Tech Review

CS 461 Fall 2018

Group 19 BrewHops: Ninkasi Brewing, Automating the brewing process

Brennan Douglas
douglbre@oregonstate.edu

October 25, 2018

Abstract

During the brewing of a batch of beer there are many variables that need to be tested and tracked. This helps determine when the beer is ready and how it compares to other batches. Ninkasi — a brewery in Eugene, Oregon — is using a single large excel spreadsheet to track and store all their information. As they have grown, this has become increasingly unwieldy. There is already the beginning of a solution which includes a small set of applications designed to manage this data automatically while allowing for editing and visualization. This system will manage brewing data and user permissions to reduce the possibility for human error. This document examines different technologies that can be used for the back-end (data side) of this application. The format the data is stored in, how the data is stored, and the archival methods that can be used are covered.

Contents

1 INTRODUCTION

This tech review focuses on the back-end aspects of the application being built for Ninkasi. The sections will build off of the previous based on the conclusion reached for the previous technology. This helps limit the scope of the technologies to choose from, otherwise the number of technologies would be overwhelming. First, the format the data should be stored in needs to be determined. This involves discussions of how structured it needs to be, the convince factor of the different methods, and the support for the style. From there, databases that support that type of format will then be researched and compared to make the best decision. Finally, for a specific database the different archival methods available will be researched to determine which is the best fit for Ninkasi.

2 DATA MODEL

2.1 Introduction

The way that the data is modeled is very important to an application. It defines how the different query behaviors will be implemented and what different aspects of the application will be easier or harder to implment. There are two main categories of data modeling that are revelvant to look at: SQL, and NoSQL. SQL has been the standard for a long time, however with issues of scale at the much larger tech companies NoSQL formats have been created [**?**].

2.2 NoSQL

2.2.1 Summary

There are many different types of NoSQL data models. This is because NoSQL is equivalent to the phrase non-relational data bases, ”not SQL”. NoSQL models are primary designed around being very fast at scale for their particular use case. The various different types include: key-value pairs, document, and graph [**?**]. The most common type is the document class of NoSQL models. In these, entries are just JSON objects, meaning they can be easily spread out across multiple different machines to horizontally scale the database. Out of all the different forms of the NoSQL data models this class, the document style, would be the most appropriate for our use case. This is because we have a few main entities that could be represented as separate documents. Each of these documents would then contain very piece of information they need, rather than being normalized across many different tables.

2.2.2 Pros

- Horizontally scalable
- Quick development
- Simple queries
- No dedicated database server required (depending on database engine)

2.2.3 Cons

- Deep models
- Data replication
- Large query responses

2.3 SQL

2.3.1 Summary

As mentioned previously SQL is the oldest of the database models. It has been around for over 40 years [**?**]. It allows the data to be normalized across many different tables reducing data replication. It also has far more support than other technologies due to its ubiquity. It would also allow for more precise query calls to be made retrieving only the information that is required from the database. This last point is key as our application is targeting mobile devices where networking are a limiting factor.

2.3.2 Pros

- No data replication
- Precise queries
- Ubiquitous support
- Data integreity

2.3.3 Cons

- Structured (sometimes overly so)
- Dedicated database server required
- Slow development

2.4 Conclusion

In conclusion, simply due to the support available, SQL is the best choice. Ninkasi should have the highest number of options available to them when it comes to choosing their environment, this SQL provides. A large factor is also the data integrity. These SQL engines have been being used for decades, they are well and thoroughly tested. Ninkasi can be assured that their data will not be lost or corrupted.

3 Databases

3.1 Introduction

Now that the data model has been chosen, different database engines can be looked into. There are a few different options that are standard when wanting to work with SQL: PostgreSQL, MySQL, SQL Server. This vary very little in their syntax and functions as they have all been copying each other for a long time. There are categories which they differ in, most notably if they are free or not. This is important for small projects as it is hard to justify spending money on something that has very few benefits over another. Especially when it is well supported and free to use.

3.2 PostgreSQL

3.2.1 Summary

PostgreSQL is a powerful open source SQL database engine [**?**]. It has the same query syntax as the other open source SQL database engines out there. However, it also has strong support for some very NoSQL like features. This allows the advantages of SQL and NoSQL to be combined from the model viewpoint. Though, it cannot capture the scalability advantages of the different NoSQL systems as it is NoSQL within a SQL database.

