# Semester Project 4

## **GROUP MEMBERS:**

### Krzysztof Majcher (253784)

### Maja Petrusic (253899)

### Maria José Ferreira (254175)

### Sara Nunes (254272)

## **Supervisors:**

### Bo Brunsgaard

### Richard Brooks

# VIA UNIVERSITY COLLEGE

# INFORMATION TECHNOLOGY COMMUNICATION ENGINEERING

# DATA ENGINEEring

# 4TH SEMESTER

# 07/06/2018

Table of Content

[Abstract 3](#_Toc516169391)

[Introduction 4](#_Toc516169392)

[Requirements 5](#_Toc516169393)

[Functional-requirements 5](#_Toc516169394)

[Non-functional requirements 5](#_Toc516169395)

[ANALYSIS 6](#_Toc516169396)

[DESIGN 7](#_Toc516169397)

[IMPLEMENTATION 11](#_Toc516169398)

[TEST 23](#_Toc516169399)

[Results and discussion 25](#_Toc516169400)

[Conclusion 26](#_Toc516169401)

[References 27](#_Toc516169402)

[Appendix 28](#_Toc516169403)

[Appendix A: Data base ER Diagram 28](#_Toc516169404)

[Appendix B: ER Diagram for Dimensional Model 28](#_Toc516169405)

# abstract

This project represents a solution to a problem presented to us by our teacher, Bo Brunsgaard Christensen. As a pilot himself, Bo flies gliders and utilizes rising air (thermals) in the atmosphere to remain airborne. Gliders are aircraft design to fly without an engine. Their areodetic construction and low weight makes it possible. However, Gliders need some source of energy to stay in air therefore pilots needs to find tubes of hot air. By circulating in areas where hot spots appear pilots can increase their altitude.

The task was to build the necessary data structures and processes to make it possible to analyze flight and weather data to identify possible thermals in Denmark.

The mapping and implementation of a data warehouse and its related process ETL system makes it possible to create this system by standardizing flight data and comparing it to weather, thermal and date data.

The Data warehouse system fits both the club members database and thermals location. Our ETL transforms the data from a source table that contains log files and loads in the database and it’s prepared to receive more log files in the future and handle new data.

The result of the system is a user-friendly that allows people who have no database knowledge to analyze the data by having a map with all the locations and the transformed data. The main user will be pilots that want to fly longer distances by predicting the thermals in the location and we expect them not to have a database knowledge.

# Introduction

Gliding is a recreational activity and competitive air sport in which pilots fly unpowered

aircraft known as gliders or sailplanes using naturally occurring currents of rising air in

the atmosphere to remain airborne.

In Denmark there are annual gliding competitions. To be part of the competition

it is necessary to have experience and authorization to flight a glider. To win the

competition it is not only necessary to have good performance skills, but, to flight longer

distances. Competitive soaring flight involves flying set tasks of between 100 and 500+

km in gliders. This involves alternating phases of altitude gain in columns of rising hot

air and gliding flight by converting altitude to distance.

Thermals can form when sunlight heats up some spot on the Earth's surface more than

the surroundings. This heat is reflected to the air, which warms up. At some point,

the heated air is sufficiently warmer than the surrounding air, and a "bubble" of warm

air detaches from the surface and rises. The secret to soaring flight is to be able to

locate those bubbles and gain altitude by circling in them. The secret to successful

competitive flight is to know beforehand where the thermals will form.

These thermals occur in Denmark but their locations are unknown because that can

change depending on the weather. There is not a specific map with all the possible

thermal locations in Denmark. Therefore, the pilots that participate in competitions

cannot be certain about how far they will go because, even if they have found thermals

in previous flights, that doesn't guarantee that they will find the thermals

in the same location.

It is difficult to forecast where the thermals can be located before starting a flight, but

there is information about the location of flights where the thermals occurred and there

is also weather condition information during the days that the thermals occurred. That

information can be matched to make a possible thermal forecast for the pilots.

# REQUIREMENTS

### Functional-requirements

* Have an estimation of all the hotspots in Denmark found based on the coordinates where they are found repetitive times.
* The solution should include a way to match the thermals and the weather data

### Non-functional requirements

* The data will be received as igc and txt files where a line has multiple information and must be described as attributes in the respective tables.
* The system should work with existing and future log files.
* The system should keep track of changes and updates.

# ANALYSIS

The goal of the analysis phase in this project was to gain a deep understanding of the problem.

The problem in this project was that pilots need to find thermals to stay in the air for longer and

To represent the entities there were identified three entities key on the resolution of the problem: “Weather”, “Flight” and “Thermal”

The thermal it’s defined by the following attributes: “id” with the unique key of the thermal, “flight\_id” with the unique id of a flight, “date found” with the date when the thermal was found and with the “maximum latitude” and “minimum latitude” and “minimum longitude” and “maximum longitude” of the area where the thermal was found.

To find these thermal tubes it’s necessary to have the flight data because the thermals are in the location where the plane’s altitude started to increase. For the values of the considered altitude, there must be some constrains to separate altitude increase caused by starts or other maneuvers and increases caused by pilot flying in thermal tubes. The altitude increase could be a well caused by increasing glider bill angle and transforming velocity into altitude. Considering these facts, the following constraints has been established. It is possible to for pilots to find thermals on altitude lower then altitude after start however such an event will be considered as special circumstances and ignored in this project.

