Université Libre de Bruxelles

INFO-H419 - Data Warehouses

Final Project

Brahim Amssafi, Soumoy Astrid

January 18, 2021

Table Content

1 Introduction

In this project, we will implement a comprehensive data warehouse application. The steps will be to first, extract the data from our source website. Then, from this data we will create a conceptual schema to structure the fields we have. From this conceptual schema, we will get a relational data warehouse by designing a schema and implement the tables in SQL. After that, we have to create our database and make it going threw an ETL process (designed in BPMN). This process will load the source data into the data warehouse. Finally, we will defined 20 queries to clearly understand the different aggregations and comparison between ancestors/descendants.

2 Database Schema

2.1 Extract Data

In order to create the conceptual schema, the first thing we need to do is to extract the information from the website[1]. When we are on the site, we can see multiple field names with their description. For some of them (*Month, UniqueCarrier*, ...) we can also have a look at their possible values (for month their is the list of all the month and their associated number value). We decided to start a schema without any links but just the dimension names and their attributes. The following picture show the data we extracted.

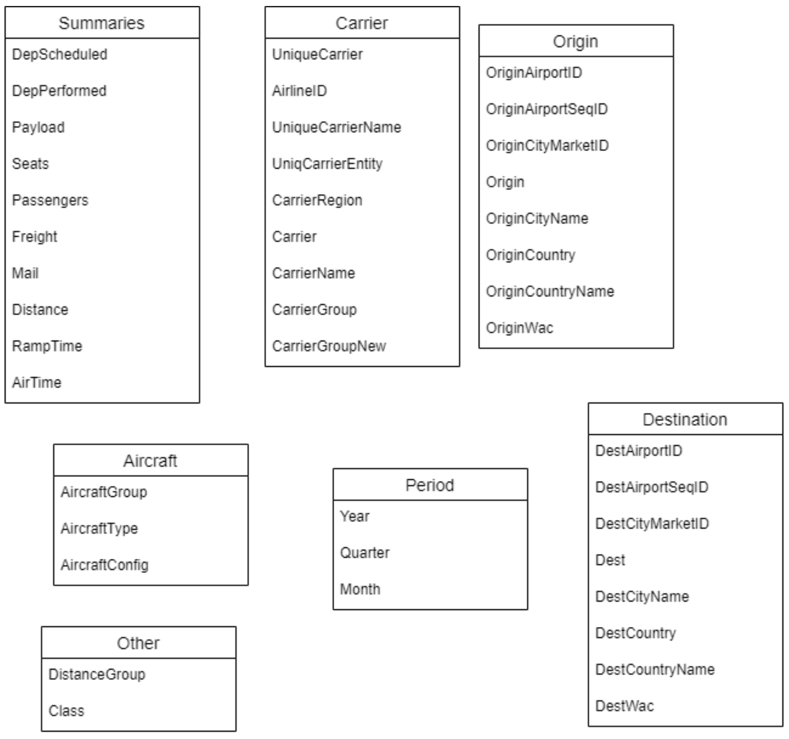

Figure 1: Data from website

[1]https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=261

2.2 MultiDim or DFM

The second step to create our conceptual schema is to choose either MultiDim or DFM conceptual model. They are both conceptual models, the difference is the way to present the data. We decided to design our schema with the DFM model because it has a simpler way to show relations and hierarchy. Moreover, it is the one we found easier and we were more comfortable using it. One of the main difference is that DFM use boxes for the fact table and circles for the other dimensions. Conversely, MultiDim use only boxes for every dimensions.

3 Conceptual Schema

3.1 Choices

To create our conceptual schema, we made multiple choices in order to defined the dimensions, their attributes and their relations.

- The Facts : the main fact we have is the flights information threw the months. So we made a box for the *Flight* fact dimension and we listed their attributes. We took all the attributes that we initially put in the Summaries dimension because it is the one that contains the elements of a flight.

- The dimensions :

  - The first field of the website we separe into a dimension was the Date. It was logical for us to have separated dimensions for *Year*, *Quarter* and *Month.* Since there is months in a quarter and quarters in a year, the dimensions will have to be related.

  - When we were looking at the parameters we had, we decided to create a dimension for *Airport.* We noticed that there were Origin Airport and Destination Airport, but since they are the same "object" we build a dimension and put two links (origin and dest) with the flight fact.

  - Another parameter that was in commun is the City, Country and WAC. As it is a distinct location object, it seems logical to have separated dimensions related between them like for the date.

