Mahitha Valluru

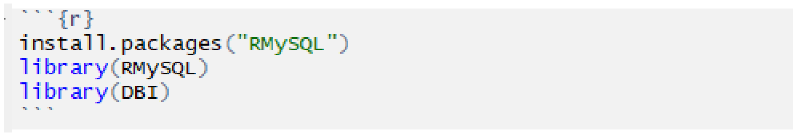
DS4100

Professor Schedlbauer

Assignment 9

To first do this assignment, there are several things that must be done. First, I will have to load any necessary libraries and set my working directory.

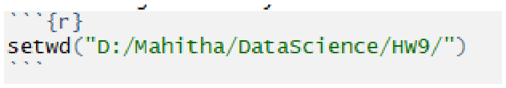
Here is a screenshot of the libraries I used:



~~~
```{r}
install.packages("RMySQL")
library(RMySQL)
library(DBI)
```
~~~

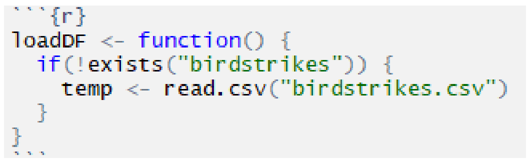
The two libraries are for loading the data frames into my MySQL database. Nothing much is needed for getting the data frames.

Now, I will have to set the working directory:



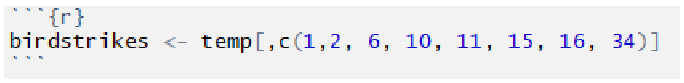
~~~
```{r}
setwd("D:/Mahitha/DataScience/HW9/")
```
~~~

Nothing too hard so far. Now to load the csv file:



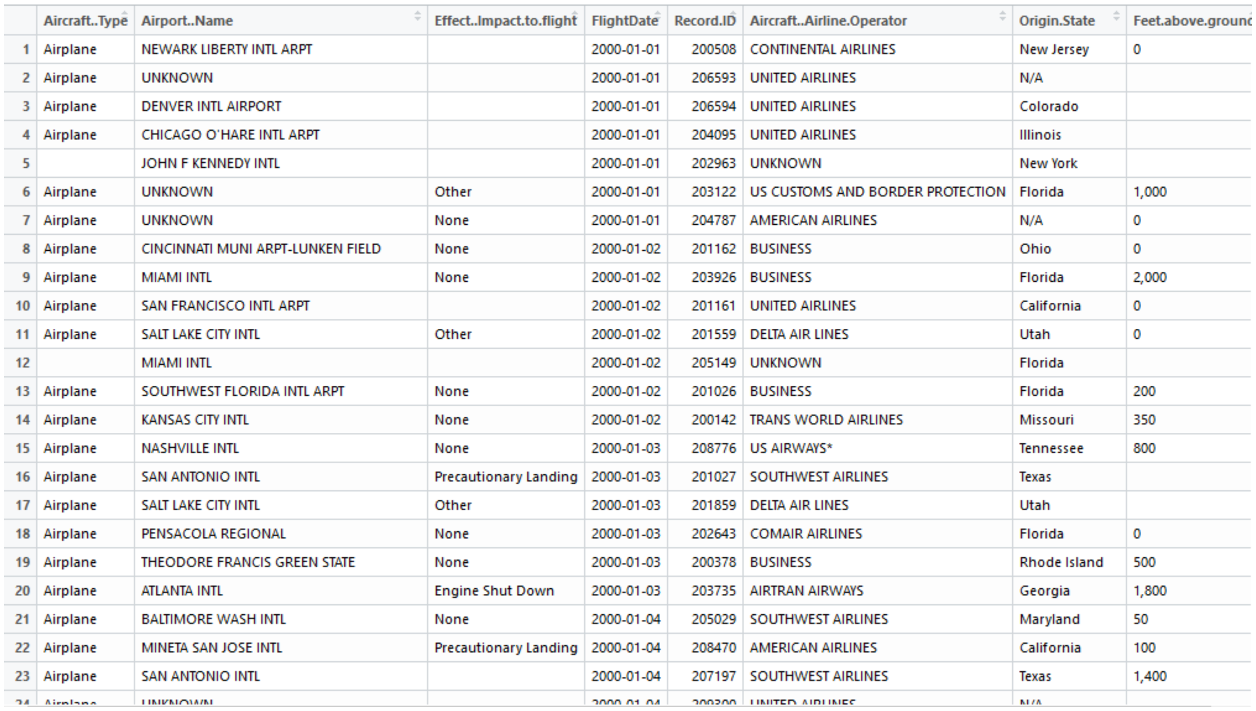
~~~
```{r}
loadDF <- function() {
  if(!exists("birdstrikes")) {
    temp <- read.csv("birdstrikes.csv")
  }
}
```
~~~

