Practicum I CS5200 Full Summer

```
# Packages used for Practicum I
packages <- c("base","datasets","DBI","graphics","grDevices","methods", "RMySQL",
"stats","utils")

# Install packages not yet installed
installed_packages <- packages %in% rownames(installed.packages())
if (any(installed_packages == FALSE)) {
  install.packages(packages[!installed_packages])
}

# Packages loading
invisible(lapply(packages, library, character.only = TRUE))

# Name: connectToDB
# Description: Connects to remote SQL server limit 5mb
# Parameters: N/A
# Returns: The DB

connectToDb <- function () {

db_name_fh <- "sql9628267"
db_user_fh <- "sql9628267"
db_host_fh <- "sql9.freemysqlhosting.net"
db_pwd_fh <- "ke4WFIpEZG"
db_port_fh <- 3306

# 3. Connect to remote server database
mydb.fh <-  dbConnect(RMySQL::MySQL(), user = db_user_fh, password = db_pwd_fh,
                      dbname = db_name_fh, host = db_host_fh, port = db_port_fh)

mydb <- mydb.fh

return(mydb)

}
```

```
# Name: connectToDB
# Description: Connects to local SQL server limit
# Parameters: N/A
# Returns: The DB

connectToLocalDB <- function() {
  db_name_fh <- "test"
  db_user_fh <- "root"
  db_host_fh <- "localhost"
  db_pwd_fh <- "password"
  db_port_fh <- 3306

  # Connect to the MySQL server
  mydb <- dbConnect(RMySQL::MySQL(), user = db_user_fh, password = db_pwd_fh,
                    dbname = db_name_fh, host = db_host_fh, port = db_port_fh)


  return(mydb)
}

# remote db
#mydb <- connectToDb()

# local db will use for demo because of limits
mydb <- connectToLocalDB()


# Creating db locally for assignment
CREATE DATABASE IF NOT EXISTS birdStrikes;


# allows to read from csv file
SET GLOBAL local_infile = TRUE;

# Name: dropAirportsTable
# Description: Drops airport table
# Parameters: db connection
# Returns: N/A
```

```
dropAirportsTable <- function(db) {
  createDropStatement <- "DROP TABLE IF EXISTS airports;"
  dbExecute(db,createDropStatement)
}

createAirportsTable <- function(db) {

  ## 4B Create table airports

  # Name: createAirportsTable
  # Description: creates airport table
  # Parameters: db connection
  # Returns: N/A

  createTableStatement <- "CREATE TABLE airports (
    aid INTEGER AUTO_INCREMENT PRIMARY KEY,
    airportState TEXT,
    airportCode TEXT
  )"

  dbExecute(db, createTableStatement)
}

  # Name: dropConditionsTable
  # Description: drops conditions table
  # Parameters: db connection
  # Returns: N/A

dropConditionsTable <- function(db) {
  createDropStatement <- "DROP TABLE IF EXISTS conditions;"
  dbExecute(db,createDropStatement)
}

createConditionsTable <- function(db) {

  ## 4D Create look-up table conditions

  # Name: createConditionsTable
```

```
  # Description: creates conditions table
  # Parameters: db connection
  # Returns: N/A

  createTableStatement <- " CREATE TABLE conditions (
    cid INTEGER AUTO_INCREMENT PRIMARY KEY,
    sky_condition TEXT(255),
    explanation TEXT(255)
  )"

  dbExecute(db, createTableStatement)
}

  # Name: dropFlightsTable
  # Description: drops flights table
  # Parameters: db connection
  # Returns: N/A

dropFlightsTable <- function(db) {
  createDropStatement <- "DROP TABLE IF EXISTS flights;"
  dbExecute(db,createDropStatement)
}

createFlightsTable <- function(db) {

  ## 4A Create table flights
  ## 4C link flights with airport

  # Name: createFlightsTable
  # Description: create flights table
  # Parameters: db connection
  # Returns: N/A

  createTableStatement <- " CREATE TABLE flights (
  fid INTEGER AUTO_INCREMENT PRIMARY KEY,
  date DATE,
  origin INTEGER NOT NULL,
  airline TEXT,
```

```
  aircraft TEXT,
  altitude INTEGER CHECK (altitude >= 0),
  heavy BOOLEAN,
  FOREIGN KEY (origin) REFERENCES airports (aid)