For the flight entity it was defined that from the log files the information should be separated into the “id” that is the unique key for the flight, “flight id” with the key from the source file, “gps altitude” with the altitude of the plane registered on the gps in meters, “pressure altitude” with the altitude measured by pressure in meters, “satellite coverage” with the area the satellite can cover, “position longitude” with the longitude registered, “position latitude” with the position latitude and “log time” with current time registered UTC.

Also, it’s necessary to have the locations of the thermals that appear constantly during flights and depending on the weather so for that it’s necessary to match the thermals with the weather data and for that the log weather files had to be interpreted.

After interpreting the weather files, it was defined that they will be categorized be “airport” with the issuing airport code, “dew point temperature” is the temperature to which air must be cooled to become saturated, “cloud cover” with the fraction of the sky obscured by clouds, “pressure” with the pressure, “visibility” with the visibility in meters and the codes, “wind direction” with the wind direction in knots, “wind speed” in degrees and “date time” with the current time UTC and date.

The implementation of a Data Warehouse, in this case inputting data from different flights and weather to then match it and calculate thermals seems to solve the problem of handling the data and analyze it since a data Warehouse stores the data in a way that is optimal for reporting and other analysis and the data in it is available to develop in specific business purposes. Combining the data from all the other databases in the environment, the data warehouse becomes the single source for user to obtain data and maintain historical data.

The system will have to be flexible so in the future it can take new files.

The next stage was to analyze the ETL Flow and how it would interact and use the data in our system.

The next stage was the design phase

# DESIGN

The design phase had two major steps: The Database structure and the Data Warehouse and ETL Process.

The first phase was the Database structure, it must be decided on what tables the source database needs to have and what attributes the tables should have. The data had to be obtained by the log files that contained the information in sections, so the attributes were based on the log files and the information used from the log files was the one required to make the program.

The tables were as follows: “Weather”, “Thermal”, “Flight”.

|  |  |
| --- | --- |
| Table description | Table Name |
| Table where the data from the weather log files is stored | Weather |
| Table where the results of the thermal data calculations are stored | Thermal |
| Table where the data from the flights log files is stored | Flight |

The second phase was the Data Warehouse that needed to be designed. It was determined that the dimensional model would contain the following dimensions and facts:

|  |  |  |
| --- | --- | --- |
| Table Name | Table Description | Related Dimensions |
| D\_Date | Dimension date containing all the information needed about a certain date |  |
| F\_Weather | Fact weather with transformed data | D\_Date |
| F\_Thermal | Fact thermal with normalized thermals | D\_Flight |
| D\_Grid | Dimension grid that stores the different grids of the map |  |
| D\_Flight | Dimension flight with optional log names |  |
| F\_Flight | Fact flight that stores transformed information of a flight | D\_Flight, d\_launch\_method, bridge\_mf |
| D\_Launch\_Method | Dimension Launch Method that has the type of launch method of a flight |  |
| F\_Ownserhip | Fact ownership that contains information about the plane, the owner and club | D\_Member, D\_Date |
| F\_Membership | Fact membership that contains | D\_Club, D\_Date |
| D\_Club | Dimension club that contains the information about a club |  |
| Bridge\_mf | Bridge between the many to many relationships between fact flight and dimension member | D\_member |
| D\_Age | Dimension age that keeps track of the change of the age of member | Bridge\_mf |
| D\_Audit | Dimension audit that keeps track of all the rows that were fixed or rejected |  |

The dimension flight was made to connect the thermal to a flight and the log name is an optional field to recognize which was the log file the flight is in. The model done in the data warehouse course didn’t match with the new data from the flight logs. Therefore, the f\_flight was considered just the flight duration and the d\_flight stored the actual information (only log\_name for the purposes of this project), and an id for the flight.

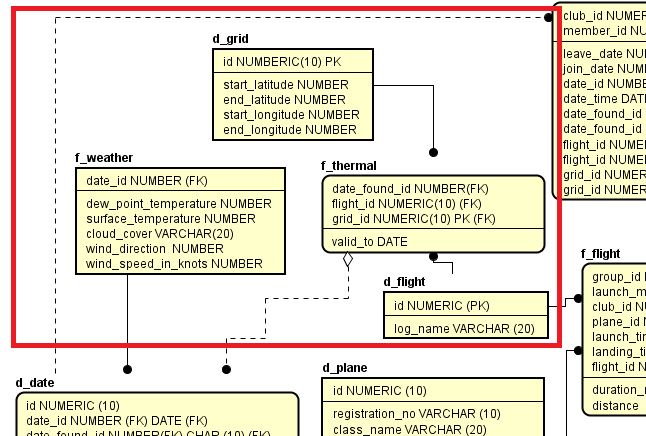
The weather fact was designed to store the relevant weather information such as dew point temperature, surface temperature, cloud cover, and wind speed and direction. The weather happens during different dates and times; therefore, it has a foreign key to the d\_date dimension.

The grid dimension was designed to store the latitudes and longitudes in Denmark divided into grids. Each grid has a unique id and a unique location with start and end longitude and start and end longitude.

The thermal fact was designed to store the location of a specific flight when it found the respective thermal. Therefore, the thermal keeps a foreign key to the flight dimension, a foreign key to the grid with the id that matches its location and a foreign key to the d\_date id that contains the date and time when the thermal was found. The thermal fact also contains a date column called valid\_to because the thermal for this project have a duration of five years in the data warehouse.

