Lab 10 - SQL (Sequel) - Example Solution

Lab Purpose

All semester, we've used R as our programming language. But you have probably seen other languages (especially in CS courses) such as Java and Python. There are a variety of other statistical programs that people use (SAS, SPSS, and Minitab for example) which all require different syntax to run code or scripts.

In order to work with relational databases though, we need another language. A common one is SQL (often pronounced `sequel` but also `S-Q-L`). This lab is designed to help you learn some SQL, and to help you see how your skills translate from one language to another. For example, the principles of order of operations still apply - we can't filter for a variable if we don't have it available first. As you saw in the text, there are different versions of SQL available. For this lab, we will be focusing on *RMySQL*. The package is loaded above (be sure you install as needed).

Data

For the lab, we'll use data generated at a local institution.

Researchers at Smith College aimed to develop wideband acoustic immittance (WAI) measures as a noninvasive tool to diagnose hearing problems. The collection of WAI measures include *absorbance*, *power*, *reflectance*, *impedence*, and related quantities. As part of the project, they have developed the world's only online WAI database that collates data from different studies on this topic. We will use SQL to connect to and query the database.

Run the code below to use SQL to connect to the MySQL server that hosts the Smith WAI database.

```
con <- dbConnect(MySQL(),
                 host = "scidb.smith.edu",
                 user = "waiuser",
```

```
                    password = "smith_waiDB",
                    dbname = "wai")
```

For the rest of the lab, you'll be running queries on `con`. This is common - the connection is often called `con`. What if you are accessing two databases at once through different connections? Change the names. For example, we could have called this one `conWAI` if desired, to keep it separate from `conAirlines` if we renamed the one in the prep.

1 - Introduction to SQL

A single server can support many databases, each containing many tables, with each table having a variety of columns—it's easy to get lost when working with databases! We'll work through some commands to help figure out what's available to access in the database.

part a - What tables are included in the database (`con`)?

Solution:

```
dbGetQuery(con, "SHOW TABLES")
```

```
          Tables_in_wai
1              Codebook
2          Measurements
3 Measurements_pre2020
4               PI_Info
5           PI_Info_OLD
6              Subjects
7      Subjects_pre2020
```

We see there are 7 tables, with names printed above. There is a codebook and then 2 versions each of 3 tables - measurements, PI_info, and subjects. One version is labeled as old or pre2020.

part b - What is in the `PI_Info` table?

Solution:

```
dbGetQuery(con, "EXPLAIN PI_Info")
```

```
                Field        Type Null Key Default Extra
1          Identifier varchar(50)   NO PRI    <NA>
2                Year         int   NO        <NA>
3             Authors        text   NO        <NA>
4    AuthorsShortList        text   NO        <NA>
5               Title        text   NO        <NA>
6             Journal        text   NO        <NA>
7                 URL        text   NO        <NA>
8            Abstract        text   NO        <NA>
9   DataSubmitterName        text   NO        <NA>
```

```
10 DataSubmitterEmail        text    NO         <NA>
11      DateSubmitted        text    NO         <NA>
12           PI_Notes        text    NO         <NA>
```

Explain let's us see the variables in the data set, sort of like glimpse. We see there are 12 variables (names in list above).

part c - View the first five observations of the PI_Info table:

Solution:

```
dbGetQuery(con, "SELECT *
                 FROM PI_Info
                 LIMIT 0, 5")
```

```
    Identifier Year
1    Abur_2014 2014
2  Aithal_2013 2013
3  Aithal_2014 2014
4 Aithal_2014b 2014
5  Aithal_2015 2015
                                                                                    Authors
1                                  Defne Abur, Nicholas J. Horton, and Susan E. Voss
2                  Sreedevi Aithal, Joseph Kei, Carlie Driscoll, and Asaduzzaman Khan
3                                  Sreedevi Aithal, Joseph Kei, and Carlie Driscoll
4                                  Sreedevi Aithal, Joseph Kei, and Carlie Driscoll
5 Sreedevi Aithal, Joseph Kei, Carlie Driscoll, Asaduzzaman Khan, and Andrew Swanston
  AuthorsShortList
1     Abur et al.
2   Aithal et al.
3   Aithal et al.
4   Aithal et al.
5   Aithal et al.

1
2
3
6 months): A Cross-Sectional Study
4
5 Wideband Absorbance Outcomes in Newborns: A Comparison With High-Frequency Tympanomet
                               Journal
1                   J Am Acad Audiol
```

```
2 Int. J. Pediatr. Otorhinolaryngol.
3                  J Am Acad Audiol
4                  J Am Acad Audiol
5                          Ear Hear
                                          URL
1 https://www.ncbi.nlm.nih.gov/pubmed/25257718
2    https://pubmed.ncbi.nlm.nih.gov/23047065/
3    https://pubmed.ncbi.nlm.nih.gov/25257721/
4     https://pubmed.ncbi.nlm.nih.gov/25257722
5    https://pubmed.ncbi.nlm.nih.gov/25951046/

1 "<p> <strong> Background: </strong> Power reflectance measurements are an active area
2
3
4
86 h) and 281 ears from 158 Caucasian neonates (mean age, 42.4 h; SD, 23.0 h; range, 8.1-
152 h) who passed or failed 1000-Hz tympanometry and DPOAEs were included in the study.   </
5
  DataSubmitterName                DataSubmitterEmail DateSubmitted
1       Susan Voss                   svoss@smith.edu   24-Aug-2016
2   Sreedevi Aithal Sreedevi.aithal@health.qld.gov.au    2-Jan-2023
3   Sreedevi Aithal Sreedevi.aithal@health.qld.gov.au    2-Jan-2023
4   Sreedevi Aithal Sreedevi.aithal@health.qld.gov.au   26-Feb-2023
5   Sreedevi Aithal Sreedevi.Aithal@health.qld.gov.au   26-Feb-2023

1 Measurements made on 7 subjects across multiple sessions and 3 probe locations for ea
2
3
4
5
```

Using select we can get whatever entries we want. Here it's just the first 5. Note the indexing starts at 0.

part d - What would you change to look at records 14-16? Try it out.

Solution:

There are several ways to approach this. First, let's see what observations we are trying to get. We can use the above code to look at the first 16 entirely, and see what those are.

```
dbGetQuery(con, "SELECT *
                 FROM PI_Info
```

```
                 LIMIT 0, 16")
```

```
       Identifier Year
1       Abur_2014 2014
2     Aithal_2013 2013
3     Aithal_2014 2014
4    Aithal_2014b 2014
5     Aithal_2015 2015
6    Aithal_2017a 2017
7    Aithal_2019a 2019
8    Aithal_2019b 2019
9    Aithal_2020a 2020
10   Aithal_2020b 2020
11    Aithal_2022 2022
12 AlMakadma_2021 2021
13   Downing_2022 2022
14   Ellison_2012 2012
15    Feeney_2017 2017
16     Groon_2015 2015

1
2                                                                          Sreedevi Ai
3
4
5                                                                Sreedevi Aithal, Joseph Kei
6                             Sreedevi Aithal,  Joseph Kei, Venkatesh Aithal, Alehandrea Manue
7                                                          Sreedevi Aithal and Venkatesh Aithal
8                                                                        Sreedevi Aith
9                                                                           Sreedevi A
10                                                                          Sreedevi A
11                                                                          Sreedevi A
12
13                                                                      Cerys Downing,
14                                                 John C. Ellison, Michael Gorga, Edward Cohn
15 M. Patrick Feeney,  Douglas H. Keefe, Lisa L. Hunter,  Denis F. Fitzpatrick, Angela C
16                                                  Katherine A. Groon, Daniel M. Rasetshwane
       AuthorsShortList
1          Abur et al.
2        Aithal et al.
3        Aithal et al.
4        Aithal et al.
5        Aithal et al.
```

6 Aithal et al.
7 Aithal et al.
8 Aithal et al.
9 Aithal et al.
10 Aithal et al.
11 Aithal et al.
12 AlMakadma and Prieve
13 Downing et al.
14 Ellison et al.
15 Feeney et al.
16 Groon et al.

1
2
3
6 months): A Cross-Sectional Study
4
5 Wideband Absorbance Outcomes in Newborns: A Comparison With High-Frequency Tympanomet
6
7 Eustacl
8 Effect of Negative Middl
9 Predictive Accuracy of Wideband Absorbance at Ambient and Tympanometric P
10
11
12
13 Wideband
14
15 Normative wideband reflectanc
16

	Journal
1	J Am Acad Audiol
2	Int. J. Pediatr. Otorhinolaryngol.
3	J Am Acad Audiol
4	J Am Acad Audiol
5	Ear Hear
6	J Speech Lang Hear Res
7	J Am Acad Audiol
8	J Speech Lang Hear Res
9	J Am Acad Audiol
10	J Am Acad Audiol
11	J Am Acad Audiol
12	Ear Hear
13	Ear Hear

14 The Laryngoscope
15 Ear Hear
16 Ear Hear

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4272628/https://journals.lww.com/ear-hea

