STA141B Assignment 3 Databases

```r
library(RSQLite)
library(DBI)
db <- dbConnect(RSQLite::SQLite(),
"C:/Users/10bra.LAPTOP-539022QK/Downloads/stats.stackexchange (1).db")
dbListTables(db)
```

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

1. How many posts are there?

```r
bodycount <-"SELECT COUNT(*) as numposts
FROM Posts
WHERE Body IS NOT NULL"
bodylen <- dbGetQuery(db, bodycount)
```

I decided that the body of the posts is what the question is asking for and I didn't want to include any posts that were empty or null, so I chose to not include any null posts and there weren't any luckily enough. Since we were looking for the number of posts, it was important to use the "Posts" table to access the data we needed. Using count(*), I was able to gather a sum of all of the instances where there was a post, which gathered the data I needed for this question. The Posts table is the most helpful for finding the number of Posts which is what I used.

```r
print(bodylen)
```

```
##   numposts
## 1   405220
```

It looks like there are 405,220 posts in this database. This number is a count of every post, so repeated posts are included in this total. I figured the total posts would include every iteration of a post even if there are replicas because it didn't ask for unique posts.

2. How many posts are there since 2020? (Hint: Convert the CreationDate to a year.)

```r
selectdate <- "SELECT COUNT(*) AS after2020
FROM Posts
WHERE strftime('%Y', CreationDate) >= '2020' and Body IS NOT NULL"
datesfor2020 <- dbGetQuery(db, selectdate)
```

Once again, to find the number of posts after 2020, it is ideal to use the Posts table. In SQL, the strftime function let me create a format for which I wanted the column CreationDate, and all of the creation dates were in the same format. All of the posts that are not empty and have a creation date after 2020 are included. I used a pattern of %Y to isolate the year part of the creation date and set it greater than or equal to 2020 so dates before 2020 do not show. Using the "Posts" table for this problem was necessary because we wanted to know how many posts there were after 2020.

```
print(datesfor2020)
```

```
##   after2020
## 1    110949
```

After 2020, there are 110,949 posts. This is from 2020-2023, being about a quarter of the total posts, which is logical since the data starts from 2009.

3. How many posts are there each year? Describe this with a plot, commenting on any anomalies

```
annualdates <- "SELECT strftime('%Y', CreationDate) AS year,
COUNT(*) AS posts_per_year
FROM Posts GROUP BY year ORDER BY year;"
peryear <- dbGetQuery(db,annualdates)
```

To find the number of posts per year, it is important to use the Posts table as it has all of the posts within it. I used the same idea as problem 2 where I isolated the creation date using the pattern %Y, but I also included a count that would group each year that is exactly the same. This means that 2020 posts would be grouped together and 2021 posts would be grouped together etc.

```
print(peryear)
```

```
##    year posts_per_year
## 1  2009             18
## 2  2010           5393
## 3  2011          12921
## 4  2012          19562
## 5  2013          25024
## 6  2014          32618
## 7  2015          37150
## 8  2016          40986
## 9  2017          42288
## 10 2018          39705
## 11 2019          38606
## 12 2020          39909
## 13 2021          32518
## 14 2022          32813
## 15 2023           5709
```

The data starts from 2009 and goes all the way until 2023. There aren't very many posts in 2009 or 2023, but most years the site had around 30-40 thousand posts, showing how active the site is.

4. How many tags are in most questions?

```r
#average tags per question
taggs <- "SELECT ID, COUNT(DISTINCT Tag) AS NumTags
FROM TagPosts
WHERE ID IN (SELECT ID FROM Posts WHERE PostTypeId = 1)
GROUP BY ID"
tagresult <- dbGetQuery(db, taggs)
avgtags <- mean(tagresult$NumTags)
```

Using the TagPosts and Posts table I was able to find the number of tags with the questions as a parameter. I decided that to calculate how many tags are in mostquestions, it would be sufficient to calculate the tags that are in PostTypeId=1 which isolates the questions by themselves without the other types of posts. After calculating all of the tags, I used R to calculate the average tags.

```r
print(avgtags)
```

```
## [1] 3.08534
```

The average number of tags is around 3 tags. Some people like to use a multitude of tags while others opt for little to no tags, so this number is logical considering how many people don't use tags at all.

