicitation so the TagPost table has 3 columns with each of these tags, and each is associated with the id 1.

```
posts_db[1,]$Tags
```

```
## [1] "<bayesian><prior><elicitation>"
```

Therefore, we will find the frequency of tags for each id.

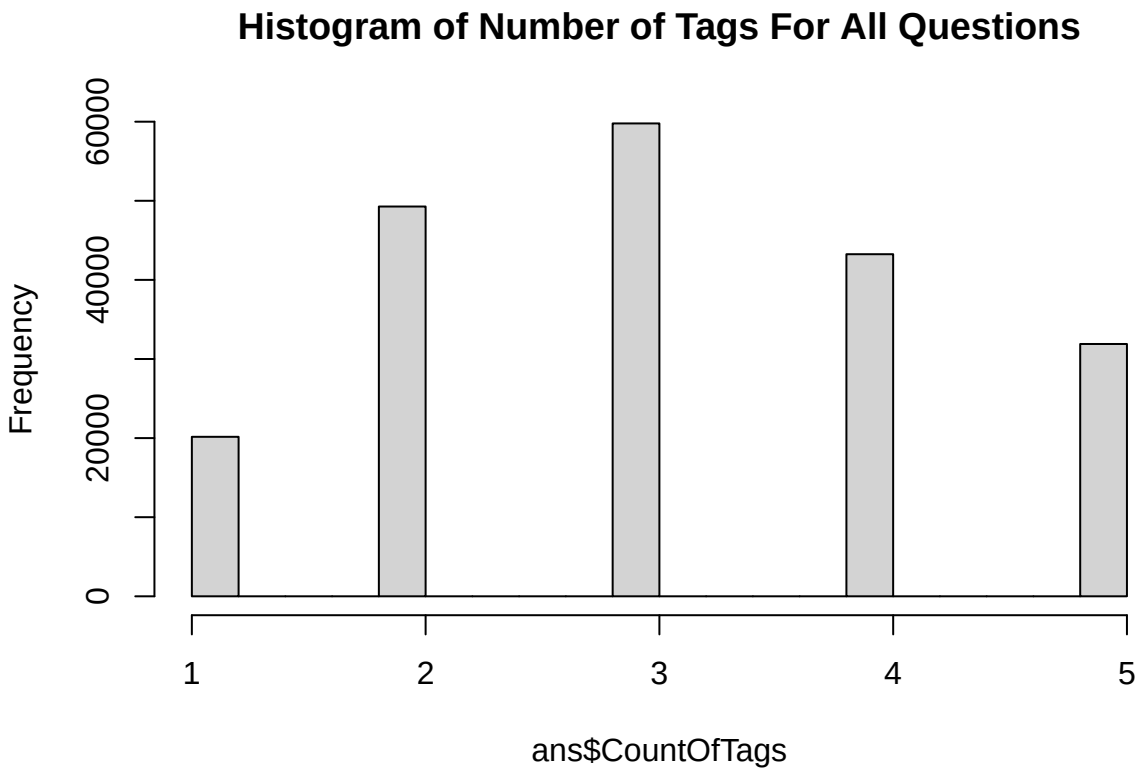
```
query = "SELECT Id AS Question_Id, COUNT(Id) AS CountOfTags FROM TagPosts GROUP BY Id"

ans = dbGetQuery(db, query)
head(ans)
```

```
##   Question_Id CountOfTags
## 1           1           3
## 2           2           2
## 3           3           2
```

```
## 4           4           2
## 5           6           2
## 6           7           4
```

Now we will plot the distribution of the number of tags there are for all question posts with a histogram.

```
hist(ans$CountOfTags, main = "Histogram of Number of Tags For All Questions")
```



In this histogram, we see that most questions – around 60,000 of them – have 3 tags. Our histogram seems to follow a bell curve. Therefore, we would assume that the average number of tags per post would fall around 3, and we will verify this below using SQL commands.

I used this website to help me find how to get the average of a count column. https://www.sqlteam.com/forums/topic.asp?TOPIC_ID=33152

```
#query = "SELECT DISTINCT Id, COUNT(Id) AS Num_Tags FROM TagPosts GROUP BY Id"
query = "SELECT AVG(CountOfTags) FROM
(
SELECT COUNT(Id) AS CountOfTags FROM TagPosts GROUP BY Id
)"

dbGetQuery(db, query)
```

```
##   AVG(CountOfTags)
## 1          3.08534
```

Therefore, this means that there are 3 tags in most questions since we will round down the average to the nearest whole number.

```
query = "SELECT COUNT(*) AS Num_Answers
FROM Posts, PostTypeIdMap
WHERE Posts.PostTypeId = PostTypeIdMap.Id AND PostTypeIdMap.value = 'Answer'"
dbGetQuery(db, query)
```

8. How many answers are there?

```
##   Num_Answers
## 1      197928
```

There are 197,928 answers posted. Similarly to question 5 which was asking us how many posted questions there are, I am going to look for how many posts of category answers there are and count those.

I will now do this same idea with JOIN.

```
query = "SELECT COUNT(*) AS Num_Ans
FROM Posts
JOIN PostTypeIdMap
ON Posts.PostTypeId = PostTypeIdMap.Id AND PostTypeIdMap.value = 'Answer'"
dbGetQuery(db, query)
```

```
##   Num_Ans
## 1  197928
```

9. What's the most recent question (by date-time) in the Posts table? Find it on the stats.exchange.com Web site and provide the URL. How would we map a question in the Posts table to the corresponding SO URL? To find the most recent question, we are going to filter the Posts table by the PostTypeId of 1 since that correlates to questions, and then we are going to look for the most recent Creation Date.

```
query = "SELECT Posts.Id, DATETIME(Posts.CreationDate) as Date, Posts.CreationDate, Posts.Body
FROM Posts
WHERE Posts.PostTypeId = 1
ORDER BY Date DESC"
head(dbGetQuery(db, query))
```

```
##       Id                Date            CreationDate
## 1 608405 2023-03-05 05:10:18 2023-03-05T05:10:18.393
## 2 608403 2023-03-05 04:33:18 2023-03-05T04:33:18.820
## 3 608401 2023-03-05 03:21:49 2023-03-05T03:21:49.153
## 4 608400 2023-03-05 02:51:55 2023-03-05T02:51:55.797
## 5 608398 2023-03-05 01:37:48 2023-03-05T01:37:48.233
## 6 608397 2023-03-05 01:23:04 2023-03-05T01:23:04.347
##
## 1
## 2
## 3
## 4
## 5 <h4>Motivating Background Info</h4>\\\\n<p>I was recently in a grad class and someone was presenti
## 6
```

We use the DATETIME function in order to sort the strings by date time in order of most recently added. This will allow us to see which question is most recent by ordering in descending order. The associated Post Id with the most recent question is 608405.

```
query = "SELECT Posts.Id, DATETIME(Posts.CreationDate) as Date, Posts.Body, Posts.OwnerUserId
FROM Posts
WHERE Posts.PostTypeId = 1
ORDER BY Date DESC
LIMIT 1"
dbGetQuery(db, query)
```

```
##       Id                Date
## 1 608405 2023-03-05 05:10:18
##
## 1 <p>Are there any methods that combine VI and MCMC? If it exists, why isn't it used prominently ove
##   OwnerUserId
## 1      382402
```

I looked up the following body of the question online: *Are there any methods that combine VI and MCMC? If it exists, why isn't it used prominently over techniques such as NUTS or other VIs.*

I found this stack overflow link: https://stats.stackexchange.com/questions/608458/are-there-any-methods-that-combine-mcmc-and-vi

We can verify whether or not this is the correct post is by checking the owner user id. We can't rely on the view count since the website was most likely viewed more times, especially considering that this assignment was posted and many other students could have recently been looking at it. By looking at the actual SO website, I clicked on the user who posted that question, and their name is JJbox. Their associated user Id is 382402 which helps us confirm that this is the right question.

```
query = "SELECT Posts.Id, Posts.Tags, Posts.OwnerUserId
FROM Posts
WHERE Posts.Id IN ('608405')"
dbGetQuery(db, query)
```

```
##       Id
## 1 608405
##                                                                      Tags
## 1 <markov-chain-montecarlo><variational-inference><hamiltonian-monte-carlo>
##   OwnerUserId
## 1      382402
```

```
query = "SELECT Users.Id, Users.DisplayName
FROM Users
WHERE Users.Id = '382402'"
dbGetQuery(db, query)
```

```
##       Id DisplayName
## 1 382402       JJbox
```

Now, we will answer the second part of the question: how would we map a question in the Posts table to the corresponding SO URL?

10. For the 10 users who posted the most questions: 1) How many questions did they post?, 2) What are the users' names?, 3) When did they join SO?, 4) What is their Reputation?, 5) What country do they have in their profile? NOT FINISHED YET First we will find the users who posted the most questions. The Users table provides information about each user, including the number of upvotes and downvotes that they have and their reputation, as just 2 examples, but it doesn't include anything about how many questions they have posted.

I will check the posts table to see if it has any information about which user is posting the question, and then create a frequency table from that information.

```
colnames(posts_db)
```

```
##  [1] "Id"                    "PostTypeId"            "AcceptedAnswerId"
##  [4] "CreationDate"          "Score"                 "ViewCount"
##  [7] "Body"                  "OwnerUserId"           "LastActivityDate"
## [10] "Title"                 "Tags"                  "AnswerCount"
## [13] "CommentCount"          "ContentLicense"        "LastEditorDisplayName"
## [16] "LastEditDate"          "LastEditorUserId"      "CommunityOwnedDate"
## [19] "ParentId"              "OwnerDisplayName"      "ClosedDate"
## [22] "FavoriteCount"
```

The Posts table has a column called the OwnerUserId, and this value correlates to the User themselves who initially posted the question. I will query in the Posts table for all posts which are questions, and then group these posts by the OwnerUserId to see how many questions these users posted.

```
query = "SELECT Posts.OwnerUserId, COUNT(Posts.OwnerUserId) AS Freq
FROM Posts
WHERE Posts.PostTypeId = 1
GROUP BY Posts.OwnerUserId
ORDER BY Freq DESC"
ans = dbGetQuery(db, query)
```

```
## Warning: Column `OwnerUserId`: mixed type, first seen values of type string,
## coercing other values of type integer
```

```
query = "SELECT DISTINCT Tag, COUNT(Tag) AS Number_of_Posts
FROM TagPosts
WHERE Tag = 'regression' OR Tag = 'anova' OR Tag = 'data-mining' OR Tag = 'machine-learning' OR Tag = '
GROUP BY Tag
ORDER BY Number_of_Posts DESC"
dbGetQuery(db, query)
```

12. For each of the following terms, how many questions contain that term: Regression, ANOVA, Data Mining, Machine Learning, Deep Learning, Neural Network.

```
##                Tag Number_of_Posts
## 1       regression           28146
## 2 machine-learning           19355
## 3  neural-networks            7793
## 4            anova            5100
## 5    deep-learning            1878
## 6      data-mining            1180
```

The way I am solving it is by looking for the exact keywords as specified in the prompt. I manually looked through the keywords after loading in the TagPosts dataframe to find which specific word represented the terms in the prompt. There are 28,146 questions containing the term Regression, 19,355 questions containing the term Machine Learning, 7,793 questions containing the term Neural Networks, 5,100 questions containing the term ANOVA, 1,878 questions containing the term Deep Learning, and 1,180 questions containing the term Data Mining.

