DEVLOPING FOR NOSQL

Contents

[**Data Models** 2](#_Toc153542142)

[**Types of schemas:** 2](#_Toc153542143)

[**-**Conceptual scheme 3](#_Toc153542144)

[-logical scheme 3](#_Toc153542145)

[-Physical scheme 4](#_Toc153542146)

[**Re-design of existing RDBMS** 5](#_Toc153542147)

[Mongo 7](#_Toc153542148)

[BERKELEY 9](#_Toc153542149)

[**History of DBMS** 10](#_Toc153542150)

[MYSQL VS NoSQL 11](#_Toc153542151)

[**Berkeley DB** 12](#_Toc153542152)

[ Berkeley DB Java Edition 12](#_Toc153542153)

[ Berkeley DB XML 12](#_Toc153542154)

[**Mongo DB** 12](#_Toc153542155)

[ Mongo DB community server 12](#_Toc153542156)

[ Mongo DB enterprise server 12](#_Toc153542157)

[ Mongo DB atlas 12](#_Toc153542158)

[**Query Mechanism** 13](#_Toc153542159)

[Language support and connectivity 13](#_Toc153542160)

[CRUD 14](#_Toc153542161)

[Mongo 14](#_Toc153542162)

[Queries 15](#_Toc153542163)

[Mongo 15](#_Toc153542164)

[**Management of NoSQL** 16](#_Toc153542165)

[MONGO 16](#_Toc153542166)

[Aspects of Mongo 17](#_Toc153542167)

[BERKELEY 18](#_Toc153542168)

[Aspects of BERKELEY 18](#_Toc153542169)

[**ACID vs Capt.** 19](#_Toc153542170)

[ACID 19](#_Toc153542171)

[Capt. 19](#_Toc153542172)

[**Tools for Design and dev** 20](#_Toc153542173)

[References 21](#_Toc153542174)

[Diagrams/pictures: 22](#_Toc153542175)

# **Data Models**

To begin I should first say what data is in generally, data is essentially like a raw product. They are facts, figures or statistics that are researched and collected and even analysed. The idea of data is that we use it to process it into information.in other words data is just a raw ingredient and information is like a cooking recipe that helps and tells us how to use and process the data into something useful and readable for us humans. (Suszterova, 2023)

A data model is a virtual representation of an enterprise data elements and its connections between them. They also enable business to collaboratively work together and decide how data will be stored, accessed, shared and updated. The purpose of databases is to help bring all segments of an enterprise to collectively design information system and the databases they rely on. (Anon., n.d.). Data models are used to create databases hence why data models are created. Overall data models allow for companies to have a huge increase in communication among all other segments of their enterprise. Also, they are a type of SQL as they are reinforced by RDBMS.

Before I explain the type of data models, I should first explain the key components about a data models. First, we have an **entity** which are the objects we want to represent in our model and are represented by a table, we have **attributes** which are columns in the tables. There’s **relationship** which defines associations between entity and there’s also **cardinality** which expresses the numerical relationship between two entities. Finally, we have the keys both **Primary** key and **foreign** key. Primary key is a unique identifier of each record, and a foreign key is a group of columns that point towards a primary key.

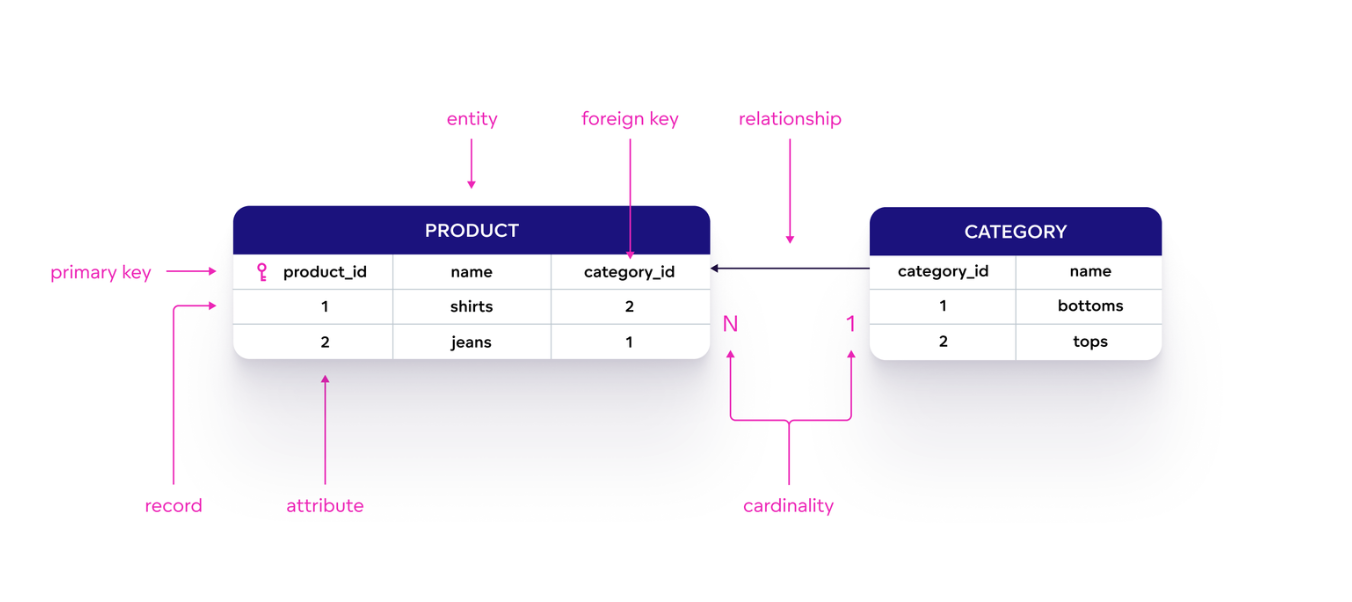


Figure [1]

So, once we have the components, we can now create our database. But which one?

