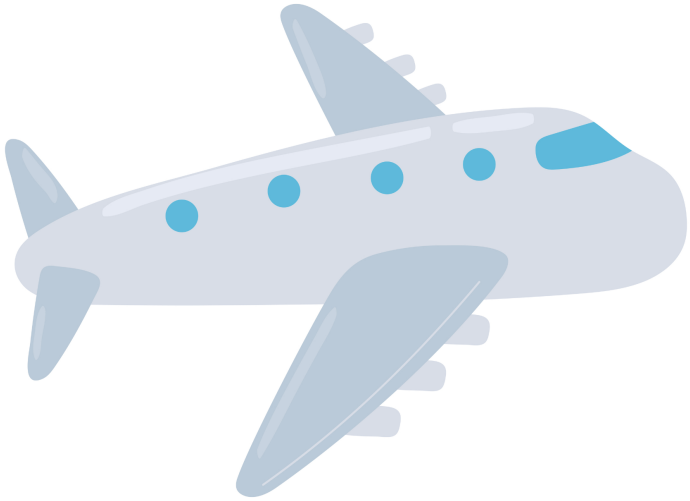
Final Project Phase II



Our application domain is a commercial flight database that data analysts can use to compare how certain factors, such as weather, affect flights in and out of states/cities. The database is also used to observe how these factors affect popular airlines in their activity and stock prices.

Tony Jung (tjung8@jhu.edu)

David Lu (dlu17@jhu.edu)

601.315

https://www.ugrad.cs.jhu.edu/~tjung8/

Github: https://github.com/Lu-David/databases-final-project

Loading the Database

The majority of the database was loaded by finding datasets online in a csv format. A Python script was used to read in the csv data and output SQL insert commands in a SQL file. The airline_company.sql file was written manually by selecting popular airlines and inputting their specifications. Accident_reader.py was used to read in accidents data, airport_reader.py was used to read in the airport data, disaster_reader.py was used to read in disaster data, flight_reader.py was used to read in flight, cancels, and delay data, fuel_reader.py was used to read in fuel data, and weather_reader.py was used to read in weather data and city locations. Due to the extensive amount of flight data, we randomly sampled 1% of the total flights per month.

Dataset	**Link**
Accidents	https://www.ntsb.gov/Pages/AviationQuery.aspx#:~:text=The%20NTSB%20aviation%20accident%20database,few%20days%20of%20an%20accident
Airport	https://data.world/ourairports/989444cc-447b-4030-a866-57fcd6c2d3ee/workspace/file?filename=list-of-airports-in-united-states-of-america-hxl-tags-1.csv
Canceled/Delays/Flights	https://www.transtats.bts.gov/DL_SelectFields.aspx?gnoyr_VQ=FGJ&QO_fu146_anzr=b4vtv0%20n0q%20Qr56v0n6v10%20f748rB
City Locations	https://www.kaggle.com/datasets/selfishgene/historical-hourly-weather-data
Disasters	https://www.kaggle.com/datasets/headsortails/us-natural-disaster-declarations
Fuel Costs	https://www.transtats.bts.gov/fuel.asp
Stocks	https://finance.yahoo.com/
Weather	https://www.kaggle.com/datasets/selfishgene/historical-hourly-weather-data

How To Use

Running the Python Scripts

For accident_reader.py, airport_reader.py, flight_reader.py, and fuel_reader.py, all a user needs to do is to make sure that the datasets that they want to add is correctly listed in the main function and to run the python script (python {name}_reader.py). Assuming that the path is correctly specified, the program will output insert commands into a SQL file. As for weather_reader.py, the script takes in datasets from google drive instead of a directory on one's device. You can find the csv files at this google drive folder.

Using the Interface

To use the interface, first visit https://www.ugrad.cs.jhu.edu/~tjung8/. There are a variety of query options that a user can choose from. Entering a date and city into city specific data will show the user weather data and flight information in airports at the selected city. This can also be generalized to state data using the state specific query. Furthermore, inputting a year and airline company into the airline data query will provide flight data respective to the chosen airline over the course of the given year, as well as stock data to show how a company's stock may correlate to flights, delays, and cancellations.