The d\_audit was updated with new columns to store the changes in the thermal fact and flight dimension.

The following image shows the diagram with the new dimensions and facts.



The complete diagram with all the tables is located in the Appendices.

# IMPLEMENTATION

**Populating the Database**

1. **Populating table Flight**

The next step was populating the tables created. For the table Flight, there was a system that reads data form given igc log files and inserts them into database. This part requires both implementation of table and Java program starting with table. IGC logs contains more information’s then this project needs so the system is ignoring lines that don’t start with letter “B” which contains glider altitude and position or with “HFDTE” which contains date of flight. In a first steps program is listing all files that folder logs contain and then loop thru them inserting data that “B” line contains with flight data and log file name. After reading file is compiled the system move file to separated folder.

1. **Populating table Weather**

The weather log files had to be handled as well, so the first step was to create a Weather table. Then it was necessary to find a way to translate the information on the log files files and store it in the database table Weather. The implementation solution was to build three different java classes.

The first one is the class Weather that declares all the different attributes and sets the constructor of the class with its attributes that will be used on the WeatherConverter and to return these attributes.

Weather.java

**public** **class** Weather

{

**private** **double** pressure;

**private** **double** dewPointTemperature;

**private** **double** surfaceTemperature;

**private** String cloudCover;

**private** String visibility;

**private** **double** windDirection;

**private** **double** windSpeed;

**private** **int** DD;

**private** **int** HH;

**private** **int** MM;

**private** String airport;

**public** Weather(String airport, **int** dD, **int** hH, **int** mM,**double** windDirection, **double** windSpeed,

String visibility, String cloudCover, **double** surfaceTemperature, **double** dewPointTemperature, **double** pressure) {

**this**.airport = airport;

DD = dD;

HH = hH;

MM = mM;

**this**.windDirection = windDirection;

**this**.windSpeed = windSpeed;

**this**.visibility = visibility;

**this**.cloudCover = cloudCover;

**this**.surfaceTemperature = surfaceTemperature;

**this**.dewPointTemperature = dewPointTemperature;

**this**.pressure = pressure;

}

**public** **double** getPressure() {

**return** pressure;

}

The second class is called Database Communication where the connection to the instance of the database is created and the method to insert a prepared statement into it, InsertWeather.

DatabaseCommunication.java

**public** **class** DatabaseCommunication

{

**private** **static** DatabaseCommunication *instance*;

**private** **final** **static** String ***connectString*** = "jdbc:oracle:thin:@localhost:1521:orabbc12c"; // orabbc12c

**private** **final** **static** String ***userName*** = "sep";

**private** **final** **static** String ***password*** = "sep";

**private** **static** Connection *conn*;

**private** DatabaseCommunication()

{

**try**

{

DriverManager.*registerDriver*(**new** oracle.jdbc.OracleDriver());

*conn* = DriverManager.*getConnection*(***connectString***, ***userName***, ***password***);

*conn*.setAutoCommit(**false**);

}

**catch** (SQLException e)

{

e.printStackTrace();

}

}

**public** **static** DatabaseCommunication getInstance()

{

**if** (*instance* == **null**)

{

*instance* = **new** DatabaseCommunication();

}

**return** *instance*;

}

**public** **void** insertWeather(Weather weather, String fileName, **int** month, **int** year) **throws** SQLException

{

Statement statement = *conn*.createStatement();

statement.executeQuery("insert into weather "

+ "(ID, PRESSURE, DEW\_POINT\_TEMPERATURE, SURFACE\_TEMPERATURE, CLOUD\_COVER, VISIBILITY, ISSUING\_AIRPORT, WIND\_DIRECTION, WIND\_SPEED, DATE\_TIME)"

+ " values (idWeatherSequence.nextval" + "," + weather.getPressure() + ","

+ weather.getDewPointTemperature() + "," + weather.getSurfaceTemperature() + ",'" + weather.getCloudCover()

+ "'" + ",'" + weather.getVisibility() + "'" + ",'" + weather.getAirport() + "'" + ","

+ weather.getWindDirection() + "," + weather.getWindSpeed() + ", timestamp '" + (year) + "-" + month + "-"

+ weather.getDD() + " " + weather.getHH() + ":" + weather.getMM() + ":" + 00 + " " + "UTC'" + ")");

statement.close();

*conn*.commit();

}

Last class is called WeatherConverter that parses the weather files lines and its substrings. In this case all the lines that were relevant started with ‘M’ and those were the lines read by the bufferedReader, the lines also had different lengths so there were different cases for the different lengths as the information was in different position on the String. For example, the issuing airport was in the substring starting on the position 6 and ending on the position 10. After reading the file, the weather object, file name, day and month are attributed to the insertWeather prepared statement, the attributes are inserted following the attributes in the table weather, and the database connection is established.

