Question 1

Note: You need to be either on the campus or connected to UCD vpn in order to connect the postgres server on alan.ucdavis.edu.

```
library(DBI)
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────────────────────
──────── tidyverse 1.3.0 ──
```

```
## ✓ ggplot2 3.3.0     ✓ purrr   0.3.3
## ✓ tibble  3.0.0     ✓ dplyr   0.8.5
## ✓ tidyr   1.0.2     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.5.0
```

```
## ── Conflicts ────────────────────────────────────────────────────────────────
── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
chinook <- dbConnect(
  RPostgres::Postgres(),
  dbname = "chinook",
  user = "psqluser", password = "secret", host = "alan.ucdavis.edu"
)
```

The Chinook data model represents a digital media store, including tables for artists, albums, media tracks, invoices and customers.

- Media-related data was created using real data from an Apple iTunes library.
- Customer and employee information was created using fictitious names and addresses that can be located on Google maps, and other well formatted data (phone, fax, email, etc.)
- Sales information was auto generated using random data for a four year period.
- There are 11 tables in the chinook sample database.
    - `Employee` table stores employees data such as employee id, last name, first name, etc. It also has a field named ReportsTo to specify who reports to whom.
    - `Customer` table stores customers data.
    - `Invoice` & `InvoiceLine` tables: these two tables store invoice data. The invoices table stores invoice header data and the invoice line table stores the invoice line items data.
    - `Artist` table stores artists data. It is a simple table that contains only the artist id and name.
    - `Albums` table stores data about a list of tracks. Each album belongs to one artist. However, one artist may have multiple albums.
    - `MediaType` table stores media types such as MPEG audio and AAC audio file.
    - `Genre` table stores music types such as rock, jazz, metal, etc.
    - `Track` table store the data of songs. Each track belongs to one album.

- `Playlist` & `PlaylistTrack` tables: playlists table store data about playlists. Each playlist contains a list of tracks. Each track may belong to multiple playlists. The relationship between the playlists table and tracks table is many-to-many. The playlist track table is used to reflect this relationship.

See for example https://docs.yugabyte.com/images/sample-data/chinook/chinook-er-diagram.png (https://docs.yugabyte.com/images/sample-data/chinook/chinook-er-diagram.png) for a database diagram.

For each of the followings, use both dplyr and sql approaches to get the answer.

```
chinook %>% dbListTables()
```

```
##  [1] "Customer"      "InvoiceLine"   "Artist"        "Playlist"
##  [5] "Employee"      "Album"         "Invoice"       "MediaType"
##  [9] "PlaylistTrack" "Track"         "Genre"
```

(a) What is the title of the album with AlbumId 31?

```
chinook %>%
  tbl("Album") %>%
  filter(AlbumId == 31) %>%
  select("AlbumId", "Title")
```

```
## # Source:   lazy query [?? x 2]
## # Database: postgres [psqluser@alan.ucdavis.edu:5432/chinook]
##   AlbumId Title
##     <int> <chr>
## 1      31 Bongo Fury
```

```
SELECT "AlbumId", "Title"
FROM "Album"
WHERE ("AlbumId" = 31);
```

1 records

AlbumId	Title
31	Bongo Fury

The title of the album with AlbumID 31 is “Bongo Fury”.

(b) List all the albums by artists with the word ‘black’ in their name.

```
#chinook %>% dbListTables()
inner_join(
  tbl(chinook, "Album"),
  tbl(chinook, "Artist"),
  by = "ArtistId") %>%
  filter(grepl("black", Name, ignore.case = TRUE)) %>%
  select(Title, Name)
```

```
## # Source:   lazy query [?? x 2]
## # Database: postgres [psqluser@alan.ucdavis.edu:5432/chinook]
##   Title                                  Name
##   <chr>                                  <chr>
## 1 Alcohol Fueled Brewtality Live! [Disc 1] Black Label Society
## 2 Alcohol Fueled Brewtality Live! [Disc 2] Black Label Society
## 3 Black Sabbath                          Black Sabbath
## 4 Black Sabbath Vol. 4 (Remaster)        Black Sabbath
## 5 Live [Disc 1]                          The Black Crowes
## 6 Live [Disc 2]                          The Black Crowes
```

```
SELECT "Title", "Name"
FROM "Album" a
INNER JOIN "Artist" b
ON (a."ArtistId" = b."ArtistId")
WHERE (("Name") ILIKE ('%black%'))
```