  )"

  dbExecute(db, createTableStatement)
}

  # Name: dropStrikesTable
  # Description: drops strikes table
  # Parameters: db connection
  # Returns: N/A

dropStrikesTable <- function(db) {
  createDropStatement <- "DROP TABLE IF EXISTS strikes;"
  dbExecute(db,createDropStatement)
}

createStrikesTable <- function(db) {

  ## 4E Create table strikes
  ## 4F Link strikes and flights

  # Name: createStrikesTable
  # Description: creates strikes table
  # Parameters: db connection
  # Returns: N/A

  createTableStatement <- " CREATE TABLE strikes (
    sid INTEGER AUTO_INCREMENT PRIMARY KEY,
    fid INTEGER NOT NULL,
    numbirds INTEGER,
    impact TEXT,
    damage BOOLEAN,
    altitude INTEGER CHECK (altitude >= 0),
    conditions INTEGER NOT NULL,
    FOREIGN KEY (conditions) REFERENCES conditions (cid),
```

```
   FOREIGN KEY (fid) REFERENCES flights (fid)
  )"

  dbExecute(db, createTableStatement)
}

# creating tables for 4G showing tables

  # Name: testingTables
  # Description: creates tables to show they are working
  # Parameters: N/A
  # Returns: N/A

dropStrikesTable(mydb)

## [1] 0

dropFlightsTable(mydb)

## [1] 0

dropConditionsTable(mydb)

## [1] 0

dropAirportsTable(mydb)

## [1] 0

createAirportsTable(mydb)

## [1] 0

createConditionsTable(mydb)

## [1] 0

createFlightsTable(mydb)

## [1] 0

createStrikesTable(mydb)
```

```
## [1] 0


#4G Test table definitions

# Name: airportInsert
# Description: inserts into airport table
# Parameters: N/A
# Returns: N/A

INSERT INTO airports (airportState, airportCode)
VALUES ('IL', 'JFK'),
      ('CA', 'LAX'),
      ('MA', 'BOS');


# Name: airportSelect
# Description: selects all records from airport table
# Parameters: N/A
# Returns: N/A

SELECT * FROM airports;


# Name: airportDelete
# Description: deletes one record with the pk of 2.
# Parameters: N/A
# Returns: N/A

DELETE FROM airports WHERE aid = 2;


# Name: airportSelect
# Description: selects all records for airports table
# Parameters: N/A
# Returns: N/A

SELECT * FROM airports;
```

```
#4G Test table definitions

# Name: airportInsert
# Description: inserts into airports table
# Parameters: N/A
# Returns: N/A

INSERT INTO airports (airportState, airportCode)
VALUES ('Testing', 'Testing')


#4G Test table definitions

# Name: conditionsInsert
# Description: inserts into conditions table
# Parameters: N/A
# Returns: N/A

INSERT INTO conditions (sky_condition, explanation)
VALUES ('Clear', "testing"),
      ('Cloudy', "testing"),
      ('Rainy', "testing");


# Name: conditionsSelect
# Description: selects all from conditions table
# Parameters: N/A
# Returns: N/A

SELECT * FROM conditions;


#4G Test table definitions

# Name: flightsInsert
# Description: inserts into flights table
# Parameters: N/A
```

```
# Returns: N/A

INSERT INTO flights ( date, origin, airline, aircraft, altitude, heavy)
VALUES ('2023-06-01', 1, 'Delta Air Lines', 'Boeing 737', 35000, TRUE),
      ('2023-06-02', 3, 'United Airlines', 'Airbus A320', 28000, FALSE),
      ('2023-06-03', 4, 'American Airlines', 'Boeing 777', 32000, TRUE);


# Name: flightsSelect
# Description: selects all from flights table
# Parameters: N/A
# Returns: N/A

SELECT * FROM flights;


#4G Test table definitions

# Name: strikesInsert
# Description: inserts into strikes table
# Parameters: N/A
# Returns: N/A

INSERT INTO strikes (fid, numbirds, impact, damage, altitude, conditions)
VALUES (1, 3000, 'Engine ingestion', TRUE, 32000, 1),
      (2, 2000, 'Windshield strike', FALSE, 25000, 2),
      (3, 1000, 'Wing collision', TRUE, 31000, 3);


#4G Test table definitions

# Name: strikesSelect
# Description: selects all from strikes table
# Parameters: N/A
# Returns: N/A

SELECT * FROM strikes;
```