  - Carrier was also an obvious dimension but in its parameters we decided to make three dimensions (CarrierGroup, CarrierGroupNew and CarrierRegion). The other one are attributes of this dimension.

  - For the MarketCityId, we thought at the beginning that it should be a link or an attribute of City. When we were looking closer at the information we had, we noticed that multiple cities can have the same marketCityID. So it can not be directly linked to a City and must be a entire dimension

3.2 Design

All those choices gives the following conceptual schema :

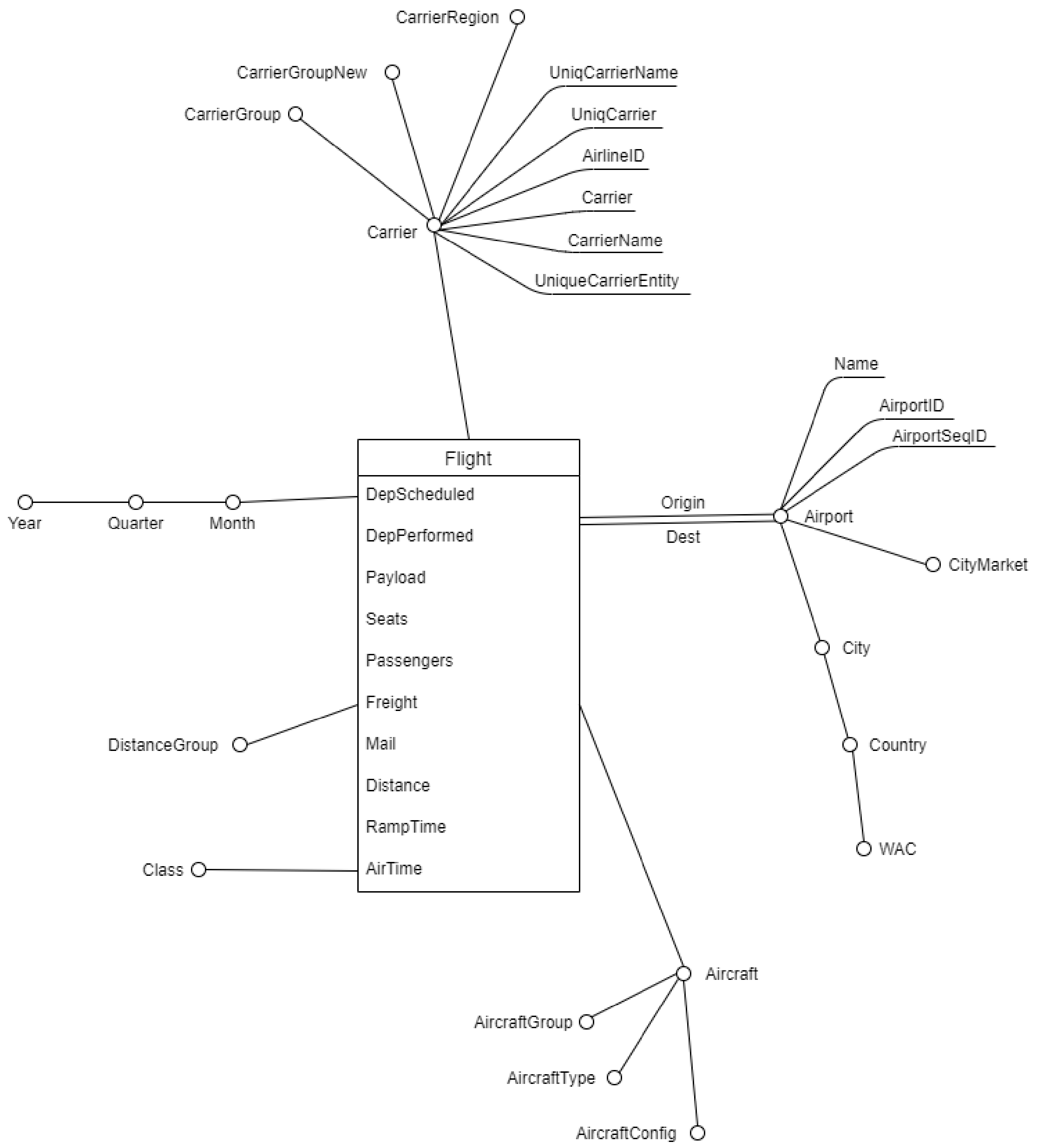

Figure 2: Conceptual Schema

4 Relational Data Warehouse

4.1 Choices

To create our relational data warehouse from the conceptual schema, we had to choice between two relational design :

1. Snowflake Schema
2. Starflake Schema : combination between Star and Snowflake schema
3. Star Schema

We decided to choose the star schema because the snowflake could be a little bit confusing with our data. The star schema is easier to do for what we have. Moreover, this schema doesn't have any consistency problems (it helps for the ETL process) and it doesn't use surrogate keys. We didn't choose the snowflake or the Starflake because we would have to normalized our data. For the time we had, it was faster to do the Star Schema. If there is duplication, it won't be a big problem for this project.