# **Types of schemas:**

Database schemas are blueprints that describe how data are organized and the relationships between the different elements amongst the database as said early on they are there to help analysts to import data from else wise and then they can use this data to help with their own systems. Schemas are usually represented using an entity-relationship diagram that displays values your database will store. (Wang, 2022)

The process of building a data scheme is called data modelling and the schemas help programmers understand the overall degree and size of a project and a good database scheme can be a huge difference as it can have a processing speed of seconds or hours.

data modelling is the process of creating data models to organize and structure data making it easier and more understandable to read. It’s used to develop blueprints for a database. In data modelling the three types of data models are conceptual, logical and physical. The first step is to make the conceptual data model and there’s not really any set of rules to follow, after that you can now create the logical data model, this helps to identify and gather all requirements for the system and finally you then create a physical data model from the logical and once that’s done it can now be implemented into a database system ready to store data and update it (Suszterova, 2023)

## **-**Conceptual scheme

Is a high-level overview of what your database will contain and how data will be organized, this is to establish the entities and attributes and relationships. This is the first schema that will be created in the initial state. This focuses on the main concepts and doesn’t really go into detail. An example of what a conceptual scheme will look like is:

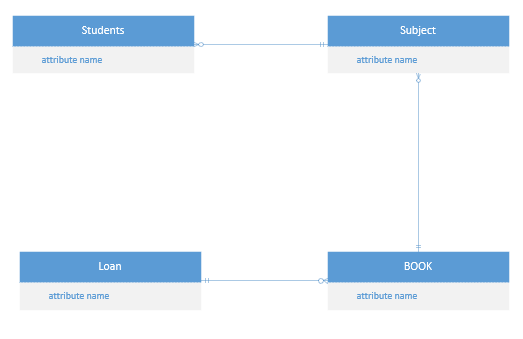


Figure [2.1]

As you can see as is the initial scheme there isn’t much that’s needed to be created, it is basically the blueprint of what your database is going to be like so in a conceptual scheme you will only have the entity (name) and the relationship.

## -logical scheme

A logical schema defines all the elements within the database and relevant information. Essentially the logical scheme is carried on from the conceptual but with added things. Them being field names, entity relationships (aka cardinality) and table names. This type of schema states the logical rules or constraints that show how the elements must interact within the database. An example is:

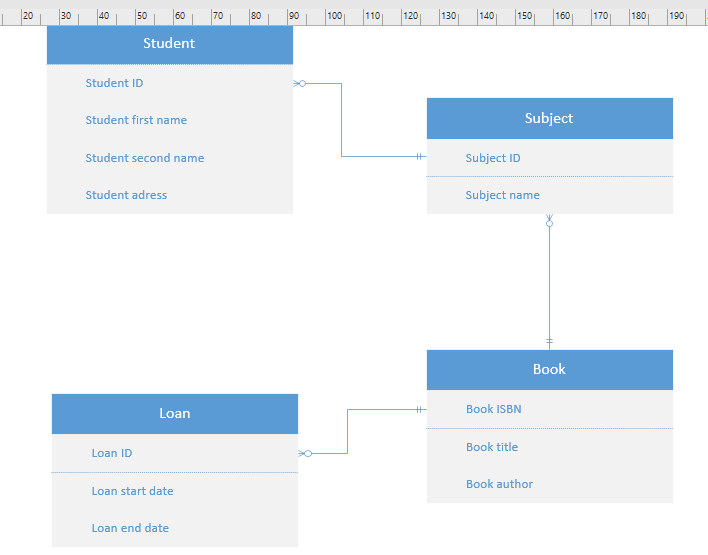


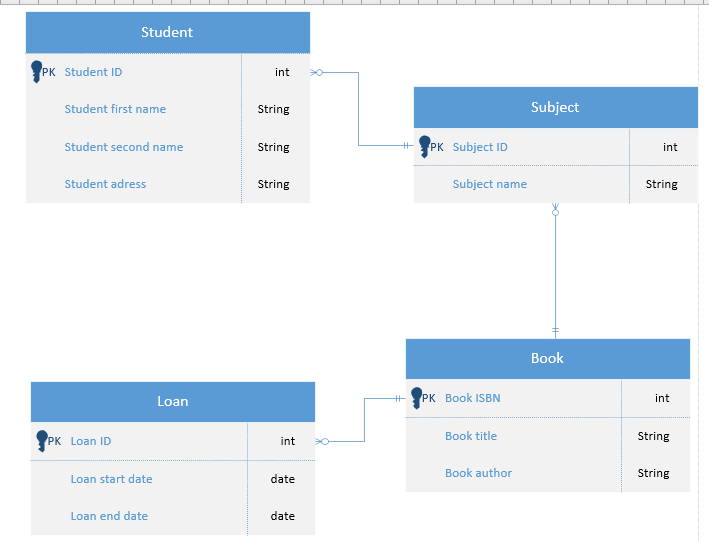
Figure [2.2]

As shown in diagram above this has more detail than the conceptual that’s because it has the added key features columns and cardinality.

## -Physical scheme

The last type being the physical scheme is the grand scheme as it combines the contextual and logical information and schemes together while adding the final requirements. Physical schema has a Primary and Foreign key added and the column type.

As you can see in diagram 2.3 below this is the ideal result of a database scheme that you want your business or enterprise to have when creating a unified database scheme.

 Figure [2.3]

logical and physical data models are the most widely used data model types both used rom either a business POV or database POV, but the real difference is as shown in the table below.

|  |  |
| --- | --- |
| **Logical data model** | **Physical data model** |
| BUSINESS POV | DATABASE POV |
| A model of rules and data structure | A model of actual database tables and views |
| Involves entities, attributes, relationships and Primary and Foreign keys | Adds onto the logical model with data types, cardinality name of tables and columns and how they are defined |
| No data | Data has been already stored and Users create views as SELECT or triggers as INSERT UPDATE AND DELETE |

Figure [3]

# **Re-design of existing RDBMS**

I will now shortly compare the development of a DBMS with NoSQL

Starting off with **MONGO** I will use an example of my previous project which included using SQL to create a ‘training ‘centre database which included students name, courses, grade etc.

‘

CREATE TABLE take

(code CHAR(2) ,

no INT NOT NULL,

grade TINYINT NOT NULL,

CONSTRAINT codex PRIMARY KEY (code, no),

CONSTRAINT codey FOREIGN KEY (code) REFERENCES module (code),

CONSTRAINT no1 FOREIGN KEY (no) REFERENCES delegate (no));

‘

Figure [4]

CREATE TABLE module

(code CHAR(2),

name VARCHAR(30) NOT NULL,

cost DECIMAL(8,2)NOT NULL,

credits TINYINT NOT NULL,

course\_code CHAR(3),

CONSTRAINT code PRIMARY KEY (code));

CREATE TABLE delegate

(no INT,

name VARCHAR(30) NOT NULL,

phone VARCHAR(30) NULL,

CONSTRAINT no PRIMARY KEY (no));

INSERT INTO take (code,no,grade)

VALUES

('A2','2003',68),

('A3','2003',72),

('A4','2003',53),

('A2','2005',48),

('A3','2005',52),

('A2','2002',20),

('A3','2002',30),

('A4','2002',50),

('B2','2008',90),

('B2','2007',73),

('B3','2007',63);

Figure [5] [Figure 6]

As you can see from the fig 4 above this is how I used SQL to create a database that contained the necessary information with the necessary results, keep in mind this is only a small section of the overall project. Fig 4 Above shows how I created a table with the create table function and the attributes that would be present in. fig5 shows the insert value which is what allowed us to tell the computer to put the values in the following order as if it was any other way there would be an error. As you will see in fig 4 the table take has foreign keys this is because it is a key that is found in another table that is primary hence, making it a composite key. This was used to show the relationship between the tables. You can confirm this in fig 6 in the two tables where there is an attribute present in the tables that can be linked to the take table. Essential this is how using SQL I was able to create a database in the way I needed for my project. Now using the same project, I am going to show how it will look different in NoSQL more specifically Mongo and Berkeley.