1
2
3
4
86 h) and 281 ears from 158 Caucasian neonates (mean age, 42.4 h; SD, 23.0 h; range, 8.1-
152 h) who passed or failed 1000-Hz tympanometry and DPOAEs were included in the study. </
5
6
2 kHz and the 2nd peak at 5-8 kHz, while normative admittance phase data showed 2 peaks at 0
7
8
9
10 <p> Objectives: The objective of this study was to describe wideba
0.16 between 0.5 and 1.5 kHz) than for the cholesteatoma group (0.03-0.11 between 0.6 and 3kl
0.16 between 0.5 and 1.5 kHz) than for the cholesteatoma group (0.03-0.11 between 0.6 and 3kl
11
12
13
300 daPa. WBT is a suitable test to evaluate middle-ear function in children, but there is a
14
15
16

	DataSubmitterName
1	Susan Voss
2	Sreedevi Aithal
3	Sreedevi Aithal
4	Sreedevi Aithal
5	Sreedevi Aithal
6	Sreedevi Aithal
7	Sreedevi Aithal
8	Sreedevi Aithal
9	Sreedevi Aithal
10	Sreedevi Aithal
11	Sreedevi Aithal
12	Hammam AlMakadma
13	Cerys Downing
14	Douglas Keefe
15	M. Patrick Feeney; Douglas H. Keefe
16	Steve Neely

	DataSubmitterEmail	DateSubmitted
1	svoss@smith.edu	24-Aug-2016
2	Sreedevi.aithal@health.qld.gov.au	2-Jan-2023
3	Sreedevi.aithal@health.qld.gov.au	2-Jan-2023
4	Sreedevi.aithal@health.qld.gov.au	26-Feb-2023
5	Sreedevi.Aithal@health.qld.gov.au	26-Feb-2023
6	Sreedevi.Aithal@health.qld.gov.au	17-Apr-2023
7	Sreedevi.Aithal@health.qld.gov.au	26-Feb-2023
8	Sreedevi.Aithal@health.qld.gov.au	17-Apr-2023
9	Sreedevi.Aithal@health.qld.gov.au	24-Apr-2023
10	Sreedevi.Aithal@health.qld.gov.au	17-Apr-2023
11	Sreedevi.Aithal@health.qld.gov.au	20-Apr-2023
12	ha.almakadma@louisville.edu	20-Apr-2023
13	cerys.downing@uq.edu.au	13-Sep-22
14	Douglas.Keefe@boystown.org	23-Jun-2021
15	Patrick.Feeney@va.gov; Douglas.Keefe@boystown.org	7-June-2018
16	Stephen.Neely@boystown.org	18-Jun-2019

1	
2	
3	
4	
5	
6	
7	
8	

```
9
10
11
12
13
14
15 Database includes measurements on 32 subjects, most with left and right ears and mos
16
```

So, the 14 through 16 ones are Ellison_2012, Feeney_2017, and Groon_2015. Now we want to get just those, and not the others. You can play around with this, but basically, the first argument for limit tells us where to start, and the second number is how many to display. We want 14-16, so we start at 13 (because it starts at 0, not 1) and tell it to show 3. These match what we said we wanted from looking at the first 16.

```
dbGetQuery(con, "SELECT *
                 FROM PI_Info
                 LIMIT 13, 3")
```

```
    Identifier Year
1 Ellison_2012 2012
2  Feeney_2017 2017
3   Groon_2015 2015

1                                           John C. Ellison, Michael Gorga, Edward Cohn
2 M. Patrick Feeney,  Douglas H. Keefe, Lisa L. Hunter,  Denis F. Fitzpatrick, Angela C
3                                             Katherine A. Groon, Daniel M. Rasetshwane
  AuthorsShortList
1   Ellison et al.
2    Feeney et al.
3     Groon et al.

1                                                                        Wideband Acoustic Tra
2 Normative wideband reflectance, equivalent admittance at the tympanic membrane, and ac
3                                                                                Air-le
           Journal
1 The Laryngoscope
2         Ear Hear
3         Ear Hear

1
2
```

```
3 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4272628/https://journals.lww.com/ear-hea

1
2 "<p> <strong> Objectives: </strong> Wideband acoustic immittance (WAI) measures such
3
                   DataSubmitterName
1                      Douglas Keefe
2 M. Patrick Feeney; Douglas H. Keefe
3                        Steve Neely
                               DataSubmitterEmail DateSubmitted
1                       Douglas.Keefe@boystown.org   23-Jun-2021
2 Patrick.Feeney@va.gov; Douglas.Keefe@boystown.org   7-June-2018
3                       Stephen.Neely@boystown.org   18-Jun-2019

1
2 Database includes measurements on 32 subjects, most with left and right ears and most
3
```

part e - How many observations are in the `PI_Info` table?

