Project 2
Database Design and ETL

Out: *October 1st, 2014*
Due: *October 20th, 2014, 11:59 P.M.*

1 Introduction: What is this project all about?

We've now studied many techniques that help in modeling data (E-R diagrams), which can then be migrated to a relational model (schemas), for which we have a declarative syntax for querying and modifying (SQL, modeled after relational algebra), which can be optimized to have many desirable properties (normalization into BCNF, 3NF, etc. for lossless joins, dependency preservation...).

This project is about putting it all together. Given a large amount of unstructured airline data, we want you to create a working database of that data.

Part of the title, ETL, stands for *Extract, Transform, Load*, a process for unifying multiple sources of complimentary data stored in different formats. Companies spend a **huge** amount of money on this every year, because the problem is just slippery and hairy enough to escape the grasp of most algorithms, leaving the problem to us, the DBA's.

2 Goal

Before we jump into explaining the individual components, here's a broad overview of what we'd like you to do for this project:

- Model the data in the system as both an E-R diagram and as SQL. There are a few caveats:
    1. Your model must contain all of the data we supply you with (unless otherwise specified). You are not allowed to omit any fields and any actual data we provide should be reflected in your database. The one exception is in cases of data integrity issues, which will be discussed in more depth later.
    2. The resulting schema must be in BCNF or 3NF (this shouldn't be too difficult, as the few FD's are pretty clear)
    3. For the SQL, we will look for more than naive table creation: this means labeling your primary keys, foreign keys, constraints, etc.
    4. You will be importing your schema into a SQLite database using standard SQL constructs.

- Write an application, `import`, which will use the various CSV files to populate a SQLite database.
- Write an application, `query`, which will make pre-defined queries against the SQLite database and print the results to the console.

3 Overview of the Data

The data you are working with for this project is in the form of several CSV files (available in `/course/cs127/pub/etl/`). The provided stencil code makes parsing the data trivial: the emphasis for this project is on what you do with the data once it's parsed.

To help you out, what follows is a basic overview of the data contained in the various files you will be working with. Read it over carefully and be on the lookout for structural elements to incorporate into your design.

Note: This overview may not fully explain all of the nuances of the data: you are encouraged to look at the files yourselves (CSVs are human-readable) to better understand them. You should take all of this data and be able to enter it into database of your design, avoiding redundancies.

3.1 airlines.csv

This file contains basic informations on all of the airlines. There are two fields: the first is a code that is unique to the airlines (eg: YX) and the second is the name of the airline (eg: Republic Airlines). Note that not all airlines may have flight data associated with them.

3.2 airports.csv

This file contains information on all of the airports. There are two fields: the first is a code that corresponds uniquely to a particular airport and the second is the full, canonical name of the airport. Note that not all airports may have flight data associated with them.

3.3 flights.csv

It contains information on every single flight limited to a single month of data (note that your design should still be able to accommodate data from other months and/or years!).

`flights.csv` has the following fields:

- A code that corresponds uniquely to a particular airline
- A flight number (eg: Delta Flight 123, now boarding)
- A code that corresponds uniquely to a particular airport (in this case, the origin)

- The originating airport's city
- The originating airport's state
- A code that corresponds uniquely to a particular airport (in this case, the destination)
- The destination airport's city
- The destination airport's state
- A date representing the day when the flight was scheduled to depart. Possible formats: YYYY-MM-DD, YYYY/MM/DD, MM-DD-YYYY, and MM/DD/YYYY.
- A time (either in AM/PM or 24 hour format) representing when the flight was scheduled to depart (timezone UTC)
- The difference in minutes between scheduled and actual departure time. Early departures show negative numbers.
- A date representing the day when the flight was scheduled to arrive. Possible formats: YYYY-MM-DD, YYYY/MM/DD, MM-DD-YYYY, and MM/DD/YYYY.
- A time (either in AM/PM or 24 hour format) representing when the flight was scheduled to arrive (timezone UTC)
- The difference in minutes between scheduled and actual arrival time. Early arrivals show negative numbers.
- A boolean (1 or 0) field that indicates whether a flight was cancelled
- A field indicating the number of minutes the plane was delayed due to carrier issues
- A field indicating the number of minutes the plane was delayed due to weather
- A field indicating the number of minutes the plane was delayed due to air traffic control
- A field indicating the number of minutes the plane was delayed due to security concerns