To begin I will explain first my step by step in making this. First, I used the command ‘use school’ which allowed me to use it as a database that was not yet specified but existing and after that I used the db.createCollection to create the collection called take as show in the code below.

Use school.

db. createCollection("take")

Figure 7

After this I was ready to get started and the first thing I had done after essentially creating the table in NoSQL was started to input the values into the collection take just to show its difference. its essentially the same with the layout but as you will see in the snippet below its more long winded as you must type what attribute it is with every new one, but it’s not that much more difficult.

{

acknowledged: true,

insertedIds: {

'0': ObjectId("65771fc41de62ba1b2accad4"),

'1': ObjectId("65771fc41de62ba1b2accad5"),

'2': ObjectId("65771fc41de62ba1b2accad6"),

'3': ObjectId("65771fc41de62ba1b2accad7"),

'4': ObjectId("65771fc41de62ba1b2accad8"),

'5': ObjectId("65771fc41de62ba1b2accad9"),

'6': ObjectId("65771fc41de62ba1b2accada"),

'7': ObjectId("65771fc41de62ba1b2accadb"),

'8': ObjectId("65771fc41de62ba1b2accadc"),

'9': ObjectId("65771fc41de62ba1b2accadd"),

'10': ObjectId("65771fc41de62ba1b2accade")

}

db.take.insertMany([{code:'A2', no: 2003, grade:68},

{code:'A3', no: 2003, grade:72},

{code:'A4', no: 2003, grade:53},

{code:'A2', no: 2005, grade:48},

{code:'A3', no: 2005, grade:52},

{code:'A2', no: 2002, grade:20},

{code:'A3', no: 2002, grade:30},

{code:'A4', no: 2002, grade:50},

{code:'B2', no: 2008, grade:90},

{code:'B2', no: 2007, grade:73},

{code:'B3', no: 2007, grade:63}])

Figure 9

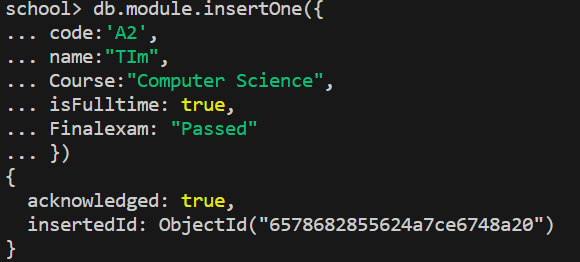
Figure 8

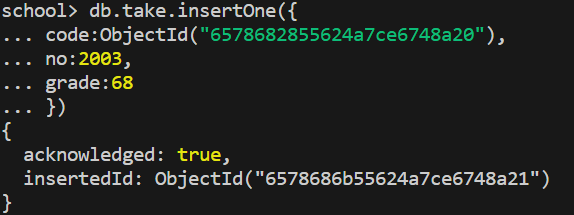
After inserting the values on the left and pressing enter we get the following message show on the right and with the function db.take.find() it shows us all of the documents in the collection.

### Mongo

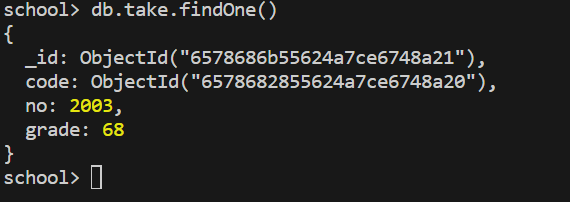
Mongo also has its own relationships that are modelled in its own way. It doesn’t have as many as Berkeley, but it does have a one-to-one and a many-to-many. Mongo can have embedded document model which basically mean that if we have two or more documents, we can embed them together to make one single document which will help the user retrieve the data using single queries rather than writing a bunch of queries.

One-to-one

 As you can see first, we make our document that we want to show the link to another: I made up attributes of my own, but it doesn’t make a difference to the overall document. So, this is document module, shown in figure 10.

 Figure 11 Figure 10

now making the second document called take you can see one difference is that the code has the object id for the table module. This is to show the relationship between the modules as you can now just look for one document that being take and if you want to find any documents that have a relationship to it you can now see the object id and find it. Figure 12 below shows the result of when you bring up the take documents as you can see it has its own ID and then its code links to the module document.

 Figure 12

Many-to-Many

There are 2 ways in which this can be showed by either making single documents multiple times, which means making three documents and having one contain all the others, or by making an embedded document which is the approach I chose to do.

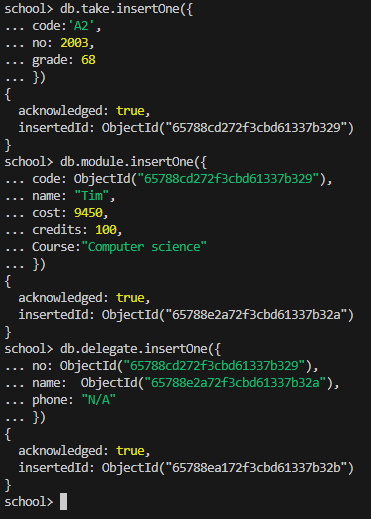


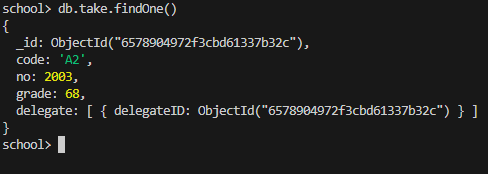
Figure 13

The snippet to the right is to show what I meant by making single documents to show the relationships between the documents. The main one to look at is the delegate which shows how it contains two documents linked to it showing a many to many relationships.

But as you will see this is not an embedded document as I will go on to show you in the snippet below. (Ignore the slight changes in attributes, one again it doesn’t make much of a difference)

 Figure 14

Now this snippet is different to the previous as its to show the layout for how we would embed documents within each other. We first make the documents and once that done, we can no go and update one of the already existing documents as show in the update function. The result will be the snippet shown below.

 Figure 15

### BERKELEY

Berkeley is a lot more complicated and lots of coding, but I will show the basics once again. First, to code it to make a table of some sort we need the following:

@Entity and public/private class

The @Entity basically declares an entity class which is a class with a primary index and the public class simply just states that table name and what’s going to be included in it.