For further investigation, I am going to use R to analyze the tags. I want to look deeper since terms like regression are umbrella terms and there are many different types of regression so there could be many more posts that fall under that category.

```
tagkeywds = unique(tagposts_db$Tag)
length(tagkeywds)
```

```
## [1] 1587
```

There are 1587 unique tag keywords. Many of them may contain parts of the keywords that we are searching for. For example, there are many different types of regression so just using the keyword regression may not be a thorough enough filter. This website helped me filter out the table to contain rows that had a specific part of a string: https://www.tutorialspoint.com/select-where-row-value-contains-string-in-mysql#:~:text=To%20select%20the%20row%20value,a%20table%20is%20as%20follows.

```
print("How many terms containing regression are there?")
```

```
## [1] "How many terms containing regression are there?"
```

```
length(grep("regression", tagkeywds, value = TRUE))
```

```
## [1] 25
```

```
#grep("anova", tagkeywds, value = TRUE)
```

```
query = "SELECT DISTINCT Tag, COUNT(Tag) AS Number_of_Posts
FROM TagPosts
WHERE Tag like '%regression%'
GROUP BY Tag
ORDER BY Number_of_Posts DESC"

num_regression = dbGetQuery(db, query)
nrow(num_regression)
```

```
## [1] 25
```

We have verified that our query correctly gives us the count of posts associated with each of the terms that contain regression. We can now do the same with all of the other keyterms.

```
query = "SELECT DISTINCT Tag, COUNT(Tag) AS Number_of_Posts
FROM TagPosts
WHERE Tag like '%regression%' OR Tag like '%anova%' OR Tag like '%data-mining%' OR Tag like '%machine-l
GROUP BY Tag
ORDER BY Number_of_Posts DESC"

dbGetQuery(db, query)
```

```
##                              Tag Number_of_Posts
## 1                     regression           28146
## 2                machine-learning           19355
## 3                  neural-networks            7793
## 4             multiple-regression            5265
## 5                           anova            5100
## 6          regression-coefficients            1968
## 7                    deep-learning            1878
## 8                     data-mining            1180
## 9            nonlinear-regression            1113
## 10             poisson-regression             904
## 11           vector-autoregression             785
## 12               ridge-regression             743
## 13                         manova             434
## 14            quantile-regression             371
## 15            weighted-regression             353
## 16         multivariate-regression             337
## 17            stepwise-regression             318
## 18          regression-strategies             292
## 19                meta-regression             203
## 20                beta-regression             198
## 21             dynamic-regression             144
## 22               tobit-regression             135
## 23         constrained-regression             133
## 24       regression-discontinuity              73
## 25       nonparametric-regression              66
## 26           segmented-regression              55
## 27 seemingly-unrelated-regressions              48
## 28         regression-to-the-mean              23
## 29              deming-regression              22
## 30        reduced-rank-regression              17
## 31           dirichlet-regression              14
## 32        geometric-deep-learning               2
```

13. Using the Posts and PostLinks tables, how many questions gave rise to a "related" or "duplicate" question? And how many responses did these questions get? How experienced were the users posting these questions. We will look at the structure of the Posts, PostLinks, and LinkTypeMap tables to get an idea of what the data in the table looks like:

```
postlinks_db = dbReadTable(db, 'PostLinks')
linktypemap_db = dbReadTable(db, 'LinkTypeMap')
head(postlinks_db)
```

```
##    Id            CreationDate PostId RelatedPostId LinkTypeId
## 1 108 2010-07-21T14:47:33.983    395           173          1
## 2 145 2010-07-23T16:30:41.780    548           539          1
## 3 151 2010-07-24T09:11:01.413    536           590          1
## 4 217 2010-07-26T20:12:15.600    375            30          1
## 5 263 2010-07-27T16:00:22.133    769            31          1
## 6 264 2010-07-27T16:00:22.133    769             6          1
```

```
head(posts_db)
```

```
##   Id PostTypeId AcceptedAnswerId            CreationDate Score ViewCount
## 1  1          1               15 2010-07-19T19:12:12.510    49      5364
## 2  2          1               59 2010-07-19T19:12:57.157    34     33588
## 3  3          1                5 2010-07-19T19:13:28.577    71      6622
## 4  4          1              135 2010-07-19T19:13:31.617    23     45393
## 5  5          2                0 2010-07-19T19:14:43.050    90         0
## 6  6          1                0 2010-07-19T19:14:44.080   486    172176
##
## 1
## 2
## 3
## 4
## 5
## 6 <p>Last year, I read a blog post from <a href="http://anyall.org/">Brendan O'Connor</a> entitled <a
##   OwnerUserId        LastActivityDate
## 1           8 2020-11-05T09:44:51.710
## 2          24 2022-11-23T13:03:42.033
## 3          18 2022-11-27T23:33:13.540
## 4          23 2010-09-08T03:00:19.690
## 5          23 2010-07-19T19:21:15.063
## 6           5 2021-01-19T17:59:15.653
##                                                                Title
## 1                                  Eliciting priors from experts
## 2                                          What is normality?
## 3 What are some valuable Statistical Analysis open source projects?
## 4       Assessing the significance of differences in distributions
## 5
## 6               The Two Cultures: statistics vs. machine learning?
##                                      Tags AnswerCount CommentCount
## 1            <bayesian><prior><elicitation>           6            1
## 2    <distributions><normality-assumption>           7            1
## 3                  <software><open-source>          19            3
## 4 <distributions><statistical-significance>           5            2
## 5                                    <NA>           0            3
## 6          <machine-learning><pac-learning>          20           10
##   ContentLicense LastEditorDisplayName            LastEditDate LastEditorUserId
## 1   CC BY-SA 2.5
## 2   CC BY-SA 2.5                user88 2010-08-07T17:56:44.800
## 3   CC BY-SA 2.5                       2011-02-12T05:50:03.667              183
## 4   CC BY-SA 2.5
## 5   CC BY-SA 2.5                       2010-07-19T19:21:15.063               23
## 6   CC BY-SA 3.0                       2017-04-08T17:58:18.247            11887
##       CommunityOwnedDate ParentId OwnerDisplayName ClosedDate FavoriteCount
## 1
## 2
## 3 2010-07-19T19:13:28.577
## 4
## 5 2010-07-19T19:14:43.050        3
## 6 2010-08-09T13:05:50.603
```

```
linktypemap_db
```

```
##   id                                              value
## 1  1   Linked (PostId contains a link to RelatedPostId)
## 2  3 Duplicate (PostId is a duplicate of RelatedPostId)
```

There are only two types of links with LinkTypeIds of 1 and 3. Overall Steps: * We are asked to find out how many **questions** gave rise to a duplicate or related question so that means that we first will need to filter the posts by those which have a PostIdType of 1. * We will look at the PostIds in the Posts table which have a PostTypeId of 1, and then we will look at the corresponding PostId in the PostLinks table to see what LinkTypeId it has.

```
query = "SELECT Posts.Id AS PostId, Posts.PostTypeId, PostLinks.LinkTypeId, LinkTypeMap.value
FROM Posts
LEFT JOIN PostLinks
ON Posts.Id = PostLinks.PostId
LEFT JOIN LinkTypeMap
ON PostLinks.LinkTypeId = LinkTypeMap.id
WHERE Posts.PostTypeId = 1"
ans = dbGetQuery(db, query)
head(ans)
```

```
##   PostId PostTypeId LinkTypeId                                          value
## 1      1          1         NA                                           <NA>
## 2      2          1         NA                                           <NA>
## 3      3          1         NA                                           <NA>
## 4      4          1          1 Linked (PostId contains a link to RelatedPostId)
## 5      6          1          1 Linked (PostId contains a link to RelatedPostId)
## 6      6          1          1 Linked (PostId contains a link to RelatedPostId)
```

This table gives us all the posts in the Posts table which are questions, and maps it to its corresponding PostTypeId (which will be 1 since we filtered the table in such a way that we would only get questions), and then we also mapped it to the LinkTypeId (if it exists for that question), and then to the value which the LinkTypeId correlates to.

Now we will want to constrain this table even more to only show the posts which have LinkTypeIds. This means that we will want to use an INNER JOIN since we need the post to have both a postId and a LinkTypeId since then we know that the question gave rise to a related or duplicated question.

```
query = "SELECT Posts.Id AS PostId, PostLinks.RelatedPostId, PostLinks.LinkTypeId, LinkTypeMap.value
FROM Posts
INNER JOIN PostLinks
ON Posts.Id = PostLinks.PostId
LEFT JOIN LinkTypeMap
ON PostLinks.LinkTypeId = LinkTypeMap.id
WHERE Posts.PostTypeId = 1"
ans = dbGetQuery(db, query)
head(ans)
```

```
##   PostId RelatedPostId LinkTypeId
## 1    395           173          1
## 2    548           539          1
```

```
## 3    375            30          1
## 4    769            31          1
## 5    769             6          1
## 6    790           298          1
##                                                value
## 1 Linked (PostId contains a link to RelatedPostId)
## 2 Linked (PostId contains a link to RelatedPostId)
## 3 Linked (PostId contains a link to RelatedPostId)
## 4 Linked (PostId contains a link to RelatedPostId)
## 5 Linked (PostId contains a link to RelatedPostId)
## 6 Linked (PostId contains a link to RelatedPostId)
```

```
query = "SELECT COUNT(*)
FROM Posts
INNER JOIN PostLinks
ON Posts.Id = PostLinks.PostId
LEFT JOIN LinkTypeMap
ON PostLinks.LinkTypeId = LinkTypeMap.id
WHERE Posts.PostTypeId = 1"
dbGetQuery(db, query)
```

```
##   COUNT(*)
## 1    80155
```

There are **80,155** questions which give rise to a related or duplicate question. We know this is true because we filtered the Posts table to only contain the posts which are questions, and those which correlate to a LinkTypeId. Now I will answer the subquestions.

The first subquestion is asking how many responses these questions got. I am going to interpret responses as the number of comments and the number of answers a question gets.

```
query = "SELECT Posts.Id AS PostId, PostLinks.RelatedPostId, PostLinks.LinkTypeId, LinkTypeMap.value, P
FROM Posts
INNER JOIN PostLinks
ON Posts.Id = PostLinks.PostId
LEFT JOIN LinkTypeMap
ON PostLinks.LinkTypeId = LinkTypeMap.id
WHERE Posts.PostTypeId = 1"
ans = dbGetQuery(db, query)
head(ans)
```

```
##   PostId RelatedPostId LinkTypeId
## 1    395           173          1
## 2    548           539          1
## 3    375            30          1
## 4    769            31          1
## 5    769             6          1
## 6    790           298          1
##                                                value AnswerCount CommentCount
## 1 Linked (PostId contains a link to RelatedPostId)           2            1
## 2 Linked (PostId contains a link to RelatedPostId)           1            0
## 3 Linked (PostId contains a link to RelatedPostId)           0            1
## 4 Linked (PostId contains a link to RelatedPostId)           2            4
```

```
## 5 Linked (PostId contains a link to RelatedPostId)           2            4
## 6 Linked (PostId contains a link to RelatedPostId)           2            3
##   Total_Num_Responses
## 1                   3
## 2                   1
## 3                   1
## 4                   6
## 5                   6
## 6                   5
```

The table above answers the question: *how many responses did these questions get?*. We have all the questions which gave rise to related or duplicate questions, the Id of the related or duplicate question, and the number of total responses this question got which is in the Total_Num_Responses column.

Now to answer the second part of the question: How experienced were the users posting these *questions*, we are going to look in the Users table to see the reputation of the User who posted these questions. THe higher the reputation, the higher the experience. I got this idea of using reputation as a metric from Piazza question 232 https://piazza.com/class/lfxbfh6er6b2jo/post/232.

```
query = "SELECT Posts.Id AS PostId, Posts.OwnerUserId, Users.Reputation AS User_Reputation, PostLinks.R
FROM Posts
INNER JOIN PostLinks
ON Posts.Id = PostLinks.PostId
LEFT JOIN LinkTypeMap
ON PostLinks.LinkTypeId = LinkTypeMap.id
LEFT JOIN Users
ON Posts.OwnerUserId = Users.Id
WHERE Posts.PostTypeId = 1
ORDER BY User_Reputation DESC"
ans = dbGetQuery(db, query)
```

```
## Warning: Column `OwnerUserId`: mixed type, first seen values of type integer,
## coercing other values of type string
```

```
head(ans)
```

```
##   PostId OwnerUserId User_Reputation RelatedPostId
## 1  41208         919          304878         23779
## 2   1963         919          304878         99376
## 3 204843         919          304878        200500
## 4 415435         919          304878        484495
## 5 204843         919          304878        243126
## 6 576735         919          304878        265939
##                                              value AnswerCount CommentCount
## 1 Linked (PostId contains a link to RelatedPostId)          34           26
## 2 Linked (PostId contains a link to RelatedPostId)           4            7
## 3 Linked (PostId contains a link to RelatedPostId)           3           10
## 4 Linked (PostId contains a link to RelatedPostId)           1            0
## 5 Linked (PostId contains a link to RelatedPostId)           3           10
## 6 Linked (PostId contains a link to RelatedPostId)           1            0
##   Total_Num_Responses
## 1                  60
## 2                  11
```

```
## 3                  13
## 4                   1
## 5                  13
## 6                   1
```

The table above answers all of the questions. The total number of rows in our data frame tells us how many questions gave rise to a duplicate or related question. Then, we added different columns to our table to tell us how many responses each question was getting by summing up the number of comments and responses on each of these questions that were linked in some way to another question. Lastly, we added another column to show the correlating User who posted the original question, and their reputation. I ordered the table in descending order so PostIds of questions associated with Users who wrote them with the highest experience/reputation are shown. For example, the user with UserId 919 has the highest reputation out of all the questions which are linked to other posts in some way.

