Comparing Storage Technologies

There are many ways of representing and storing data, some highlighted in this course being: JSON, XML, relational systems like SQL, NOSQL. All of these has very useful advantages and weaknesses like all forms of technology. In this essay I will be covering the advantages and disadvantages of these technologies and comparing their weaknesses to each other.

First there's JSON, standing for JavaScript Object Notation, is based off a subset of the programming language known as JavaScript. JSON is in a text format that is completely independent of normal languages, but it makes use of a programming style very similar to the C-Family of programming languages. JSON is a very simple format that can only store Boolean, text, numbers (including integers and floating point), other JSON objects, arrays of JSON values and null as well. Due to the small number of things a JSON file can store it can be fairly limited, however this does make JSON very easy to read and write for humans. It is also quite easy to parse and generate JSON values with a machine. Another problem to note is that since JSON doesn't differentiate between integer and floating point then it could possibly cause some compatibility issues when a program is parsing them. Overall, JSON is a very simple and easy to use storage type however can be quite limiting due to its simple nature.

Next there is XML, standing for Extensible Markup Language, is a very important and widespread file format (even the program being used to write this essay will store it with XML). XML uses a tag-based system with <tag> and </tag> to delimit things. As long as things are inside elements there is no need for quotes, and it can have nested elements allowing for a lot more complex structures to be represented. XML is still fairly human readable (but without indentation it can be quite difficult to read) though not nearly as much as JSON is, however it is a lot more flexible and can store a larger variety of data. Overall, XML is very useful and it's good there is a widespread standardised file format that is simple to use.

That was semi structured data model now we are going to discuss a structured data model. The entity relationship model is a very useful model. Databases can store many, many values and are very organised and easy for humans to understand. They are widely used in many circumstances to store gigantic pieces of data. However, it can be quite a lot of work to actually implement them, as this model in very inflexible. It allows for a lot of data types to be stored but for modelling things it has a strict set of rules that if deviated from breaks the system entirely. The reason it can be hard to model something with a database is because there are many steps when implementing them into this format. They main reason it can be hard is because of the relationships between entities, many of the same entity type cannot be related to many of another entity without a strange solution of inserting another entity type that doesn't mean anything just to get the model working. However due to the rigid nature it does mean searching for data and generally manipulating the data is very easy to do. SQL is very useful as it does not need to include how to find the data just what the

data you want to find is. Databases make use of ACID (Atomicity, Consistency, Isolation, Durability) which is a system to ensure that any interaction with the database is perfect and if something does go wrong it doesn't affect the data, however, this can be very difficult to achieve as complex operations and transactions can happen and errors could occur at many points so it's very hard to plan for, for more complex database systems they may be many constraints from the schema that need to be checked, multiple transactions at once and databases can be on multiple drives/computers. Overall, databases are incredibly useful for many situations but can be bogged down by their rigidity and may not be appropriate for all situations to be modelled.

Luckily many people agree that databases aren't the best data storage technology and so the NOSQL movement was created. Standing for Not Only SQL the NOSQL movement sought out different ways of modelling and storing data structures to accommodate for the weaknesses of databases. There are a few different types of NOSQL however, those being key value (MongoDB is of this type), column based, document database and graph database (Neo4J is of this type). Instead of ACID, NOSQL makes use of BASE, where instead of consistency availability is preferred. Generally, NOSQL systems take after the CAP theorum in which there are three aspects, and each system can only have two. These aspects being consistency, availability and partition tolerance.

Obviously, there are quite a lot of difference between relational and NOSQL systems. Relational systems have very structured and organised data but NOSQL has no declarative querying language and no predefined schema. Relational have data and relationships stored in separate entities but NOSQL is usually unstructured or semi structured.

MongoDB is an example of the NOSQL movement and works on the basis of Collections (similar to tables) and Documents (similar to records). The database just acts as a storage space for these collections and documents. MongoDB uses CRUD queries to search and edit collections and it supports many programming languages.

Neo4J is a type of NOSQL known as graphical database, which is basically a direct translation of an ERD, as in if you were to make a model diagram to show how you wanted a system implemented then Neo4J would look exactly like that diagram. Neo4J like other NOSQL systems has a very flexible schema but does let you provide schema constraints and even ACID transactions. Instead of tables there are graphs and instead of records there are nodes. Neo4J is very useful for visualising data and seeing how certain objects relate, a lot like UML in programming but with actual functionality.

Overall, the NOSQL model is very useful and helps make up for the weaknesses of the relational model. However, NOSQL isn't appropriate in all situations just like relational. Sometimes relational works just fine and the scenario requires those strict rules and thus NOSQL wouldn't be appropriate.

In summary, all these data structures and models are all equally valid and each are vital in many scenarios. None are completely useless, and you could theoretically interchange them. It's just some are way better for certain scenarios than others.

Most of the information in this essay was from common knowledge or the lectures from the CS1003 module.