Notice that I labeled the data frame temp, because this has way too many columns that I will eventually will not need. This is also stated in the assignment, so I weeded out the unnecessary columns and created an actual data frame:



~~~
```{r}
birdstrikes <- temp[,c(1,2, 6, 10, 11, 15, 16, 34)]
```
~~~

This will result in a data frame that looks like this:

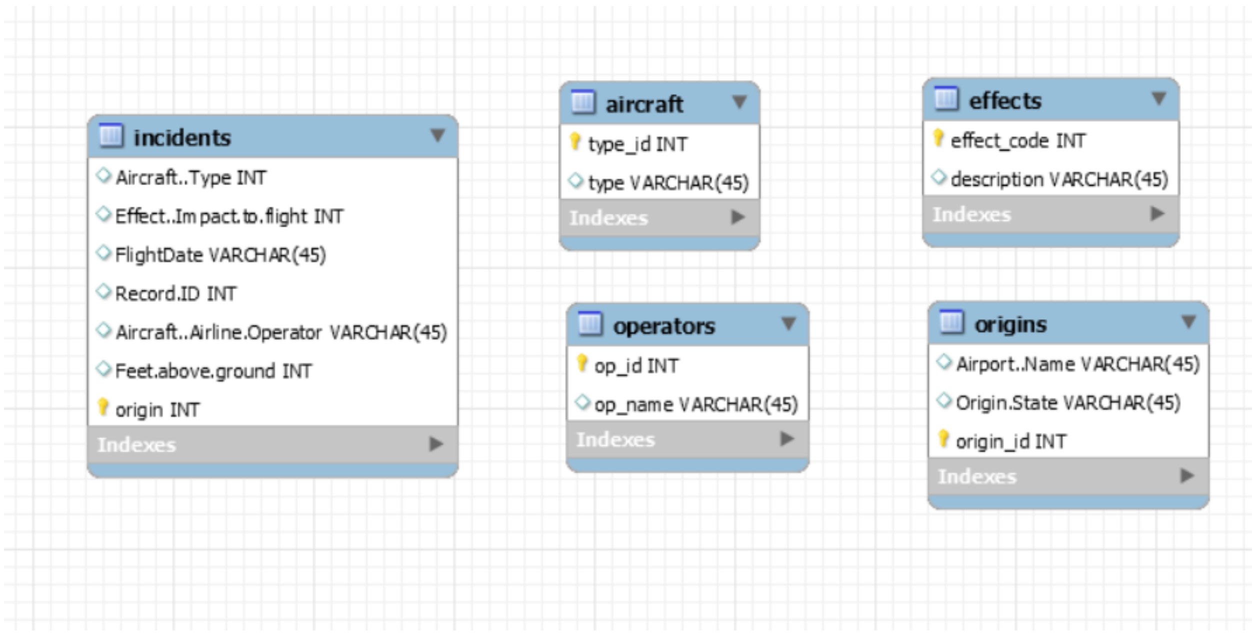


	Aircraft..Type	Airport..Name	Effect..Impact.to.flight	FlightDate	Record.ID	Aircraft..Airline.Operator	Origin.State	Feet.above.groun
1	Airplane	NEWARK LIBERTY INTL ARPT		2000-01-01	200508	CONTINENTAL AIRLINES	New Jersey	0
2	Airplane	UNKNOWN		2000-01-01	206593	UNITED AIRLINES	N/A	
3	Airplane	DENVER INTL AIRPORT		2000-01-01	206594	UNITED AIRLINES	Colorado	
4	Airplane	CHICAGO O'HARE INTL ARPT		2000-01-01	204095	UNITED AIRLINES	Illinois	
5		JOHN F KENNEDY INTL		2000-01-01	202963	UNKNOWN	New York	
6	Airplane	UNKNOWN	Other	2000-01-01	203122	US CUSTOMS AND BORDER PROTECTION	Florida	1,000
7	Airplane	UNKNOWN	None	2000-01-01	204787	AMERICAN AIRLINES	N/A	0
8	Airplane	CINCINNATI MUNI ARPT-LUNKEN FIELD	None	2000-01-02	201162	BUSINESS	Ohio	0
9	Airplane	MIAMI INTL	None	2000-01-02	203926	BUSINESS	Florida	2,000
10	Airplane	SAN FRANCISCO INTL ARPT		2000-01-02	201161	UNITED AIRLINES	California	0
