

UWTSD

DVLA: MSc Software Engineering (Web Development)

ACCA7002

Data storage investigation

Callan Birch-Williams, 2222124

Contents

Introduction	3
Related Documents	3
NoSQL Databases	3
Key-Value Stores	4
Document Stores	5
Wide-Column Stores	6
Graph Databases	7
NoSQL vs Relational Databases	9
NoSQL Databases	9
Relational Databases	9
Conclusion	10
Bibliography	10
Figure 1: Telephone directory example of a key-value store [7]	4
Figure 2: Customer document example of a document store [7] [11]	5
Figure 3: Company information wide-column store example [7]	7
Figure 4: Example social network graph schema [21]	8

Introduction

The purpose of this document is to outline and evaluate some of the different types of NoSQL databases and their common use-cases, for the consideration of Abertawe Entertainment. An additional comparison will be drawn between traditional relational databases and NoSQL databases.

Related Documents

In addition to the comparative research outlined in this document, mock database implementations of the two approaches have been created to provide Abertawe Entertainment with the means for analysis and testing when determining the appropriate solution for their application needs.

A traditional relational database has been implemented using PostgreSQL [1], whereas a document store (discussed later in this document) has been implemented using MongoDB [2] for the consideration of a NoSQL database. The files required to instantiate these databases have been submitted in conjunction with this document. While both introducing some language specific features, they can easily be adapted to Abertawe Entertainments technological choices and more so function as a demonstration of different data models. Both databases have been populated with mock data generated by Mockaroo [3].

The document store solution implements the document reference pattern to associate users with owned assets [4]. This was deemed more appropriate than the embedded document pattern [5] due to the logical application need to store assets prior to user acquisition.

When considering the storage of the actual digital assets within the database, there are various options. Both PostgreSQL and MongoDB have systems capable of storing large data (BLOBs). Alternatively, it may be deemed appropriate to store digital assets in a filesystem [6] with references or utilise a separate application layer to handle the streaming/distribution of assets to users, using the user data within the database layer as confirmation of ownership.

NoSQL Databases

NoSQL databases are databases that store data differently than traditional relational tables. NoSQL databases exist in a variety of types identified by their data model. The main types (The subjects of this document) are key-value stores, document stores, wide-column stores, and graph databases. They are characterised by their flexible schemas and scalability [7].

Key-Value Stores

A key-value store is a type of NoSQL database where data is stored in the form of key-value pairs, where the key is a unique identifier, and the value is the data associated with it. In most key-value stores, the key must be a string. The corresponding data is usually some form of primitive datatype. A key-value store replaces the need for a strict schema and can handle a large amount of data.

Doyle Inc	726 478 2399
Dietrich, Stokes and McGlynn	641 850 8987
Powlowski LLC	683 110 5530
Hartmann, Lehner and Ebert	172 855 2333
Johnston and Sons	426 560 7634

Figure 1: Telephone directory example of a key-value store [3].

Key-value stores are comparable to a relational database within which each table has only two columns. Some key-value stores support basic join operations; however, the frequent use of joins is often an indication that a key-value store is not the correct approach for a given problem.

Some of the benefits of key-value stores when compared with relational databases can be defined as follows:

- **CRUD performance:** Key-value stores are optimal for fast read and write operations, providing a high-performance alternative to traditional relational database systems in cases that require low latency data access and high throughput [8].
- **Scalability:** Key-value stores are horizontally scalable, meaning it is usually easy to accommodate an increase in the amount of data.
- **Flexibility:** As key-value stores are defined so simply and do not enforce a strict schema, they can be extended and modified in numerous ways.

There are various use-cases where a key-value store may be the correct approach to a given problem:

- Storing and retrieving large amounts of simple data quickly, for example in a caching system or real-time applications such as online gaming, finance, etc [9].
- Storing data that does not require complex queries or joins.
- Storing data that does not have a fixed schema and is anticipated to change over time.

Document Stores

A document store is a type of NoSQL database that stores data in the form of semi-structured documents, such as JSON and XML. It can be considered a superset of key-value stores as it also uses unique keys to encapsulate key-value pairs within documents. Unlike key-value stores, the data associated with a key can be a nested document.

Each document within a document store typically contains information related to a single unit (E.g., Customer, Asset). As a result, documents are comparable to records in relational databases, however they are more flexible as they are schema-less and can contain dissimilar data [10].

```
{
  "_id": 1,
  "username": "fb1own0",
  "password": "q8Jfpu1u",
  "telephone": "996 161 1714",
  "assets": [
    {
      "title": "Perfect",
      "release_date": "2022-05-17",
      "price_pence": 735,
      "genres": [
        "Comedy",
        "Romance"
      ]
    },
    {
      "title": "I Want to Be a Soldier",
      "release_date": "2022-08-14",
      "price_pence": 80,
      "genre": "Western"
    }
  ]
}
```

Figure 2: Customer document example of a document store [3] [11].