5. How many posted questions are there?

```r
# number of questions
totalquest <- "SELECT COUNT(*) AS totalquestions
FROM Posts
WHERE PostTypeId = 1"
quest <- dbGetQuery(db,totalquest)
```

I used the Posts table to find the number of questions because every post was classified by a type of post, question being one of them. I counted all of the questions from the posts which is where PostTypeId = 1. In the PostTypeIdMap, it says that questions are denoted with a PostTypeId of 1, which is what I used to gather the amount of questions in the database. The posted questions were counted up and put into a table with the total amount.

```r
print(quest)
```

```
##   totalquestions
## 1         204370
```

There are 204,370 questions, which means that around half of the database is questions. This is quite logical as the site is used to ask questions and receive answers.

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.

```r
#50 most common tags
comtags <- "SELECT Tag, COUNT(*) AS TagCount
FROM (SELECT DISTINCT Id, Tag FROM TagPosts WHERE Id IN
(SELECT ID FROM Posts WHERE PostTypeId = 1)) AS SubQuery
GROUP BY Tag
ORDER BY TagCount DESC
LIMIT 50"
commontags <- dbGetQuery(db, comtags)
```

Using the TagPosts table and Posts table, it was easy to find the 50 most common tags. For this question, I needed to count tags, but I needed the values to be seperate. By ordering the tags that were counted in descending order, the tags that were most used were on the top of the table and they went all the way down until the 50th value which was set by the limit of 50.

```r
print(commontags)
```

```
##                          Tag TagCount
## 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
```

It seems the data generally uses the r or regression tags which are more general instead of more specific tags like chi-squared-test or linear-model which are less used as specific tags will most likely be less frequent then general tags.

7. How many answers are there?

```
#number of answers
numansw <- "SELECT COUNT(*) AS NumAnswers FROM Posts WHERE PostTypeId = 2"
ans <- dbGetQuery(db, numansw)
```

To find the number of answers, I found it beneficial to use the Posts table. Instead of using the "AnswerCount" from the posts table, I chose to use the PostTypeID = 2 because the answer count was equivalent to the number of posts. I figured that was an inaccurate count of purely answers, so I chose to use the PostTypeID which was about half of the total posts which is logical.

```
print(ans)
```

```
##   NumAnswers
## 1     197928
```

The number of answers is 197,928, which is less than half of the database. This answer is logical as the PostTypeIdMap shows that there are 8 types of posts which means that even if a majority are questions and answers, there needs to be some space for the other 6 types.

8. 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?

```
#most recent question by date time
recquestion <- "SELECT  Title, Body, CreationDate
FROM Posts
WHERE PostTypeId = 1
ORDER BY CreationDate DESC
LIMIT 1"
questdate <- dbGetQuery(db, recquestion)
head(questdate)
```

```
##                                                                            Title
## 1 Are there any papers or methods that combine mcmc and variational inference
##
## 1 <p>Are there any methods that combine VI and MCMC? If it exists, why isn't it used prominently ove
##              CreationDate
## 1 2023-03-05T05:10:18.393
```

```
# URL https://stats.stackexchange.com/questions/608458/are-there-any-methods-that-combine-mcmc-and-vi
```

The Posts table had all of the information I needed to solve this problem. To find the most recent question, I decided to include the title,body, and creation date so that I could make sure that the question being asked at the latest part of the creation date is being accounted for. The title and body are redundant and the limit is 1 because the question is only asking for the last question which means only one question. PostTypeId is still 1 because we are only looking for questions and the Posts table is once again used to grab the question we need. The URL is within the code chunk.

```
print(questdate)
```

```
##                                                                              Title
## 1 Are there any papers or methods that combine mcmc and variational inference
##
## 1 <p>Are there any methods that combine VI and MCMC? If it exists, why isn't it used prominently ove
##               CreationDate
## 1 2023-03-05T05:10:18.393
```

The data shows that someone wants to know how to combine VI and MCMC and why it isn't used over other methods.