Specialization

- **Visualization:** We used jpgraph to create plots of our data for easy visualization and analysis of our dataset(s)
- **Query Optimization.** We created non-primary key indexes to speed up selections and joins.
- **Data Extraction.** We used python libraries to efficiently upload 1 million + rows of data into our database.

Selling Points

1) Data Analysts may find our dashboard useful because it condenses relevant information into graphical plots that are labeled, descriptive, and insightful.
2) Additionally, our dashboard is easy to use and navigate, making it accessible for those who may be unfamiliar with SQL to query data.

3) We explore many unconventional and interesting joins (such as natural disasters with cancellations and airlines with accidents).
4) We work with high volume data. We imported hourly weather from 30 plus cities over 5 years and we had to find efficient ways of querying such tables. We have two years of flight data.

Limitations and Areas of Improvement

As stated above, we randomly sampled from the flight dataset due to the copious amount of data that interfered with our ability to load the database and run queries. We also randomly sampled from the number of cancellations per month as well. This may have led to loss of important data points that may have helped delineate ambiguities in correlations between different factors. For example, keeping the data may have shown a clearer relation between number of delays and extreme weather. Another limitation was that we were unable to add certain features that may have helped in optimizing our queries. We were unable to add certain foreign key constraints due to missing data. For example, we would have liked to add a foreign key constraint for flights's origin/destination to reference the airport code; however, we couldn't find an airport dataset online that contained all corresponding airports. Therefore, this inhibited our ability to optimize by adding a foreign key. In addition to the inability to implement keys due to lack of data, missing data also restricted potential for certain queries and joins. We were unable to find a comprehensive city location dataset that could have allowed us to join and query more data for a variety of cities, rather than just the ones we have now. Furthermore, due to the fact that we implemented the database on dbase.cs.jhu.edu, we were not able to add indexes because the server forbids access to those permissions.

Taking all these limitations into consideration, we can improve our database by having comprehensive datasets that would allow us to add more foreign keys. We can also improve the database by using a database system that grants permissions for implementing indexes, which would allow us to populate our dataset with the entirety of the flight data as it could potentially optimize queries to the point that it would be feasible to have an extensive dataset. Another point for improvement is that we could

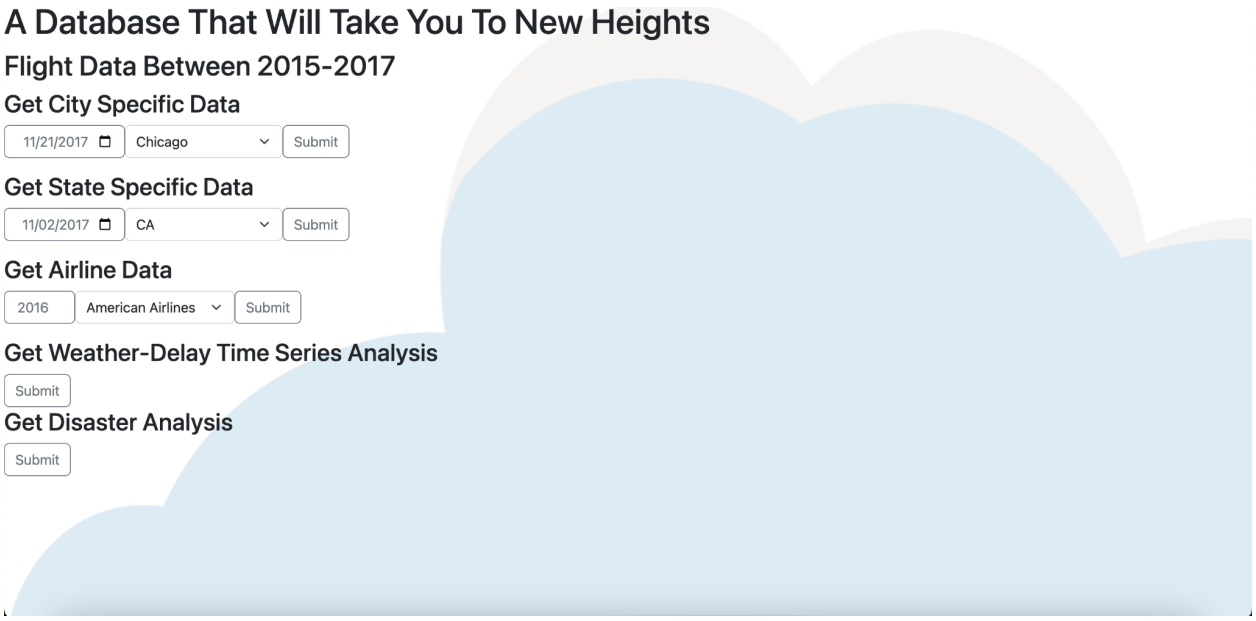
add additional relevant tables that may correlate with some of our data. For example, we could potentially include employment data for each airline company and observe how it correlates with stocks and flights over the course of a year.

