BIRMINGHAM CITY
University

CMP6207 - Assignment 1

IOThings Application Report

Lewis Higgins - Student ID 22133848

CMP6207 - Modern Data Stores

Module Coordinator: Konstantinos Vlachos

Contents

Introduction

The report will be based on the design and implementation of a MongoDB NoSQL database system for the company "IoThings Home Automation Solutions". There are two assignments in this module, where this report is worth **60%**, and a presentation is the remaining **40%**. This module will also incorporate elements of web design, with HTML and CSS also playing a role alongside the primary use of JavaScript.

- The presentation is about this report. It must cover the design, implementation and data management.
  - You would show him your database cluster (local or Atlas?) as part of it.
- IT WILL BE THIRTY MINUTES. If the report contains all the screenshots (which it will) you might not need to make slides, though consider it anyway. You can still show the report alongside the slides where needed.
- He wants you to make an Atlas account even if you do it locally.
- You are creating the dataset for this assignment. You will describe it in an appendix rather than the report's main body.
- This is a "professional report", and as such the title page with the BCU logo and your info should probably change. It needs to also show the date.

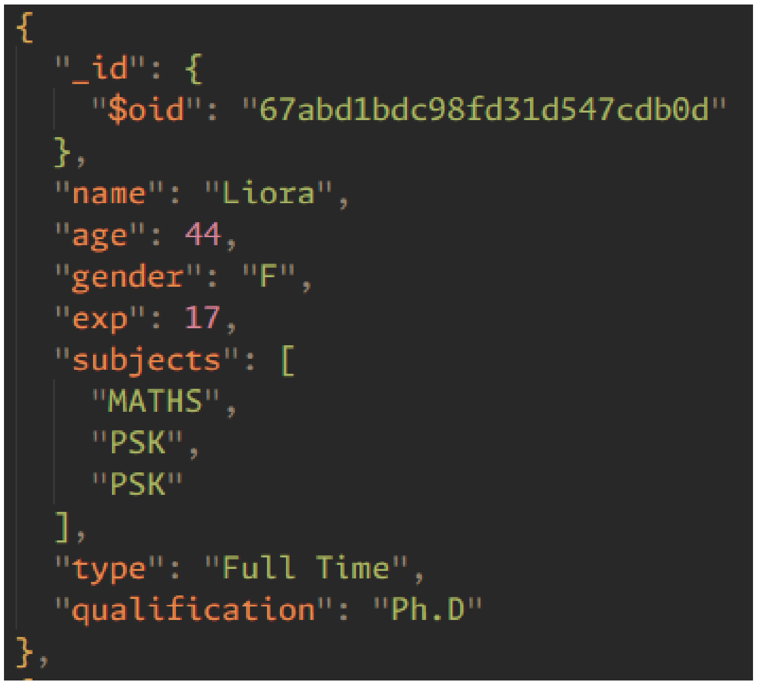
Types of NoSQL databases

Structured Query Language, or SQL, was developed by IBM following Codd (1970)'s ground-breaking publication in the ACM journal, with the first commercial SQL implementation being published by Oracle in 1979 (Oracle, 2025). SQL powers many relational database systems even today, though the problems associated with its age, most notably in the speed of its operations, are beginning to show in modern systems. Therefore, NoSQL ("Not Only SQL") was developed as an extension of SQL, allowing data to be stored in a non-tabular, non-relational format for efficient storage of semi-structured and unstructured data in a flexible, functional and scalable model for faster operations than standard relational databases in most scenarios (Google Cloud, 2025; AWS, 2025e).

There are a wide variety of NoSQL database types - all of which vary in complexity, functionality and purpose, meaning that the identification of the most suitable type is paramount for maximum efficiency in terms of speed, storage and scalability.

1.1 Document database

Document databases are intuitive, flexible and horizontally scalable databases that work well in a wide variety of use cases for both transactional and analytical purposes, including IoT data and real-time analytics (MongoDB, 2025a). They store records as "documents", which store an object's data and metadata, in a format such as JSON, BSON, or XML[1]. Details and examples of these file types can be found in Appendix A.



```
{
  "_id": {
    "$oid": "67abd1bdc98fd31d547cdb0d"
  },
  "name": "Liora",
  "age": 44,
  "gender": "F",
  "exp": 17,
  "subjects": [
    "MATHS",
    "PSK",
    "PSK"
  ],
  "type": "Full Time",
  "qualification": "Ph.D"
},
```

Figure 1.1: An example of a JSON Document.

[1]JavaScript Object Notation, Binary JSON and Extensible Markup Language, respectively.