Figure 17

Figure 16

@Entity

public class module {

}

private String name;

private int cost;

private int credits

Figure 16 is what you will use to create the table and figure 17 is the attributes in it but now we need to define our primary and secondary key.

// The primary key must be unique in the database.

@PrimaryKey

private String code;

Figure 18

Figure 18 is how we define our primary key with the @primarykey function.

@SecondaryKey(relate=ONE\_TO\_MANY)

private String course\_code;

Figure 19

Figure 19 is how we defined the secondary key and as you may see this is also used to show the relationship between another entity class. Essentially with just these few commands I am able to remake a table coding with Berkeley.

Berkeley relationships

Just like a SQL they also have modelling relationships but unlike SQL it can’t be represented in a table like form. Berkeley specifically has four types being: one-to-one, one-to-many, many-to-one, many-to-many. To fully explain its best, I will show the elements you will need to display a relationship in Berkeley. First, there is a required element that you need that being ‘Relationship’ or more in depth you need the phrase ‘public abstract Relationship relate’ which this defines the relationship between instances of the entity class and the secondary keys. So, in other words this is what essentially what enables us to create relationships. There are optional elements that you can have as well such as ‘java. lang. string’ which is used to specify the name of the key in order to use a name that is different than the field name.

As already mentioned, Data models (database system) are SQL related as they are backed by a relational database management system (RDBMS) which suggest that any type of model not backed by RDBMS is an alternative to SQL. Alternatives to SQL are called NoSQL and two main examples I will talk about is MONGO DB and BERKELEY DB. BUT first let’s find out about the history of DBMS AND THE RISE IN NOSQL

# **History of DBMS**

The first DBMS were developed in the 1960s and the 1970s. they were based on hierarchical or network models. These models represent data as trees or graphs, and they required users to define the structure and relationships of data beforehand. (AI, n.d.). A database management system (aka DBMS) allows a person to organize store and retrieve data from a computer in other words it’s a way of communicating with a stored memory within a computer. DBMS were introduced so that databases, a collection of information, could be organized. The first two DBMS to be created were the ’integrated database system’ by Charles W. Bachman and the ‘Information Management System’ (aka IMS) by the IBM as they didn’t want to be left out. Then came RDBMS in the 1970 first researched by IBM, which was an efficient way to store and process structured data. Unstructured data, being both non-relational and schema-less, became more common and processing speed got faster as a result and they were not designed to handle this kind of data so in came MYSQL. (Foote, 2021)

## MYSQL VS NoSQL

In 1995 SQL was launched by Davide Axmark, Allan Larsson and Michael Widenius. SQL being an open-sourced relational database management system that is based on the structure of query language became one of the most popular databases in the world. Key features of my SQL are:

* It is easy to deploy and use.
* It supports ACID – which I will talk about later.
* It is obviously a RDBMS that has very fast loading utilities.
* And offers good security.

My SQL evolved into an extremely scalable data base system with the ability to operate on multiple platforms. (Foote, 2021). And just like in life there’s always room for improvement and more innovations this being NOSQL.

Rise of NOSQL

The acronym was first used in 1998 by Carlo Strozzi while naming his lightweight open-source relational database that didn’t use SQL.it stands for ‘NOT ONLY SQL’ and was developed as a response to the need for processing unstructured data and the need for faster processing. (Upadhyay, 2022)

The reason why NoSQL became so popular as quick as it did was because not only does NoSQL handle both structured and unstructured data it can also process unstructured big data very quickly.

The noticeable advantages that NoSQL has over SQL are the following:

* Scalability – they are designed to scale out by using distributed clusters of hardware instead of adding expensive and robust servers.
* Cost effective – leading on from the previous point, as a result they have much lower cost than RDBMS as they only rely on cheap hardware compared to high end server that SQL uses.
* Big data management – NoSQL works well with big data where large amounts of data is being generated by the web. So, they can handle a lot more data.
* Flexible data models – as already mentioned they can store and process unstructured and structed data, example text or audio and video.

With the rise of social media, it was hard for SQL alone to keep track and process data so with the help of this its arguably the reason why NoSQL grew in popularity so much.

There are four types of NoSQL: Key Value database, Document database, column family store and graph databases. But I will only talk about the first two with examples of them.

Key value database is as simple as it sounds, they are the uncomplicated data stores and the client can either put, get or a delete a value for a specific key from the data store. They have primary access and hence allowing for easy scalability and great performance. (Monnappa, 2022)

And the other one I will go into is Document databases which the concept is its focused in documents. They are stored and received from the databases that can be stored in XML and JSON. (Monnappa, 2022). Now that I have explained SQL history and the rise in NoSQL and the types of it, I will now explain the examples I previously mentioned.

## **Berkeley DB**

Berkeley DB is a key value store and is solely embedded. It was originally developed by Sleepy Cat before being bought by oracle in 2006. There are three editions one of them just being its normal state Berkeley DB. This edition supports B-tree, which is an organizational structure for information storage and retrieval in the form of a tree, and extendable Hashing and multiple language bindings (Hughes, n.d.)

## Berkeley DB Java Edition

Open source, embeddable, transactional storage written entirely in java. The architecture of oracle Berkeley java edition supports very high performance and concurrency for both read and write intensive workloads. This edition is the best if applications require something outside the bounds of relational databases. It also stores data quickly and easily.

## Berkeley DB XML

Embedded xml database specifically Designed solely for processing storing and retrieving XML documents.

## **Mongo DB**

Mongo DB, which is also a NoSQL, is a document store that originally developed by 10gen, before they became MongoDB Inc in 2013. It supports CAPt. And data storage and networking employs BSON (Binary JSON). and just like Berkeley there is three editions. (Hughes, n.d.) They are the following:

## Mongo DB community server

Free available edition for windows, Linux and OSX however typically installs as a service which is basically the same as MySQL. Also offers fully managed services with atlas.

## Mongo DB enterprise server

Commercial edition of MongoDB which includes additional capabilities such as memory storage for high throughput, advanced security features and encryption.

## Mongo DB atlas

Integrated suite of data services centred around a cloud database designed to accelerate and simplify and how This document enables developers to store data as JSON-like objects that resemble objects in application code. (N/A, n.d.)

As already the advantages/benefits of using these NoSQL have been talked about but now it comes down to which is better if any is better. SQL vs NoSQL.

There are five critical differences between SQL databases and NoSQL databases. These being SQL is relational and NoSQL is non-relational. SQL databases uses SQL and have schemas whereas NoSQL have dynamic schemas for unstructured data- meaning SQL deals with structured data and NoSQL deals with both structured and unstructured. While both are scalable SQL is vertically and NoSQL is horizontally. SQL are table based whereas NoSQL is varied on four types (refer to the four types explained earlier on).and finally SQL is better for multi-row transactions whereas NoSQL is better with unstructured data such as JSON. (Smallcombe, 2023)

NoSQL is preferred as it offers more benefits such as flexibility and scalability and the primary benefit of using this is it provides developers the ability to store and access data quickly so it’s why its typically used over SQL but just like anything else it also has its own weakness, some NoSQL can be quite resource initiative demanding high RAM AND CPU allocations (Foote, 2021), so the real question is which is better ... SQL or NoSQL?

