Practicum I CS5200

Madhavi Arvind Saraf

Spring 2023

Connect to database

[Question 1 & 2] This code chunk connects to AWS database.

For this code chunk I am assuming there already exists a database named ‘Birdstrikes’.So the database name parameter is passed in the dbConnect() function.This chunk is not evaluated by default. But can be changed later.

```
# 1. Library
if (!require("RMySQL")) install.packages("RMySQL")
library(RMySQL)

# 2. Settings
db_user <- 'admin'
db_password <- 'root.13.'
db_name <- 'Birdstrikes'
db_host <- 'cs5200p1.cdulqqytkodx.us-east-2.rds.amazonaws.com'
db_port <- 3306
#
# 3. Read data from db
dbConnection <-  dbConnect(MySQL(), user = db_user, password =
db_password,dbname = db_name, host = db_host, port = db_port)
```

For this code chunk I am assuming the database does not exist while establishing the connection.So the connection is established without the database name paremter in dbConnect().Then the required database is created after the connection is established. This is is the chunk that is evaluated by default for establishing connection.

```
# 1. Library
if (!require("RMySQL")) install.packages("RMySQL")

## Loading required package: RMySQL

## Loading required package: DBI

library(RMySQL)

# 2. Settings
db_user <- 'admin'
db_password <- 'root.13.'
db_name <- 'Birdstrikes'
db_host <- 'cs5200p1.cdulqqytkodx.us-east-2.rds.amazonaws.com'
```

```
db_port <- 3306
#
# 3. Read data from db
dbConnection <-  dbConnect(MySQL(), user = db_user, password =
db_password,host = db_host, port = db_port)
dropDBqry <- paste0("DROP DATABASE IF EXISTS ",db_name)
createDBqry <- paste0("CREATE DATABASE IF NOT EXISTS ",db_name)
dropDBqry <- paste0("DROP DATABASE IF EXISTS ",db_name)
useDBqry <- paste0("USE ",db_name)

dbExecute(dbConnection,dropDBqry)

## [1] 4

dbExecute(dbConnection,createDBqry)

## [1] 1

dbExecute(dbConnection,useDBqry)

## [1] 0
```

Create DATABASE [create tables for question 4]

[Question:4B] This code chunk creates the airports table in the database, since aid is the synthetic primary key its kept as Auto increment)

```sql
CREATE TABLE airports
(
  airportName TEXT NOT NULL,
  state TEXT NOT NULL,
  airportCode TEXT NOT NULL,
  aid INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT
)
```

[Question:4D] This code chunk creates the conditions table in the database, since cid is the synthetic primary key its kept as Auto increment)

```sql
CREATE TABLE conditions
(
  cid INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `condition` TEXT NOT NULL,
  explanation TEXT
)
```

[Question:4E] This code chunk creates the airlines table in the database, since eid is the synthetic primary key its kept as Auto increment.

```sql
CREATE TABLE airlines
(
  airlineName TEXT NOT NULL,
  airlineCode TEXT,
  flag TEXT,
  eid INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT
)
```

[Question:4A,4C,4F] This code chunk creates the incidents table in the database.

```
CREATE TABLE incidents
(
  rid INTEGER PRIMARY KEY,
  origin INTEGER NOT NULL,
  airline INTEGER NOT NULL,
  aircraft TEXT NOT NULL,
  `flight.phase` TEXT NOT NULL,
  `dep.date` DATE NOT NULL,
  conditions INTEGER NOT NULL,
  altitude INTEGER NOT NULL DEFAULT 0 CONSTRAINT `check_altitude` CHECK
(altitude >= 0),
  FOREIGN KEY(conditions) REFERENCES conditions(cid) ON DELETE CASCADE ON
UPDATE CASCADE,
  FOREIGN KEY(origin) REFERENCES airports(aid) ON DELETE CASCADE ON UPDATE
CASCADE ,
  FOREIGN KEY(airline) REFERENCES airlines(eid) ON DELETE CASCADE ON UPDATE
CASCADE,
  warned BOOLEAN NOT NULL DEFAULT false
)
```

Testing the tables created and verification

[Question 4G] This code chunk is used to display the details of airports table created.

```
DESC airports
```

[Question 4G] This code chunk is used to display the details of airlines table created.

```
DESC airlines
```

[Question 4G] This code chunk is used to display the details of incidents table created.

```
DESC incidents
```

[Question 4G] This code chunk is used to display the details of conditions table created.

```
DESC conditions
```

[Question 4G] Testing the airports table by inserting the record

```
INSERT INTO airports (aid,airportName,state,airportCode) VALUES (1,"BOSTON
AIRPORT","MA","BOSLOG1")
```

[Question 4G] Testing the airlines table by inserting the record

```
INSERT INTO airlines (eid,airlineName,airlineCode,flag) VALUES (1,"American
Airlines","AA1","USA")
```

[Question 4G] Testing the conditions table by inserting the record

```
INSERT INTO conditions (cid,`condition`,explanation) VALUES
(1,"OVERCAST","overcast condition");
```