6 records

Title	Name
Alcohol Fueled Brewtality Live! [Disc 1]	Black Label Society
Alcohol Fueled Brewtality Live! [Disc 2]	Black Label Society
Black Sabbath	Black Sabbath
Black Sabbath Vol. 4 (Remaster)	Black Sabbath
Live [Disc 1]	The Black Crowes
Live [Disc 2]	The Black Crowes

List all the albums by artists with the word ‘black’ in their name.

The albums by artist who have the word ‘black’ in their names are as follows: “Alcohol Fueled Brewtality Live! [Disc 1]” (Black Label Society), “Alcohol Fueled Brewtality Live! [Disc 2]” (Black Label Society), “Black Sabbath” (Black Sabbath), “Black Sabbath Vol. 4 (Remaster)” (Black Sabbath), “Live [Disc 1]” (The Black Crowes), “Live [Disc 2]” (The Black Crowes).

(c) Find the name and length (in seconds) of all tracks that have both length between 30 and 40 seconds, and genre Latin.

```
# chinook %>% dbListTables()
# chinook %>% dbListFields("Track")  # column names

left_join(
  tbl(chinook, "Track"),
  tbl(chinook, "Genre"),
  by = "GenreId") %>%
  mutate(Seconds = Milliseconds / 1000) %>%
  filter(Name.y == "Latin" & Seconds > 30  & Seconds < 40) %>%
  rename(Artist = Name.x, Genre = Name.y) %>%
  select(Artist, Seconds, Genre)
```

```
## # Source:   lazy query [?? x 3]
## # Database: postgres [psqluser@alan.ucdavis.edu:5432/chinook]
##   Artist       Seconds Genre
##   <chr>          <dbl> <chr>
## 1 Deixa Entrar    33.6 Latin
```

```
SELECT a."Name" as "Title", a."Milliseconds"/1000 as "Seconds", b."Name" as "Genre"
FROM "Track" a
LEFT JOIN "Genre" b
ON (a."GenreId" = b."GenreId")
WHERE b."Name" = 'Latin'
  AND a."Milliseconds" BETWEEN 30000 AND 40000;
```

1 records

Title	Seconds	Genre
Deixa Entrar	33	Latin

There is only one track with genre “Latin” and between 30 and 40 seconds. The track is “Deixa Entrar”.

(d) List each country and the number of customers in that country. (You only need to include countries that have customers.)

```
# Understanding the table structure/columns
chinook %>% dbListTables()
```

```
##  [1] "Customer"      "InvoiceLine"   "Artist"        "Playlist"
##  [5] "Employee"      "Album"         "Invoice"       "MediaType"
##  [9] "PlaylistTrack" "Track"         "Genre"
```

```
chinook %>% dbListFields("Customer")
```

```
##  [1] "CustomerId"   "FirstName"    "LastName"     "Company"      "Address"
##  [6] "City"         "State"        "Country"      "PostalCode"   "Phone"
## [11] "Fax"          "Email"        "SupportRepId"
```

```
# Grouping by Country and finding the number of customers in that country
chinook %>%
  tbl("Customer") %>%
  group_by(Country) %>%
  summarise(n=n()) %>%
  collect()
```

```
## # A tibble: 11 x 2
##    Country        n
##    <chr>          <int64>
##  1 France          5
##  2 Netherlands     1
##  3 Australia       1
##  4 Chile           1
##  5 USA            13
##  6 Ireland         1
##  7 Canada          7
##  8 United Kingdom  3
##  9 Italy           1
## 10 Sweden          1
## 11 India           2
```

```
SELECT "Country", COUNT(*) AS "n"
FROM "Customer"
GROUP BY "Country"
```