11	Airplane	SALT LAKE CITY INTL	Other	2000-01-02	201559	DELTA AIR LINES	Utah	0
12		MIAMI INTL		2000-01-02	205149	UNKNOWN	Florida	
13	Airplane	SOUTHWEST FLORIDA INTL ARPT	None	2000-01-02	201026	BUSINESS	Florida	200
14	Airplane	KANSAS CITY INTL	None	2000-01-02	200142	TRANS WORLD AIRLINES	Missouri	350
15	Airplane	NASHVILLE INTL	None	2000-01-03	208776	US AIRWAYS*	Tennessee	800
16	Airplane	SAN ANTONIO INTL	Precautionary Landing	2000-01-03	201027	SOUTHWEST AIRLINES	Texas	
17	Airplane	SALT LAKE CITY INTL	Other	2000-01-03	201859	DELTA AIR LINES	Utah	
18	Airplane	PENSACOLA REGIONAL	None	2000-01-03	202643	COMAIR AIRLINES	Florida	0
19	Airplane	THEODORE FRANCIS GREEN STATE	None	2000-01-03	200378	BUSINESS	Rhode Island	500
20	Airplane	ATLANTA INTL	Engine Shut Down	2000-01-03	203735	AIRTRAN AIRWAYS	Georgia	1,800
21	Airplane	BALTIMORE WASH INTL	None	2000-01-04	205029	SOUTHWEST AIRLINES	Maryland	50
22	Airplane	MINETA SAN JOSE INTL	Precautionary Landing	2000-01-04	208470	AMERICAN AIRLINES	California	100
23	Airplane	SAN ANTONIO INTL		2000-01-04	207197	SOUTHWEST AIRLINES	Texas	1,400
24	Airplane	UNKNOWN		2000-01-04	[illegible]	UNITED AIRLINES	N/A	

I also cleaned up the FlightDate column:

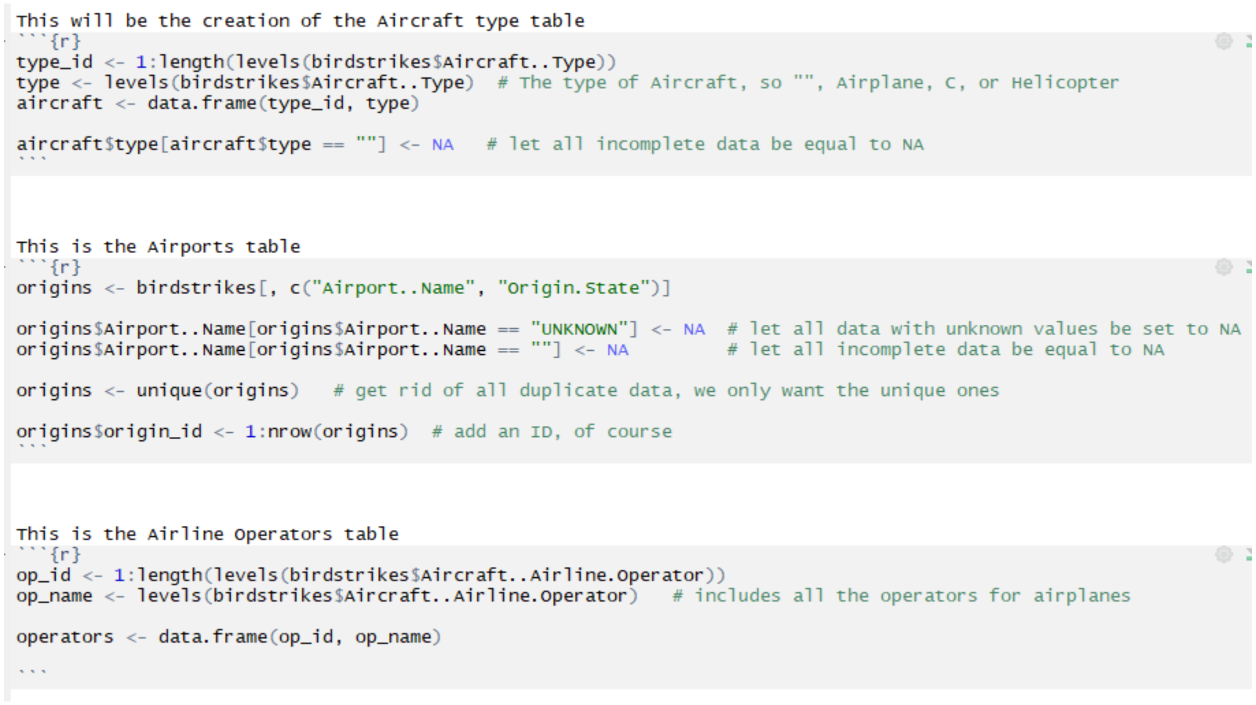
~~~
```{r}
birdstrikes$FlightDate <- as.POSIXct(birdstrikes$FlightDate, format="%m/%d/%Y")
```
~~~