To design our relational schema we first had to defined the tables (dimensions and fact). We took the first linked dimensions of the Fact. Then we had to define the primary keys and foreign keys for each of them :

- dimDate : for this table, we didn't have any attribute that can be a primary key because neither a month, quarter or year is unique for each date. We decided to add a attribute named *DateID* that will be the primary key of this table.

- dimAircraft : the same as in date, we didn't have any attribute that could be a primary key, so we add one. The AircraftID is the primarykey for this table.

- dimAirport : we already had an attribute named AirportID that is unique for each airport. Therefore, it is the primary key for the dimAirport table.

- dimCarrier : the attribute *UniqueCarrier* is as we can guess by its named, unqiue for each Carrier. We decided to add the "ID" string at this attribute for a clarity purpose. *UniqueCarrierID* is the primary key for the dimCarrier Table.

- Flight : this table is defined from our Fact in the conceptual schema. We add the foreign keys that link it to the other tables (*UniqueCarrierID*, *DateId*, *AirportID* and *AircraftID*). All those foreign key are primary keys to the Flight table because they are unique for each flight.

We can see the star schema because all the tables are linked to the *Flight* table in the middle of the schema.

4.2 Schema

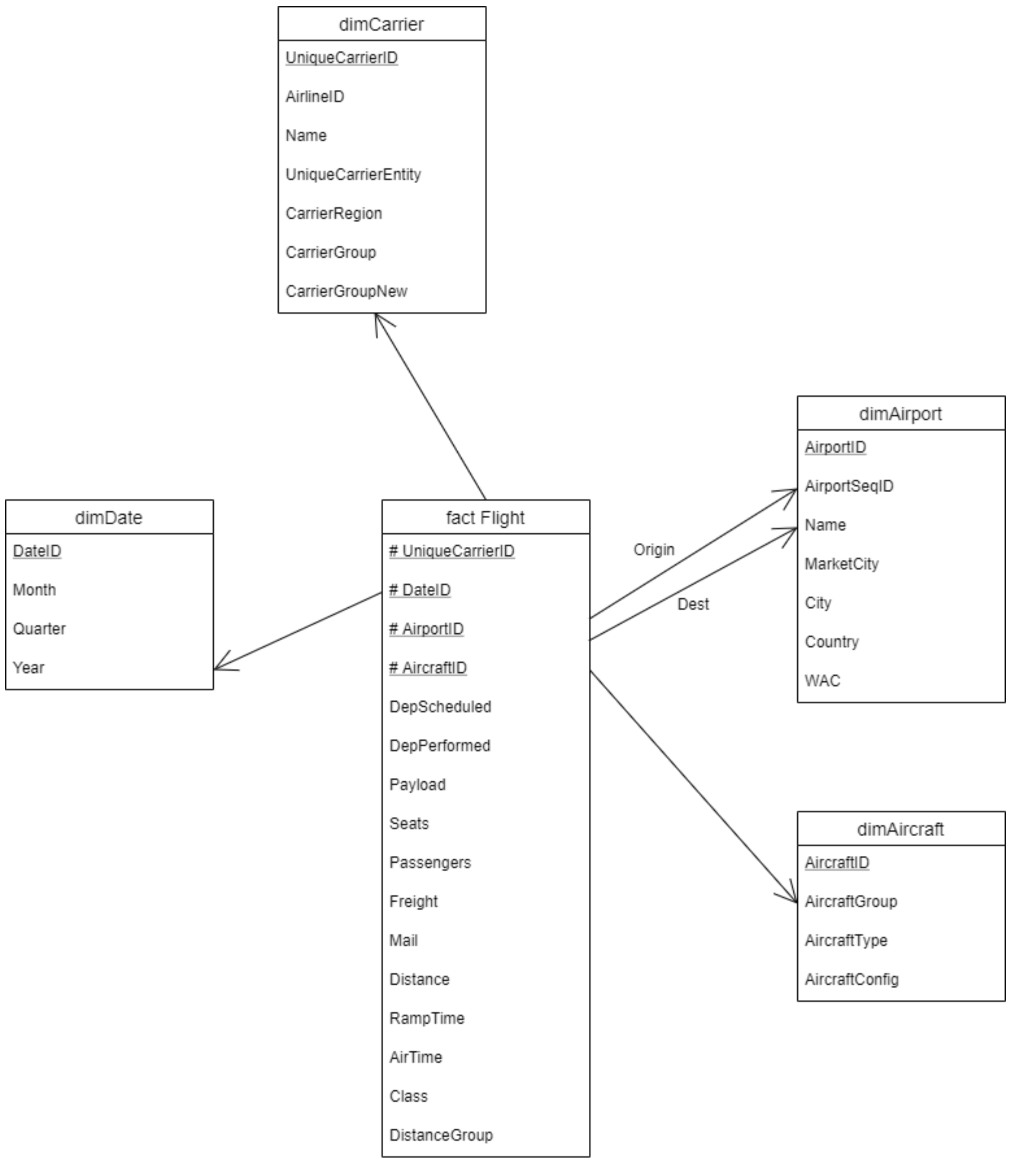

Figure 3: Relational Design

4.3 Tables

From this schema, we can create our tables with the corresponding SQL code (*see below*). This code is implemented in the file *create_table.sql* that will be run in order to implement the relational data warehouse.

```
CREATE TABLE dimDate ( DateID INTEGER NOT NULL IDENTITY (1,1) PRIMARY KEY,
                         Mounth numeric(2) Not NULL,
                         Quarter numeric(1) Not NULL,
                         Year numeric(4) Not NULL
);
```

```
CREATE TABLE dimCarrier ( UniqueCarrierID VARCHAR(50) NOT NULL PRIMARY KEY,
                         AirlineID INTEGER,
                         UniqueCarrierName varchar(100),
                         CarrierName varchar(100),
                         UniqueCarrierEntity VARCHAR(50),
                         UniqueCarrierRegion Varchar(1),
                         Carrier Varchar(100),
                         CarrierGroup INTEGER,
                         CarrierGroupNew INTEGER
);
```

```
CREATE TABLE dimAirCraft ( AirCraftID INTEGER NOT NULL IDENTITY (1,1) PRIMARY
