# Building a Bloodbank Database

Nicholas Gordon

University of Memphis

**Abstract** A report on the appropriateness, development, and construction of a database suitable for use by a blood bank. Includes short introduction to databases and competing vendors and web technologies as well. Discussed are the design decisions made, including a discussion of the schema used for the database, as well as the user interface design. Finally, a discussion of the implementation itself is given, including problems faced, analysis of the solution, and future expansions of the solution.

## 1 Introduction

Databases are ubiquitous in computers. They are an extremely powerful tool for organizing data, which is an imperative in computer science; randomness is tantamount to uselessness. Once data is organized, it can be manipulated in powerful ways to derive meaning from data. One useful example of this is a blood bank. In a blood bank, the data are a pool of patients, each with varying blood types, blood-borne diseases, names, addresses, etc. Without organization, the best you could hope to do is keep track of everyone. With organization, you can generate lists of patients to contact for donation, who can donate to some arbitrary patient, or even location-based demographics on frequency of donation, disease distribution, etc.

Now we come to the database itself. We could simply organize these lists by hand, periodically generating all of these metrics, but we can use computers to muscle this data into formats we perfer for us. This is the impetus for this project: generate a database that can be populated with data from a blood bank which will provide to the bank useful meaning from their data. The rest of this report will deal with some knowledge upon which this report is incumbent, such as differing database vendors and web technologies available for interfacing with these databases, the overall design of the database that has been produced, including the structure of the tables and the database itself, and a discussion of the problems faced, the solution presented, and its appropriateness to the problem at hand.

## 2 Related Work

In order to fully understand the decisions made during this project and the design of the database given, some background knowledge needs to be established. Given that a not-narrow audience is likely to be reading a paper such as this, this

explanation is warranted. Databases often are just referred to as "the database" because of their ubiquity, but the truth of the matter is that there are a great many databases that any sysadmin must choose from, each having their own quirks, costs, and intended uses. According to db-engines.com, which uses a variety of metrics to determine database popularity, the five most popular databases in February 2016[1] are, in order:

1. Oracle
2. MySQL
3. Microsoft SQL Server
4. MongoDB
5. PostgreSQL

This list is not linear, however. The top three entries have reasonably close numbers in terms of popularity, but MongoDB and PostgreSQL have only about 30% each of the share of Microsoft SQL Server. Most databases are relational, which means their records are stored in tables or similar, and relate multiple fields together in items called records. These databases are the dominant paradigm, and only MongoDB on this list is not a relational database. MongoDB fits into another paradigm known as document-store database, which effectively are a dictionary that stores complex objects in a key-value system. The differences between these databases largely come down to individual feature support, cost of use, and enterprise-support. Of note is that MySQL, MongoDB, and PostgreSQL are all open-source software and as such are free to use or have a free version available.

Racket is a Scheme derivative, which emphasizes being a platform for extending the language and being a full-spectrum programming language suitable for a broad variety of uses.[2] It includes as built-in packages tools to develop a dynamically-generated website, as well as for interfacing with a large variety of databases from different vendors.

## 3 Design

The database itself is described by a schema, which is essentially some logical design of the database stipulating things such as how the tables are interrelated, what tables have which attributes, and how those attributes are constrained. The schema for the table in this database is as follows:

**Donor**

| Attribute | Constraints |
|---|---|
| id | serial primary key |
| lastname | varchar(255) not null |
| firstname | varchar(255) not null |
| bloodtype | varchar(3) not null |
| address | varchar(255) |
| testsdone | text[][] |
| knowndiseases | text[][] |
| lastdonationdate | date |
| phone | char(14) |

As a demonstration of the database's capability and real-world usefulness, I have elected to build a webpage. This webpage permits a user, here speculated to be bloodbank nurses or doctors, to view donors, and to manipulate this data accordingly. They would be able to view details about a patient they select from a brief view, and then be able to find a list of patients who could donate blood to this user. I've chosen Racket because of the aforementioned tight integration of website devlopment capability and database interaction. These qualities make Racket an ideal platform for implementing this design.

## 4 Analysis

<UNFINISHED>

## 5 Conclusion

<UNFINISHED>

## References

1. Db-engines ranking. Online (February 2016), http://db-engines.com/en/ranking
2. The racket language. Online (March 2016), http://www.racket-lang.org/

# 1 Projected Status in One Month

In one month I hope to have fully generated test data. As it currently stands, the database is populated with data that's about 70% true-to-life; the records are missing addresses, as well as randomly-generated diseases and tests that have been done.

Additionally, in one month I hope to have a website where a user can search for donors based on first name, last name, or ID, which will then show a list of matching results. Provided no issues arise, I would additionally like the website to enable a user to click on a donor and to do a "search for compatible donors" search, that would find a list of all donors who could give blood to the selected donor/patient. I do not anticipate any filtering to be available, i.e. no filtering for donors known to have herpes, HIV, tuberculosis, etc.

# 2 Problems I'm Having

The only issues I'm having are grappling Javascript, Node.js, and this stack I'm choosing to use. The state of the market is that most of the Node.js stacks use MongoDB for their backend, which does not support SQL statements. There exist converters from SQL to MongoDB, but they are simple transliterators and not true converters. As such, I was forced to use Postgres. The interaction between Node.js, the website at large, and Postgres is complicated. Not helping is that I do not have much (read: any at all) experience developing web applications, and as such I am bootstrapping my own knowledge here as well. I do not expect this to be a major stumbling block, though, as what appears to be sufficient documentation exists for me to build a minimally-functioning website.

# March Progress Update

## 1 Progress Since Last Report

Since my last report I changed my decision about web technology to an implementaion in Racket. The rationale for this decision is that Racket provides built-in capabilities for both website production and database interaction. This provides an ideal framework for a project like this. Using Racket, I have constructed a simple website that views all donors fetched from the database. I have laid down a database-website interaction framework that enables future developments. What I have not accomplished is populating randomized disease lists, tests done, addresses. Additionally, the website lacks user search and user match capability.

I have elected to postpone the completion of data population for two reasons:

1. This is a minimal set of data that could exist in a real situation; name, phone, bloodtype.

2. Until I develop search capabilities, the data is unnecessary and thus does not detract from current development.

## 2 Problems I'm Having

I am having issues with developing an effective and aesthetically pleasing display for the donor list. A simple way to accomplish this is in Javascript, but Racket documentation about using Racket to produce Javascript is sparse. Until I get a grasp on the Javascript aspect I cannot make progress, so that is a very high-priority goal. I do not anticipate needing support from teaching faculty to accomplish this.

## 3 What I Plan To Do About It

I am considering a paradigm similar to those of other web stacks such as PHP/Javascript integration, or CGI approaches, which generate the output HTML according to a host language, but embed Javascript for advanced operations like jQuery.