# DB Normalization & Analytical

# SQL query practice

**Name**: Hai Vu

**Date**: April 24, 2022

# Table of Contents

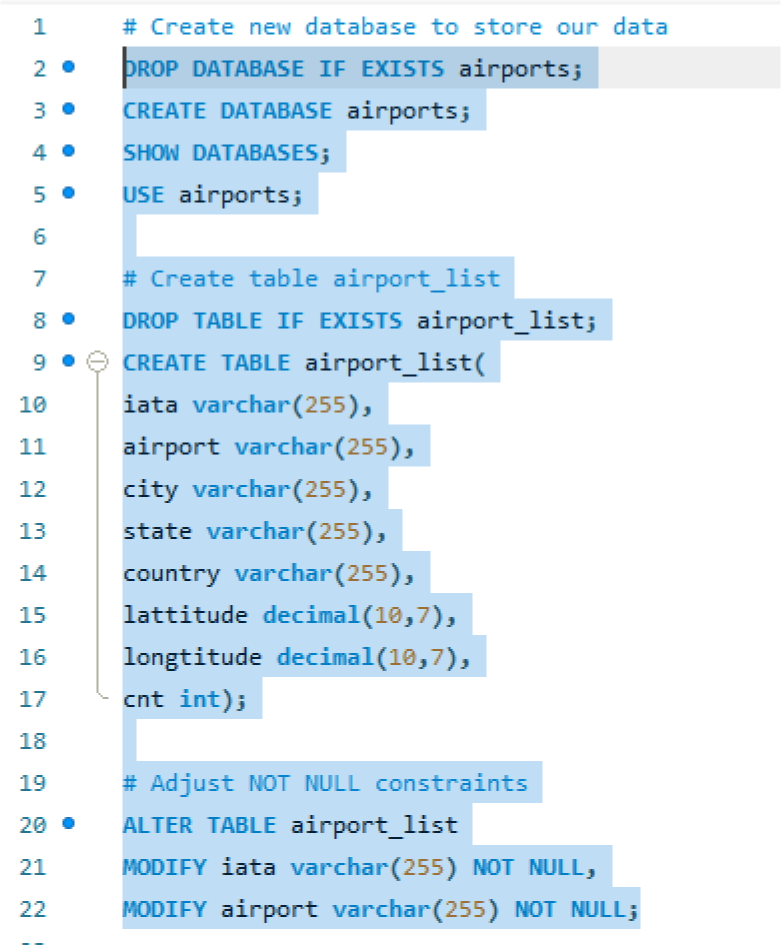
## I. INTRODUCTION

For this project, I will use MySQL Workbench to try normalizing a given data set to the Third Normal Form (3NF). I will then perform several data extraction queries using SQL to answer a few relevant business questions.

## II. ANALYSIS

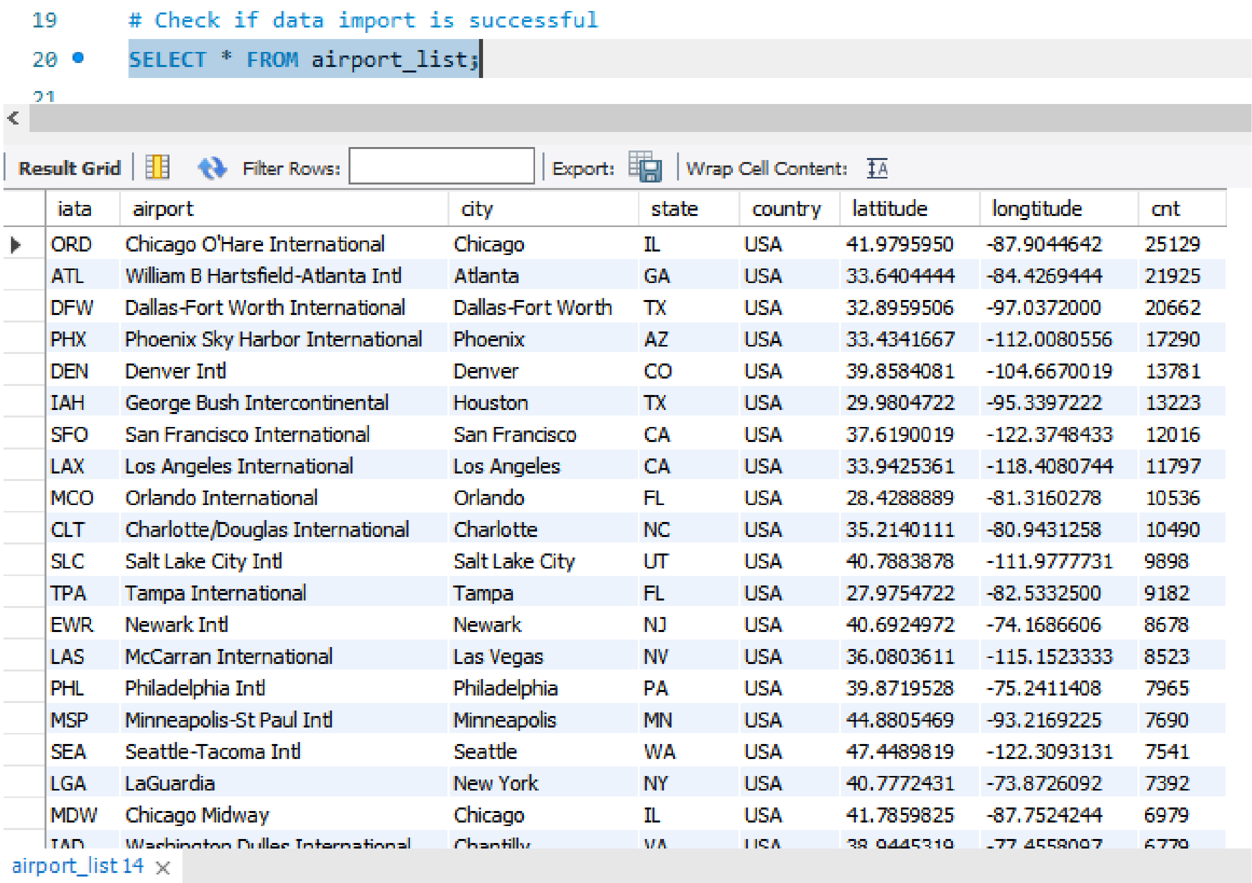
1. <u>Create database and import data using MySQL Workbench</u>

For this part, I picked the data set Airports to do my normalization up to Third Normal Form (3NF). First, I create a new data base and create an empty table to import the data set. There are 8 fields of data within the "Airports" data set and I will create these fields accordingly (similar to image below):



```
# Create new database to store our data
DROP DATABASE IF EXISTS airports;
CREATE DATABASE airports;
SHOW DATABASES;
USE airports;

# Create table airport_list
DROP TABLE IF EXISTS airport_list;
CREATE TABLE airport_list(
iata varchar(255),
airport varchar(255),
city varchar(255),
state varchar(255),
country varchar(255),
lattitude decimal(10,7),
longtitude decimal(10,7),
cnt int);

# Adjust NOT NULL constraints
ALTER TABLE airport_list
MODIFY iata varchar(255) NOT NULL,
MODIFY airport varchar(255) NOT NULL;
```

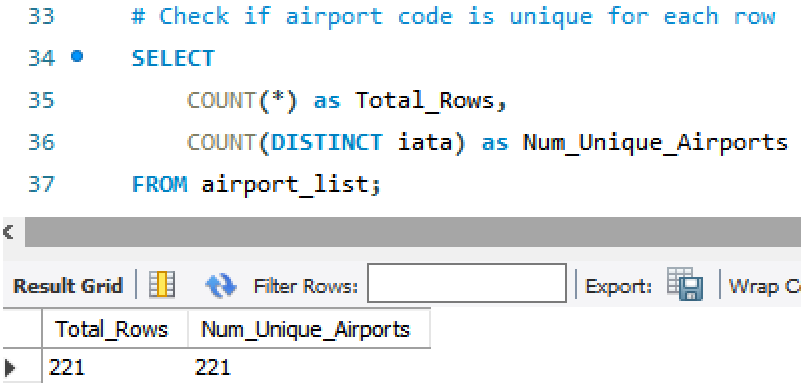
I don't want rows without the airport names and codes, thus, I specify them as "NOT NULL". After that, I import the data set using MySQL import wizard and begin to verify is the data has been successfully:



```sql
# Check if data import is successful
SELECT * FROM airport_list;
```

| iata | airport | city | state | country | lattitude | longtitude | cnt |
|---|---|---|---|---|---|---|---|
| ORD | Chicago O'Hare International | Chicago | IL | USA | 41.9795950 | -87.9044642 | 25129 |
| ATL | William B Hartsfield-Atlanta Intl | Atlanta | GA | USA | 33.6404444 | -84.4269444 | 21925 |
| DFW | Dallas-Fort Worth International | Dallas-Fort Worth | TX | USA | 32.8959506 | -97.0372000 | 20662 |
| PHX | Phoenix Sky Harbor International | Phoenix | AZ | USA | 33.4341667 | -112.0080556 | 17290 |
| DEN | Denver Intl | Denver | CO | USA | 39.8584081 | -104.6670019 | 13781 |
| IAH | George Bush Intercontinental | Houston | TX | USA | 29.9804722 | -95.3397222 | 13223 |
| SFO | San Francisco International | San Francisco | CA | USA | 37.6190019 | -122.3748433 | 12016 |
| LAX | Los Angeles International | Los Angeles | CA | USA | 33.9425361 | -118.4080744 | 11797 |
| MCO | Orlando International | Orlando | FL | USA | 28.4288889 | -81.3160278 | 10536 |
| CLT | Charlotte/Douglas International | Charlotte | NC | USA | 35.2140111 | -80.9431258 | 10490 |
| SLC | Salt Lake City Intl | Salt Lake City | UT | USA | 40.7883878 | -111.9777731 | 9898 |
| TPA | Tampa International | Tampa | FL | USA | 27.9754722 | -82.5332500 | 9182 |
| EWR | Newark Intl | Newark | NJ | USA | 40.6924972 | -74.1686606 | 8678 |
| LAS | McCarran International | Las Vegas | NV | USA | 36.0803611 | -115.1523333 | 8523 |
| PHL | Philadelphia Intl | Philadelphia | PA | USA | 39.8719528 | -75.2411408 | 7965 |
| MSP | Minneapolis-St Paul Intl | Minneapolis | MN | USA | 44.8805469 | -93.2169225 | 7690 |
| SEA | Seattle-Tacoma Intl | Seattle | WA | USA | 47.4489819 | -122.3093131 | 7541 |
| LGA | LaGuardia | New York | NY | USA | 40.7772431 | -73.8726092 | 7392 |
| MDW | Chicago Midway | Chicago | IL | USA | 41.7859825 | -87.7524244 | 6979 |
| IAD | Washington Dulles International | Chantilly | VA | USA | 38.9445319 | -77.4558097 | 6779 |

It seems that the data has been imported successfully. Next, I want to check with current normalization form the data is in:



```sql
# Check if airport code is unique for each row
SELECT
    COUNT(*) as Total_Rows,
    COUNT(DISTINCT iata) as Num_Unique_Airports
FROM airport_list;
```

| Total_Rows | Num_Unique_Airports |
|---|---|
| 221 | 221 |

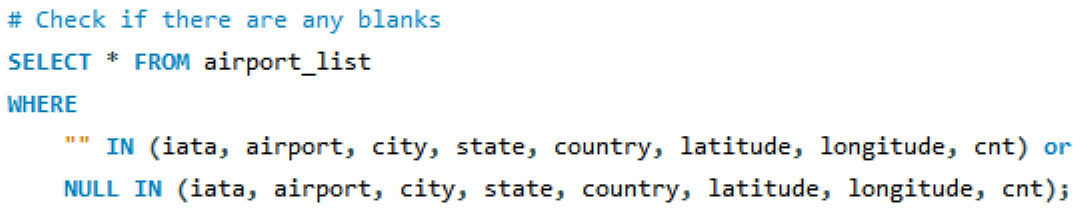
There are 221 records of data in this data set, and each record has a unique airport code. This means that the table is already in 1NF, where the table rows contain no repeating groups or arrays.