    KEY,
                         AirCraftGroupt INTEGER Not NULL,
                         AirCraftType INTEGER Not NULL,
                         AirCraftConfig INTEGER Not NULL,
);
```

```
CREATE TABLE factFlight( DateID INTEGER FOREIGN KEY REFERENCES dimDate(DateID),
                     UniqueCarrierID VARCHAR(50) FOREIGN KEY REFERENCES
                         dimCarrier(UniqueCarrierID),
                     OriginAirportID INTEGER FOREIGN KEY REFERENCES dimAirport(
                         AirportID),
                     DestAirportID INTEGER FOREIGN KEY REFERENCES dimAirport(
                         AirportID),
                     DimAirCraft INTEGER FOREIGN KEY REFERENCES dimAirCraft(
                         AirCraftID),
                     DepScheduled REAL NOT NULL,
                     DepPerformed REAL NOT NULL,
                     Payload REAL ,
                     Seats REAL NOT NULL,
                     Passengers REAL NOT NULL,
                     Freight REAL NOT NULL,
                     Mail REAL NOT NULL,
                     Distance REAL NOT NULL,
                     RampTime REAL NOT NULL,
                     AirTime REAL NOT NULL,
                     DistanceGroup Varchar(50),
                     Class Varchar(50)
                     PRIMARY KEY (DateID,UniqueCarrierID,OriginAirportID,
                         DestAirportID,DimAirCraft)
);
```

```
CREATE TABLE dimAirport ( AirportID INTEGER NOT NULL PRIMARY KEY,
                          AirportSeqID INTEGER Not NULL,
                          AirportName varchar(100) Not NULL,
                          Marketcity INTEGER ,
                          City Varchar(100) Not NULL,
                          Country Varchar(100) Not NULL,
                          WCA Varchar(100) Not NULL,
);
```

4.4 Implementation

The implementation of the Relational data warehouse contains two main actions. Those actions can be done separately from each other since they are just a preparation for the ETL process.

STEP 1 :

- We created an empty database that we named *RITA* (referring to The Research and Innovative Technology Administration).
- We executed the file *create_table.sql* that is containing the SQL code we saw above. It will run the CREATE TABLE requests and create the structure of our data warehouse.

STEP 2 :

- We downloaded all the files from the website[2] for every month needed.
- We unzipped the files we just downloaded.
- We put all those files in a directory in order to give data to our database.