Solution:

```
dbGetQuery(con, "SELECT COUNT(*)
                 FROM PI_Info")
```

```
  COUNT(*)
1       41
```

```
# name the returned value "num_obs"
dbGetQuery(con, "SELECT COUNT(*) as num_obs
                 FROM PI_Info")
```

```
  num_obs
1      41
```

There appear to be a total of 41 observations in the PI_Info table.

part f - Explore the `Measurements` and `Subjects` tables. What type of information is in each table? How many observations are in each table?

Solution:

There are 4,925,346 rows in the `Measurements` table and the variables include the paper identifier, the subject number from that study, the session number for that subject, and various ear measurements. There are 9902 rows in the `Subject` table and variables include the paper and subject identifier, as well as summary/demographic information on the subjects (age, sex, race, etc.).

```
dbGetQuery(con, "EXPLAIN Measurements")
```

```
            Field        Type Null Key Default Extra
1      Identifier varchar(50)   NO PRI    <NA>
2   SubjectNumber         int   NO PRI    <NA>
3         Session         int   NO PRI    <NA>
4             Ear varchar(50)   NO PRI
5      Instrument varchar(50)   NO PRI
6             Age       float  YES        <NA>
7     AgeCategory varchar(50)  YES        <NA>
8       EarStatus varchar(50)  YES        <NA>
9             TPP       float  YES        <NA>
10      AreaCanal       float  YES        <NA>
11  PressureCanal       float   NO PRI       0
12 SweepDirection varchar(50)   NO PRI
13      Frequency       float   NO PRI       0
14     Absorbance       float  YES        <NA>
15           Zmag       float  YES        <NA>
16           Zang       float  YES        <NA>
```

```
dbGetQuery(con, "SELECT *
                 FROM Measurements
                 LIMIT 0, 5")
```

```
  Identifier SubjectNumber Session  Ear Instrument Age AgeCategory EarStatus
1  Abur_2014             1       1 Left     HearID  20       Adult    Normal
2  Abur_2014             1       1 Left     HearID  20       Adult    Normal
3  Abur_2014             1       1 Left     HearID  20       Adult    Normal
4  Abur_2014             1       1 Left     HearID  20       Adult    Normal
5  Abur_2014             1       1 Left     HearID  20       Adult    Normal
  TPP AreaCanal PressureCanal SweepDirection Frequency Absorbance      Zmag
1  -5 0.0000442             0        Ambient   210.938  0.0333379 113780000
2  -5 0.0000442             0        Ambient   234.375  0.0315705 103585000
```

```
3  -5 0.0000442              0        Ambient   257.812  0.0405751  92951700
4  -5 0.0000442              0        Ambient   281.250  0.0438399  86058000
5  -5 0.0000442              0        Ambient   304.688  0.0486400  79492800
       Zang
1 -0.233504
2 -0.235778
3 -0.233482
4 -0.233421
5 -0.232931
```

```
dbGetQuery(con, "SELECT COUNT(*)
                 FROM Measurements")
```

```
  COUNT(*)
1  4925346
```

```
dbGetQuery(con, "EXPLAIN Subjects")
```

```
                             Field        Type Null Key Default Extra
1                       Identifier varchar(50)   NO PRI    <NA>
2                    SubjectNumber         int   NO PRI    <NA>
3                     SessionTotal         int   NO        <NA>
4              AgeFirstMeasurement       float  YES        <NA>
5      AgeCategoryFirstMeasurement varchar(50)  YES        <NA>
6                              Sex varchar(50)   NO        <NA>
7                             Race varchar(50)   NO        <NA>
8                        Ethnicity varchar(50)   NO        <NA>
9    LeftEarStatusFirstMeasurement varchar(50)   NO        <NA>
10 RightEarStatusFirstMeasurement varchar(50)   NO        <NA>
11                    SubjectNotes        text  YES        <NA>
```

```
dbGetQuery(con, "SELECT *
                 FROM Subjects
                 LIMIT 0, 5")
```

```
  Identifier SubjectNumber SessionTotal AgeFirstMeasurement
1  Abur_2014             1            7                  20
2  Abur_2014             3            8                  19
3  Abur_2014             4            7                  21
```

```
4  Abur_2014             6            8                 21
5  Abur_2014             7            5                 20
  AgeCategoryFirstMeasurement    Sex    Race Ethnicity
1                       Adult Female Unknown   Unknown
2                       Adult Female Unknown   Unknown
3                       Adult Female Unknown   Unknown
4                       Adult Female Unknown   Unknown
5                       Adult Female Unknown   Unknown
  LeftEarStatusFirstMeasurement RightEarStatusFirstMeasurement
1                        Normal                         Normal
2                        Normal                         Normal
3                        Normal                         Normal
4                        Normal                         Normal
5                        Normal                         Normal
                             SubjectNotes
1
2 Session 5 not included do to acoustic leak
3
4
5
```

```
dbGetQuery(con, "SELECT COUNT(*)
                 FROM Subjects")
```

```
  COUNT(*)
1     9902
```

