AMADEUS

APE Web Scraping Tool

Report

Internship

8/29/2013

Lucas WEISSERT

Summary

The purpose of this internship is to scrape the Kayak Website as to extract information, simulate actions, log and analyze results. Web scraping was something completely new for me; in fact it was the first time that I will use Python. During this internship I was in charged to parse the Kayak Website to compare Amadeus versus ITA which the main competitor. On the website, when you are looking for a flight, you have the choice to book the ticket on different website. Most of the time, when you can book on an airline's website, the ticket has been proposed by Amadeus or ITA. The purpose of the first scrip was to compare the price proposed by Amadeus and by ITA to know which was proposing the cheapest one. The second script was to check if the flight proposed by Amadeus was the same when we wanted to book the ticket. If the two flights were different, it was named a Bad Click.

To do these two scripts I used Python, I stored results in a MySQL Database then I exploit results using R which is a mathematic language such as matlab.

Table of contents

1. ACKNMOWLEDGEMENTS

I would like to thank:

- All of my team, they gave me a nice welcome and they helped me a lot to a great internship.
- Renaud Arnoux-Prost who helped me during my internship’s search
- Antoine Menard who accepted to give an interesting project
- Ludovic Dufrenoy who was my coach during this internship
- All people that I met for the welcoming

2. INTRODUCTION

I have been working as an intern for three months at AMADEUS IT North America in Boston (1A), leader on transaction processor for the global travel and tourism industry. My team was in charge of the development part in Boston. This permitted me to improve my knowledge on some development tools and to have a better overview of team working.

Firstly I will do a presentation of the company through an "identity card", then, summarizing the company history, I will describe the main events which contributed to make the company the leader on his market.

It is also important to describe the company activities, to explain what it proposes exactly in term of products and services, and in which market it is present today

Then I will talk about the company organization as to show the different job positions and describe the organization at Amadeus especially the one in my team, then I will address the human resources management and to finish this part I will do a brief financial analysis.

To finish my report, I will explain my mission by showing some examples and finally I will finish with a critical analysis by giving my point of view about the company.

3. COMPANY’S PRESENTATION

4. INTERNSHIP PRESENTATION

4.1. INTRODUCTION

4.1.1. Overview

This document describes the project “APE Web Scraping Tool”. The goal of this internship was to use Web scraping tools to extract data from different websites. During this internship, two programs have been developed and their objectives were:

- To compare Amadeus and ITA to identify which one is proposing the best price .
- To check if the flight proposed on the Kayak website is the same when we redirect to the Airline website; if it is not the case, we call it ‘Bad Click’.

Those two scripts have been developed to extract data from the Kayak website. For Bad Click verifications, United and HawaianAir websites have been parsed.

The application is a console-based application; we can launch several scripts in parallel as there is no contention between scripts and each script can process several OnD/Dates. The development is modular: the MySQL and R sections are independent from Kayak; it is also easy to add additional Parsers for BadClicks as we have used dynamic functions that are supported by Python..

4.1.2. Scope

This document describes the software architecture of the project , the languages and the tools used. It describes also in detail the sofware, the database and the different graphics built using R.

4.1.3. Definitions

OnD: Origin and Destination.
Bad Click: A bad click is considered when the flight proposed on the website (Kayak as example) is not the same as the one proposed on the Airline booking site (for example United’s website). The price could be different but the number of stops, airports or departure and arrival times as well.
Provider: A provider is the booking site where you can book the ticket for a flight (United, Expedia, Opodo)
Solution: A solution is an itinerary for an OnD/Date proposed on Kayak

4.2. TOOLS

4.2.1 Languages Used

For this project, we used several languages and the main part of the project has been realized using Python and Selenium. **Python** is an interpreted language which is easy for coding and testing but which is slower than Java , C or C++. It could be interesting to compare the speed of execution of those languages however much of the time is spent on the network and inside the MySQL or Selenium libraries. **Selenium** is a very thorough library which can be used with different languages such as Python and Java to interact with Web sites. It is extremely convenient for parsing websites as it can deal with all web pages features: Buttons, Xpaths but more importantly JavaScript. I had first evaluated BeautifulSoup and Mechanizer but I switched to Selenium as invoking JavaScript is a Must for this project. BeautifulSoup and Mechanizer do not support this feature.
Results are stored in a **MySQL Database** instance which contains three main tables and several views. This provides full independence from the Parsing and allows to keep all results.
Interpretation of those results is done using **R** which is a statistical language that is very convenient to read Excel files, SQL tables and that provides powerful statistic analysis and plotting modules.

4.2.1.1 Tool Versions and overview

For the project, the following versions have been used for the different tools .

Firefox: Version 22.0
Python : Version 2.7.3
Selenium : Version 2.35.0
MySQL : Version 14.14 Distrib 5.5.22
R : Version 2.14.1
The software has been developped using Linux Ubuntu 12.

4.2.1.2 Selenium

Selenium is a portable software for web applications. It is very convenient to parse a website which contains JavaScript and is the only tool that allows to do that. However, we have to be very careful about the version that is used as there are compatibility issues between Firefox versions and Selenium versions. Any upgrade of one of those software implies a retest of the application as sometimes bugs introduced by upgrades are fixed very slowly by vendors. For example, the Selenium mouse function (move_to_element) was not working with the last versions of Firefox and Selenium.

4.2.1.3 How to use Selenium with Python

There are some **Key imports** that are necessary to use Selenium:

To use the webdriver

from selenium import webdriver

To use WebDriverWait

from selenium.webdriver.support.wait import WebDriverWait

To create interactions with the mouse

from selenium.webdriver import ActionChains

The elementary steps to parse the website with Selenium are the followings:

1. Create a browser (Here with Firefox)

```
browser = webdriver.Firefox()
```

2. Load a page in the browser

```
browser.get(URL)
```

Here URL is a string, for example browser.get("www.google.fr")

3. Find an element inside the source code

There are different ways to find a single or several elements; you can retrieve elements using class names, element IDs , CSS selectors or XPath expressions . You have to choose between those different ways, often one method is easier than another one. I found that CSS and XPath are very convenient to locate elements .

Xpath is a language used for locating nodes in an HTML document. It allows to find elements by relative position : you can target a nearby element and locate your element based on Keywords or Ids. Any element can be retrieved with minimum coding .

I have used primarily the following Selenium methods for my project:

Find_elements_by_xpath

Find_elements_by_class_name

Below is a small program which will extract the title of the website using Selenium:

```
browser = webdriver.Firefox()
URL = "https://pypi.python.org/pypi/selenium"
browser.get(URL)
title = browser.find_element_by_class_name("section")
print title.text
```

We could have alternatively used Xpath to find the “section” element with :

```
            title =browser.find_element_by_xpath(“.//div[@class='section'])
```

The difficulty with Parsing is that you need first to understand the HTML display and to identify the patterns, links, classnames that you need to retrieve. For this , you can use Firebug or tools provided in the Firefox menu. I also had to save the HTML display to a file for analysis. In particular, patterns may change according to the context and this takes time to find out; I noticed different patterns for single versus round-trip itinerary.
It is also very important to keep traces of executions as the vendor may evolve his Website. In this case, we would need to be very reactive and to have the right traces and error logs is important.

Waiting for an element to be loaded (WebDriverWait)

This feature is very important as HTML pages are changing while they are loaded. WebDriverWait allows to wait for an element to be present or visible on the page. This means that a thorough examination of the HTML source is necessary to understand which are the elements we have to wait for. Those elements may also vary with the functional content of the request and may also change as the Website is evolved by the vendor.
This has been taken unfortunately a lot of my time at the start of the internship. An example of usage of WebDriverWait is as follows:

```
  WebDriverWait(browser, 150).until(lambda br:
br.find_element_by_id("fs_airlines_content").is_displayed())
```

In this example we are waiting until the element which contains the ID 'fs_airlines_content' is displayed. We are also asking the system to time-out if the element is not displayed within 150 seconds.

4.3. SYSTEM DESCRIPTION

4.3.1 Command line

This part describes the Command Lines that are used to launch the different scripts.

To execute our Python script we have to enter a command having the following format:

```
lucas@Ubuntu:~$ python MainKayak.py OnDFile.txt  Parse.txt AMADEUS
```

python is to the keyword required to use python.
MainKayak.py is the main script.
OnDFile.txt is the file which contains the list of companies, and all OnD / Dates to process.
Parse.txt is the file which contains the Parsers for Bad Clicks.
AMADEUS is the system that we want to force on the Kayak URL.
The acceptable values are: AMADEUS, ITA or All if we want no filter.

Here are the commands to execute our R-script

```
lucas@Ubuntu:~$ R
R > source('ScriptR.R')
R > q=ScriptR('generalXX.csv', 'percentXX.csv')
```

source is to load our script
q= ScriptR('generalXX.csv', 'percentXX.csv') is to call our functions wrote in the script, this function will generate all different plots. Arguments are the two csv that we created.

Here are the commands to execute our SQL-script

```
lucas@Ubuntu:~$ chmod 777 ScriptSQL.sql
lucas@Ubuntu:~$ ./ScriptSQL.sql
```

chmod 777 ScriptSQL.sql we give the permission to launch the script
./ScriptSQL.sql we execute the script SQL

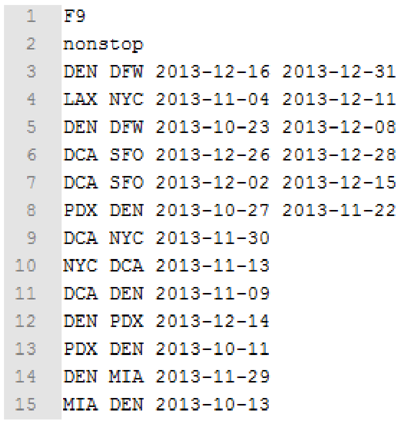
4.3.2 Input File

4.3.2.1 Description

There are two input files to the script.
The first file (OD file) allows to input the list of companies we want to process, to indicate if we want to limit to nonstop flights and also to input the list of OnD/Dates that we wish to analyze. The script will connect to the Kayak website , and will do the search for every OnD/Date and company present in the input file.
The second file (Parser file) allows to input Airlines for which we want to examine bad Clicks as well as the name of the Parser to execute. Parsing for United and Hawaiianair have been developed but new parsers can be introduced without changing the core of the application. This has been made feasible using dynamic invocation of functions that are supported by Python (as does C or Java).
We could extend this to the parsing itself but the internship has been short and I have had not enough time to do this.

4.3.2.2 Format and Example of input files

Here is an example for the first file (OD file):



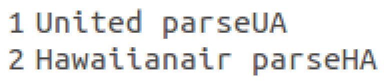
```
F9
nonstop
DEN DFW 2013-12-16 2013-12-31
LAX NYC 2013-11-04 2013-12-11
DEN DFW 2013-10-23 2013-12-08
DCA SFO 2013-12-26 2013-12-28
DCA SFO 2013-12-02 2013-12-15
PDX DEN 2013-10-27 2013-11-22
DCA NYC 2013-11-30
NYC DCA 2013-11-13
DCA DEN 2013-11-09
DEN PDX 2013-12-14
PDX DEN 2013-10-11
DEN MIA 2013-11-29
MIA DEN 2013-10-13
```

We can observe on this display three types of lines.

- The **first line** contains the **list of companies** to select; you can input more than one company separated by a ‘space’ or you can search for all companies by placing a star **‘*’** at the start of the line.
- The **second line** allows to select if we wish to search for direct flights only or for 1-stop and 2-stops. For a direct flights search, we input **nonstop** otherwise we place a star **‘*’** to request any number of stops.
- **The following lines** will contain the different **OnD/Dates** that we wish to analyze: we

need to input the IATA-Code for both arrival and departure airports as well as dates of travel. If there is only one date, it is considered to be a **one-way** otherwise it is a **round-trip**.

Here is an example of the second file (Parser file) :



```
United parseUA
Hawaiianair parseHA
```

In this second file, the first field is the name of the airline, the second one (parseXX) is the name of the parser to call to check for BadClicks.

4.4. PARSING

4.4.1 Overview

The purpose of this internship is to scrape the Kayak Website as to extract information, simulate actions ,log and analyze results. As explained before, I chose Selenium for webscraping as it allows to easily extract information but also to simulate all actions that a user can do. We have tried to have large part of the code independent from the Kayak Website as to allow development of webscraping of additional sites such as Expedia. However, due to lack of time, another site could not be implemented. I did a lot of sub-functions to have a code as generic as possible and also the parts using MySQL and R are fully independent from Kayak.

From Selenium, I used mainly the function **find_elements_by_Xpath** which allows to have cleaner code and also which would allow to incorporate an additional site more easily. During this internship, two main scripts were develop to do web scraping: one extracts information from the Kayak website and the other extracts information from an airline website to check Bad clicks.