[Question 4G] Testing the conditions table by inserting the record

```
INSERT INTO
incidents(rid,aircraft,`flight.phase`,`dep.date`,altitude,warned,conditions,o
rigin,airline) VALUES (1313,"airplane","Takeoff",'2005-07-09',10,0,1,1,1);
```

Removing the records inserted for testing

```
DELETE FROM airports;
```

Removing the records inserted for testing

```
DELETE FROM airlines;
```

Removing the records inserted for testing

```
DELETE FROM conditions;
```

Removing the records inserted for testing

```
DELETE FROM incidents;
```

Loading data from csv into the data frames

[Question 5] This code chunk populates the bds.raw data frame with the data from .csv. The csv file needs to be placed in the same path as R notebook and it should have the name BirdStrikesData-V2.csv.All the unnecessary columns which are not needed for this practicum are removed.

```
bds.raw <- read.csv("BirdStrikesData-V2.csv", header = TRUE, sep = ",")
bds.raw <- bds.raw[,!names(bds.raw) %in% c("model", "wildlife_struck",
"impact","damage","remains_collected_flag","Remarks","wildlife_size","species
","heavy_flag")]
```

[Question 6] This code chunk loads the data into the airports database table.

```
airportData <-bds.raw

#load bds.raw into new airportData data frame for further manipulation so
that
#bds.raw remains intact

airportData$airport[airportData$airport==""] <- "UNKNOWN"
airportData$origin[airportData$origin==""] <- "UNKNOWN"

#if origin,airport contain null values or no values, the default is set
#as "UNKNOWN"
```

```
airportData <- airportData[!duplicated(airportData[ , c("airport")]), ]
#extracting all the unique airports from the csv (data frame)

airport_df <- airportData[,!names(airportData) %in% c("rid", "aircraft",
"flight_date","airline","flight_phase","sky_conditions",
"pilot_warned_flag","altitude_ft")]
#remove  columns which are not needed for airports table

airport_df["airportCode"] <- "code"
#set default values for airportCode column

airport_df["aid"] <- as.integer(1)

names(airport_df)[names(airport_df) == "origin"] <- "state"
#renaming columns in data frame to match the database table names

names(airport_df)[names(airport_df) == "airport"] <- "airportName"
#renaming columns in data frame to match the database table names

for(i in 1:nrow(airport_df))
{
  airport_df[i,4] <- as.integer(i)
}
 #for aid column
dbWriteTable(dbConnection,"airports",airport_df,append=TRUE,row.names=F)

## [1] TRUE

#write the data into the mysql airports table
```

[Question 6] This code chunk loads the data from csv into the airlines table

```
airlineData <- bds.raw
#load bds.raw into new airlineData data frame for further manipulation so
#that bds.raw remains intact

airlineData["airlineCode"] <- "default_code"
airlineData["flag"] <- "default_country"
#add new column and set default too match the database table structure.

airlineData$airline[airlineData$airline==""] <- "UNKNOWN"
#if airline contain null values or no values, the default is set as "UNKNOWN"

airlineData <- airlineData[,!names(airlineData) %in% c("rid", "aircraft",
"flight_date","airport","origin","flight_phase","sky_conditions",
"pilot_warned_flag","altitude_ft")]
#removing columns which are not needed for airlines table

airlineData_df <- airlineData[!duplicated(airlineData[ , c("airline")]), ]
```

```
#extracting all the unique airlines from the csv (data frame)

names(airlineData_df)[names(airlineData_df) == "airline"] <- "airlineName"
#renaming columns corresponding to database table

airlineData_df["eid"] <- as.integer(1)

for(i in 1:nrow(airlineData_df))
{
  airlineData_df[i,4] <- as.integer(i)      #for eid column
}

dbWriteTable(dbConnection,"airlines",airlineData_df,append=TRUE,row.names=F)

## [1] TRUE

#write the data into the mysql airlines table
```

[Question 6] This code chunk loads the data from csv into the conditions table

```
skycnd <- unique(bds.raw$sky_conditions)
#extract unique conditions from bds.raw

condition_df <- setNames(data.frame(matrix(ncol = 3, nrow = 0)),
c("cid", "condition", "explanation"))
#create data frame corresponding to the table structure

i <- as.integer(1)

#populate the data frame columns with data
for(condition in skycnd)
{

  condition_df[i,1] <- as.integer(i)
  if(condition==''){condition <- paste0("Unknown")}
  condition_df[i,2] <- paste0(condition)
  condition_df[i,3] <- paste0("condition explanation")
  i <- i + 1
}

dbWriteTable(dbConnection,"conditions",condition_df,append=TRUE,row.names=F)

## [1] TRUE

# load the data from data frame into the database table
```