14. What is the date range for the questions and answers in this database? The way I am interpreting this problem is to find the date range for posts which are questions only. I am then going to find the date range for posts which are answers only.The date range will be the earliest/minimum date time, and the latest/maximum date time in the column of date time strings. I will also try to find the duration between these date values.

Below, I will be looking into all of the posts which are questions.

```
query = "SELECT Posts.Id AS PostId, Posts.Body FROM Posts Where PostTypeId = 1"
head(dbGetQuery(db, query))
```

```
##   PostId
## 1      1
## 2      2
## 3      3
## 4      4
## 5      6
## 6      7
##
## 1
## 2
## 3
## 4
## 5 <p>Last year, I read a blog post from <a href="http://anyall.org/">Brendan O'Connor</a> entitled <a
## 6
```

```
query = "SELECT MIN(DATETIME(Posts.CreationDate)) AS Min_Question_Date, MAX(DATETIME(Posts.CreationDate)
FROM Posts
WHERE PostTypeId = 1"
head(dbGetQuery(db, query))
```

```
##     Min_Question_Date   Max_Question_Date Duration_Years Duration_Days
## 1 2009-02-02 14:21:12 2023-03-05 05:10:18             14      5143.617
```

Below are the posts which are answers, and this is because the PostTypeId is equal to 2.

```
query = "SELECT Posts.Id AS PostId, Posts.Body FROM Posts Where PostTypeId = 2"
head(dbGetQuery(db, query))
```

```
##   PostId
## 1      5
## 2      9
## 3     12
## 4     13
## 5     14
## 6     15
##
## 1 <p>The R-project</p>\\\\n\\\\n<p><a href="http://www.r-project.org/">http://www.r-project.org/</a>
## 2
## 3
## 4                                                <p>Machine Learning seems to have its basis in the prag
## 5
## 6
```

```
query = "SELECT MIN(DATETIME(Posts.CreationDate)) AS Min_Question_Date, MAX(DATETIME(Posts.CreationDate)
FROM Posts
WHERE PostTypeId = 2"

head(dbGetQuery(db, query))
```

```
##     Min_Question_Date   Max_Question_Date Duration_Years Duration_Days
## 1 2009-02-02 14:24:31 2023-03-05 04:48:34             14        5143.6
```

```
query = "SELECT Id, Body, CommentCount, AnswerCount
FROM Posts
WHERE Posts.PostTypeId = 1
ORDER BY CommentCount DESC"
question_post_comments = dbGetQuery(db, query)
head(question_post_comments)
```

15. What question has the most comments associated with it? How many answers are there for this question?

```
##       Id
## 1 328630
## 2 357466
## 3 298917
## 4 195034
## 5 286415
## 6 349922
##
## 1
=FALSE),   \\\## 2 <h2>TL;DR</h2>\\\\n<p>See title=TRUE)\\\\n<hr />\\\\n<h1>Motivation</h1>\\\\n<p>I&#39;m
##
## 3
tatistics-in-d##a4science/">http://magazine.amstat.org/blog/2015/10/01/asa-statement-on-the-role-of-statistics-in-data
```

```
## 5
e^{-(b+c)t}}{1 + \\frac{c}{b}e^{-(b+c)t}}\\space .$$ This equation has a sigmoidal shape and without mu
## 6
##   CommentCount AnswerCount
## 1           54           6
## 2           53           2
## 3           46          16
## 4           44          13
## 5           39           0
## 6           39           2
```

In order to solve this problem, I filtered the dataframe by posts with a PostTypeId of 1 since those posts are questions. I then order the table by comment count since I want to see which questions have the most comments. Out of the 204,370 questions that we have found, the most comments associated with a question is 54, and there are 6 answers to this question. This question specifically is below and the associated post Id is 328630.

```
query =  "SELECT Body
FROM Posts
WHERE Posts.PostTypeId = 1 AND Posts.Id = 328630"
print(dbGetQuery(db, query)$Body)
```

```
## [1] "<p>Consider a good old regression problem with $p$ predictors and sample size $n$. The usual wi
```

To check if this is true, we are going to look into the Comments table.

```
query = "SELECT DISTINCT PostId, COUNT(PostId) As Num_Comments
FROM Comments
GROUP BY PostId
ORDER BY Num_Comments DESC"
head(dbGetQuery(db, query))
```

```
##   PostId Num_Comments
## 1 386853           66
## 2 551190           62
## 3 451817           54
## 4 328630           54
## 5 357466           53
## 6 471672           48
```

I will now manually check the first few PostIds before the PostId 328630 which I found to make sure that they are not questions. This link helped me with the In keyword: https://www.w3schools.com/sql/sql_in.asp

```
query = "SELECT Posts.Id, Posts.PostTypeId
FROM Posts
WHERE Posts.Id IN ('386853', '551190', '451817', '328630')"
head(dbGetQuery(db, query))
```

```
##       Id PostTypeId
## 1 328630          1
## 2 386853          2
## 3 451817          2
## 4 551190          2
```

We have just verified that we found the correct post/question since the other posts that had more comments are not questions, but are answers.

16. How many comments are there across all posts? How many posts have a comment? What is the distribution of comments per question? In order to solve this problem, we will first examine the structure of the Comments table.

```
comments_db = dbReadTable(db, 'Comments')
```

```
## Warning: Column `UserId`: mixed type, first seen values of type integer,
## coercing other values of type string
```

```
head(comments_db)
```

```
##   Id PostId Score
## 1  1      3     7
## 2  2      5     0
## 3  3      9     1
## 4  4      5    11
## 5  6     14    10
## 6  7     18     1
##
## 1                          Could be a poster child for argumentative and subjective.  At the least,
## 2                                                                                Yes, R is nice
## 3                                                    Again- why?  How would I convince my boss to
## 4 It's mature, well supported, and a standard within certain scientific communities (popular in our
## 5                                    why ask the question here?  All are community-wiki, why not just
## 6                                                    also the US census data http://www.censu
##              CreationDate UserId ContentLicense UserDisplayName
## 1 2010-07-19T19:15:52.517     13   CC BY-SA 2.5
## 2 2010-07-19T19:16:14.980     13   CC BY-SA 2.5
## 3 2010-07-19T19:18:54.617     13   CC BY-SA 2.5
## 4 2010-07-19T19:19:56.657     37   CC BY-SA 2.5
## 5 2010-07-19T19:22:27.947     23   CC BY-SA 2.5
## 6 2010-07-19T19:25:47.877     36   CC BY-SA 2.5
```

```
print(colnames(comments_db))
```

```
## [1] "Id"              "PostId"          "Score"           "Text"
## [5] "CreationDate"    "UserId"          "ContentLicense"  "UserDisplayName"
```

The PostId column in the Comments table represents the post for which the comment is associated with. If there are multiple comments associated with a post, the PostId will be the same for all of those comments. Therefore, to find the total number of columns, we can count the number of rows there are overall in the Comments table. This is because every comment will be associated with a post so we will count the number of these comments.

```
query = "SELECT COUNT(*) FROM Comments"
dbGetQuery(db, query)
```

```
##   COUNT(*)
## 1   768069
```

There are **768,069** total comments across all posts. We can check if this is right in another way as well by summing over the CommentCount attribute in the Posts table.

```
query = "SELECT SUM(CommentCount) FROM Posts"
dbGetQuery(db, query)
```

```
##   SUM(CommentCount)
## 1            768069
```

The second method gives us the same number of total comments, so we have verified that we found the right value for the total number of comments across all posts.

Now we will answer the sub questions – the first being how many posts have a comment?

```
query = "
SELECT Posts.CommentCount, COUNT(Posts.CommentCount) AS Frequency
FROM Posts
GROUP BY CommentCount
"
comment_freq = dbGetQuery(db, query)
head(comment_freq)
```

```
##   CommentCount Frequency
## 1            0    175361
## 2            1     60722
## 3            2     56286
## 4            3     34971
## 5            4     24968
## 6            5     16241
```

The frequency table above tells us how many posts have 0 comments, 1 comments, 2 comments, etc. The first row in our frequency table tells us that there are 175,361 posts with 0 comments.

To confirm that our comment count frequency table for all posts is correct, we need to check whether or not the sum of the Frequency table is equal to the total number of posts there are. If we did it correctly, the sum of the Frequency column should be equal to 405,220 which is the total number of posts there are.

```
print("Total number of posts there are:")
```

```
## [1] "Total number of posts there are:"
```

```
query = "SELECT COUNT(*) FROM Posts"
dbGetQuery(db, query)
```

```
##   COUNT(*)
## 1   405220
```

```
query = "SELECT SUM(Frequency) FROM
(
SELECT Posts.CommentCount, COUNT(Posts.CommentCount) AS Frequency
FROM Posts
GROUP BY CommentCount
)"
dbGetQuery(db, query)
```

```
##   SUM(Frequency)
## 1         405220
```

We have now verified that our comment count frequency table for all posts is correct since the sum of the number of posts there are for each comment count sums up to the total number of posts which is 405,220.

Now we will limit the resulting output of our sequel command to just sum of frequencies for posts which don't have a comment count of 0. Posts which have a comment count of 0 don't have any comments so we don't want this in our total sum.

```
query = "SELECT SUM(Frequency) FROM
(
SELECT Posts.CommentCount, COUNT(Posts.CommentCount) AS Frequency
FROM Posts
WHERE Posts.CommentCount != '0'
GROUP BY CommentCount
)"
dbGetQuery(db, query)
```

```
##   SUM(Frequency)
## 1         229859
```

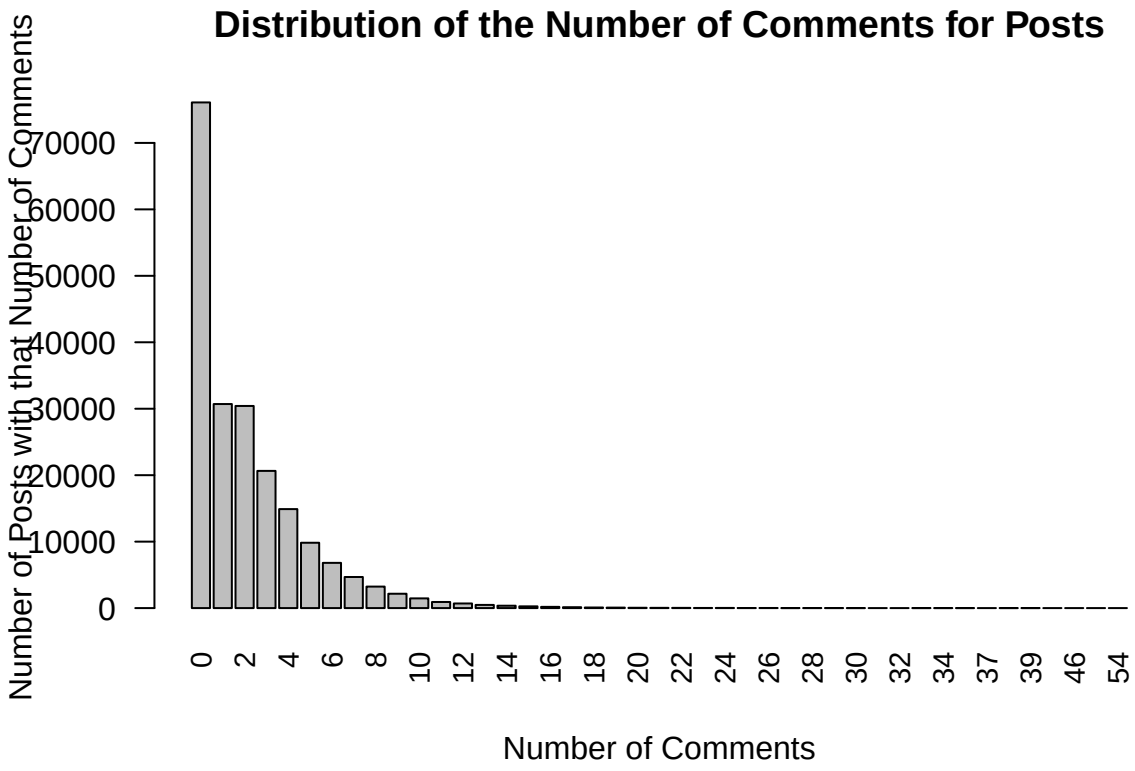
The first sub-question we answered above is: how many posts have a comment? I got that **229,859** posts have a comment. The way that I interpreted this question is to find the number of posts that don't have 0 comments. This means that I will be finding the number of posts that have 1, 2, 3, … etc. comments.