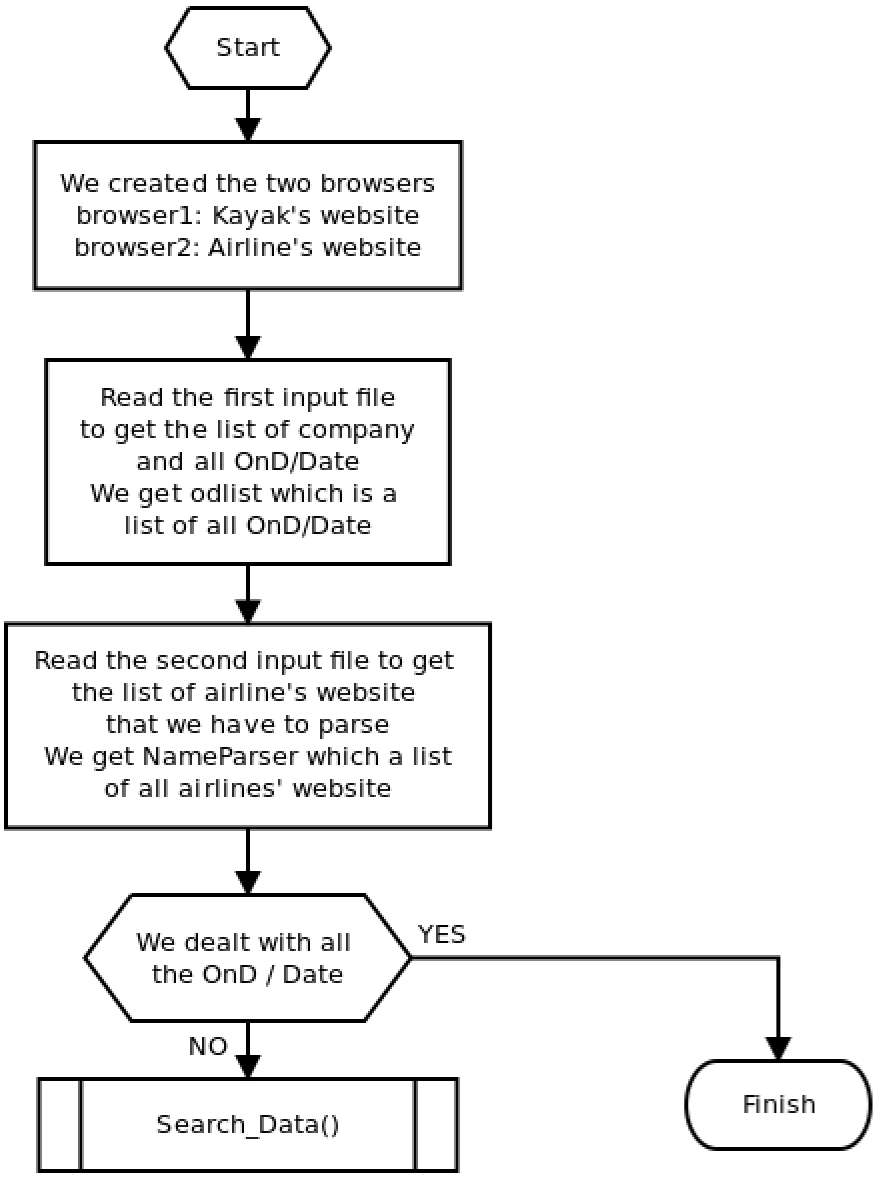
4.4.2 Kayak website parsing

4.4.2.1 Description

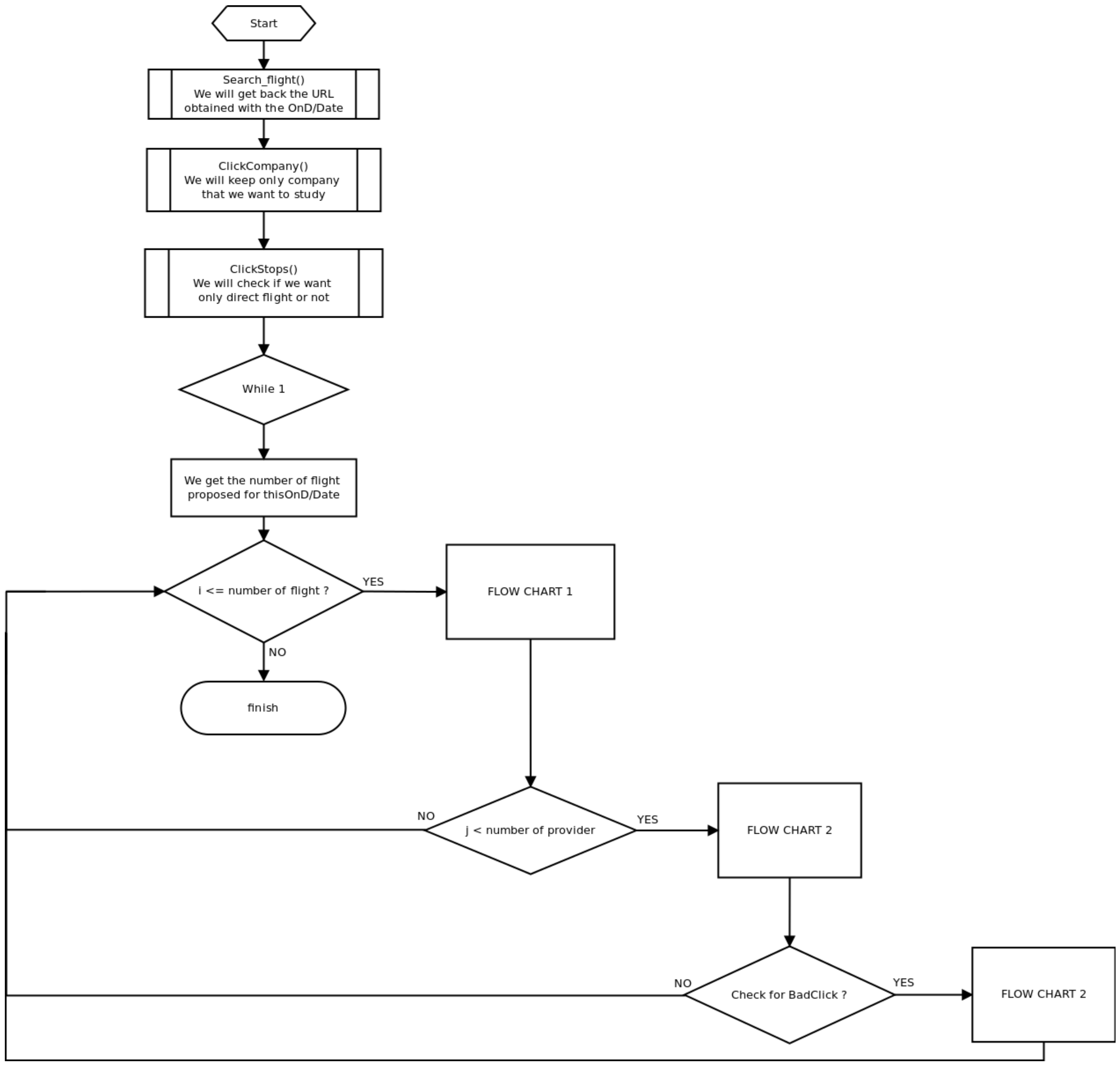
The purpose of our first script is to parse the Kayak website and to compare for each flight the price proposed by Amadeus and the price proposed by ITA. First we have to force the Kayak URL to get flights proposed only by Amadeus or only by ITA.
In this script, we retrieve all important information related to a flight such as departure time, arrival time, origin and destination, dates and prices. To compare prices, we need to check all providers which propose this flight and to check if the system accessed is Amadeus, ITA or another system such as Opodo or Expedia. To identify precisely on which system the booking will be made, we need to also look for the specific URL towards which the customer is redirected for the booking (United Website for example). On Kayak this URL contains as a keyword AMADEUS, ITA, OPODO or other systems. Once the parsing is done, all information is stored in a MySQL database for each flight and for each provider.

4.4.2.2 Logical flow chart

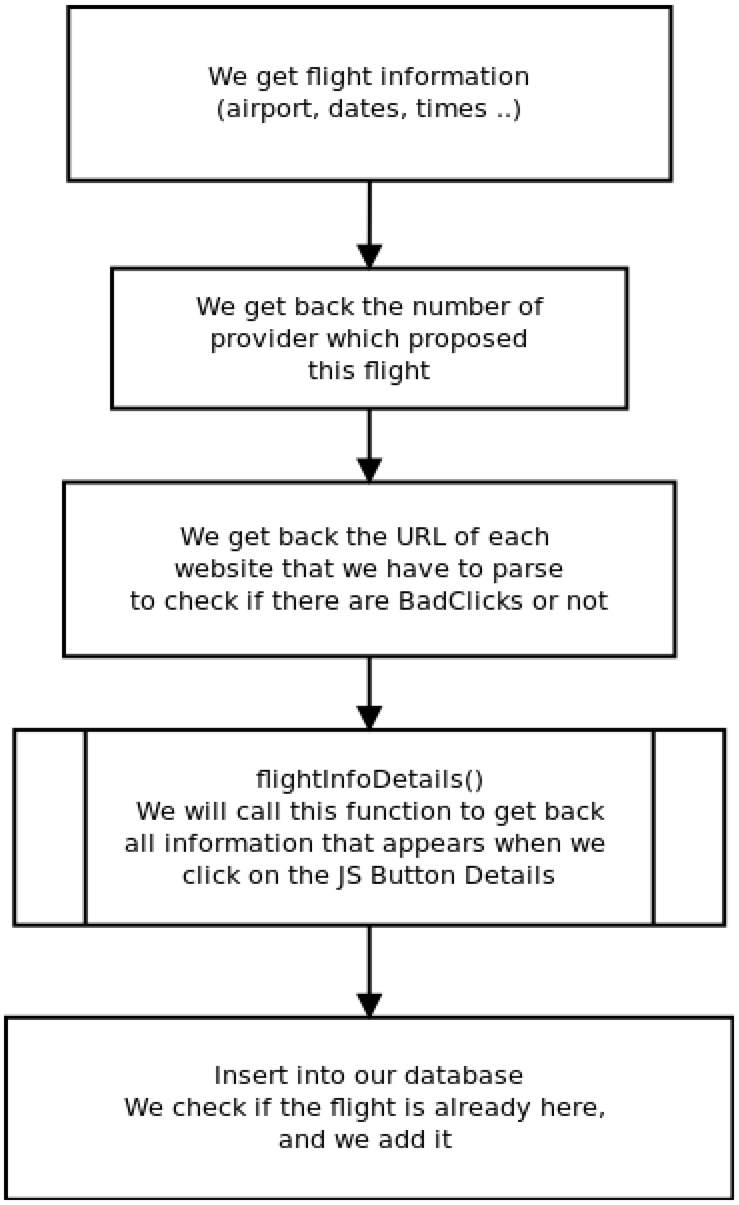
Hereafter is the logical description of the flow for the main script:


Start
We created the two browsers
browser1: Kayak's website
browser2: Airline's website
Read the first input file
to get the list of company
and all OnD/Date
We get odlist which is a
list of all OnD/Date
Read the second input file to get
the list of airline's website
that we have to parse
We get NameParser which a list
of all airlines' website
We dealt with all
the OnD / Date
YES
NO
Search_Data()
Finish

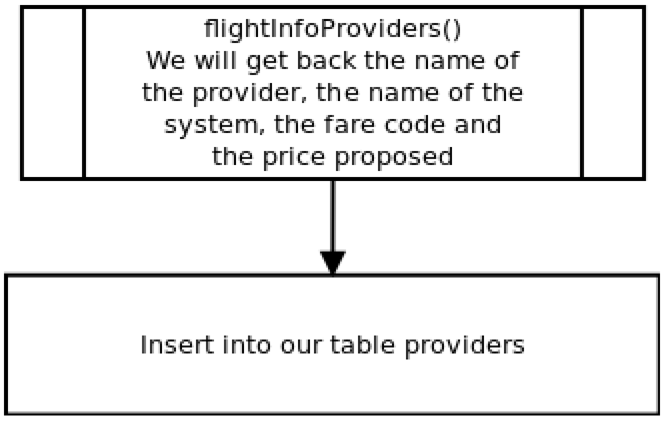
This flow chart is an overview about what we are doing in the function “Search_data” which is coded into the script FunctionKayak.py


Start
Search_flight()
We will get back the URL obtained with the OnD/Date
ClickCompany()
We will keep only company that we want to study
ClickStops()
We will check if we want only direct flight or not
While 1
We get the number of flight proposed for thisOnD/Date
i <= number of flight ?
YES
FLOW CHART 1
NO
finish
NO
j < number of provider
YES
FLOW CHART 2
NO
Check for BadClick ?
YES
FLOW CHART 2

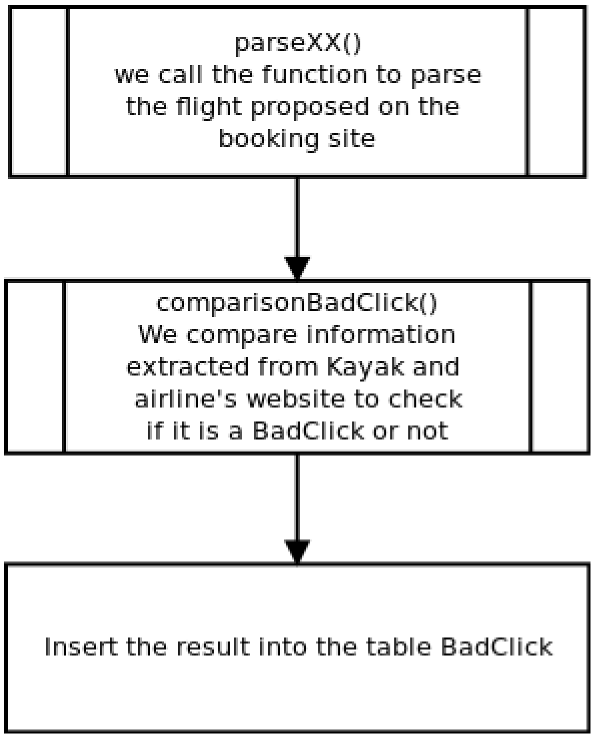
Here it is the flow chart related to the box "FLOW CHART 1"


We get flight information
(airport, dates, times ..)
We get back the number of provider which proposed this flight
We get back the URL of each website that we have to parse to check if there are BadClicks or not
flightInfoDetails()
We will call this function to get back all information that appears when we click on the JS Button Details
Insert into our database
We check if the flight is already here, and we add it

Here it is the flow chart related to the box “FLOW CHART 2”


flightInfoProviders()
We will get back the name of the provider, the name of the system, the fare code and the price proposed
Insert into our table providers

Here it is the flow chart related to the box “FLOW CHART 3”


parseXX()
we call the function to parse the flight proposed on the booking site
comparisonBadClick()
We compare information extracted from Kayak and airline's website to check if it is a BadClick or not
Insert the result into the table BadClick

4.4.2.3 Parsing main steps

There are 6 main sections for the parsing:

- Build the URL for the OnD/Date
- Select the list of companies we want to study.
- Indicate if we want non-stop or all flights
- Find flight information
- Find fare information
- Find URL of the site on which the booking is redirected

Each of those sections is further developed below and for each section , I have attached an extract of the corresponding HTML source.