2 - SQL tables vs R data frames

Keep in mind that we are connecting to a massive database held on a server, and although we are coding in this R environment, the computations are being done on the MySQL server (we send messages to the server to tell it what to do; the server does the heavy lifting and sends us back the results).

As with R objects, it can be useful to save SQL objects before we continue working with them. The `tbl()` function allows us to save a SQL table on the server, which also shows up as a `tbl_sql` object in our R environment.

```
# Assign object as SQL table
PI_Info_sql <- tbl(con, "PI_Info")
class(PI_Info_sql)
```

```
[1] "tbl_MySQLConnection" "tbl_dbi"             "tbl_sql"
[4] "tbl_lazy"            "tbl"
```

part a - The functions in **dplyr** are designed to translate automatically to SQL commands, but this is not true for other packages we've worked with. With this in mind, do you expect either block of code below to work? Why or why not?

Solution:

```
# Block 1
PI_Info_sql %>%
  filter(Year == 2010) %>%
  select(Identifier, Year, AuthorsShortList)

# Block 2
PI_Info_sql %>%
 separate(Identifier, into = c("Author", "Year"),
          sep = "_", remove = FALSE) %>%
 select(Identifier, PI_Year, PI)
```

The code in Block 1 should work because the functions are all **dplyr** functions. The code in Block 2 should not work because `separate()` is a function from the **tidyr** package, and this function does not get translated to SQL.

part b - SQL can be very helpful and efficient for querying huge datasets and relational databases, but its analytic capabilities are limited. When analyzing data or creating figures, we may want to convert SQL queries or tables into local R data frames, which we can then work with in all the ways we have learned this semester. We can do so using the `collect()` function:

```r
PI_Info_df <- PI_Info_sql %>%
  collect()

class(PI_Info_sql)
class(PI_Info_df)

# Doesn't work (SQL table)
PI_Info_sql %>%
   separate(Identifier, into = c("Author", "Year"),
            sep = "_", remove = FALSE) %>%
   select(Identifier, Author, Year)

# Works (R dataframe)
PI_Info_df %>%
   separate(Identifier, into = c("Author", "Year"),
            sep = "_", remove = FALSE) %>%
   select(Identifier, Author, Year)
```

3 - SQL code chunks

Syntax highlighting is an incredibly useful tool for quickly identifying errors or typos as you code. However, the `dbGetQuery()` command has us place the SQL code within quotation marks, which makes all the SQL syntax the same color. If you'd prefer to see the color-coding, you can write SQL directly in a SQL code chunk. In addition to specifying the language of the code chunk (`sql`), you need to specify the server connection within the code chunk option using `connection = ...`. For the remainder of the lab, we'll use SQL code chunks whenever we want to query the Smith WAI database.

> part a - The R code chunk below shows a query using `dbGetQuery()` in R. The second code chunk shows the exact same query but in a SQL code chunk. Can you identify what the code is doing?

Solution:

R code chunk

```
dbGetQuery(con, "SELECT SessionTotal, COUNT(SessionTotal) as n,
                        AVG(Sex = 'Female') as prop_fem,
                        AVG(AgeFirstMeasurement) as avg_age
                 FROM Subjects
                 GROUP BY SessionTotal")
```

```
   SessionTotal    n prop_fem   avg_age
1             7    5   0.8000 16.800000
2             8    4   0.7500 14.500000
3             5   50   0.6600  8.734427
4             1 9183   0.4575 16.593362
5             6   11   0.2727  4.818182
6            11    9   0.5556  0.000000
7            13    6   0.1667  0.000000
8            12   13   0.6154  0.000000
9            14    7   0.7143  0.000000
10            9    2   0.5000  0.000000
11           15    1   1.0000  0.000000
12           17    1   0.0000  0.000000
13            2  403   0.5136  3.700513
14            4   68   0.4706  1.887987
15            3  139   0.5540  1.240666
```