Now we will answer the next question asking us what the distribution of comments is per question. In order to do this, I will first find all of the posts which are questions by limiting the posts to those that have a post type Id of 1, and then count the frequency of each of the comment counts for the questions.

```
query = "
SELECT Posts.CommentCount, COUNT(Posts.CommentCount) AS Frequency
FROM Posts
WHERE Posts.PostTypeId = 1
GROUP BY CommentCount
"
freq = dbGetQuery(db, query)
head(freq)
```

```
##   CommentCount Frequency
## 1            0     76090
## 2            1     30707
## 3            2     30417
## 4            3     20645
## 5            4     14897
## 6            5      9832
```

```
barplot(freq$Frequency, xlab = "Number of Comments", ylab = "Number of Posts with that Number of Commen
```



In order to check if this is the correct frequency table, we need to make sure that the sum of the frequency column is equal to the total number of posts which are questions. There are 204,370 total questions.

```
print("The total number of questions is equal to: ")
```

```
## [1] "The total number of questions is equal to: "
```

```
query = "SELECT COUNT(Posts.PostTypeId) FROM Posts WHERE PostTypeId = 1"
dbGetQuery(db, query)
```

```
##   COUNT(Posts.PostTypeId)
## 1                  204370
```

```
print("The sum of the frequency column in our frequency table from above is: ")
```

```
## [1] "The sum of the frequency column in our frequency table from above is: "
```

```
query = "SELECT SUM(Frequency) FROM
(
SELECT Posts.CommentCount, COUNT(Posts.CommentCount) AS Frequency
FROM Posts
WHERE Posts.PostTypeId = 1
GROUP BY CommentCount
)"
dbGetQuery(db, query)
```

```
##   SUM(Frequency)
## 1         204370
```

18. Do the people who vote tend to have badges? NOT FINISHED YET For this problem, we will examine the votes, votetypemap, and badges tables. The votes table has the following columns:

```
votes_db = dbReadTable(db, "Votes")
```

```
## Warning: Column `UserId`: mixed type, first seen values of type string, coercing
## other values of type integer
```

```
## Warning: Column `BountyAmount`: mixed type, first seen values of type string,
## coercing other values of type integer
```

```
colnames(votes_db)
```

```
## [1] "Id"           "PostId"       "VoteTypeId"   "CreationDate" "UserId"
## [6] "BountyAmount"
```

We will look at the UserId attribute in the table to find out which user voted on the post which is specified with the PostId column. Looking at the UserId column, a lot of the values are empty which makes sense since there are lots of different voting types which wouldn't need to record a user. Looking at the votetypemap, we can see what types of votes will record the UserId.

```
votetypemap_db = dbReadTable(db, "VoteTypeMap")
head(votetypemap_db)
```

```
##   id
## 1  1
## 2  2
## 3  3
## 4  4
## 5  5
## 6  6
##                                                                    value
## 1                                                  AcceptedByOriginator
## 2                                                    UpMod (AKA upvote)
## 3                                                 DownMod (AKA downvote)
## 4                                                             Offensive
## 5                                  Favorite (UserId will also be populated)
## 6 Close (effective 2013-06-25: Close votes are only stored in table: PostHistory)
```

Votes with an associated vote type ID of 5 and 8 will record the userId, and this is said in the value column in parentheses of the VoteTypeMap table. We can also verify this by looking for the entries in the UserId column of the Votes table which are **not** empty since the majority of UserId column values are empty. I am using R to visualize and analyze the data and then I will later query the data with sequel commands after I understand more about the structure of the data.

```
non_empty_userId = votes_db[which(votes_db$UserId != ""),]
head(non_empty_userId)
```

```
##        Id PostId VoteTypeId            CreationDate UserId BountyAmount
## 1405 1733     93          8 2010-07-24T00:00:00.000     61           50
## 1805 2238    575          8 2010-07-27T00:00:00.000    196           50
## 3049 3819    608          8 2010-08-03T00:00:00.000    196           50
## 5055 6209   1841          8 2010-08-20T00:00:00.000    990           50
## 6426 7849   2272          8 2010-09-03T00:00:00.000     71           50
## 6437 7862   2245          8 2010-09-03T00:00:00.000      5           50
```

```
print("Vote Type IDs associated with User Idss")
```

```
## [1] "Vote Type IDs associated with User Idss"
```

```
print(unique(non_empty_userId$VoteTypeId))
```

```
## [1] 8 5
```

Our analysis has proven our hypothesis that VoteTypeIds of 5 and 8, correlating to Favorites and Bounty Starts respectively, are recorded. Now we can move forward with the data analyzing.

```
query = "SELECT Votes.PostId, Votes.UserId, Votes.VoteTypeId
FROM Votes
WHERE Votes.UserId != ''"
numVotingUsers = dbGetQuery(db, query)
print("The number of users who are voting is: ")
```

```
## [1] "The number of users who are voting is: "
```

```
nrow(numVotingUsers)
```

```
## [1] 6552
```

```
head(numVotingUsers)
```

```
##   PostId UserId VoteTypeId
## 1     93     61          8
## 2    575    196          8
## 3    608    196          8
## 4   1841    990          8
## 5   2272     71          8
## 6   2245      5          8
```

This means that there are 6,552 votes affiliated with a UserId, so those users have voted. Now we will link these UserIds that we have found to the data in the Badges table.

```
badges_db = dbReadTable(db, 'Badges')
head(badges_db)
```

```
##   Id UserId    Name                    Date Class TagBased
## 1  1      5 Teacher 2010-07-19T19:39:07.047     3    False
## 2  2      6 Teacher 2010-07-19T19:39:07.220     3    False
## 3  3      8 Teacher 2010-07-19T19:39:07.330     3    False
## 4  4     23 Teacher 2010-07-19T19:39:07.437     3    False
## 5  5     36 Teacher 2010-07-19T19:39:07.593     3    False
## 6  6     37 Teacher 2010-07-19T19:39:07.687     3    False
```

```
print(colnames(badges_db))
```

```
## [1] "Id"       "UserId"   "Name"     "Date"     "Class"    "TagBased"
```

The Badges table has a UserId column which tells us which badge a user has. We need to keep in mind that the Votes UserId column can have repeats since that just means that the same user has voted multiple times.

```
query = "SELECT DISTINCT Votes.UserID, Votes.VoteTypeId, Badges.Class, bcm.value
FROM Votes
LEFT JOIN Badges
ON Votes.UserID = Badges.UserID
JOIN BadgeClassMap AS bcm
ON bcm.id = Badges.Class"
ans = dbGetQuery(db, query)
head(ans)
```

```
##   UserId VoteTypeId Class  value
## 1     61          8     1   Gold
## 2     61          8     2 Silver
## 3     61          8     3 Bronze
## 4    196          8     1   Gold
## 5    196          8     2 Silver
## 6    196          8     3 Bronze
```

Required Questions

21. Compute the table that contains: the question, the name of the user who posted it, when that user joined, their location, the date the question was first posted, the accepted answer, when the accepted answer was posted, the name of the user who provided the accepted answer. We will need the following information to create this table: * The question is the body from the Posts table where these are for the Posts of PostTypeId = 1 (just in case, we can include the question title which may or may not be blank) and this is Posts.Title * Posts.OwnerUserId correlates to the Users.Id which is connected to their display name which is in Users.DisplayName * Users.CreationDate is when they first joined * Users.Location is where their location is * Posts.CreationDate is the date the question was first posted * Posts.AcceptedAnswerId is the id of the accepted answer and we will need to connect this to the actual answer itself. This will be in Posts.Id and it is the body so Posts.Body such that Posts.AcceptedAnswerId = Posts.Id. * When the actual answer was posted is Posts.CreationDate for the post that correlates to that Answer Id. * The name of the user who provided the accepted answer will be the Posts.OwnerUserId for that answer

I am going to do this problem step by step in small chunks and then build up on it. In my first table chunk, there should be 204,370 rows since that is how many questions there are.

```
query = "SELECT Posts.Id AS PostId, Posts.Body AS Question, Posts.OwnerUserId, Users.DisplayName As Ques
FROM Posts
LEFT JOIN Users
ON Users.Id = Posts.OwnerUserId
WHERE Posts.PostTypeId = 1"
head(dbGetQuery(db, query))
```

```
## Warning: Column `OwnerUserId`: mixed type, first seen values of type integer,
## coercing other values of type string
```

```
## Warning: Column `AcceptedAnswerId`: mixed type, first seen values of type
## integer, coercing other values of type string
```

```
##   PostId
## 1      1
## 2      2
## 3      3
## 4      4
## 5      6
## 6      7
##
## 1
## 2
## 3
## 4
## 5 <p>Last year, I read a blog post from <a href="http://anyall.org/">Brendan O'Connor</a> entitled <a
## 6
##   OwnerUserId Question_User_Name When_Question_User_Joined
## 1           8        csgillespie   2010-07-19T19:04:52.280
## 2          24             A Lion   2010-07-19T19:09:32.157
## 3          18             grokus   2010-07-19T19:08:29.070
## 4          23        Jay Stevens   2010-07-19T19:09:16.917
## 5           5              Shane   2010-07-19T19:03:57.227
## 6          38             EAMann   2010-07-19T19:11:57.393
##        Question_User_Location    When_Question_Posted AcceptedAnswerId
## 1   Newcastle, United Kingdom 2010-07-19T19:12:12.510               15
## 2                             2010-07-19T19:12:57.157               59
## 3               United States 2010-07-19T19:13:28.577                5
## 4       Jacksonville, FL, USA 2010-07-19T19:13:31.617              135
## 5                New York, NY 2010-07-19T19:14:44.080                0
## 6 Tualatin, OR, United States 2010-07-19T19:15:59.303               18
```

It is possible for some questions to not have answers, and those questions will have an empty AcceptedAnswerId.

```
query = "SELECT Questions.Id AS Question_Id, Question_User.DisplayName AS User_DisplayName, Question_Use
FROM Posts AS Questions
LEFT JOIN Posts AS Answers
ON Questions.PostTypeId = 1 AND Answers.Id = Questions.AcceptedAnswerId
LEFT JOIN Users as Question_User
ON Questions.OwnerUserId = Question_User.Id
LEFT JOIN Users as Answer_Users
```

```
ON Answer_Users.Id = Answers.OwnerUserId
WHERE Questions.PostTypeId = 1"
table = dbGetQuery(db, query)
```

```
head(table)
```

```
##   Question_Id User_DisplayName  Question_User_JoinDate
## 1           1      csgillespie 2010-07-19T19:04:52.280
## 2           2           A Lion 2010-07-19T19:09:32.157
## 3           3           grokus 2010-07-19T19:08:29.070
## 4           4      Jay Stevens 2010-07-19T19:09:16.917
## 5           6            Shane 2010-07-19T19:03:57.227
## 6           7           EAMann 2010-07-19T19:11:57.393
##        Question_User_Location      Question_Post_Date
## 1   Newcastle, United Kingdom 2010-07-19T19:12:12.510
## 2                             2010-07-19T19:12:57.157
## 3               United States 2010-07-19T19:13:28.577
## 4       Jacksonville, FL, USA 2010-07-19T19:13:31.617
## 5                New York, NY 2010-07-19T19:14:44.080
## 6 Tualatin, OR, United States 2010-07-19T19:15:59.303
##
## 1
## 2
## 3
## 4
## 5 <p>Last year, I read a blog post from <a href="http://anyall.org/">Brendan O'Connor</a> entitled <a
## 6
##   Answer_DisplayName Answer_Id
## 1             Harlan        15
## 2     John L. Taylor        59
## 3        Jay Stevens         5
## 4     John L. Taylor       135
## 5               <NA>        NA
## 6     Stephen Turner        18
##
## 1
## 2 <p>The assumption of normality is just the supposition that the underlying <a href="http://en.wikip
## 3
## 4
Smirnov test</a>, or the like.  The two-sample Kolmogorov-Smirnov test is based on comparing differences
## 5
## 6
##               Answer_Date
## 1 2010-07-19T19:19:46.160
## 2 2010-07-19T19:43:20.423
## 3 2010-07-19T19:14:43.050
## 4 2010-07-19T21:36:12.850
## 5                    <NA>
## 6 2010-07-19T19:24:18.580
```

This table gives me the associated values for every single question in our Posts table. It gives me the associated answer (if it exists), but also includes the row for it even if there is no accepted associated answer. If this is the case, the row contains NA values. It is promising that table contains 204,370 rows since we know

that we have 204,370 questions. Now I am going to filter the table to only keep rows where the accepted answer is not empty.