Build URL for the OnD/Date :

This is achieved in the routine Search_OnD_Url which is less than 15 lines of code. There is no parsing as such but it requires to understand how the website can be accessed directly for an OnD/Date without filling fields and simulating clicks. This routine needs to be re-implemented if we want to access another site than Kayak

<u>Selection of the list of companies to process (see appendix 1):</u>

This is achieved by the routine ClickCompany which is less than 30 lines of code . For this we need to parse the HTML source and we have 4 actions:

- Find the flight section using:

```
find_element_by_css_selector("div[class='filterSectionOptions'][id='fs_airl
ines_content']")
```

- Retrieve information for all airlines using:

```
find_elements_by_xpath(".//input[@name='airlines']")
```

- Find the company to select and verify if it is checked using:

```
find_elements_by_xpath(".//input[@name='airlines']")
get_attribute('id') to find the airline name
```

`get_attribute('checked')` to verify if it is checked

- Finally we click on the element to select the company

APPENDIX 1: Html part used for Airline selection

This section shows an extract for 2 airlines

```
<div id="fs_airlines_content" class="filterSectionOptions">
    <input type="checkbox" checked="checked" value="AS" name="airlines"
id="fs_airlines_AS_input" class="r9-checkbox-input newCheckbox" />
    <input type="checkbox" checked="checked" value="AA" name="airlines"
id="fs_airlines_AA_input" class="r9-checkbox-input newCheckbox" />
```

<u>Non Stop selection (see appendix 2):</u>

This is achieved by the routine ClickStops which is less than 30 lines of code. We have 3 actions

- Find the filter section using:

```
find_element_by_css_selector("div[class='filterSectionOptions'][id='fs_stop
s__content']")
```

- Find the nonstop or one stop section using :

  `get_attribute('id')` and `get_attribute('checked')`

The id is named "**fs_stops_0_input**" for nonstop, "**fs_stops_1_input**" for 1-stop.

- We click on the element to select it or not

APPENDIX 2: Html part used for non stop selection

This section shows Html for nonstop,1-stop and 2-stop selection

```
<input type="checkbox" checked="checked" value="0" name="stops"
id="fs_stops_0_input" class="r9-checkbox-input newCheckbox" />
<input type="checkbox" checked="checked" value="1" name="stops"
id="fs_stops_1_input" class="r9-checkbox-input newCheckbox" />
<input type="checkbox" checked="checked" value="2" name="stops"
id="fs_stops_2_input" class="r9-checkbox-input newCheckbox" />
```

<u>Flight information (see appendix 3):</u>

This is achieved by the routine Search_data which is less than 200 lines of code. This the core of the parsing.

The parsing of flight information is achieved in 6 steps:

- Find the flight information section using:

```
find_element_by_class_name(“tripdetailholder”)
```

- Find the different airports:

```
find_elements_by_xpath(“.//div[@class= ‘airport’]”)
```

- Find the duration of the flight:

```
find_element_by_xpath(“.//div[@class= ‘duration’]”)
```

- Find the departure times:

```
find_elements_by_xpath(“.//div[@class= ‘flighttime flightTimeDeparture’]”)
```

- Find the arrival times:

```
find_element_by_xpath(“.//div[@class= ‘flighttime flightTimeArrival’]”)
```

- Find the number of stops

```
find_elements_by_xpath(“.//div[@class= ‘stopsLayovers’]”)
```

APPENDIX 3: Html part used for flight information

This section shows the information for one flight

```
<div data-index="1050" data-resultid="32a330c7116eb1b3dce8323cc5e888ea"
class="flightresult resultrow     clicked lastclicked" id="tbd1050">
  ................
      <div class="tripdetailholder">

         <div class="airport" title="Miami, FL- Miami">
          MIA
         </div>
         <div class="flighttime flightTimeDeparture">
          8:45a
         </div>
         <div class="airport" title="Los Angeles, CA- Los Angeles">
          LAX
         </div>
         <div class="flighttime flightTimeArrival">
          2:25p
         </div>
         <div class="duration">
          8h 40m
         </div>
         <div class="stopsLayovers">
          <span class="airportslist" title="Chicago, IL - O'Hare
International (ORD)">
           1 stop (ORD)
          </span>
         </div>
        </div>
         <div class="stopsLayovers">
          <span class="airportslist" title="Houston, TX - George Bush Intcntl
```

```
(IAH)">
           1 stop (IAH)
          </span>
         </div>

      <!-- end of trip detail holder -->
```

Find fare information (see appendix 4):

This is achieved by the routine flightInfoProviders which is less than 50 lines of code.

This is achieved in 5 steps:

- Click on the JavaScript Button FARES:

```
find_elements_by_xpath(".//button[@class='ui-button ui-button-small ui-
button-gray']")
```

- Find the name of the booking site (provider):

```
find_elements_by_xpath(".//td[@class='name']")
```

- Find the cost of the ticket:

```
find_elements_by_xpath(".//td[@class='total']")
```

- Find Fare codes:

```
find_elements_by_xpath(".//td[@class='fareCodes']")
```

- Retrieve the system which proposes the flight

```
get_attribute('href')
```

- Find if the proposal comes from Amadeus, ITA or other

```
find('AMADEUS')
```

APPENDIX 4: Html part used for fare information

This section shows the information for one flight

```
<div data-cabin="Economy" class="fareInformation active">
   .............
              <tbody>
               <tr>
                <td class="image">
                 <img
src="http://cdn3.kayak.com/v628241/images/booking/smaller/AIRFAREDOTCOMWHISKY
.png" />
                </td>
              <tbody>
               <tr>
                <td class="image">
                 <img
src="http://cdn4.kayak.com/v628216/images/booking/smaller/BA.png" />
                </td>
                <td class="name">
                 <div>
                  Britishairways
                 </div>
                </td>
                <td class="fareCodes">
                 <div>
                  OKW7Q2G1, OKX7Q2G1
                 </div>
                </td>
                <td class="total">
                 <a onclick="javascript: itemClicked('1902', 'Welcome back
from britishairways.com…', '', 'details/resultclick/price');"
href="/book/flight?code=-
575220996.GkACBOleZH.0.F.BA,AMADEUS.117821.2bc8ad2c6927ebbcbefb228453231fa0&a
mp;sub=E-1434b31a0e3&amp;resultid=2bc8ad2c6927ebbcbefb228453231fa0"
title="Book at britishairways.com" target="bookit1902BA">
                  $1178.21
                 </a>
                </td>
                <td class="button">
                 <button type="button"
onclick="window.open('/book/flight?code=-
575220996.GkACBOleZH.0.F.BA,AMADEUS.117821.2bc8ad2c6927ebbcbefb228453231fa0&a
mp;sub=E-1434b31a0e3&amp;resultid=2bc8ad2c6927ebbcbefb228453231fa0',
'bookit1902BA');itemClicked('1902', 'Welcome back from britishairways.com…',
'', 'details/resultclick/button');" class="ui-button ui-button-small ui-
button-gray">
                  <span>
                   Go
                  </span>
                 </button>
                </td>
               </tr>
              </tbody>
 .............
```

4.4.3 BadClick Script

4.4.3.1 Description

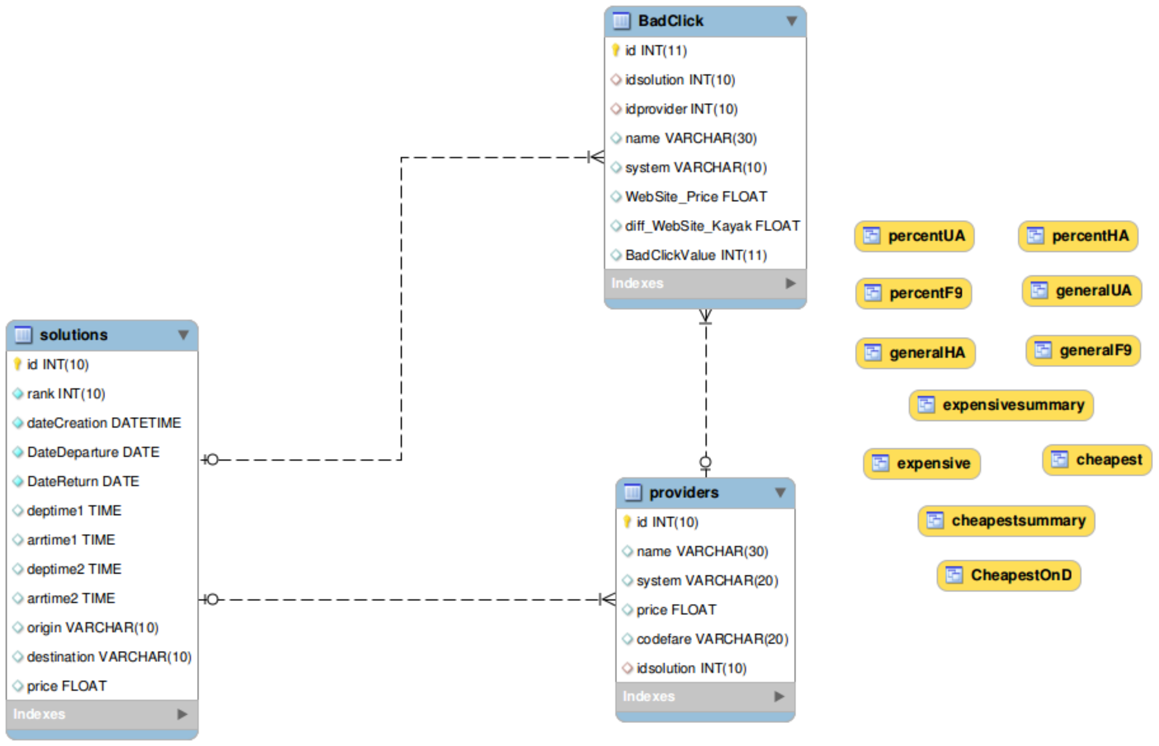
The objective of this second script is to check if the flight proposed on Kayak is the same as on the airline website. If the flight information or prices are different, we name this type of error a BadClick. In this script we parse the selected airline website to retrieve information on the flight with an ad-hoc parser . We then compare the information with the one obtained in the first script and store discrepencies in the BadClick table. Parsers can be added by creating a function for each Airline website to parse; the reference of the parser needs then to be added to the second Input file described in Paragraph 2 . For the moment two airlines (United and Hawaianair) have been parsed to check the BadClick.

4.5. DATABASE

4.5.1 Description

As mentioned previously. all information is stored in a MySQL database instance. This allows easy manipulation of data and storage of all results. History can be used, for example, to analyze how prices are varying over time. It also allows to launch different process in parallel as the contention is handled by the database engine.
For this internship project, three main tables and several views were created. SQL views are very useful for extracting and analyzing specific information.

Here it is the data model of our database. To do this model, we used the software MySQL Workbench.



Here it is an overview of our database. You can see the three main table (Solutions / Providers / BadClick) contained in our database and our several (11) views (fill in yellow) to extract information from these tables

4.5.1.1 Tables description

There are three tables in our MySQL database.

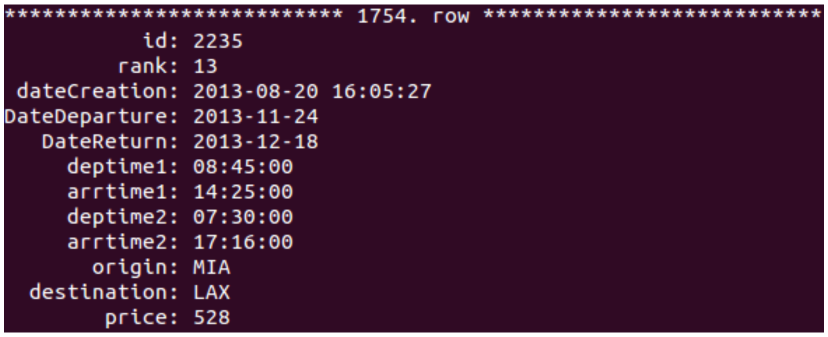
The first one is the table **Solutions**. In this table we store all flight propositions that appear on the Website for each OnD/Date in the input file. In particular, we store all flight details such as date, time of travel and airports. We store as well the date of creation to allow the script to reprocess a given OnD/Date at a later time. There will be no duplicate if we reprocess an OD/Date within 2 hours (parameter in the program) .

