DigIt

Department of Computing Science, Umeå University, Sweden
9th June 2005

Mattias Bäckström,
Thomas Sandström Bertlin,
Niclas Dimberg
{ei00mbm,c02tsn,c02ndg}@cs.umu.se

Abstract
This report describes how to gather information about air tickets from the web and store the information in a temporal database. The information is then accessed with a web page using PHP.

1 Introduction
Ever since the first low fare airlines started operating in Europe, prices have been spiralling downward and the available number of flights have been increasing. Since the prices vary between different airlines and over time one will often have a hard choice selecting the best flight. To make a good choice price wise, one has to have a way of comparing prices from the various agencies over time since the prices change regularly. Clearly a system that is able to present price trends from different airlines, would be desirable.

This project is about creating a system that can do all that, i.e. collect information about air tickets from various web pages over time and present it in a good way. The gathered information would be stored in a database so it can be retained and analyzed.

2 Approach
The project can be divided into three parts, each covering different functionality. These are:

- Information gathering
- Database design
- Web interface

2.1 The information gatherer
To extract the information from Ryanair an effective way of getting the source data had to be found. Ryanair and several other low cost travel agencies use the javascript software Skylights. This simplified the source data collection a bit since it would be possible to use the same collection mechanism for several agencies.

The plan was to use the same http string that an ordinary user would generate when he searches for a flight. To access the parameters in an easy
way, the packet logging software *ethereal*, was used to analyze the http packets sent. The original target for data collection was Sterling but since they encrypt their traffic, parameters extraction was hard. Instead Ryanair was selected since they had no traffic encryption.

![Figure 1: Ethereal](image)

The parameters recovered were the following:

```
travel_type=onesector1_c=VST1sector1_d=LTN1sector1_d=01sector1_c=02sector1_d=01 sector_2_d=02&sector_2_p=000000&adult=1&infant=0&pup=0&c=1&1d=language=EN&mode=PARK&module=58&page=SELECT&openjaw_flag=true&ml=2 0050603VSTLTN6date1=20050603&date2=20050620&adult=2&infant=0&pup=0
```

![Figure 2: The parameters](image)

The interesting parts here are the underlined sections of the string which is used to control origin, destination, first date to travel and the number of days forward in time to be listed. This string also works on Sterling.

Now when it was possible to collect source data it was only a matter of parsing it in a smart way. For this, regular expressions were used. Since the web pages were largely static and the information desired was organized in an ordered way, this was a pretty easy task. However a different parser would have to be made for each travel agency since their layout vary.

The data retrieved was then uploaded to a postgresql database with insert and function calls.

### 2.2 The web interface

The web interface consists of a series of html sites with forms. PHP is used to make the forms dynamic with the information from the database. The final queries are also done with PHP.

Via the web interface the user can get three different kinds of information. The first kind is where the user can enter the date, origin and destination of a flight. The interface then makes a query against the database which returns the least expensive flight for the given parameters. The interface also outputs a link to the airline web page where you can book the flight.
The second kind of information is where the user enters origin, destination and the earliest date to search flights from. Now the data is returned in the form of a list sorted with the least expensive first. The departure date is not sorted. The last type draws a diagram over the price change of a specific flight entered by the user.

### 2.3 The database design

The work started by establishing the requirements for the task and the database was constructed in an Entity-Relation (ER) schema which then was translated into an SQL database schema.

![Database schema](image)

**Figure 3: The database schema**

The prices on flights change regularly over time especially near the departure date. To record these changes a temporal approach to the database was needed and was subsequently incorporated in the design.

The primary table in the database is the FLIGHT table which is constructed of several items. The primary key is build upon the values FLIGHT NUMBER and DEPARTURE. The same flight might depart more than twice a day making it possible that the same flight travels the same date. This was solved by simply building the departure key with a timestamp which is build upon the year/month/date and time of the day. This eliminates duplicate flights.