[Question 6] Load the data from csv into the incidents database table.

```
if (!require ("dplyr")) install.packages("dplyr")

## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(dplyr)

incidentData <- bds.raw
#load bds.raw into new airlineData data frame for further manipulation so
#that bds.raw remains intact

#NOTE : FOR rid column its assumed that the values are never NULL or missing.
All the values are always and are UNIQUE.

incidentData$flight_phase[incidentData$flight_phase=="Landing Roll"] <-
"Landing"
incidentData$flight_phase[incidentData$flight_phase=="Approach"] <- "Landing"
incidentData$flight_phase[incidentData$flight_phase=="Take-off run"] <-
"Takeoff"
incidentData$flight_phase[incidentData$flight_phase=="Climb"] <- "Takeoff"
incidentData$flight_phase[incidentData$flight_phase=="Parked"] <- "Onground"
incidentData$flight_phase[incidentData$flight_phase=="" ] <- "Unknown"
incidentData$flight_phase[incidentData$flight_phase=="Descent"] <- "Inflight"
incidentData$flight_phase[incidentData$flight_phase=="Taxi"] <- "Onground"

#Assumption for flight phases is as follows:
#Landing - "Landing Roll","Approach"
#Takeoff - "Take-off run","Climb"
#"Onground" - "Parked","Taxi" [New phase introduced]
#"Descent" - "Inflight" [Even though the landing has just begun the flight
has
# significant altitude, so I am considering this as Inflight]
#"Unknown" - for all the other phases

incidentData$altitude_ft[incidentData$altitude_ft==""] <- as.integer(0)
incidentData$aircraft[incidentData$aircraft==""] <- "UNKNOWN"
incidentData$airport[incidentData$airport==''] <- "UNKNOWN"
incidentData$airline[incidentData$airline==''] <- "UNKNOWN"
incidentData$flight_date[incidentData$flight_date==""] <- "01-01-5555"
incidentData$airport[incidentData$airport==""] <- "UNKNOWN"
incidentData$airline[incidentData$airline==""] <- "UNKNOWN"

incidentData$pilot_warned_flag[incidentData$pilot_warned_flag!="Y"] <- FALSE
incidentData$pilot_warned_flag[incidentData$pilot_warned_flag=="Y"] <- TRUE
```

```
#If any values are null or no values in csv for all the above columns set to
#default values, default date of 01-01-555 is kept as default date if the
#date column contains empty values.

names(airport_df)[names(airport_df) == "airportName"] <- "airport"
names(incidentData)[names(incidentData) == "sky_conditions"] <- "condition"
names(airlineData_df)[names(airlineData_df) == "airlineName"] <- "airline"

# For the purpose of referencing the aid.cid,eid from the other three tables
to
# origin,condion,airline respectively in incidents, I am using the merging
technique
# of all the three previous data frames with the incidents data frame. For
that purpose one column
# common between the two data frames is required for reference.The common
columns are already
# present in the data frames. Just renaming them here for merge functionality
so that
# common columns have same names.

temp_df_1 <- merge(incidentData,airport_df,by="airport")
temp_df_2 <- merge(temp_df_1,airlineData_df,by="airline")
temp_df_3 <- merge(temp_df_2,condition_df,by="condition")
incident_df <- temp_df_3

incident_df <- incident_df[,!names(incident_df) %in% c("airportCode",
"state",
"airlineCode","flag","airline","airport","explanation","condition","origin")]
#remove the columns which are not needed for incidents table

names(incident_df)[names(incident_df) == "aid"] <- "origin"
names(incident_df)[names(incident_df) == "eid"] <- "airline"
names(incident_df)[names(incident_df) == "cid"] <- "conditions"
names(incident_df)[names(incident_df) == "pilot_warned_flag"] <- "warned"
names(incident_df)[names(incident_df) == "flight_phase"] <- "flight.phase"
names(incident_df)[names(incident_df) == "flight_date"] <- "dep.date"
names(incident_df)[names(incident_df) == "altitude_ft"] <- "altitude"
#rename the column to match the incidents table in database.

incident_df$dep.date <- gsub('/','-',incident_df$dep.date)
incident_df$dep.date <- as.Date(incident_df$dep.date,'%m-%d-%Y')
#parse the dates to match the mysql format

dbWriteTable(dbConnection,"incidents",incident_df,append=TRUE,row.names=F)

## [1] TRUE

#When using dbWriteTable with overwrite, the constraints are gone. So re
assignment of constraints is #done. This piece of code is kept as commented
```

```
in case its needed for future use.
#
# addConstraint1Qry <- paste0("ALTER TABLE incidents ADD CONSTRAINT
PK_incidents PRIMARY KEY(rid)")
# dbExecute(dbConnection,addConstraint1Qry)
# addConstraint2Qry <- paste0("ALTER TABLE incidents MODIFY origin INTEGER")
# dbExecute(dbConnection,addConstraint2Qry)
# addConstraint3Qry <- paste0("ALTER TABLE incidents ADD CONSTRAINT
FK_airport FOREIGN KEY(origin) REFERENCES airports(aid)")
# dbExecute(dbConnection,addConstraint3Qry)
# addConstraint4Qry <- paste0("ALTER TABLE incidents MODIFY airline INTEGER")
# dbExecute(dbConnection,addConstraint4Qry)
# addConstraint5Qry <- paste0("ALTER TABLE incidents ADD CONSTRAINT
FK_airline FOREIGN KEY(airline) REFERENCES airlines(eid)")
#  dbExecute(dbConnection,addConstraint5Qry)
# addConstraintAQry <-paste0("ALTER TABLE incidents MODIFY conditions
INTEGER")
# dbExecute(dbConnection,addConstraintAQry);
# addConstraint6Qry <- paste0("ALTER TABLE incidents ADD CONSTRAINT
FK_condition FOREIGN KEY(conditions) REFERENCES conditions(cid)")
#   dbExecute(dbConnection,addConstraint6Qry)
#   addConstraint7Qry <- paste0("ALTER TABLE incidents ADD CONSTRAINT
check_alt CHECK(altitude>=0);")
#   dbExecute(dbConnection,addConstraint7Qry)

#write data into the table in database
```