The decision all comes down to needs and requirements of a project, for example if it’s an important project that requires fast and reliable database then it’s preferred to use a NoSQL database. But if your project is more complex than its better to use SQL.

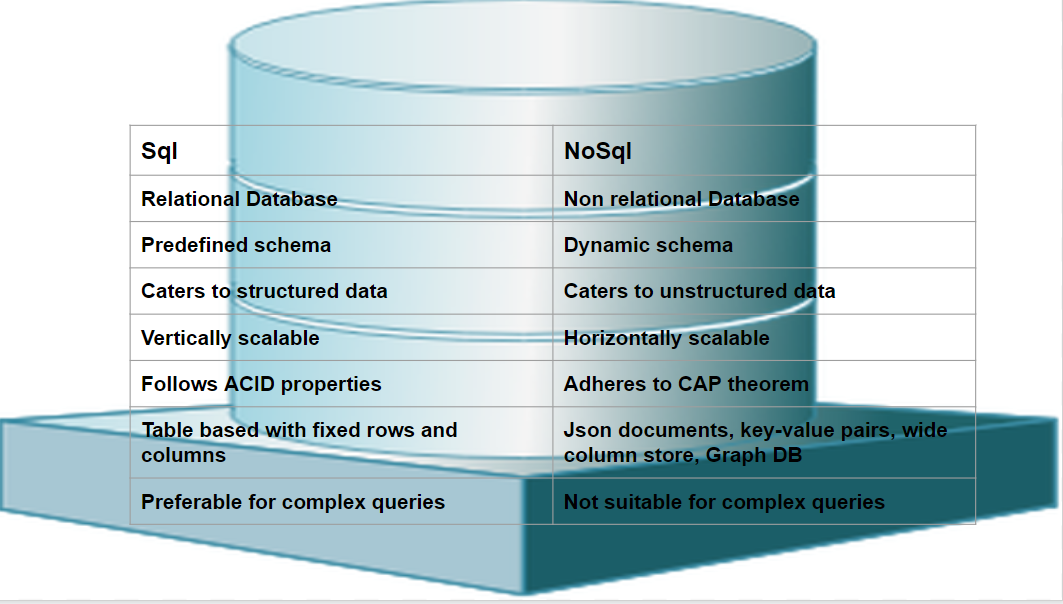


Figure [20]

# **Query Mechanism**

Going back to the redesign of existing DBMS with NoSQL, you will have noticed I have already given a small example on how we would redesign projects made in SQL but into NoSQL. First let’s talk about the,

## Language support and connectivity

MongoDB supports a lot of languages such as the following:

 [C](https://www.mongodb.com/languages#languages) : used for embedded systems so useful for mongo.

 [C++](https://www.mongodb.com/languages#c) : accessible through the mongoxcc driver

 [C#](https://www.mongodb.com/languages#c-) : accessible through MongoDB C#.NET

 [Go](https://www.mongodb.com/languages#go) : data friendly syntax used with Mongos flexible schema.

 [Java](https://www.mongodb.com/languages#java) : support multiple platforms.

 [Node.js](https://www.mongodb.com/languages#nodejs) : good for developers due to the JSON-like syntax that are used by Mongo.

 [PHP](https://www.mongodb.com/languages#php)

 [Python](https://www.mongodb.com/languages#python)

Etc. (Mongo, N/A). as mentioned later mongo uses MongoDB Compass to connect to servers as well as Mongo Shell.

Berkeley supports a lot of the languages like Mongo these being: C, C++, java, PHP, Python.

## CRUD

CRUD, which is the acronym for create, read, update and delete. These terms are the four main operations for creating and managing both SQL and NoSQL database. The create operation adds a new record to the database. The read returns records from the database. The update is used to modify existing records and finally the delete operation allows user to remove records from the database.

### Mongo

**Create**- in mongo we use the function db.createCollection(“”) which creates a collection in which we can start adding documents into it. Refer to figure 21 below.

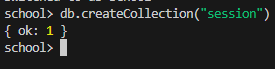
This simple function allows us to create collections.

figure 21.1

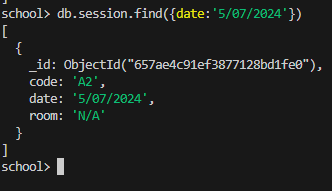
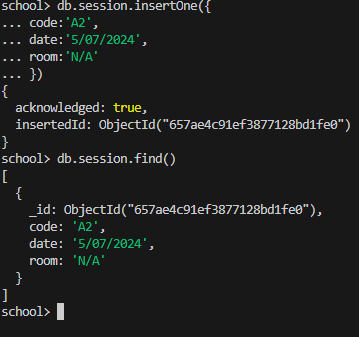
**Read**- to basically display what information we want we just use the db.\*Collection name\*.find() . if we wanted, we could also just search specifical attributes to find and read documents as shown below.

figure 21.2

**Update-** to do this there is also a few options but the main thing is the function db.\*Collection name\*.update({},{}). We can updateOne or updateMany .

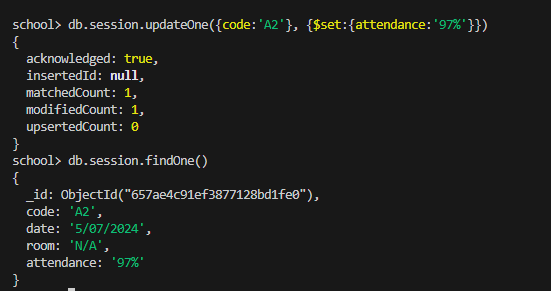
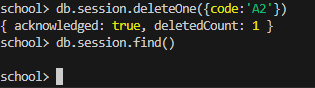


figure 21.3

Figure 21.4

As displayed in figure21.3 the update function is shown and within, we see the first set of {} is to determine which document/attribute we want to update and the second one determines what we want to update it to. Refer to 21.2 you can see there is no attendance attribute until we decided to update it and add one.

Finally **delete-** just like the others there’s multiple ways to delete a collection but the function I uses is db.\*Collection name\*.deleteOne(). As simple as shown in figure 21.4 it just simply deletes the collection you choose, and if you use Many instead of One you delete all documents in the collection.

