

# Project 1

## Database Design and ETL

*Out: October 5, 2009*

***Due: October 26, 2009, 11:59 P.M.***

### 1 Silly Premise

The Stargazers at the PALS4LIFE Observatory have been keeping records of their observations through tried-and-true paper and pencil, since they used to only have 4 members. As their distribution needs increased, they printed their findings using the analog Linotype™ technology.

Due to the fact that Linotype™ replacement parts are no longer sold, PALS4LIFE seeks to digitize their information in one of those ‘databases’ they keep hearing about, and they’ve hired you to do it. Armed with nothing but their data, you need to design and implement a schema for the structurally-impaired.

### 2 Introduction

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 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 DBA’s. ETL is the reason that CS127 became CSC1270 with the introduction of Banner. The Observatory not only needs their data modeled, but they need data from different sources unified as well. . .

What is required from you is the following:

- Write a program that will query the Java API for information specified in a ”Queries” section below. You are not allowed to use SQL or a database for this portion of the assignment.
- Model the data of the system (e.g. your model must contain all of the data we supply you with). Begin with an E-R diagram, and from there convert it to a relational

schema. The resulting schema must be in BCNF or 3NF (this shouldn't be too difficult, as the few FD's are pretty clear).

- Write a `schema.sql` file, which when executed will initialize your design in SQL. We will look for more than naive table creation: this means labeling your primary keys, foreign keys, constraints, etc. Also, please use standard SQL: no implementation-specific constructs (e.g. MySQL's `auto_increment` is frowned upon, and we take points off for things that make us frown).
- Write a small program that will output a `data.sql` file, which will provide a means to load the Observatory's data into your database. There are many ways of doing this, such as SQL `INSERT` statements, or a bulk load. We provide support code to remove most of the low-level details of this, you need only supply the logic.
- Finally, write a program to retrieve the same information of part 1, this time using SQL on your schema and not the Java API. Feel the love of gathering data declaratively.

The (pretty ample) handin should include your Java querying program, your E-R diagram, your `schema.sql`, your `data.sql`, the program that generated `data.sql`, and the SQL queries returning the same values as the Java program from part 1.

### 3 Astronomy Primer

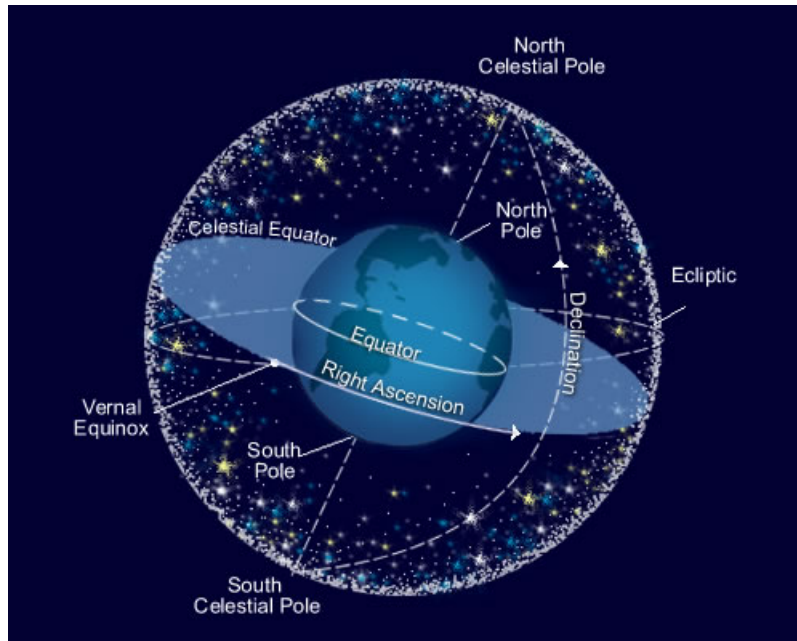
Before we describe the current structure of the data, we will describe what the data *is* with basic astronomy definitions.