Next, the Second Normal Form (2NF) requires that the table is in 1NF and that each non-key column is fully functionally dependent on the primary key. This means that if a table has a composite primary key, each non-key column must depend on the entire composite key, not just part of it. In our data set, the composite key is consisted of “iata” and “airport” (the codes and names of airports). This means that to be 2NF, every other columns should be dependent on both “iata” and “airport”. In this case, our data set is not yet in 2NF since a lot of information is only dependent on either “iata” and “airport”, or that “iata” and “airport” are showing similar information. We need to create separate table to store data of just airport names and codes.

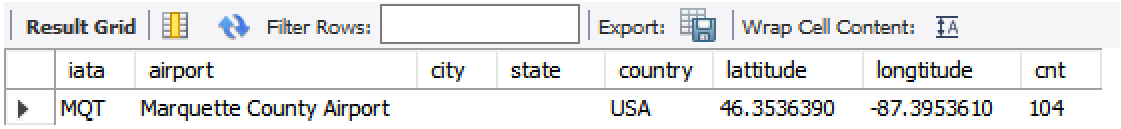
Finally, the Third Normal Form (3NF) requires that the table is in 2NF and that each non-key column is not transitively dependent on the primary key. This means that if a non-key column depends on another non-key column, it should be moved to its own table. This means that we will need to create separate tables for “city” and “state” columns to reach 3NF.

## 2. Clean data and begin normalization process

After that, I check to see if there are any blanks in the data set, and do my modifications based on that:

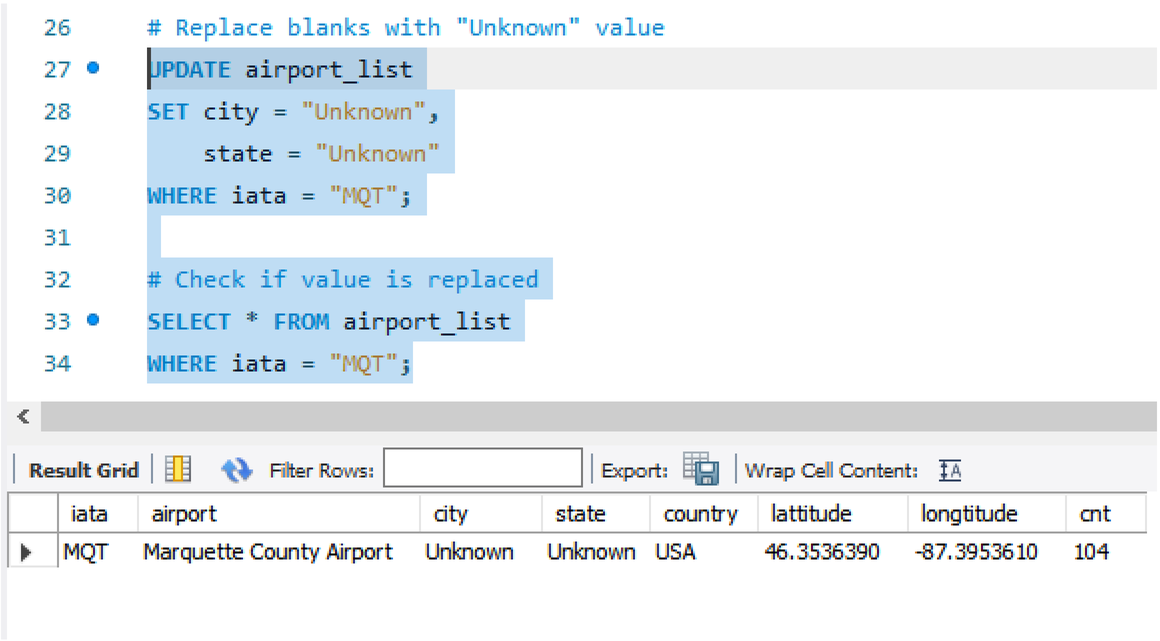


```
# Check if there are any blanks
SELECT * FROM airport_list
WHERE
    "" IN (iata, airport, city, state, country, latitude, longitude, cnt) or
    NULL IN (iata, airport, city, state, country, latitude, longitude, cnt);
```



| iata | airport | city | state | country | lattitude | longtitude | cnt |
|---|---|---|---|---|---|---|---|
| MQT | Marquette County Airport | | | USA | 46.3536390 | -87.3953610 | 104 |

There is 1 row containing blanks as shown above. I will replace the blank values with unknown so that no NULLs exist.

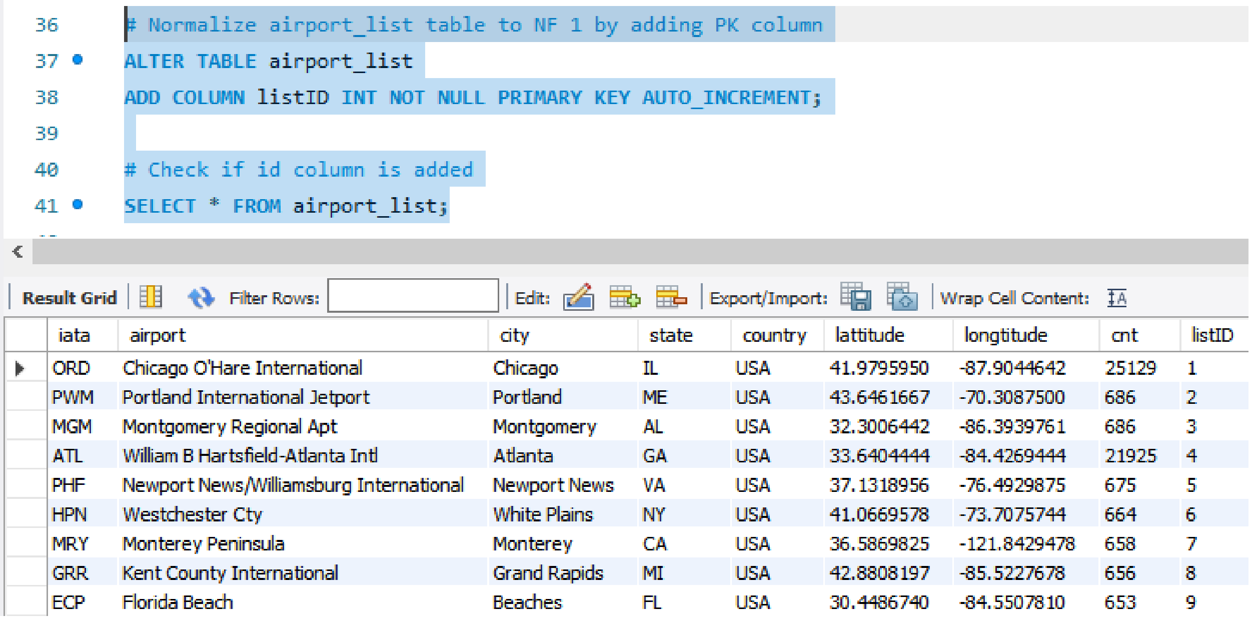


```
# Replace blanks with "Unknown" value
UPDATE airport_list
SET city = "Unknown",
    state = "Unknown"
WHERE iata = "MQT";

# Check if value is replaced
SELECT * FROM airport_list
WHERE iata = "MQT";
```

| iata | airport | city | state | country | lattitude | longtitude | cnt |
|---|---|---|---|---|---|---|---|
| MQT | Marquette County Airport | Unknown | Unknown | USA | 46.3536390 | -87.3953610 | 104 |

The records has been modified successfully.

Next, I create a primary key column “listID” for the airport_list table using increments.

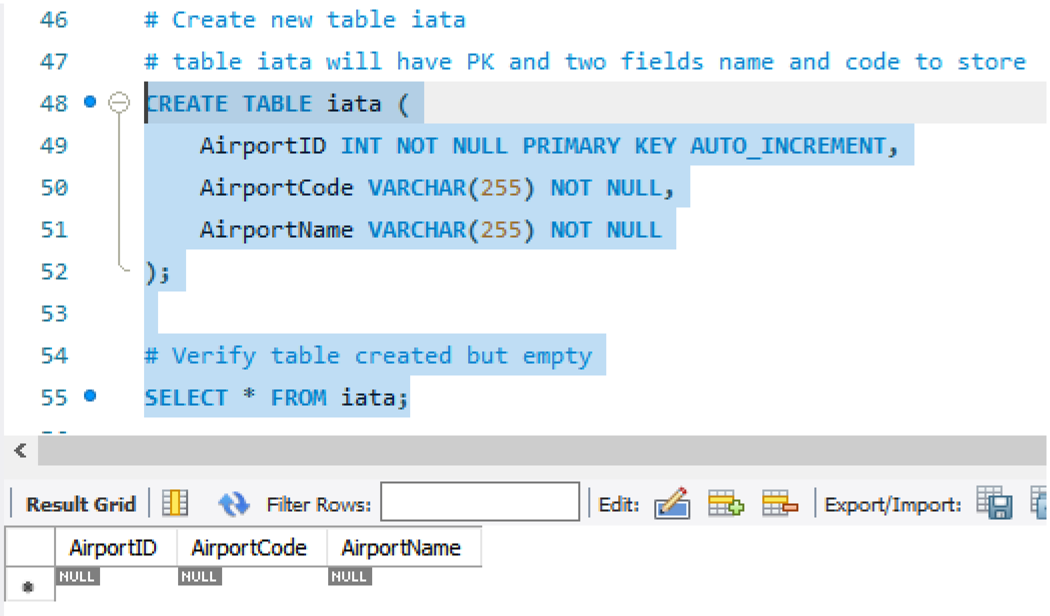


```
# Normalize airport_list table to NF 1 by adding PK column
ALTER TABLE airport_list
ADD COLUMN listID INT NOT NULL PRIMARY KEY AUTO_INCREMENT;

# Check if id column is added
SELECT * FROM airport_list;
```

| iata | airport | city | state | country | lattitude | longtitude | cnt | listID |
|---|---|---|---|---|---|---|---|---|
| ORD | Chicago O'Hare International | Chicago | IL | USA | 41.9795950 | -87.9044642 | 25129 | 1 |
| PWM | Portland International Jetport | Portland | ME | USA | 43.6461667 | -70.3087500 | 686 | 2 |
| MGM | Montgomery Regional Apt | Montgomery | AL | USA | 32.3006442 | -86.3939761 | 686 | 3 |
| ATL | William B Hartsfield-Atlanta Intl | Atlanta | GA | USA | 33.6404444 | -84.4269444 | 21925 | 4 |
| PHF | Newport News/Williamsburg International | Newport News | VA | USA | 37.1318956 | -76.4929875 | 675 | 5 |
| HPN | Westchester Cty | White Plains | NY | USA | 41.0669578 | -73.7075744 | 664 | 6 |
| MRY | Monterey Peninsula | Monterey | CA | USA | 36.5869825 | -121.8429478 | 658 | 7 |
| GRR | Kent County International | Grand Rapids | MI | USA | 42.8808197 | -85.5227678 | 656 | 8 |
| ECP | Florida Beach | Beaches | FL | USA | 30.4486740 | -84.5507810 | 653 | 9 |