Some of the benefits of document stores when compared with relational databases can be defined as follows:

- **Flexibility:** Document stores do not enforce a fixed data model and as a result are very flexible. Additionally, some document stores support the creation of schema validation rules where appropriate.
- **CRUD performance:** Document stores have been shown to have better performance in CRUD operations on basic schema, especially when working on large data with concurrent access [12]. This can be attributed to indexing capabilities.

- Ease of use: Working with data in readable documents can be considered easier to understand and more intuitive to work with. Developers can interact directly with data without the need to use Object–relational mapping. Considering the example shown in Figure 2, getting information about an individual customer requires one query and no joins, whereas a traditional normalised relational database would require the joining of tables used to represent the one-to-many relationships.

Document stores are an extremely flexible approach to database design and can be adapted to various applications. To that end, it can be used to implement the various databases discussed in this document:

- Key-value pairs can be modelled with fields and values within a document.
- Graph nodes (discussed in detail later in this document) can be modelled as documents. Edges can be modelled using references between documents.
- Relational data can be modelled differently (and sometimes more intuitively) by storing related data in the same document using nested documents and arrays. Alternatively, references between documents can be used to connect related data. On the other hand, replacing join operations with aggregate data encourages data redundancy [13] and is often an indication that a traditional relational approach could be more suitable.

Despite the flexibility of document stores, there are obvious limitations when considering a use case that requires various complex joins and transactions. Additionally, for applications that are required to ensure data integrity, a different database approach that satisfies ACID [14] would be suitable.

Wide-Column Stores

The concept of a wide-column store was first described by Google as “Bigtable” [15], a distributed storage system designed to support performant read and write operations for a very large amount of data. It is described as richer than key-value stores with support for sparse semi-structured data.

In a wide-column store, data is organised into tables, comparable with traditional relational databases. However, each row can contain a different set of columns (Additionally, one column may be present in multiple rows). Each row can be considered to represent an individual record, while the columns represent attributes of said record.

A column in a wide-column store is logically comparable to a key-value pair from key-value stores, though in practice is often a tuple that additionally contains a timestamp. A super-column is the same as a column in that it is a key-value pair, however the data associated with its key is instead a list of columns. Both columns and super-columns are grouped together in column families, a structure that contains a collection of rows. Each row contains a key (The row key) and a collection of columns. A super-column family therefore contains a collection of rows that contain a collection of super-columns. Finally, a keyspace is the outer most grouping of data, and contains all column families. An important distinction to make is that unlike in a traditional relational database, there is no relationship between the column families within a keyspace and therefore they cannot be joined [16] [17].

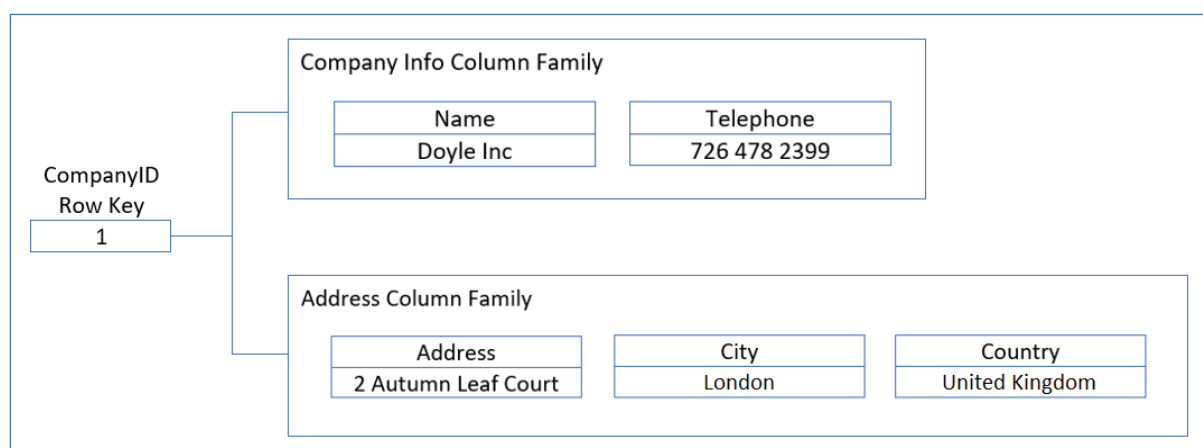


Figure 3: Company information wide-column store example [3].