And now, the boring parts are over. I will now be creating tables. Here is a visual model/diagram that I have created in MySQL Workbench. Since I was going to be using MySQL, it made sense to me to use that to create a diagram. However, it was difficult for me to show primary and foreign keys as arrows, but there are yellow key signs right next to what should be primary and foreign keys. Hopefully they will be acceptable.



As you can see, there are 5 tables and the incidents table mirrors the birdstrikes table. The rest have some values but not all of them.

Now that I have an outline of how many tables I need and what I need for each table, I can now go on to create these tables.



This will be the creation of the Aircraft type table

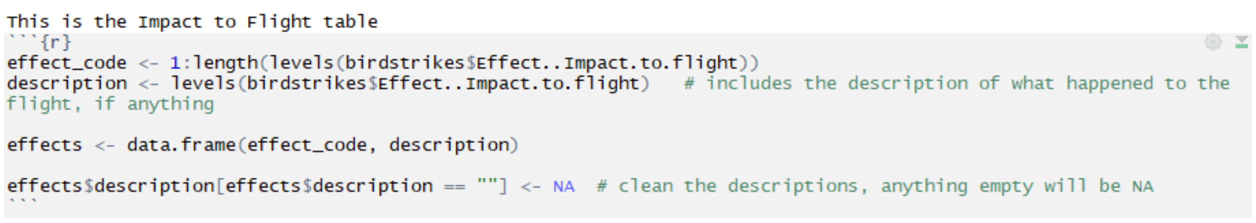
~~~
```{r}
type_id <- 1:length(levels(birdstrikes$Aircraft..Type))
type <- levels(birdstrikes$Aircraft..Type)  # The type of Aircraft, so "", Airplane, C, or Helicopter
aircraft <- data.frame(type_id, type)

aircraft$type[aircraft$type == ""] <- NA   # let all incomplete data be equal to NA
```
~~~