0 records

sid	fid	numbirds	impact	damage	altitude	conditions

```
# Name: readingCsvFile
# Description: reads into csv file
# Parameters: N/A
# Returns: bds.raw: dataframe

# Read all data from the csv file
bds.raw <- read.csv("birdStrikesData-V2.csv", header = TRUE)

# Name: readingCsvFile
# Description: reads a subset of csv file
# Parameters: N/A
# Returns: subset_data: dataframe

subset_size <- 4

# Create a subset of data
subset_data <- bds.raw[sample(nrow(bds.raw), subset_size), ]

# Name: prepForCsvExtraction
# Description: deleting test data and re-creating for csv extraction
# Parameters: N/A
# Returns: N/A


dropStrikesTable(mydb)

## [1] 0

dropFlightsTable(mydb)

## [1] 0

dropConditionsTable(mydb)

## [1] 0
```

```
dropAirportsTable(mydb)

## [1] 0

createAirportsTable(mydb)

## [1] 0

createConditionsTable(mydb)

## [1] 0

createFlightsTable(mydb)

## [1] 0

createStrikesTable(mydb)

## [1] 0

# Name: populateAirportsTableFromCsv
# Description: populates data from csv and inserts into airport table
# Parameters: N/A
# Returns: N/A

populateAirportsTableFromCsv <- function(db, airportDataFrame) {

  table_name <- "airports"

  # creating data frame using ifelse to error check data
  data_selected <- data.frame(
    airportState = ifelse(airportDataFrame$origin == "N/A", "UNKNOWN",
airportDataFrame$origin)
  )

  # Making airportCode NULL
  data_selected$airportCode <- "NULL"

  # Adding aid(PK) which are sequential values 1-N
  data_selected$aid <- seq_len(nrow(data_selected))
```

```
  # append data into target table
  dbWriteTable(db, table_name, data_selected, append = TRUE, row.names = FALSE)
}

# Name: populateAirportsTableFromCsv
# Description: running function
# Parameters: N/A
# Returns: N/A

populateAirportsTableFromCsv(mydb, bds.raw)

## [1] TRUE


# Name: airportSelect
# Description: showing data from airport table after extraction
# Parameters: N/A
# Returns: N/A

SELECT * FROM airports;
```

Displaying records 1 - 10

aid	airportState	airportCode
1	New York	NULL
2	Texas	NULL
3	Louisiana	NULL
4	Washington	NULL
5	Virginia	NULL
6	UNKNOWN	NULL
7	Delaware	NULL
8	DC	NULL
9	Georgia	NULL
10	Florida	NULL

```
# Name: populateConditionsTableFromCsv
# Description: populates data from csv and inserts into conditions table
# Parameters: N/A
# Returns: N/A

populateConditionsTableFromCsv <- function(db, dataFrame) {

table_name <- "conditions"

# creating data frame using ifelse to error check data
data_selected <- data.frame(
   sky_condition = dataFrame$sky_conditions,
   explanation = ifelse(dataFrame$Remarks == "", "UNKNOWN", dataFrame$Remarks)
)

 # Adding cid(PK) which are sequential values 1-N
data_selected$cid <- seq_len(nrow(data_selected))

# append data into target table
```

```
dbWriteTable(db, table_name, data_selected, append = TRUE, row.names = FALSE)

}

populateConditionsTableFromCsv(mydb, bds.raw)

## [1] TRUE


# Name: conditionsSelect
# Description: showing data from conditions table after extraction
# Parameters: N/A
# Returns: N/A

SELECT * FROM conditions;
```