Wide-column stores solve the issue of storing sparse data efficiently by not requiring fields to always be present (Only the columns that have data are stored), preserving storage space [18]. In a traditional relational database, this would result in space wasted on specifying empty values.

Wide-column stores are ideal for use cases where a large dataset is required that can be distributed across multiple nodes, specifically where attributes are not guaranteed to be the same for every row [19].

Graph Databases

Graph databases are designed to map the relationship between data. In a graph database, connections between data are stored directly within the database. In contrast, connections in a traditional relational database are implied by the data within them. Multiple table joins are required in a relational database whereas a graph database overcomes this overhead. This is achieved by associating each node (Data entities) with edges to adjacent nodes, resulting in no index lookups. As a result of this, the overhead associated with finding relations between data does not increase as the amount of data increases.

Unlike a traditional relational database and various NoSQL databases, the wide use of graph databases has yet to establish a standard language for traversal and insertion. As a result, implementations and frameworks vary significantly [20].

Not many real-world applications are applicable to a pure graph database approach (Though an argument can be made that given the social nature of modern technology that use cases will only increase). As a result, graph databases are usually used in conjunction with other databases such as key-value stores, document stores, as well as traditional relational databases.

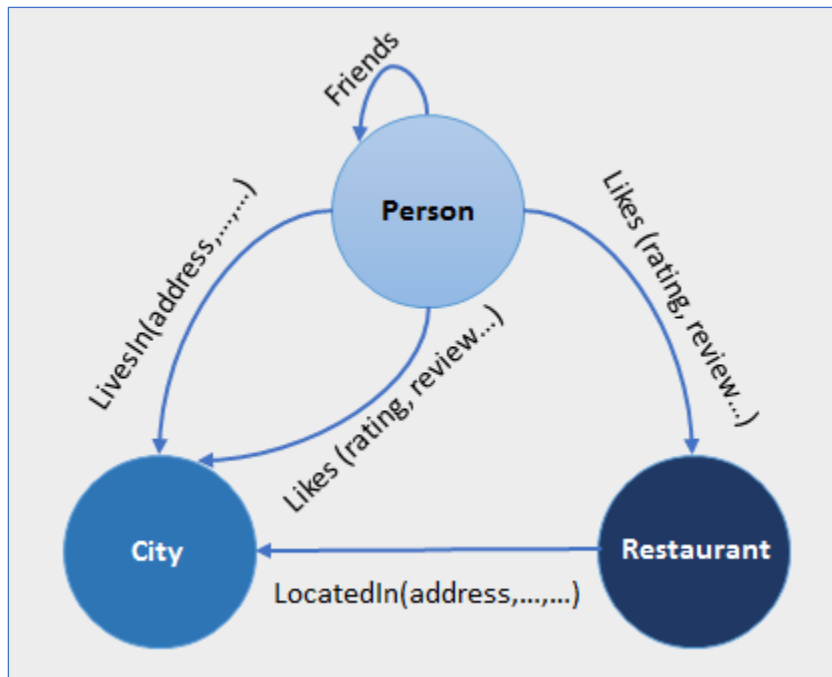


Figure 4: Example social network graph schema [21].

Some of the benefits of graph databases are defined as follows:

- Reliability: Unlike many NoSQL databases, graph databases are fully ACID compliant [14].
- Relational performance: As connected data is locally indexed, finding relationships between data is extremely fast.

Some of the suitable graph databases use cases are defined as follows:

- Recommendation engines
- Social networks
- Fraud detection: Graph databases can be used to process financial transactions in real time. With fast relational queries, graph databases can be harnessed to detect, for example, purchases made using information associated with previous fraud cases. This concept can be applied to detecting various other relationship patterns such as multiple people sharing the same IP address, while residing at different physical locations [22].
- Network/Infrastructure mapping: By mapping relationships between infrastructure components, analysis can be conducted on latency, bottlenecks, points of failure, etc. [23].

NoSQL vs Relational Databases

NoSQL Databases

NoSQL databases are characterised by their scalability, performance, flexibility, and ease of use. Traditional relational databases are characterised by their strict tabular data model, focus on avoidance of duplicate data and expensive vertical scaling. Most NoSQL databases compromise consistency (when considering the CAP theorem) [24] [25], to provide an always available, performant and partition tolerant alternative to a relational database.

Applications at early stages of development may benefit from the flexible data models offered by NoSQL databases. Flexible schemas accommodate databases changes as application requirements are established. Iterative development is made possible whereas a traditional relational database would require the revaluation of rigid schema and data migration.

