Aether: 600.415 Final Project

Matt Ziegelbaum, H. Parker Shelton
mziegelbaum@acm.jhu.edu, parker.shelton@jhu.edu

December 21, 2009

Contents

1 Project Description **1**

2 Database Design **2**

3 Database Population **3**

4 Architecture **4**
4.1 Backend 4
4.1.1 Stored Procedures 4
4.1.2 Dijkstra's Algorithm 4
4.2 Front End 4
4.2.1 Architecture 4
4.2.2 User Interface 4
4.2.3 Processing Data 5
4.2.4 Comments 5

5 Installation **5**
5.1 Requirements 5
5.2 Checking Out Code and Data 6
5.3 Importing Data 6
5.4 Running the Server 6
5.5 Deploying to a Production Environment 7

1 Project Description

We have developed a system, aether, that allows users to make queries for information about international air travel. We currently support queries regarding airlines, airports, and routes flown by the various airlines.

Aether is publicly available at `aether.acm.jhu.edu`.

2 Database Design

The database has the following structure:

```
CREATE TABLE Airports (
    ID INT(11) NOT NULL primary key,
    Name VARCHAR(255),
    City VARCHAR(255),
    Country VARCHAR(255),
    IATA VARCHAR(3) NOT NULL,
    ICAO VARCHAR(4) NOT NULL,
    Latitude DECIMAL(12,9),
    Longitude DECIMAL(12,9),
    Altitude INT(11),
    Timezone INT(11),
    DST CHAR(1),
    NumRunways INT(2) UNSIGNED
);

CREATE TABLE Airlines (
    ID INT(11) NOT NULL primary key,
    Name VARCHAR(255),
    Alias VARCHAR(255),
    IATA VARCHAR(2),
    ICAO VARCHAR(3) NOT NULL,
    Callsign VARCHAR(255),
    Country VARCHAR(255),
    Active CHAR(1)
);

CREATE TABLE Equipment (
    ID VARCHAR(3) NOT NULL primary key,
    Name VARCHAR(255)
);

CREATE TABLE Routes (
    Airline VARCHAR(3) NOT NULL,
    AirlineID INT(11),
    Source VARCHAR(4),
    SourceID INT(11),
    Dest VARCHAR(4),
    DestID INT(11),
    Codeshare CHAR(1),
```

```
    Equipment VARCHAR(255),
    TicketPrice FLOAT UNSIGNED,
    International CHAR(1)
);
```

3 Database Population

The core database information on airports, airlines, routes, and equipment was loaded with information extracted and modified from CSV files available from `www.openflights.org`, and AirlineRouteMapper (`arm.64hosts.com`). Additional information giving the number of runways at each airport was extracted from `www.ourairports.com`. The total data inserted into the database can be seen in the included aetherdb.sql file.

As this data set was fairly dirty, a large amount of effort was spent cleaning it appropriately, namely, ensuring that all airports and airlines had either a valid IATA or ICAO code and each route's origin and destination airports were contained in the database. Thus, 3,854 airlines without a valid IATA or ICAO code, three airlines without a valid call sign, and 70 airports without a valid IATA or ICAO code were deleted from the database. A large number of airports with no inbound or outbound routes exist in the database, but the query for their removal proved to be prohibitively costly to run, and since these airports are all well-formed, their presence does not interfere with query processing.

A heuristic algorithm was generated to populate the route information with approximate ticket prices:

```
CREATE FUNCTION TicketPrice(dist FLOAT, stops INT, startRunways
                INT, endRunways INT) RETURNS FLOAT DETERMINISTIC
BEGIN
    RETURN ((dist*0.9) / LOG((stops+2)*100)) + 100 - 40*(stops)
           - endRunways*8 - startRunways*5;
END
```

This heuristic has the properties that the longer the flight, the more expensive the ticket, flights with more stops are exponentially cheaper, and the larger the size of the departing and arriving airports, the cheaper the ticket. The distance between airports was calculated using a readily-available formula for distance given latitude and longitude. This heuristic was able to produce surprisingly accurate results for its simplicity:

```
American Airlines, Dallas-Fort Worth International Airport to
    Abiline Regional Airport = $67.74
American Airlines, Dallas-Fort Worth International Airport to
    Baltimore-Washington International Airport = $239.31
British Airways, John F. Kennedy International Airport to
    London Heathrow International Airport = $648.66
US Airways, John F. Kennedy International Airport to
    Los Angeles International Airport = $467.45
```

Any flight with the same departure and arrival airports but with with a layover (not shown) is understandably cheaper.

4 Architecture

Aether is implemented as a web interface constructed with HTML/CSS/ JavaScript over a Ruby backend and a MySQL database. It implements the Google Maps API, and uses Sinatra to handle URLs and interface with the database. The site is served in development mode by Mongrel or WebBrick. In production, the site is deployed using the fantastic Capistrano tool, and served by Apache via mod_rails (Phusion Passenger).

4.1 Backend

4.1.1 Stored Procedures

MySQL stored procedures were created to handle requests from the web interface and return result sets to Ruby, which were then filtered, organized, and returned to the client as JSON. The full list of stored procedures implemented can be seen in the included StoredProcedures.txt.