## Queries

This section is to demonstrate some queries in

Mongo

To better demonstrate how this whole project would work in NoSQL I made a partially complete database with multiple tables and the values inserted into it. Each number is an individual collection with attributes but as you may notice in some, they have the attribute of another tables object id, and this is to show the relationship between the tables as mentioned already.

1.db.module.insertMany([{code:'A2',name:"ASP.NET",cost:250,credits:25,course\_code:'WSD'},{code:'B2',name:"Oracle",cost:750,credits:50,course\_codes:25,course\_code:'NSF'}])

2. db.take.insertMany([{no: 2003,code: ObjectId("657af4481ef3877128bd1fe1"),grade:68},{no:2008,code:ObjectId("657af4481ef3877128bd1fe2"),grade:90},{no:2008,code:ObjectId("657af4481ef3877128bd1fe3"),grade: 'N/A'}])

3. db.session.insertMany([{code:ObjectId("657af4481ef3877128bd1fe1"),date:' 2023.06.05',room: 305},{code:ObjectId("657af4481ef3877128bd1fe2"),date:'2023.08.23',room: 208},{code:ObjectId("657af4481ef3877128bd1fe3"),date: 'N/A’, room: 'N/A'}])

4. db.delegate.insertMany([{no:2003,name:"Sarah",phone: 'N/A'},{no:2008, name:"Tom",phone: 'N/A'},{no:2008,name:"John",phone:'N/A'}])

5. db.course.insertMany([{code:ObjectId("657af4481ef3877128bd1fe1"),name:" Web Systems Development",credits:75},{code:ObjectId("657af4481ef3877128bd1fe2"),name:"Database Design and Management",credits:100},{code: ObjectId("657af4481ef3877128bd1fe3"),name:"Network Security and Forensics",credits:75}])

Figure 22

The figure above just shows the code I made to writer each table and as you can see, I made a few have some sort of links back to other tables as they had foreign keys, so they linked and had the same attribute. And using the function db.collectionname.find() I was able to display all the values but what if there was only one specific document I wanted to find amongst multiple.

Thankfully I did this with the functions db.collectionname.find({}) which allowed to me look for a document by the attribute defined.

Another query I will demonstrate has already been mentioned earlier on and this is the find documents within documents.

After using the find function to call session it shows all documents with in it but I wanted to find one with the room number 305 assuming there was hundreds making it to many to read. Using the function db.session.find({room:305}) I was able to single out this document as it was the only room with this room number

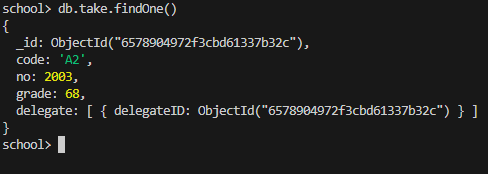


Figure 23

This snippet above shows a collection with a document in it which has the reference to another document in another collection making this many-to-many. This idea being an embedded document is that when I read from one document, and I want to find out more or who it made about it will show the id of the document.

db.delegate.find({delegateID: ObjectId(“6578904972f3cbd61337b32c”)})

Upon using the function above it should display the collection delegate with the specific id above and its following attributes.

# **Management of NoSQL**

As shown, you will have already seen a few snippets of what the code would look like and what more, but I will explain just a little more how you would manage Mongo DB and Berkeley DB.

# MONGO

First, to manage mongo you will need to download it and you can do this through various of ways this being using Mongo community edition and its compass but the way I am using it is through visio code, so I am going to take you through the process. The steps to downloading and being able to manage Mongo in vs code is:

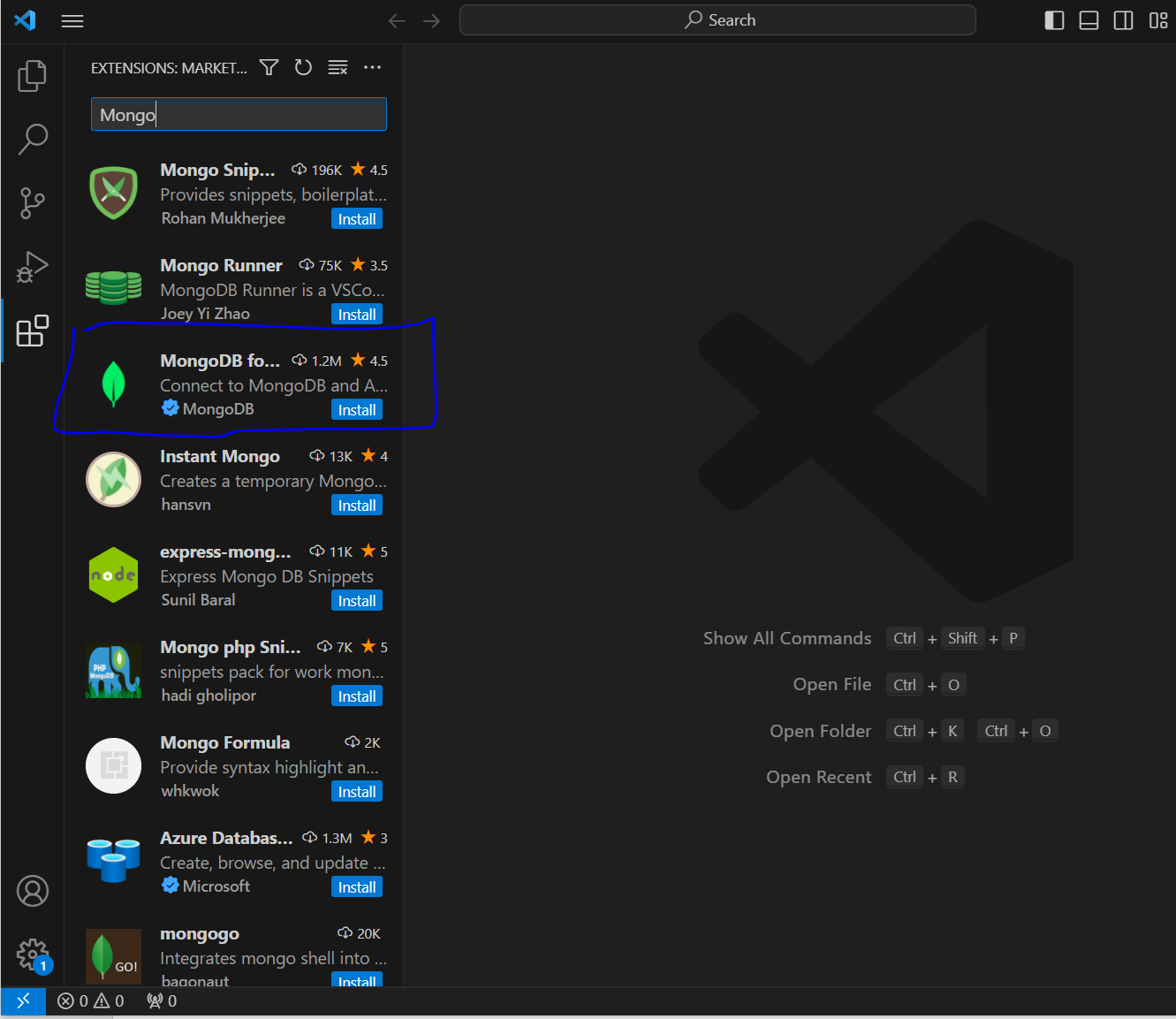
* Install an extension in VS code. – simple and easy step just look up Mongo in the extension part and install it
* Before we do that, we need to make sure our mongo server is running and to do this we verify with the help of mongo DB Compass