The SQL statement for this table is the following:

```sql
CREATE TABLE `solutions` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `rank` int(10) unsigned NOT NULL,
  `dateCreation` datetime NOT NULL,
  `DateDeparture` date NOT NULL,
  `DateReturn` date NOT NULL,
  `deptime1` time DEFAULT NULL,
  `arrtime1` time DEFAULT NULL,
  `deptime2` time DEFAULT NULL,
  `arrtime2` time DEFAULT NULL,
  `origin` varchar(10) DEFAULT NULL,
  `destination` varchar(10) DEFAULT NULL,
  `nbprov` int(10) unsigned DEFAULT NULL,
  `nb1A` int(10) unsigned DEFAULT NULL,
  `nbITA` int(10) unsigned DEFAULT NULL,
  `price` float DEFAULT NULL,
  PRIMARY KEY (`id`)
)
```

We generate a primary key using auto_increment to be able to express the relationships with the other tables
Rank is the place of the flight on the Kayak website display
dateCreation is the system date when we launch the script
DateDeparture and **DateReturn** are the dates of the departure and the return of the flight
deptime and **arrtime** are the times of the departure and arrival
origin and destination are the IATA Code of the two airports
price is the price indicated by Kayak on their website

Exemple of a row for this table:



```
*************************** 1754. row ***************************
           id: 2235
         rank: 13
  dateCreation: 2013-08-20 16:05:27
DateDeparture: 2013-11-24
   DateReturn: 2013-12-18
     deptime1: 08:45:00
     arrtime1: 14:25:00
     deptime2: 07:30:00
     arrtime2: 17:16:00
       origin: MIA
  destination: LAX
        price: 528
```

Our second table is the table **Providers**; in this table we will store all providers for each solution. Each solution comes indeed with several providers even if we have selected Amadeus or ITA-only on the command line .We store the name of the provider and the price proposed

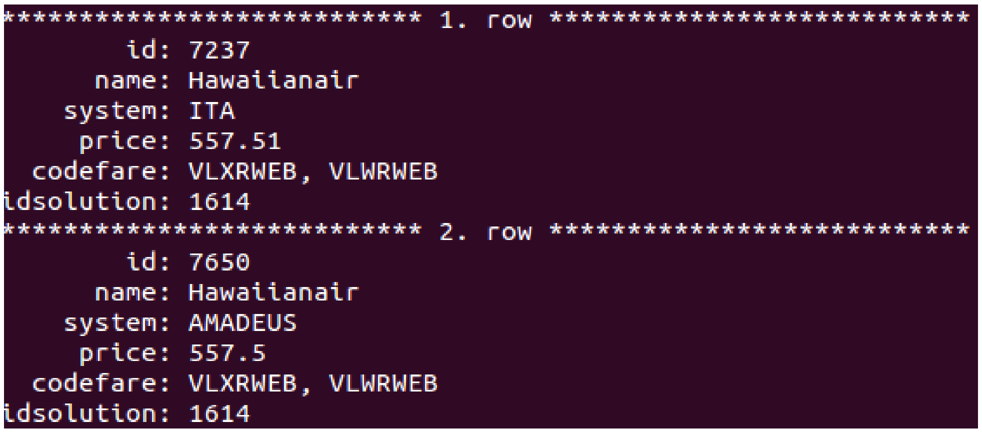
The SQL statement for the table is the following:

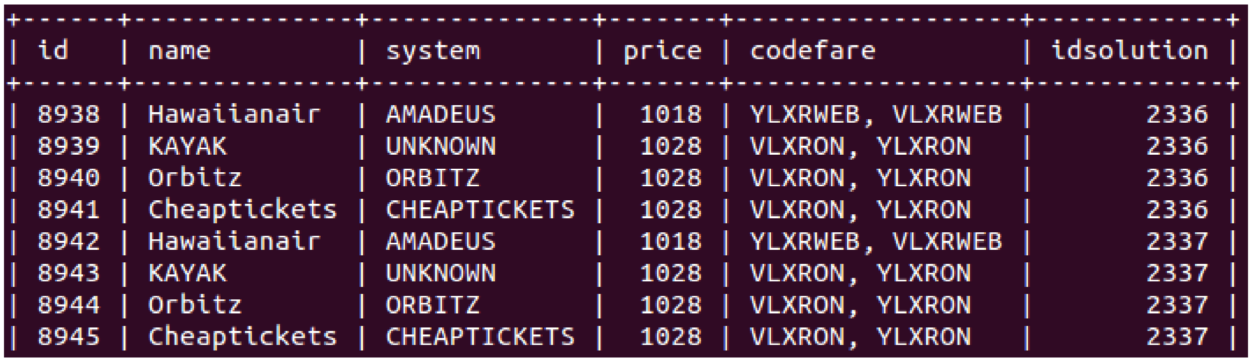
```
CREATE TABLE `providers` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(30) DEFAULT NULL,
  `system` varchar(20) DEFAULT NULL,
  `price` float DEFAULT NULL,
  `codefare` varchar(20) DEFAULT NULL,
  `idsolution` int(10) unsigned DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `const1` (`name`,`system`,`idsolution`),
  KEY `idsolution` (`idsolution`),
  CONSTRAINT `providers_ibfk_1` FOREIGN KEY (`idsolution`) REFERENCES `solutions` (`id`) ON DELETE CASCADE
)
```

In this table, we have a primary key generated using auto_increment, but we also have a foreign key which will be the id (primary key) of the table solutions. This allows to express the relationship “Solution has several Prices“

name is the name of the provider
system is the system which proposes the flight (Amadeus, ITA or others (Opodo, expedia..))
price is the price proposed by the provider on his website
codefare is the fare code for the flight
idsolution is the primary key for the flight in the table solution

Exemple of rows for this table:


```
*************************** 1. row ***************************
        id: 7237
      name: Hawaiianair
    system: ITA
     price: 557.51
  codefare: VLXRWEB, VLWRWEB
idsolution: 1614
*************************** 2. row ***************************
        id: 7650
      name: Hawaiianair
    system: AMADEUS
     price: 557.5
  codefare: VLXRWEB, VLWRWEB
idsolution: 1614
```


```
+------+-------------+-------------+-------+------------------+------------+
| id   | name        | system      | price | codefare         | idsolution |
+------+-------------+-------------+-------+------------------+------------+
| 8938 | Hawaiianair | AMADEUS     |  1018 | YLXRWEB, VLXRWEB |       2336 |
| 8939 | KAYAK       | UNKNOWN     |  1028 | VLXRON, YLXRON   |       2336 |
| 8940 | Orbitz      | ORBITZ      |  1028 | VLXRON, YLXRON   |       2336 |
| 8941 | Cheaptickets | CHEAPTICKETS |  1028 | VLXRON, YLXRON   |       2336 |
| 8942 | Hawaiianair | AMADEUS     |  1018 | YLXRWEB, VLXRWEB |       2337 |
| 8943 | KAYAK       | UNKNOWN     |  1028 | VLXRON, YLXRON   |       2337 |
| 8944 | Orbitz      | ORBITZ      |  1028 | VLXRON, YLXRON   |       2337 |
| 8945 | Cheaptickets | CHEAPTICKETS |  1028 | VLXRON, YLXRON   |       2337 |
```

The third table allows to store **BadClick** information for a provider/solution.
In this development , we have implemented United and Hawaianair but it would be possible to check other providers with limited effort.

The SQL statement for this table is the following:

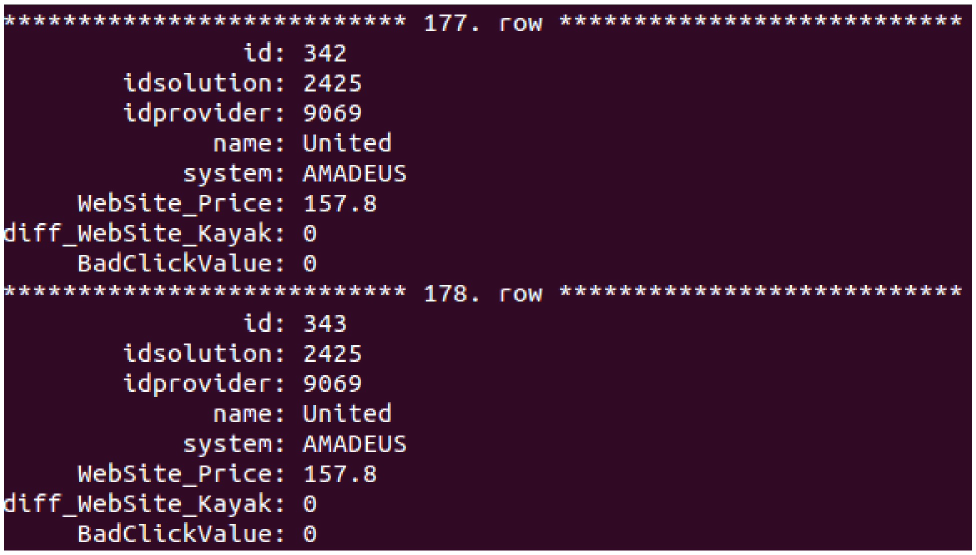
```
CREATE TABLE `BadClick` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `idsolution` int(10) unsigned DEFAULT NULL,
  `idprovider` int(10) unsigned DEFAULT NULL,
  `name` varchar(30) DEFAULT NULL,
  `system` varchar(10) DEFAULT NULL,
  `WebSite_Price` float DEFAULT NULL,
  `diff_WebSite_Kayak` float DEFAULT NULL,
  `BadClickValue` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `idprovider` (`idprovider`),
  UNIQUE KEY `sol_prov` (`idsolution`,`idprovider`),
  CONSTRAINT `BadClick_ibfk_1` FOREIGN KEY (`idsolution`) REFERENCES `solutions` (`id`) ON DELETE CASCADE,
  CONSTRAINT `BadClick_ibfk_2` FOREIGN KEY (`idprovider`) REFERENCES `providers` (`id`) ON DELETE CASCADE
)
```

In this table, we have a primary key which is generated using auto_increment. We have also two foreign keys; one is to reference the primary key of the table **Solutions** and the other is to reference the primary of the table **Providers**. Those two keys are useful because we can know for each BadClick, the flight and the provider which is affected using a SQL JOIN. Those 2 keys allow to express the relationship “A bad Click is associated to one solution and one provider”.

As said previously, this modeling would allow to extend the handling of bad clicks to other providers than United and Hawaianair.
We store in this table a BadClick value which can take the following values:
. 0 if there is no discrepency.
. -1 if prices are different.
. -2 if itinerary dates are different.

We store as well the price retrieved from the Airline Website and the difference with the price retrieved from Kayak.

Exemple of rows for this table:



```
*************************** 177. row ***************************
               id: 342
       idsolution: 2425
       idprovider: 9069
             name: United
           system: AMADEUS
    WebSite_Price: 157.8
diff_WebSite_Kayak: 0
    BadClickValue: 0
*************************** 178. row ***************************
               id: 343
       idsolution: 2425
       idprovider: 9069
             name: United
           system: AMADEUS
    WebSite_Price: 157.8
diff_WebSite_Kayak: 0
    BadClickValue: 0
```

There is no discrepancy for the two rows.

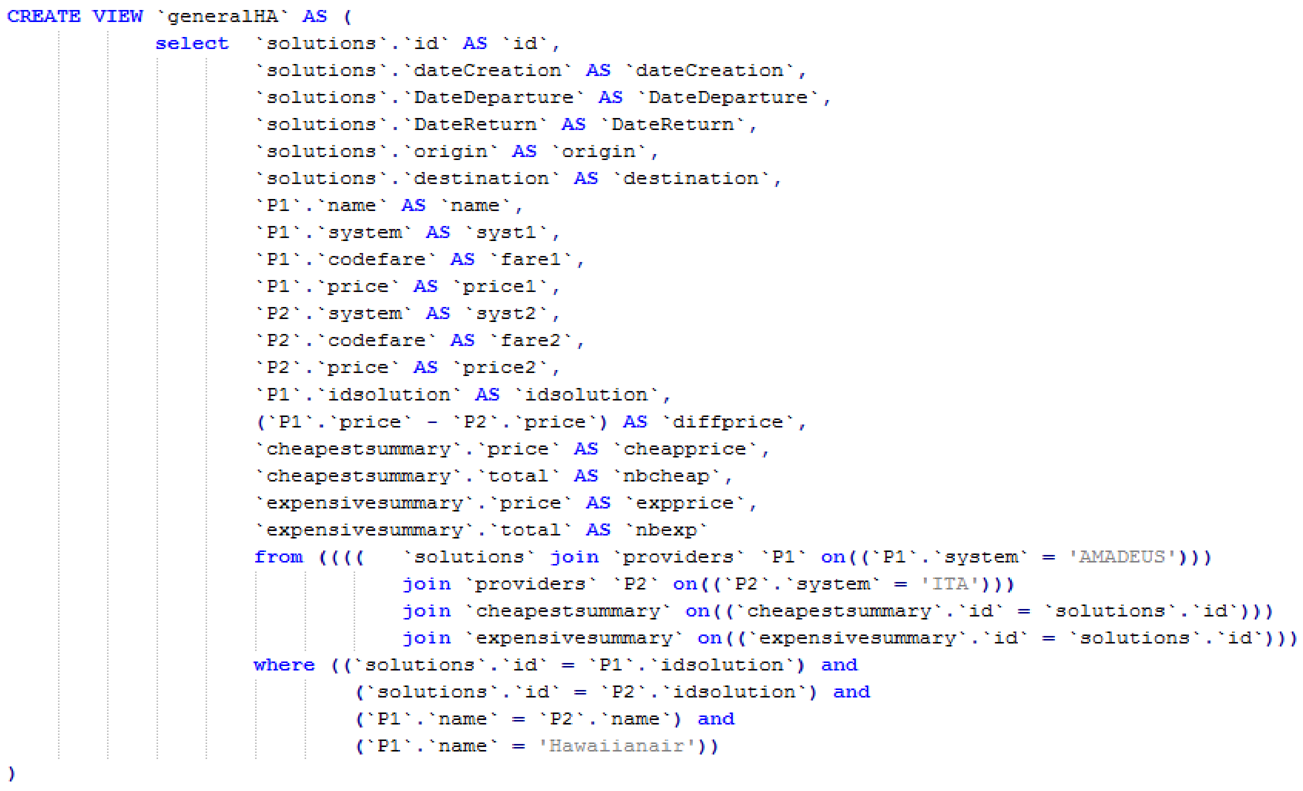
4.5.1.2 Views

To extract and summarize information from our database, we propose to create SQL Views. In SQL language, a view is a virtual table based on the result set of an SQL statement. In particular, a view will allow to select specific rows and columns from different tables and to aggregate them for a specific output.
We have created 5 types of views:

First view type is generalXX where XX is the airline code.