WeatherConverter.java

**public** **class** WeatherConverter

{

**private** File[] files;

**private** DatabaseCommunication db;

**private** **int** day;

**private** **int** month;

**private** **int** year;

**public** WeatherConverter() **throws** IOException

{

File folder = **new** File("weather\_logs");

files = folder.listFiles();

db = DatabaseCommunication.*getInstance*();

**for** (**int** a = 0; a < files.length; a++)

{

System.***out***.println(files[a].getName());

convert(files[a]);

}

}

**public** **void** convert(File file) **throws** IOException

{

FileReader fileReader = **new** FileReader(file);

BufferedReader bufferedReader = **new** BufferedReader(fileReader);

String line;

Weather weather = **null**;

**while** ((line = bufferedReader.readLine()) != **null**)

{

**if** ((line.charAt(0) == 'M' && line.length() < 56))

{

weather = **new** Weather(line.substring(6, 10), Integer.*parseInt*(line.substring(11, 13)),

Integer.*parseInt*(line.substring(13, 15)), Integer.*parseInt*(line.substring(15, 17)),

Double.*parseDouble*(line.substring(24, 27)), Double.*parseDouble*(line.substring(27, 29)),

line.substring(32, 39), line.substring(40, 43), Double.*parseDouble*(line.substring(44, 46)),

Double.*parseDouble*(line.substring(47, 49)), Double.*parseDouble*(line.substring(51, 55)));

**if** (weather != **null**)

db.insertWeather(weather, file.getName().substring(0, file.getName().length() - 4), month, year);

}

**catch** (SQLException e)

{

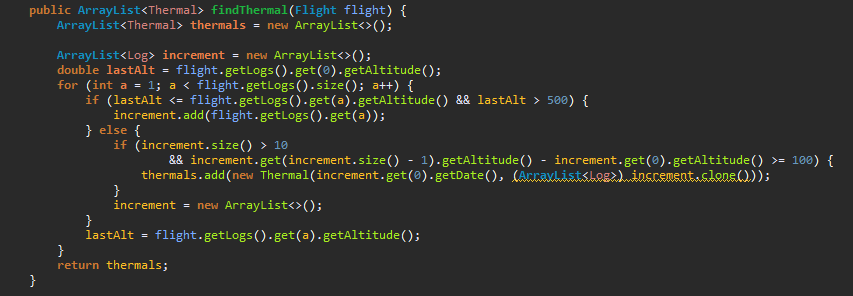
System.***out***.println(line);

e.printStackTrace();

}

1. **Populating Thermal table**

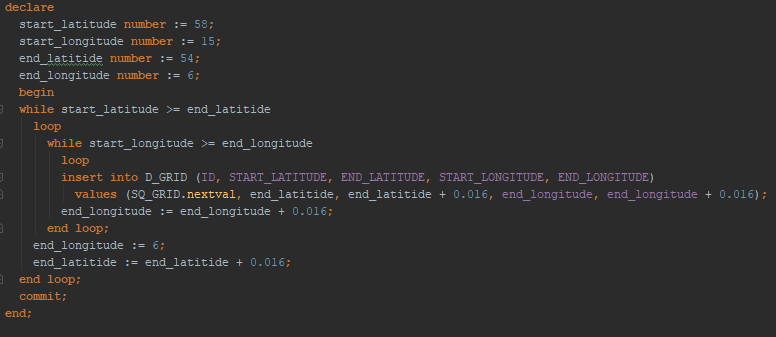
The next set is to determinate thermals and to populate the Thermal table with their locations Java program made for that peruse is reading flight logs data form database and looks for places where pilots are increasing their altitude. System is ignoring event that happened below 500m to avoid reading starts as thermal locations. Also, to avoid reading other maneuvers as thermals program is selecting logs in which increase last longer than 10 second and altitude increase is no less than 100m.



**Implementing the Datawarehouse**

**Populating D\_Grid**

To have a normalized structure of thermals location Denmark map was split into smaller square areas with border of value close to one minute. Every of them needs to be store in dimension grid. To archive it the script to populate table D\_Grid was implemented. The scrip is an inner loop that populate grid starting from point 54°, 6° and ends at 58°, 15°



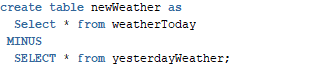
**Populating the date dimension:**

The date dimension was populated with the dates obtained from the flights used. This can always be modified to allow the storage of more flight and weather logs.

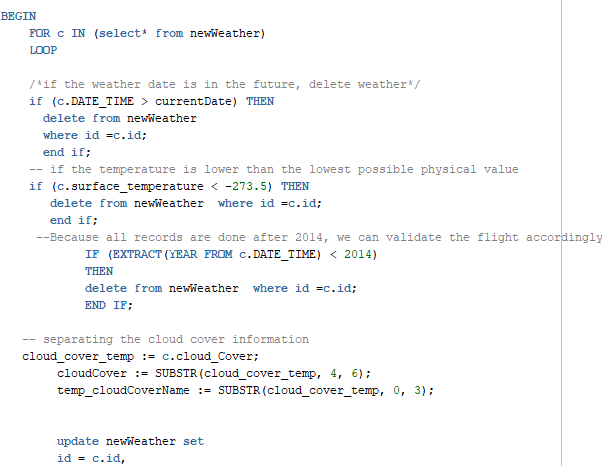
**Implementing the ETL**

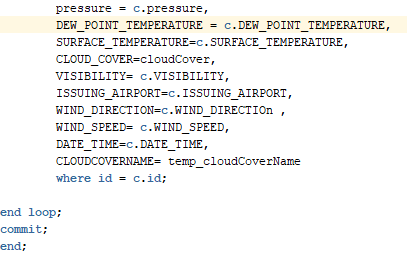
**ETL F\_Weather**

To validate and keep track of the new information in the fact weather a ETL was created. In this ETL, there was a table newWeather where the weather is updated with the new rows obtained in the current day data.



