Project 2
Database Design and ETL

1 Introduction: What is this project all about?

We’ve now studied how to model data with 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.

ETL stands for Extract, Transform, Load, a process for unifying multiple sources of complimentary data stored in different formats. Companies spend a lot of money on this every year, because the problem is just slippery 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.
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/cs1270/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 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.\footnote{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.}
## 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 the arrival time of flights should be neither earlier nor the same as the departure time, and certain variables such as delay times should not be negative.

## 3.6 Schema

Your database should be modeled after the following schema. We’ve only labeled primary keys, so make sure to include other constraints in your SQL table creation:

```sql
airlines (airline_id, airline_code, airline_name)
airports (airport_id, airport_code, airport_name, city, state)
flights (flight_id, airline_id, flight_num, origin_airport_id, dest_airport_id, departure_dt, depart_diff, arrival_dt, arrival_diff, cancelled, carrier_delay, weather_delay, air_traffic_delay, security_delay)
```

**Note:** It’s worth noting that the schema we provided here use additional airline_id and airport_id as the Primary Key. The primary key is just a unique integer that we automatically assign to each tuple using the AUTOINCREMENT keyword in the DDL. In addition to this, you need to ensure that the airline code and airport code are unique.

## 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/cs1270/pub/etl/airports.csv \
/course/cs1270/pub/etl/airlines.csv \
/course/cs1270/pub/etl/flights.csv \
~/course/cs1270/etl/data.db
```
The script is not strictly required to output any information. However, verbose error messages are encouraged to aid in debugging.

If you find yourself repeatedly typing this long command, you can run:

```
alias im="./import \
  /course/cs1270/pub/etl/airports.csv \
  /course/cs1270/pub/etl/airlines.csv \
  /course/cs1270/pub/etl/flights.csv \
  ~/course/cs1270/etl/data.db"
```

As long as you remain in the same shell session, you can now run `im`, and it will be equivalent to running the long-form of the alias.

### 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:

```
./query \
~/course/cs1270/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/cs1270/etl/data.db`.

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

```
./query \
~/course/cs1270/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 have released 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 as 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 /usr/bin/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.

You can also test your SQL code outside of the Java application using the test database. See Appendix for details.

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/cs1270/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/cs1270/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/cs1270/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.

**Note1:** Try to think about what kind of SQL clause could be used for combining different records together into one table.

**Note2:** In SELECT clauses, unnamed fields are automatically given their order number as a name, for example, the first unnamed field is given the name '1' since it is the 1st field.

5. 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.

6. 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.

7. 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

**Note:** Using WITH clauses in SQL can help us saving codes and processing resources. In this query, students are expected to use WITH clause to avoid using overly repetitive code. Otherwise, points will be taken off.

8. 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 actual departure time (HH:MM)
(e) The arrival airport code
(f) The actual arrival time (HH:MM)
(g) The total duration of the flight in minutes.

9. Same as above, but for two hops. Results should be sorted by total duration, then sorted alphabetically by airline code for each hop, and then sorted by the actual depart time of the first hop, from the earliest to the latest.
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 actual departure time (HH:MM)
(e) The arrival airport code
(f) The actual 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.

Note: You cannot visit an airport in the same city and state as the origin or the destination on your way from the origin to the destination. For example, if the origin is New York, New York, and the destination is Providence, Rhode Island, then JFK → LGA, LGA → PVD is invalid because LGA is in the same city as JFK.

10. Same as above, but for three hops. Results should be sorted by total duration, then sorted alphabetically by airline code for each hop, and then sorted by the actual depart time of the first hop, from the earliest to the latest.

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 actual departure time (HH:MM)
(e) The arrival airport code
(f) The actual 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.

Note: The city, state restriction from Query 10 still holds.

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 /usr/bin/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 https://www.tutorialspoint.com/jdbc/index.htm 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 that DATETIME is a valid type.

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

In the raw data, date time is stored in string format. So students might want to use the date/time function in SQLite to convert the string into corresponding date format. Function strftime(format, timestring, modifier, modifier, ...) could be useful. It returns
the date formatted according to the format string specified in argument first. The second parameter is used to mention the time string and followed by one or more modifiers can be used to get a different result.

For example, `SELECT strftime('%Y-%m-%d %H:%M:%S', 'now')` returns the formatted text string of current date. And here is a complete list of valid strftime() substitutions:

- %d day of month: 00
- %f fractional seconds: SS.SSS
- %H hour: 00-24
- %j day of year: 001-366
- %J Julian day number
- %m month: 01-12
- %M minute: 00-59
- %s seconds since 1970-01-01
- %S seconds: 00-59
- %w day of week 0-6 with Sunday==0
- %W week of year: 00-53
- %Y year: 0000-9999
- %% %

Students can refer to the following link [http://www.sqlite.org/lang_datefunc.html](http://www.sqlite.org/lang_datefunc.html) for more details, which might prove useful.

Also, you might need to do string concatenation, and “∥” is the string concat operator in SQLite instead of “+”, which is more commonly seen in other languages.

### 8.4 Date/Time Normalization

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

Please note that our testers use Java 7 and any classes you utilize for normalization should be compatible with Java 7.

### 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/cs1270/bin/cs127_handin etl

10 Q&A

Here are some FAQs. If you have any question, check this section first to see whether there is an answer here.

• Are we being graded on coding style, efficiency and commenting for our importer code for the ETL?

No, but if your importer is incorrect then having comments and neat code would help your grader in allotting partial credit where it is due.

• When creating the airports db, should we use a JOIN to add the city and state to the airports table or is it preferable to do that within Java code?

You can use Java data structures or SQL to do that, it’s up to you! But it may be easier to use Java data structures such as HashMap/Sets however.

• Is it suitable to do some of the data integrity analysis to omit some rows before insertion attempts are made in Java?

Yeah that works. As long as your importer passes the basic test cases and you don’t get rid of the rows that are valid, it’s fine.

• How to import CSVReader? When I run "/usr/bin/ant”, I get error “package opencsv does not exist”.

import au.com.bytecode.opencsv.CSVReader;

• I find I don’t have the permission to read ouput files in the test.

That is intentional. Running ”/usr/bin/ant test” should give you a good indication of whether your queries are correct but will not reveal what is missing extra in your query results if they do not match up - that’s for you to find!

• For some airports, we cannot find their cities and states. What’s their expected value in the table?

NULL or empty string.

• For query 7, if the airport name doesn’t exist, such as ‘ABCD’, should we output a line like (‘ABCD’, 0, 0) for it, or just omit it?

Omit it.
For Query 6, you are expected to get **ALL** airlines, along with the number of flights for that airline for the given day. The question is not “get the airlines that have a flight on the given day”. Therefore, an airline could have 0 flight for a day and you would want to have that in your output.

**Are we allowed to use WITH clauses for queries 9 and 10?**

Yes.

**Should we check if the user inputs the correct number of args for our queries?**

Not necessary. You can assume that the correct number of args is always given.

**For the middle hop in query 10, would the hop for XXX → JFK then LGA → YYY be a valid path from XXX to YYY since JFK and LGA are both in New York, NY?**

No. You always have to depart from the airport you just arrived at.

## 11 Appendix

### 11.1 Test database

In addition to the test scripts mentioned above, there is a test database that contains the results of all the queries given certain parameters. This will allow you to test your SQL queries outside of the Java application using those parameters, which may be helpful in isolating bugs. The test database is located at