Figure 24

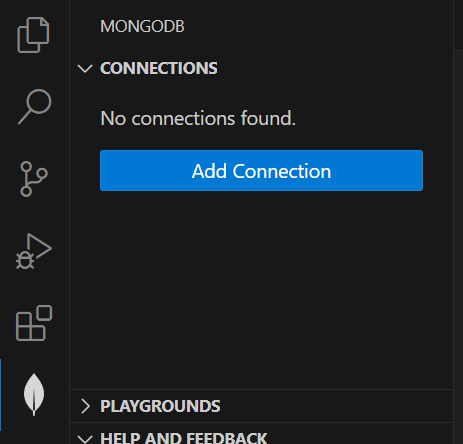
* Next step after all that is done you head over to the left-hand side and click the leaf icon. Here we can add connections but as you may tell we get the same pop up as when we installed it but this is just for future purposes

Figure 25

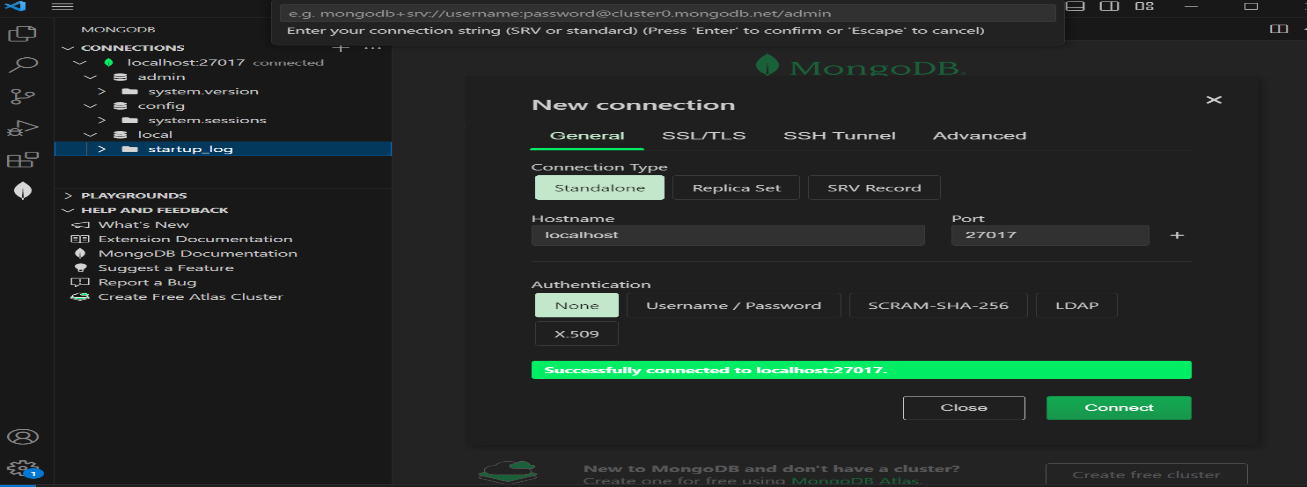
* After the pop up comes up, we then just click on add form and as everything was already set up for us just click connect and we are connected as you will see in the pictures to the below. This is how we get mongo to connect to VS Code.

Figure 26

* Finally, now we need to attach the MongoDB SHELL and simply doing this we just right click on the connections and click the option add MongoDB Shell this helps us work with the terminal.

From here on out we can now code what we want using Mongo but that will come later. For now, I will explain the 5 aspects of this NoSQL.

## Aspects of Mongo

Security

* Authentication

Because of its combination of disabled authentication and just being opened to the internet (mongo community server) it is susceptible to many breaches however MongoDB provides support for authentication on a per database level. Also, it does not support items like passwords and other stuff but thankfully LDAP does, only downside is its not available with the community server version but instead the Mongo DB enterprise. Finally, Mongo supports other various mechanisms for secure access such as x.509 certificates for SSL/TLS which helps integrate with external directory services. (Grippa, 2023)

* Authorization

Enable role-based access control requiring users to identify themselves. Users can only perform actions determined by their roles. (Grippa, 2023)

To create a role in MongoDB and add it to the user you use the db.createRole function.

Indexing

Indexes are one of the most essential tools the database administrator can use to consciously aid the database engine and improve its performance.

Indexes are special data structures that store only a small subset of data held in a collection document separately from the document itself. They are used so that database can quickly and efficiently traverse them when searching for values. (Papiernik, 2021)

Concurrency

Mongo allows multiple clients to read and write the same data. To ensure consistency it uses Locking, which is a mechanism that’s used to maintain concurrency in the database, and Concurrency control to prevent users from modifying the same data simultaneously.

Backup

Mongo Allow backups and you can create this by using the **mongodump** command which dumps data of your server into a directory.

Recovery

Like the backup there is a **mongostore** command that restores a binary backup created by monogdump.

# BERKELEY

Berkeley is different to mongo as it is a key-store value instead of document. But I will be coding this in eclipse. I won’t go through the installation progress this time as its basically a code itself so you can code it in any IDE you choose, mine being eclipse.

## Aspects of BERKELEY

Security

Has good encryption which is intended to protect data from attackers. There’s a utility you can use if you are receiving Berkeley files from other party or if we are worried a file is corrupt and that is ‘db\_verify’. (Anon., n.d.)

indexing

Berkeley DB XML offers effective and flexible indexing functions that give huge advantages to app developers as they have powerful control over query performance.

Concurrency

Has Multi-version Concurrency control (MVCC) which offers repeatable reads but with performance trade off. MVCC makes extra copies of database pages when performing write operations.

Backup

‘Java.long.object \ com.sleepycat.je.util.DbBackup’ is the code you use to simplify backup operations. Also lets the app to create a backup which can support restoring the environment to a specific point in time.

Recovery

Have a write-ahead logging which means that when a change is made to a database page a description of the change is written into a logfile. This is the feature of the logging system that makes durability and rollback work. This suggest if there an error any changes that weren’t added to the log file will be lost but the previous things will be there.

