Practicum I CS5200

Omer Koc, Etki Acilan

Spring 2023

Required Libraries

```
# Downloading required libraries, comment-out to download
#install.packages("RMySQL")
#install.packages("DBI")
```

Step 3:Connect to Database

```
# 1. Library (must be installed prior to loading)
library(RMySQL)     ### MySQL
```

```
## Loading required package: DBI
```

```
library(DBI)     ### DBI

# 2. My SQL Settings
db_user <- 'acilankoc'
db_password <- 'acilankoc123'
db_name <- 'dbacilankoc'
db_host <- 'db4free.net'
db_port <- 3306 # always this port unless you change it during installation

# 3. Connect to DB (mydb)
mydb = dbConnect(MySQL(), user = db_user, password = db_password,
                 dbname = db_name, host = db_host, port = db_port)
```

Step 4: Create Database.

```
# The existence of the table is checked, if any, it is dropped.
DROP TABLE IF EXISTS incidents
```

```
# The existence of the table is checked, if any, it is dropped.
DROP TABLE IF EXISTS airports
```

```
# The existence of the table is checked, if any, it is dropped.
DROP TABLE IF EXISTS conditions
```

```
# The existence of the table is checked, if any, it is dropped.
DROP TABLE IF EXISTS airlines
```

E.

```
# Create the airlines table.
# airlines (eid : integer, airlineName : text)
# airlineCode TEXT, this is not necessary for this database.
# flag TEXT, this is not necessary for this database.
CREATE TABLE IF NOT EXISTS airlines(
# eid is a synthetic primary key,
# airlineName corresponds to the airline column from the data file
    -- Columns
    eid INTEGER PRIMARY KEY,
    airlineName TEXT
);
```

D.

```
# Create the conditions lookup table.
# conditions (cid, condition)
# explanation TEXT, this is not necessary for this database
CREATE TABLE IF NOT EXISTS conditions(
    -- Columns
    cid INTEGER PRIMARY KEY,
    `condition` TEXT
);
```

B.

```
# Create the airports table.
# airports (aid : integer, airportName : text, state : text)
# airports international code is not specified as it is not required
CREATE TABLE IF NOT EXISTS airports(
    -- Columns
    aid INTEGER PRIMARY KEY,
    airportName TEXT,
    state TEXT
);
```

A.

```
# Create the Incidents table.
# Each feature value and data type given in the Practicum I have been set.
# Flight phase and altitude variables are specified in the conditions section.
# Question C, D and F are listed at the bottom as Foreign key relations
CREATE TABLE IF NOT EXISTS incidents(
    -- Columns
    rid INTEGER PRIMARY KEY,
    `dep.date` DATE,
    origin INTEGER,
    airline INTEGER,
    aircraft TEXT,
    `flight.phase` TEXT,
    altitude INTEGER,
    conditions INTEGER,
    warned BOOLEAN,
    -- Conditions
    CHECK (`flight.phase` IN ('takeoff', 'landing', 'inflight', 'unknown')),
    CHECK (altitude >= 0),
    -- Foreign key relations
    FOREIGN KEY (origin) REFERENCES airports(aid), /*Links incidents table via origin FK, to airports table via a
id PK. // C */
    FOREIGN KEY (conditions) REFERENCES conditions(cid), /*Links incidents table via conditions FK, to conditions
(look-up) table via cid PK. // D*/
    FOREIGN KEY (airline) REFERENCES airlines(eid) /*Links incidents table via airline FK, to airlines table via
eid PK. // F*/
);
```

G. Test Chunk

```
# With the DESCRIBE command, each field of the incidents table and the necessary information about that field are
printed
# This shows us that the incidents table we selected for testing is correctly defined.
DESCRIBE incidents
```

Step 5: Read csv file.

```
# Load it into a dataframe called bds.raw
bds.raw <- read.csv("BirdStrikesData-V2.csv")
```