```
/course/cs1270/pub/etl/test.db
```

Here are the parameters:

1. N/A
2. N/A
3. N/A
4. N/A
5. Month: 2
   Day: 1
   Year: 2012
6. Month: 2
   Day: 1
Year: 2012

7. Airport Name: ‘Southwest Airlines Co.’
   Flight Number: 2101
   Start date (MM/DD/YYYY): 01/01/2012
   End date (MM/DD/YYYY): 01/31/2012

8. Departure city: ‘Boston’
   Departure state: ‘Massachusetts’
   Arrival city: ‘New York’
   Arrival state: ‘New York’
   Date (MM/DD/YYYY): 2012-01-23

   Departure state: ‘Massachusetts’
   Arrival city: ‘New York’
   Arrival state: ‘New York’
   Date (MM/DD/YYYY): 2012-01-03

10. Departure city: ‘Boston’
    Departure state: ‘Massachusetts’
    Arrival city: ‘New York’
    Arrival state: ‘New York’
    Date (MM/DD/YYYY): 2012-01-03

11.2 How to check against the test database

To check your query, \( q \), against the expected results \( q' \), you need to make sure that \( q \) has no tuples that do not appear in \( q' \) and vice versa. You can do this by selecting \((q \cup q') - (q \cap q')\). If the result is the empty set, then \( q \) returns the same tuples as \( q' \). You can also check that the orders are the same with rowid. However, to get rowid, you need to first put \( q \) into a table, which you can do by creating a TEMP table to store \( q \). Here is the SQLite code you can run, where \(<X>\) is the query number you want to check:

```sql
CREATE TEMP TABLE IF NOT EXISTS query<X>_test AS
  <Your query here>;

SELECT rowid, * FROM query<X>_test
UNION SELECT rowid, * FROM query<X>
```
EXCEPT SELECT * FROM
    (SELECT rowID, * FROM query<X>_test
     INTERSECT
     SELECT rowID, * FROM query<X>
    )

11.3 Storing and running queries in a file

A query may be long and difficult to copy-paste in sqlite3 over and over to test minor changes. To alleviate this, we suggest that you save your query in a file, say 'query.sql', then to run it use the following terminal command:

    sqlite3 path/to/db < query.sql

12 Final Words

Good luck, and as always, feel free to ask TA’s any questions!