3.4 A Note on Functional Dependencies

The functional dependencies in the data may seem a bit strange at first. For instance, a flight number is not unique to an airline. The combination of an airline and flight number can be repeated multiple times per day, and is not unique even unique to an origin and a destination! Because of the messy and unregulated functional dependencies by which the

airline system operates, you may find it useful to create your own numeric primary key for flights.[1]

3.5 Data Integrity

While importing this data, you may run across some data that violates one or more foreign key constraints in your design (eg: a flight to/from an unknown airport, or by an unknown airline). In those specific cases, you must omit the violating data. Note that, for instance, an airport without a corresponding flight **is not** a data integrity issue: it's just an under-utilized airport.

Some other constraints to consider are that flights should not arrive before they depart and certain variables such as delay times should not be negative.

4 The Applications

4.1 `import`

This script is designed to load data from CSV files for flights, airport, and airlines, normalize it, and create a SQL database containing the information. It should be callable from the command line as `./import`.

A standard call to the script looks like this:

```
./import \
/course/cs127/pub/etl/airports.csv \
/course/cs127/pub/etl/airlines.csv \
/course/cs127/pub/etl/flights.csv \
~/course/cs127/etl/data.db
```

The script is not strictly required to output any information. However, verbose error messages are encouraged to aid in debugging.

4.2 `query`

This script should make pre-defined queries against a specified SQL database. The query to be executed will be specified via the command line. If the query requires input from the user (ie: a name, a start/end date, etc), that information will also be passed in via the command line.

The simplest call to the script looks like this:

[1] In general, flight numbers are up to individual airlines to assign. Many airlines tend to assign even numbers to flights headed in one direction, and odd numbers to the other direction (so return flights will often be one number higher). Sometimes, flight numbers are assigned for marketing reasons as well.

```
./query \
~/course/cs127/etl/data.db \
query1
```

Given those inputs, the application should execute Query #1 (queries are defined and numbered below) against the SQLite database at `~/course/cs127/etl/data.db`.

A more complex call for the program might look like this:

```
./query \
~/course/cs127/etl/data.db \
query8 \
"Southwest Airlines Co." \
2101 \
01/01/2012 \
01/31/2012
```

The script is expected to output the results of the query in CSV format (omitting any "header row"). The expected input and output columns for each query are described in more detail below.

4.3 Testing your Applications

4.3.1 import

To check if your `import` application is correct, we will be releasing comprehensive results for the first 3 queries. Use those results to check if you have all the correct information before proceeding onto the following queries. Just a warning, if your first 3 queries are not correct, it will be much harder for you to check your query results against ours for the following queries.

4.3.2 query

To test your `query` application, `cd` to your ETL directory and run `ant test`. Your application will be run using various inputs and compared to outputs from the TA solution code. The testcases can be found in `/course/cs127/pub/etl/tests`. Your application should pass all of the testcases: this is one of the major ways your handin will be evaluated.
If you believe that the script is returning incorrect results, please feel free to contact the TAs. Be sure to provide relevant lines of code so the TAs can evaluate your objection.

5 Queries

You will need to design SQL queries for your database that answer the following questions. Unless otherwise noted, all queries should be composed of a single SQL statement.

1. Count the number of airport codes.

   Input: N/A

   Output: One column. Number of airport codes.

   Note: You can check the correct output at `/course/cs127/pub/etl/tests/0001/output`

2. Count the number of airline codes.

   Input: N/A

   Output: One column. Number of airline codes.

   Note: You can check the correct output at `/course/cs127/pub/etl/tests/0002/output`

3. Count the number of total flights.

   Input: N/A

   Output: One column. Number of flights.

   Note: You can check the correct output at `/course/cs127/pub/etl/tests/0003/output`

4. Get all the reasons flights were delayed, along with their frequency, in order from highest frequency to lowest.

   Input: N/A

   Output: Two columns. The first column should be a string describing the type of delay. The four types of delays are Carrier Delay, Weather Delay, Air Traffic Delay, and Security Delay (Make sure to adhere these delay names). The second column should be the number of flights that experienced that type of delay. The results should be in order from largest number of flights to smallest.

5. Return details for a specified airline code and flight number scheduled to depart on a particular day.

   Input 1: An airline code (eg: AA)

   Input 2: A flight number.