The view general is the view which will compare Amadeus to ITA for a specific airline. In this view we will extract the flights from the table solutions then we will retrieve all information in the table providers where the system is Amadeus or ITA. It will allow us to have a better view of the comparison between Amadeus and ITA. In this view, we select only flights which are proposed by Amadeus and ITA.

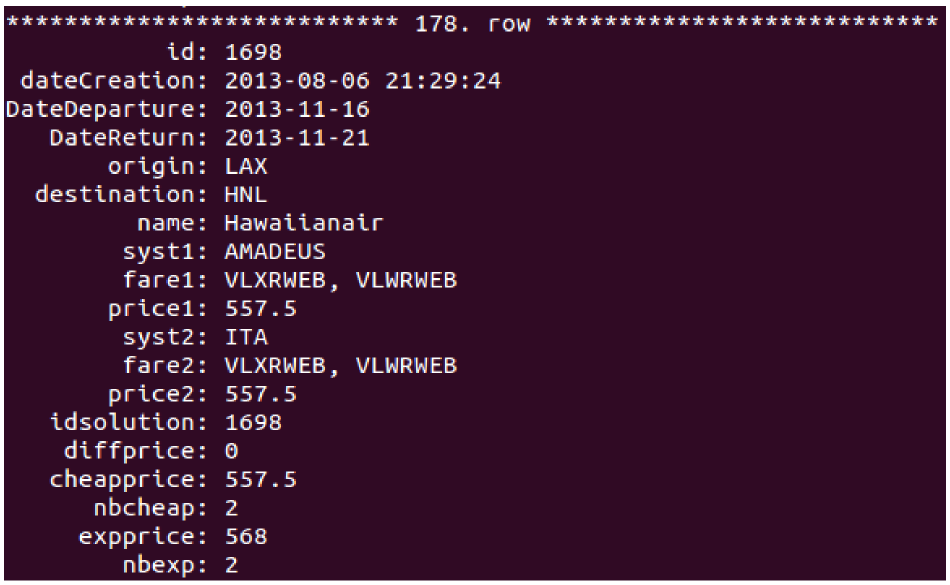
The SQL statement for this view is the following:



```
CREATE VIEW `generalHA` AS (
            select  `solutions`.`id` AS `id`,
                    `solutions`.`dateCreation` AS `dateCreation`,
                    `solutions`.`DateDeparture` AS `DateDeparture`,
                    `solutions`.`DateReturn` AS `DateReturn`,
                    `solutions`.`origin` AS `origin`,
                    `solutions`.`destination` AS `destination`,
                    `P1`.`name` AS `name`,
                    `P1`.`system` AS `syst1`,
                    `P1`.`codefare` AS `fare1`,
                    `P1`.`price` AS `price1`,
                    `P2`.`system` AS `syst2`,
                    `P2`.`codefare` AS `fare2`,
                    `P2`.`price` AS `price2`,
                    `P1`.`idsolution` AS `idsolution`,
                    (`P1`.`price` - `P2`.`price`) AS `diffprice`,
                    `cheapestsummary`.`price` AS `cheapprice`,
                    `cheapestsummary`.`total` AS `nbcheap`,
                    `expensivesummary`.`price` AS `expprice`,
                    `expensivesummary`.`total` AS `nbexp`
                    from ((((   `solutions` join `providers` `P1` on((`P1`.`system` = 'AMADEUS')))
                                join `providers` `P2` on((`P2`.`system` = 'ITA')))
                                join `cheapestsummary` on((`cheapestsummary`.`id` = `solutions`.`id`)))
                                join `expensivesummary` on((`expensivesummary`.`id` = `solutions`.`id`)))
                    where ((`solutions`.`id` = `P1`.`idsolution`) and
                            (`solutions`.`id` = `P2`.`idsolution`) and
                            (`P1`.`name` = `P2`.`name`) and
                            (`P1`.`name` = 'Hawaiianair'))
)
```

dateCreation is the system date when we launch the script
DateDeparture and **DateReturn** are the dates of the departure and of the return of the flight
origin and **destination** are the IATA Code of the two airports
name is the name of the provider
syst1, **fare1** and **price1** are information about fare code and price proposed by AMADEUS for this flight
syst2, **fare2** and **price2** are information about fare code and price proposed by ITA for this flight
idsolution is the primary key of the flight in the table solution
diffprice is the difference between the price proposed by AMADEUS and by ITA
cheapprice is the cheapest price proposed by any provider for this flight
nbcheap is the number of providers which proposed the cheapest price
expprice is the most expensive price proposed by any provider for this flight
nbexp is the number of providers which proposed the most expensive price

Exemple of a row for this table:



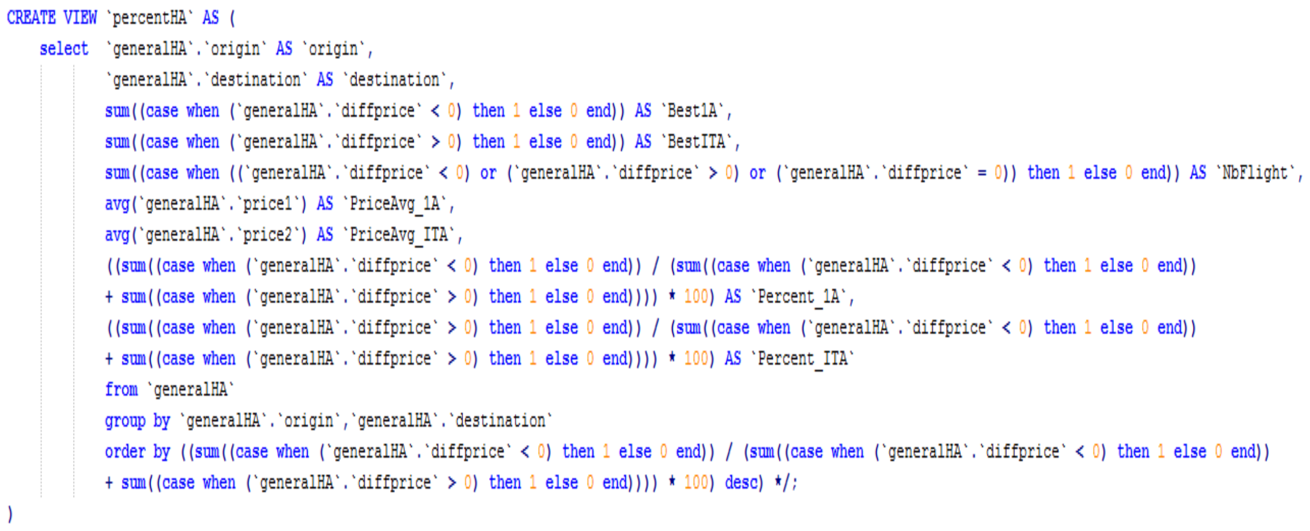
```
*************************** 178. row ***************************
           id: 1698
 dateCreation: 2013-08-06 21:29:24
DateDeparture: 2013-11-16
   DateReturn: 2013-11-21
       origin: LAX
  destination: HNL
         name: Hawaiianair
        syst1: AMADEUS
        fare1: VLXRWEB, VLWRWEB
       price1: 557.5
        syst2: ITA
        fare2: VLXRWEB, VLWRWEB
       price2: 557.5
   idsolution: 1698
    diffprice: 0
   cheapprice: 557.5
      nbcheap: 2
     expprice: 568
        nbexp: 2
```

In this case , both Amadeus and ITA have proposed the cheapest price.

Second view type is percentXX where XX is the airline code.

The view **percentHA** is specific to Hawaiianair. In this view, we will have a groupBy on OnDs. This will provide a summary of the number of times they have been cheaper with Amadeus or with ITA . This is interesting as it allows to know on which OnD ITA is better than Amadeus and to take action.

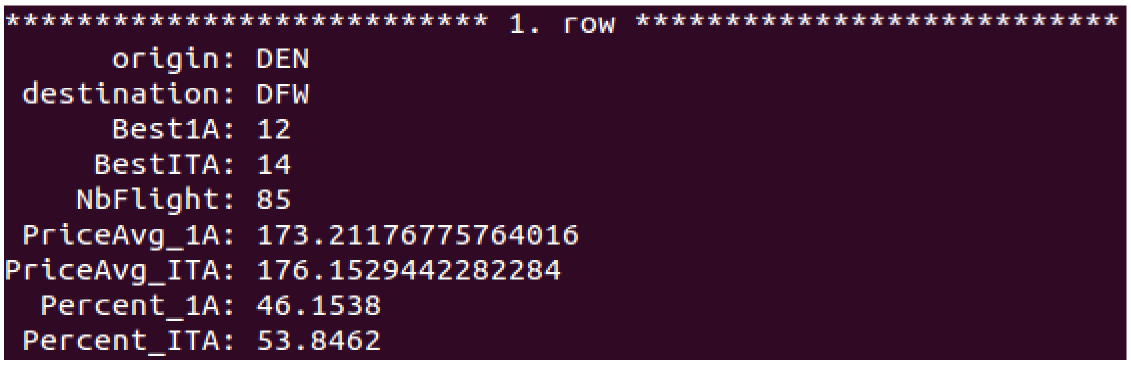
The SQL statement for this view is the following:



```
CREATE VIEW `percentHA` AS (
    select  `generalHA`.`origin` AS `origin`,
            `generalHA`.`destination` AS `destination`,
            sum((case when (`generalHA`.`diffprice` < 0) then 1 else 0 end)) AS `Best1A`,
            sum((case when (`generalHA`.`diffprice` > 0) then 1 else 0 end)) AS `BestITA`,
            sum((case when ((`generalHA`.`diffprice` < 0) or (`generalHA`.`diffprice` > 0) or (`generalHA`.`diffprice` = 0)) then 1 else 0 end)) AS `NbFlight`,
            avg(`generalHA`.`price1`) AS `PriceAvg_1A`,
            avg(`generalHA`.`price2`) AS `PriceAvg_ITA`,
            ((sum((case when (`generalHA`.`diffprice` < 0) then 1 else 0 end)) / (sum((case when (`generalHA`.`diffprice` < 0) then 1 else 0 end))
            + sum((case when (`generalHA`.`diffprice` > 0) then 1 else 0 end)))) * 100) AS `Percent_1A`,
            ((sum((case when (`generalHA`.`diffprice` > 0) then 1 else 0 end)) / (sum((case when (`generalHA`.`diffprice` < 0) then 1 else 0 end))
            + sum((case when (`generalHA`.`diffprice` > 0) then 1 else 0 end)))) * 100) AS `Percent_ITA`
            from `generalHA`
            group by `generalHA`.`origin`,`generalHA`.`destination`
            order by ((sum((case when (`generalHA`.`diffprice` < 0) then 1 else 0 end)) / (sum((case when (`generalHA`.`diffprice` < 0) then 1 else 0 end))
            + sum((case when (`generalHA`.`diffprice` > 0) then 1 else 0 end)))) * 100) desc) */;
)
```

origin and **destination** are the IATA Code of the two airports
Best1A is the number of cheapest flights proposed by Amadeus (we check if the difference price between Amadeus and ITA from the view generalXX is negative)
BestITA is the number of cheapest flights proposed by ITA (we check if the difference price between Amadeus and ITA from the view generalXX is positive)
NbFlight is the number of flights (we check if the difference price between Amadeus and ITA from the view generalXX is negative, positive or equal to 0)
PriceAvg_1A is the average of the price of every flight proposed by Amadeus on the OnD
PriceAvg_ITA is the average of the price of every flight proposed by ITA on the OnD
Percent_1A is the percentage of cheapest flight proposed by Amadeus
Percent_ITA is the percentage of cheapest flight proposed by Amadeus
Percent_1A and **Percent_ITA** are related to Best1A and BestITA only