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

```
#contain term
containterm <- "SELECT SUM(CASE WHEN Body LIKE '%Regression%' THEN 1 ELSE 0 END)
AS Regression, SUM(CASE WHEN Body LIKE '%ANOVA%' THEN 1 ELSE 0 END) AS ANOVA,
SUM(CASE WHEN Body LIKE '%Data Mining%' THEN 1 ELSE 0 END) AS DataMining,
SUM(CASE WHEN Body LIKE '%Machine Learning%' THEN 1 ELSE 0 END) AS MachineLearning,
SUM(CASE WHEN Body LIKE '%Deep Learning%' THEN 1 ELSE 0 END) AS DeepLearning,
SUM(CASE WHEN Body LIKE '%Neural Network%' THEN 1 ELSE 0 END) AS NeuralNetwork
FROM Posts
WHERE PostTypeId = 1"
questterm <- dbGetQuery(db,containterm)
```

Within the posts table, by using a Case When within while selecting the words, I was able to count 1 if the term is used in the body instead of counting every time the term is used. I concluded that the vague question was only interested if a term was used at all inside the body instead of counting every instance the term was used. Once again, PostTypeId=1 as we are only interested in questions and not answers. I figured I didn't need to account for titles because if the user didn't ask it in their question their question was problaly not too related to the key word we were asked to find.

```
print(questterm)
```

```
##   Regression ANOVA DataMining MachineLearning DeepLearning NeuralNetwork
## 1      39296  7167        405            5404         1124          5063
```

The table shows the amount of times each term was found inside a title or body. Regression is used many times at 39296 , but data mining is only used 405 times. More specific problems are less likely to be asked as much as generic problems like Regression.

14. What is the date range for the questions and answers in this database?

```
rangequestion <- "SELECT CASE WHEN PostTypeId = 1
THEN 'Question' WHEN PostTypeId = 2 THEN 'Answer'END AS PostType,
MIN(CreationDate) AS EarliestDate,
MAX(CreationDate) AS LatestDate
FROM Posts
WHERE PostTypeId IN (1, 2)
GROUP BY PostTypeId"
queans <- dbGetQuery(db,rangequestion)
```

The Posts table had the range for waht I needed in this question. I interpreted this question to mean that they wanted to know the earliest question + answer and the latest question + answer. This is because the latest question and answer would give the range for the database. To achieve this, I found the minimum and maximum values for the questions and answers using the creation date. I couldn't use the year because there are many dates that would be within the same year and we may not receive the correct answer for the very first or last of what we were looking for. By using the creation date itself, we can calculate the earliest and latest date.

```
print(queans)
```

```
##   PostType             EarliestDate               LatestDate
## 1 Question 2009-02-02T14:21:12.103 2023-03-05T05:10:18.393
## 2   Answer 2009-02-02T14:24:31.740 2023-03-05T04:48:34.853
```

The table shows that the earliest question was in 2009 and the earliest answer was 3 minutes later. The latest question was in 2023, but it seems that question was unanswered as the latest answer was earlier than the last question.

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

```
#21
 q21 <- "SELECT Posts.Body AS Question, Users.DisplayName AS Username,
 Users.CreationDate AS JoinDate, Users.Location AS Location,
Posts.CreationDate AS PostDate, AcceptedPosts.Body AS AcceptedAnswer,
AcceptedPosts.CreationDate AS AcceptedAnswerPostDate,
AcceptedUsers.DisplayName AS AcceptedAnswerUsername
FROM Posts
JOIN Users ON Posts.OwnerUserId = Users.Id
LEFT JOIN Posts AS AcceptedPosts ON Posts.AcceptedAnswerId = AcceptedPosts.Id
LEFT JOIN Users AS AcceptedUsers ON AcceptedPosts.OwnerUserId = AcceptedUsers.Id
WHERE Posts.PostTypeId = 1"
table21 <- dbGetQuery(db,q21)
```