Most SQL databases can only be scaled vertically by increasing the hardware capacity of a single server. NoSQL databases support horizontal scaling in the form of sharding [26]. Scaling horizontally is not significantly affected by hardware limitations as less powerful server machines can be combined to offer higher levels of scalability [27]. This can be appropriate for use cases where there is a requirement to be able to respond to change or application growth is expected.

As NoSQL databases are designed to handle big data, additional considerations are not required when handling web-scale applications, whereas they are with traditional relational databases. Additionally, the scale-out approach NoSQL databases take allow database structure changes to be made without downtime.

In performance critical applications with high throughput, NoSQL should be considered for it faster queries. Data in traditional relational databases is usually normalised. As a result, expensive join operations are required to retrieve data from multiple tables. Data in NoSQL databases are typically stored in a manner than is optimised for basic queries [28] [29].

The data models of many NoSQL databases can be mapped directly to various programming languages. This allows developers to handle data in application code in the same way that it is stored. When compared to a traditional relational database, the removal of a dependency on ORM (Object-Relational Mapping) results in less mapping code and faster development time.

Due to the flexibility of NoSQL data models, developers are usually responsible for the structure of they data. This aligns with modern agile software development practices [30] of iterative change and frequent deployment. Traditional relational databases would require the migration of data from the old data model to the new, hampering development.

Relational Databases

Traditional relational databases adhere to the ACID model (Atomicity, Consistency, Isolation and Durability) [14], guaranteeing the completion of transactions. NoSQL databases on the other hand are generally not ACID-compliant, or only at the document level. This is an important consideration for use cases where ACID is required across multiple records.

NoSQL databases are broadly optimised for query performance and not for avoiding data duplication. As a result, a NoSQL database can be comparatively larger than its relational counterpart. While storage concerns are becoming less common, specific use cases may consider this an important difference.

Traditional relational databases are generally more appropriate in applications where frequent, complex join operations or complex analytical queries are expected. NoSQL has no true implementation of this, but rather achieves the functionality of joins through denormalization [29].

A relational database is more suitable when the stored data is structured, NoSQL databases are designed to be flexible and support ever-changing data. A traditional relational database ensures high data integrity with features such as constraints and triggers, which are often not found in NoSQL databases [31].

Another common issue with NoSQL databases is a lack of encryption support for data, weak client-server authentication and vulnerability to various forms of attacks [31].

All relational databases are based on SQL. While different vendors impose various SQL dialects a knowledge of the SQL standard allows a developer to work on any relational database without much difficulty. NoSQL databases on the other hand have no common format to query data or a common query language. As a result, they require specialist knowledge. This can be a minor consideration when deciding which database type to use.

Conclusion

This document has outlined the concept of a NoSQL database, as well as its most notable types. Comparisons have been drawn between these types with respect to each other, in addition to traditional relational databases. Different considerations related to the suitable use cases for both NoSQL and traditional relational databases have been discussed. In some applications, the most suitable database architecture is one that takes advantage of both NoSQL and traditional relational databases. The terminology “polyglot persistence” is used to describe the idea that the data storage technology used to create an application should be varied, to take advantage of the fact that different languages are suitable for approaching different problems [32].

Bibliography

- [1] “PostgreSQL: The World's Most Advanced Open Source Relational Database,” The PostgreSQL Global Development Group, [Online]. Available: <https://www.postgresql.org/>. [Accessed 01 2023].
- [2] MongoDB, Inc., “MongoDB Cloud Services,” [Online]. Available: <https://www.mongodb.com/cloud>. [Accessed 01 2023].

