**Portfolio Part 5: Final Report**

**T00155775**

**Joe O Flaherty**

**BSc (Hons) Computing with Multimedia**

# Introduction

The databases used to complete this portfolio have been a complete change from anything encountered previously. Whether Redis and Neo4J should even be referred to as databases in the traditional sense of a database is debateable. Installing, populating and interacting with these effectively has been difficult in some cases and extremely challenging and frustrating in others. Much of the difficulty encountered with the databases initially was attributable to lack of familiarity and in each case given a little time the basic operations proved quite manageable. There is a learning curve but not one that is insurmountable, the more complex operations have been a slightly different matter. The most immediate challenge with each implementation was installation and set-up which often posed problems, as the book we have been using states, *“Installing the databases in this book is sometimes easy, sometimes challenging, and sometimes downright ugly”.* (Redmond & Wilson, 2012) There is also a noticeable difference between installation on a Windows based system and Linux based systems, with far fewer problems encountered when using Linux, stemming from the fact that most of the databases are open source and were mainly designed with \*nix users in mind. These databases are also designed to be distributed across multiple machines/nodes with data often shared (sharded) or copied to different nodes, a concept that takes time to get to grips with coming from the much more highly structured SQL type databases we are used to.

# CRUD

The easiest way to give an overview of experiences with the databases is to deal with how the CRUD operations compared to each other. The most immediate observation is that the products that come with a GUI interface were the easiest to work with, CouchDB through its Futon interface being by far the most user friendly. What can be tricky actions on other databases (inserting images/attachments) can be completed simply via the interface with a few button clicks. Importantly from a user view point what to select and click is clearly evident and very intuitive; select the upload attachment option and browse to the required file. Similarly the GUI provided by Neo4J at least provides the ability to get a graph up and running quickly and makes the process of building relationships less daunting as you have a visual point of reference. This feature was found to be invaluable when performing operations against Neo4J as the structure is so alien to anything previously used, really Neo4J is more a store of interconnectivity (relationships) than raw data. MongoDB the second database we investigated is like CouchDB a document database or probably more specifically a document- oriented database that stores documents in JSON format and allows for nesting of documents. MongoDB and CouchDB differ in that MongoDB allows for ad-hoc querying of the database whereas CouchDB does not, indexed views are used instead. The use of id and ‘rev’ numbers by CouchDB was interesting as a safety and history tracking measure, but also a potential source of problems if id numbers are not recorded effectively. The issue of id numbers only become relevant if using REST to work with CouchDB when an id number needs to be attached to the tail of a curl request. Both document databases also differ in their intended usage, MongoDB is designed to handle vast amounts of data while providing easy accessibility, CouchDB is designed with the web and web based data in mind. CouchDB is also much more highly available than all the other databases encountered and provides for availability even when connectivity is minimal.

JSON as a format along with the ability to interact with the databases using REST was a common theme across 3 of the databases used the exception being Redis. Performing the basic CRUD operations against MongoDB was initially difficult because they were performed via a shell and the lack of accessibility a GUI provides was a little difficult initially. However after spending time with MongoDB it turned out to be very accessible and very logical in how commands are structured. The main problem is and was forgetting commas, opening and closing braces and quotation marks. Commands in Mongo read in a very logical way making them easily understood and can be recalled quickly without too much need to reference documentation. The general format is a database has a collection(s) and you want to perform operations against the collection(s), these operations are extremely well named e.g. .find ( ), .insert ( ) and .update ( ) and therefore logical, this made MongoDB very useable at its most basic level. While CouchDB and MongoDB are new technologies and differ totally from traditional RDBMS they still seemed familiar in contrast to Redis and Neo4J which were totally alien and a little difficult to comprehend initially. Redis stores key-value pairs at its basic level but also provides for more advanced structures, lists, sets etc. and is extremely difficult to classify, *“It can be a bit difficult to classify exactly what Redis is. At a basic level, it’s a key-value store, of course, but that simple label doesn’t really do it justice.”* (Redmond & Wilson, 2012) *.*