Here is an example of a row from the view percent_XX



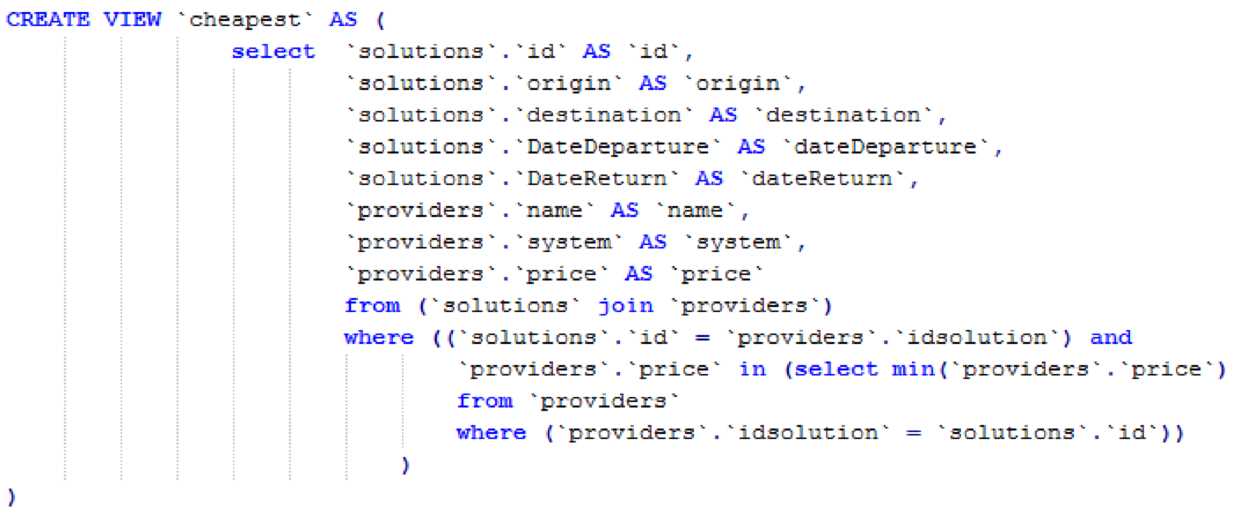
```
*************************** 1. row ***************************
      origin: DEN
 destination: DFW
      Best1A: 12
     BestITA: 14
    NbFlight: 85
 PriceAvg_1A: 173.2117677564016
PriceAvg_ITA: 176.1529442282284
  Percent_1A: 46.1538
 Percent_ITA: 53.8462
```

Here we can see that on the OnD DEN-DFW, ITA is cheaper than Amadeus, however, in most of cases, Amadeus and ITA proposed the same price (59 flights)

Third view type is cheapest

The view **cheapest** will extract the less expensive provider for each flight. There is the same view for the most expensive price and it is called **expensive**

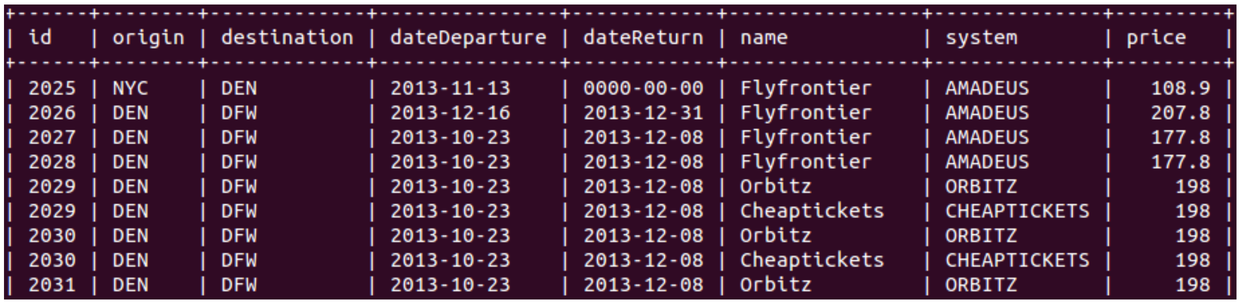
The SQL statement for this view is the following:



```
CREATE VIEW `cheapest` AS (
                select  `solutions`.`id` AS `id`,
                        `solutions`.`origin` AS `origin`,
                        `solutions`.`destination` AS `destination`,
                        `solutions`.`DateDeparture` AS `dateDeparture`,
                        `solutions`.`DateReturn` AS `dateReturn`,
                        `providers`.`name` AS `name`,
                        `providers`.`system` AS `system`,
                        `providers`.`price` AS `price`
                        from (`solutions` join `providers`)
                        where ((`solutions`.`id` = `providers`.`idsolution`) and
                                `providers`.`price` in (select min(`providers`.`price`)
                                from `providers`
                                where (`providers`.`idsolution` = `solutions`.`id`))
                            )
)
```

id is the primary key for the flight in the table solutions .
origin and **destination** are the IATA Code of the two airports
DateDeparture and **DateReturn** are the dates of the departure and the return of the flight
name is the name of the provider
system is the name of the system which proposed the price (AMADEUS or ITA)
price is the price proposed

Here is an example for the rows in this view:



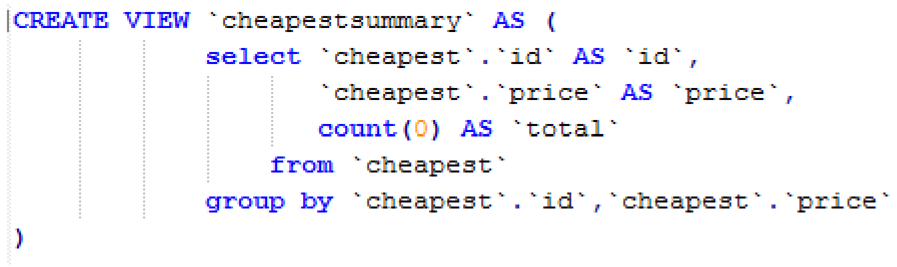
```
+------+--------+-------------+---------------+------------+----------------+--------------+---------+
| id   | origin | destination | dateDeparture | dateReturn | name           | system       | price   |
+------+--------+-------------+---------------+------------+----------------+--------------+---------+
| 2025 | NYC    | DEN         | 2013-11-13    | 0000-00-00 | Flyfrontier    | AMADEUS      |   108.9 |
| 2026 | DEN    | DFW         | 2013-12-16    | 2013-12-31 | Flyfrontier    | AMADEUS      |   207.8 |
| 2027 | DEN    | DFW         | 2013-10-23    | 2013-12-08 | Flyfrontier    | AMADEUS      |   177.8 |
| 2028 | DEN    | DFW         | 2013-10-23    | 2013-12-08 | Flyfrontier    | AMADEUS      |   177.8 |
| 2029 | DEN    | DFW         | 2013-10-23    | 2013-12-08 | Orbitz         | ORBITZ       |     198 |
| 2029 | DEN    | DFW         | 2013-10-23    | 2013-12-08 | Cheaptickets   | CHEAPTICKETS |     198 |
| 2030 | DEN    | DFW         | 2013-10-23    | 2013-12-08 | Orbitz         | ORBITZ       |     198 |
| 2030 | DEN    | DFW         | 2013-10-23    | 2013-12-08 | Cheaptickets   | CHEAPTICKETS |     198 |
| 2031 | DEN    | DFW         | 2013-10-23    | 2013-12-08 | Orbitz         | ORBITZ       |     198 |
```

amadeus
Your technology partner

Fourth view type is cheapestsummary:

cheapestsummary is a view that counts the number of providers which are proposing the less expensive price. This view is related to the cheapest view. There is the same view for the most expensive price . It is called **Expensivesummary** and is related to the view expensive.

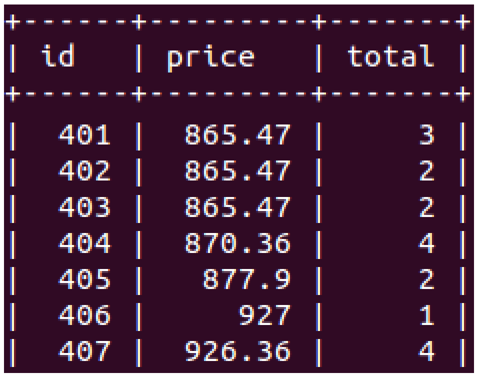
The SQL statement for this view is the following:



```
CREATE VIEW `cheapestsummary` AS (
            select `cheapest`.`id` AS `id`,
                   `cheapest`.`price` AS `price`,
                   count(0) AS `total`
                from `cheapest`
            group by `cheapest`.`id`,`cheapest`.`price`
)
```

id is the primary key for the flight in the table solution
price is the price of the less expensive provider
total is the number of provider which proposed this price

Here is an example for the rows in this view:



```
+------+---------+-------+
| id   | price   | total |
+------+---------+-------+
|  401 |  865.47 |     3 |
|  402 |  865.47 |     2 |
|  403 |  865.47 |     2 |
|  404 |  870.36 |     4 |
|  405 |   877.9 |     2 |
|  406 |     927 |     1 |
|  407 |  926.36 |     4 |
```

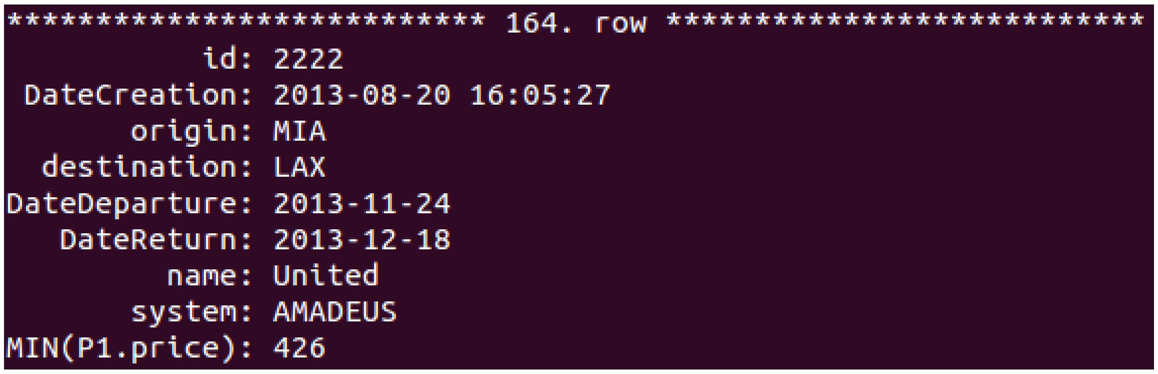
Fifth view type is **CheapestOnD**
This view allows to retrieve the cheapest provider (Amadeus or ITA) for each OnD / Dates.

The SQL statement for this view is the following:

```
CREATE VIEW `CheapestOnD` AS (
            select `solutions`.`id` AS `id`,
                   `solutions`.`dateCreation` AS `DateCreation`,
                   `solutions`.`origin` AS `origin`,
                   `solutions`.`destination` AS `destination`,
                   `solutions`.`DateDeparture` AS `DateDeparture`,
                   `solutions`.`DateReturn` AS `DateReturn`,
                   `P1`.`name` AS `name`,
                   `P1`.`system` AS `system`,
                   min(`P1`.`price`) AS `MIN(P1.price)`
                   from (`solutions` join `providers` `P1` on((`P1`.`idsolution` = `solutions`.`id`)))
                   where ((`P1`.`system` = 'AMADEUS') or (`P1`.`system` = 'ITA'))
                   group by `solutions`.`origin`,
                            `solutions`.`destination`,
                            `solutions`.`DateDeparture`,
                            `solutions`.`DateReturn`
                    order by `solutions`.`id`
)
```

id is the primary key for the flight in the table solutions .
dateCreation is the system date when we launch the script
origin and **destination** are the IATA Code of the two airports
DateDeparture and **DateReturn** are the dates of the departure and of the return of the flight
name is the name of the provider
system is the name of the system which proposed the price (AMADEUS or ITA)
price is the cheapest price proposed on this OnD / Date

Here is an example of a row for this view:

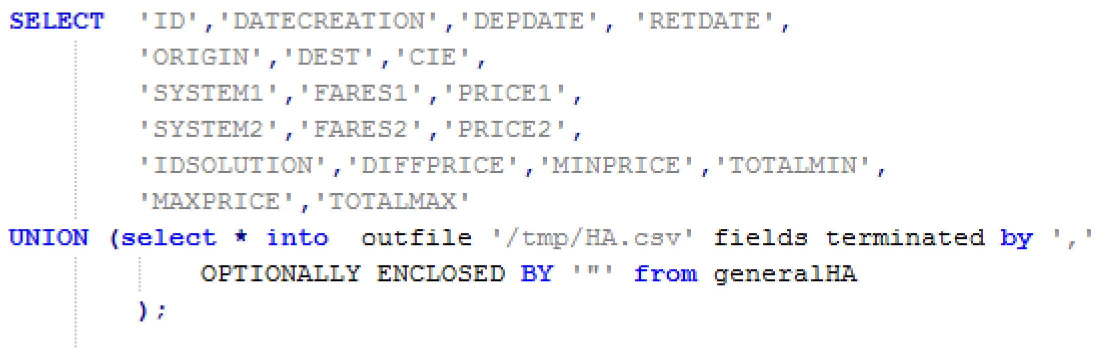


```
*************************** 164. row ***************************
           id: 2222
 DateCreation: 2013-08-20 16:05:27
       origin: MIA
  destination: LAX
DateDeparture: 2013-11-24
   DateReturn: 2013-12-18
         name: United
       system: AMADEUS
MIN(P1.price): 426
```

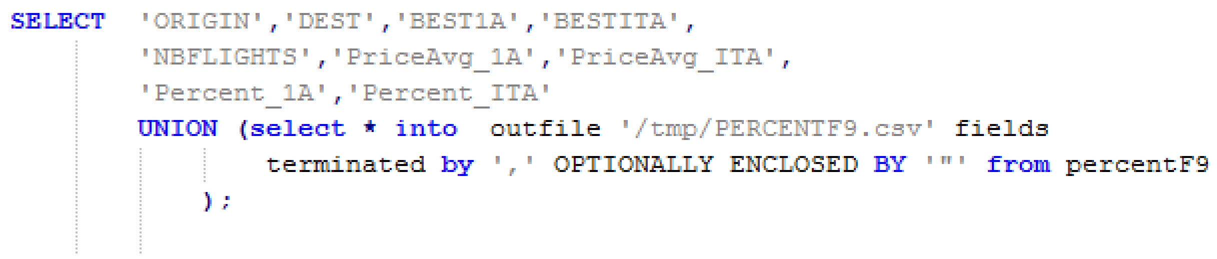
In this case, the cheapest price was proposed by Amadeus.