Displaying records 1 - 10

cid	sky_condition	explanation
1	No Cloud	FLT 753. PILOT REPTD A HUNDRED BIRDS ON UNKN TYPE. #1 ENG WAS SHUT DOWN AND DIVERTED TO EWR. SLIGHT VIBRATION. A/C WAS OUT OF SVC FOR REPAIRS TO COWLING, FAN DUCT ACCOUSTIC PANEL. INGESTION. DENTED FAN BLADE #26 IN #1 ENG. HEAVY BLOOD STAINS ON L WINGTIP
2	Some Cloud	102 CARCASSES FOUND. 1 LDG LIGHT ON NOSE GEAR WAS DAMAGED AND REPLACED.
3	No Cloud	FLEW UNDER A VERY LARGE FLOCK OF BIRDS OVER APCH END OF RWY. NO DMG. JUST A LOT OF BIRD DROPPINGS ON WINDSCREEN.
4	Some Cloud	NOTAM WARNING. 26 BIRDS HIT THE A/C, FORCING AN EMERGENCY LDG. 77 BIRDS WERE FOUND DEAD ON RWY/TWY WITH GRASSHOPPERS IN THEIR STOMACHS. SAFETY AREAS COULD NOT BE THOROUGHLY INSPCTD DURING 14 MINUTE SHUTDOWN OF RWY 34L. NO DMG. A/C OUT OF SVC 40 MINS. PHOT

cid	sky_condition	explanation
5	No Cloud	NO DMG REPTD.
6	No Cloud	NO DMG. BIRD REMAINS ON F/O WINDSCREEN.
7	No Cloud	UNKNOWN
8	Some Cloud	WS ASSISTED IN CLEAN-UP OF 273 STARLINGS AND 1 BROWN-HEADED COWBIRD FROM RWY THRESHOLD. PHOTOS OF A/C TAKEN. BORESCOPED BOTH ENGS. FOUND DENTS AND NICKS IN STAGES 3-6. ALL WITHIN LIMITS. CLEANED RADOME, L WING, FLAPS, PYLON, GEAR AND LEADING EDGE FLAPS. R

cid	sky_condition	explanation
9	Some Cloud	UNKNOWN
10	Some Cloud	FLT 057

```
# Name: populateFlightsTableFromCsv
# Description: populates data from csv and inserts into flights table
# Parameters: N/A
# Returns: N/A
populateFlightsTableFromCsv <- function(db, dataFrame) {

table_name <- "flights"

# Create data frame
dataFrame$flight_date <- as.Date(dataFrame$flight_date, format = "%m/%d/%Y")

data_selected <- data.frame(
  date = dataFrame$flight_date,
  airline = ifelse(is.na(dataFrame$airline), "UNKNOWN",dataFrame$airline),
  aircraft = ifelse(is.na(dataFrame$aircraft), "UNKNOWN",dataFrame$aircraft),
```

```
  altitude = ifelse(is.na(dataFrame$altitude_ft), 0,dataFrame$aircraft),
  heavy = ifelse(tolower(dataFrame$heavy_flag) == "yes", 1, 0)
)

 # Adding fid(PK) which are sequential values 1-N
data_selected$fid <- seq_len(nrow(data_selected))
 # Adding origin(FK) which are sequential values 1-N
data_selected$origin <-seq_len(nrow(data_selected))

# append data into target table
dbWriteTable(db, table_name, data_selected, append = TRUE, row.names = FALSE)
}

# Name: populateFlightsTableFromCsv
# Description: populates data from csv and inserts into flights table
# Parameters: N/A
# Returns: N/A
populateFlightsTableFromCsv(mydb, bds.raw)

## [1] TRUE


# Name: flightsSelect
# Description: showing data from flights table after extraction
# Parameters: N/A
# Returns: N/A

SELECT * FROM flights;
```