5 ETL process

5.1 Choices

We used SSIS for the ETL process, because it is the technology we previously used for the other project, we are familiar with it.

[2]https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=261

5.2 Design

To design our ETL process, we based our model on BPMN (Business Process Modeling Notation). It provides a conceptual and implementation-independent specification of processes. Also, the processes that are model with BPMN can be translated into executable specifications like in Microsoft's integration services for example.

We had some difficulties making the BPMN design for the ETL process. It is not a standard that we know very well and the documentation about it is confusing. We decided to design the two sub-process separately. As you can see on the top of the picture, the sub-process *dimTables* and the sub-process *FactTable* are both in a loop. We are executing them for every dimTables, then for every Fact Tables (here, there is just one). Below this ETL process overview, you can find in details the ETL process with the BPMN tasks.

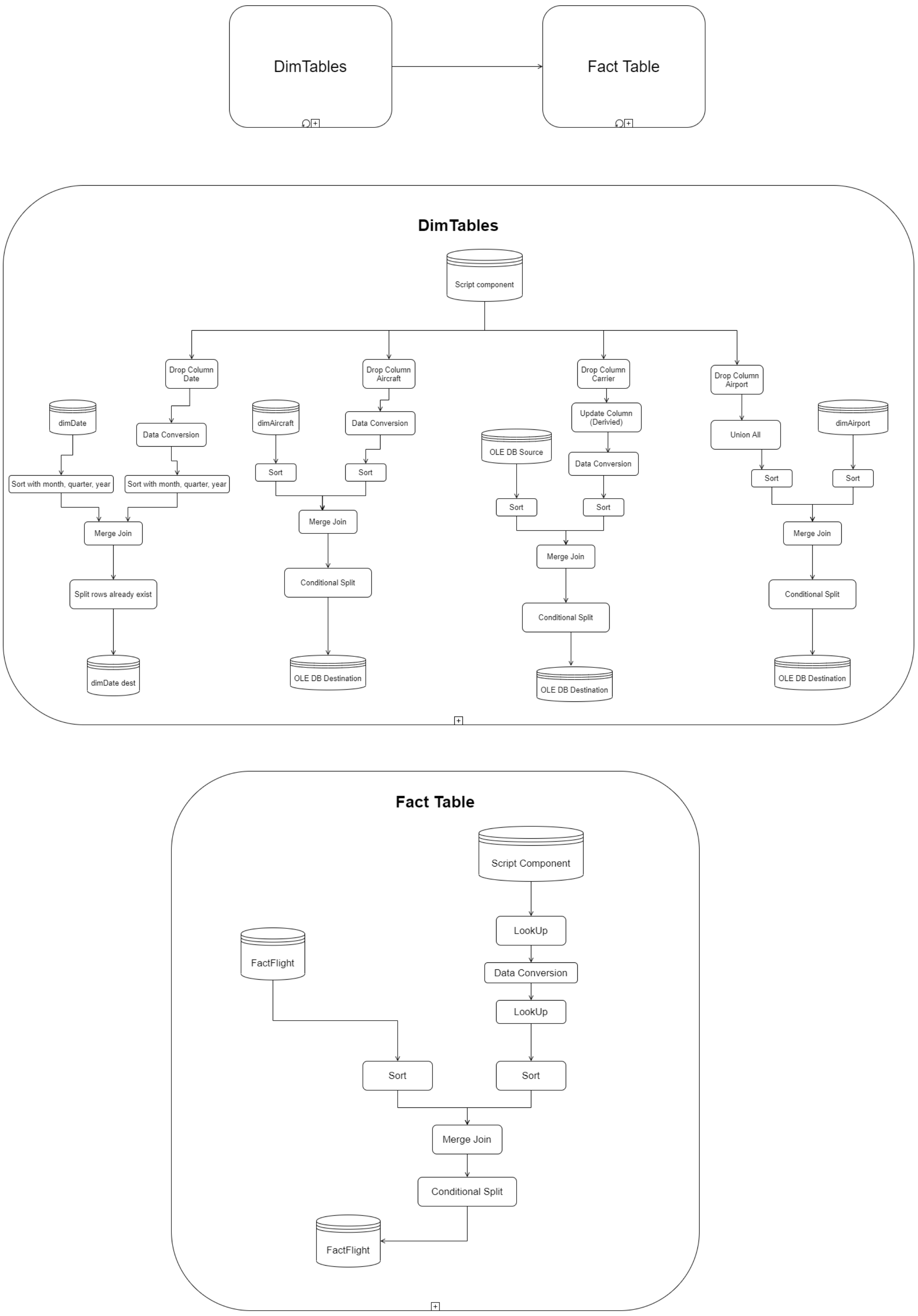

Figure 4: BPMN Design

5.3 Implementation

In order to implement the ETL process, we have to start by creating a foreach loop container. This container will make the sub-process defined inside the loop for every file. We will have two foreach loop container, one for our dimensions and one for our Fact. when creating a foreach loop container, we have to specify in the foreachloop editor the repertory and create a variable that will indicate the file. Once the container created, we will create a *Data Flow Task* for each of them. Here is what will contain our two tasks in our loops :