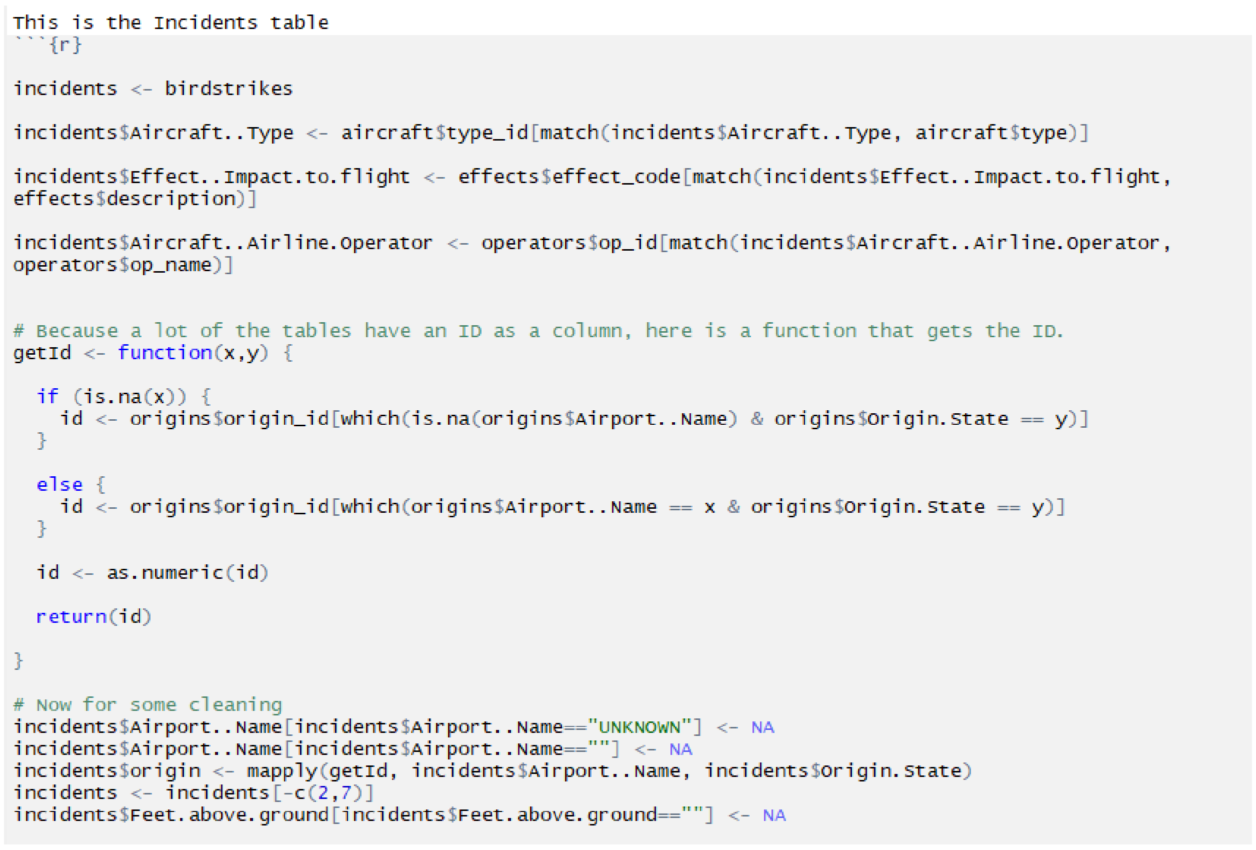
This is the Airports table

~~~
```{r}
origins <- birdstrikes[, c("Airport..Name", "Origin.State")]

origins$Airport..Name[origins$Airport..Name == "UNKNOWN"] <- NA  # let all data with unknown values be set to NA
origins$Airport..Name[origins$Airport..Name == ""] <- NA         # let all incomplete data be equal to NA

origins <- unique(origins)   # get rid of all duplicate data, we only want the unique ones

origins$origin_id <- 1:nrow(origins)  # add an ID, of course
```
~~~

This is the Airline Operators table

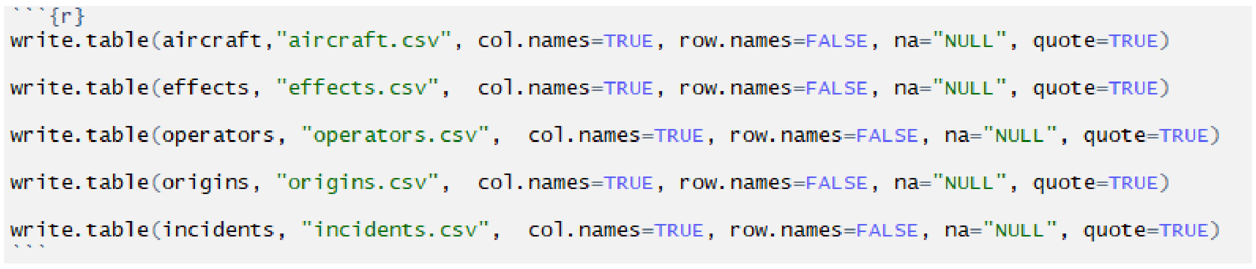
~~~
```{r}
op_id <- 1:length(levels(birdstrikes$Aircraft..Airline.Operator))
op_name <- levels(birdstrikes$Aircraft..Airline.Operator)   # includes all the operators for airplanes