Displaying records 1 - 10

fid	date	origin	airline	aircraft	altitude	heavy
1	2000-11-23	1	US AIRWAYS*	Airplane	0	1
2	2001-07-25	2	AMERICAN AIRLINES	Airplane	0	0
3	2001-09-14	3	BUSINESS	Airplane	0	0
4	2002-09-05	4	ALASKA AIRLINES	Airplane	0	1
5	2003-06-23	5	COMAIR AIRLINES	Airplane	0	0
6	2003-07-24	6	AMERICAN AIRLINES	Airplane	0	0
7	2003-08-17	7	BUSINESS	Airplane	0	0
8	2006-03-01	8	UNITED AIRLINES	Airplane	0	0
9	2000-01-06	9	AIRTRAN AIRWAYS	Airplane	0	0
10	2000-01-07	10	AIRTOURS INTL	Airplane	0	0

```
# Name: populateStrikesTableFromCsv
# Description: populates data from csv and inserts into strikes table
# Parameters: N/A
# Returns: N/A
populateStrikesTableFromCsv <- function(db, dataFrame) {
  table_name <- "strikes"

  # create data frame
  data_selected <- data.frame(
    numbirds = ifelse(is.na(dataFrame$wildlife_struck), 0,dataFrame$wildlife_struck),
    impact = ifelse(is.na(dataFrame$impact), "UNKNOWN",dataFrame$impact),
    damage = ifelse(tolower(dataFrame$damage) == "no damage", 0, 1),
    altitude = ifelse(is.na(dataFrame$altitude_ft), 0, dataFrame$altitude_ft)
  )
  # Creating PK and FK for table
  data_selected$sid <- seq_len(nrow(data_selected))
  data_selected$fid <- seq_len(nrow(data_selected))
  data_selected$conditions <- seq_len(nrow(data_selected))

  dbWriteTable(db, table_name, data_selected, append = TRUE, row.names = FALSE)
}
```

```
# Name: populateStrikesTableFromCsv
# Description: populates data from csv and inserts into strikes table
# Parameters: N/A
# Returns: N/A
populateStrikesTableFromCsv(mydb, bds.raw)
```

```
## [1] TRUE
```

```
# Name: strikesSelect
# Description: showing data from strikes table after extraction
# Parameters: N/A
# Returns: N/A
SELECT * FROM strikes;
```

Displaying records 1 - 10

sid	fid	numbirds	impact	damage	altitude	conditions
1	1	859	Engine Shut Down	1	1	1
2	2	424	None	1	0	2
3	3	261	None	0	50	3
4	4	806	Precautionary Landing	0	50	4
5	5	942	None	0	50	5
6	6	537	None	0	0	6
7	7	227	Other	1	150	7
8	8	320	Other	1	100	8
9	9	9	Aborted Take-off	0	0	9
10	10	4	None	0	0	10

```
# description
# 1: Select statement - selecting the airportState and counting the number of
# incidents labeled as num_incidents
# 2: Selecting from airports table
# 3: Joining airports table and flights table into a temp table
# 4: Joining flights table and strikes table into a temp table
# 5: Group data by the state
# 6: Order by the number of incidents in descending order
# 7: Limit to the first 10 records

query <- "SELECT airportState, COUNT(*) AS num_incidents
        FROM airports
        JOIN flights ON airports.aid = flights.origin
        JOIN strikes ON flights.fid = strikes.fid
        GROUP BY airportState
        ORDER BY num_incidents DESC
        LIMIT 10;"
result <- dbGetQuery(mydb, query)
```

```
print(result)
```

```
##    airportState num_incidents
## 1    California          2520
## 2         Texas          2453
## 3       Florida          2055
## 4      New York          1319
## 5      Illinois          1008
## 6  Pennsylvania           986
## 7      Missouri           960
## 8      Kentucky           812
## 9          Ohio           778
## 10       Hawaii           729
```