1. **dimTables** : The first step is a script component that will separate the data for our tables. For the Date informations, we will make a Data conversion and sort the result. This will be merge join with the sorted result of *dimData*. Finally we will split rows that already exists and send the final result to the dimData destination. The process will be approximatively the same for all our information (*Aircraft*, *Carrier*, *Airport*) except that for Carrier we will make a Derived Columns before the data conversion. The detail of the ETL process can be find on the pictures below.

2. **FactTable** : Here also we will need a script component but we will make two lookup, one before the data conversion and one after. The final data is sent to the FactFligth.

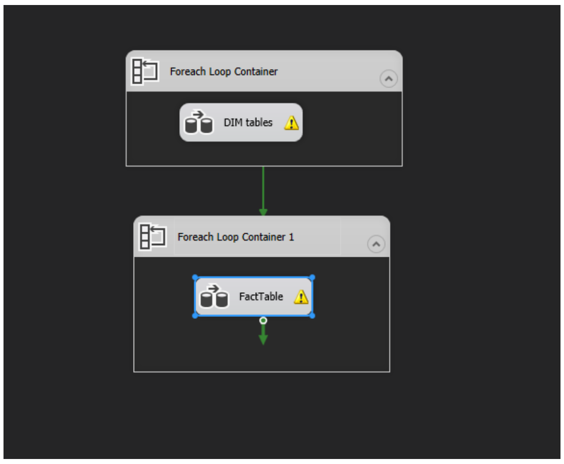

Figure 5: ETL process overview

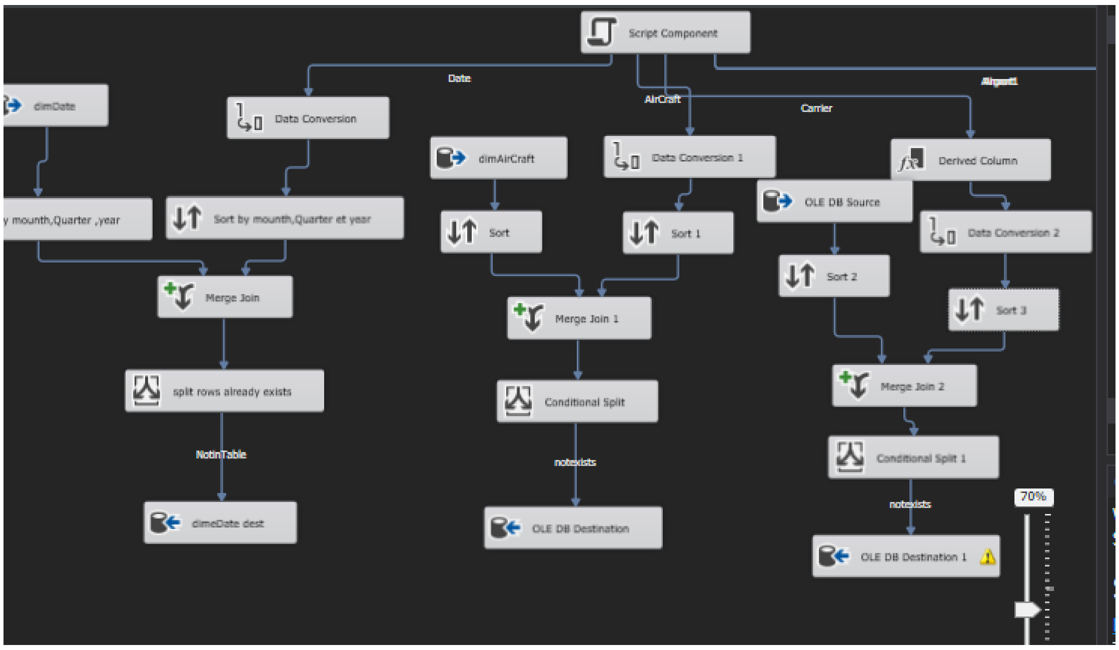

Figure 6: dimTables Data Flow task

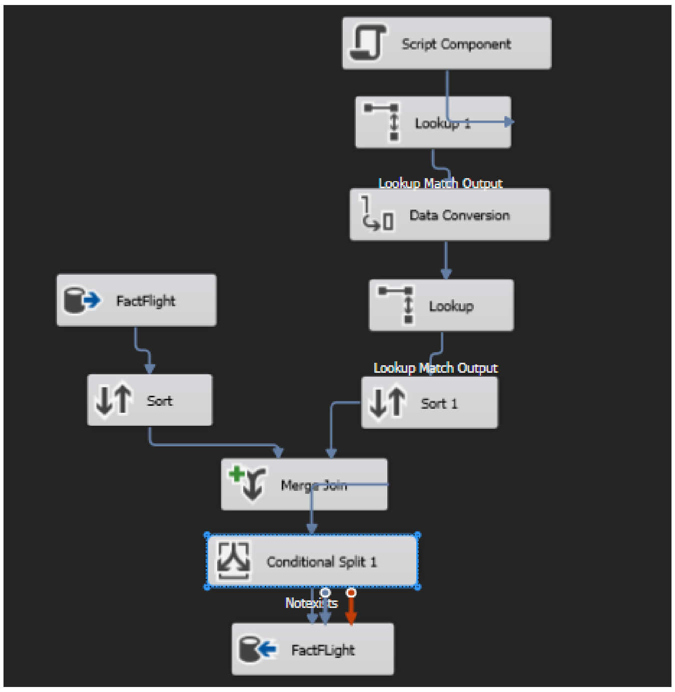

Figure 7: FactTable Data Flow task

6 Analytical Queries

6.1 Different classes

The 5 first queries are the ones given in the statement of this project.
We first have to identify some of the query classes to not implement just one of them. Here some of the classes we identified with a brief explanation and on of the query number that correspond in the section below.

- Select Query : it will select the data from tables and displays the data with the filter that you want. It is used in the query number 6 because it is a simple select of all AirportID.

- Totals or Summary Query : this query class is a subset of a select query but you can compute a sum, a count or an average for example. It is used for example with the query number 3, where we want the total number of seat sold. We can also see it in the query number 7, when we ask for the average of the distance for the flights.

- Parameter Query : it will ask for a specific parameter before displaying the table. For example in the query number 3, the parameter is the seats sold. We are comparing the seats sold year with "2019".