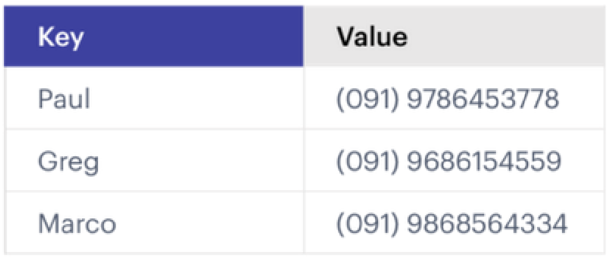
Figure 1.1 depicts an example JSON document in MongoDB[2], a popular DBMS for document databases. It contains the data of a singular example school teacher, storing details such as their name, age and subject expertise, as well as a unique internal object ID used by MongoDB to identify that document. The data itself is of varying types including strings, integers and arrays, which makes document databases easily integrable into a development workflow due to the direct storage of object types used in programming languages like Python and JavaScript.

Relationships in document databases can be modelled with great ease, with the ability for documents to contain other documents. For example, in MongoDB, a document for a social media user may contain the Object ID of each of their posts. Groups of documents can then be stored in **collections**, similar to tables in relational databases. These collections can be queried to perform all CRUD operations.

Popular services for document databases include Databricks (Databricks, 2023) Couchbase (Couchbase, 2025), and the previously mentioned MongoDB (MongoDB, 2025a).

1.2 Key-value database

Key-value databases are primarily reputed for their speed and simplicity, functioning by storing each record as a key-value pair. Rather than having to search through massive amounts of irrelevant data for a query, key-value databases can instead search through their stored keys to retrieve results within milliseconds or even microseconds if used in-memory (Redis, 2025a). While this is excellent for simple queries to retrieve specific known records, this same property also causes major limitations in that retrieving data based on values, such as finding all users over a given age, would require the entire database to be searched, making key-value databases best suited for real-time data access and caching where simpler queries are used (MongoDB, 2025b).



Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334

Figure 1.2: An example key-value pair (Redis, 2025b).

Figure 1.2 depicts an example key-value pair, mapping names to phone numbers. If a query for "Greg" was given, the associated value would be returned. There is a significant similarity between key-value stores and document stores, though key-value stores can only store simple key-value pairs, whereas document stores can have flexible schemas of complex, nested structures. Furthermore, as previously mentioned, querying key-value databases is very limited to only simple queries if speed is a decisive factor.

[2]MongoDB actually stores data as BSON, though it translates between the two when queried. See Appendix B for more information.

Notable software options used across industry for key-value databases include Amazon's DynamoDB (AWS, 2025b), Redis (Redis, 2025b), and Memcached (memcached, 2025). MongoDB can also be used as a key-value store, though it is not primarily intended to do so.

1.3 Wide-column database

Wide-column databases, also known as wide-column stores, store data across columns rather than rows, which allows for data schemas built for these databases to be very flexible. Records stored in wide-column stores do not require the same columns for every row, which can greatly help to reduce duplication and redundancy, and storage requirements by extension. Instead, Figure 1.3 depicts how wide-column databases can store data in "column groups", which would be defined when the database is created.

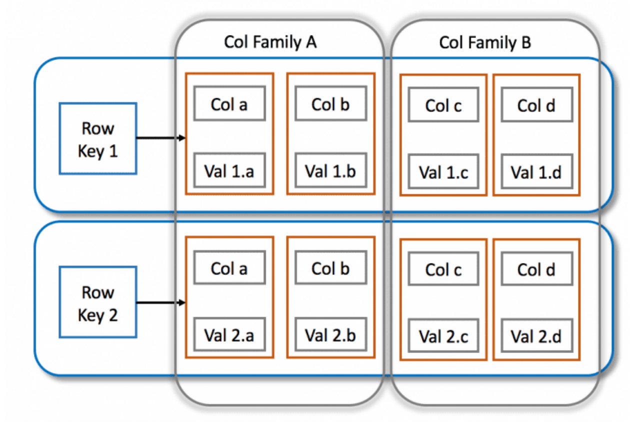


Figure 1.3: An example of wide-column store groups (Pore, 2018).

Column groups are greatly beneficial for **distributed databases**, which leverage sharding for horizontal scalability, a concept explored further in Chapter 2. For example, one node could store the columns of group A, whereas another stores group B. If done correctly, this can greatly help the speed of queries on the database, but it could also cause major slowdowns if the column groups are incorrectly defined, such as with data that would typically be queried together being split over multiple column groups. Therefore, column groups should consist of data that is frequently queried together, such as a group for personal information, then another for financial information (Cattell, 2011).