I want to know how many of these rows in the table above have an empty NA value for the Accepted Answer Id. In my next sequel command, I will be expecting this same number of rows.

```
length(which(!is.na(table$Answer_Id)))
```

```
## [1] 68004
```

I will make this change by doing a self join (inner join) between the post tables instead of left joining.

```
query = "SELECT Questions.Id AS Question_Id, Question_User.DisplayName AS User_DisplayName, Question_Us
FROM Posts AS Questions
JOIN Posts AS Answers
ON Questions.PostTypeId = 1 AND Answers.Id = Questions.AcceptedAnswerId
LEFT JOIN Users as Question_User
ON Questions.OwnerUserId = Question_User.Id
LEFT JOIN Users as Answer_Users
ON Answer_Users.Id = Answers.OwnerUserId
WHERE Questions.PostTypeId = 1"
table_updated = dbGetQuery(db, query)
```

```
head(table_updated)
```

```
##   Question_Id User_DisplayName  Question_User_JoinDate
## 1           1      csgillespie 2010-07-19T19:04:52.280
## 2           2           A Lion 2010-07-19T19:09:32.157
## 3           3           grokus 2010-07-19T19:08:29.070
## 4           4      Jay Stevens 2010-07-19T19:09:16.917
## 5           7           EAMann 2010-07-19T19:11:57.393
## 6          10           A Lion 2010-07-19T19:09:32.157
##        Question_User_Location      Question_Post_Date
## 1   Newcastle, United Kingdom 2010-07-19T19:12:12.510
## 2                              2010-07-19T19:12:57.157
## 3               United States 2010-07-19T19:13:28.577
## 4       Jacksonville, FL, USA 2010-07-19T19:13:31.617
## 5 Tualatin, OR, United States 2010-07-19T19:15:59.303
## 6                              2010-07-19T19:17:47.537
##
## 1
## 2
## 3
## 4
## 5 <p>I've been working on a new method for analyzing and parsing datasets to identify and isolate sub
## 6
##   Answer_DisplayName Answer_Id
## 1             Harlan        15
## 2     John L. Taylor        59
## 3        Jay Stevens         5
## 4     John L. Taylor       135
## 5     Stephen Turner        18
## 6                chl      1887
```

```
##
## 1
## 2
## 3
## 4
Smirnov test</a>, or the like.  The two-sample Kolmogorov-Smirnov test is based on comparing differences
## 5
## 6 <p>Maybe too late but I add my answer anyway...</p>\\\\n\\\\n<p>It depends on what you intend to d
##               Answer_Date
## 1 2010-07-19T19:19:46.160
## 2 2010-07-19T19:43:20.423
## 3 2010-07-19T19:14:43.050
## 4 2010-07-19T21:36:12.850
## 5 2010-07-19T19:24:18.580
## 6 2010-08-19T10:00:00.370
```

```
length(which(is.na(table_updated$Answer_Id)))
```

```
## [1] 0
```

Now we have verified that every single question in our new table has an associated accepted answer since there are no null, empty, or 0-valued answer Ids.

Now I will verify in R if we have the same number of rows as there are valid accepted answer Ids. From the last table we found above, we discovered that there are 68,004 valid Accepted Answer Ids so the total number of rows in my current table needs to be equal to that.

```
nrow(table_updated)
```

```
## [1] 68004
```

This means that there are 68,004 questions with accepted answers.

I am going to check with R to see how many questions have an accepted answer. I will do this by indexing into the data frame that I get to see how many of the accepted answer Ids are equal to 0 since when we read in the table from SQL, the empty values in the Accepted Answer Id column are coerced to 0.

```
posts_db = dbReadTable(db, 'Posts')
```

```
## Warning: Column `AcceptedAnswerId`: mixed type, first seen values of type
## integer, coercing other values of type string

## Warning: Column `ViewCount`: mixed type, first seen values of type integer,
## coercing other values of type string

## Warning: Column `OwnerUserId`: mixed type, first seen values of type integer,
## coercing other values of type string

## Warning: Column `AnswerCount`: mixed type, first seen values of type integer,
## coercing other values of type string
```

```
## Warning: Column `LastEditorUserId`: mixed type, first seen values of type
## string, coercing other values of type integer

## Warning: Column `ParentId`: mixed type, first seen values of type string,
## coercing other values of type integer

## Warning: Column `FavoriteCount`: mixed type, first seen values of type string,
## coercing other values of type integer
```

```
acc_answer_id = posts_db[which(posts_db$AcceptedAnswerId != 0),] #these are all of the accepted answer
nrow(acc_answer_id)
```

```
## [1] 68005
```

In R, it tells us that there are 68,005 posts with accepted answer Ids. In sequel, we are one off. We need to verify that our sequel count of 68,004 is correct. Why is R giving us one more value? In order to find this one value that doesn't match, I am going to do a set difference with the R answer and my sequel answer.

```
my_answer_accepted_answer_ids = table_updated$Answer_Id
R_answer = acc_answer_id$AcceptedAnswerId
setdiff(R_answer, my_answer_accepted_answer_ids)
```

```
## [1] 8713
```

8713 is the value that is not included in our sequel table. This is the value of the accepted answer Id. We will now look into why this is missing. This is or should be associated with the post that is the accepted answer.

```
which(posts_db$Id == 8713)
```

```
## integer(0)
```

For some reason, this Post Id doesn't exist in our table since there is no index for which this accepted answer Id is equal to a Post Id. In our sequel commands, we ensure that every accepted answer Id is equal to a question post Id. Therefore, our sequel table doesn't include this one value and that is the reason for the number of rows being one off than our R answer. Therefore, we have concluded that the correct number of accepted answers is 68,004 instead of 68,005 and our R verification helps us prove why we are right.

22. Determine the users that have only posted questions and never answered a question? (Compute the table containing the number of questions, number of answers and the user's login name for this group.) How many are there? In order to get a better idea of how many users there should be that satisfy this condition, I am going to first work in R. I want the collection of users who have posted a question who haven't posted an answer. This means that we will restrict our Posts table output to posts which are associated with a PostTypeId of 1 and a PostTypeId of 2 to get questions and answers respectively.

```
posts_db = dbReadTable(db, 'Posts')
```

Below I have found the number of users who have posted an answer. This does not count any of the users more than once since we have found the unique number so there are 27,002 unique users who have posted an answer.

```
num_users_who_posted_answer = length(unique(posts_db[which(posts_db$PostTypeId == 2),]$OwnerUserId))
num_users_who_posted_answer
```

```
## [1] 27002
```

Below we have found the number of unique users who have ever posted a question. There are 88,924 unique users who have posted a question.

```
num_users_who_posted_question = length(unique(posts_db[which(posts_db$PostTypeId == 1),]$OwnerUserId))
num_users_who_posted_question
```

```
## [1] 88924
```

We will do the set difference to find all of the users who have ever posted at least 1 question who have NOT ever posted an answer. Using set difference will remove all of the values which are in the collection of users who have posted an answer and in the collection of users who have posted a question which is what we want since we only want to keep users who have only posted a question and not an answer.

```
final_user_list = setdiff(unique(posts_db[which(posts_db$PostTypeId == 1),]$OwnerUserId), unique(posts_
length(final_user_list)
```

```
## [1] 76410
```

Therefore, as shown above, we will expect our data frame for this problem to contain 76410 rows.

Now I will find this table in sequel and I will be expecting 76,410 rows. I searched up how to do set difference in SQL so I could follow the same logic that I did when finding the solution in R. I found that EXCEPT is the SQLite equivalent: https://www.techonthenet.com/sqlite/except.php.

```
query = "SELECT DISTINCT Questions.OwnerUserId
FROM Posts AS Questions
WHERE Questions.PostTypeId = 1
EXCEPT
SELECT DISTINCT Answers.OwnerUserId
FROM Posts AS Answers
WHERE Answers.PostTypeId = 2"
users_who_question_but_dont_answer = dbGetQuery(db, query)
head(users_who_question_but_dont_answer)
```

```
##   OwnerUserId
## 1          18
## 2          24
## 3          38
## 4          52
## 5          53
## 6          58
```

In order to confirm that we found the correct list of users who have posted at least one question but never an answer, I am going to make sure that this list intersects fully with the solution I got in R. If I am correct, the length of the intersection needs to be equal to the original length of the list which is 76,410. Below, I have verified that the list of users is correct.

```
length(intersect(final_user_list, users_who_question_but_dont_answer$OwnerUserId))
```

```
## [1] 76410
```

Therefore, I can confidently conclude that there are **76410** users that have only posted questions and never answered a question.

Now I will use this to create the entire table containing the number of questions, number of answers and the user's login name for this group.

```
query = "SELECT User_ID, Users.DisplayName, COUNT(Posts.OwnerUserId) AS Num_Questions FROM (SELECT DIST
FROM Posts AS Questions
WHERE Questions.PostTypeId = 1
EXCEPT
SELECT DISTINCT Answers.OwnerUserId
FROM Posts AS Answers
WHERE Answers.PostTypeId = 2)
LEFT JOIN Users ON User_ID = Users.Id
LEFT JOIN POSTS
WHERE Posts.PostTypeId = 1 AND User_ID = Posts.OwnerUserId
GROUP BY User_ID"
fin_tab = dbGetQuery(db, query)
head(fin_tab)
```

```
##   User_ID DisplayName Num_Questions
## 1      18      grokus             2
## 2      24      A Lion             2
## 3      38      EAMann             1
## 4      52     Alan H.            13
## 5      53        kyle             2
## 6      58      Preets             1
```

Below is my testing to see how to make the number of questions per unique user ID work.

```
##   UID Freq
## 1  18    2
## 2  24    2
## 3  38    1
## 4  52   13
## 5  53    2
## 6  58    1
```

Now I am going to add to the table the number of answers each of these users has posted. In R, below I will verify that all of these users have posted 0 answers so I will add this to my dataframe.

```
query = "SELECT User_ID, Users.DisplayName, COUNT(Posts.OwnerUserId) AS Num_Questions, IIF(TRUE, 0, 0) A
FROM Posts AS Questions
WHERE Questions.PostTypeId = 1
EXCEPT
SELECT DISTINCT Answers.OwnerUserId
FROM Posts AS Answers
```

```
WHERE Answers.PostTypeId = 2)
LEFT JOIN Users ON User_ID = Users.Id
LEFT JOIN POSTS
WHERE Posts.PostTypeId = 1 AND User_ID = Posts.OwnerUserId
GROUP BY User_ID"
head(dbGetQuery(db, query))
```

```
##   User_ID DisplayName Num_Questions Num_Answers
## 1      18      grokus             2           0
## 2      24      A Lion             2           0
## 3      38      EAMann             1           0
## 4      52     Alan H.            13           0
## 5      53        kyle             2           0
## 6      58      Preets             1           0
```

I will do this check in R to make sure all of these users that we have found have not posted any answers.

```
answer_userids = posts_db[which(posts_db$PostTypeId == 2),]$OwnerUserId #user ids of those who answered
unique_answer_userids = unique(answer_userids)
length(unique_answer_userids)
```

```
## [1] 27002
```

There are 27,002 unique users who have answered posts.

```
table(fin_tab$User_ID %in% unique_answer_userids)
```

```
##
## FALSE
## 76410
```

This table output shows us that none of the unique user Ids of users who post questions and have posted no answers (the column that we got from our data table above) are in the column of unique users who post answers. Therefore, this means that all of our user ids have answered 0 questions since their id is never considered as an answer id.

23. Compute the table with information for the 75 users with the most accepted answers. This table should include: the user's display name, the creation date, the location, the number of badges they have won (the names of the badges as a single string), the dates of the earliest and most recent accepted answer (as two fields), the (unique) tags for all the questions for which they had the accepted answer (as a single string) In this problem, we want to find information for the number of users who posted the most accepted answers. In order to do this, we will first have to find the total number of accepted answer Ids which will be all of the values which have nonzero and non-null values for answerIds, and then map that back to the Post Id to find the OwnerUserId of that answer post. Then we need to find the OwnerUserId of this Answer Post Id where this is the Id of an accepted answer post and then find the frequency table of how many times each of those Owner users have posted an accepted answer Id.