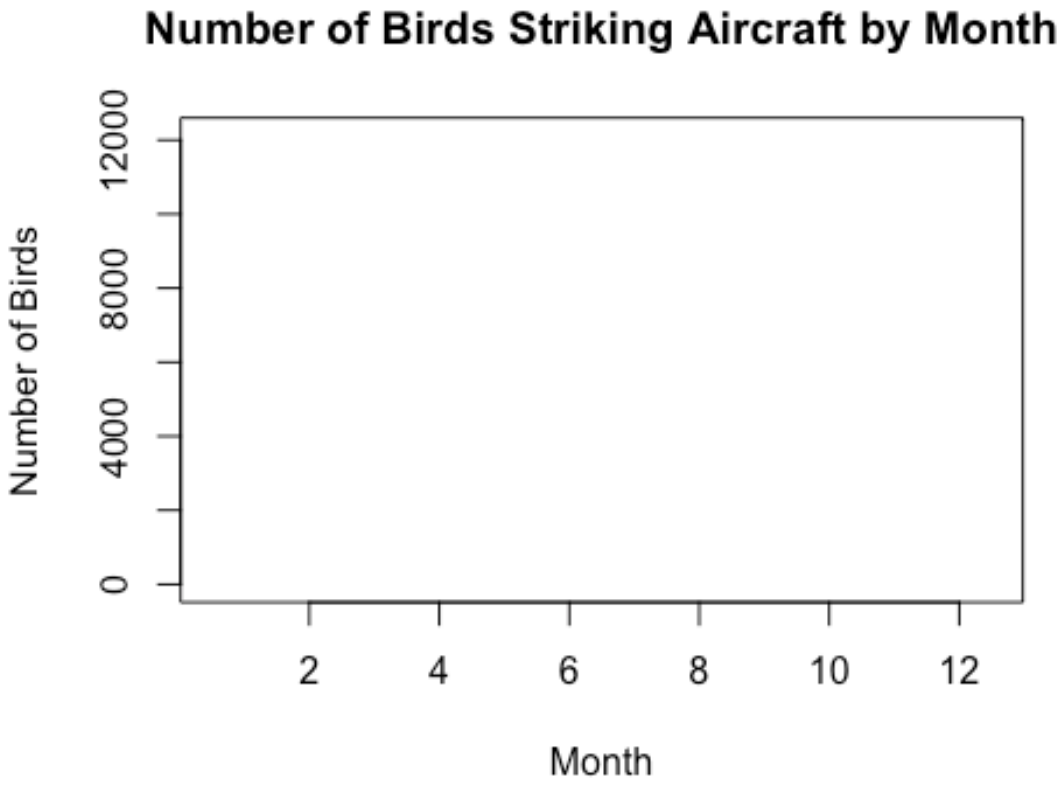
```
# 1: Selects airline and will count the incidents as num_incidents
# 2: Selecting from flights
# 3: Joining both tables into a temporary table
# 4: group by airline
# 5C: Find the total number of incidents for each airline
# 5B: Find the average of those incidents
# 5A: Find each record that is greater than the average.
# 6: Order incidents in descending order

query <- "SELECT airline, COUNT(*) AS num_incidents
        FROM flights
        JOIN strikes ON flights.fid = strikes.fid
        GROUP BY airline
        HAVING COUNT(*) > (SELECT AVG(incident_count) FROM (SELECT COUNT(*) AS
incident_count FROM flights JOIN strikes ON flights.fid = strikes.fid GROUP BY airline) AS
subquery)
        ORDER BY num_incidents DESC;"
result <- dbGetQuery(mydb, query)

print(result)
```

```
##                      airline num_incidents
## 1          SOUTHWEST AIRLINES          4628
```