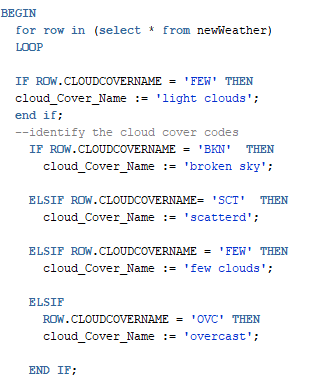
In the next script there’s a loop that selects everything in the weather. Here the date of the weather data is validated by checking if the data is in the future, it also checks if the temperature is inferior than the lowest possible, lastly checks if the weather data is from before 2014 and in the end, it deletes the rows where the statement was true. Next, for separating the cloud cover and the cloud cover name a script for parsing the substring is used and it then updates the new Weather Table with these new variables.





The transformed weather is created. The transformed weather table will identify the different cloud cover codes to make it easier for the user to understand. Then there’s a script for getting date id with the same date found values. These new values will be inserted in the transformed table and the new weather values will be inserted in the other fields. Finally, the transformed table will be inserted in the fact weather









ETL for d\_flight:

The flight dimension had to be created because the previous data model (created for the data warehouse course) did not match the needs for this new data about the thermals. The flight dimension has an id for each row and it includes the log\_name. There is a fact called flight, but for this project f\_flight is considered as flight duration.

During the extract process we take all the flights from the fligh table, which is the results of the java program that converts the log files into proper and understable columns.

The validation only checks for the log\_name, if it has a null value then the name “unknown” is set. the amount of flights fixed are inserted in the audit dimension.

During the transformation phase, the validated values are inserted into the transformed new flights table. then, the rows in that table are loaded to the flight dimension.

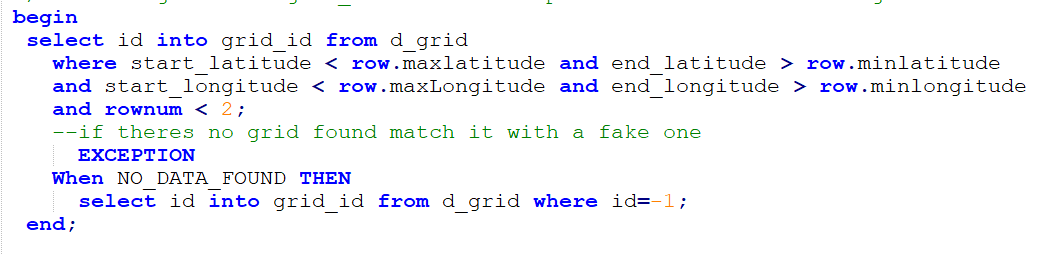
ETL for f\_thermal:

Before inserting rows in the f\_thermal table, an ETL process was run. the source table that we use to get the information to populate the f\_thermal, is the thermal table which is the result of a java program that finds thermals depending of the change of altitudes in the flight table (the log’s information).

In the ETL we check the following:

* The date when the flight was found: if it is in the future, the flight is not considered for the f\_thermal population.
* The date when the flight was found is older than 5 years: if it is before 2013, the flight is not considered for the f\_thermal population because the thermal is valid only for 5 years in this project.
* Latitude and longitude values: there is a validation to check if this value were swapped and if they are valid values. to check if they are valid we check if all of those values are between 0 and 360, and to check if they were swapped we check their values. this validation would only be valid for Denmark because the maximum and minimum Danish altitudes and longitudes are used. the latitude values have to be between 6 and 15, and the longitude values have to be between 54 and 58. the values are swapped if the minimum values are bigger than the maximum and if the latitude values correspond to the longitude values.
* Audit changes: the amount of thermals that were fixed or rejected are stored in the audit dimension.

During the transformation process, we search for the correct flight, date and grid id’s. the following image shows how the grid id is set:



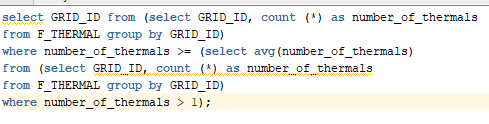
During the loading process we set the transformed and validated values to the f\_thermal and we set the end\_date to five years from the current date.

Visualization of the thermals:

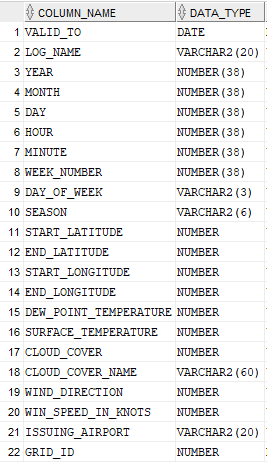
A view was made in the database so that we could use it in the tool called PowerBI which is a business intelligence tool that includes maps and data slicers.

The view has the information from the dimension and facts that were relevant for the final results. for example, it joins the weather information with the thermals and the grid’s location so that the information could be vitualized easier. The view only matches the first weather row for a specific day.