This table gives us all of the accepted answers (since we are searching for all of the posts associated with accepted answers Ids which are not equal to 0, and this will happen explicitly since accepted answer Ids from questions will only associate with nonzero post answer Ids). The number of rows currently checks out since we have found in a previous problem that there are 68,004 accepted answers.

```
query = "SELECT Questions.AcceptedAnswerId, Answers.Id AS Accepted_Answer_Post, Answers.OwnerUserId AS A
FROM Posts AS Questions
JOIN Posts AS Answers
ON Questions.PostTypeId = 1 AND Answers.Id = Questions.AcceptedAnswerId
WHERE Questions.PostTypeId = 1"
a = dbGetQuery(db, query)
```

```
## Warning: Column `Accepted_Answer_UserID_Poster`: mixed type, first seen values
## of type integer, coercing other values of type string
```

```
head(a)
```

```
##   AcceptedAnswerId Accepted_Answer_Post Accepted_Answer_UserID_Poster
## 1               15                   15                             6
## 2               59                   59                            39
## 3                5                    5                            23
## 4              135                  135                            39
## 5               18                   18                            36
## 6             1887                 1887                           930
```

```
print(nrow(a))
```

```
## [1] 68004
```

The total number of rows in our table above is 68,004 which we know is the number of accepted answer posts there are. Now we want to find how many times each user has posted an accepted answer post. We need to make sure that we exclude the Accepted AnswerIds which are 0 since those correlate to questions which don't have an accepted answer and we don't want to count those.

```
query = "SELECT DISTINCT Accepted_Answer_UserID_Poster, COUNT(Accepted_Answer_UserID_Poster) FROM (SELE
FROM Posts AS Questions
JOIN Posts AS Answers
ON Questions.PostTypeId = 1 AND Answers.Id = Questions.AcceptedAnswerId
WHERE Questions.PostTypeId = 1)
WHERE Accepted_Answer_UserID_Poster != ''
GROUP BY Accepted_Answer_UserID_Poster
ORDER BY COUNT(Accepted_Answer_UserID_Poster) DESC LIMIT 75"
b = dbGetQuery(db, query)
head(b, 20)
```

```
##    Accepted_Answer_UserID_Poster COUNT(Accepted_Answer_UserID_Poster)
## 1                            805                                 2335
## 2                            919                                 1781
## 3                          28500                                 1246
## 4                         204068                                 1119
## 5                          35989                                 1004
## 6                           1352                                  985
## 7                           7224                                  984
## 8                          53690                                  872
## 9                         173082                                  826
## 10                         85665                                  814
```

```
## 11                            686                            718
## 12                           7290                            670
## 13                          22311                            611
## 14                          11887                            592
## 15                           7486                            590
## 16                         164061                            475
## 17                           8013                            452
## 18                         247274                            439
## 19                         116195                            424
## 20                          28746                            422
```

The frequency table above tells us how many times the user posted an accepted answer out of all the 68,004 accepted answer posts. I limit it to the top 75 users who posted the most accepted answers. Now I will get the additional information for the users including the user's display name, the creation date, and the location.

```
query = "SELECT DISTINCT Accepted_Answer_UserID_Poster, Users.DisplayName, Users.CreationDate, Users.Lo
FROM Posts AS Questions
JOIN Posts AS Answers
ON Questions.PostTypeId = 1 AND Answers.Id = Questions.AcceptedAnswerId
WHERE Questions.PostTypeId = 1)
LEFT JOIN Users
ON Accepted_Answer_UserID_Poster = Users.Id
WHERE Accepted_Answer_UserID_Poster != ''
GROUP BY Accepted_Answer_UserID_Poster
ORDER BY Num_Ans DESC
LIMIT 75"
c = dbGetQuery(db, query)
head(c, 20)
```

```
##    Accepted_Answer_UserID_Poster             DisplayName
## 1                            805                   Glen_b
## 2                            919                   whuber
## 3                          28500                      EdM
## 4                         204068                    gunes
## 5                          35989                      Tim
## 6                           1352          Stephan Kolassa
## 7                           7224                    Xi'an
## 8                          53690            Richard Hardy
## 9                         173082                      Ben
## 10                         85665                  BruceET
## 11                           686                Peter Flom
## 12                          7290 gung - Reinstate Monica
## 13                         22311                  Sycorax
## 14                         11887       kjetil b halvorsen
## 15                          7486              Robert Long
## 16                        164061         Sextus Empiricus
## 17                          8013                    AdamO
## 18                        247274                     Dave
## 19                        116195                     Noah
## 20                         28746      Alecos Papadopoulos
##               CreationDate                      Location Num_Ans
## 1  2010-08-07T08:40:07.287                I'm right here    2335
```

```
## 2  2010-08-13T15:29:47.140                                1781
## 3  2013-07-26T15:11:03.380                                1246
## 4  2018-04-12T10:42:43.307                Cambridge, UK   1119
## 5  2013-12-10T21:19:06.223               Warsaw, Poland   1004
## 6  2010-09-18T10:55:08.240                  Switzerland    985
## 7  2011-11-05T07:56:15.903                Paris, France    984
## 8  2014-08-08T10:57:13.613                       Europe    872
## 9  2017-08-10T03:27:26.793          Canberra, Australia    826
## 10 2015-08-11T17:22:01.590       San Francisco Bay Area    814
## 11 2010-08-03T19:42:40.907                 New York, NY    718
## 12 2011-11-09T04:43:15.613              Kingdom of Zhao    670
## 13 2013-03-20T23:59:56.610 Washington, DC, United States    611
## 14 2012-06-09T22:52:37.473                      Bolivia    592
## 15 2011-11-20T14:30:04.120        Leeds, United Kingdom    590
## 16 2017-06-05T10:39:01.763            Sion, Switzerland    475
## 17 2011-12-14T21:46:36.197                  Nakoja Abad    452
## 18 2019-05-08T13:23:32.777                                 439
## 19 2016-05-19T15:46:14.703  Cambridge, MA, United States    424
## 20 2013-08-02T14:24:21.923                                 422
```

Next I will start with the task of finding the number of badges these users have won. The tables that are associated with Badges are BadgeClassMap and Badges. To find the number of badges that each user has won, we will find the frequency of each of the user ids in the Badges table in the UserId column. In order to join together all badge names in a string, I found this link with a similar idea that I am going to use: https://www.sqlshack.com/string_agg-function-in-sql/.

```
head(badges_db)
```

```
##   Id UserId    Name                    Date Class TagBased
## 1  1      5 Teacher 2010-07-19T19:39:07.047     3    False
## 2  2      6 Teacher 2010-07-19T19:39:07.220     3    False
## 3  3      8 Teacher 2010-07-19T19:39:07.330     3    False
## 4  4     23 Teacher 2010-07-19T19:39:07.437     3    False
## 5  5     36 Teacher 2010-07-19T19:39:07.593     3    False
## 6  6     37 Teacher 2010-07-19T19:39:07.687     3    False
```

```
badgeclassmap_db = dbReadTable(db, 'BadgeClassMap')
head(badgeclassmap_db)
```

```
##   id  value
## 1  1   Gold
## 2  2 Silver
## 3  3 Bronze
```

I am not going to filter the table by the TagBased attribute, and instead I will include all entries with both True and False TagBased values. The way that I am interpreting this task for counting the number of badges a user has won will be by including all badges for all badge class types of 1, 2, and 3 which correlate to Gold, Silver, and Bronze Class values. Therefore, the unique number of badges a user has won will be equal to the number of times that the userId appears in the table which is equivalent to the number of unique Badge Id values there are for that specific user id. I won't be splitting up my count based on class value or specific badge name.

```
query = "SELECT DISTINCT Accepted_Answer_UserID_Poster, Users.DisplayName, Users.CreationDate, Users.Lo
FROM Posts AS Questions
JOIN Posts AS Answers
ON Questions.PostTypeId = 1 AND Answers.Id = Questions.AcceptedAnswerId
WHERE Questions.PostTypeId = 1)
LEFT JOIN Users
ON Accepted_Answer_UserID_Poster = Users.Id
WHERE Accepted_Answer_UserID_Poster != ''
GROUP BY Accepted_Answer_UserID_Poster
ORDER BY Num_Ans DESC
LIMIT 75"
dg = dbGetQuery(db, query)
head(dg, 20)
```

```
##    Accepted_Answer_UserID_Poster             DisplayName
## 1                            805                  Glen_b
## 2                            919                  whuber
## 3                          28500                     EdM
## 4                         204068                   gunes
## 5                          35989                     Tim
## 6                           1352         Stephan Kolassa
## 7                           7224                  Xi'an
## 8                          53690           Richard Hardy
## 9                         173082                     Ben
## 10                         85665                 BruceET
## 11                           686              Peter Flom
## 12                          7290 gung - Reinstate Monica
## 13                         22311                 Sycorax
## 14                         11887      kjetil b halvorsen
## 15                          7486             Robert Long
## 16                        164061        Sextus Empiricus
## 17                          8013                   AdamO
## 18                        247274                    Dave
## 19                        116195                    Noah
## 20                         28746     Alecos Papadopoulos
##               CreationDate                     Location Num_Ans
## 1  2010-08-07T08:40:07.287               I'm right here    2335
## 2  2010-08-13T15:29:47.140                                 1781
## 3  2013-07-26T15:11:03.380                                 1246
## 4  2018-04-12T10:42:43.307                Cambridge, UK    1119
## 5  2013-12-10T21:19:06.223               Warsaw, Poland    1004
## 6  2010-09-18T10:55:08.240                  Switzerland     985
## 7  2011-11-05T07:56:15.903                Paris, France     984
## 8  2014-08-08T10:57:13.613                       Europe     872
## 9  2017-08-10T03:27:26.793          Canberra, Australia     826
## 10 2015-08-11T17:22:01.590       San Francisco Bay Area     814
## 11 2010-08-03T19:42:40.907                 New York, NY     718
## 12 2011-11-09T04:43:15.613              Kingdom of Zhao     670
## 13 2013-03-20T23:59:56.610 Washington, DC, United States     611
## 14 2012-06-09T22:52:37.473                      Bolivia     592
## 15 2011-11-20T14:30:04.120        Leeds, United Kingdom     590
## 16 2017-06-05T10:39:01.763            Sion, Switzerland     475
## 17 2011-12-14T21:46:36.197                  Nakoja Abad     452
```

```
## 18 2019-05-08T13:23:32.777                                    439
## 19 2016-05-19T15:46:14.703  Cambridge, MA, United States     424
## 20 2013-08-02T14:24:21.923                                    422
```

This is how I created a View to more easily organize my table: https://www.w3schools.com/sql/sql_view.asp

```
query = "CREATE VIEW [tab2] AS
SELECT DISTINCT Accepted_Answer_UserID_Poster, Users.DisplayName, Users.CreationDate, Users.Location, C
FROM Posts AS Questions
JOIN Posts AS Answers
ON Questions.PostTypeId = 1 AND Answers.Id = Questions.AcceptedAnswerId
WHERE Questions.PostTypeId = 1)
LEFT JOIN Users
ON Accepted_Answer_UserID_Poster = Users.Id
WHERE Accepted_Answer_UserID_Poster != ''
GROUP BY Accepted_Answer_UserID_Poster
ORDER BY Num_Ans DESC
LIMIT 75"
dbExecute(db, query)
```

```
## [1] 0
```

```
query = "SELECT T.*, M.Freq FROM [tab2] AS T
LEFT JOIN (SELECT DISTINCT Badges.UserId AS BID, COUNT(Badges.UserId) AS Freq FROM Badges GROUP BY BID)
ON T.Accepted_Answer_UserID_Poster = M.BID"
output_tab = dbGetQuery(db, query)
head(output_tab)
```

```
##   Accepted_Answer_UserID_Poster     DisplayName            CreationDate
## 1                           805          Glen_b 2010-08-07T08:40:07.287
## 2                           919          whuber 2010-08-13T15:29:47.140
## 3                         28500             EdM 2013-07-26T15:11:03.380
## 4                        204068           gunes 2018-04-12T10:42:43.307
## 5                         35989             Tim 2013-12-10T21:19:06.223
## 6                          1352 Stephan Kolassa 2010-09-18T10:55:08.240
##         Location Num_Ans Freq
## 1 I'm right here    2335 1605
## 2                   1781 1942
## 3                   1246  318
## 4  Cambridge, UK    1119  129
## 5 Warsaw, Poland    1004  717
## 6    Switzerland     985  638
```

Now the output table printed above gives us the user's display name, the location, the creation date, and the number of badges they have won – in the way that I have interpreted it which is to include the badges for all badge classes. Next, I need to combine all of the unique badge names that the user won.

```
query = "CREATE VIEW [tab3] AS SELECT T.*, M.Freq FROM [tab2] AS T
LEFT JOIN (SELECT DISTINCT Badges.UserId AS BID, COUNT(Badges.UserId) AS Freq FROM Badges GROUP BY BID)
ON T.Accepted_Answer_UserID_Poster = M.BID"
dbExecute(db, query)
```

```
## [1] 0
```

```
#create a view to store this progres of the table
```