Step 6:Data manipulation

```
# Delete unnecessary data.
bds.raw <- bds.raw[ , c("rid","airport","aircraft","flight_date","airline","origin","flight_phase","sky_condition
s","pilot_warned_flag","altitude_ft")]
```

```
# Delete no DATE data.
sample_idx <- which(bds.raw$flight_date == "" | bds.raw$flight_date == "N/A")
bds.raw = bds.raw[-sample_idx,]
```

```r
# Update the date data type in the flight date
bds.raw$flight_date <- as.Date(bds.raw$flight_date, format = "%m/%d/%Y %H:%M")
#bds.raw$flight_date <-format(bds.raw$flight_date, "%m/%d/%Y")
```

```r
# Assign 'Unknown' to those with N/A in airline and airport tables.
# Also, assign 'Unknown' to those with "" (no space) in airline and airport tables.
bds.raw$airline[which(bds.raw$airline == "N/A" | bds.raw$airline == "")] <- "Unknown"
bds.raw$airport[which(bds.raw$airport == "N/A"| bds.raw$airport == "")] <- "Unknown"
```

Populate airports table.

```r
# Find unique airport names.
unique_airports <- unique(bds.raw$airport)
# Set a sample_idx list of the unique airport names to retrieve their unique origin later.
sample_idx <- which(!duplicated(bds.raw$airport))
# Retrieve origins.
origins_of_airports <- bds.raw$origin[sample_idx]
# If there is a NULL airportName, assign "Unknown".
unique_airports[which(unique_airports == "N/A")] <- "Unknown"
# If there is a NULL origin, assign "Unknown".
origins_of_airports[which(origins_of_airports == "N/A")] <- "Unknown"
```

Batch approach

```r
# Insert required columns into airports table
# Populate lookup table using unique_airports values
# By using gsub(), avoided the problem of quotation marks (') in some airport names.

batch <- list()
for(i in 1:length(unique_airports)){
  aid_synth <- i
  values <- paste0("(", aid_synth, ",'", gsub("'", "''", unique_airports[i]), "','", origins_of_airports[i], "')"
)
  batch <- c(batch, values)
}

dbExecute(mydb, paste0("INSERT INTO airports (aid, airportName, state) VALUES ", paste(batch, collapse = ",")))
```

```
## [1] 1109
```

Populate airlines table.

```r
# Find unique airline names.
unique_airline <- unique(bds.raw$airline)
# If there is a NULL airline name, assign "Unknown".
# If there is a no space("") airline name, assign "Unknown".
unique_airline[which(unique_airline == "N/A" | unique_airline == "")] <- "UNKNOWN"
```

Batch approach

```r
batch <- list()
for(i in 1:length(unique_airline)){
  aid_synth <- i
  values <- paste0("(", aid_synth, ",'", gsub("'", "''", unique_airline[i]), "')")
  batch <- c(batch, values)
}

dbExecute(mydb, paste0("INSERT INTO airlines (eid, airlineName) VALUES ", paste(batch, collapse = ",")))
```

```
## [1] 292
```

Populate conditions table.

```r
# Find unique conditions.
unique_sky_conditions <- unique(bds.raw$sky_conditions)
```

Batch approach

```
# Insert required columns into conditions table
# Populate lookup table using unique_sky_conditions values
# By using gsub(), avoided the problem of quotation marks (') in some sky conditions
batch <- list()
for(i in 1:length(unique_sky_conditions)){
  aid_synth <- i
  values <- paste0("(", aid_synth, ",'", gsub("'", "''", unique_sky_conditions[i]), "')")
  batch <- c(batch, values)
}

dbExecute(mydb, paste0("INSERT INTO conditions (cid, `condition`) VALUES ", paste(batch, collapse = ",")))
```

```
## [1] 3
```

Data conversion / arrangement.

Pilot Warned Flag

```
# Convert TRUE values to 1 and FALSE values to 0
bds.raw$pilot_warned_flag <- ifelse(bds.raw$pilot_warned_flag == 'Y', 1, 0)
```

Airport Table

```
# Construct the airport table.
airport_table <- data.frame(keys = 1:length(unique_airports), airports = unique_airports, origins_of_airports = o
rigins_of_airports)

# Find the matching values.
idx <- match(bds.raw$airport, airport_table$airports)

# Set the origins to corresponding aid.
bds.raw$origin <- airport_table$keys[idx]
```

Airline Table

```
# Construct the airline table.
airline_table <- data.frame(keys = 1:length(unique_airline), airlineName = unique_airline)

# Find the matching values.
idx <- match(bds.raw$airline, airline_table$airlineName)

# Set the origins to corresponding eid.
bds.raw$airline <- airline_table$keys[idx]
```

Conditions Table

```
# Construct the airline table.
conditions_table <- data.frame(keys = 1:length(unique_sky_conditions), condition = unique_sky_conditions)

# Find the matching values.
idx <- match(bds.raw$sky_conditions, conditions_table$condition)

# Set the origins to corresponding eid.
bds.raw$sky_conditions <- conditions_table$keys[idx]
```