3.2.2 Pros

- Open source
- Free
- NoSQL features
- Runs on almost anything [**?**]

3.2.3 Cons

- Not the most popular
- Not the same level of support as MySQL

3.3 MySQL

3.3.1 Summary

MySQL is another powerful open source database management system. It is more popular that PostgreSQL, yet actually has fewer features [**?**]. MySQL stands in a funny spot within the open source line up, it has been acquired by Oracle [**?**]. This makes some in the community unsure of its future longevity as a free and open piece of software. This is due to it directly competing with one of Oracle's main paid offerings. However, MySQL is often the go to open source SQL database management system, especially for small projects.

3.3.2 Pros

- Popular
- Highest level of third-party support
- Open source
- Free

3.3.3 Cons

- Not as many features as PostgreSQL
- Rudimentary NoSQL support

3.4 SQL Server

3.4.1 Summary

SQL Server is Microsoft's SQL database management system. It is a commercial grade application, and comes with the price to prove it. It does provide more advanced functionality than either MySQL or PostgreSQL, however these are features that are not needed by the application [**?**]. There is no strong point for a discussion with Ninkasi to even be approached about buying a license for SQL Server. Also, MySQL's and PostgreSQL's syntax is identical in almost all cases, this is not the true for SQL Server as it diverges in some subtle locations.

3.4.2 Pros

- Advanced functions
- Best performance

3.4.3 Cons

- Cost
- Commercial
- Different syntax

3.5 Conclusion

In conclusion, PostgreSQL provides the application with the most flexibility and support while remaining free. It will run on anything that Ninkasi could ever want, and has a large community of support and third-party interfaces ready to be used. SQL Server is too much for this project as it does not have the complexity or amount of data where its features and performance benefit over MySQL or PostgreSQL. It is also important to keep the cost of the system to a minimum for Ninkasi as this is replacing a currently free process.

4 Archival Process

4.1 Introduction

It is important to make sure that Ninkasi never loses any of the data they store in the application. So, a database archival process will be implmented. This allows them to restore or role back their data to a previous time if they ever need to. "Database archiving is the process of intelligently moving historical data from production environments to an archive environment, freeing up IT storage space and other resources" [**?**]. As we have chosen PostrgreSQL as our database management system only archival solutions for that platform will be focused on. Key factors that are under consideration for an archival process are: data integrity, consistent archiving, and minimized storage space.

4.2 PostgreSQL WAL

4.2.1 Summary

PostgreSQL's write ahead log, WAL, is the log file where it stores every operation done to the database along with the time at which it happened [**?**]. This file was originally designed, and still used for, crash recovery where some of the command had yet to be completed at time of failure. PostgreSQL simply rolls back to the last safe state and then replays the executions from the WAL. The WAL is simply just a file that is stored on the file system, this means that it can be copied to a safe archival location. If this is done on a regular process the entire database can be reconstructed from these files. It provides another feature hidden in its time stamps on the queries, it allows for point-in-time recovery [**?**]. This means the database can be reconstructed up until any point in time the user would like, with no extra storage cost associated.

4.2.2 Pros

- Continuous archiving
- Point in time archiving
- First-party support
- Assured data integrity

4.2.3 Cons

- Must manually create scripts to copy the files
- Special configuration setting must be turned on
- Data isn't available until a database is built from the files

4.3 Manual Archival Scripts

4.3.1 Summary

Manual archival scripts refers to the process of manually writing archival queries that transfer specific data to the archive location. This would require a lot of manual work to figure out what needs to be saved, the difference between the current database and the archive database, and the expense cost of needing to run this while the database is trying to function for the application. This is inherently error prone as it is only supported by the developers of the application. Therefore, it would need to be updated with any updates to the database. This provides another point of failure.

4.3.2 Pros

- Only copies and stores data that cannot be replaced
- Data will be stored in another database that can be queried

4.3.3 Cons

- Higher potential for loss of data due to human error
- Manually intensive
- Not maintainable

4.4 Conclusion

The choice is pretty simple given that there are not many options, PostgreSQL's WAL archival process. This provides the most features right out of the box and will require the least manual work to getting moving. It also has the promise of strong data integrity which is a key aspect of archiving.