I notice that there are three subset tables that can be created to remove redundant information from this main table. I plan to create 3 tables called: iata (stored airport names and codes), city (store city names), and state (store state names).

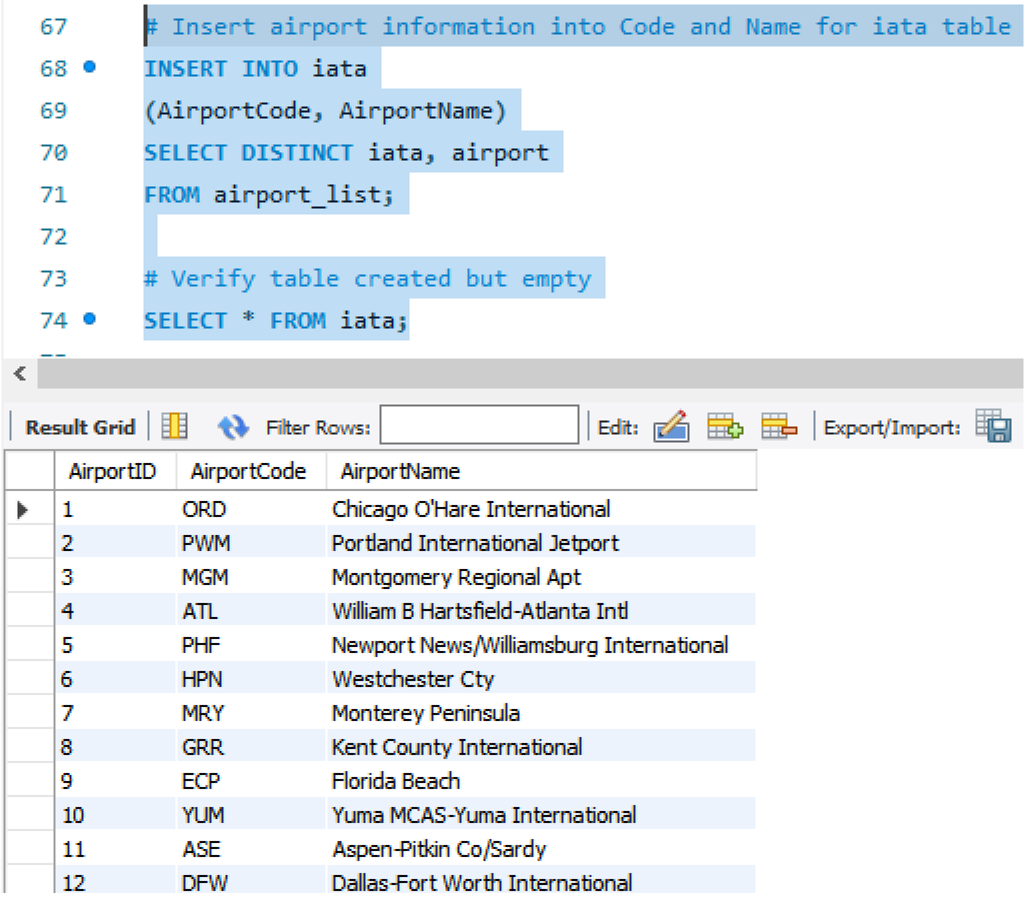
Let’s first create the iata table:



```sql
# Create new table iata
# table iata will have PK and two fields name and code to store
CREATE TABLE iata (
    AirportID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    AirportCode VARCHAR(255) NOT NULL,
    AirportName VARCHAR(255) NOT NULL
);

# Verify table created but empty
SELECT * FROM iata;
```

| AirportID | AirportCode | AirportName |
|---|---|---|
| NULL | NULL | NULL |

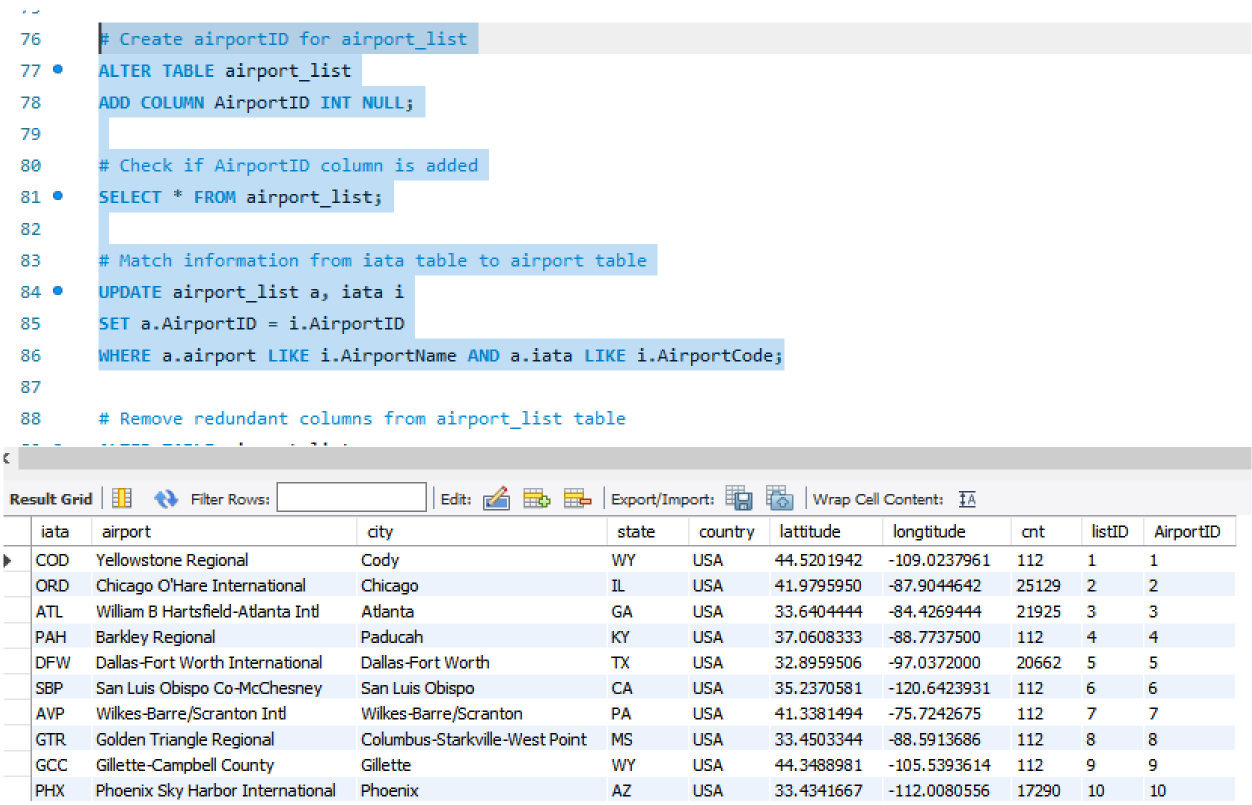
The empty table has been created. Let's fill them up using information from the main table:



```sql
# Insert airport information into Code and Name for iata table
INSERT INTO iata
(AirportCode, AirportName)
SELECT DISTINCT iata, airport
FROM airport_list;

# Verify table created but empty
SELECT * FROM iata;
```

| AirportID | AirportCode | AirportName |
|---|---|---|
| 1 | ORD | Chicago O'Hare International |
| 2 | PWM | Portland International Jetport |
| 3 | MGM | Montgomery Regional Apt |
| 4 | ATL | William B Hartsfield-Atlanta Intl |
| 5 | PHF | Newport News/Williamsburg International |
| 6 | HPN | Westchester Cty |
| 7 | MRY | Monterey Peninsula |
| 8 | GRR | Kent County International |
| 9 | ECP | Florida Beach |
| 10 | YUM | Yuma MCAS-Yuma International |
| 11 | ASE | Aspen-Pitkin Co/Sardy |
| 12 | DFW | Dallas-Fort Worth International |

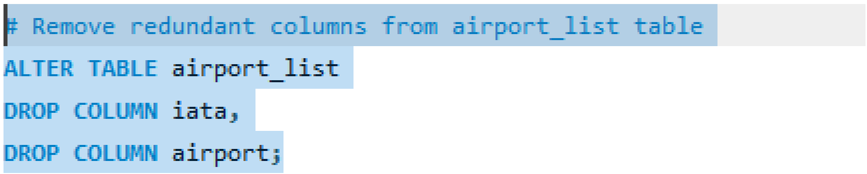
Next, we create an Airport ID column in the airport_list table (the main table) and match data between the two tables using the update function



```
# Create airportID for airport_list
ALTER TABLE airport_list
ADD COLUMN AirportID INT NULL;

# Check if AirportID column is added
SELECT * FROM airport_list;

# Match information from iata table to airport table
UPDATE airport_list a, iata i
SET a.AirportID = i.AirportID
WHERE a.airport LIKE i.AirportName AND a.iata LIKE i.AirportCode;

# Remove redundant columns from airport_list table
```