Populate incidents table.

```
# Remove commas in altitude values
bds.raw$altitude_ft <- gsub(",", "", bds.raw$altitude_ft)
# Convert altitude_ft column to integer
bds.raw$altitude_ft <- as.integer(bds.raw$altitude_ft)
```

Map the flight.phase into a predefined set of rule.

```
# takeoff
bds.raw$flight_phase[which(bds.raw$flight_phase == "Take-off run" | bds.raw$flight_phase == "Taxi")] <- "takeoff"

# landing
bds.raw$flight_phase[which(bds.raw$flight_phase == "Landing Roll" | bds.raw$flight_phase == "Parked" | bds.raw$fl
ight_phase == "Descent")] <- "landing"

# inflight
bds.raw$flight_phase[which(bds.raw$flight_phase == "Climb" | bds.raw$flight_phase == "Approach")] <- "inflight"

# unknown
bds.raw$flight_phase[which(bds.raw$flight_phase == "")] <- "Unknown"
```

Batch aproach

```
rid_values <- unique(bds.raw$rid)
batch <- list()
for (i in rid_values) {
  row <- bds.raw[bds.raw$rid == i,]
  values <- paste0("(", row$rid, ",'" , row$flight_date, "','" , row$origin, "','" , row$airline, "','" , row$air
craft, "','" , row$flight_phase, "','" , row$altitude, "','" , row$sky_conditions, "','" , row$pilot_warned_flag,
"')")
  batch <- c(batch, values)
}

dbExecute(mydb, paste0("INSERT INTO incidents (rid, `dep.date`, origin, airline, aircraft, `flight.phase`, altitu
de, conditions, warned) VALUES ", paste(batch, collapse = ",")))
```

```
## [1] 25429
```

Step 7:Checking that data is loaded

```
# Show that loading data works by displaying the first 5 rows of each table
# To display the first 5 rows of the incidents table
SELECT * FROM incidents LIMIT 5;
```

5 records

rid	dep.date	origin	airline	aircraft	flight.phase	altitude	conditions	warned
1195	2002-11-13	37	21	Airplane	inflight	2000	3	1
3019	2002-10-10	706	21	Airplane	inflight	400	1	1
3500	2001-05-15	37	21	Airplane	inflight	1000	1	1
3504	2001-05-23	37	21	Airplane	inflight	1800	1	1
3597	2001-04-18	122	21	Airplane	inflight	200	2	1

```
# To display the first 5 rows of the airports table
SELECT * FROM airports LIMIT 5;
```

5 records

aid	airportName	state
1	LAGUARDIA NY	New York
2	DALLAS/FORT WORTH INTL ARPT	Texas
3	LAKEFRONT AIRPORT	Louisiana
4	SEATTLE-TACOMA INTL	Washington
5	NORFOLK INTL	Virginia

```
# To display the first 5 rows of the conditions table
SELECT * FROM conditions LIMIT 5;
```

3 records

cid	condition
1	No Cloud

2	Some Cloud
3	Overcast

```
# To display the first 5 rows of the airlines table
SELECT * FROM airlines LIMIT 5;
```

5 records

eid	airlineName
1	US AIRWAYS*
2	AMERICAN AIRLINES
3	BUSINESS
4	ALASKA AIRLINES
5	COMAIR AIRLINES

Step 8: 10 states with the greatest number of incidents

```
# Find the 10 states with the greatest number of incidents
# Calculated the total number of incidents of each state with COUNT()
# For this, the airports table is joined to incidents.
# Sorted num_incidents column with ORDER BY
# Limited the top 10 states with LIMIT
SELECT a.state, COUNT(*) AS num_incidents
FROM incidents i
JOIN airports a ON i.origin = a.aid
GROUP BY a.state
ORDER BY num_incidents DESC
LIMIT 10;
```

Displaying records 1 - 10

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

Step 9: Airlines with above-average bird strikes

```
# Find the airlines that had an above average number bird strike incidents.
# Calculated the total number of incidents of each airlines with COUNT()
# Grouped for each airline
# Airways with higher than average bird strikes values were filtered out with the HAVING function.
# Sorted num_incidents column with ORDER BY
SELECT a.airlineName, COUNT(*) AS num_incidents
FROM airlines a
JOIN incidents i ON a.eid = i.airline
GROUP BY a.airlineName
HAVING COUNT(*) > (SELECT AVG(cnt) FROM (SELECT COUNT(*) AS cnt FROM incidents GROUP BY airline) AS counts)
ORDER BY num_incidents DESC;
```