Now I am going to figure out how to merge the badge names for each distinct user. Below we get the table of all unique badge names for every single user in the Badges table. After this step, I will need to limit it to only take the strings that are in my [tab3] table View that I made.

```
query = "SELECT DISTINCT Badges.UserId, GROUP_CONCAT(DISTINCT Badges.Name) AS Badge_Names
FROM Badges
GROUP BY Badges.UserId"
head(dbGetQuery(db, query))
```

```
##   UserId
## 1      2
## 2      3
## 3      4
## 4      5
## 5      6
## 6      7
##
## 1
## 2
## 3
## 4 Teacher,Student,Editor,Supporter,Self-Learner,Commentator,Mortarboard,Organizer,Critic,Nice Questi
## 5
## 6
```

```
query = "SELECT C.*, P.Badge_Names FROM [tab3] AS C
LEFT JOIN (SELECT DISTINCT Badges.UserId, GROUP_CONCAT(DISTINCT Badges.Name) AS Badge_Names
FROM Badges
GROUP BY Badges.UserId) AS P
ON C.Accepted_Answer_UserID_Poster = P.UserId"
new_output_tab = dbGetQuery(db, query)
head(new_output_tab)
```

```
##   Accepted_Answer_UserID_Poster     DisplayName            CreationDate
## 1                           805          Glen_b 2010-08-07T08:40:07.287
## 2                           919          whuber 2010-08-13T15:29:47.140
## 3                         28500             EdM 2013-07-26T15:11:03.380
## 4                        204068           gunes 2018-04-12T10:42:43.307
## 5                         35989             Tim 2013-12-10T21:19:06.223
## 6                          1352 Stephan Kolassa 2010-09-18T10:55:08.240
##         Location Num_Ans Freq
## 1 I'm right here    2335 1605
## 2                   1781 1942
## 3                   1246  318
## 4  Cambridge, UK    1119  129
## 5 Warsaw, Poland    1004  717
## 6    Switzerland     985  638
##
## 1 Teacher,Editor,Supporter,Yearling,Commentator,Critic,Student,Scholar,Nice Answer,Analytical,Custodi
```

```
## 2
## 3
## 4
## 5
## 6
```

```
query = "CREATE VIEW [tab4] AS SELECT C.*, P.Badge_Names FROM [tab3] AS C
LEFT JOIN (SELECT DISTINCT Badges.UserId, GROUP_CONCAT(DISTINCT Badges.Name) AS Badge_Names
FROM Badges
GROUP BY Badges.UserId) AS P
ON C.Accepted_Answer_UserID_Poster = P.UserId"
dbExecute(db, query)
```

```
## [1] 0
```

In our table, the owner user ID will guide us to the actual answer posts themselves. From there, I will look into CreationDate of them all and find the MIN and MAX. Below in our tab4 table, we have the user's display name, the creation date of the user, the location of the user, the number of badges that they have won (which is in the column named Freq), the Num_Ans attribute which tells us how many times this user has posted an accepted answer. This is used to find the top 75 users since we order by this column. I also have the names of the badges in a single string in the Badge_Names column.

```
head(dbGetQuery(db, "SELECT * FROM [tab4]"))
```

```
##   Accepted_Answer_UserID_Poster     DisplayName            CreationDate
## 1                           805          Glen_b 2010-08-07T08:40:07.287
## 2                           919          whuber 2010-08-13T15:29:47.140
## 3                         28500             EdM 2013-07-26T15:11:03.380
## 4                        204068           gunes 2018-04-12T10:42:43.307
## 5                         35989             Tim 2013-12-10T21:19:06.223
## 6                          1352 Stephan Kolassa 2010-09-18T10:55:08.240
##         Location Num_Ans Freq
## 1 I'm right here    2335 1605
## 2                   1781 1942
## 3                   1246  318
## 4  Cambridge, UK    1119  129
## 5 Warsaw, Poland    1004  717
## 6    Switzerland     985  638
##
## 1 Teacher,Editor,Supporter,Yearling,Commentator,Critic,Student,Scholar,Nice Answer,Analytical,Custodi
## 2
## 3
## 4
## 5
## 6
```

Now what we have to do is find the dates of the earliest and most recent accepted answers that the user posted along with the unique tags for all the questions for which they had the accepted answers. We have the user ID who posted the accepted answers, and we need to now go through these answers and find the min and max creation date.

```
query = "SELECT T.Accepted_Answer_UserID_Poster, MIN(DATETIME(Answers.CreationDate)) AS Earliest, MAX(DA
FROM [tab4] AS T
JOIN Posts AS Answers
WHERE T.Accepted_Answer_UserID_Poster = Answers.OwnerUserId AND Answers.PostTypeId = 2
"
```

Now that we are done using our tables, we can get rid of them.

```
query = "DROP VIEW tab2"
dbExecute(db, query)
```

```
## [1] 0
```

```
query = "DROP VIEW tab3"
dbExecute(db, query)
```

```
## [1] 0
```

```
query = "DROP VIEW tab4"
dbExecute(db, query)
```

```
## [1] 0
```

24. How many questions received no answers (accepted or unaccepted)? How many questions had no accepted answer? Saisha Hongal helped me figure out this problem, thank you Saisha! First I will find out how many questions received no answers. This will mean that I am going to find all the question posts which have an answer count of 0 since that means that there are no affiliated answers with that question.

```
query = "SELECT COUNT(Posts.Id) AS Num_Questions_With_No_Answers
FROM Posts
WHERE Posts.PostTypeId = 1 AND Posts.AnswerCount = 0"
a1 = dbGetQuery(db, query)
a1
```

```
##   Num_Questions_With_No_Answers
## 1                         66970
```

Now I am going to find out how many questions had no **accepted answers**. There could be questions that have answers but not accepted answers, but the way that I am interpreting this question is to find the number of questions that have no accepted answers. This means that they will have an empty value for the AcceptedAnswerId part of the table. The first way will be to find questions with no accepted answers have an empty value in the AcceptedAnswerId column.

```
query = "SELECT COUNT(Posts.Id) AS Num_Questions_With_No_Accepted_Answer
FROM Posts
WHERE PostTypeId = 1 AND AcceptedAnswerId = ''"
dbGetQuery(db,  query)
```

```
##   Num_Questions_With_No_Accepted_Answer
## 1                                136365
```

Therefore, we can conclude that there are **136,365** questions that have no accepted answer.

25. What is the distribution of answers per posted question? I am going to approach the problem in the following way: -Each post is associated with a Parent ID if and only if the post is an answer (meaning if and only if the PostTypeId = 2) -A parent ID associated with an answer post is the Id referring back to the question -Therefore, in order to find the distribution of answers per posted question, I am going to find the frequency table of how many times each ParentId occurs. For example, if there is a ParentId 99989 that occurs two times, that means that there are two answers to the question 99989.

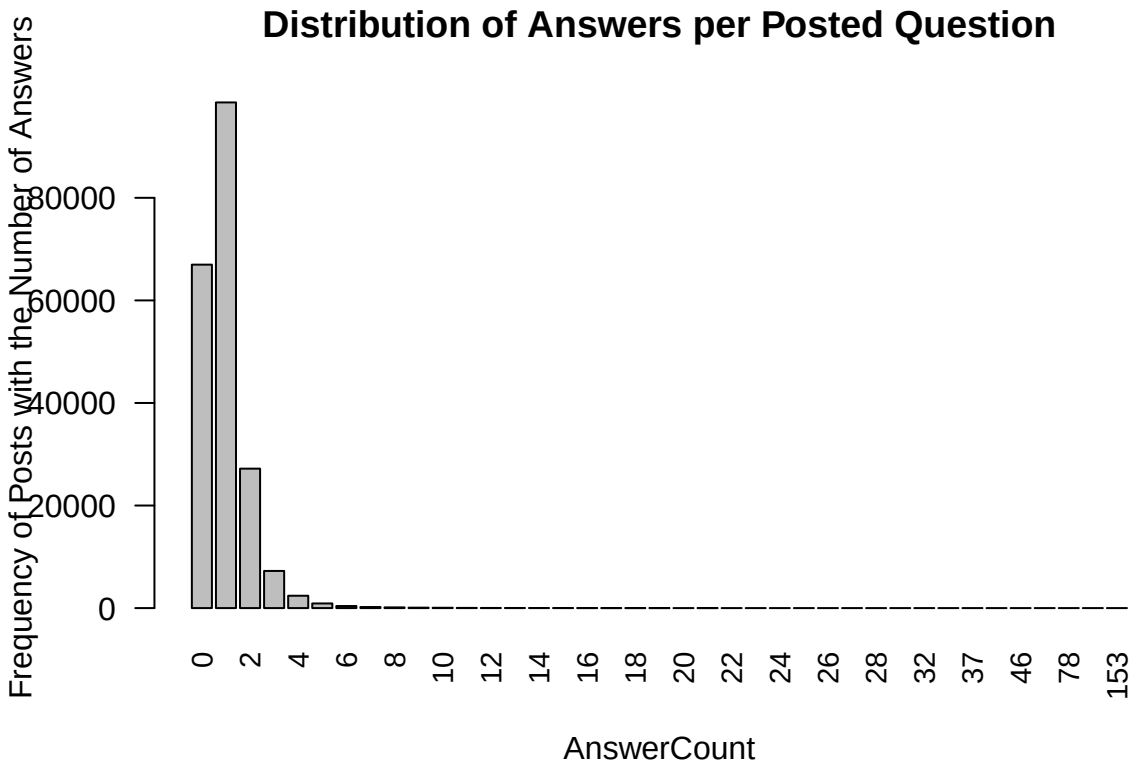
Another way that we can approach this problem is by extracting the answer count values that are present for all the questions in the posts category. I will do this method first, and then verify my code with the approach that I wrote first.

```
query = "
SELECT DISTINCT Posts.AnswerCount, COUNT(Posts.AnswerCount) AS Frequency
FROM Posts
WHERE Posts.PostTypeId = 1
GROUP BY AnswerCount
"
freq_table = dbGetQuery(db, query)
head(freq_table)
```

```
##   AnswerCount Frequency
## 1           0     66970
## 2           1     98602
## 3           2     27191
## 4           3      7246
## 5           4      2408
## 6           5       905
```

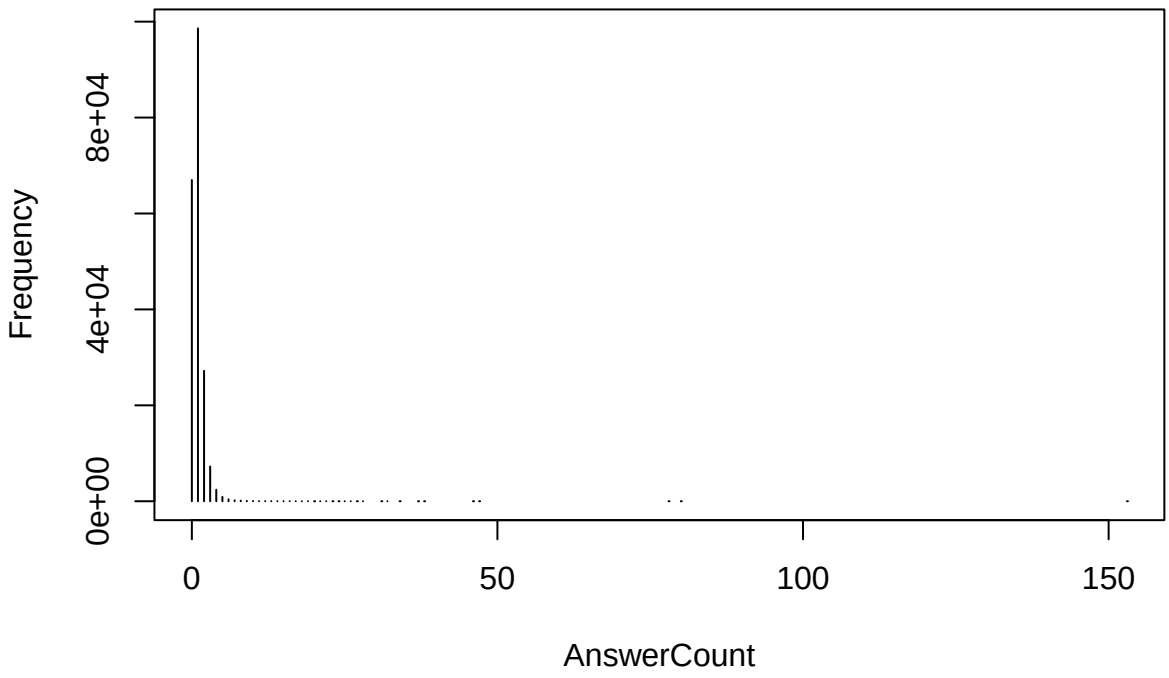
Above I have the distribution of the number of answers there are for all posted questions. I am going to make a plot of this data to more clearly see the distribution.

```
barplot(freq_table$Frequency, xlab = "AnswerCount", ylab = "Frequency of Posts with the Number of Answe
        cex.names = 0.9)
```



Here is another way that we can plot the data: https://www.learnbyexample.org/r-plot-function/

```
plot(freq_table,type = "h")
```



In order to verify if my table above makes sense, the total sum of the frequency should add up to the total number of questions there are in our Posts table. In previous questions, we found that this value is equal to 204,370.

```
query = " SELECT SUM(Frequency) FROM
(
SELECT DISTINCT Posts.AnswerCount, COUNT(Posts.AnswerCount) AS Frequency
FROM Posts
WHERE Posts.PostTypeId = 1
GROUP BY AnswerCount
)
"
dbGetQuery(db, query)
```

```
##   SUM(Frequency)
## 1         204370
```

In order to verify if the results above are correct, I am now going to find this same frequency table with another method by counting the number of occurrences for each ParentId, and then finding the frequency of this resulting query. Considering the structure and the schema of this database, both method should give us the same answer, and this is how I will verify if I am correct.