Components

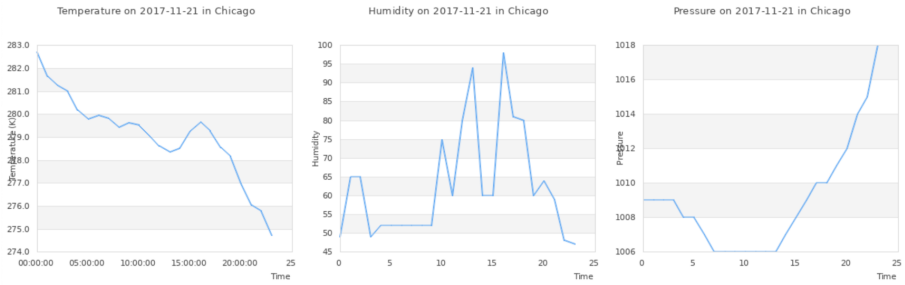
We used jpgraph to create barplots, line graphs, and scatter plots. Source to jpgraph is here: https://jpgraph.net/download/. We copied and pasted relevant files into our frontend page.

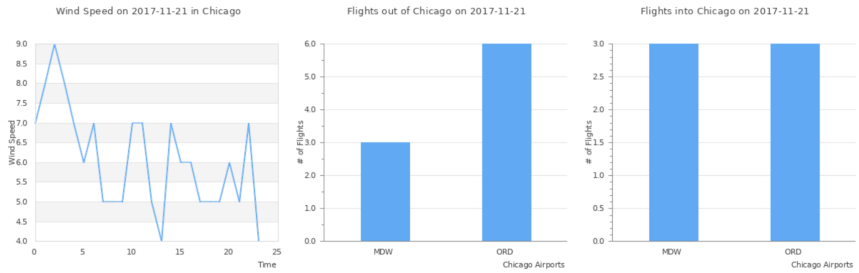
Output

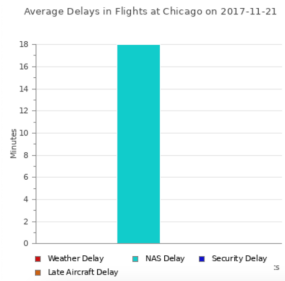
Homepage



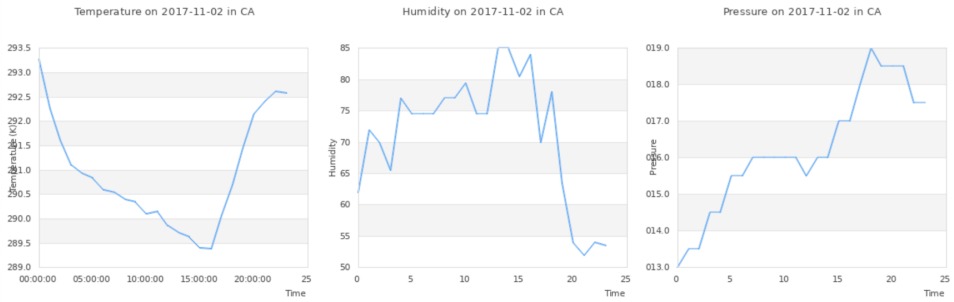
City Data Analysis Page

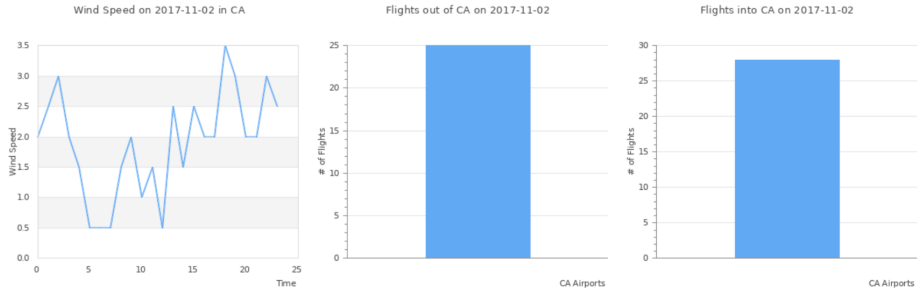




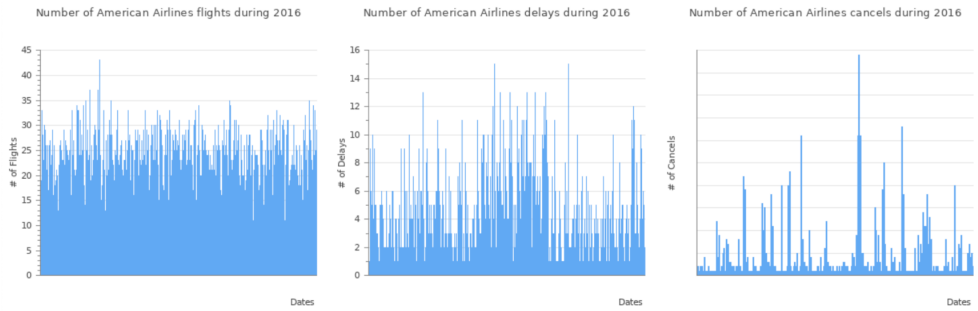


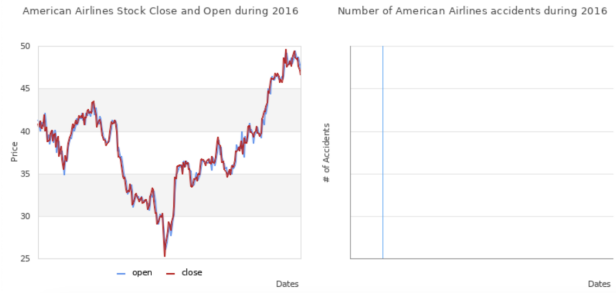
State Data Analysis Page



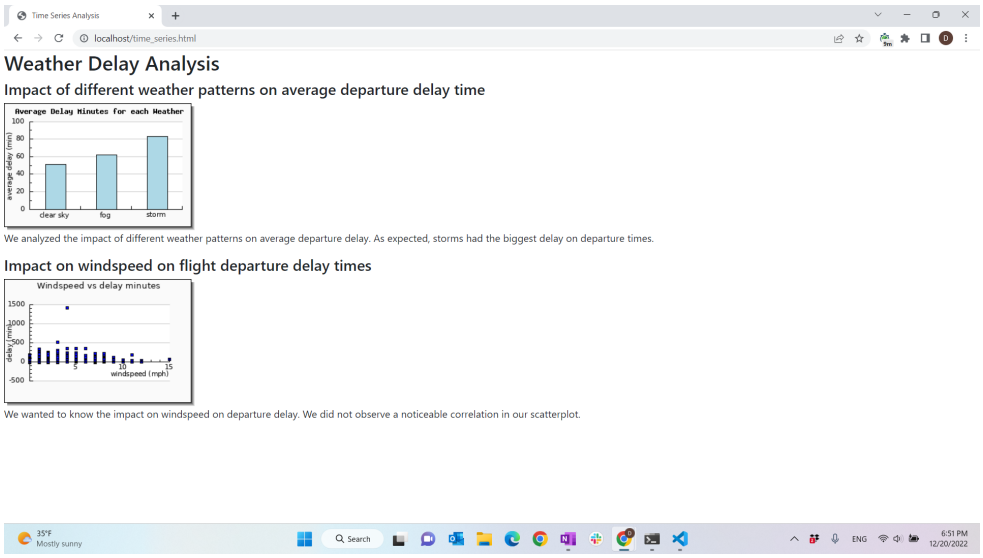


Airline Data Anlaysis Page

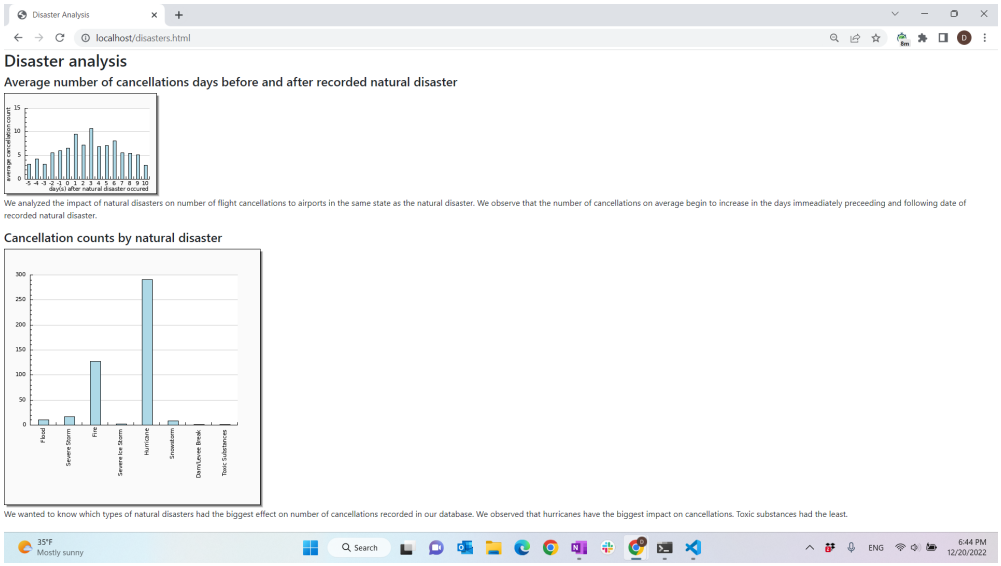




Weather Delay Analysis Page



Disaster Cancelled Analysis Page



Relational Table

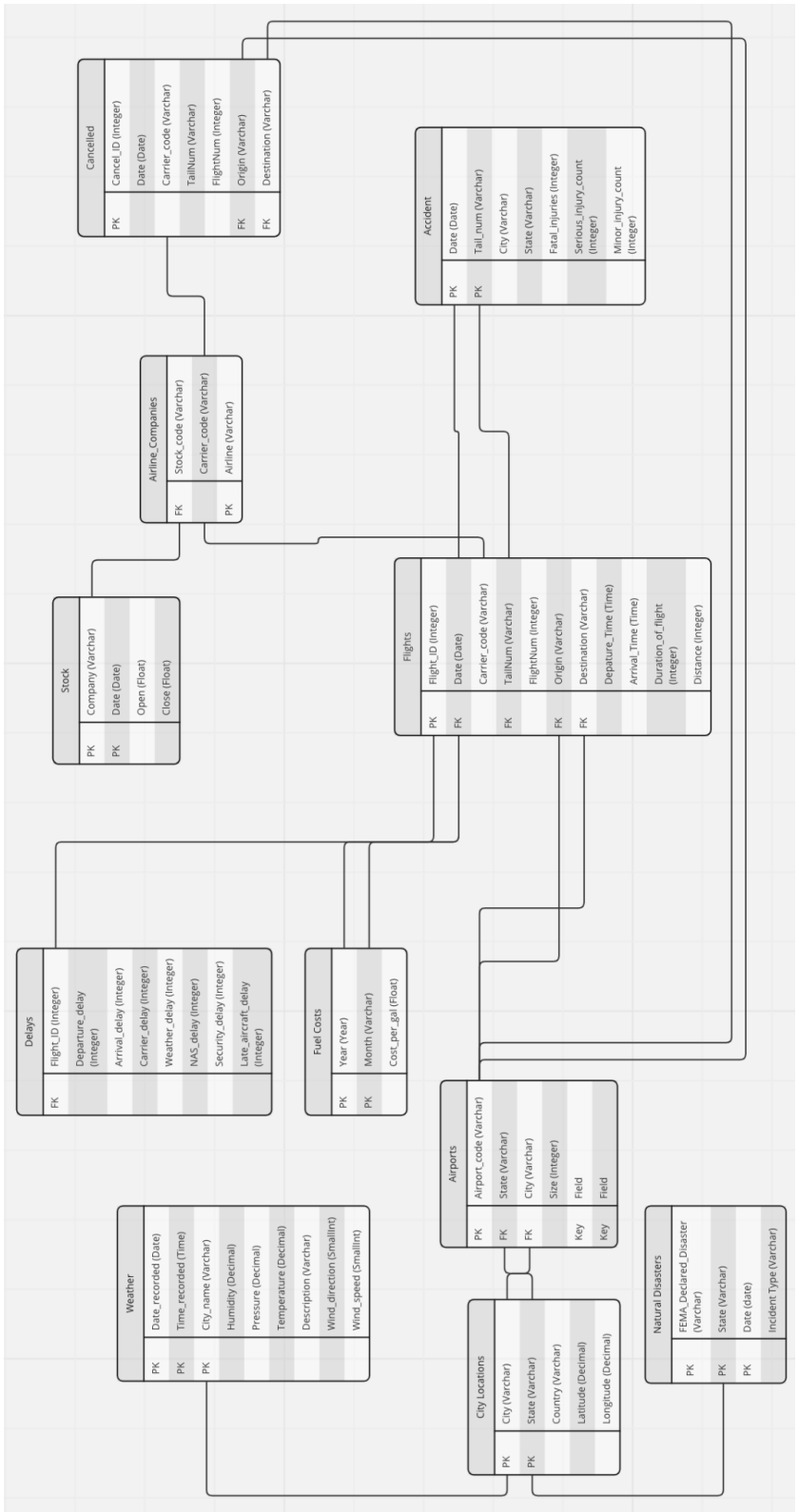
```
CREATE TABLE accidents (
        date                    DATE,
        tail_num                VARCHAR(100),
        city                    VARCHAR(100),
        state                   VARCHAR(100),
        fatal_injuries          INTEGER,