#### 3.1 Star Coordinates

Astronomers record the location of stars with a system of *Celestial Coordinates*, these allow astronomers to find stars in the sky they have seen previously.

How to do this is a bit tricky, since the Earth spins continuously, as well as traveling around the Sun. The solution most astronomers use (as well as our observatory) is the *Right Ascension/Declination* coordinate system, commonly abbreviated to RA/Dec.

Imagine the Earth is in the center of a much larger sphere, like the peach pit inside a peach. We then map the celestial bodies like stars, planets, and quasars as points on that outer sphere, much like we plot points on a globe representing Earth.



Like the Equator and Prime Meridian on Earth (the basis for the Latitude/Longitude coordinate system), astronomers have devised similar lines on the imaginary Celestial Sphere, called the Celestial Equator and the Celestial Prime Meridian. An object's *Right Ascension* is the angle measured from the Celestial Prime Meridian, and its *Declination* is the angle from the sphere's Celestial Equator. These are both stored as doubles.

A star's *magnitude* is a measure of its brightness. Generally, magnitude is expressed as one number (such that a lower magnitude means a star is brighter), but for many scientific purposes, this number and how it is obtained isn't specific enough for what they seek to investigate. Therefore, there are many ways recording magnitude more specifically, and the astronomy uses *ugriz* system, which stores magnitude as five floating point numbers ( $u, g, r, i, z$ ).

The exact mechanics of how *ugriz* works isn't important for the assignment;<sup>1</sup> what matters is knowing that it represents brightness as 5 floating point numbers.

## 4 Overview of the Data

The following is a simple overview of the data the observatory kept, and how they kept it. Read it carefully and look for structural elements to incorporate into your design. Any astronomical terms and important equations are explained in the sections that follow:

The Observatory recorded some of its data in a calendar format: every day the astronomer on duty would record the time of the sunrise, the time of the sunset, and record how long daylight lasted. If the day was an equinox, a solstice, aphelion, or perihelion, the

<sup>1</sup>The curious can find more here: [https://secure.lcogt.net/en/user/apickles/dev/STD/SDSS\\_historical.pdf](https://secure.lcogt.net/en/user/apickles/dev/STD/SDSS_historical.pdf)

astronomer would highlight the day with the appropriate color. Finally, the astronomer would note how much of the moon was illuminated.

The Observatory has recorded all their data for 2009 (?). You will be interfacing with the data in this calendar through a Java API.

Meanwhile, employees kept their personal information in a binder near the front desk, that contains: each employee's full name, department, birthday, and date employed. Each employee then has a description of what they do in the company (in their own words), and each time they solicit another donation (since they call for pledges during the day), they note the donor, amount, and time the donation was received next to their name.

Meanwhile, two employees, Eva Lu Ator and Cy D. Fect, were in charge of the actual star catalogue, and each kept a notebook with the following data about each star:

- The star's coordinates (expressed in RA/Dec coordinates).
- The star's magnitude (expressed as  $u, g, r, i, z$  variables).

Note that not all stars in both notebooks were unique. Sometimes stars were counted twice, so somewhere between 5-25% of stars are in both notebooks. It is up to you to test which stars are repeated. Worse, the data for repeat stars doesn't match exactly! Stars that have been recorded twice have each of their attributes differing from each other by at most 4.5% percent.

Take all this data and enter it into database of your design, avoiding redundancies.

## 5 Working on the Project

### 5.1 Getting Started

To get started, copy `/course/cs127/asgn/cs127et1.tgz` into your course directory, and unpack it with `tar -xvzf cs127et1.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 and running your project. To compile, while in that directory type `ant`, and to run, type `ant run`. This automatically includes the support code in your classpath when compiling and running.

The support code is included as a jar in the `lib/` directory, and your code should go in `src/`, using the provided `Cs127Et1Main` class to instantiate it.