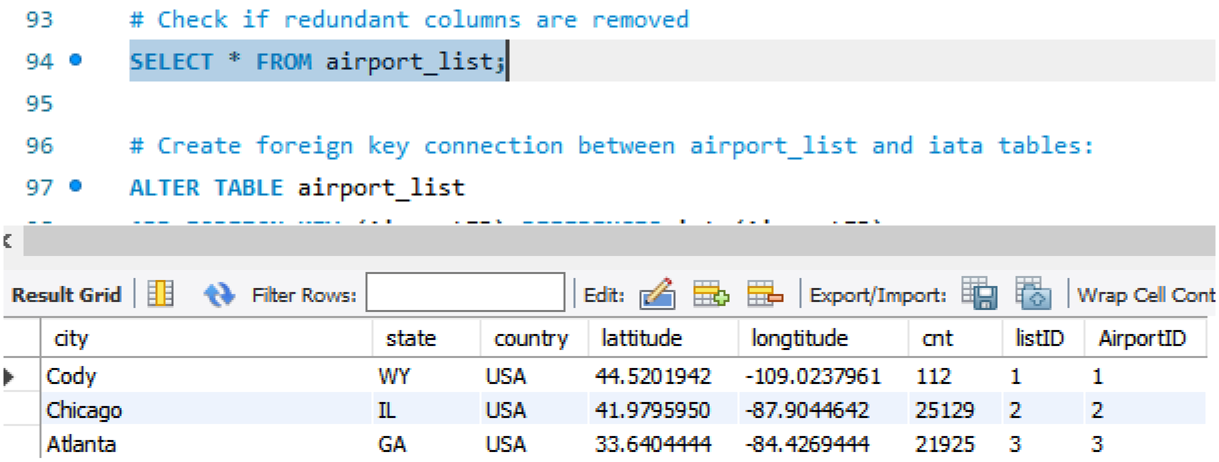
| iata | airport | city | state | country | lattitude | longtitude | cnt | listID | AirportID |
|---|---|---|---|---|---|---|---|---|---|
| COD | Yellowstone Regional | Cody | WY | USA | 44.5201942 | -109.0237961 | 112 | 1 | 1 |
| ORD | Chicago O'Hare International | Chicago | IL | USA | 41.9795950 | -87.9044642 | 25129 | 2 | 2 |
| ATL | William B Hartsfield-Atlanta Intl | Atlanta | GA | USA | 33.6404444 | -84.4269444 | 21925 | 3 | 3 |
| PAH | Barkley Regional | Paducah | KY | USA | 37.0608333 | -88.7737500 | 112 | 4 | 4 |
| DFW | Dallas-Fort Worth International | Dallas-Fort Worth | TX | USA | 32.8959506 | -97.0372000 | 20662 | 5 | 5 |
| SBP | San Luis Obispo Co-McChesney | San Luis Obispo | CA | USA | 35.2370581 | -120.6423931 | 112 | 6 | 6 |
| AVP | Wilkes-Barre/Scranton Intl | Wilkes-Barre/Scranton | PA | USA | 41.3381494 | -75.7242675 | 112 | 7 | 7 |
| GTR | Golden Triangle Regional | Columbus-Starkville-West Point | MS | USA | 33.4503344 | -88.5913686 | 112 | 8 | 8 |
| GCC | Gillette-Campbell County | Gillette | WY | USA | 44.3488981 | -105.5393614 | 112 | 9 | 9 |
| PHX | Phoenix Sky Harbor International | Phoenix | AZ | USA | 33.4341667 | -112.0080556 | 17290 | 10 | 10 |

We then remove redundant columns “iata” and “airport” from this main table:



```
# Remove redundant columns from airport_list table
ALTER TABLE airport_list
DROP COLUMN iata,
DROP COLUMN airport;
```

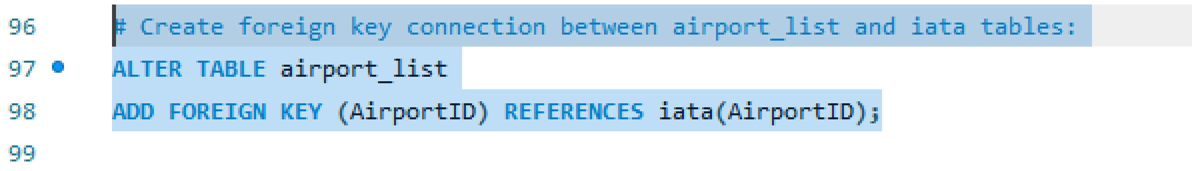
Below is the current look of our main table:



```
# Check if redundant columns are removed
SELECT * FROM airport_list;

# Create foreign key connection between airport_list and iata tables:
ALTER TABLE airport_list
```

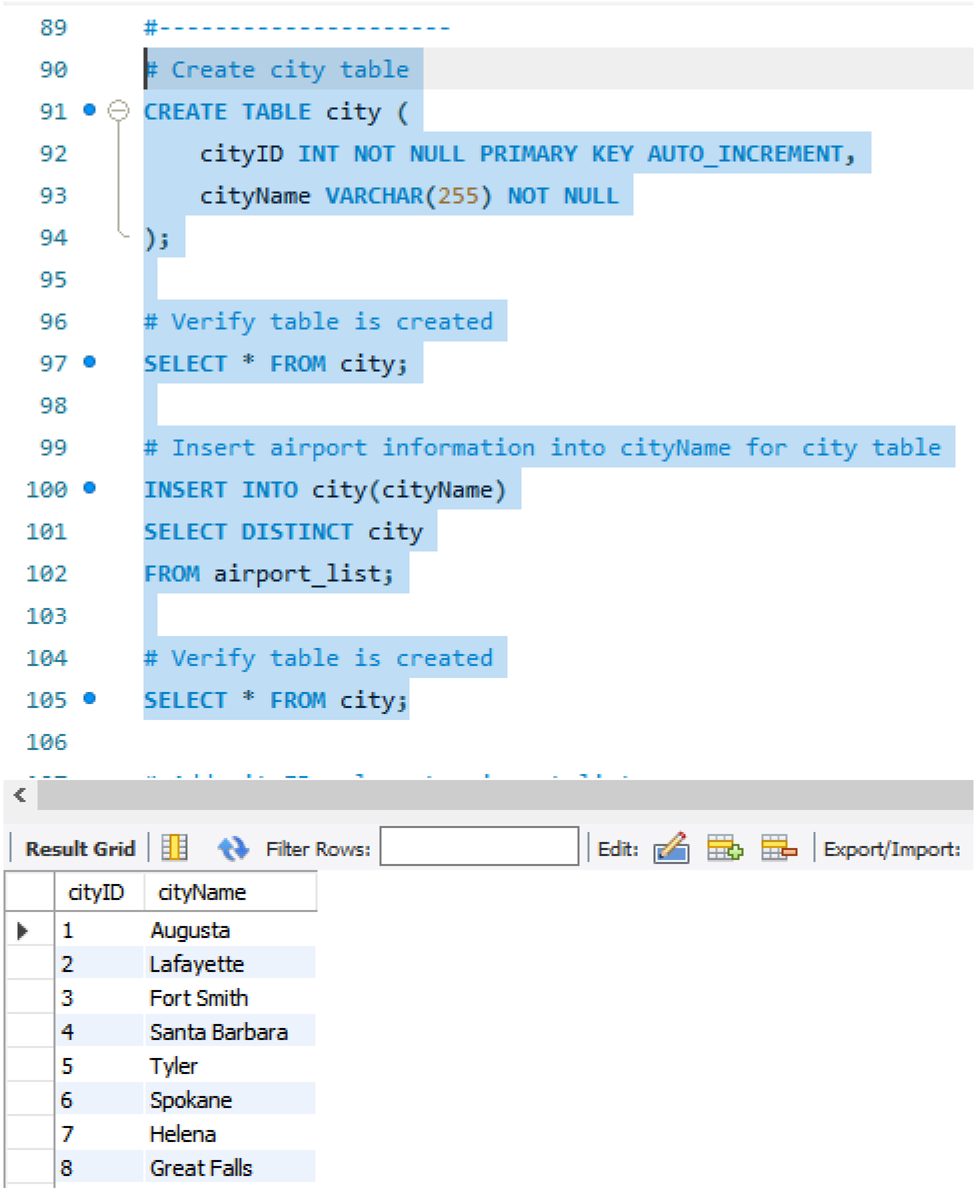
| city | state | country | lattitude | longtitude | cnt | listID | AirportID |
|---|---|---|---|---|---|---|---|
| Cody | WY | USA | 44.5201942 | -109.0237961 | 112 | 1 | 1 |
| Chicago | IL | USA | 41.9795950 | -87.9044642 | 25129 | 2 | 2 |
| Atlanta | GA | USA | 33.6404444 | -84.4269444 | 21925 | 3 | 3 |

Finally, we create a foreign key connection between the 2 tables iata and airport_list, referencing the AirportID columns in both tables:



```
# Create foreign key connection between airport_list and iata tables:
ALTER TABLE airport_list
ADD FOREIGN KEY (AirportID) REFERENCES iata(AirportID);

```

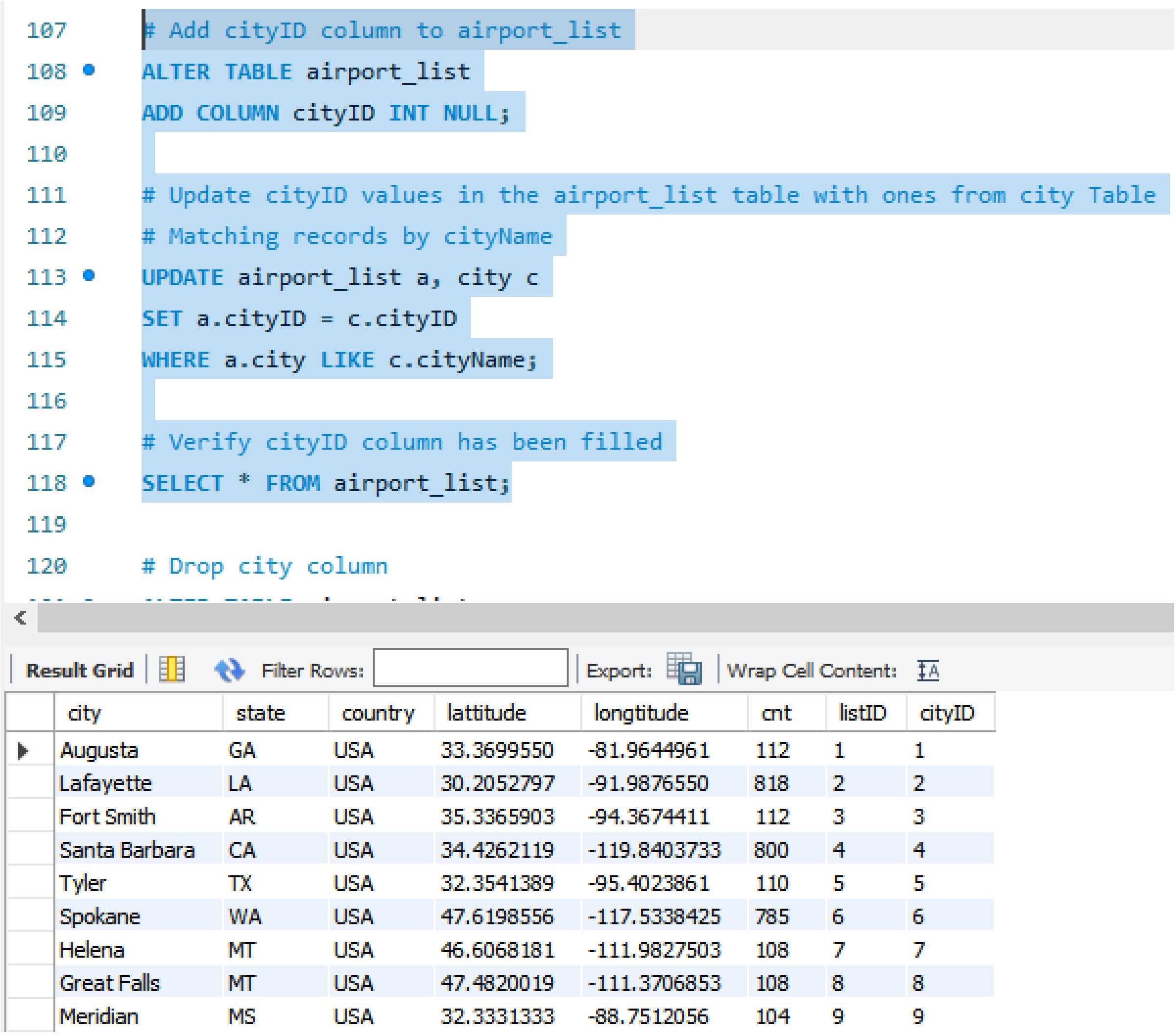
Similarly, we will go ahead and create the “city” table and “state” table. First, we create the city table with cityID as its primary key. We then insert values into this newly created table using information from the main table:



```
#--------------------
# Create city table
CREATE TABLE city (
    cityID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    cityName VARCHAR(255) NOT NULL
);

# Verify table is created
SELECT * FROM city;

# Insert airport information into cityName for city table
INSERT INTO city(cityName)
SELECT DISTINCT city
FROM airport_list;

# Verify table is created
SELECT * FROM city;