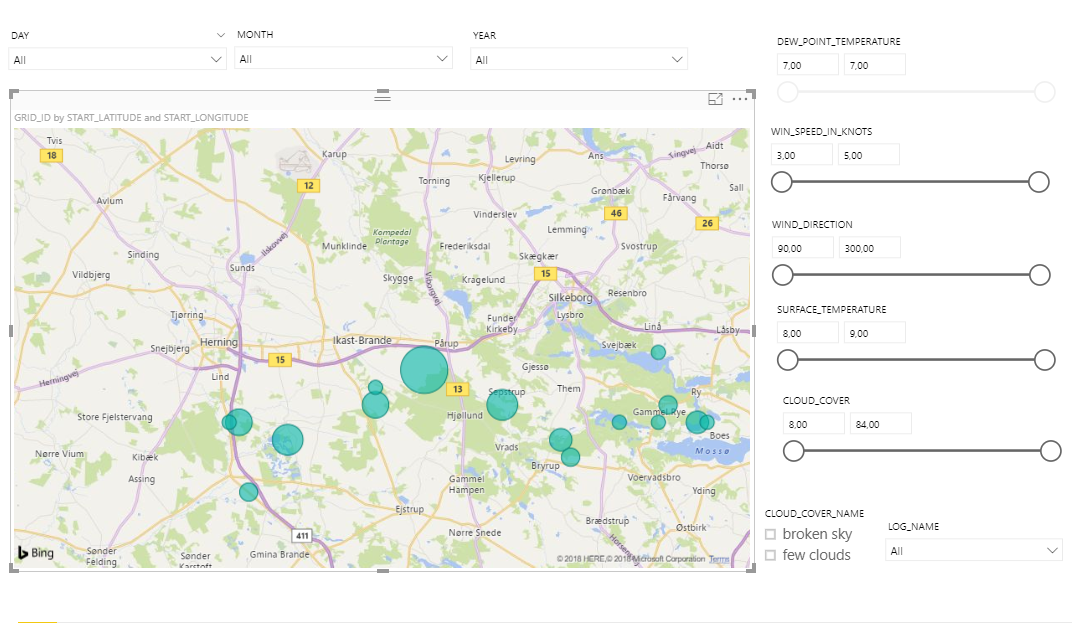
The thermals that appear constantly in the same location are also filtered through this view. the following query shows that the constant used as the location repetition, is an average of the number of thermal in a grid. Then, only the location which has had more than the average of thermals was considered in this view. this can always be changed depending on the analysis purposes of the pilot.



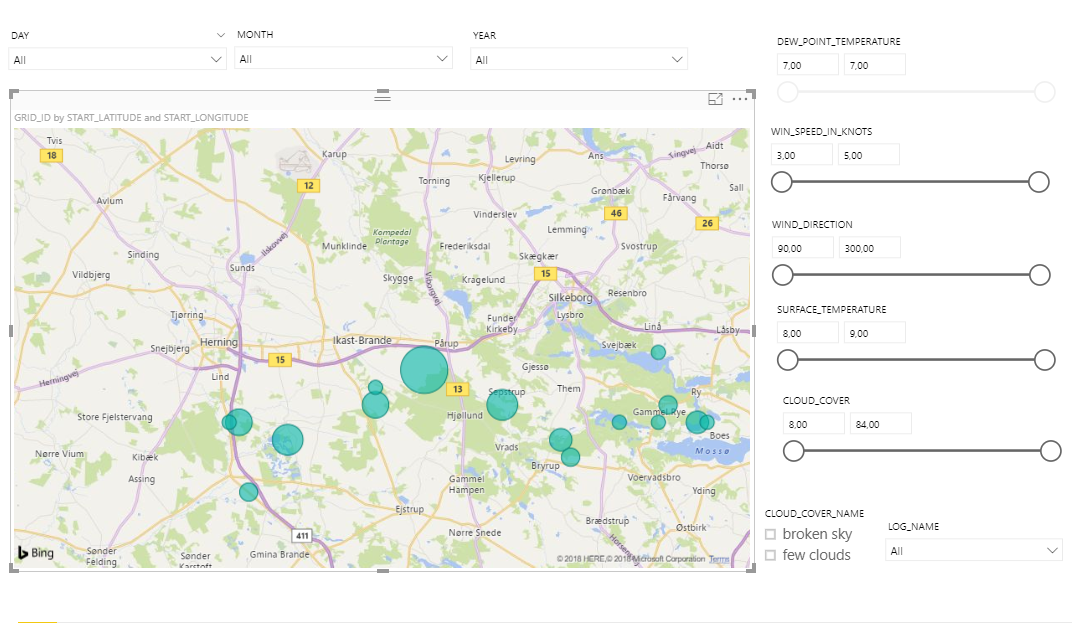
the view has the following columns:



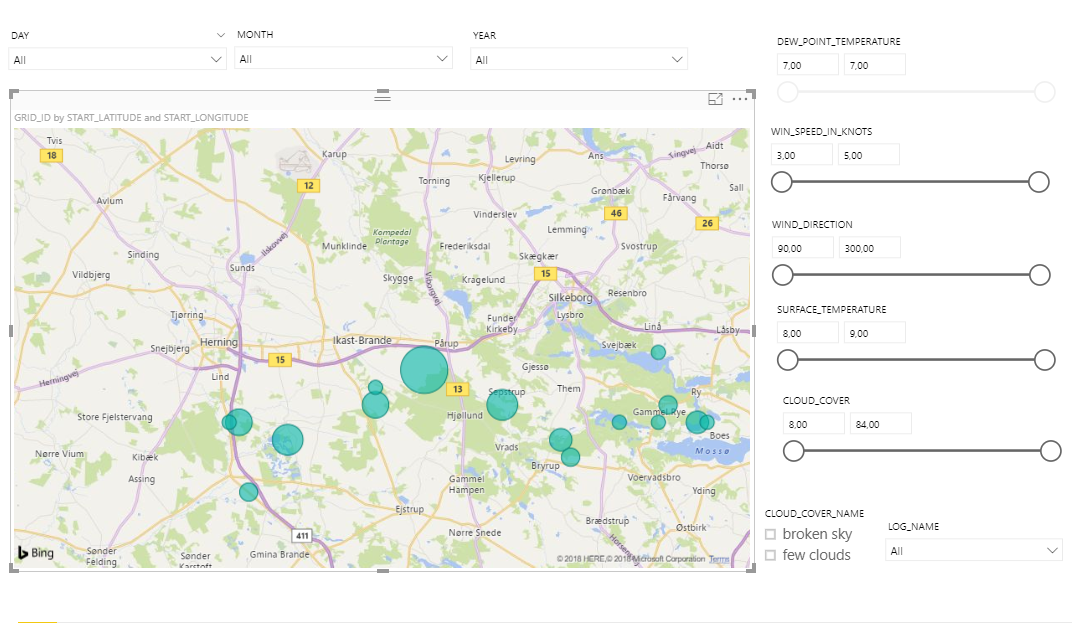
A map was made in powerBI, which showed the grid locations as blue points. the points size depend on the NUMBER of thermals that occurred in that location. Only the thermals location that occurred more than the average of thermals in a specifi location are shown.