### 5.2 Java API

We provide the following classes, which can be instantiated and queried for the appropriate information: `CalendarData`, which has information from the calendar (e.g. sunrises and sunsets), `EvaCatalog` and `CyCatalog`, with the actual star data, and `AstronomerCatalog`, which has all the employee information. These classes contain an abstract representation of

all the data of the Observatory, accessible through `get` methods. See the linked Javadocs on the Assignments page for the exact API.

## 6 Queries

You will need to fetch us the following information, the first time by writing a Java program that queries our API, the second time with SQL queries on your database:

1. Print the Right Ascension and Declination, followed by a newline, of all stars whose whose Right Ascension is less than their Declination.
2. Execute the same as above, but only print stars who also match the predicate where the  $u$  magnitude value is greater than the  $r$  one.
3. Print the same as 2), but include stars who may meet neither criteria but happen to lie above the Celestial Equator.
4. Print the names of the employees who have birthdays before July 1st.
5. Print the names of the top 3 fundraisers.
6. Print the names of the donors from greatest donations to least great donations.

When printing queries, please only print the data your queries return, with each tuple on its own line. This means the following would pass:

```
-15.002 -3  
11.1 4  
4.41 7.64
```

but anything else (usually label or formatting), such as

```
RA: -15.002 Dec: -3  
RA: 11.1 Dec: 4  
RA: 4.41 Dec: 7.64
```

or

```
---  
STAR  
Right Ascension: -15.002  
Declination: -3  
---  
STAR  
Right Ascension: 11.1
```

Declination: 4

---

STAR

Right Acension: 4.41

Declination: 7.64

would not pass.

Please print the results in the order of the queries (e.g. query 1 first, then query 2, ...). Finally, our grader will not consider blank lines, or lines containing SQL Comments (two dashes: --), so feel free to use either separators.

## 7 Graduate Credit

For students seeking Graduate Credit, we'll ask for another handin to this assignment: you must build a tool to generalize the ETL process. Write tool(s) that can perform the following tasks:

1. *Automatically find (likely) conflicting data fields in various tables, given the DDL for some schema.* For example, *first\_name* and *last\_name* in *T1* may conflict or be similar in structure to *name* in *T2*, and therefore be a candidate for unification into one field in a larger table.

Like managing conflicts in a VCS, your program should take a list of table definitions in SQL and return a list of 'conflicts' between tables where it believes data can be unified. This list of conflicts can be expressed as mappings of columns in the tables, such as

```
((name), (first_name, last_name))  
((last_updated), (update_history))  
...
```

The specifics on how you do this is intentionally left vague: we have a few ideas, but want to see yours. If you have any questions on your approach, feel free to e-mail a TA. Note that this shouldn't automatically perform any data unification, just bring attention to potential candidates.

2. *Convert data from a vendor-specific DBMS schema to another.* In theory, this should be pretty simple because SQL is SQL. In reality, it gets a little more complicated, especially with non-standard constructs (take a MySQL schema dependent on `auto_increment` and move it to Postgres, for example) and international support (characters sets, etc.).

List in your README two SQL implementations from the following (MySQL, Postgres, Microsoft SQL Server) and write a program that takes a schema from one as input, and outputs (to the best that it can) the same schema in the other.

As before, we won't tell you how to do this, and we expect some noise and error.

Due to the open-endedness/freedom of the assignment, you should make good use of your README when handing in.

## 8 Handin

We expect the following components to be included in your handin:

- The initial Java querying program, which prints solutions to the queries mentioned above.
- An E-R Diagram of your design.
- A `schema.sql` file that creates the necessary tables of your design.
- A `data.sql` file, containing all the data to fill those tables with the data from the calendar.
- Your program, which uses the various Catalog and Data classes to take the Observatory data and create the `data.sql` file.
- A set of queries that return the same data as your first Java program, using your SQL database rather than the Java API.
- If you're pursuing graduate credit, the two extra handins specified above.

You can handin your project using the script from the directory containing all your files:

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

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