4.5.1.3 Excel files created from the SQL views

Excel files are very useful to analyze the results.
We have created 2 Excel files from the views using the following SQL statements:



```
SELECT  'ID','DATECREATION','DEPDATE', 'RETDATE',
        'ORIGIN','DEST','CIE',
        'SYSTEM1','FARES1','PRICE1',
        'SYSTEM2','FARES2','PRICE2',
        'IDSOLUTION','DIFFPRICE','MINPRICE','TOTALMIN',
        'MAXPRICE','TOTALMAX'
UNION (select * into  outfile '/tmp/HA.csv' fields terminated by ','
          OPTIONALLY ENCLOSED BY '"' from generalHA
        );
```



```
SELECT  'ORIGIN','DEST','BEST1A','BESTITA',
        'NBFLIGHTS','PriceAvg_1A','PriceAvg_ITA',
        'Percent_1A','Percent_ITA'
        UNION (select * into  outfile '/tmp/PERCENTF9.csv' fields
                 terminated by ',' OPTIONALLY ENCLOSED BY '"' from percentF9
              );
```

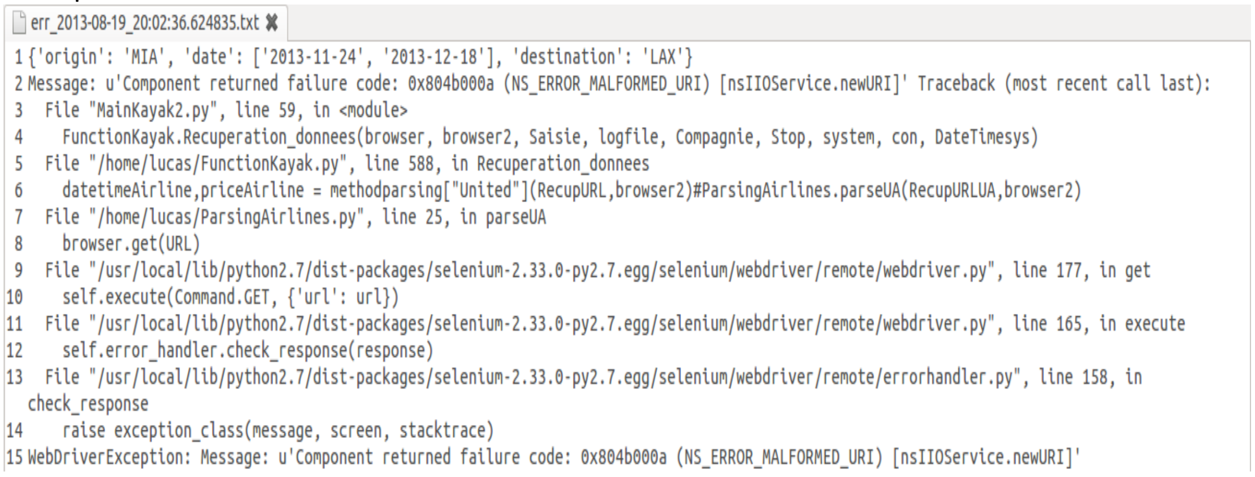
The first statement is to create the Excel file related to the View general XX and the second one is related to the View percentXX. Those views are described above.
Those files are inputs to the R script that is plotting the results.

4.6. ERRORS AND TRACES

Trace and Error files are very important. They are very convenient if we need to examine traces for the purpose of debugging. We created three files: one for traces, one for errors and the other to write the HTML from the website.
Traces allow to log the different information that we would otherwise print to the console including some HTML sources that we may wish to examine.
The error file contains the reference of the error including the full stack trace of execution. Both files allow quick investigations. They are very important as errors may occur as the vendor evolves its website.
The log file is always created and has a unique name with following pattern: log_<timestamp>.
The error file is only created in case of error and has a unique name with following pattern : err_<timestamp>.
Errors are handled using the Python Exception mechanism (try , except , raise) which is proving to be extremely useful and allows to log full trace of execution. This also allows the script to continue if an error occurs for a given OnD/Date.

This naming convention allows to execute several scripts in parallel without contention as filenames are specific to each script.

Example of an error file:



err_2013-08-19_20:02:36.624835.txt

```
{'origin': 'MIA', 'date': ['2013-11-24', '2013-12-18'], 'destination': 'LAX'}
Message: u'Component returned failure code: 0x804b000a (NS_ERROR_MALFORMED_URI) [nsIIOService.newURI]' Traceback (most recent call last):
  File "MainKayak2.py", line 59, in <module>
    FunctionKayak.Recuperation_donnees(browser, browser2, Saisie, logfile, Compagnie, Stop, system, con, DateTimesys)
  File "/home/lucas/FunctionKayak.py", line 588, in Recuperation_donnees
    datetimeAirline,priceAirline = methodparsing["United"](RecupURL,browser2)#ParsingAirlines.parseUA(RecupURLUA,browser2)
  File "/home/lucas/ParsingAirlines.py", line 25, in parseUA
    browser.get(URL)
  File "/usr/local/lib/python2.7/dist-packages/selenium-2.33.0-py2.7.egg/selenium/webdriver/remote/webdriver.py", line 177, in get
    self.execute(Command.GET, {'url': url})
  File "/usr/local/lib/python2.7/dist-packages/selenium-2.33.0-py2.7.egg/selenium/webdriver/remote/webdriver.py", line 165, in execute
    self.error_handler.check_response(response)
  File "/usr/local/lib/python2.7/dist-packages/selenium-2.33.0-py2.7.egg/selenium/webdriver/remote/errorhandler.py", line 158, in
check_response
    raise exception_class(message, screen, stacktrace)
WebDriverException: Message: u'Component returned failure code: 0x804b000a (NS_ERROR_MALFORMED_URI) [nsIIOService.newURI]'
```

We can see that the name contains the time at which the script was launched and the full trace of execution.

The HTML file contains the code source of the website; it is updated when the page is changing. This file is always created and has a unique name with following pattern:

HTMLfile_<timestamp>. It is easier to go through the source code to find a specific element

4.7. OUTPUT FILES

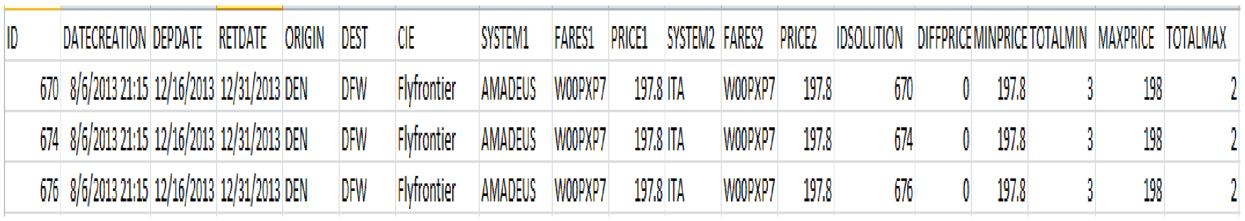
4.7.1 Overview

We have created three output files in addition to the trace and error files. Two of them are created by a SQL script and the third one with the script wrote with R-programming. The two output files created from the SQL script are Excel files described in paragraph 5 ; this makes it easier to analyze results. The R language interfaces very nicely with Excel files and provides many functions to manipulate data, to derive statistics and to generate different graphics (pie charts, curves, histograms). The file created with R is a PDF which will contain all plots resulting from the analysis of the previous files .

4.7.2 Description of files

This first CSV file is related to the View generalXX and contains the price differences between Amadeus and ITA by solution for the company XX. It contains as well the minimum and the maximum price offered for this solution by any vendor and the number of vendors proposing those prices.
This file allows to evaluate if the Amadeus proposal is competitive relative to ITA or other vendors.

Exemple of file:



ID	DATECREATION	DEPDATE	RETDATE	ORIGIN	DEST	CIE	SYSTEM1	FARES1	PRICE1	SYSTEM2	FARES2	PRICE2	IDSOLUTION	DIFFPRICE	MINPRICE	TOTALMIN	MAXPRICE	TOTALMAX
670	8/6/2013 21:15	12/16/2013	12/31/2013	DEN	DFW	Flyfrontier	AMADEUS	W00PXP7	197.8	ITA	W00PXP7	197.8	670	0	197.8	3	198	2
674	8/6/2013 21:15	12/16/2013	12/31/2013	DEN	DFW	Flyfrontier	AMADEUS	W00PXP7	197.8	ITA	W00PXP7	197.8	674	0	197.8	3	198	2
676	8/6/2013 21:15	12/16/2013	12/31/2013	DEN	DFW	Flyfrontier	AMADEUS	W00PXP7	197.8	ITA	W00PXP7	197.8	676	0	197.8	3	198	2

Description of colums:

SYSTEM, FARES and **PRICE** are information from the provider, in this excel file, the system is AMADEUS or ITA only.
DIFFPRICE is the price difference between Amadeus and ITA:

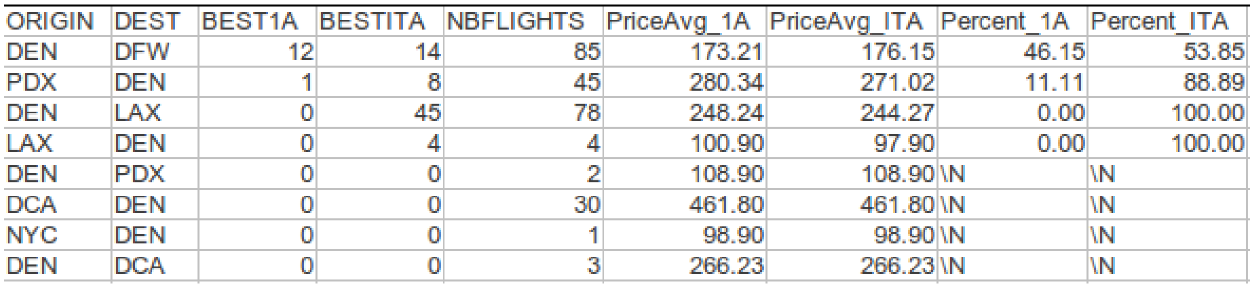
- If < 0 1A is cheaper than ITA
- If > 0 ITA is cheaper than ITA

MINPRICE is the minimum price proposed for this flight, it could be proposed by 1A, ITA or another provider as Expedia or Opodo

TOTALMIN is the number of providers that proposed the cheapest price
MAXPRICE is the most expensive price proposed for this flight, it could be proposed by 1A, ITA or another provider as Expedia or Opodo
TOTALMAX is the number of providers that proposed the most expensive price

The second CSV file is related to the View percentXX and proposes a summary per OnD. It contains ,for each OnD and all dates, which system proposes the best price (on average) between Amadeus and ITA . It contains as well the number of flights per OnD where Amadeus and ITA had respectively the best proposition.