Data slicers were used in the visialuzation to make the best out of the tool. There are data slicers that allows a pilot or any user to see the thermals that occurred in a specific day, month or year.



Data slicers were also used with the weather information, so that the pilot can choose the thermals that apeared in a specific weather and expect those thermals in his flight.



# TEST

This test was performed by performing the operations and manually comparing results with the expectations.

Here the following methods from the Flight log were tested: insertLog() and convert() with their arguments.

**Flight log converter** status: Passed

|  |  |
| --- | --- |
| Reading logs | Passed |
| Excreting data form rows | Passed |
| Inserting data to table | Passed |
| Moving files to separated folder | Passed |

**Here the following methods from the Weather log were tested: insertWeather() and convert() with their specified arguments**

**Weather log converter** status: Passed

|  |  |
| --- | --- |
| Reading logs | Passed |
| Excreting data form rows | Passed |
| Inserting data to table | Passed |

Here the following methods from the Thermal finder were tested: getFlights(), insertThermal() and findThermal() with their arguments

**Thermal finder** status: Passed

|  |  |
| --- | --- |
| Reading data form database | Passed |
| Establishing thermals locations | Passed |
| Inserting data to database | Passed |

Here it was tested that the different tables were correctly populated with the transformed data and updates.

**ETL** status: Passed

|  |  |
| --- | --- |
| Populate D\_DATE | Passed |
| Populate D\_GRID | Passed |
| D\_AUDIT | Passed |
| D\_FLIGHT | Passed |
| F\_THERMAL | Passed |
| F\_WETHER | Passed |

Here the visualization was tested in power BI with a view and the test was performed by comparing the results with the map and igc viewer.

**Visualization** status: Passed

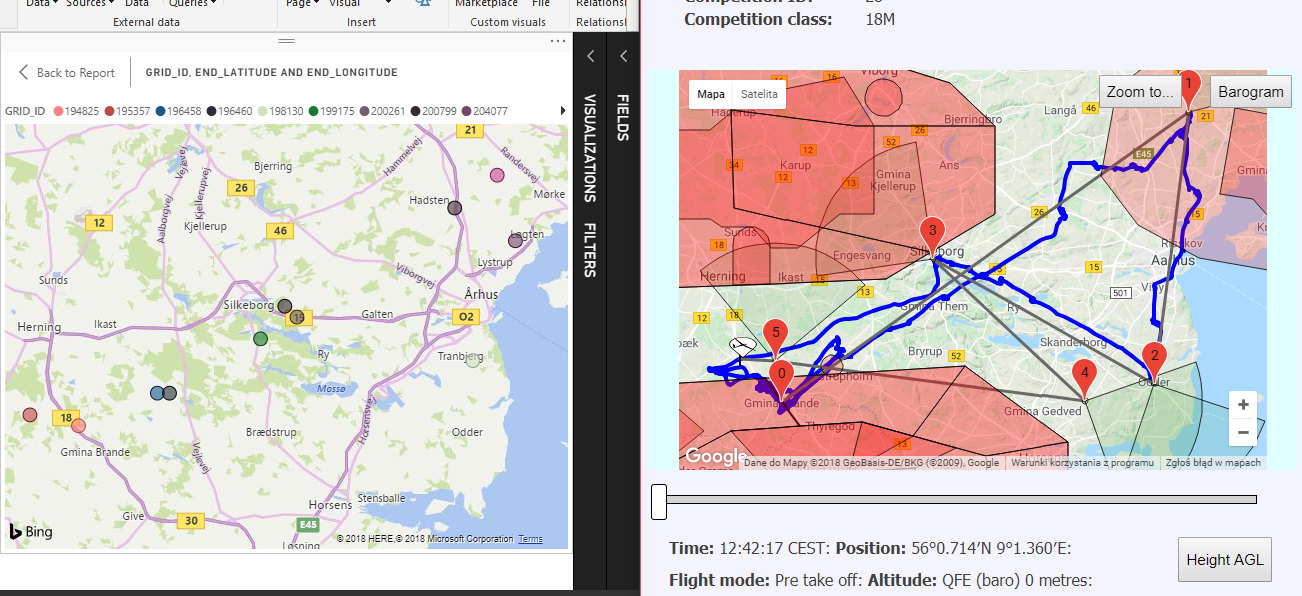
|  |  |
| --- | --- |
| Data select | Passed |
| Thermals locations | Passed |

# RESULTS AND DISCUSSION

**Results:**

The visualization tool allowed to show the results in a map. The map shows the location of the thermals that constantly appeared in a specific grid, the thermals are represented with a blue point in the map. The bigger the size of the point the more times the thermal occurred in that location. If all the thermals are selected, it was seen that the location of the thermals were in the central region of Jytland, in Denmark, which is the right location where the flights took place.