```

| cityID | cityName |
|---|---|
| 1 | Augusta |
| 2 | Lafayette |
| 3 | Fort Smith |
| 4 | Santa Barbara |
| 5 | Tyler |
| 6 | Spokane |
| 7 | Helena |
| 8 | Great Falls |

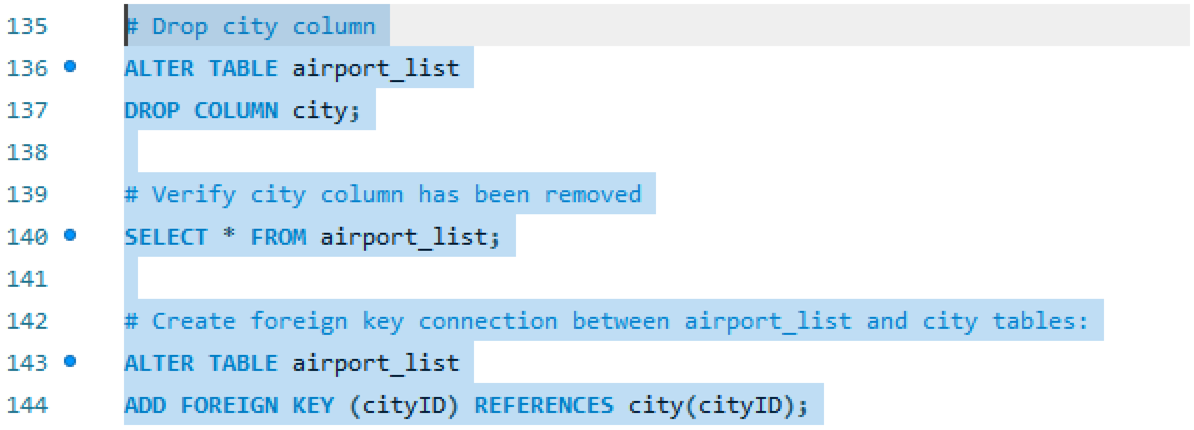
The city table has been filled with correct data. We create a “cityID” column for the airport_list table and match its values with the cityID from the city table.



```
# Add cityID column to airport_list
ALTER TABLE airport_list
ADD COLUMN cityID INT NULL;

# Update cityID values in the airport_list table with ones from city Table
# Matching records by cityName
UPDATE airport_list a, city c
SET a.cityID = c.cityID
WHERE a.city LIKE c.cityName;

# Verify cityID column has been filled
SELECT * FROM airport_list;

# Drop city column
```

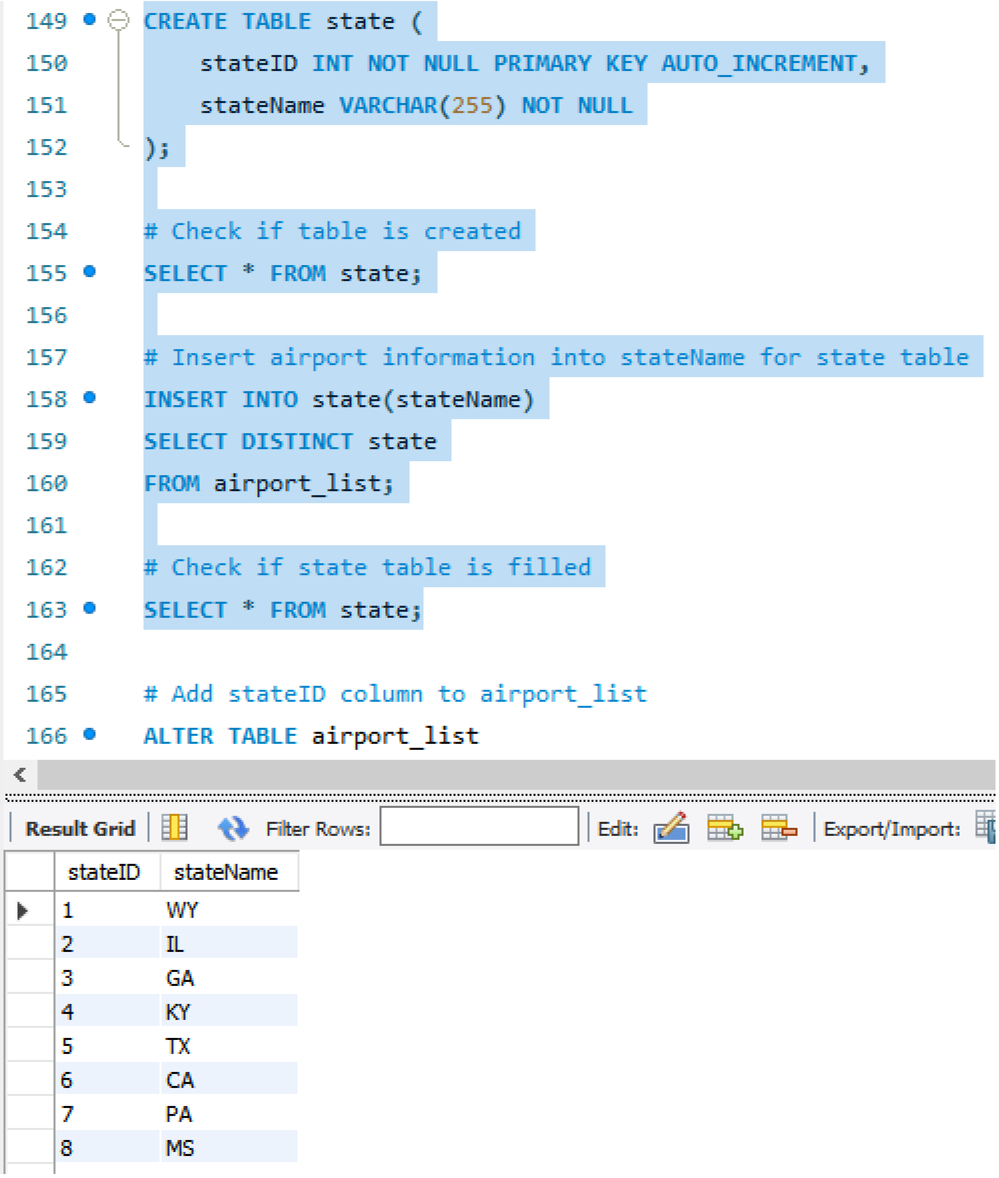
| city | state | country | lattitude | longtitude | cnt | listID | cityID |
|---|---|---|---|---|---|---|---|
| Augusta | GA | USA | 33.3699550 | -81.9644961 | 112 | 1 | 1 |
| Lafayette | LA | USA | 30.2052797 | -91.9876550 | 818 | 2 | 2 |
| Fort Smith | AR | USA | 35.3365903 | -94.3674411 | 112 | 3 | 3 |
| Santa Barbara | CA | USA | 34.4262119 | -119.8403733 | 800 | 4 | 4 |
| Tyler | TX | USA | 32.3541389 | -95.4023861 | 110 | 5 | 5 |
| Spokane | WA | USA | 47.6198556 | -117.5338425 | 785 | 6 | 6 |
| Helena | MT | USA | 46.6068181 | -111.9827503 | 108 | 7 | 7 |
| Great Falls | MT | USA | 47.4820019 | -111.3706853 | 108 | 8 | 8 |
| Meridian | MS | USA | 32.3331333 | -88.7512056 | 104 | 9 | 9 |

After that, we remove the redundant “city” column using the alter table and drop function and establish connection between the main table and the “city” table.



```
# Drop city column
ALTER TABLE airport_list
DROP COLUMN city;

# Verify city column has been removed
SELECT * FROM airport_list;

# Create foreign key connection between airport_list and city tables:
ALTER TABLE airport_list
ADD FOREIGN KEY (cityID) REFERENCES city(cityID);
```