SQL code chunk

```sql
SELECT  SessionTotal, COUNT(SessionTotal) as n,
        AVG(Sex = 'Female') as prop_fem,
        AVG(AgeFirstMeasurement) as avg_age
FROM Subjects
GROUP BY SessionTotal
```

Table 1: Displaying records 1 - 10

SessionTotal	n	prop_fem	avg_age
7	5	0.8000	16.800000
8	4	0.7500	14.500000
5	50	0.6600	8.734427
1	9183	0.4575	16.593362
6	11	0.2727	4.818182
11	9	0.5556	0.000000
13	6	0.1667	0.000000
12	13	0.6154	0.000000
14	7	0.7143	0.000000
9	2	0.5000	0.000000

Both commands are doing the same thing. The code is summarizing by total number of sessions. Participants each had between 1 and 8 sessions, and the below code gets the total number of participants that had 1-8 sessions as well as the proportion of participants that were female and the average age of the participants that were seen for those total sessions.

Note: If you get to this part, and you don't see the usual gear, down arrow and run arrow options in the SQL chunk, it's because it wants to put the output inline and you probably have it set to the console. If you click the gear window next to "Knit", you can change it to say "Chunk Output Inline". The usual run options should appear then if you make a change in the chunk and put it back. For example, I deleted the last ' and then added it back.

If you've kept the default all semester, you probably won't have issues with this. Remember you can change it back later.

> part b - The textbook tells us that the SQL equivalent of **dplyr**'s `filter()` is `WHERE`, but there is a similar SQL command called `HAVING`. When should you use `WHERE` vs. `HAVING`? You can use the code chunks below to help explain.

Solution:

```sql
SELECT  SessionTotal, COUNT(SessionTotal) as n,
        AVG(Sex = 'Female') as prop_fem,
```

```
        AVG(AgeFirstMeasurement) as avg_age
FROM Subjects
WHERE AgeFirstMeasurement < 25
GROUP BY SessionTotal
```

Table 2: Displaying records 1 - 10

SessionTotal	n	prop_fem	avg_age
7	4	0.7500	14.750000
8	4	0.7500	14.500000
5	45	0.6444	5.793808
1	5488	0.5002	7.567106
6	9	0.2222	0.000000
11	9	0.5556	0.000000
13	6	0.1667	0.000000
12	13	0.6154	0.000000
14	7	0.7143	0.000000
9	2	0.5000	0.000000

```
SELECT  SessionTotal, COUNT(SessionTotal) as n,
        AVG(Sex = 'Female') as prop_fem,
        AVG(AgeFirstMeasurement) as avg_age
FROM Subjects
GROUP BY SessionTotal
HAVING avg_age < 25
```

Table 3: Displaying records 1 - 10

SessionTotal	n	prop_fem	avg_age
7	5	0.8000	16.800000
8	4	0.7500	14.500000
5	50	0.6600	8.734427
1	9183	0.4575	16.593362
6	11	0.2727	4.818182
11	9	0.5556	0.000000
13	6	0.1667	0.000000
12	13	0.6154	0.000000
14	7	0.7143	0.000000
9	2	0.5000	0.000000

Both WHERE and HAVING are like SQL's version of the filter command, and can be used to specify which observations to include or exclude. However, they are used in different places within the commands. WHERE is specifically used on an SQL table that has not been grouped (prior to a GROUP BY statement) whereas HAVING is used on table that has been grouped (after a GROUP BY statement). For instance, the first code chunk below includes only subjects less than 25 prior to grouping by session total. In contrast, the second code chunk below groups by session total and then only keeps those groups where the average age is less than 25.

Doing a WHERE first can be more effective, if appropriate, as you are removing observations before performing some other operation. The text recommends using WHERE whenever possible, and relying less on HAVING, assuming it is appropriate (in the next part, HAVING is more appropriate).

> part c - Re-do the above query but include only those rows that have at least 10 subjects contributing (hint: update the `HAVING` line only).

Solution:

Here, updating HAVING makes more sense because we need the observations grouped_by session total before this would make sense to view.

```
SELECT SessionTotal, COUNT(SessionTotal) as n,
       AVG(Sex = 'Female') as prop_fem,
       AVG(AgeFirstMeasurement) as avg_age
FROM Subjects
GROUP BY SessionTotal
HAVING n > 10
```

Table 4: 7 records

SessionTotal	n	prop_fem	avg_age
5	50	0.6600	8.734427
1	9183	0.4575	16.593362
6	11	0.2727	4.818182
12	13	0.6154	0.000000
2	403	0.5136	3.700513
4	68	0.4706	1.887987
3	139	0.5540	1.240665