   Input 3: A month (1 = January, 2 = February, ..., 12 = December)

   Input 4: A day (1, 2 ... 31)

   Input 5: A year (2010, 2011, 2012, etc)

   Output: Three columns. In this order: departing airport code, arriving airport code, and scheduled date and time of departure (in format YYYY-MM-DD HH:MM).

6. Get all airlines, along with the number of flights by that airline which were scheduled to depart on a particular day (whether or not they departed). Results should be ordered from highest frequency to lowest frequency, and then ordered alphabetically by airline name, A-Z.

Input 1: A month (1 = January, 2 = February, ..., 12 = December)

Input 2: A day (1, 2 ... 31)

Input 3: A year (2010, 2011, 2012, etc)

Output: Two columns. The first column should be the name of the airline. The second column should be the number of flights matching the criteria.

7. For a specified set of airports, return the number of departing and the number of arriving planes on a particular day (scheduled departures/arrivals). Results should be ordered alphabetically by airport name, A-Z.

   Input 1: A month (1 = January, 2 = February, ..., 12 = December)

   Input 2: A day (1, 2 ... 31)

   Input 3: A year (2010, 2011, 2012, etc)

   Input 4 .. n: The full, canonical name of an airport (ie: LaGuardia).

   Output: Three columns. The first column should be the name of the airport. The second column should be the number of flights that were scheduled to depart the airport on the specified day. The third column should be the number of flights that were scheduled to arrive at the airport on the specified day.

8. Calculate statistics for a specified flight (Airline / Flight Number) scheduled to depart during a specified range of dates (inclusive of both start and end).

   Input 1: An airline name (ie: American Airlines Inc.).

   Input 2: A flight number.

   Input 3: A start date, in MM/DD/YYYY format.

   Input 4: An end date, in MM/DD/YYYY format.

   Output: Six columns:

   (a) The total number of times the flight was scheduled
   (b) The number of times it was cancelled
   (c) The number of times it departed early or on time and was not cancelled
   (d) The number of times it departed late and was not cancelled
   (e) The number of times it arrived early or on time and was not cancelled
   (f) The number of times it arrived late and was not cancelled

9. If I had wanted to get from one city to another on a specific day (flight must have taken off and landed on the specified day), what were my options if I limited myself to one hop (aka: a direct flight)? Results should be sorted by total flight duration, lowest to highest, and then sorted alphabetically by airline code, A-Z. Remember that we're looking at historical data: as such, we're interested in actual departure/arrival times, inclusive of delays.

Input 1: A departure city name (ie: Providence, Newark, etc).

Input 2: A departure state name (ie: Rhode Island, New York, etc).

Input 3: An arrival city name (ie: Providence, Newark, etc).

Input 4: An arrival state name (ie: Rhode Island, New York, etc).

Input 5: A date, in MM/DD/YYYY format.

Output: Seven columns, each row representing a flight:

(a) The airline code
(b) The flight number
(c) The departure airport code
(d) The departure time (HH:MM)
(e) The arrival airport code
(f) The arrival time (HH:MM)
(g) The total duration of the flight of minutes.

10. Same as above, but for two hops. Results should be sorted by total duration, and then sorted alphabetically by airline code for each hop.

Input 1: A departure city name (ie: Providence, Newark, etc).

Input 2: A departure state name (ie: Rhode Island, New York, etc).

Input 3: An arrival city name (ie: Providence, Newark, etc).

Input 4: An arrival state name (ie: Rhode Island, New York, etc).

Input 5: A date, in MM/DD/YYYY format.

Output: Thirteen columns, each row representing a series of flights. For each hop, you should have:

(a) The airline code
(b) The flight number
(c) The departure airport code
(d) The departure time (HH:MM)
(e) The arrival airport code
(f) The arrival time (HH:MM)

The final column should indicate the total travel time in minutes, from departure of the first flight to arrival of the last.

11. Same as above, but for three hops. Results should be sorted by total duration, and then sorted alphabetically by airline code for each hop. **Note:** You are allowed to create a single temporary table for this query.

Input 1: A departure city name (ie: Providence, Newark, etc).

Input 2: A departure state name (ie: Rhode Island, New York, etc).

Input 3: An arrival city name (ie: Providence, Newark, etc).