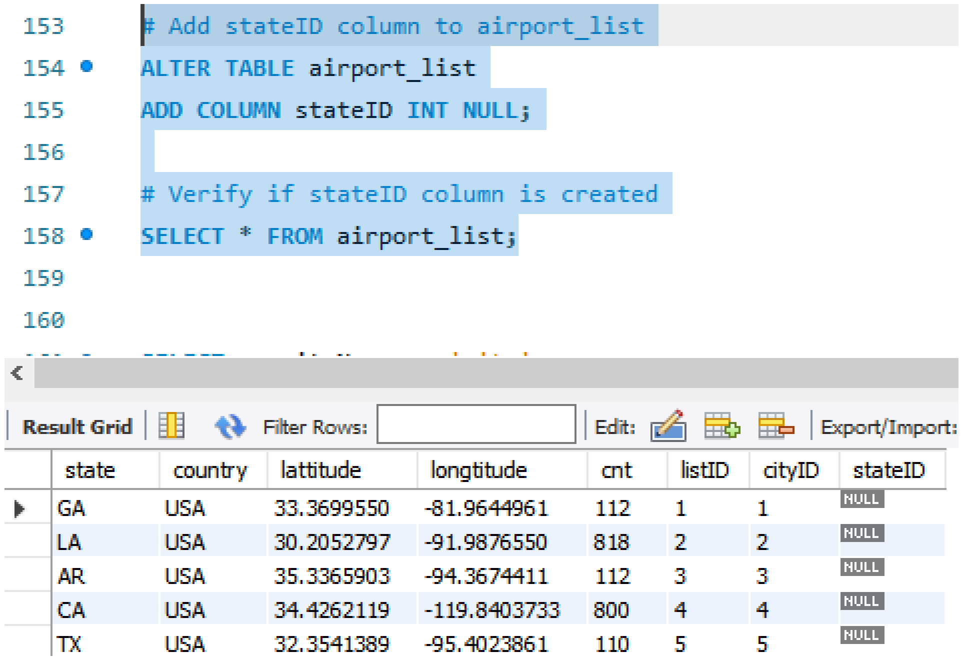
Next, we begin creating the “state” table following similar steps:



```
CREATE TABLE state (
    stateID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    stateName VARCHAR(255) NOT NULL
);

# Check if table is created
SELECT * FROM state;

# Insert airport information into stateName for state table
INSERT INTO state(stateName)
SELECT DISTINCT state
FROM airport_list;

# Check if state table is filled
SELECT * FROM state;

# Add stateID column to airport_list
ALTER TABLE airport_list
```

| stateID | stateName |
|---|---|
| 1 | WY |
| 2 | IL |
| 3 | GA |
| 4 | KY |
| 5 | TX |
| 6 | CA |
| 7 | PA |
| 8 | MS |

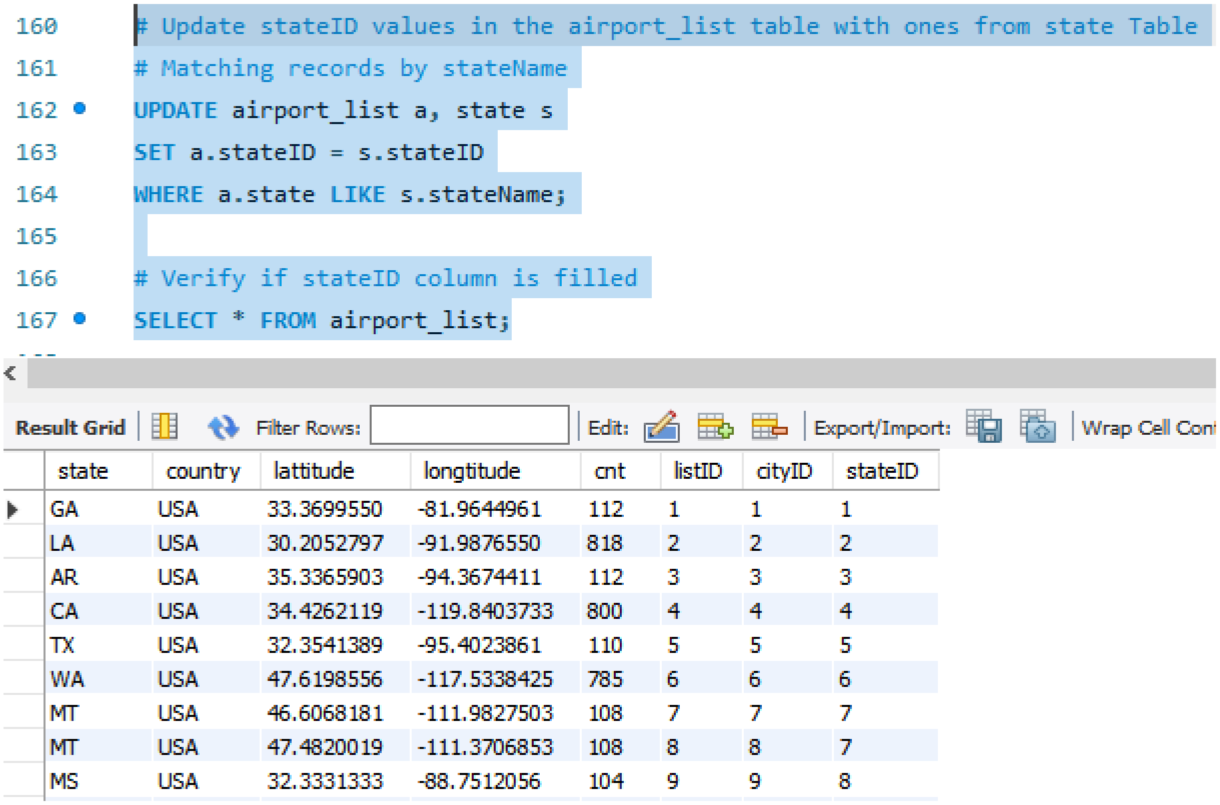
The state table has been created along with a primary key column “stateID”. We create a column with a same name in the main table to insert new information:



```
# Add stateID column to airport_list
ALTER TABLE airport_list
ADD COLUMN stateID INT NULL;

# Verify if stateID column is created
SELECT * FROM airport_list;
```

| state | country | lattitude | longtitude | cnt | listID | cityID | stateID |
|---|---|---|---|---|---|---|---|
| GA | USA | 33.3699550 | -81.9644961 | 112 | 1 | 1 | NULL |
| LA | USA | 30.2052797 | -91.9876550 | 818 | 2 | 2 | NULL |
| AR | USA | 35.3365903 | -94.3674411 | 112 | 3 | 3 | NULL |
| CA | USA | 34.4262119 | -119.8403733 | 800 | 4 | 4 | NULL |
| TX | USA | 32.3541389 | -95.4023861 | 110 | 5 | 5 | NULL |

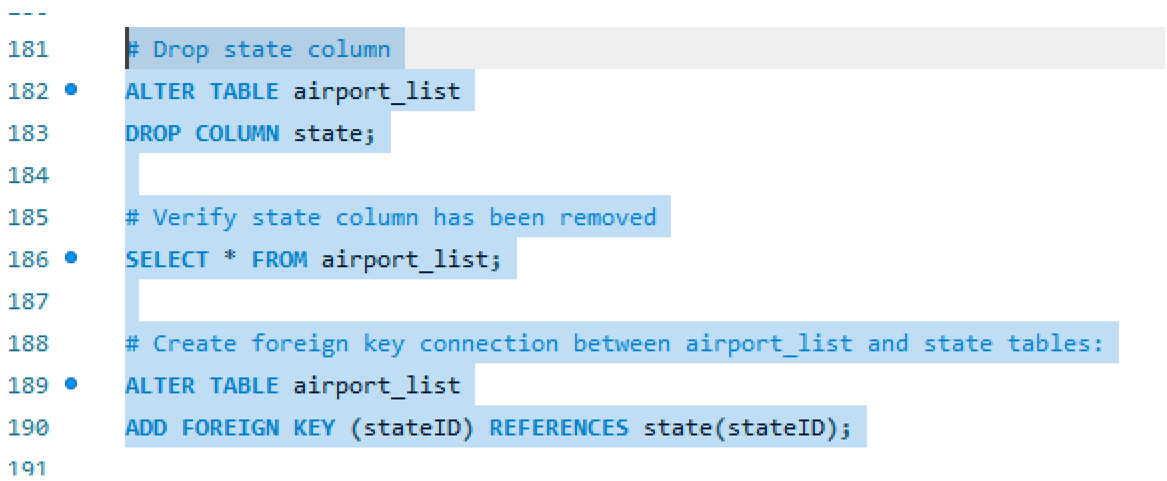
We then insert correct values into the stateID column of the main table from the "state" table:



```
# Update stateID values in the airport_list table with ones from state Table
# Matching records by stateName
UPDATE airport_list a, state s
SET a.stateID = s.stateID
WHERE a.state LIKE s.stateName;

# Verify if stateID column is filled
SELECT * FROM airport_list;
```

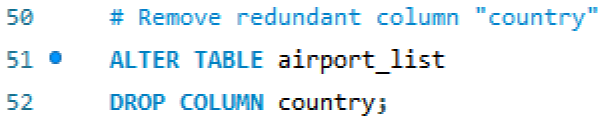
| state | country | lattitude | longtitude | cnt | listID | cityID | stateID |
|---|---|---|---|---|---|---|---|
| GA | USA | 33.3699550 | -81.9644961 | 112 | 1 | 1 | 1 |
| LA | USA | 30.2052797 | -91.9876550 | 818 | 2 | 2 | 2 |
| AR | USA | 35.3365903 | -94.3674411 | 112 | 3 | 3 | 3 |
| CA | USA | 34.4262119 | -119.8403733 | 800 | 4 | 4 | 4 |
| TX | USA | 32.3541389 | -95.4023861 | 110 | 5 | 5 | 5 |
| WA | USA | 47.6198556 | -117.5338425 | 785 | 6 | 6 | 6 |
| MT | USA | 46.6068181 | -111.9827503 | 108 | 7 | 7 | 7 |
| MT | USA | 47.4820019 | -111.3706853 | 108 | 8 | 8 | 7 |
| MS | USA | 32.3331333 | -88.7512056 | 104 | 9 | 9 | 8 |

Finally, we remove the redundant column "state" and establish foreign key connection:



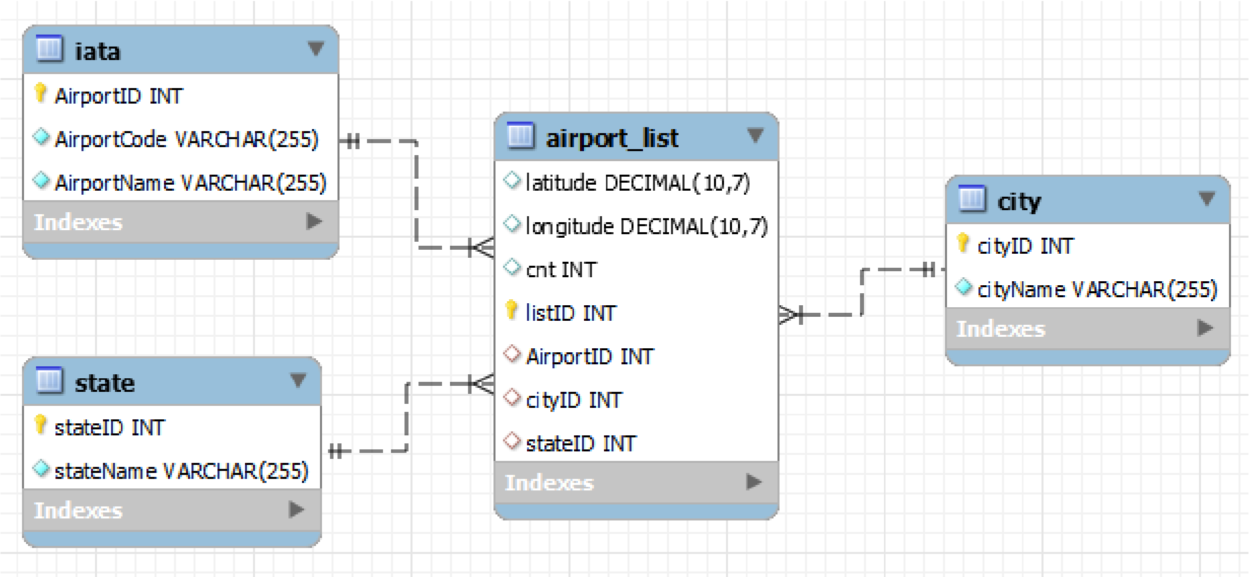
```
# Drop state column
ALTER TABLE airport_list
DROP COLUMN state;

# Verify state column has been removed
SELECT * FROM airport_list;

# Create foreign key connection between airport_list and state tables:
ALTER TABLE airport_list
ADD FOREIGN KEY (stateID) REFERENCES state(stateID);
```

Since there is only 1 country here, we can remove the column “country” as well:



```
# Remove redundant column "country"
ALTER TABLE airport_list
DROP COLUMN country;
```

The following is the final schema design I generated from the codes above:



From the schema above, we can see that our database is now in 3NF.