- [3] Mockaroo, LLC., "Mockaroo - Random Data Generator and API Mocking Tool...", [Online]. Available: <https://www.mockaroo.com/>. [Accessed 01 2023].
- [4] MongoDB, inc., "Model One-to-Many Relationships with Document References," [Online]. Available: <https://www.mongodb.com/docs/manual/tutorial/model-referenced-one-to-many-relationships-between-documents/>. [Accessed 01 2023].
- [5] MongoDB, inc., "Model One-to-Many Relationships with Embedded Documents," [Online]. Available: <https://www.mongodb.com/docs/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/>. [Accessed 01 2023].
- [6] J. Gray, To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem, 2006.
- [7] MongoDB, Inc., "What is NoSQL? NoSQL Databases Explained," [Online]. Available: <https://www.mongodb.com/nosql-explained>. [Accessed 01 2023].
- [8] M. Seeger and S. Ultra-Large-Sites, Key-value stores: a practical overview, Stuttgart: Computer Science and Media, 2009.
- [9] MongoDB, Inc., "Key-Value Store Advantages," [Online]. Available: <https://www.mongodb.com/databases/key-value-database>. [Accessed 01 2023].
- [10] V. Major, "Manoj, V. "Comparative study of nosql document, column store databases and evaluation of cassandra.,," *International Journal of Database Management Systems*, vol. 6, no. 4, 2014.
- [11] R. Nurik, "SlidesCodeHighlighter," [Online]. Available: <https://romannurik.github.io/SlidesCodeHighlighter/>. [Accessed 01 2023].
- [12] R. Deari, X. Zenuni, J. Ajdari, F. Ismaili and B. Raufi, "Analysis and comparison of document-based databases with sql relational databases: Mongodb vs mysql.," *Proceedings of the International Conference on Information Technologies*, 2018.
- [13] R. Deari, X. Zenuni, J. Ajdari, F. Ismaili and B. Raufi, "ANALYSIS AND COMPARISON OF DOCUMENT-BASED DATABASES WITH SQL RELATIONAL DATABASES: MONGODB VS MYSQL," *Proceedings of the International Conference on Information Technologies (InfoTech-2018)* 20-21 September 2018, Bulgaria.
- [14] N. Lang, "Database Basics: ACID Transactions," 02 07 2022. [Online]. Available: <https://towardsdatascience.com/database-basics-acid-transactions-bf4d38bd8e26>. [Accessed 01 2023].
- [15] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *ACM Transactions on Computer Systems*, vol. 26, no. 2, pp. 1–26, 2008.
- [16] S. Arin, "WTF is a SuperColumn? An Intro to the Cassandra Data Model," 2012. [Online]. Available: <https://arin.s3.amazonaws.com/pub/docs/WTF-is-a-SuperColumn.pdf>.
- [17] H. Lee, Schema design for NoSQL Wide Column Store, 2018.

- [18] P. Bhatt, "Performance Comparison between Column Store NoSQL Databases," *International Journal of New Innovations in Engineering and Technology*.
- [19] Scylla, "What is a Wide-Column Database? Definition and FAQs," [Online]. Available: <https://www.scylladb.com/glossary/wide-column-database/>. [Accessed 01 2023].
- [20] "A. Hogan et al., "Knowledge Graphs," *ACM Computing Surveys*, vol. 54, no. 4, pp. 1–37, Jul. 2021, doi: 10.1145/3447772".
- [21] Microsoft, "SQL Graph Database Sample," 18 11 2022. [Online]. Available: <https://learn.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-sample?view=sql-server-ver16>. [Accessed 01 2023].
- [22] Amazon Web Services, Inc., "What Is a Graph Database?," [Online]. Available: <https://aws.amazon.com/nosql/graph/>. [Accessed 01 2023].
- [23] A. S. Daniel, "Use cases for graph databases," 6point6, [Online]. Available: <https://6point6.co.uk/insights/use-cases-for-graph-databases/>. [Accessed 01 2023].
- [24] E. Brewer, "Towards robust distributed systems," in *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, Portland, Oregon, 2000.
- [25] E. Brewer, CAP Twelve Years Later: How the "Rules" Have Changed, 2012.
- [26] W. McKnight, Chapter Ten - Operational Big Data: Key-Value, Document, and Column Stores: Hash Tables Reborn, Morgan Kaufmann, 2014, pp. 97-109.
- [27] D. Kunda and H. Phiri, "A Comparative Study of NoSQL and Relational Database," *ZAMBIA INFORMATION COMMUNICATION TECHNOLOGY (ICT) JOURNAL*, vol. 1, no. 1, 2017.
- [28] MongoDB, Inc., "NoSQL vs. SQL Databases," [Online]. Available: <https://www.mongodb.com/nosql-explained/nosql-vs-sql>. [Accessed 01 2023].
- [29] T. Wanschik, "JOINS via denormalization for NoSQL coders, Part 1: Intro," 09 2010. [Online]. Available: <https://www.allbuttonspressed.com/blog/django/2010/09/joins-via-denormalization-for-nosql-coders-part-1-intro>. [Accessed 01 2023].
- [30] K. e. a. Beck, The Agile Manifesto, Agile Alliance, 2001.
- [31] C. Nance, T. Losser, R. Iype and G. Harmon, NOSQL VS RDBMS - WHY THERE IS ROOM FOR BOTH, SAIS 2013 Proceedings. 27, 2013.
- [32] P. Sadalage J. and M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Addison-Wesley Professional, 2012.