Input 4: An arrival state name (ie: Rhode Island, New York, etc).

Input 5: A date, in MM/DD/YYYY format.

Output: Nineteen columns, each row representing a series of flights. For each hop, you should have:

(a) The airline code
(b) The flight number
(c) The departure airport code
(d) The departure time (HH:MM)
(e) The arrival airport code
(f) The arrival time (HH:MM)

The final column should indicate the total travel time in minutes, from departure of the first flight to arrival of the last.

6 Working on the Project

6.1 Getting Started

To get started with the Java stencil, copy `/course/cs127/pub/etl/stencil.tgz` into your course directory, and unpack it with `tar -xvzf stencil.tgz`. `cd` into the new directory (feel free to remove the `.tgz` file).

The directory contains the build file `build.xml`. This enables automation in compiling your project. To compile, while in that directory type `ant`. This automatically includes the support code in your classpath when compiling.

The directory is also an Eclipse project. That means students using Eclipse as their IDE should be able to import the project into their workspace using Eclipse's File → Import functionality.

Libraries are included as JARs in the `lib/` directory. Your code should go in `src/`.

6.2 Importing into Eclipse

1. Expand the stencil code inside your course directory. That should create a directory named "etl"
2. Open Eclipse. From the top menu bar, navigate to File → Import.
3. From there, expand the "General" tab, and select "Existing Projects into Workspace."

4. Click the "Browse" button next to "Select root directory" and browse to the etl directory inside your course directory. Click OK.

5. Check the box next to the project (if it isn't already checked) and click Finish.

7 Working with SQLite

7.1 From the command-line

SQLite is installed on all Sunlab machines. It can be accessed from the command line using sqlite3. For more information on using SQLite from the command line, see `http://www.sqlite.org/sqlite.html`

7.2 From Java

SQLite can be accessed via JDBC (Java's main database connectivity interface). There will not be an official help session on how to use JDBC, but TAs will be happy to answer questions on hours or via email. Students are highly encouraged to check out `http://web.archive.org/web/20100814175321/http://www.zentus.com/sqlitejdbc/`, which has a wonderful tutorial on working with JDBC and SQLite.

8 Tips

8.1 INSERT OR IGNORE in SQLite

The stencil code suggests that students enable foreign key constraint checking by calling PRAGMA foreign_keys = ON. This is important for ensuring the correctness of your code and we highly recommend that students do it. After executing that statement, SQLite will enforce foreign key constraints across all future queries using the same connection.

However, there is a cost associated with that constraint checking. If you are using batch inserts and any row in the batch violates a foreign key constraint, every row in the batch will fail to be inserted into the table. We suggested using INSERT OR IGNORE as a workaround: ideally, that would mean bad rows would be ignored and the rest of the rows would be inserted. However, it turns out that INSERT OR IGNORE does not work with foreign key constraints (see `http://www.sqlite.org/lang_conflict.html` if you're interested).

So what is a CS127 student to do? Well, you can validate your foreign key constraints at the application level! Before adding a new row to be inserted, make sure that any foreign key constraints are satisfied (either via a SQL query to the corresponding table or via a lookup data structure in your application). If you've done that properly, the database should never complain about a foreign key violation.

8.2 Type System in SQLite

Students can refer to the following link `http://www.sqlite.org/datatype3.html` as a reference.

Note: Think about what datatype is most appropriate for the given field.
e.g. the pros and cons of using `TEXT` as opposed to `CHAR(n)` or `VARCHAR(n)`

8.3 Date/Time Functions in SQLite

Students can refer to the following link `http://www.sqlite.org/lang_datefunc.html`, which might prove useful.

8.4 Date/Time Normalization

Since there exist multiple formats for date and time in the data, students are reponsible for normalizing them. The database won't do this automatically. Students should take advantage of DateFormat and SimpleDateFormat classes to accomplish this.

9 Handin

We expect the following components to be included in your handin (this is a reiteration of the Goal section of the handout):

- An E-R Diagram of your design.
- Your `import` application.
- Your `query` application.
- A README file, describing any bugs in your code (or asserting their absence)

You can handin your project by running the following command from the directory containing all your files:

```
/course/cs127/bin/cs127_handin etl
```

10 Final Words

Good luck, and as always, feel free to ask TA's any questions you like!