Because of this, wide-column stores are typically used for larger databases reaching petabytes in size, using software implementations such as Google's Bigtable (Chang et al., 2008) and Apache Cassandra (Apache, 2025).

1.4 Graph database

Graph databases differ heavily from the previously mentioned types, and are instead based on mathematical graph theory (AWS, 2025d). Rather than storing data as rows and columns (or keys and values), these databases store data as nodes and edges. Nodes are connected through their relationships to other nodes, which form the edges of the graph structure as depicted in Figure 1.4.

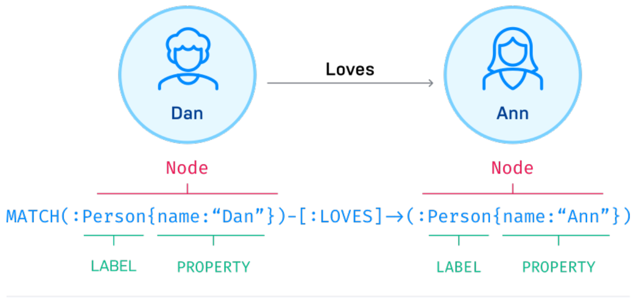


Figure 1.4: Two example nodes connected by a relationship. (Neo4j, 2025c)

Graph databases are best suited for data that is highly interconnected, and can achieve extremely fast query speeds in these scenarios due to graph traversal, where related nodes can be immediately read through the relationships between them rather than the extensive joins that would be required in a relational database equivalent (Corbellini et al., 2017). Typical use-cases of these databases include fraud detection, user recommendation systems and identity management (Neo4j, 2025b; AWS, 2025c; Memgraph, 2025a).

Software typically used for graph databases includes Neo4J (Neo4j, 2025a), Amazon Neptune (AWS, 2025c), and Memgraph (Memgraph, 2025b).

An additional key feature of graph databases is that they do not store collections of data together in groups or collections unlike other databases, with this functionality instead being dictated by labels, relationships or properties.

Label grouping

Nodes can be labelled with a tag (such as "User"), and queries can instead retrieve all nodes sharing a given label.

Relationship grouping

A node is created that represents the group, with all members of the group having a relationship with this parent node such as "Member of".

Property grouping

Nodes individually store a property such as a group ID which identifies their particular group. Queries can then fetch all nodes with the given property to identify members of the group.

Comparing NoSQL and Relational databases

Industrial needs for fast and efficient data storage continue to grow exponentially year by year, and some of these needs fail to be met by typical relational database solutions. Corbellini et al. (2017) state that the types of data that require storage have changed since the first relational database systems of the 1970s, and as such, these more mature systems do not meet the requirements of today's businesses with web-oriented systems storing magnitudes of unstructured data. The same can also be said of Internet of Things (IoT) systems, with Gubbi et al. (2013) stating that 'For the realization of a complete IoT vision, an efficient, secure, scalable and market oriented computing and storage resourcing is essential.'

These efficient, secure and scalable resources exist in the form of NoSQL databases. Chapter 1 provided an overview of the various types of NoSQL databases, with this chapter instead offering direct comparisons between modern NoSQL solutions to more traditional relational database management systems in terms of their scalability and transactional models.

2.1 The CAP theorem

2.2 Scalability

Both Gani et al. (2016) and Katal et al. (2013) state that the amount of data stored by companies is increasing at an extreme rate, forecasting that companies will likely be storing *zettabytes* of data in the near future. Because of this, Gubbi et al. (2013)'s previously mentioned efficient and scalable storage solutions are needed to meet these ever-increasing demands.

Databases can be scaled both vertically and horizontally, with relational databases typically favouring vertical scaling and NoSQL databases favouring horizontal scaling, though both are possible in either database type.

2.2.1 Vertical scaling

Scaling vertically means to upgrade the host device(s) of the database, which refers to the physical swapping or installation of parts such as CPUs, RAM and storage drives to accelerate data processing (Cattell, 2011). A significant benefit of a vertically scaled database is that data will always be consistent as it is stored on a single device rather than distributed across many over a network connection where data could be corrupted, lost or intercepted. This allows for total ACID compliance which is described further in Section 2.3.1.

Despite this key benefit, vertical scaling has some downsides. Server components cannot be swapped while the system is online, meaning that scaling vertically **requires** partial or total system downtime, therefore decreasing availability. Furthermore, vertical scaling introduces a single point of failure to a system, where the powerful server hosting the database could experience any issue such as a power cut that would cause a total system outage. Continuous vertical scaling can also become very costly and bear diminishing returns, with the significant cost of top-grade server parts potentially outweighing the performance benefit of their installation.