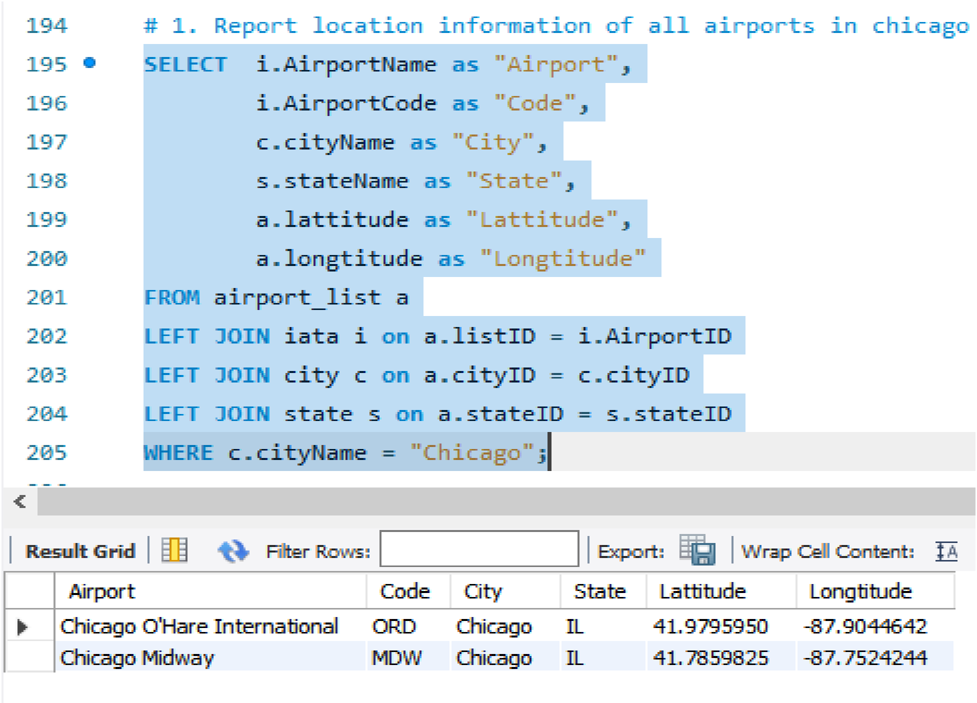
3. <u>Write query to answer business questions</u>

I will try to answer these 5 questions using SQL SELECT queries from multiple tables:

- Q1. Report location information of all airports in Chicago
- Q2. Report information of all airports and their cities in the state of MA or IL

- Q3. Report airports with unknown cities or states
- Q4. Show airport names from states whose names start with “M”
- Q5. Show states with the top 5 highest number of airports and their ranking, sorted by ranking

**Q1 query:**

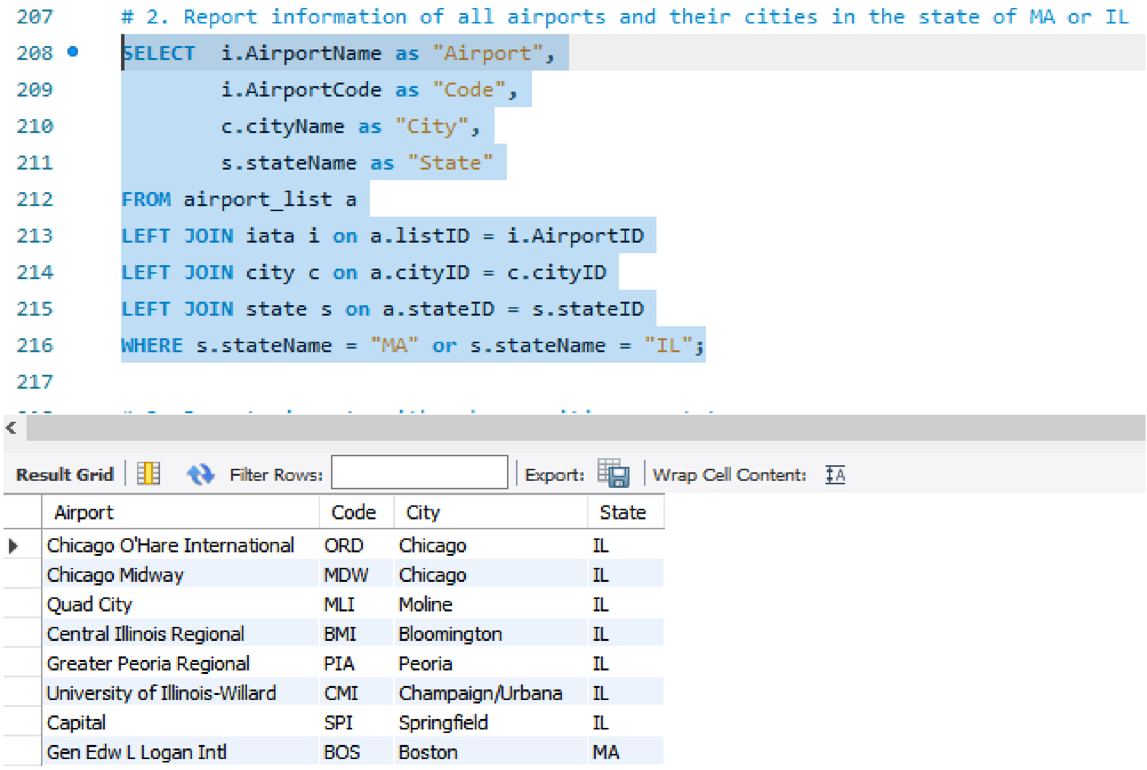


```
# 1. Report location information of all airports in chicago
SELECT  i.AirportName as "Airport",
        i.AirportCode as "Code",
        c.cityName as "City",
        s.stateName as "State",
        a.lattitude as "Lattitude",
        a.longtitude as "Longtitude"
FROM airport_list a
LEFT JOIN iata i on a.listID = i.AirportID
LEFT JOIN city c on a.cityID = c.cityID
LEFT JOIN state s on a.stateID = s.stateID
WHERE c.cityName = "Chicago";
```

| Airport | Code | City | State | Lattitude | Longtitude |
|---|---|---|---|---|---|
| Chicago O'Hare International | ORD | Chicago | IL | 41.9795950 | -87.9044642 |
| Chicago Midway | MDW | Chicago | IL | 41.7859825 | -87.7524244 |

There are 2 airports in Chicago: Chicago O’Hare International Airport and Chicago Midway Airport.

**Q2 query:**

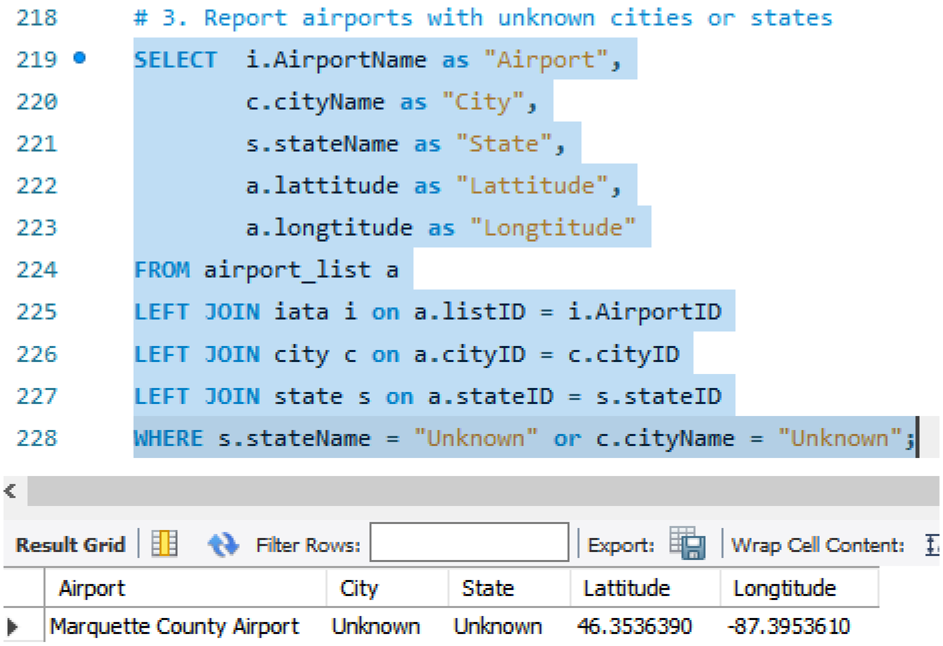


```
# 2. Report information of all airports and their cities in the state of MA or IL
SELECT  i.AirportName as "Airport",
        i.AirportCode as "Code",
        c.cityName as "City",
        s.stateName as "State"
FROM airport_list a
LEFT JOIN iata i on a.listID = i.AirportID
LEFT JOIN city c on a.cityID = c.cityID
LEFT JOIN state s on a.stateID = s.stateID
WHERE s.stateName = "MA" or s.stateName = "IL";
```

| Airport | Code | City | State |
|---|---|---|---|
| Chicago O'Hare International | ORD | Chicago | IL |
| Chicago Midway | MDW | Chicago | IL |
| Quad City | MLI | Moline | IL |
| Central Illinois Regional | BMI | Bloomington | IL |
| Greater Peoria Regional | PIA | Peoria | IL |
| University of Illinois-Willard | CMI | Champaign/Urbana | IL |
| Capital | SPI | Springfield | IL |
| Gen Edw L Logan Intl | BOS | Boston | MA |

There are 7 airports in IL and only 1 in MA.