# **ACID vs Capt.**

## ACID

A.C.I.D has four elements, that help with transactions, which stand for the following.

**A**tomicity

Based on the whole idea of ‘all or nothing’. When proceeding with any kind of database transaction it typically has multiple operations. With the help of atomicity, its either all operations success or simply none of them succeed. This is quite a huge thing as operations have big impacts on each other so even just one failing out of the many will affect and ruin the whole transaction leading to unexpected results.

**C**onsistency

This is about ensuring that changes made within transactions are consistent with any database constraints, meaning if the data at any stage goes against the constraint the whole transaction will fail.

**I**solation

This helps make sure that all transactions are run in a separated and isolated environment without interfering with each other.

**D**urability

Arguably the most important element of Acid as this ensures that no matter what happens when a transaction has been complete it will be written to the database. This makes sure that data changes are guaranteed to be there even in the possibilities of errors or power failure.

The four components help make sure that data will be transferred in a consistent and expected state. Mongo DB which has multi-document ACID transaction for more flexibility and Berkeley DB which have five subsystems: caching, datastore, locking, logging and recovery that help transactions are ACID complaint.

## Capt.

The cap theorem is used to make system designers aware of the trade-offs while designing network shared – data systems. Its very important to understand the cap theorem as it makes the basic of choosing any NoSQL database based on the requirements. (Kumar, 2020). The components of CAP are:

Consistency

Means all clients see the same data at the same time, no matter which node they connect to in a distributed system. In order to achieve consistency when a data is written to one node it must be written to all other nodes to be successful.

Availability

This basically means that if there is a successful node that returns a response for all read and write request in a reasonable amount of time even if there is one or more modes down. So, in other words all successful nodes will return a valid response for whatever request without failing regardless of if all nodes are successful.

Partition Tolerance

This enables the system to continue to work despite any message or data loss or failure of the part of the system. Essentially means that if a few computers are unreachable the system will continue to function. (Kumar, 2020)

As you will have probably seen by now that they have some overlapping features such as both ACID and CAP have consistency and that the consistency within CAP also overlaps with ACID atomicity (McQuillan, 2023) .However there’s also differences such as in ACID for it to be successful you need ALL data to be transaction and successful otherwise if just one single data is lost then the whole thing will lead to bad results, where as in CAP you can have only a few successful nodes and it will still function at a reasonable speed. NoSQL typically follow the CAP theorem although some NoSQL such as Mongo can also integrate and follow ACID rules.

# **Tools for Design and dev**

There are many tools that you can use to develop NoSQL databases, but I will give a mini definition on just the two examples of no SQL I have been explaining. Mongo uses the Ides mongo community server and mongo compass both utilise first party IDEs as they were made by themselves. And Berkeley utilises a third-party IDE as while it was made by oracle it doesn’t have a dedicated first party IDE. However, it is possible that Mono also utilises a third party.

# **Conclusions**

Overall, when it comes to choosing between SQL and NoSQL it all comes down to both benefits and preference especially when doing a project, as explained SQL has its own benefits but so does NoSQL such as when coding in SQL to combine tables SQL joins are harder to do (with the inner Join) so embedded documents are easier as you just reference a document to another. So there might be more of an inclusion to use NoSQL.

# References

AI, n.d. *Linkedin.* [Online]   
Available at: https://www.linkedin.com/advice/0/what-current-trends-innovations-database#:~:text=History%20of%20DBMS,and%20relationships%20of%20data%20beforehand.  
[Accessed 30 11 2023].

Anon., n.d. *Erwin.* [Online]   
Available at: https://www.erwin.com/solutions/data-modeling

Anon., n.d. *Oracle.* [Online]   
Available at: https://docs.oracle.com/database/bdb181/BDBSG/introducing-oracle-berkeley-db-security.htm#  
[Accessed 7 12 2023].

Foote, K. D., 2021. *DATAVERSITY.* [Online]   
Available at: https://www.dataversity.net/brief-history-database-management/  
[Accessed 1 12 2023].

Grippa, V., 2023. *PERCONA.* [Online]   
Available at: https://www.percona.com/blog/securing-mongodb-top-five-security-concerns/#  
[Accessed 07 12 2023].

Hughes, G., n.d. *Canvas.* [Online]   
Available at: https://canvas.ljmu.ac.uk/courses/90615/files/5981267?module\_item\_id=1606377  
[Accessed 30 11 2023].

Kumar, B., 2020. *Medium.* [Online]   
Available at: https://medium.com/@kumar.barmanand/cap-theorem-and-nosql-databases-589e26e15905  
[Accessed 08 12 2023].

McQuillan, R., 2023. *Budibase.* [Online]   
Available at: https://budibase.com/blog/data/cap-vs-acid/  
[Accessed 08 21 2023].

Mongo, N/A. *MongoDB.* [Online]   
Available at: https://www.mongodb.com/languages#  
[Accessed 14 12 2023].

Monnappa, A., 2022. *simplilearn.* [Online]   
Available at: https://www.simplilearn.com/rise-of-nosql-and-why-it-should-matter-to-you-article#the\_types\_of\_nosql  
[Accessed 1 12 2023].

N/A, n.d. *Mongo.* [Online]   
Available at: https://www.mongodb.com/atlas/database  
[Accessed 30 11 2023].

Papiernik, M., 2021. *DigitalOcean.* [Online]   
Available at: https://www.digitalocean.com/community/tutorials/how-to-use-indexes-in-mongodb  
[Accessed 7 12 2023].

Smallcombe, M., 2023. *Integrate.io.* [Online]   
Available at: https://www.integrate.io/blog/the-sql-vs-nosql-difference/  
[Accessed 1 12 2023].

Suszterova, S., 2023. *GoodData.* [Online]   
Available at: https://www.gooddata.com/blog/what-a-data-model/  
[Accessed 27 11 2023].

Upadhyay, P., 2022. *Linkedin.* [Online]   
Available at: https://www.linkedin.com/pulse/birth-rise-nosql-pratima-upadhyay  
[Accessed 1 12 2023].

Wang, C., 2022. *Fivetran.* [Online]   
Available at: https://www.fivetran.com/learn/what-is-a-database-schema  
[Accessed 27 11 2023].

# Diagrams/pictures:

[1] <https://www.gooddata.com/img/blog/_2000xauto/data-model-components.png>

[2.1-2.3] took from my own coursework from last year’s data modelling with a little change.

[3] <https://www.gooddata.com/blog/physical-vs-logical-data-model/>

[4-15] all snippets of codes that I have personally made.

[16-19] same as 4-15

[20-] <https://www.linkedin.com/pulse/birth-rise-nosql-pratima-upadhyay>

[21-23] are all snippets I made to demonstrate CRUD and queries in mongo.

[24-26] snippets of the software visio code