Displaying records 1 - 10

airlineName	num_incidents
SOUTHWEST AIRLINES	4628

BUSINESS	3074
AMERICAN AIRLINES	2058
DELTA AIR LINES	1349
AMERICAN EAGLE AIRLINES	932
SKYWEST AIRLINES	891
US AIRWAYS*	797
JETBLUE AIRWAYS	708
UPS AIRLINES	590
US AIRWAYS	540

Step 10: Number of bird strike incidents by month and by flight phase (across all years).

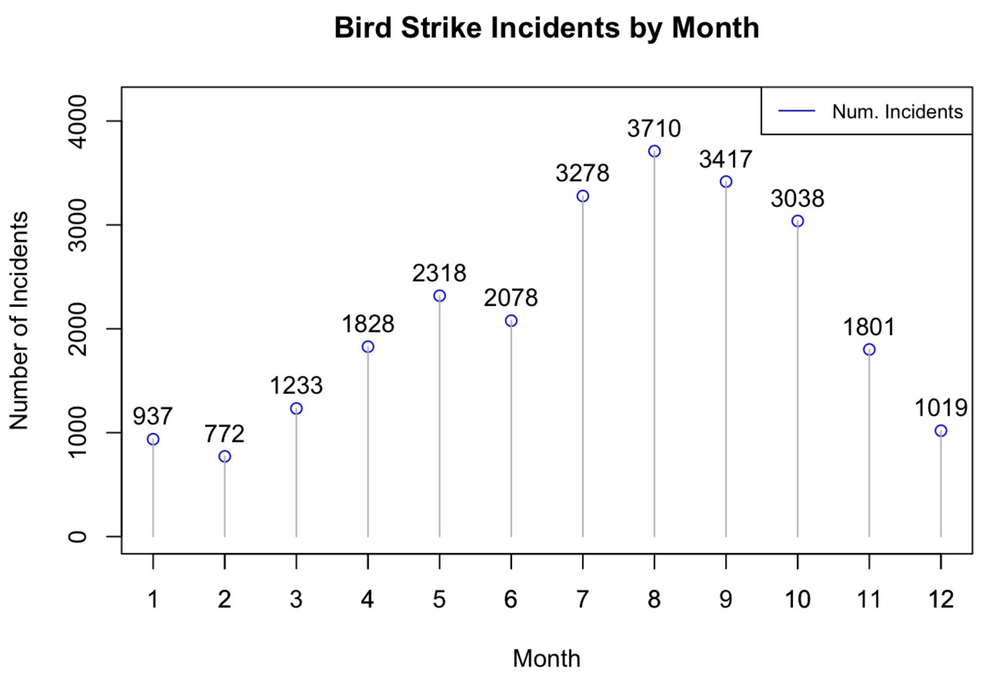
```
# Find the number of bird strike incidents by month and by flight phase (across all years).
# Parsed the month value in dep.date with the MONTH function
# Calculated the total number of incidents with COUNT()
# Grouped for each Month and flight.phase
df_month_strike_sttmnt <- "
SELECT MONTH(`dep.date`) AS Month, `flight.phase`, COUNT(*) AS num_incidents
FROM incidents
GROUP BY Month, `flight.phase`
ORDER BY num_incidents DESC"

# Get Query
df_month_strike <- dbGetQuery(mydb,df_month_strike_sttmnt)

# Print first 6 rows.
head(df_month_strike, 6)
```

1
2
3
4
5
6

6 rows | 1-1 of 4 columns

Step 11: Scatter plot that plots month along the x-axis versus number of incidents

```
# set x and y variables
sum_by_month <- aggregate(num_incidents ~ Month, data = df_month_strike, sum) # Sum of df_month_strike$num_incide
nts by month.

x <- sum_by_month$Month
y <- sum_by_month$num_incidents

# create scatter plot
plot(x, y, type="p", col="blue", ylim=c(0,max(y)+450), xlab="Month", ylab="Number of Incidents", main="Bird Strik
e Incidents by Month")

# add data labels
text(x, y, labels=y, pos=3)
axis(side=1, at=1:length(x), labels=x)

# add vertical line
segments(x, 0, x, y, col="gray")

# add legend
legend("topright", legend=c("Num. Incidents"), col=c("blue"), lty=c(1, 1), cex=0.8)
```