- Crosstab Query : this query is selecting data from multiple table, and for example compare them with a specific parameter. For example, the query number 1 is using table dimCarrier and dimDate.

The five first query make us see more the total/summary class because it is using mathematics like percentage, total etc. To variate our queries, we will make more classes that is using parameters from multiple tables to see their agregations.

6.2 20 Queries

Here is the description of our 20 queries and their SQL translation :

1. For each carrier and year, give the number of scheduled and performed Fights.

```
SELECT year,UniqueCarrierName,DepScheduled,DepPerformed
FROM factFlight fl,dimCarrier carr,dimDate dt
WHERE fl.DateID = dt.DateID AND fl.UniqueCarrierID = carr.
    UniqueCarrierID
GROUP BY dt.year,carr.UniqueCarrierName
```

2. For each airport, and country, give the number of scheduled and performed Fights in the last two years.

```
SELECT AirportName,County,DepScheduled,DepPerformed
FROM factFlight fl,dimAirport air,dimDate dt
WHERE fl.DateID=dt.DateID AND air.AirportID = fl.OrignAirport AND dt.
    year = YEAR(GETDATE())-2
GROUP BY dt.year,air.Country
```

3. For each carrier and distance group, give the total number of seats sold in 2019.

```
SELECT UniqueCarrierName ,distanceGroup ,count(seats)
FROM factFlight fl,dimCarrier carr,dimDate dt
WHERE fl.DateID = dt.DateID AND carr.uniqueCarrierID = fl.
    UniqueCarrierID
AND dt.year = 2019
GROUP BY UniqueCarrierName,distanceGroup
```

4. For cities operated by more than one airport, give the total number of arriving and departing passengers.

```
SELECT COUNT(F.passenger)
FROM dimAirport A, factFlight F
WHERE A.city in ( SELECT A.city
              FROM dimAirport A1, dimAirport A2
              WHERE A1.AirportID != A2.AirportID AND A1.city = A2.city
                  )
```

5. For each city and airport with more than 5,000 departures in 2019, show the average of the total number of departures by each carrier.

```
SELECT AVG(F.depScheduled)
FROM factFlight F, dimCarrier C, dimDate D
WHERE F.depScheduled > 5.000 AND D.Year = 2019 AND F.DateID = D.DateID
GROUP BY C.UniqueCarrierID
```