operators <- data.frame(op_id, op_name)

```
~~~



~~~
This is the Impact to Flight table
```{r}
effect_code <- 1:length(levels(birdstrikes$Effect..Impact.to.flight))
description <- levels(birdstrikes$Effect..Impact.to.flight)   # includes the description of what happened to the
flight, if anything

effects <- data.frame(effect_code, description)

effects$description[effects$description == ""] <- NA  # clean the descriptions, anything empty will be NA
~~~
```



```
This is the Incidents table
```{r}

incidents <- birdstrikes

incidents$Aircraft..Type <- aircraft$type_id[match(incidents$Aircraft..Type, aircraft$type)]

incidents$Effect..Impact.to.flight <- effects$effect_code[match(incidents$Effect..Impact.to.flight,
effects$description)]

incidents$Aircraft..Airline.Operator <- operators$op_id[match(incidents$Aircraft..Airline.Operator,
operators$op_name)]


# Because a lot of the tables have an ID as a column, here is a function that gets the ID.
getId <- function(x,y) {

  if (is.na(x)) {
    id <- origins$origin_id[which(is.na(origins$Airport..Name) & origins$Origin.State == y)]
  }

  else {
    id <- origins$origin_id[which(origins$Airport..Name == x & origins$Origin.State == y)]
  }

  id <- as.numeric(id)

  return(id)