Regarding CRUD operations Redis is by some distance the easiest of the databases to use and interact with especially considering there is no GUI. Redis stores keys and values so the commands are all about providing a key and giving it a value, CRUD operations are so easy only 3 commands are required GET, SET and DEL; you SET a key and a value, GET a value using its key, DEL a key and update by using SET to give your key a different value. Neo4J was personally found to be the most difficult of the ‘databases’ to get to grips with firstly because it is not really a database, it is a graph. CRUD operations performed were minimal simply setting up nodes and relationships using the Neo4J GUI. This was frustrating in the beginning because the documentation wasn’t properly consulted and mistakes were made in the creation of relationships, Neo4J is described as whiteboard friendly and this is a very accurate description. Once the graph to be implemented was designed on paper creating it became a trivial operation, this is also where the triviality ended. More complex operations on Neo4J require use of either Gremlin, a Groovy based graph traversal language, Cypher, Neo4J’s proprietary query language which is loosely based on SQL or via REST, all of which come with a steep learning curve and were difficult to work with. Gremlin in particular proved frustrating even down to its different naming convention, a node becomes a vertex and a relationship is an edge. Gremlin also requires a graph to be specifically saved as an XML document, failure to perform a save resulted in multiple losses of graphs while working on the portfolio document. Mastery of these languages to use Neo4J effectively for its primary purpose, extremely fast traversal of related nodes to find connected, is vital. This quick traversal makes Neo4J particularly useful as a data-store for social networking sites e.g. Facebook. Searching a Neo4J graph using Gremlin was by far the most difficult process of the entire portfolio, link-walking while very logical was because of lack of experience very difficult to implement it was also difficult to see the real power of link walking on such a small scale.

# Map-Reduce

Another new concept encountered while working with these databases is map reduce which is used to break queries (maps) into a series of smaller queries thus improving search performance particularly across vast amounts of data. CouchDB and MongoDB both allow for map reduce functions with CouchDB being the best introduction to the concept in my opinion. What makes CouchDB a better introduction is the Futon interface where you can really see what is happening by first creating a view/map and examining the result. This can then be reduced by selecting a check box and results again examined. Experimentation is made reasonably straight-forward via the interface with results and mistakes more immediately visible than with MongoDB. Map reduce is an extremely important concept in the world of NoSQL databases, we have really only had an introduction to the idea and I found the lack of expertise frustrating when attempting portfolio questions. Competency in applying map reduce effectively would require considerable study and practical experience.

# Observations

There is a learning curve attached to all the databases worked with for this portfolio, but the basic operations on each of them were easily accomplished. The steepest curves were with the concepts behind the databases as well as the new languages encountered and smaller things like installations on Linux and Windows that didn’t always go as planned. Concepts that are difficult to appreciate without using the systems on a larger scale were sharding, replication and the whole area of master and slave servers especially the process of deciding a master. Splitting data across nodes and/or copying data across multiple nodes is still a tough concept to grasp, then there are the different ways the databases adhere to the CAP theorem especially when adding the idea that data may not be totally consistent! I have already mentioned Gremlin, Cypher, REST – particularly with Neo4J but there was also curl to get to grips with. Except for Gremlin which is just difficult to use the others could probably be mastered given enough time but over a limited period were difficult. Installation of drivers for programming languages that turned out to be no good or failed to install was another source of frustration. On more than one occasion a package was installed and quite a bit of work completed before discovering that a vital piece of functionality was missing. There were a number of common themes encountered, Ruby as a language, JSON, JavaScript (especially useful to populate a MongoDB collection quickly) and REST all of which having a familiarity with would make working with the databases considerably easier. The lack of familiarity caused a lot of frustration when working through the portfolio questions as everything took longer to complete than first estimated. I would recommend using all the databases on a Linux based system as there is definitely far more support available and any installations particularly MongoDB were more straight-forward than the Windows equivalent. I chose Python as the main programming language I used over the course of the portfolio and again Linux made this less difficult as Python comes packaged with Ubuntu (the version I use) removing the need for adding it to paths and learning to use ‘pip’ for module installation(although I did have to eventually). Which database would I use is a tricky question to answer as it would be so dependent on the type and usage of data required, from the experiences working on the portfolio it would be MongoDB (with a GUI). What would I do to make use of NoSQL databases a little easier? Learn to use REST and curl to a high level and acquire a working knowledge of Ruby and JavaScript.

# References

Redmond, E. & Wilson, J. R., 2012. *Seven Databases in Seven Weeks - A Guide to Modern Databases and the NoSQL Movement.* 1st ed. Dallas , Texas: The Pragmatic Bookshelf.