Exemple of file:



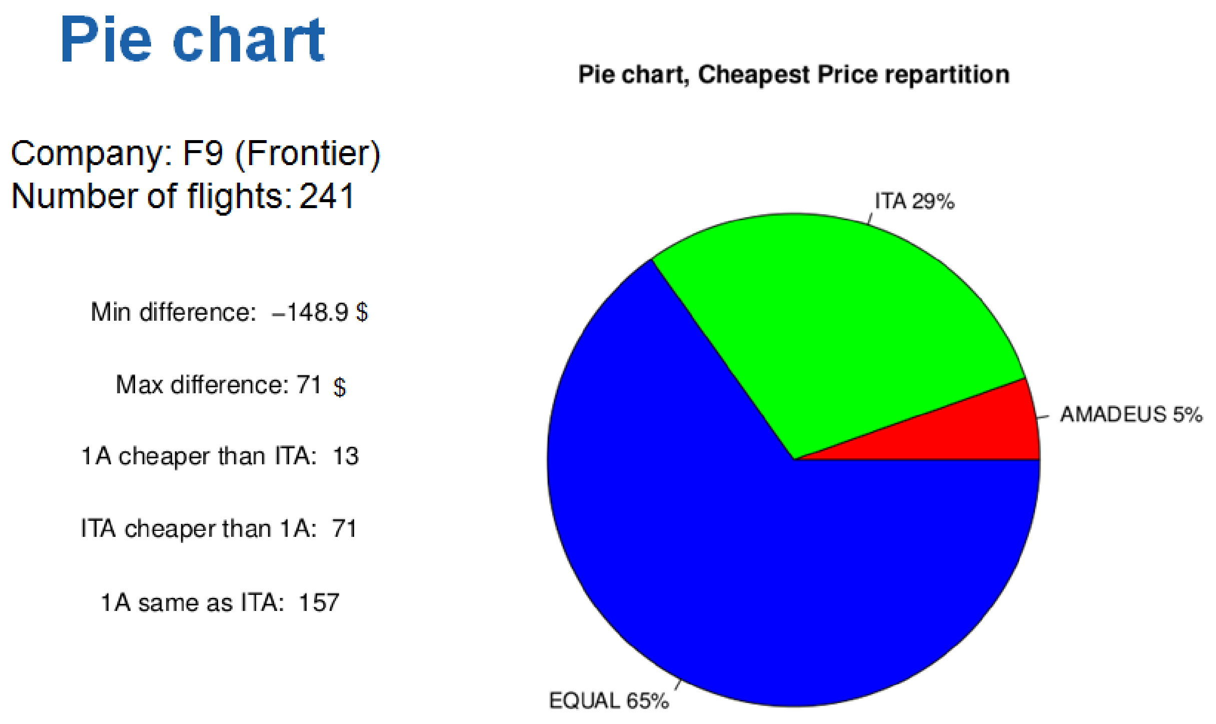
ORIGIN	DEST	BEST1A	BESTITA	NBFLIGHTS	PriceAvg_1A	PriceAvg_ITA	Percent_1A	Percent_ITA
DEN	DFW	12	14	85	173.21	176.15	46.15	53.85
PDX	DEN	1	8	45	280.34	271.02	11.11	88.89
DEN	LAX	0	45	78	248.24	244.27	0.00	100.00
LAX	DEN	0	4	4	100.90	97.90	0.00	100.00
DEN	PDX	0	0	2	108.90	108.90	\N	\N
DCA	DEN	0	0	30	461.80	461.80	\N	\N
NYC	DEN	0	0	1	98.90	98.90	\N	\N
DEN	DCA	0	0	3	266.23	266.23	\N	\N

ORIGIN and **DEST** are the IATA Code of the two airports
BEST1A is the number of flights where Amadeus was cheaper than ITA
BESTITA is the number of flights where Amadeus was more expensive
NBFLIGHT is the number of flights for each OnD
PriceAvg_XX is the average of all prices proposed by Amadeus or ITA on this OnD
Percent_XX is the percentage of cheapest flights proposed by 1A and ITA

The third output file is a PDF created with the R script. This PDF file contains the different graphics made from the results from the two previous CSV files.

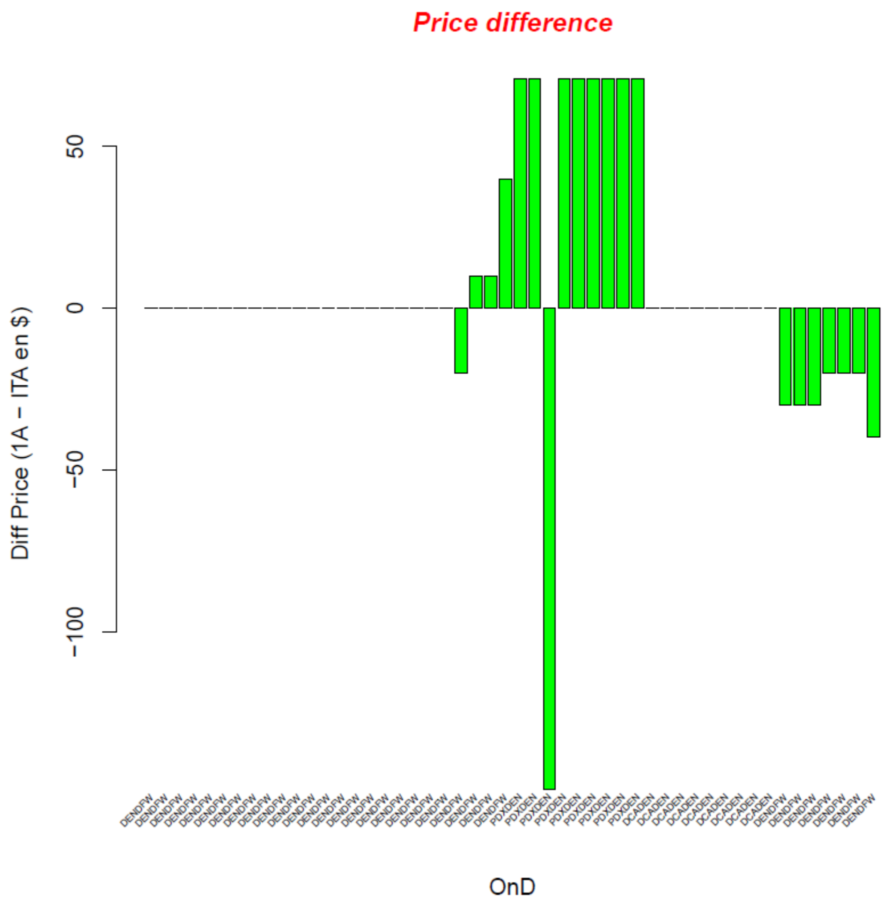
Three types of graphics have been made:

First type of graphic is a Pie Chart that shows the price repartition per vendor for all flights of a given company. It displays as well the 'min difference' which is the best difference for Amadeus and the max one which is the worst. It displays also the number of flights where Amadeus was cheaper than ITA.



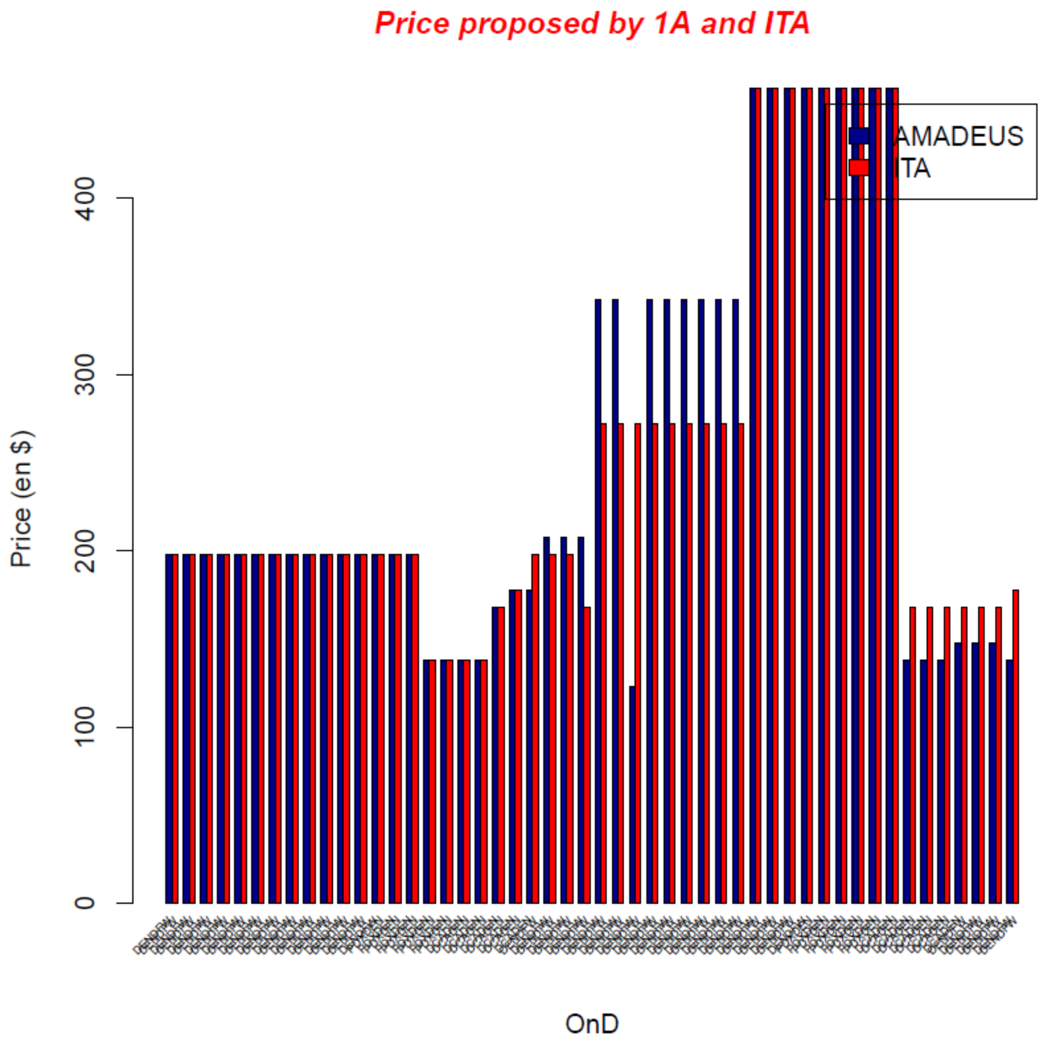
On this picture we can see that on Frontier Amadeus is not very good. In fact, Amadeus gave the cheapest price only for 170 flights while ITA gave the cheapest for 228 flights. ITA has been cheaper than Amadeus in 71 cases and Amadeus has been cheaper than ITA in 13 cases. This chart is built using information from the view generalXX. We have counted the number of cheapest price proposed by 1A and the number of similar prices.

Second type of graphic is showing the price differences between Amadeus and ITA for each flight. This plot is related to the column DIFFPRICE from the CSV created from view generalXX. This plot is still using result on Frontier. When the difference is under 0, it means that Amadeus has proposed a cheaper price than ITA. For this plot we limited the number of flights to 40 to be able to read the graph correctly.

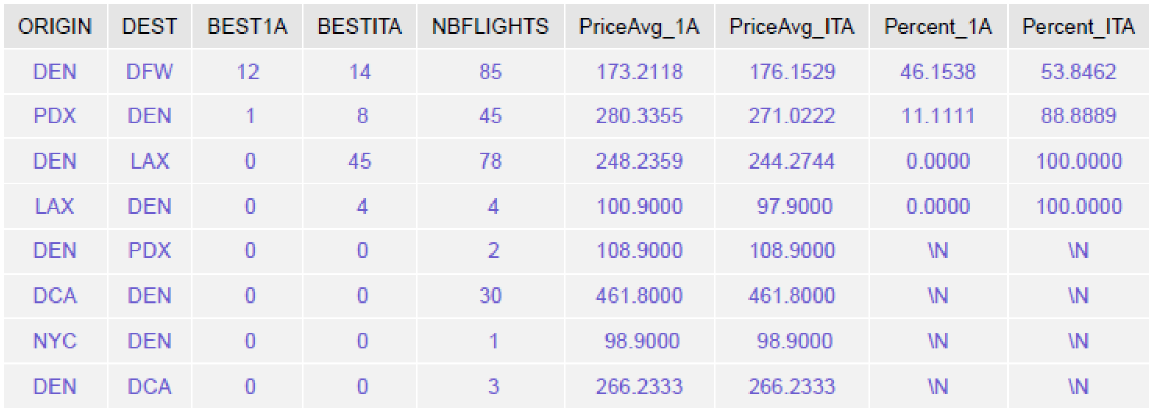


Here we can conclude that ITA is better than 1A for the first 40 OnDs even if in most of cases, Amadeus and ITA are proposing the same price. However, the biggest difference is for Amadeus.

Third type of graphic shows the price proposed by Amadeus and the price proposed by ITA for a selection of 40 OnDs. We use information from the CSV file created from the view generalXX . We retrieve PRICE1(Amadeus price) and PRICE2 (ITA price). This plot is showing results of 40 OnDs obtained for the airline F9



The fourth type of display is a table which is a summary per OnD. We use information from the CSV created from the view percentXX to get the repartition between 1A and ITA for each OnD. We then count the number of cheapest fares proposed by Amadeus and by ITA. With this table we have a better overview of which system is the best on which OnD.



ORIGIN	DEST	BEST1A	BESTITA	NBFLIGHTS	PriceAvg_1A	PriceAvg_ITA	Percent_1A	Percent_ITA
DEN	DFW	12	14	85	173.2118	176.1529	46.1538	53.8462
PDX	DEN	1	8	45	280.3355	271.0222	11.1111	88.8889
DEN	LAX	0	45	78	248.2359	244.2744	0.0000	100.0000
LAX	DEN	0	4	4	100.9000	97.9000	0.0000	100.0000
DEN	PDX	0	0	2	108.9000	108.9000	\N	\N
DCA	DEN	0	0	30	461.8000	461.8000	\N	\N
NYC	DEN	0	0	1	98.9000	98.9000	\N	\N
DEN	DCA	0	0	3	266.2333	266.2333	\N	\N

Here we can conclude that on Frontier, ITA is better on all OnD that we processed. On DEN-LAX we can see that ITA proposed the cheapest flight for more than 55%, the rest of the time Amadeus and ITA proposed the same price.

5. CONCLUSION

Appendixes