4 - Visualizations

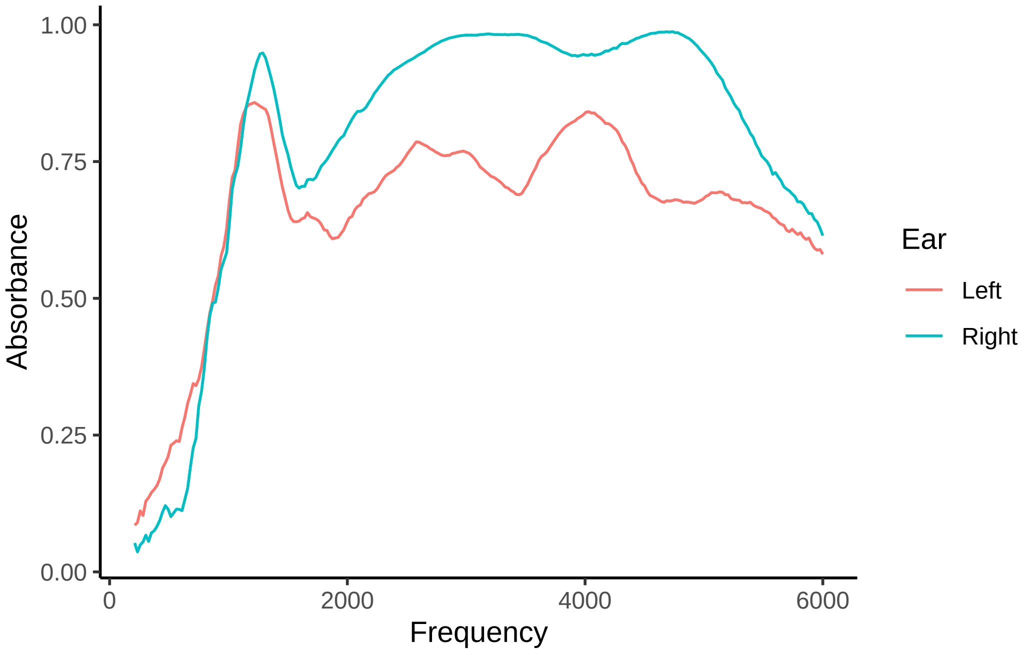
Create a figure that displays `Frequency` on the x-axis and `Absorbance` on the y-axis (both from the `Measurements` table), colored by `Ear` (left or right), for subject #3 from the Rosowski 2012 study. We will walk through three ways of doing this.

> part a - Approach 1: Use SQL code to query the appropriate table(s) (up to the point of creating the figure), and output the queried table as a dataframe by specifying the additional code chunk option `output.var = "your_desired_table_name"`. This will create an R data frame in your local environment called `your_desired_table_name`. Then switch to an R code chunk and use the outputted data frame to create the visualization of interest.

Solution:

```
SELECT Frequency, Absorbance, Ear
FROM Measurements
WHERE Identifier = "Rosowski_2012" AND SubjectNumber = 3
```

```
ggplot(data = rosowski3_sql,
       aes(x = Frequency, y = Absorbance, color = Ear)) +
  geom_line() +
  labs(x = "Frequency", y = "Absorbance")
```



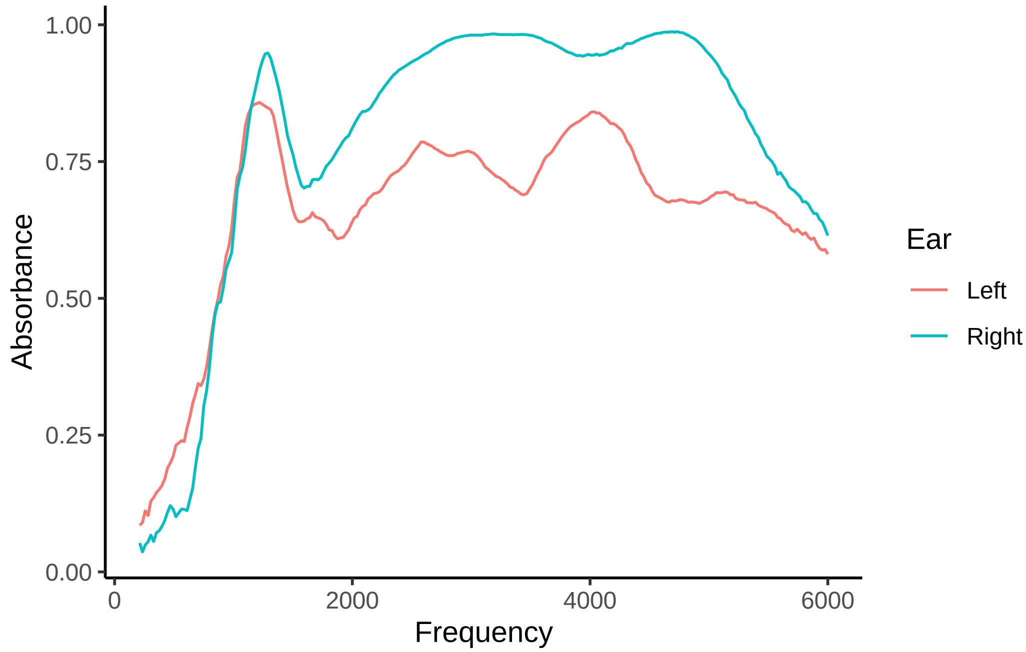
part b - Approach 2: In an R code chunk, query the table with the same code from part a but within `dbGetQuery()`, and pipe the queried table to `collect()` to convert the table to an R data frame. Then create the visualization of interest.