2.2.2 Horizontal scaling

Scaling horizontally means to add more devices (nodes) to a network, making the database a **distributed database**. Rather than having an entire DBMS stored on a single device and single point of failure, horizontal scaling divides the processing and storage loads of the database over many nodes.

2.3 Transactional models

Relational and NoSQL databases are governed by dichotomous transactional models that fundamentally change their appropriate use cases. Relational databases are designed to adhere to the ACID model that prioritises absolute integrity and consistency at the expense of both availability and scalability, whereas most NoSQL databases adhere to the opposing BASE model, prioritising availability and scalability over consistency. However, ACID is not exclusive to relational databases, and recent updates to certain NoSQL databases like MongoDB have allowed partial ACID compliance in combination with the scalability benefits of NoSQL.

2.3.1 ACID

ACID is an acronym to describe the following key attributes:

Property	Description
Atomicity	All transactions must be fully completed or not completed at all. If a transaction fails, the database must be reverted to its prior state with no changes made to the data.
Consistency	Data must meet predefined integrity constraints, and remain consistent for all users even in the event of simultaneous transactions.
Isolation	Transactions are executed sequentially, and do not interfere with each other even if they are simultaneous.
Durability	Even in the event of system failure, the database must maintain all committed records. This means that even if an error occurs, the results of any and all previous transactions must not be lost.

Table 2.1: The key properties of ACID compliance. (AWS, 2025a; Neo4j, 2023)

ACID is a very strict model that actively enforces a safe environment for data operations. Though, in maintaining such high security, a performance overhead exists with every transaction. This can massively impact the scalability of relational databases with complex join operations, which will grow slower the more users they have.

ACID compliance is not exclusive to relational databases, however; in recent years, some NoSQL databases have incorporated ACID properties to gain the best of both worlds - high speed and scalability from being non-relational, and high security and integrity from ACID compliance. However, NoSQL databases cannot be fully ACID compliant due to their architecture, as horizontal scaling while maintaining

2.3.2 BASE

BASE-compliant databases operate very differently to their ACID counterparts. Rather than the strict integrity and consistency procedures expected by ACID, the properties of BASE databases are much more relaxed:

Property	Description
Basically Available	Systems will continue to operate even if failures occur due to redundancy and replication.
Soft State	Replicas do not have to be entirely consistent with each other. This means the system can be in two different states simultaneously, which is known as a transient state.
Eventually Consistent	While updates are not occurring, transient states will converge.

Table 2.2: BASE properties. (Corbellini et al., 2017; Cattell, 2011; Neo4j, 2023; AWS, 2025a)

By forfeiting the strict regulations and associated performance overheads of ACID compliance, BASE-compliant databases can theoretically operate at much higher speeds by sacrificing their immediate consistency.

Database design and implementation

MongoDB, an acronym from "hu**mongo**us **DB**", aims to address some of SQL's antiquation issues…

API Implementation and Documentation

Conclusion

Overall, something was done…

Bibliography

Apache (2025). *Apache Cassandra | Apache Cassandra Documentation*. Apache Cassandra. URL: https://cassandra.apache.org/ (visited on 20/02/2025).

AWS (2025a). *ACID vs BASE Databases - Difference Between Databases - AWS*. Amazon Web Services, Inc. URL: https://aws.amazon.com/compare/the-difference-between-acid-and-base-database/ (visited on 21/02/2025).

AWS (2025b). *Fast NoSQL Key-Value Database – Amazon DynamoDB – AWS*. Amazon Web Services, Inc. URL: https://aws.amazon.com/dynamodb/ (visited on 17/02/2025).

AWS (2025c). *Managed Graph Database - Amazon Neptune - AWS*. Amazon Web Services, Inc. URL: https://aws.amazon.com/neptune/ (visited on 21/02/2025).

AWS (2025d). *What Is a Graph Database? - Graph DB Explained - AWS*. Amazon Web Services, Inc. URL: https://aws.amazon.com/nosql/graph/ (visited on 20/02/2025).

AWS (2025e). *What Is a NoSQL Database? - Nonrelational Databases Explained - AWS*. Amazon Web Services, Inc. URL: https://aws.amazon.com/nosql/ (visited on 04/02/2025).

Cattell, Rick (6th May 2011). 'Scalable SQL and NoSQL Data Stores'. In: *SIGMOD Rec.* 39 (4), pp. 12–27. ISSN: 0163-5808. DOI: 10.1145/1978915.1978919.