Displaying records 1 - 10

Country	n
France	5
Netherlands	1
Australia	1
Chile	1
USA	13
Ireland	1
Canada	7
United Kingdom	3
Italy	1
Sweden	1

(e) Find the artist (or several artists) with the largest number of countries where the listeners are from. To certain extent, think of the most culturally diverse artists.

```
# Understanding the columns in each table and how they connect to one another
chinook %>% dbListTables()
```

```
##  [1] "Customer"      "InvoiceLine"   "Artist"        "Playlist"
##  [5] "Employee"      "Album"         "Invoice"       "MediaType"
##  [9] "PlaylistTrack" "Track"         "Genre"
```

```
chinook %>% dbListFields("InvoiceLine")
```

```
## [1] "InvoiceLineId" "InvoiceId"     "TrackId"       "UnitPrice"
## [5] "Quantity"
```

```
chinook %>% dbListFields("Artist")
```

```
## [1] "ArtistId" "Name"
```

```
chinook %>% dbListFields("Album")
```

```
## [1] "AlbumId"  "Title"    "ArtistId"
```

```
# Series of left_join's to get one large dataframe containing artist and country
# then summarizing to see who has the most unique countries.
left_join(
  tbl(chinook, "Artist"),
  left_join(
    tbl(chinook, "Album"),
    left_join(
      tbl(chinook, "Track"),
      left_join(
        tbl(chinook, "InvoiceLine"),
        left_join(
        tbl(chinook, "Invoice"),
        tbl(chinook, "Customer"),
        by = "CustomerId"
        ),
        by = "InvoiceId"),
      by = "TrackId"),
    by = "AlbumId"),
  by = "ArtistId"
) %>%
  group_by(Name.x) %>%
  summarise(Artist = Name.x,
            n_country = n_distinct(BillingCountry)) %>%
  arrange(desc(n_country)) %>%
  select(Artist, n_country) %>%
  head()
```

```
## # Source:     lazy query [?? x 2]
## # Database:   postgres [psqluser@alan.ucdavis.edu:5432/chinook]
## # Ordered by: desc(n_country)
##   Artist                      n_country
##   <chr>                       <int64>
## 1 Iron Maiden                 8
## 2 Creedence Clearwater Revival 7
## 3 Led Zeppelin                7
## 4 U2                          7
## 5 Metallica                   6
## 6 Ozzy Osbourne               5
```

```
-- Series of LEFT JOIN's to get one large table connecting artist to country
-- Remove null artist values
-- descending order to see which artists have the most unique countries.
SELECT *
FROM (SELECT f."Name" as "Artist", COUNT(DISTINCT a."Country") as "n_country"
FROM "Customer" a
LEFT JOIN "Invoice" b ON (a."CustomerId" = b."CustomerId")
LEFT JOIN "InvoiceLine" c ON (b."InvoiceId" = c."InvoiceId")
LEFT JOIN "Track" d ON (c."TrackId" = d."TrackId")
LEFT JOIN "Album" e ON (d."AlbumId" = e."AlbumId")
LEFT JOIN "Artist" f ON (e."ArtistId" = f."ArtistId")
WHERE f."Name" IS NOT NULL
GROUP BY f."Name") complete_table
ORDER BY "n_country" DESC
LIMIT 6
```

6 records

Artist	n_country
Iron Maiden	8
U2	7
Led Zeppelin	7
Creedence Clearwater Revival	7
Metallica	6
Faith No More	5

The artist most culturally diverse artist is Iron Maiden.

Question 2

```
library(DBI)
library(tidyverse)
```

In this question, we are going to learn how to create a sql database on google cloud platform.