When a flight is inserted into the database, prices can then be bound to it by inserting into PRICE. To know if the price on that flight have changed is done by looking on the fields DEPARTURE and VALID_TO. If those two are the same, and the price is different compared to the value in the database, an update on the price has to be made. The table is then updated with the current price, the VALID_TO is set to current time and a new price row is made with the flight number, departure, VALID_FROM (which is the current time) and VALID_TO (which is same as the departure time).

### 3 Results

The following pictures are from the web interface. Figure (4a) shows the pages where the user can choose what to do. Figure (4b) is where the user enters the departure date, origin and destination. Figure (5) shows the result of the second search, that is all flights after the specified date between two
airports sorted by price. The first search is similar to the previous except that it only returns the flights for the given day. Figure (6) shows a diagram over the pricechange for a flight.

**Digit buyers guide**

<table>
<thead>
<tr>
<th>What do you want to do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search for tickets given day</td>
</tr>
<tr>
<td>List all tickets from a date in an ascending order</td>
</tr>
<tr>
<td>Show a histogram of the pricechange for a specified flight</td>
</tr>
</tbody>
</table>

![Image a)](image-a)
![Image b)](image-b)

**Figure 4:** **a:** The first page. The user can choose what to do. **b:** Here the user enters the origin and destination city/airport. Country has already been chosen.

<table>
<thead>
<tr>
<th>All flights from LIN to VST at date: 2005-06-08</th>
</tr>
</thead>
<tbody>
<tr>
<td>FI3602, 2005-06-28 06:25:00, 1.99, GBP, LIN, VST, [Link]</td>
</tr>
<tr>
<td>FI3602, 2005-06-21 06:25:00, 1.99, GBP, LIN, VST, [Link]</td>
</tr>
<tr>
<td>FI3606, 2005-06-28 18:30:00, 1.99, GBP, LIN, VST, [Link]</td>
</tr>
<tr>
<td>FI3602, 2005-06-30 06:25:00, 1.99, GBP, LIN, VST, [Link]</td>
</tr>
<tr>
<td>FI3602, 2005-06-29 06:25:00, 1.99, GBP, LIN, VST, [Link]</td>
</tr>
<tr>
<td>FI3602, 2005-06-20 06:25:00, 1.99, GBP, LIN, VST, [Link]</td>
</tr>
<tr>
<td>FI3602, 2005-06-20 06:25:00, 1.99, GBP, LIN, VST, [Link]</td>
</tr>
</tbody>
</table>

**Figure 5:** Some of the flights after the specified date sorted by price (not all of the result because it was to big)

![Image](image-5)

**Figure 6:** Shows the pricechange for a flight (this is an example flight that where constructed due to small amount of data in the database)
4 Discussion
One thing thing that can give a better result is more information in the database. Both from other airlines and over a longer period of time. With more information the queries would be more interesting. Perhaps some trends over the price changes would have appeared.

Due to lack of time a suboptimal solution for the information gatherer had to be used. A stored function is used to insert values properly into the database. This made it impossible to use the java (jdbc, postgresql) .addBatch() method which meant 2^n connections to the database instead of just 2 for each operator. Also since the parser uses regular expressions to get data, it is pretty sensitive for layout changes. However it is pretty simple to write a new one in that case.

A bigger and better interface could be implemented. Perhaps with some sort of user accounts where you can keep track of you flights. Maybe some sort of "watchdog" of the price which would e-mail the user when the price drops under a certain amount.

5 Conclusions
This project shows that a database collection prices and presenting them in a user friendly way can be done, but it is not without problems. There is a bit of work that has to be done with different parsers for different web sites.

It is quite the task to obtain the information needed for an effective price comparer. Only getting the cheapest tickets on Ryanair is of no use if the user wants to search for the lowest price available for all operators. A joint venture with the different operators would make the task much easier and more efficient, like database access. This project has proven that temporal databases can be very good for building a price image and presenting it in a nice way, but the work needed for building and maintaining a site that collects information from many different sites is a full time job.

References