Chang, Fay, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes and Robert E. Gruber (1st June 2008). 'Bigtable: A Distributed Storage System for Structured Data'. In: *ACM Trans. Comput. Syst.* 26 (2), 4:1–4:26. ISSN: 0734-2071. DOI: 10.1145/1365815.1365816.

Codd, E. F. (1st June 1970). 'A Relational Model of Data for Large Shared Data Banks'. In: *Commun. ACM* 13 (6), pp. 377–387. ISSN: 0001-0782. DOI: 10.1145/362384.362685.

Corbellini, Alejandro, Cristian Mateos, Alejandro Zunino, Daniela Godoy and Silvia Schiaffino (1st Jan. 2017). 'Persisting Big-Data: The NoSQL Landscape'. In: *Information Systems* 63, pp. 1–23. ISSN: 0306-4379. DOI: 10.1016/j.is.2016.07.009.

Couchbase (2025). *Couchbase: Best Free NoSQL Cloud Database Platform*. Couchbase. URL: https://www.couchbase.com/ (visited on 17/02/2025).

Databricks (13th Oct. 2023). *The Data and AI Company*. Databricks. URL: https://www.databricks.com/ (visited on 17/02/2025).

Gani, Abdullah, Aisha Siddiqa, Shahaboddin Shamshirband and Fariza Hanum (1st Feb. 2016). 'A Survey on Indexing Techniques for Big Data: Taxonomy and Performance Evaluation'. In: *Knowledge and Information Systems* 46 (2), pp. 241–284. ISSN: 0219-3116. DOI: 10.1007/s10115-015-0830-y.

Google Cloud (2025). *What Is NoSQL? Databases Explained*. Google Cloud. URL: https://cloud.google.com/discover/what-is-nosql (visited on 04/02/2025).

Gubbi, J., R. Buyya, S. Marusic and M. Palaniswami (2013). 'Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions'. In: *Future Generation Computer Systems* 29 (7), pp. 1645–1660. DOI: 10.1016/j.future.2013.01.010.

Katal, Avita, Mohammad Wazid and R. H. Goudar (Aug. 2013). 'Big Data: Issues, Challenges, Tools and Good Practices'. In: *2013 Sixth International Conference on Contemporary Computing (IC3)*. 2013 Sixth International Conference on Contemporary Computing (IC3), pp. 404–409. DOI: 10.1109/IC3.2013.6612229.

memcached (2025). *Memcached - a Distributed Memory Object Caching System*. URL: https://www.memcached.org/ (visited on 18/02/2025).

Memgraph (2025a). *Graph Database vs Relational Database*. URL: https://memgraph.com/blog/graph-database-vs-relational-database (visited on 21/02/2025).

Memgraph (2025b). *Memgraph Database*. URL: https://memgraph.com/memgraphdb (visited on 21/02/2025).

MongoDB (2025a). *Document Database - NoSQL*. MongoDB. URL: https://www.mongodb.com/resources/basics/databases/document-databases (visited on 16/02/2025).

MongoDB (2025b). *What Is A Key-Value Database?* MongoDB. URL: https://www.mongodb.com/resources/basics/databases/key-value-database (visited on 17/02/2025).

Neo4j (11th Aug. 2023). *Data Consistency Models: ACID vs. BASE Explained*. Data Consistency Models: ACID vs. BASE Explained. URL: https://neo4j.com/blog/graph-database/acid-vs-base-consistency-models-explained/ (visited on 21/02/2025).

Neo4j (22nd Feb. 2025a). *Neo4j Graph Database*. Graph Database & Analytics. URL: https://neo4j.com/product/neo4j-graph-database/ (visited on 21/02/2025).

Neo4j (2025b). *Graph Database Use Cases*. Graph Database & Analytics. URL: https://neo4j.com/use-cases/ (visited on 21/02/2025).

Neo4j (2025c). *What Is a Graph Database - Getting Started*. Neo4j Graph Data Platform. URL: https://neo4j.com/docs/getting-started/graph-database/ (visited on 20/02/2025).

Oracle (2025). *History of SQL*. URL: https://docs.oracle.com/cd/B13789_01/server.101/b10759/intro001.htm (visited on 04/02/2025).

Pore, Akshay (16th Feb. 2018). *NoSQL Data Architecture & Data Governance: Everything You Need to Know*. Dataversity. URL: https://www.dataversity.net/nosql-data-architecture-data-governance-everything-need-know/ (visited on 20/02/2025).

Redis (2025a). *Redis FAQ*. Docs. URL: https://redis.io/docs/latest/develop/get-started/faq/ (visited on 17/02/2025).

Redis (2025b). *What Is a Key-Value Database?* Redis. URL: https://redis.io/nosql/key-value-databases/ (visited on 17/02/2025).