[Question 6] This code chunk just shows that when there are missing values for airport and airline in the csv file. A record with airportName as "UNKNOWN" and auto incremented value for aid and default values for other columns is added into the airports database table. Similarly for missing airline values in csv, a record with airlineName as "UNKNOWN" and auto incremented value for eid and default values for other columns is added into the airlines database table.These records are referenced for missing values when loading incidents table and missing value for airline or airport is encountered.

```
airportUnknownQry<- "Select aid from airports where airportName='UNKNOWN'";
airlineUnknownQry <- "Select eid from airlines where airlineName='UNKNOWN'";

aid <-dbGetQuery(dbConnection,airportUnknownQry);
eid <-dbGetQuery(dbConnection,airlineUnknownQry);

incidentAidUnknownQry <- paste0("select * from incidents where origin=",aid,"
LIMIT 10");

incidentEidUnknownQry <- paste0("select * from incidents where
airline=",eid," LIMIT 10");

#There may be more records but we are limiting to only get 10 records
```

```
#Records from incidents table where airport was missing or unknown
incidentsResAid <- dbGetQuery(dbConnection,incidentAidUnknownQry)
print(incidentsResAid)
```

```
##       rid origin airline aircraft flight.phase   dep.date conditions
altitude
## 1  200830     83      31  UNKNOWN      Unknown 5555-01-01          1
0
## 2  202457     83      31  UNKNOWN      Unknown 5555-01-01          1
0
## 3  203353     83      31  UNKNOWN      Unknown 5555-01-01          2
0
## 4  204542     83      31  UNKNOWN      Unknown 5555-01-01          3
0
## 5  205473     83      31  UNKNOWN      Unknown 5555-01-01          2
0
## 6  206419     83      31  UNKNOWN      Unknown 5555-01-01          1
0
## 7  207513     83      31  UNKNOWN      Unknown 5555-01-01          2
0
## 8  208561     83      31  UNKNOWN      Unknown 5555-01-01          1
0
## 9  209353     83      31  UNKNOWN      Unknown 5555-01-01          3
0
## 10 210283     83      31  UNKNOWN      Unknown 5555-01-01          1
0
##    warned
## 1       0
## 2       0
## 3       0
## 4       0
## 5       0
## 6       0
## 7       0
## 8       0
## 9       0
## 10      0
```

```
#Records from incidents table where airline was missing or unknown
incidentsResEid <- dbGetQuery(dbConnection,incidentEidUnknownQry)
print(incidentsResEid)
```

```
##       rid origin airline aircraft flight.phase   dep.date conditions
altitude
## 1  200580    439      31 Airplane     Onground 2000-08-07          2
0
## 2  200830     83      31  UNKNOWN      Unknown 5555-01-01          1
0
## 3  201340     93      31 Airplane      Landing 2000-09-10          2
0
```

```
## 4  201358    369      31 Airplane      Landing 2000-08-17          2
300
## 5  201731    256      31 Airplane      Landing 2000-09-15          1
0
## 6  202457     83      31  UNKNOWN      Unknown 5555-01-01          1
0
## 7  202613     67      31 Airplane      Takeoff 2000-07-06          1
20
## 8  202895    204      31 Airplane      Landing 2000-08-25          1
50
## 9  202996    115      31 Airplane      Takeoff 2001-05-24          1
0
## 10 203353     83      31  UNKNOWN      Unknown 5555-01-01          2
0
##    warned
## 1       0
## 2       0
## 3       0
## 4       0
## 5       0
## 6       0
## 7       0
## 8       0
## 9       0
## 10      0
```

Displaying the data loaded from CSV into the database tables

[Question 7] Code chunk to show loading data into incidents table worked

```
select * from incidents LIMIT 100
```

Displaying records 1 - 10

rid	origin	airline	aircraft	flight.phase	dep.date	conditions	altitude	warned
1195	37	21	Airplane	Landing	2002-11-13	3	2	0
3019	707	21	Airplane	Takeoff	2002-10-10	1	400	0
3500	37	21	Airplane	Landing	2001-05-15	1	1	0
3504	37	21	Airplane	Landing	2001-05-23	1	1	0
3597	123	21	Airplane	Landing	2001-04-18	2	200	0
4064	37	21	Airplane	Landing	2000-04-06	1	1	0

rid	origin	airline	aircraft	flight.phase	dep.date	conditions	altitude	warned
4074	180	21	Airplane	Takeoff	2002-07-15	1	0	0
4076	37	21	Airplane	Takeoff	2002-07-15	2	500	0
4090	114	21	Airplane	Takeoff	2001-07-02	2	50	0
4091	114	21	Airplane	Takeoff	2001-07-07	2	0	0