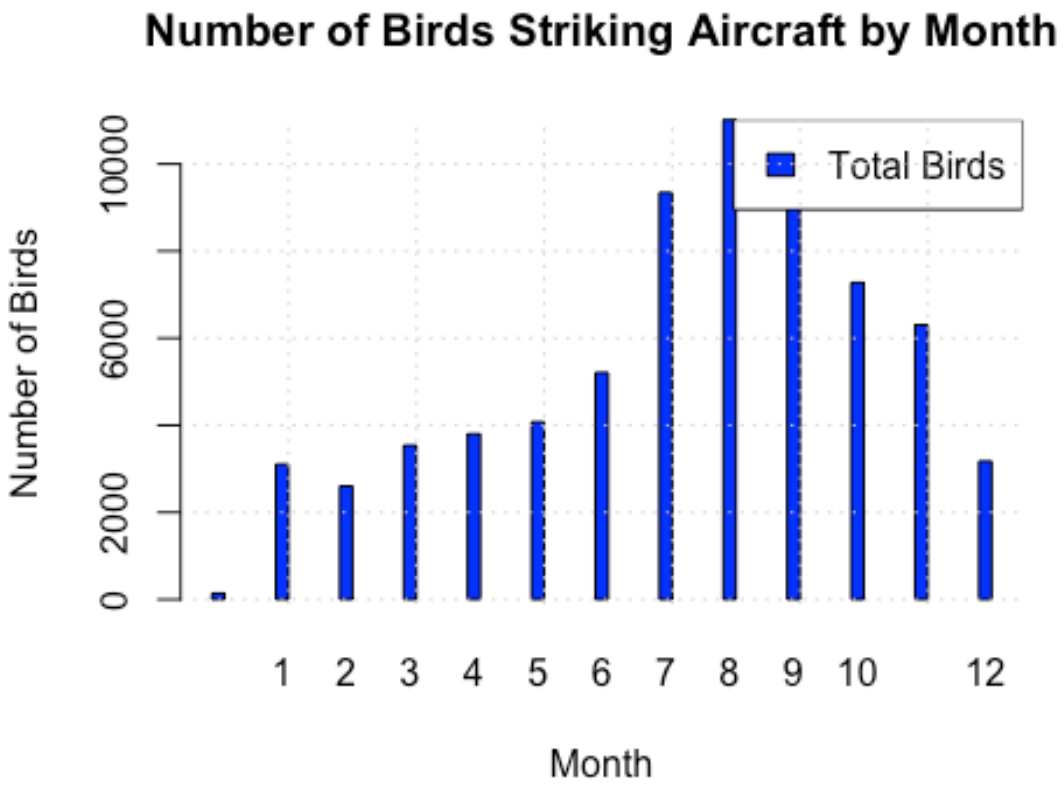
```
## 2                    BUSINESS         3074
## 3            AMERICAN AIRLINES        2058
## 4              DELTA AIR LINES        1349
## 5       AMERICAN EAGLE AIRLINES         932
## 6            SKYWEST AIRLINES         891
## 7                US AIRWAYS*          797
## 8              JETBLUE AIRWAYS        708
## 9                UPS AIRLINES         590
## 10               US AIRWAYS           540
## 11             UNITED AIRLINES        506
## 12          NORTHWEST AIRLINES        458
## 13           FRONTIER AIRLINES        416
## 14             AIRTRAN AIRWAYS        414
## 15             PRIVATELY OWNED        390
## 16                 PINNACLE           379
## 17 EXPRESSJET (CONTINENTAL EXPRS)         368
## 18         CONTINENTAL AIRLINES         345
## 19         ATLANTIC SOUTHEAST           338
## 20               HAWAIIAN AIR           332
## 21             COMAIR AIRLINES          317
## 22             ALASKA AIRLINES          304
## 23              FEDEX EXPRESS           280
## 24              MESA AIRLINES           259
## 25       AIR WISCONSIN AIRLINES         229
## 26             MESABA AIRLINES          221
## 27               HORIZON AIR            196
## 28                 ABX AIR              189
## 29         CHAUTAUQUA AIRLINES          186
## 30               PSA AIRLINES           182
## 31                GOVERNMENT            159
## 32        GREAT LAKES AIRLINES          158
## 33       AMERICA WEST AIRLINES          157
## 34             SPIRIT AIRLINES          140
## 35              ALOHA AIRLINES          135
## 36                                      129
## 37               ISLAND AIR             122
## 38          REPUBLIC AIRLINES           120
```

```
## 39       ATLANTIC COAST AIRLINES          115
## 40            PIEDMONT AIRLINES          113
## 41               ALLEGIANT AIR           107
## 42                 AIR CANADA            95
## 43                  COMMUTAIR            92
## 44                   MILITARY            89

# 1: Select the month from date ie 2000-11-23 -> 11 and sum the number of bird
#    strikes as total_birds
# 2: Selecting from strikes
# 3: Join both tables
# 4: Group by the month
# 5: Order by month in ascending order

# N/A is dates that are missing,

query <- "SELECT MONTH(date) AS month, CAST(SUM(numbirds) AS SIGNED INTEGER)
AS total_birds
        FROM strikes
        JOIN flights ON strikes.fid = flights.fid
        GROUP BY month
        ORDER BY month;"


result <- dbGetQuery(mydb, query)

head(result, 6)

##   month total_birds
## 1    NA         141
## 2     1        3106
## 3     2        2602
## 4     3        3539
## 5     4        3802
## 6     5        4077

# basic setup up for graph
plot(result$month, result$total_birds, type = "n", xlab = "Month", ylab = "Number of Birds",