        serious_injury_count    INTEGER,
        minor_injury_count      INTEGER,
        primary key             (date, tail_num)
);

CREATE TABLE city_locations (
        city                    VARCHAR(100),
        state                   VARCHAR(2),
        country                 VARCHAR(100),
        latitutde               DECIMAL(10, 5),
        longitude               DECIMAL(10, 5),
```

```
    primary key         (city, state)
);

CREATE TABLE disasters (
    fema_declaration_id     VARCHAR(100),
    state                   VARCHAR(2),
    date                    DATE,
    incident_type           VARCHAR(100),
    primary key             (fema_declaration_id, state, date)
);

CREATE TABLE fuel_costs (
    year                VARCHAR(4),
    month               VARCHAR(10),
    cost_per_gal        FLOAT,
    primary key         (year, month)
);

CREATE TABLE stocks (
    company             VARCHAR(100),
    date                DATE,
    open                FLOAT,
    close               FLOAT,
    primary key         (company, date)
);

CREATE TABLE airline_company (
    stock_code          VARCHAR(100),
    carrier_code        VARCHAR(2),
    airline             VARCHAR(100),
    primary key         (airline),
```

```
    foreign key         (stock_code)
    references          stocks(company)
);

CREATE TABLE airports (
    airport_code        VARCHAR(3),
    state               VARCHAR(2),
    city                VARCHAR(100),
    size                VARCHAR(100),
    primary key         (airport_code)
);

CREATE TABLE weather (
    date_recorded           DATE,
    time_recorded           TIME,
    city_name               VARCHAR(100),
    humidity                DECIMAL(5, 2),
    pressure                DECIMAL(7, 2),
    temperature             DECIMAL(32, 16),
    description             VARCHAR(1000),
    wind_direction          SMALLINT,
    wind_speed              SMALLINT,
    primary key             (date_recorded, time_recorded, city_name)
);

CREATE TABLE cancelled (
    cancel_id           INTEGER,
    date                DATE,
    carrier_code        VARCHAR(2),
    tail_num            VARCHAR(100),
    flight_num          INTEGER,
```

```
        origin              VARCHAR(3),
        destination         VARCHAR(3),
        primary key         (cancel_id)
);

CREATE TABLE flights (
        flight_id           INTEGER,
        date                DATE,
        carrier_code        VARCHAR(2),
        tail_num            VARCHAR(100),
        flight_num          INTEGER,
        origin              VARCHAR(3),
        destination         VARCHAR(3),
        departure_time      TIME,
        arrival_time        TIME,
        duration_of_flight  INTEGER,
        distance            INTEGER,
        primary key         (flight_id)
);

CREATE TABLE delays (
        flight_id           INTEGER,
        departure_delay     INTEGER,
        arrival_delay       INTEGER,
        carrier_delay       INTEGER,
        weather_delay       INTEGER,
        NAS_delay           INTEGER,
        security_delay      INTEGER,
        late_aircraft_delay INTEGER,
        foreign key         (flight_id)
                references      flights(flight_id)
```