[Question 7] Code chunk to show loading data into airlines table worked

```
select * from airlines LIMIT 100
```

Displaying records 1 - 10

airlineName	airlineCode	flag	eid
US AIRWAYS*	default_code	default_country	1
AMERICAN AIRLINES	default_code	default_country	2
BUSINESS	default_code	default_country	3
ALASKA AIRLINES	default_code	default_country	4
COMAIR AIRLINES	default_code	default_country	5
UNITED AIRLINES	default_code	default_country	6
AIRTRAN AIRWAYS	default_code	default_country	7
AIRTOURS INTL	default_code	default_country	8
AMERICA WEST AIRLINES	default_code	default_country	9
EXECUTIVE JET AVIATION	default_code	default_country	10

[Question 7] Code chunk to show loading data into airports table worked

```
select * from airports LIMIT 100
```

Displaying records 1 - 10

airportName	state	airportCode	aid
LAGUARDIA NY	New York	code	1
DALLAS/FORT WORTH INTL ARPT	Texas	code	2
LAKEFRONT AIRPORT	Louisiana	code	3
SEATTLE-TACOMA INTL	Washington	code	4
NORFOLK INTL	Virginia	code	5
GUAYAQUIL/S BOLIVAR	N/A	code	6
NEW CASTLE COUNTY	Delaware	code	7

airportName	state	airportCode	aid
WASHINGTON DULLES INTL ARPT	DC	code	8
ATLANTA INTL	Georgia	code	9
ORLANDO SANFORD INTL AIRPORT	Florida	code	10

[Question 7] Code chunk to show loading data into conditions look up table worked

```
select * from conditions
```

3 records

cid	condition	explanation
1	No Cloud	condition explanation
2	Some Cloud	condition explanation
3	Overcast	condition explanation

[Question 7] Testing Primary key, Foreign key linking from incidents table to airports, airlines table

```
incidentQry<- dbGetQuery(dbConnection,"Select * from incidents LIMIT 1");
print(incidentQry);

##    rid origin airline aircraft flight.phase   dep.date conditions altitude
## 1 1195     37      21 Airplane      Landing 2002-11-13          3        2
##   warned
## 1      0

origin <- incidentQry[1,7]
airline <- incidentQry[1,8]

#get the airport table referenced by the incident
airportRec <- dbGetQuery(dbConnection,paste0("Select * from airports where
aid=",origin))
print(airportRec)

##         airportName     state airportCode aid
## 1 LAKEFRONT AIRPORT Louisiana        code   3

# get the airline table referenced by the incident
airlineRec <- dbGetQuery(dbConnection,paste0("Select * from airlines where
eid=",airline))
print(airlineRec)

##         airlineName  airlineCode            flag eid
## 1 AMERICAN AIRLINES default_code default_country   2
```

Execution of Queries

[Question 8] Finding the 10 states with the greatest number of incidents

```
SELECT state,COUNT(*) FROM incidents i JOIN airports a where i.origin=a.aid
GROUP BY a.state ORDER BY COUNT(*) DESC LIMIT 10
```

Displaying records 1 - 10

state	COUNT(*)
California	2499
Texas	2445
Florida	2045
New York	1316
Illinois	1007
Pennsylvania	985
Missouri	956
Kentucky	806
Ohio	773
Hawaii	716

[Question 9] Finding airlines with greater than average number of incidents.

```
SELECT COUNT(airline) as 'Incident_count',airlineName FROM incidents,airlines
WHERE airline=eid GROUP BY airline HAVING Incident_count > (SELECT
AVG(A.Incident_count) FROM (SELECT COUNT(airline) as
'Incident_count',airlineName,airline FROM incidents,airlines WHERE
airline=eid GROUP BY airline) AS A)
```

Displaying records 1 - 10

Incident_count	airlineName
797	US AIRWAYS*
2058	AMERICAN AIRLINES
3074	BUSINESS
304	ALASKA AIRLINES
317	COMAIR AIRLINES
506	UNITED AIRLINES
414	AIRTRAN AIRWAYS
157	AMERICA WEST AIRLINES
332	HAWAIIAN AIR
1349	DELTA AIR LINES

[Question 10] Finding number of incidents by phase and month

```
dbquery <- "SELECT `flight.phase` AS 'FLIGHT_PHASE',
SUBSTRING(MONTHNAME(`dep.date`),1,3) AS 'INCIDENT_MONTH', COUNT(rid) as
'INCIDENT_COUNT',`dep.date` FROM incidents GROUP BY `flight.phase`,
MONTHNAME(`dep.date`)ORDER BY INCIDENT_COUNT DESC"
inciCountMapping <- dbGetQuery(dbConnection, dbquery)
head(inciCountMapping)
```

```
##   FLIGHT_PHASE INCIDENT_MONTH INCIDENT_COUNT   dep.date
## 1      Landing            Sep           2165 2000-09-08
## 2      Landing            Aug           2137 2000-08-31
## 3      Landing            Oct           1924 2001-10-04
## 4      Landing            Jul           1903 2002-07-28
## 5      Takeoff            Aug           1494 2002-08-13
## 6      Landing            May           1463 2001-05-15
```