For individual flights, the thermals location was compared to the trace given by an online IGC viewer and the locations in the trace were in the same quadrants as the location displayed in the PowerBI view with the information taken from the dimensional model. The following image shows an example, where the thermals are the points with different colors, and the points match the same locations from the trace in the right:



**Discussion:**

The results only show possible thermals by collecting previous data. It is not warranted that the thermals will be in the same location for specific weather conditions.

The results were obtained using flight data from competitions during two days only. The more data registered about the flight locations and weather conditions, the better possibilities of finding a thermal the pilot will have. But still, the system cannot warrantee the thermals locations because there might be more factors that could not be consider in the elaboration of this project, for example, the different type of weather in locations that are near each other.

Nevertheless, the goal for the project was achieved, it was possible to store, calculate and display possible thermal locations.

# Conclusion

Throughout the project period weeks, the team worked hard to reach their final goal that was to solve the problem and implement the solution.

In the first phase of the project was given by the professors and after analyzing it was resolved that the problem was finding thermal locations. During the analysis phase it was determined that the weather and flight information was essential to get the possible locations, it was also necessary to calculate the locations where many thermals occur. Therefore, a data warehouse and ETL processes needed to be done.

In the design phase the tables that store the data from flight and weather were implemented, their data was obtained from java programs that allowed making the information readable and accessible. During this phase it was decided which dimension and facts were needed for the data warehouse. The data warehouse allowed the information to be tracked an analyzed over time.

The programs that read the log files and convert to the right columns in the database tables were made firstly during the implementation phase. Then a new java program was made to read the flight altitudes from the database and calculate the thermals locations based on the changes. In this phase the rest of the dimensions and facts in the data warehouse were populated. To match the flight data with the weather data, a view was made, where the information from the weather, flight and date tables from the data warehouse were joined. This made possible to see the weather conditions that happened during each flight. In the same view only, the thermals that constantly occur in specific locations were selected, those thermals where selected if they appeared more times than the average number of thermals in the location.

Power BI was used to present the data, this view has data slicers that allows the user to see specific thermals by date, weather conditions or specific flights. This tool also allows the user to see the results in the form of a map, where the thermals occur.

Finally, the results were tested and even if they were not fully precise, the solution for the problem was achieved. This means that it was possible to display the thermals that occurred constantly and their locations.

# ReferENCES

Anon., 2018. *Data Warehousing.* [Online]   
Available at: https://studienet.via.dk/Class/IT-DWH1Y-S18/Session%20Material/Forms/Default.aspx

Anon., 2018. *Glid. web.* [Online]   
Available at: http://glidingweb.org/igcWebview/

Anon., 2018. *Ogimet ( Weather data).* [Online]   
Available at: https://www.ogimet.com/metars.phtml.en

Anon., n.d. *Data warehousing etl tutorial..* [Online]   
Available at: http://learndatamodeling.com/blog/data-warehouse-etl-tutorial/

Anon, 2018. *Data warehousing tutorial.* [Online]   
Available at: http://etl-tools.info/data-warehousing\_tutorial.htm

Anon, 2018. *Semestar project 4.* [Online]   
Available at: https://studienet.via.dk/Class/IT-SEP4D-S18/Course%20Info/Forms/Default.aspx?RootFolder=%2FClass%2FIT-SEP4D-S18%2FCourse%20Info%2FProjectProposal&FolderCTID=0x0120006E1B111046F26F4FB00425FB741EDD49&View=%7BB017CEF7-FA98-42C1-8B7E-4E9078B36CD5%7D

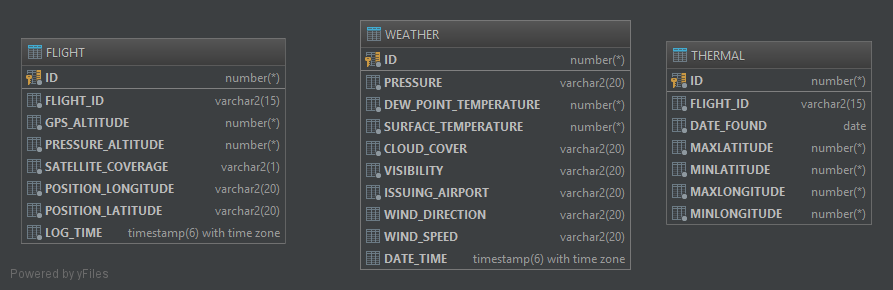
Kimball, R., 2008. *The Data Warehouse Lifecycle Toolkit.* s.l.:Wiley.

NISO, 2010. *Scientific and Technical Reports -,* Baltimore: National Information Standards Oganization.

VIA Engineering, in preparation. *Confidential Student Reports,* s.l.: s.n.

# Appendix

### Appendix A: Data base ER Diagram



### Appendix B: ER Diagram for Dimensional Model