This table asks us to combine data from the Users and Posts table. We select the columns that we need from each of the tables. After tinkering with the join clauses, I realized that the original join function wasn't working because not every question is answered, so we need to use a left join to pass by it. This being the case, it makes sense as to why there would be NA values in the table, since we need to make sure that all of the information for questions are accounted for including questions that do not have answers.

```
head(table21)
```

```
##
## 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
##      Username                JoinDate                     Location
## 1 csgillespie 2010-07-19T19:04:52.280   Newcastle, United Kingdom
## 2      A Lion 2010-07-19T19:09:32.157
## 3      grokus 2010-07-19T19:08:29.070               United States
## 4 Jay Stevens 2010-07-19T19:09:16.917       Jacksonville, FL, USA
## 5       Shane 2010-07-19T19:03:57.227                New York, NY
## 6      EAMann 2010-07-19T19:11:57.393 Tualatin, OR, United States
##                 PostDate
## 1 2010-07-19T19:12:12.510
## 2 2010-07-19T19:12:57.157
## 3 2010-07-19T19:13:28.577
## 4 2010-07-19T19:13:31.617
## 5 2010-07-19T19:14:44.080
## 6 2010-07-19T19:15:59.303
##
## 1
## 2 <p>The assumption of normality is just the supposition that the underlying <a href="http://en.wikip
## 3
## 4
## 5
## 6
##   AcceptedAnswerPostDate AcceptedAnswerUsername
## 1 2010-07-19T19:19:46.160                 Harlan
## 2 2010-07-19T19:43:20.423         John L. Taylor
## 3 2010-07-19T19:14:43.050            Jay Stevens
## 4 2010-07-19T21:36:12.850         John L. Taylor
## 5                    <NA>                   <NA>
## 6 2010-07-19T19:24:18.580         Stephen Turner
```

As shown on the table, there are certainly some missing values which makes sense. Sometimes not all information is available to be seen from someone's account, accounting for the values that are NA in the table.

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?

```
#22
q22 <- "SELECT Users.DisplayName AS UserName,
COUNT(DISTINCT Questions.Id) AS NumQuestions, 0 AS NumAnswers
FROM Users
LEFT JOIN Posts Questions ON Users.Id = Questions.OwnerUserId AND Questions.PostTypeId = 1
LEFT JOIN Posts Answers ON Users.Id = Answers.OwnerUserId AND Answers.PostTypeId = 2
```

```
WHERE Answers.Id IS NULL
GROUP BY Users.Id, Users.DisplayName
HAVING NumQuestions > 0"

table22 <- dbGetQuery(db,q22)
```

Once again, since I am interested in the Users of the website, I mainly use the Users table and join the Posts table in order to filter the data where I can figure out who has posted only questions and never answered questions. These tables allow us to find the number of questions people have asked given that they have 0 answers for other people.

```
head(table22)
```

```
##   UserName NumQuestions NumAnswers
## 1   grokus            2          0
## 2   A Lion            2          0
## 3   EAMann            1          0
## 4  Alan H.           13          0
## 5     kyle            2          0
## 6   Preets            1          0
```

The table shows that there are 76,410 people who have asked questions and not answered any questions. I was told to include the number of answers, even though it is quite obvious that it will be a bunch of zero's going down.