[Question 10 Alternate] This is a alternate code chunk for query to return the months as numeric values

```
dbquery <- "SELECT `flight.phase` AS 'FLIGHT_PHASE', MONTH(`dep.date`)  AS
'INCIDENT_MONTH', COUNT(rid) as 'INCIDENT_COUNT',`dep.date` FROM incidents
GROUP BY `flight.phase`, MONTH(`dep.date`)ORDER BY INCIDENT_COUNT DESC"
incidentCountMappingAlt <- dbGetQuery(dbConnection, dbquery)
head(incidentCountMappingAlt)
```

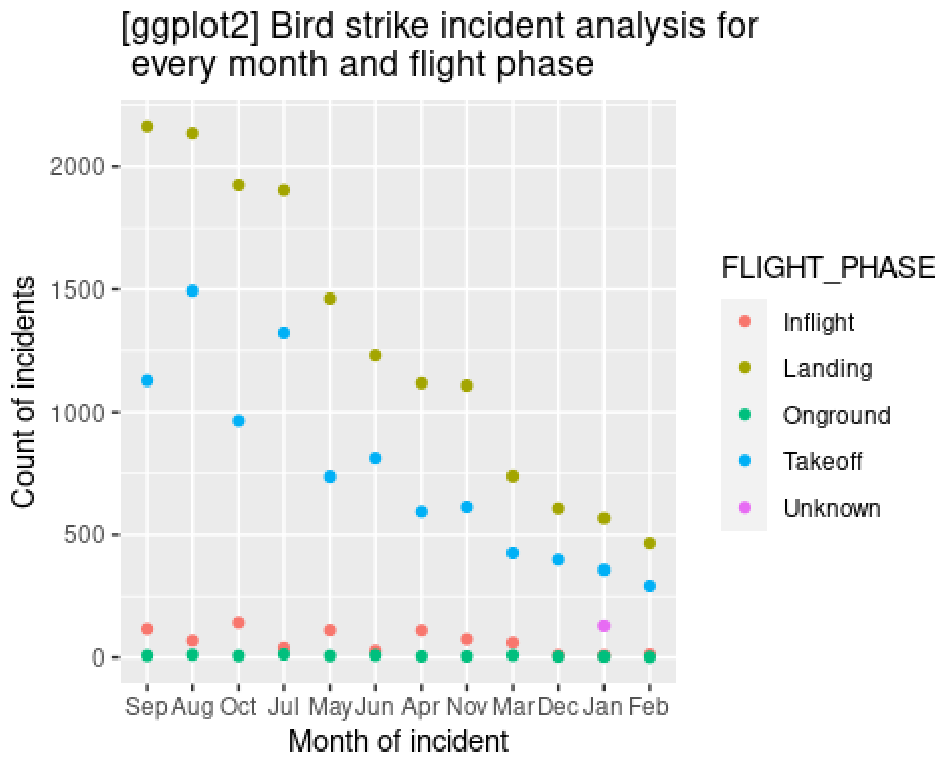
```
##   FLIGHT_PHASE INCIDENT_MONTH INCIDENT_COUNT   dep.date
## 1      Landing              9           2165 2000-09-08
## 2      Landing              8           2137 2000-08-31
## 3      Landing             10           1924 2001-10-04
## 4      Landing              7           1903 2002-07-28
## 5      Takeoff              8           1494 2002-08-13
## 6      Landing              5           1463 2001-05-15
```