Solution:

```
# you can also save the query separately

rosowski3_r <- dbGetQuery(con, "SELECT Frequency, Absorbance, Ear
                                FROM Measurements
                                WHERE Identifier = \"Rosowski_2012\" AND SubjectNumber = 3
  collect()

ggplot(data = rosowski3_r,
       aes(x = Frequency, y = Absorbance, color = Ear)) +
  geom_line() +
  labs(x = "Frequency", y = "Absorbance")
```

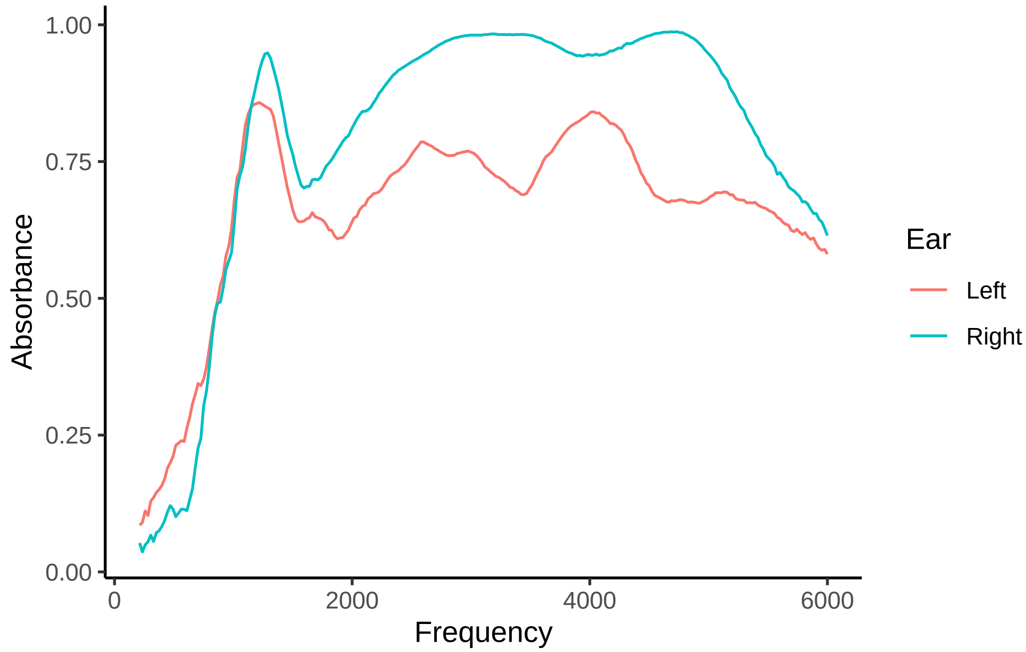


part c - Approach 3: In an R code chunk, query the entire `Measurements` table using `tbl()`, use **dplyr** verbs instead of SQL commands to subset the data as desired, then convert the output to an R data frame by piping to `collect()`, then make the visual. Why do we want to use **dplyr** verbs *before* instead of *after* converting to an R data frame?

Solution:

```
# Option 3: Relying more heavily on **dplyr**
rosowski3_dplyr <- tbl(con, "Measurements") %>%
  filter(Identifier == "Rosowski_2012", SubjectNumber == 3) %>%
  collect()

ggplot(data = rosowski3_dplyr,
       aes(x = Frequency, y = Absorbance, color = Ear)) +
  geom_line() +
  labs(x = "Frequency", y = "Absorbance")
```



part d - Which approach do you prefer? Why?

Answers will vary depending on your preference. All 3 generate the data set for us to work with in R, which we can feed to ggplot2.