**Q3 query:**

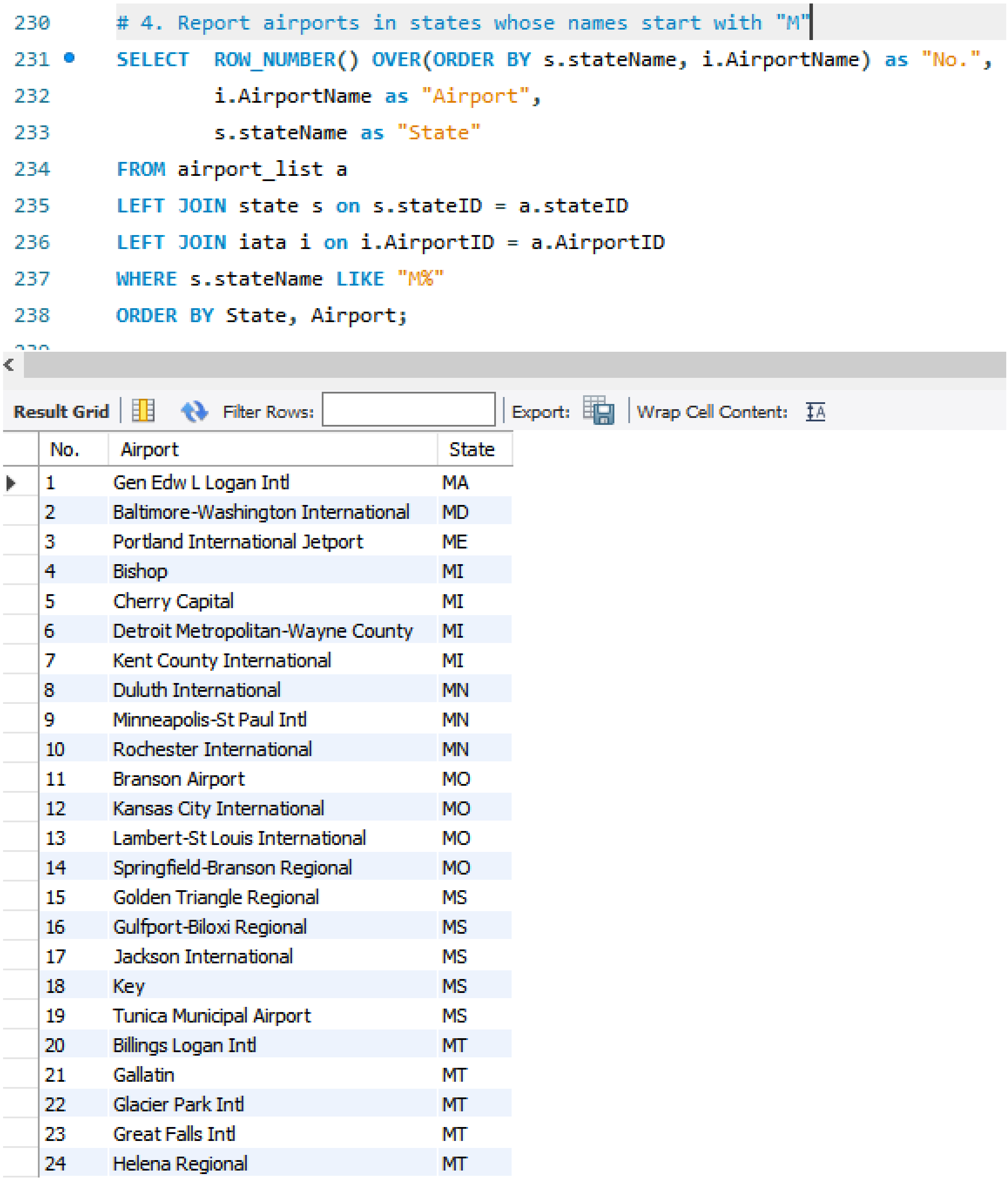


```
# 3. Report airports with unknown cities or states
SELECT  i.AirportName as "Airport",
        c.cityName as "City",
        s.stateName as "State",
        a.lattitude as "Lattitude",
        a.longtitude as "Longtitude"
FROM airport_list a
LEFT JOIN iata i on a.listID = i.AirportID
LEFT JOIN city c on a.cityID = c.cityID
LEFT JOIN state s on a.stateID = s.stateID
WHERE s.stateName = "Unknown" or c.cityName = "Unknown";
```

| Airport | City | State | Lattitude | Longtitude |
|---|---|---|---|---|
| Marquette County Airport | Unknown | Unknown | 46.3536390 | -87.3953610 |

The Marquette County Airport does not have its city and state information available. From my research, it may be because the name of the airport has been changed to the Sawyer International Airport, which is also located in Marquette County, Michigan (MI).

**Q4 query:**

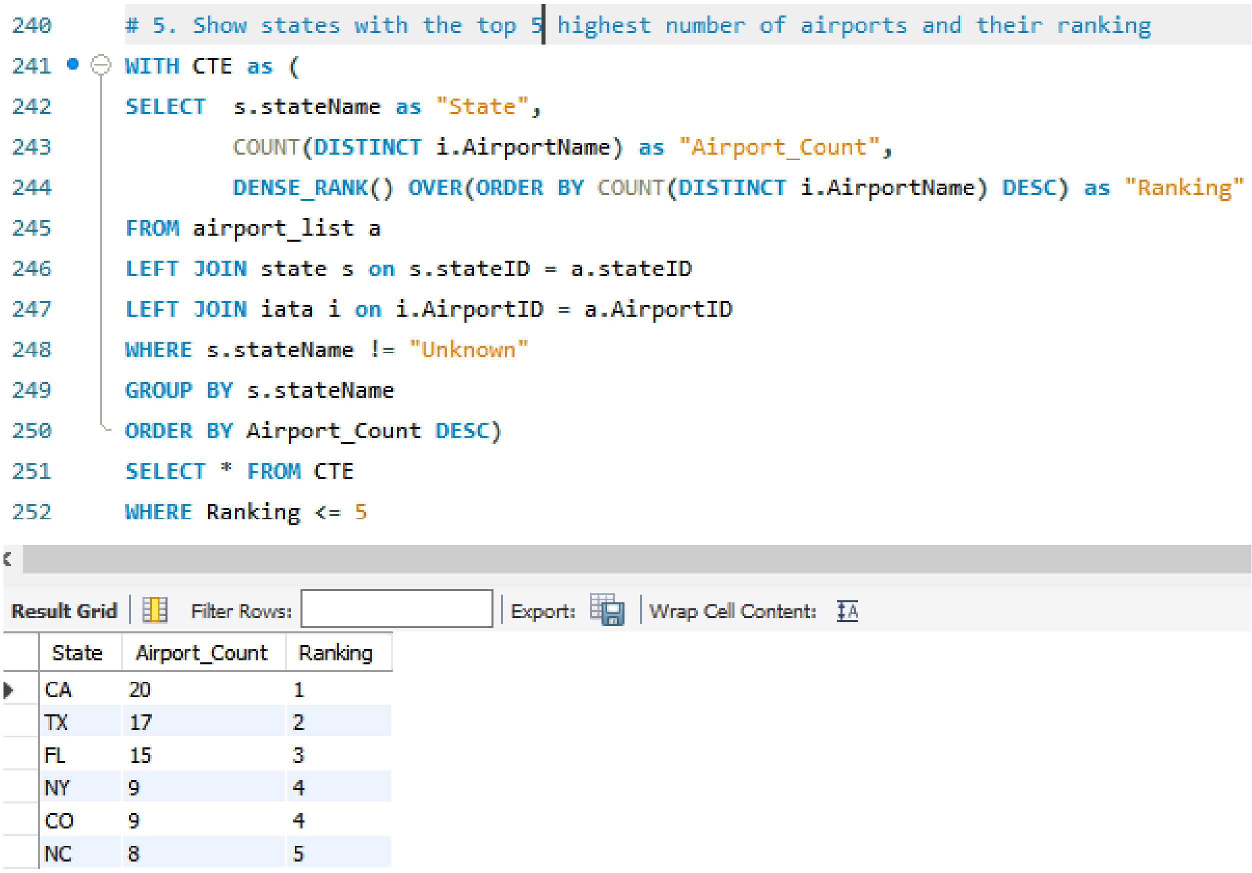


```sql
# 4. Report airports in states whose names start with "M"
SELECT  ROW_NUMBER() OVER(ORDER BY s.stateName, i.AirportName) as "No.",
        i.AirportName as "Airport",
        s.stateName as "State"
FROM airport_list a
LEFT JOIN state s on s.stateID = a.stateID
LEFT JOIN iata i on i.AirportID = a.AirportID
WHERE s.stateName LIKE "M%"
ORDER BY State, Airport;
```

| No. | Airport | State |
|---|---|---|
| 1 | Gen Edw L Logan Intl | MA |
| 2 | Baltimore-Washington International | MD |
| 3 | Portland International Jetport | ME |
| 4 | Bishop | MI |
| 5 | Cherry Capital | MI |
| 6 | Detroit Metropolitan-Wayne County | MI |
| 7 | Kent County International | MI |
| 8 | Duluth International | MN |
| 9 | Minneapolis-St Paul Intl | MN |
| 10 | Rochester International | MN |
| 11 | Branson Airport | MO |
| 12 | Kansas City International | MO |
| 13 | Lambert-St Louis International | MO |
| 14 | Springfield-Branson Regional | MO |
| 15 | Golden Triangle Regional | MS |
| 16 | Gulfport-Biloxi Regional | MS |
| 17 | Jackson International | MS |
| 18 | Key | MS |
| 19 | Tunica Municipal Airport | MS |
| 20 | Billings Logan Intl | MT |
| 21 | Gallatin | MT |
| 22 | Glacier Park Intl | MT |
| 23 | Great Falls Intl | MT |
| 24 | Helena Regional | MT |

There are 24 airports coming from states whose names start with "M".

**Q5 query:**



```sql
# 5. Show states with the top 5 highest number of airports and their ranking
WITH CTE as (
SELECT  s.stateName as "State",
        COUNT(DISTINCT i.AirportName) as "Airport_Count",
        DENSE_RANK() OVER(ORDER BY COUNT(DISTINCT i.AirportName) DESC) as "Ranking"
FROM airport_list a
LEFT JOIN state s on s.stateID = a.stateID
LEFT JOIN iata i on i.AirportID = a.AirportID
WHERE s.stateName != "Unknown"
GROUP BY s.stateName
ORDER BY Airport_Count DESC)
SELECT * FROM CTE
WHERE Ranking <= 5
```

| State | Airport_Count | Ranking |
|---|---|---|
| CA | 20 | 1 |
| TX | 17 | 2 |
| FL | 15 | 3 |
| NY | 9 | 4 |
| CO | 9 | 4 |
| NC | 8 | 5 |

California, Texas, Florida, New York, Colorado, and North Carolina are among the top 5 states with the highest number of airports in the US.

## III. CONCLUSIONS

Normalization is a crucial part in pre-processing data. It helps minimize redundancy within the data tables as well as making it easier to update and modify data while maintaining their logical constraints. I will look to practice these skills further so as to be more knowledgeable in data normalization and analysis.