23. Compute the table with information for the 75 users with the most accepted answers. This table should include • the user's display name, • creation date, • 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)

```
ub <- "SELECT Users.Id AS UserId, Users.DisplayName AS UserName,
Users.CreationDate AS UserJoinedDate, Users.Location AS Location,
COUNT(Badges.Id) AS NumBadges,GROUP_CONCAT(Badges.Name) AS BadgeNames
FROM Users
LEFT JOIN Badges ON Users.Id = Badges.UserId
GROUP BY Users.Id
ORDER BY NumBadges DESC
LIMIT 75"

aad <- "SELECT OwnerUserId,
MIN(CreationDate) AS EarliestAcceptedAnswerDate, MAX(CreationDate)
AS MostRecentAcceptedAnswerDate
FROM Posts
WHERE PostTypeId = 2 AND AcceptedAnswerId IS NOT NULL
GROUP BY OwnerUserId"

ut <- "SELECT OwnerUserId,
GROUP_CONCAT(DISTINCT SUBSTR(Tag, 2, LENGTH(Tag) - 2)) AS UniqueTags
FROM TagPosts
JOIN Posts ON TagPosts.Id = Posts.Id
WHERE Posts.PostTypeId = 2 AND Posts.AcceptedAnswerId IS NOT NULL
```

```
GROUP BY OwnerUserId"


user_badges <- dbGetQuery(db, ub)
accepted_answer_dates <- dbGetQuery(db, aad)
```

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

```
unique_tags <- dbGetQuery(db, ut)


table23 <- merge(user_badges, accepted_answer_dates,
by.x = "UserId", by.y = "OwnerUserId", all.x = TRUE)
table23 <- merge(table23, unique_tags,
by.x = "UserId", by.y = "OwnerUserId", all.x = TRUE)
table23$UserId <- NULL
```

My code would not run using a single query, so I had to separate it so that it wouldn't have to compute so much data at once. By creating separate queries and merging them together, I was able to achieve the desired results for the table I was asked to make. Once again, left joins are used on mismatching columns and joins for matching columns in order to provide the necessary results. Since I used user id for merging, I simply removed it from the result after by making it null.

```
head(table23)
```

```
##        UserName          UserJoinedDate
## 1         Shane 2010-07-19T19:03:57.227
## 2   Rob Hyndman 2010-07-19T23:05:39.653
## 3 Jeromy Anglim 2010-07-20T02:56:34.160
## 4 russellpierce 2010-07-20T05:57:22.890
## 5    Tal Galili 2010-07-21T07:53:50.990
## 6        Henrik 2010-07-27T10:01:37.337
##                                                                    Location
## 1                                                              New York, NY
## 2                                                     Melbourne, Australia
## 3                                                     Melbourne, Australia
## 4                                                Dallas, TX, United States
## 5                                                                    Israel
## 6 University Of Warwick, Gibbet Hill Road, Coventry, Vereinigtes Königreich
##   NumBadges
## 1       178
## 2       346
## 3       423
## 4       186
## 5       369
## 6       205
##
## 1
## 2
## 3                Announcer,Announcer,Announcer,Announcer,Announcer,Announcer,Announcer,Announcer,Ann
## 4
## 5 Announcer,Announcer,Autobiographer,Benefactor,Beta,Booster,Caucus,Citizen Patrol,Civic Duty,Commen
## 6
##   EarliestAcceptedAnswerDate MostRecentAcceptedAnswerDate UniqueTags
## 1    2010-07-19T19:18:41.370      2011-01-31T19:01:59.317         NA
## 2    2010-07-19T23:26:31.473      2023-02-16T05:25:18.147         NA
## 3    2010-07-20T03:11:36.027      2021-12-10T05:06:02.473         NA
## 4    2010-07-20T07:56:06.767      2021-12-24T12:14:07.350         NA
## 5    2010-07-22T03:55:48.147      2020-12-22T09:32:58.873         NA
## 6    2010-07-27T10:24:44.870      2021-04-26T16:01:42.300         NA
```

It seems that there aren't any unique tags among the top 75 users with the most accepted answers. This is definitely not logical, so there may be a mistake in my code, but I'm not sure what I did wrong if that isn't the correct answer.