- First, go to canvas and click the Google Cloud Student Coupon Retrieval Link. You will be given $50 dollars to use any google cloud services.
- Then, go to https://console.cloud.google.com/ (https://console.cloud.google.com/) and create a new project by clicking on the [Select a project] button. You project should be under the organization `UCDAVIS.EDU`.
- Click on [Cloud SQL] icon on the startup page or the [SQL] tab in the menu.
- In the SQL tab, choose PostgreSQL and create an new instance.
- Click on “Show configuration options”. In “Machine types and storage”, drag the slidebar to the left to choose “1 shared vCPU” and memory 0.6G. (It is the cheapest option for testing purpose)
- Choose a password for the server, I suggest using the generated password. (And keep it somewhere else)
- It may take around 5 minutes for the server to setup. Copy the public ip address once it is done.
- Then open the instance and go the [Databases] tab; create a new database, call it “demo”.
- Then go to [Connections], click [Add network] and type “0.0.0.0/0” in Network. It will make sure that your server is visible to the internet.

public ip: 35.233.226.110

Of course you don’t want to put the password in your assignment. Password should be stored in a file `.Renviron`. The file could be created by using the following command.

```
usethis::edit_r_environ("project")
```

Put down your password in the file, (replace XXXXXXXXXXXXXXXXX with your password)

```
DATABASEPW=XXXXXXXXXXXXXXXXX
```

One your are done, run the following line

```
readRenviron(".Renviron")
```

Then your password could be retrived by `Sys.getenv("DATABASEPW")`.

The `.Renviron` won’t be pushed to the git repo because the file was specified in `.gitignore`.

Connect to the database

```
library(DBI)
host <- "35.233.226.110"  # replace it with your server ip
mydb <- dbConnect(
  RPostgres::Postgres(),
  dbname = "demo",
  user = "postgres", password = Sys.getenv("DATABASEPW"), host = host
)
```

Then we import the January part of `nycflights13::flights` to the database. (This chunk of code should only be execuuted once, so we have `eval = FALSE` to avoid running it again when knitting.)

```
# to save time, we only import January data
mydb %>% dbWriteTable(
  "flights",
  nycflights13::flights %>% filter(month == 1)
)
```

(a) Verify that we have a table `flights` in the database and count the number of rows.

```
SELECT COUNT(*) AS "n"
FROM "flights";
```

1 records

n
27004

(b) Use SQL to count the number of flights by destinations in January.

```
SELECT "dest", COUNT(*) AS "n"
FROM "flights"
GROUP BY "dest";
```

Displaying records 1 - 10

dest	n
CHS	91
MIA	981
CMH	265
GSP	57
LGB	52
PSP	4
SNA	56
DEN	563
HOU	146
IND	118

(c) Use SQL to count the average air time by carrier in January.

```
SELECT "carrier", AVG("air_time") AS "Average Airtime"
FROM "flights"
GROUP BY "carrier";
```

Displaying records 1 - 10

carrier	Average Airtime
VX	349.26752
AA	199.54332
F9	243.83051
B6	155.71901
DL	180.66347
UA	213.70218
WN	152.56447
US	90.63771
MQ	98.10123
EV	91.72603

Question 3

We use the Northwind database (northwind.sqlite) in the question. It provides you with a good database structure and sales data.

See https://www.zentut.com/sql-tutorial/sql-sample-database/ (https://www.zentut.com/sql-tutorial/sql-sample-database/) for a diagram.

Database tables The following explains each table in the Northwind database:

- Customers – stores customer master data
- Orders – stores transaction sale orders from customers
- OrderDetails – stores line items of sale orders
- Products – stores products master data
- Suppliers – stores suppliers master data
- Shippers – stores shippers master data
- Region – stores region master data
- Territories – store territories master data
- Employees – store employees master data
- EmployeeTerritories – store relationship between employee and territory.

Use SQL queries to answer the following questions.