Not all questions have answers, and this makes sense since some questions have an AnswerCount of 0. For those questions with no answers, there will be no ParentIds that associate with those questions. In this case, we will expect that the number of unique ParentIds (which will be the number of questions with answers) will be less than the number of questions. This is because some questions won't have ParentIds. If the

ParentId is 0, that means that R coerced the empty ParentId to be 0 since all of the other values are of type int. Therefore, I am going to make sure that I don't include any rows where the ParentId is 0 since that means that those are for posts which are questions.

```
query = "SELECT DISTINCT Posts.ParentId, COUNT(Posts.ParentId) AS Freq
FROM Posts
WHERE Posts.ParentId != 0 AND Posts.PostTypeId = 2
GROUP BY Posts.ParentId"
f = dbGetQuery(db, query)
head(f)
```

```
##   ParentId Freq
## 1        1    6
## 2        2    7
## 3        3   19
## 4        4    5
## 5        6   20
## 6        7   25
```

```
head(dbGetQuery(db, "SELECT COUNT(Freq) AS Num_Questions_With_Answers FROM (SELECT DISTINCT Posts.Paren
FROM Posts
WHERE Posts.ParentId != 0 AND Posts.PostTypeId = 2
GROUP BY Posts.ParentId)"))
```

```
##   Num_Questions_With_Answers
## 1                     137400
```

We are finding the number of questions with answers by finding the number of rows in our table which also correlates to the unique ParentIds. According to the query above, there are 137,400 unique ParentIds which correlates to 137,400 questions that have answers.

Now we will find the distribution of the number of answers for all posted questions. We are excluding questions which have 0 answers since those don't have valid ParentIds (the ParentId is empty since it is a question and not an answer).

```
query = "SELECT DISTINCT Freq, COUNT(Freq) AS Num_Posts FROM (SELECT DISTINCT Posts.ParentId, COUNT(Pos
FROM Posts
WHERE Posts.ParentId != 0 AND Posts.PostTypeId = 2
GROUP BY Posts.ParentId) GROUP BY Freq"
head(dbGetQuery(db, query))
```

```
##   Freq Num_Posts
## 1    1     98602
## 2    2     27191
## 3    3      7246
## 4    4      2408
## 5    5       905
## 6    6       401
```

Interpreting the table above, this tells us that there are 98,602 posts with 1 answer.

To confirm that this is right, the number of of this frequency table above needs to be equal to the total number of questions with at least 1 answer. We have verified this below where the sum of this frequency

table is equal to 137,400. From the first frequency table we made above for this problem, we found that 66970 questions have 0 answers. We will subtract this from the total number of questions there are, which we found to be 204,370.

204370 (total number of questions) - 66970 (questions with 0 answers) = 137400 (number of questions with at least 1 answer). Therefore, we have verified that we have the right frequency table for the distribution of answers per posted question.

```
query = "SELECT SUM(Num_Posts) FROM (SELECT DISTINCT Freq, COUNT(Freq) AS Num_Posts FROM (SELECT DISTINC
FROM Posts
WHERE Posts.ParentId != 0 AND Posts.PostTypeId = 2
GROUP BY Posts.ParentId) GROUP BY Freq)"
head(dbGetQuery(db, query))
```

```
##   SUM(Num_Posts)
## 1         137400
```

26. What is the length of time for a question to receive an answer? To obtaining an accepted answer? I will first work on figuring out the length of time for a question to receive an accepted answer. This should contain 68,004 rows since we discovered earlier that there are 68,004 questions with accepted answers. In order to do this problem, I get the question ID, the question's creation date, and I find the creation date of the affiliated accepted answer.

Below I am verifying that the total number of questions that have an accepted answer is 68,004.

```
query = "SELECT COUNT(*) FROM (SELECT Question.Id AS QuestionID, COUNT(Answer.Id) AS Num_Total_Answers
FROM Posts AS Question
JOIN Posts AS Answer
ON Question.Id = Answer.ParentId AND Answer.PostTypeId = 2 AND Answer.Id = Question.AcceptedAnswerId
GROUP BY Question.Id)"
dbGetQuery(db, query)
```

```
##   COUNT(*)
## 1    68004
```

For this problem, I worked with Saisha Hongal to get some help on the approach. We started off by getting all of the questions that correlate to questions which have accepted answers, and then collected the creation date of this question. Now we need to traverse the creation date for the accepted answer. This wil be the one creation date after we correctly choose all tuples from our joining process.

We need to do a self join to get these question and answer pairs, and we need to make sure that the questions are of type question (by using the PostTypeId), and we need to make sure that the Answer has the Id of the question's accepted answer Id. This will ensure that all of the answers we find are of type PostTypeId 2 since only accepted answers will be linked in the Question AcceptedAnswerId category.

In order to find the duration, I used https://www.techonthenet.com/sqlite/functions/julianday.php for JULIANDAY.

```
query = "SELECT Question.Id AS QuestionId, Question.CreationDate AS Question_Ask_Date, MIN(Answers.Creat
FROM Posts AS Question
JOIN Posts AS Answers
ON Question.PostTypeId = 1 AND Answers.Id = Question.AcceptedAnswerId
WHERE Answers.PostTypeId = 2
GROUP BY Question.Id"
head(dbGetQuery(db, query))
```

```
##   QuestionId       Question_Ask_Date        Answer_Post_Date Duration_Days
## 1          1 2010-07-19T19:12:12.510 2010-07-19T19:19:46.160  5.250579e-03
## 2          2 2010-07-19T19:12:57.157 2010-07-19T19:43:20.423  2.110262e-02
## 3          3 2010-07-19T19:13:28.577 2010-07-19T19:14:43.050  8.619558e-04
## 4          4 2010-07-19T19:13:31.617 2010-07-19T21:36:12.850  9.908835e-02
## 5          7 2010-07-19T19:15:59.303 2010-07-19T19:24:18.580  5.778669e-03
## 6         10 2010-07-19T19:17:47.537 2010-08-19T10:00:00.370  3.061265e+01
```

In order to check if the correct accepted answer post date was found, I manually checked the post table. For example, for the question id 10, I checked the Posts table to find the associated accepted answer Id.

```
ansId = posts_db[which(posts_db$Id == 10),]$AcceptedAnswerId
print("Accepted Answer Creation Date for Question Id 10")
```

```
## [1] "Accepted Answer Creation Date for Question Id 10"
```

```
posts_db[which(posts_db$Id == ansId),]$CreationDate
```

```
## [1] "2010-08-19T10:00:00.370"
```

This checks out in our table.

Now I will generalize this problem to find the length of time for a question to receive an answer in general where the answer does not need to be an accepted answer. In R, I am going to verify how many rows I will be expecting in my sequel output. I am going to get rid of the empty ParentId value since that correlates to all of the entries which could be types of posts other than answers which don't have a parentId attribute so they are empty. Therefore, I will be expecting 137,400 rows in my output.

```
length(unique(posts_db[which(posts_db$ParentId !=''),]$ParentId))
```

```
## [1] 137400
```

```
query = "SELECT DISTINCT(Question.Id) AS Question_ID, DATETIME(Question.CreationDate) AS Question_Post_
MIN(DATETIME(Answer.CreationDate)) AS Fastest_Answer_Post_Date, JULIANDAY(MIN(DATETIME(Answer.CreationDa
FROM Posts as Question
JOIN Posts AS Answer
ON Question.Id = Answer.ParentId AND Question.PostTypeId = 1 AND Answer.PostTypeId = 2
GROUP BY Question.ID
ORDER BY Question.ID ASC"

head(dbGetQuery(db, query))
```

```
##   Question_ID  Question_Post_Date Fastest_Answer_Post_Date Duration_In_Days
## 1           1 2010-07-19 19:12:12      2010-07-19 19:19:46     0.0052546295
## 2           2 2010-07-19 19:12:57      2010-07-19 19:24:35     0.0080787037
## 3           3 2010-07-19 19:13:28      2010-07-19 19:14:43     0.0008680555
## 4           4 2010-07-19 19:13:31      2010-07-19 21:31:53     0.0960879629
## 5           6 2010-07-19 19:14:44      2010-07-19 19:18:56     0.0029166662
## 6           7 2010-07-19 19:15:59      2010-07-19 19:18:41     0.0018749996
```

27. How many answers are typically received before the accepted answer? For this problem, I was getting really stuck on it and Summer Monga and Saisha Hongal helped me in my thinking approach.

First I will find the number of answers for each question. This includes the non accepted and accepted answers – if the accepted answer exists.

```
query = "SELECT Question.Id AS QuestionID, COUNT(Answer.Id) AS Num_Total_Answers
FROM Posts AS Question
JOIN Posts AS Answer
ON Question.Id = Answer.ParentId AND Answer.PostTypeId = 2
GROUP BY Question.Id"
head(dbGetQuery(db, query))
```

```
##   QuestionID Num_Total_Answers
## 1          1                 6
## 2          2                 7
## 3          3                19
## 4          4                 5
## 5          6                20
## 6          7                25
```

We know that 137,400 correlates to the number of questions with at least one answer, and we don't know if these answers are accepted or not but we know that these questions have answers. We can order the questions by ascending creation date so we know that wherever the accepted answer ID is, the number of unaccepted answers before that will have a creation date less than the accepted answer creation date.

Only Answers have a ParentId since that will point towards the question to which it is answering, and only Questions will have AcceptedAnswerIds – if they exist.

We are going to use the ParentIds as the indexes for the questions since that is how we find every question that has an answer. Then, we are going to count the distinct number of question Ids there are after joining all of our tuples to make sure that we keep the rows in which the accepted answer is the answer we are counting for. The first joining gave us the number of answers in total including the accepted answer.

```
query = "SELECT Question.ParentId AS Question_Id, COUNT(DISTINCT Question.Id) AS Num_Unaccepted_Answers_
FROM Posts AS Question
JOIN Posts AS Answer
LEFT JOIN Posts AS Bf ON Answer.AcceptedAnswerId = Bf.Id
WHERE Question.ParentId = Answer.Id AND Answer.AcceptedAnswerId != '' AND Answer.PostTypeId = 1
AND Question.CreationDate <= Bf.CreationDate
GROUP BY Question_Id
ORDER BY Question_Id ASC"
head(dbGetQuery(db, query))
```

```
##   Question_Id Num_Unaccepted_Answers_Before_Accepted
## 1           1                                     1
## 2           2                                     2
## 3           3                                     1
## 4           4                                     2
## 5           7                                     2
## 6          10                                     3
```

Therefore, in the next go we need to subtract 1 from the count of question ids in order to get rid of the count of the accepted answer id since we only want to know how many unaccepted answers come before. Below is the final answer.

```
query = "SELECT Question.ParentId AS Question_Id, COUNT(DISTINCT Question.Id)-1 AS Num_Unaccepted_Answe
FROM Posts AS Question
JOIN Posts AS Answer
LEFT JOIN Posts AS Bf ON Answer.AcceptedAnswerId = Bf.Id
WHERE Question.ParentId = Answer.Id AND Answer.AcceptedAnswerId != '' AND Answer.PostTypeId = 1
AND Question.CreationDate <= Bf.CreationDate
GROUP BY Question_Id
ORDER BY Question_Id ASC"
head(dbGetQuery(db, query))
```

```
##   Question_Id Num_Unaccepted_Answers_Before_Accepted
## 1           1                                     0
## 2           2                                     1
## 3           3                                     0
## 4           4                                     1
## 5           7                                     1
## 6          10                                     2
```