6. For each Airport, give the AirportID.

```
SELECT AirportID
FROM dimAirport
```

7. For each Flights, give the average of the distance.

```
SELECT AVG(distance)
AS distance
FROM Flight
```

8. For each country, give the average of the Airtime in January 2019.

```
SELECT AVG(F.Airtime)
FROM Airport A, Flight F, Date D
WHERE A.AirportID = F.AirportID AND D.DateID = F.DateID AND D.Month = 1
     AND D.Year = 2019
```

9. For each Airport, give the AircraftID group by AircraftType

```
SELECT AC.AirecraftID
FROM Airport A, Aircraft AC, Flight F
WHERE F.AirportID = A.AirportID AND F.AircraftID = AC.AircraftID
GROUP BY AC.AircraftType
```

10. For each Flight with more than 200 passengers, give the carrier name .

```
SELECT UniqueCarrierName FROM FactFlight fl,dimCarrier car WHERE fl.
    UniqueCarrierID=car.UniqueCarrierID and car.passengers>20
```

11. For each Flight departure schedule between January 2020 and June 2020, give the total number of different Airports.

```
SELECT COUNT(F.AirportID)
FROM factFlight F, dimDate D
WHERE D.DateID = F.DateID AND D.Month > 1 AND D.Month < 6 AND D.Year =
    2020
GROUP BY F.AirportID
```

12. For each AircraftType, give the total number of Carrier.

```
SELECT AirCraftType,count(UniqueCarrierName)
FROM dimairCraft acr, factFlight fl, dimCarrier carr
WHERE mc.UniqueCarrierID = carr.UniiqueCarrierID AND fl.airCraftID =
    acr.AircraftID
GROUP BY acr.AirCraftConfig
```

13. Give the total number of passenger in the first quarter of 2019.

```
SELECT sum(passengers)
FROM FactFlight fl, dimDate dt
WHERE dt.dateID = fl.DateID AND dt.year = 2019 AND dt.quarter = 1
```

14. Give the number of seats and the name of the carrier for the Flight with the less passenger of 2019.

```
SELECT seats, UniqueCarrierName
FROM FactFlight fl, dimDate dt
WHERE fl.DateID=dt.DatId
AND dt.year=2019
AND fl.passengers=(SELECT min(passengers) FROM FactFlight)
```

15. For each country, give the average of Airtime.

```
SELECT AVG(Fl.DepScheduled)
FROM dimAirport A, FactFlight Fl
WHERE Fl.AirportID = A.AirportID
GROUP BY A.Country
```

16. For each City, give the total number of departure flight.

```
SELECT COUNT(Fl.DepScheduled)
FROM dimAirport A, FactFlight Fl
WHERE Fl.AirportID = A.AirportID
GROUP BY A.City
```

17. For each AircraftType, give the flight with the more passengers.

```
SELECT AirCraftType
FROM FactFlight fl,dimAircraft arc
WHERE fl.AircraftID=arc.aircraftID
and fl.passenger=(SELECT max(passengers) from FactFlight)
Group by arc.aircraftType
```

18. For each Carrier, give the airport name all the flights of 2019.

```
SELECT A.name
FROM dimAirport A, factFlight F, dimCarrier C, dimDate D
WHERE A.AirportID = F.AirportID AND C.UniqueCarrierID = F.
    UniqueCarrierID AND D.DateID = F.DateID AND D.year = 2019
```

19. For each Quarter of 2019, give the total number of country visited.

```
SELECT COUNT(A.country)
FROM dimAirport A, factFlight F, dimDate D
WHERE D.DateID = F.DateID AND A.AirportID = F.AirportID AND D.Year =
    2019
GROUP BY D.quarter
```

20. For each Flight, give the total number of passenger of a specific AircraftConfig.

```
    SELECT aircraftConfig, SUM(passenger)
    FROM FactFlight fl, airCraft aircf
    WHERE fl.aircraftID = aircf.AircraftID
    GROUP BY aircfAircraftConfig
```

6.3 Implementation

Now that we have our 20 queries translated in SQL, we can simply implement them with this code into our data warehouse.

7 Conclusion

We have seen how to go from standard information taken from a public website to a relational data warehouse. This project summarize very well the objective of this class. All the steps help us to understand the link between all the notions we have seen. The harder point of this project, for us, was to regroup the information from the website and design the conceptual schema. Once this first step done, the other parts were mostly based on it.