Graph plotting

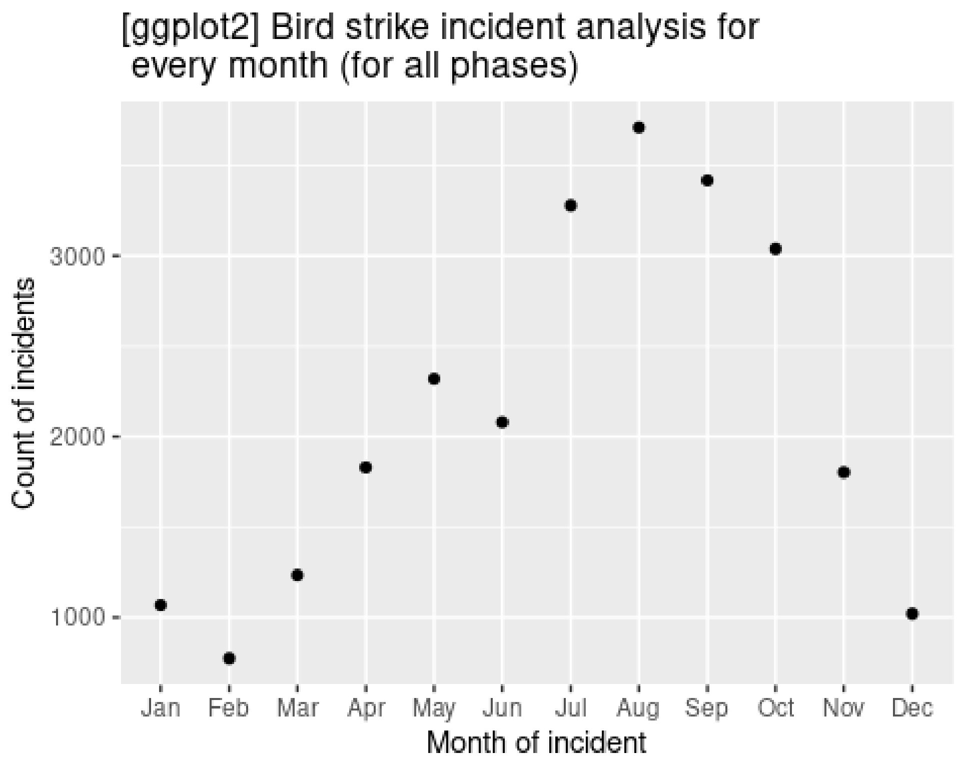
[Question 11] This code chunk displays the scatter plot using ggplot2 library.Here the months are present as month names on x-axes

```
if (!require("sqldf")) install.packages("sqldf")

## Loading required package: sqldf

## Loading required package: gsubfn

## Loading required package: proto

## Loading required package: RSQLite

##
## Attaching package: 'RSQLite'

## The following object is masked from 'package:RMySQL':
##
##     isIdCurrent

## sqldf will default to using MySQL
```

```
if (!require("ggplot2")) install.packages("ggplot2")

## Loading required package: ggplot2

if (!require("forcats")) install.packages("forcats")

## Loading required package: forcats

library(ggplot2)
library(forcats)
library(sqldf)

query <- "select FLIGHT_PHASE,SUM(INCIDENT_COUNT) as
COUNT_OF_INCIDENTS,`dep.date`,INCIDENT_MONTH from inciCountMapping group by
INCIDENT_MONTH ORDER BY strftime('%m',`dep.date`)"
inciCountMappingBlk <- sqldf(query,drv="SQLite")

ggplot(inciCountMapping
       , aes(x = fct_inorder(INCIDENT_MONTH)
             , y = INCIDENT_COUNT,color=FLIGHT_PHASE)) +
geom_point()+scale_x_discrete()+labs(x="Month of incident",y = "Count of
incidents")+ggtitle("[ggplot2] Bird strike incident analysis for\n every
month and flight phase")
```

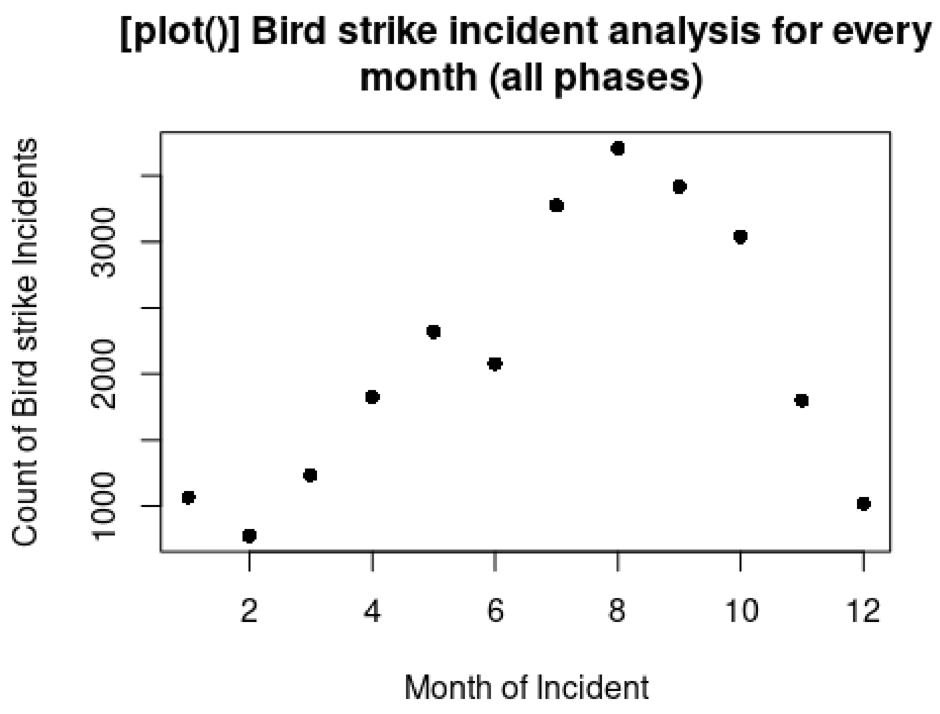


```
ggplot(inciCountMappingBlk
       , aes(x = fct_inorder(INCIDENT_MONTH)
             , y = COUNT_OF_INCIDENTS)) +
geom_point()+scale_x_discrete()+labs(x="Month of incident",y = "Count of
incidents")+ggtitle("[ggplot2] Bird strike incident analysis for\n every
month (for all phases)")
```