24. How many questions received no answers (accepted or unaccepted)? How many questions had no accepted answer?

```r
hi <- "UPDATE Posts
SET AcceptedAnswerId = CAST(AcceptedAnswerId AS INTEGER)"


hihi <- "SELECT PostTypeId, AcceptedAnswerId
FROM Posts
WHERE PostTypeId = 1 AND AcceptedAnswerId = 0"
hihihi <- dbGetQuery(db,hihi)

#24
noanswers <- "
SELECT COUNT(*) AS QuestionsWithNoAnswers
FROM Posts
WHERE PostTypeId = 1 AND AnswerCount = 0"

noanswers1 <- dbGetQuery(db, noanswers)

noacceptedanswer <- "
SELECT COUNT(*) AS QuestionsWithNoAcceptedAnswer
FROM Posts
WHERE PostTypeId = 1 AND (AcceptedAnswerId = 0)"

noacceptedanswer1 <- dbGetQuery(db, noacceptedanswer)
```

I updated the Posts table to take in certain columns as Integers because the calcluations were not working before that. By using PostTypeId=1 and either AcceptedAnswerId=0 or Answer Count = 0 we are able to determine the questions that have no answers vs questions that do not have any answers that the poster liked.

```r
print(noanswers1)
```

```
##   QuestionsWithNoAnswers
## 1                  66970
```

There are 66,970 questions that do not have answers. This number is logical as there are many questions that are either not seen or perhaps too difficult or unique.

```r
print(noacceptedanswer1)
```

```
##   QuestionsWithNoAcceptedAnswer
## 1                        136365
```

There are 136,365 questions that do not have an accepted answer. This is logical as there are questions with no answers and questions that have answers but weren't accepted being counted, making the number seem high but it is quite logical.

25. What is the distribution of answers per posted question?

```r
q25 <-  "SELECT AnswerCount, COUNT(*) AS QuestionCount
FROM (SELECT ParentId, COUNT(*) AS AnswerCount
FROM Posts
WHERE PostTypeId = 2
GROUP BY ParentId) AS Subquery
GROUP BY AnswerCount
ORDER BY AnswerCount"
table25 <- dbGetQuery(db,q25)
```

The ParentId was crucial in figuring out how many questions had x amount of answer counts. The groups I used would account for the number of answers in a question and count up the number of questions that had that amount of answers to it.

```r
head(table25)
```

```
##   AnswerCount QuestionCount
## 1           1         98602
## 2           2         27191
## 3           3          7246
## 4           4          2408
## 5           5           905
## 6           6           401
```

It makes sense that most questions only have one answer as people are less likely to answer a question that has already been previously answered. As the number of answers increases, the number of questions decreases significantly which is completely logical in this case.

26. What is the length of time for a question to receive an answer? to obtaining an accepted answer?

```r
q26 <- "SELECT(CAST(strftime('%s', Answers.CreationDate) AS INTEGER) -
CAST(strftime('%s', Questions.CreationDate) AS INTEGER)) / 60 AS TimeToFirstAnswerInMinutes,
(CAST(strftime('%s', MIN(CASE WHEN Posts.Id = Questions.AcceptedAnswerId
THEN Posts.CreationDate END)) AS INTEGER) - CAST(strftime('%s', Questions.CreationDate)
AS INTEGER)) / 60 AS TimeToAcceptedAnswerInMinutes
FROM Posts AS Questions
LEFT JOIN Posts AS Answers ON Questions.Id = Answers.ParentId AND Answers.PostTypeId = 2
LEFT JOIN Posts AS Posts ON Questions.Id = Posts.ParentId AND Posts.PostTypeId = 2
WHERE Questions.PostTypeId = 1
GROUP BY Questions.Id"
table26 <- dbGetQuery(db, q26)
```

I took a very literal route to answering this question. I used the difference between the dates I needed and put it over 60 to calculate the minutes between values which seems to be enough to find out what we want. I really learned a lot about the strftime function while doing this project and I'm really glad I learned more about SQL through this project.

```
head(table26)
```

```
##   TimeToFirstAnswerInMinutes TimeToAcceptedAnswerInMinutes
## 1                          7                             7
## 2                         11                            30
## 3                          1                             1
## 4                        138                           142
## 5                    3812840                            NA
## 6                          2                             8
```

It's logical that some of the first answers are earlier than or equal to the first accepted answer. Also, it makes sense that the first accepted answer is sometimes NA as not all questions have an accepted answer.