```

```
    main = "Number of Birds Striking Aircraft by Month", ylim = c(0,
max(result$total_birds)*1.1),
    xlim = c(0.5, 12.5))  # Adjusted x-axis limits
```




```
# structure for the graph
barplot(result$total_birds, names.arg = result$month, col = "blue", border = "black",
        xlab = "Month", ylab = "Number of Birds", main = "Number of Birds Striking Aircraft by
Month", space = 4)


# Legend
legend("topright", legend = "Total Birds", fill = "blue")

# grid lines on
grid()
```




```
# Deleting data to show procedure
dropStrikesTable(mydb)
```

```
## [1] 0
```

```
createStrikesTable(mydb)
```

```
## [1] 0
```

```
DROP PROCEDURE IF EXISTS AddBirdStrike;


# 1: IN requires all the fields to create a new strike
# 2: DECLARE the PK/FKs needed
# 3: Insert into tables
```

```
# 4: LAST_INSERT_ID() returns the last auto-increment
# 5:


CREATE PROCEDURE AddBirdStrike (
  IN date DATE,
  IN airline TEXT,
  IN aircraft TEXT,
  IN altitude INT,
  IN heavy BOOLEAN,
  IN num_birds INT,
  IN impact TEXT,
  IN damage BOOLEAN,
  IN sky_condition TEXT,
  IN explanation TEXT,
  IN airportState TEXT,
  IN airportCode TEXT
)
BEGIN
  DECLARE aid INT;
  DECLARE cid INT;
  DECLARE fid INT;

  INSERT IGNORE INTO airports (airportState, airportCode)
  VALUES (airportState, airportCode);

  SET aid = LAST_INSERT_ID();

  INSERT IGNORE INTO flights (date, origin, airline, aircraft, altitude, heavy)
  VALUES (date, aid, airline, aircraft, altitude, heavy);

  SET fid = LAST_INSERT_ID();

  INSERT IGNORE INTO conditions (sky_condition, explanation)
  VALUES (sky_condition, explanation);

  SET cid = LAST_INSERT_ID();
```

```
  INSERT INTO strikes (fid, numbirds, impact, damage, altitude, conditions)
  VALUES (fid, num_birds, impact, damage, altitude, cid);


END;

statement <- sprintf("CALL AddBirdStrike('%s', '%s', '%s', %d, %s, %d, '%s', %s, '%s', '%s',
'%s', '%s')",
                     "2023-06-01",
                     "TESTING",
                     "TESTING",
                     35000,
                     "TRUE",
                     3000,
                     "TESTING",
                     "TRUE",
                     "Normal",
                     "Explanation",
                     "New York",
                     "JFK")

# Execute the stored procedure
dbExecute(mydb, statement)

## [1] 1


SELECT * FROM strikes
```

1 records

sid	fid	numbirds	impact	damage	altitude	conditions
1	25559	3000	TESTING	1	35000	25559

```
dropStrikesTable(mydb)
```

```
## [1] 0
dropFlightsTable(mydb)
## [1] 0
dropConditionsTable(mydb)
## [1] 0
dropAirportsTable(mydb)
## [1] 0
  dbDisconnect(mydb)
## [1] TRUE
```