[Question 11 Alternate] Graph plotting with months as numeric values using the usual plot() function

```
if (!require("sqldf")) install.packages("sqldf")
library(sqldf)
query <- "select SUM(INCIDENT_COUNT) as COUNT_OF_INCIDENTS,INCIDENT_MONTH
from incidentCountMappingAlt group by INCIDENT_MONTH"
incidentCountMappingPlot <- sqldf(query,drv="SQLite")

attach(incidentCountMappingPlot)
plot(INCIDENT_MONTH, COUNT_OF_INCIDENTS, main="[plot()] Bird strike incident
analysis for every\n month (all phases)",xlab="Month of Incident",
ylab="Count of Bird strike Incidents", pch=16 )
```



[Question 12] Choosing the right mode for the flight phase to fit the enums is done in the procedure. There are 5 enums : Takeoff,Landing,Onground,Inflight,Unknown.Since the flight.phase of the database can only accept fixed values, check is being done in the procedure to verify if the phases are valid, if values other than the allowed ENUMS is added, the phase is made as 'Unknown'. If there are new airport,airline inserted , new records are automatically inserted in their respective tables and the new incident record references the newly inserted record of other tables.NOTE:If the condition_var, the value referencing the condition look up table has a values which is not present in the look up table. Then the execution is aborted and record is NOT inserted. Since conditions is a look up table, it cannot be updated frequently like airport and airline table. So conditions value in incidents table can only accept existing values in the conditions look up table.

```
CREATE PROCEDURE insertIncidentProcedure(IN inci_id INTEGER,IN aircraft
TEXT,IN flight_phase TEXT,IN flight_date DATE,IN altitude_ft INTEGER,IN
warned_flag INTEGER,IN conditions_var INTEGER,IN airport INTEGER,IN airline
INTEGER)
exit_point:BEGIN
   DECLARE condition_temp INTEGER;
   DECLARE airport_temp INTEGER;
   DECLARE airline_temp INTEGER;
   DECLARE flag INTEGER DEFAULT 0;
   DECLARE phase TEXT;
   DECLARE tempvar TEXT;

   SELECT cid INTO condition_temp from `conditions` where cid=conditions_var;
   IF condition_temp is NULL THEN
```

```
      LEAVE exit_point;
   END IF;

   SET phase=LOWER(flight_phase);


    IF phase='takeoff' THEN
      SET tempvar="Takeoff";
      SET flag = 1;
    END IF;

    IF phase='onground' THEN
      SET tempvar="Onground";
      SET flag = 1;
    END IF;

    IF phase='inflight' THEN
      SET tempvar="Inflight";
      SET flag = 1;
    END IF;

   IF phase='landing' THEN
      SET tempvar="Landing";
      SET flag = 1;
    END IF;

    IF phase='takeoff' THEN
      SET tempvar="Takeoff";
      SET flag = 1;
    END IF;


   IF flag=0 THEN
      SET tempvar="Unknown";
    END IF;

   SELECT aid INTO airport_temp from airports where aid=airport;
   SELECT eid INTO airline_temp from airlines where eid=airline;

   IF airport_temp is NULL THEN
   INSERT INTO airports(aid,airportName,airportCode,state) VALUES
(airport,'Unknown','default_code','Unknown');
   END IF;

  IF airline_temp is NULL THEN
   INSERT INTO airlines(eid,airlineName,airlineCode,flag) VALUES
(airline,'Unknown','default_code','default_country');
   END IF;
   INSERT INTO
incidents(rid,aircraft,`flight.phase`,`dep.date`,altitude,warned,conditions,o
rigin,airline) VALUES
(inci_id,aircraft,tempvar,flight_date,altitude_ft,warned_flag,conditions_var,
airport,airline);

END
```

[Question 12] This code chunk attempts to insert the new record in incident table using procedure. Since all thevalues are valid, this record will be successfully inserted.

```
CALL insertIncidentProcedure(4,'Aircraft','Takeoff','2002-09-
09',100,0,1,5555,5555);
```

[Question 12] This code chunk tests if the procedure call just above was successful or NOT. Its supposed to return one row

```
dbquery <- "SELECT * FROM incidents WHERE rid=4"
testingProcedure <- dbGetQuery(dbConnection, dbquery)
print(testingProcedure)

##   rid origin airline aircraft flight.phase   dep.date conditions altitude
## 1   4   5555    5555 Aircraft      Takeoff 2002-09-09          1      100
##   warned
## 1      0
```

[Question 12] This code chunk attempts to insert the new record in incident table using procedure. Here the conditions values hold a invalid value which is NOT present in the look up table, so the insertion fails.

```
CALL insertIncidentProcedure(76589,'Aircraft','Takeoff','2002-09-
09',0,0,999,5555,5555);
```

[Question 12] This code chunk tests if the procedure call just above was successful or NOT. Since invalid values were passed, this should return 0 zero rows as the record we were expecting to be present in the database was not inserted at all.

```
dbquery <- "SELECT * FROM incidents WHERE rid=76589"
testingProcedure <- dbGetQuery(dbConnection, dbquery)
print(testingProcedure)

## [1] rid          origin       airline      aircraft     flight.phase
## [6] dep.date     conditions   altitude     warned
## <0 rows> (or 0-length row.names)
```

This code chunk disconnects the database

```
dbDisconnect(dbConnection)
```