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────────────────────
──────── tidyverse 1.3.0 ──
```

```
## ✓ ggplot2 3.3.0     ✓ purrr   0.3.3
## ✓ tibble  3.0.0     ✓ dplyr   0.8.5
## ✓ tidyr   1.0.2     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.5.0
```

```
## ── Conflicts ────────────────────────────────────────────────────────────
── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(DBI)
knitr::opts_chunk$set(warning = FALSE, message = FALSE)
northwind_lite <- dbConnect(RSQLite::SQLite(), dbname = "northwind.sqlite")
northwind_lite %>% dbListTables()
```

```
##  [1] "Category"             "Customer"             "CustomerCustomerDemo"
##  [4] "CustomerDemographic"  "Employee"             "EmployeeTerritory"
##  [7] "Order"                "OrderDetail"          "Product"
## [10] "ProductDetails_V"     "Region"               "Shipper"
## [13] "Supplier"             "Territory"
```

(a) Write a query to get Product name and quantity/unit.

```
SELECT `ProductName`, `QuantityPerUnit`
FROM `Product`
```

Displaying records 1 - 10

ProductName	QuantityPerUnit
Chai	10 boxes x 20 bags
Chang	24 - 12 oz bottles
Aniseed Syrup	12 - 550 ml bottles
Chef Anton’s Cajun Seasoning	48 - 6 oz jars
Chef Anton’s Gumbo Mix	36 boxes
Grandma’s Boysenberry Spread	12 - 8 oz jars
Uncle Bob’s Organic Dried Pears	12 - 1 lb pkgs.
Northwoods Cranberry Sauce	12 - 12 oz jars
Mishi Kobe Niku	18 - 500 g pkgs.
Ikura	12 - 200 ml jars

(a) Write a query to get discontinued Product list (Product ID and name).

```
SELECT `Id`, `ProductName`
FROM `Product`
WHERE (`Discontinued` = 1)
```

8 records

Id	ProductName
5	Chef Anton’s Gumbo Mix
9	Mishi Kobe Niku
17	Alice Mutton
24	Guaraná Fantástica
28	Rössle Sauerkraut
29	Thüringer Rostbratwurst
42	Singaporean Hokkien Fried Mee
53	Perth Pasties

(b) Write a query to get Product list (id, name, unit price) where current products cost less than $20.

```
SELECT `Id`, `ProductName`, `UnitPrice`
FROM `Product`
WHERE (`UnitPrice` < 20)
```

Displaying records 1 - 10

Id	ProductName	UnitPrice
1	Chai	18.00
2	Chang	19.00
3	Aniseed Syrup	10.00
13	Konbu	6.00
15	Genen Shouyu	15.50
16	Pavlova	17.45
19	Teatime Chocolate Biscuits	9.20
21	Sir Rodney’s Scones	10.00
23	Tunnbröd	9.00
24	Guaraná Fantástica	4.50

(c) Write a query to count current and discontinued products.

```
SELECT `Discontinued`, COUNT() as `n`
FROM `Product`
GROUP BY `Discontinued`
```

2 records

Discontinued	n
0	69
1	8

(d) Write a query to get most expense and least expensive Product list (name and unit price).

```
-- Most expensive
SELECT `ProductName`, MAX(`UnitPrice`) AS `UnitPrice`
FROM `Product`
UNION
--Lease expensive
SELECT `ProductName`, MIN(`UnitPrice`)
FROM `Product`
```

2 records

ProductName	UnitPrice
Côte de Blaye	263.5
Geitost	2.5

(e) Write a query to get Product list (name, unit price) of above average price

```
SELECT `ProductName`, `UnitPrice`
FROM `Product`
WHERE `UnitPrice`>
  (SELECT AVG(`UnitPrice`)
   FROM `Product`)
```

Displaying records 1 - 10

ProductName	UnitPrice
Uncle Bob’s Organic Dried Pears	30.00
Northwoods Cranberry Sauce	40.00
Mishi Kobe Niku	97.00
Ikura	31.00
Queso Manchego La Pastora	38.00
Alice Mutton	39.00
Carnarvon Tigers	62.50
Sir Rodney’s Marmalade	81.00
Gumbär Gummibärchen	31.23
Schoggi Schokolade	43.90