}

# Now for some cleaning
incidents$Airport..Name[incidents$Airport..Name=="UNKNOWN"] <- NA
incidents$Airport..Name[incidents$Airport..Name==""] <- NA
incidents$origin <- mapply(getId, incidents$Airport..Name, incidents$Origin.State)
incidents <- incidents[-c(2,7)]
incidents$Feet.above.ground[incidents$Feet.above.ground==""] <- NA
```

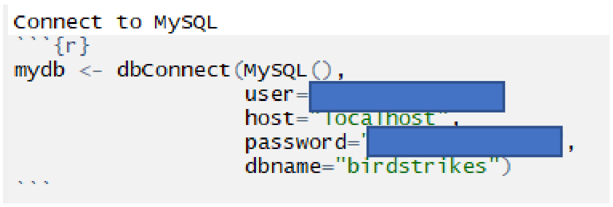
As you can see, the incidents table is quite extensive compared to the other ones mainly because I had to clean it so much, as it is a mirror of the birdstrikes table. I had to create a getID function to get the IDs of several variables and add them to the table.

On to the next part: implementing the tables into my database design and loading the data from the excel file. First, I had to write the tables:



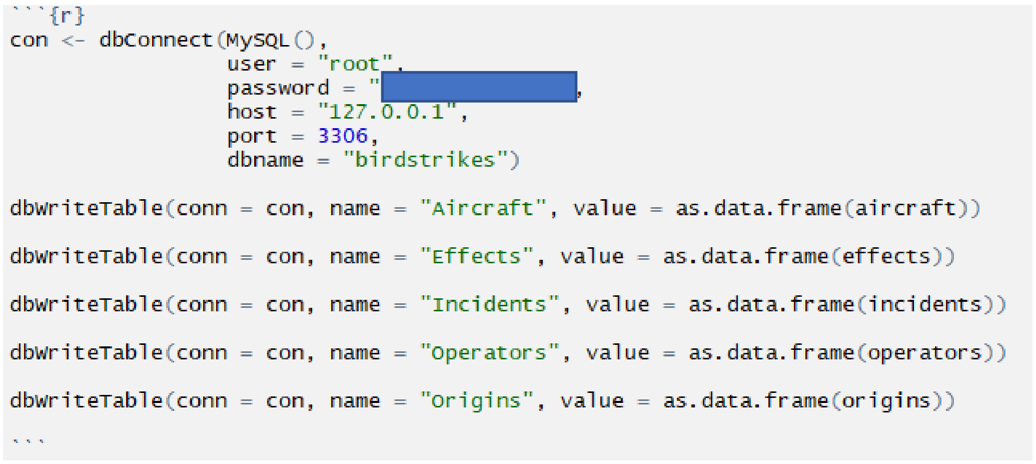
~~~
```{r}
write.table(aircraft,"aircraft.csv", col.names=TRUE, row.names=FALSE, na="NULL", quote=TRUE)

write.table(effects, "effects.csv",  col.names=TRUE, row.names=FALSE, na="NULL", quote=TRUE)

write.table(operators, "operators.csv",  col.names=TRUE, row.names=FALSE, na="NULL", quote=TRUE)

write.table(origins, "origins.csv",  col.names=TRUE, row.names=FALSE, na="NULL", quote=TRUE)

write.table(incidents, "incidents.csv",  col.names=TRUE, row.names=FALSE, na="NULL", quote=TRUE)
```
~~~

Once this was done, I had to connect to the database I had created in MySQL:



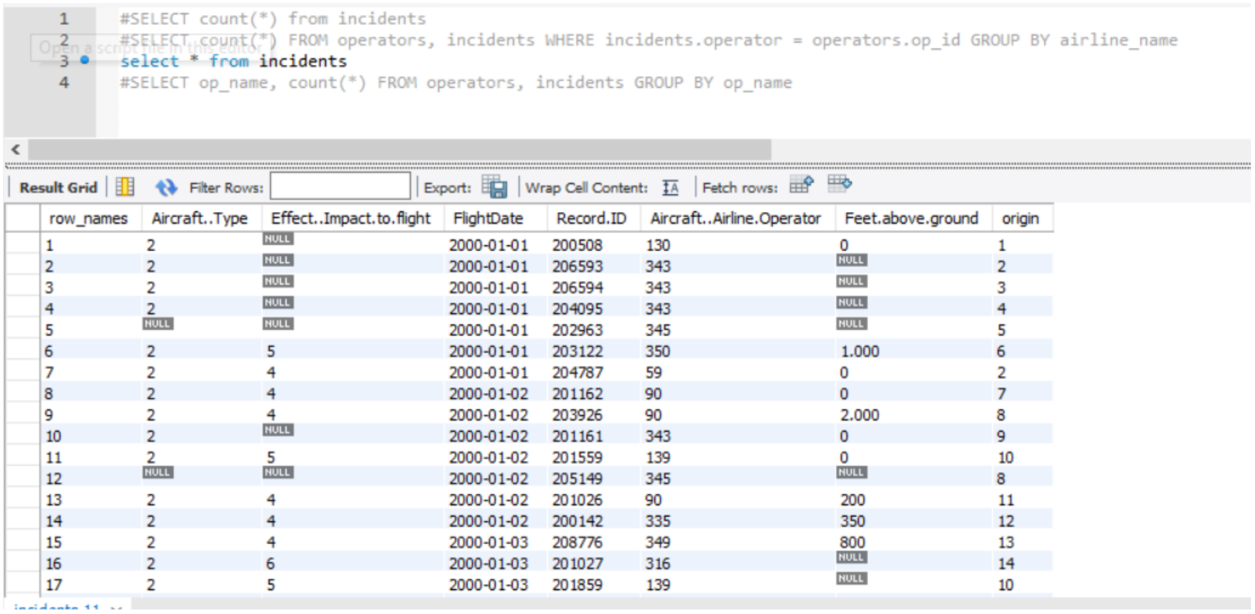
~~~
Connect to MySQL
```{r}
mydb <- dbConnect(MySQL(),
                  user=[redacted]
                  host="localhost",
                  password='[redacted],
                  dbname="birdstrikes")
```
~~~

Now, I had to create my connection and write tables into the database:



~~~
```{r}
con <- dbConnect(MySQL(),
                 user = "root",
                 password = "[redacted],
                 host = "127.0.0.1",
                 port = 3306,
                 dbname = "birdstrikes")

dbWriteTable(conn = con, name = "Aircraft", value = as.data.frame(aircraft))

dbWriteTable(conn = con, name = "Effects", value = as.data.frame(effects))

dbWriteTable(conn = con, name = "Incidents", value = as.data.frame(incidents))

dbWriteTable(conn = con, name = "Operators", value = as.data.frame(operators))

dbWriteTable(conn = con, name = "Origins", value = as.data.frame(origins))
```
~~~

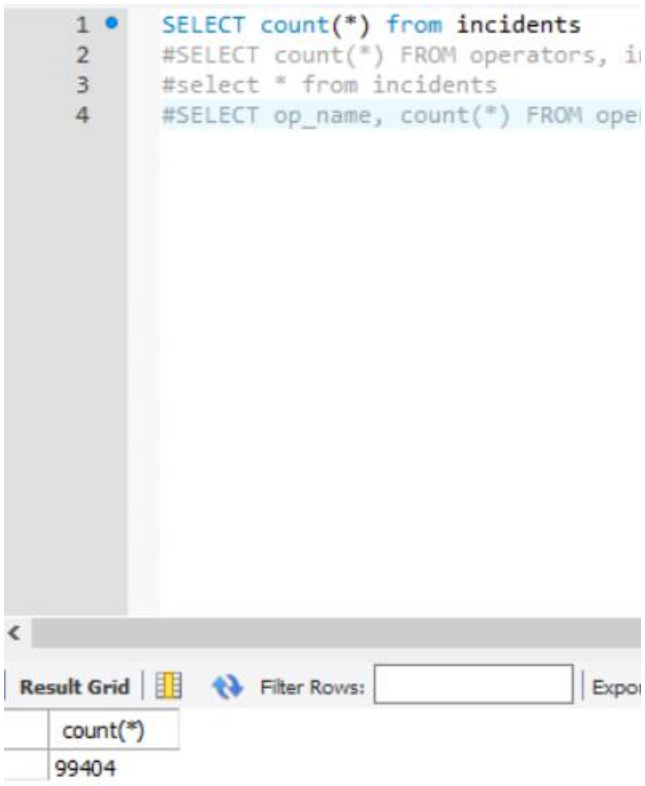
From here, I can go to MySQL and type in commands:



```
#SELECT count(*) from incidents
#SELECT count(*) FROM operators, incidents WHERE incidents.operator = operators.op_id GROUP BY airline_name
select * from incidents
#SELECT op_name, count(*) FROM operators, incidents GROUP BY op_name
```

row_names	Aircraft..Type	Effect..Impact.to.flight	FlightDate	Record.ID	Aircraft..Airline.Operator	Feet.above.ground	origin
1	2	NULL	2000-01-01	200508	130	0	1
2	2	NULL	2000-01-01	206593	343	NULL	2
3	2	NULL	2000-01-01	206594	343	NULL	3
4	2	NULL	2000-01-01	204095	343	NULL	4
5	NULL	NULL	2000-01-01	202963	345	NULL	5
6	2	5	2000-01-01	203122	350	1.000	6
7	2	4	2000-01-01	204787	59	0	2
8	2	4	2000-01-02	201162	90	0	7
9	2	4	2000-01-02	203926	90	2.000	8
10	2	NULL	2000-01-02	201161	343	0	9
11	2	5	2000-01-02	201559	139	0	10
12	NULL	NULL	2000-01-02	205149	345	NULL	8
13	2	4	2000-01-02	201026	90	200	11
14	2	4	2000-01-02	200142	335	350	12
15	2	4	2000-01-03	208776	349	800	13
16	2	6	2000-01-03	201027	316	NULL	14
17	2	5	2000-01-03	201859	139	NULL	10

As you can see, I had other commands commented out, as I can only have one “Select” on screen at once. Once I run the command, it either prints out the value or a table with the command outputted. Here is a screenshot of a command that outputs a number:



```
SELECT count(*) from incidents
#SELECT count(*) FROM operators, i
#select * from incidents
#SELECT op_name, count(*) FROM ope
```

```
count(*)
99404
```

This is also the answer to the first problem of part 3, I just wanted to let you know that it works on MySQL.

I used many resources along with Professor Schedlbauer's notes to get this to work with MySQL. The website links are:

- http://www.jason-french.com/blog/2014/07/03/using-r-with-mysql-databases/
- https://www.r-bloggers.com/accessing-mysql-through-r/
- https://stackoverflow.com/questions/41466031/how-to-write-entire-dataframe-into-mysql-table-in-r
- https://stackoverflow.com/questions/41848862/how-to-check-if-the-connection-to-mysql-through-rmysql-persists-or-not
- https://www.siteground.com/kb/how_can_i_empty_out_an_sql_database/
- http://g2pc1.bu.edu/~qzpeng/manual/MySQL%20Commands.htm
- https://github.com/smartinsightsfromdata/rpostgresql/issues/43
- https://www.w3schools.com/sql/sql_select.asp
- https://technet.microsoft.com/en-us/library/ms190742(v=sql.105).aspx
- https://mkmanu.wordpress.com/2014/07/24/r-and-mysql-a-tutorial-for-beginners/
- https://www.connectionstrings.com/mysql/

I had a lot of trouble connecting to the database from RStudio, but after much researching (obviously), I was able to do it.