4.1.2 Dijkstra's Algorithm

The algorithm for both cheapest flight and shortest path is an adaptation of Dijkstra's shortest-path algorithm to the structure of our database and the constraints of MySQL. The implementation of both procedures can be seen in the included StoredProcedures.txt. The query requires approximately 90 seconds to calculate the shortest path through 5,425 nodes and 54,000 edges, quite impressive performance. Credit to Peter Larsson for the original SQL algorithm (`http://www.sqlteam.com/forums/topic.asp?TOPIC_ID=772620`).

4.2 Front End

4.2.1 Architecture

Aether's frontend is set up as a simple HTML page (generated from `index.haml`) with most of the functionality coming from the JavaScript code. We are using jQuery 1.3.2 to do most of the heavy lifting, and use its jQuery.getJSON function to call back to the server asynchronously.

4.2.2 User Interface

Using the Google Maps API, impressive user-interface functionality was able to be implemented in a short time. JQuery UI 1.7.2 provided a template for the accordion menu on the left, allowing the user to select which queries to run on the database. The dialog messages that appear were also provided as part of JQuery UI. The results table was implemented on

top of TableSorter 2.0, which provided the sorting and pagination functionality, and supplemented with click listeners to trigger the marker popups in the map area. Clicking on a row or a marker in the map area triggers a database request for information on the appropriate airport, which is displayed in the marker popup.

4.2.3 Processing Data

Data processing was broken up into fourt parts. First, a request is sent to the server for a query along with whatever parameters were needed. The server responds with a JavaScript Object Notation (JSON) Object which contains three lists: the records for the table, the markers for the map, and any routes to draw. The records are handled by a function that draws them out to a table, while the markers and routes are handled by another that processes them and adds them to the MarkerManager and Map. Overall, data processing is relatively swift, even for thousands of markers and table rows.

4.2.4 Comments

Through developing the front-end, we discovered that the Google Maps API, while very robust, is missing support for certain key features. Most notably, the Maps API does not have any way of managing markers or overlays. Marker management was easily taken care of by one of Google's open source libraries, MarkerManager.js, but overlay management (for the route lines) was not. Maps scales incredibly well, but browsers simply cannot handle more than a few thousand lines drawn to a map at any given time.[1]

5 Installation

The following documents how to get aether up and running locally. We assume a working installation of Ruby and RubyGems, along with a working MySQL database available.

5.1 Requirements

1. Ruby 1.8.6, Rubygems
2. MySQL
3. Git (or gzip, if you're using a tarball)
4. The following gems: mysql, sinatra, haml, sass, compass, json. To install:

```
~ $ sudo gem install mysql sinatra haml sass compass json
~ $ # Ensure MySQL gem is working:
~ $ irb <Enter>
```

[1] If you click "All Routes", you can get Safari to use more than 2GB of RAM within 5 minutes.

```
>> require 'mysql'
=> true
>> Mysql.init
=> #<Mysql:0x101179da8>
```

5. Optional gems: shotgun, capistrano (`sudo gem install shotgun capistrano`)

5.2 Checking Out Code and Data

```
~ $ git clone git://github.com/ziegs/aether.git
... wait ...
~ $ cd aether
```

5.3 Importing Data

Create a database in the MySQL server. These instructions assume the name `aether_dev`. The following commands import the data and stored procedures, respectively.

```
~ $ cd /path/to/aether
~ $ mysql -u username -p -h dbase.server aether_dev
    < data/aetherdb.sql
~ $ mysql -u username -p -h dbase.server aether_dev
    < docs/StoredProcedures.txt
```

Next, create a file in the `config` directory called `db.rb`. The contents should look like the following code block. All fields are required.

```
$db_conf = {
  :host => 'localhost',
  :username => 'username',
  :password => 'password',
  :database => 'aether_dev',
  :port => 3306
}
```

5.4 Running the Server

You have two options. The best option for development is option one below using shotgun, the other option is to use ruby.

1. Shotgun[2]

[2] `sudo gem install shotgun`

```
~ $ cd /path/to/aether
~ $ ./aether_dev
```

2. Ruby

```
~ $ cd /path/to/aether
~ $ ruby aether.rb
```

The server will now be running on port 4567. Navigate to http://localhost:4567 and start playing! Using shotgun is slightly slower, but it will automatically refresh the server if the ruby code changes. If you use option two and make changes to `aether.rb`, you will need to manually restart the server (using C-c to kill it first).

5.5 Deploying to a Production Environment

We use capistrano[3] (www.capify.org) to simplify deployment. To deploy, ensure you have access to a webserver with Phusion Passenger (mod_rails), Apache2, and git. Modify the file `config/deploy.rb` to point to the right paths and servers, then type:

```
~ $ cap deploy:check
~ $ cap deploy:setup
~ $ cap deploy
```

You'll need to place a `db.rb` file in `/var/aether`. This is currently a hardcoded path in `aether.rb`.

[3] `sudo gem install capistrano`.