);


Cancelled
PK Cancel_ID (Integer)
Date (Date)
Carrier_code (Varchar)
TailNum (Varchar)
FlightNum (Integer)
FK Origin (Varchar)
FK Destination (Varchar)
Accident
PK Date (Date)
PK Tail_num (Varchar)
City (Varchar)
State (Varchar)
Fatal_injuries (Integer)
Serious_injury_count (Integer)
Minor_injury_count (Integer)
Airline_Companies
FK Stock_code (Varchar)
Carrier_code (Varchar)
PK Airline (Varchar)
Stock
PK Company (Varchar)
PK Date (Date)
Open (Float)
Close (Float)
Flights
PK Flight_ID (Integer)
FK Date (Date)
Carrier_code (Varchar)
FK TailNum (Varchar)
FlightNum (Integer)
FK Origin (Varchar)
FK Destination (Varchar)
Depature_Time (Time)
Arrival_Time (Time)
Duration_of_flight (Integer)
Distance (Integer)
Delays
FK Flight_ID (Integer)
Departure_delay (Integer)
Arrival_delay (Integer)
Carrier_delay (Integer)
Weather_delay (Integer)
NAS_delay (Integer)
Security_delay (Integer)
Late_aircraft_delay (integer)
Fuel Costs
PK Year (Year)
PK Month (Varchar)
Cost_per_gal (Float)
Airports
PK Airport_code (Varchar)
FK State (Varchar)
FK City (Varchar)
Size (Integer)
Key Field
Key Field
Weather
PK Date_recorded (Date)
PK Time_recorded (Time)
PK City_name (Varchar)
Humidity (Decimal)
Pressure (Decimal)
Temperature (Decimal)
Description (Varchar)
Wind_direction (Smallint)
Wind_speed (Smallint)
City Locations
PK City (Varchar)
PK State (Varchar)
Country (Varchar)
Latitude (Decimal)
Longitude (Decimal)
Natural Disasters
PK FEMA_Declared_Disaster (Varchar)
PK State (Varchar)
PK Date (date)
Incident Type (Varchar)