Stored Procedure

```
# Drop procedure if exists
DROP PROCEDURE IF EXISTS add_birdstrike_incident
```

```
# stored procedure to add a bird strike incident
# Although it is a new airport, we also requested the input_Airportname variable
# Each data entry is TEXT/INTEGER as in the data file
# By making the necessary transformations, our data has been made suitable for storage.
CREATE PROCEDURE add_birdstrike_incident (IN `dep.date` DATE,
IN input_origin TEXT,
IN input_AirlineName TEXT,
IN aircraft TEXT,
IN `flight.phase` TEXT,
IN altitude INTEGER,
IN input_Skycondition TEXT,
IN warned BOOLEAN,
IN input_AirportName TEXT)

BEGIN
    /*declared the required id values*/
    DECLARE origin_id INTEGER;
    DECLARE eid_id INTEGER;
    DECLARE cid_id INTEGER;
    DECLARE rid_id INTEGER;

    /*Map the flight.phase into a predefined set of rule.*/
    SET `flight.phase` =
    CASE
        WHEN `flight.phase` = 'Take-off run' OR `flight.phase` = 'Taxi' THEN 'takeoff'
        WHEN `flight.phase` = 'Landing Roll' OR `flight.phase` = 'Parked' OR `flight.phase` = 'Descent' THEN 'lan
ding'
        WHEN `flight.phase` = 'Climb' OR `flight.phase` = 'Approach' THEN 'inflight'
        WHEN `flight.phase` = '' THEN 'Unknown'
        ELSE `flight.phase`
    END;

    /* find the airport */
    SET origin_id = (
        SELECT aid FROM airports WHERE airportName = input_AirportName AND state = input_origin
    );

    /* create a new airport */
    /* determined the id value by adding +1 to the last id value */
    IF origin_id IS NULL THEN
        SET origin_id = ( (SELECT MAX(aid) FROM airports) + 1 );
        INSERT INTO airports (aid, airportName, state) VALUES (origin_id, input_AirportName, input_origin);
    END IF;

    /* Find the airline */
    SET eid_id = (
        SELECT eid FROM airlines WHERE airlineName = input_AirlineName
    );

    /* Create airline */
    /* determined the id value by adding +1 to the last id value */
    IF eid_id IS NULL THEN
        SET eid_id = ( (SELECT MAX(eid) FROM airlines) + 1 );
        INSERT INTO airlines (eid, airlineName) VALUES (eid_id,input_AirlineName);
    END IF;

    /* Find the sky condition */
    SET cid_id = (SELECT cid FROM conditions WHERE conditions.condition = input_Skycondition);

    /* Create sky condition */
    /* determined the id value by adding +1 to the last id value */
    IF cid_id IS NULL THEN
        SET cid_id = ( (SELECT MAX(conditions.cid)+1 FROM conditions) + 1 );
        INSERT INTO conditions (conditions.cid, conditions.condition) VALUES (cid_id, input_Skycondition);
    END IF;

    /* Insert new incident */
    /* determined the id value by adding +1 to the last id value */
    SET rid_id = (SELECT MAX(rid) FROM incidents) + 1;
    INSERT INTO incidents (rid,`dep.date`, origin, airline, aircraft, `flight.phase`, altitude, conditions, warne
d)
    VALUES (rid_id,`dep.date`, origin_id, eid_id, aircraft, `flight.phase`, altitude, cid_id, warned);
END
```

Try add_birdstrike_incident

```
# We created an entry for add_birdstrike_incident and inserted it into our database.
CALL add_birdstrike_incident("2027-5-5", "Georgia","AIRTRAN AIRWAYS","Airplane","Taxi",50,"No Cloud",1,"ATLANTA I
NTL")
```

Check add_birdstrike_incident

```
# To check whether it works or not, we returned the last added entry with the sql command and showed that it work
ed successfully.
SELECT * FROM incidents
ORDER BY rid DESC
LIMIT 1
```

1 records

rid	dep.date	origin	airline	aircraft	flight.phase	altitude	conditions	warned
321910	2027-05-05	9	7	Airplane	takeoff	50	1	1

Disconnect DB

```
# Disconnect mydb
dbDisconnect(mydb)
```

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