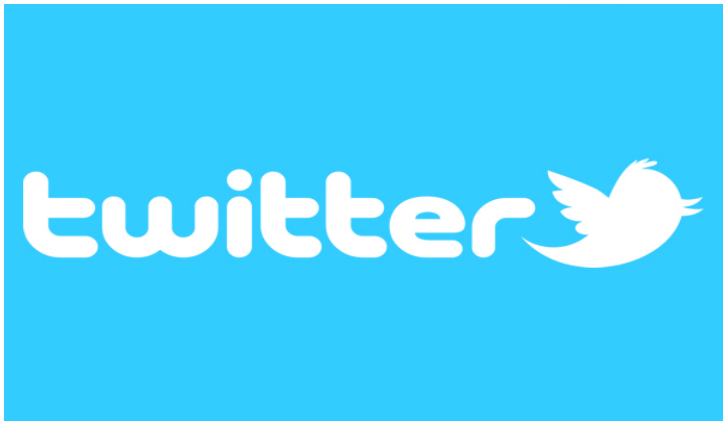
Introduction to NoSQL

COMP23111 – Database Systems

Gareth Henshall
Department of Computer Science

Introduction to NoSQL Systems

Who Uses Not Only SQL (NOSQL) Systems



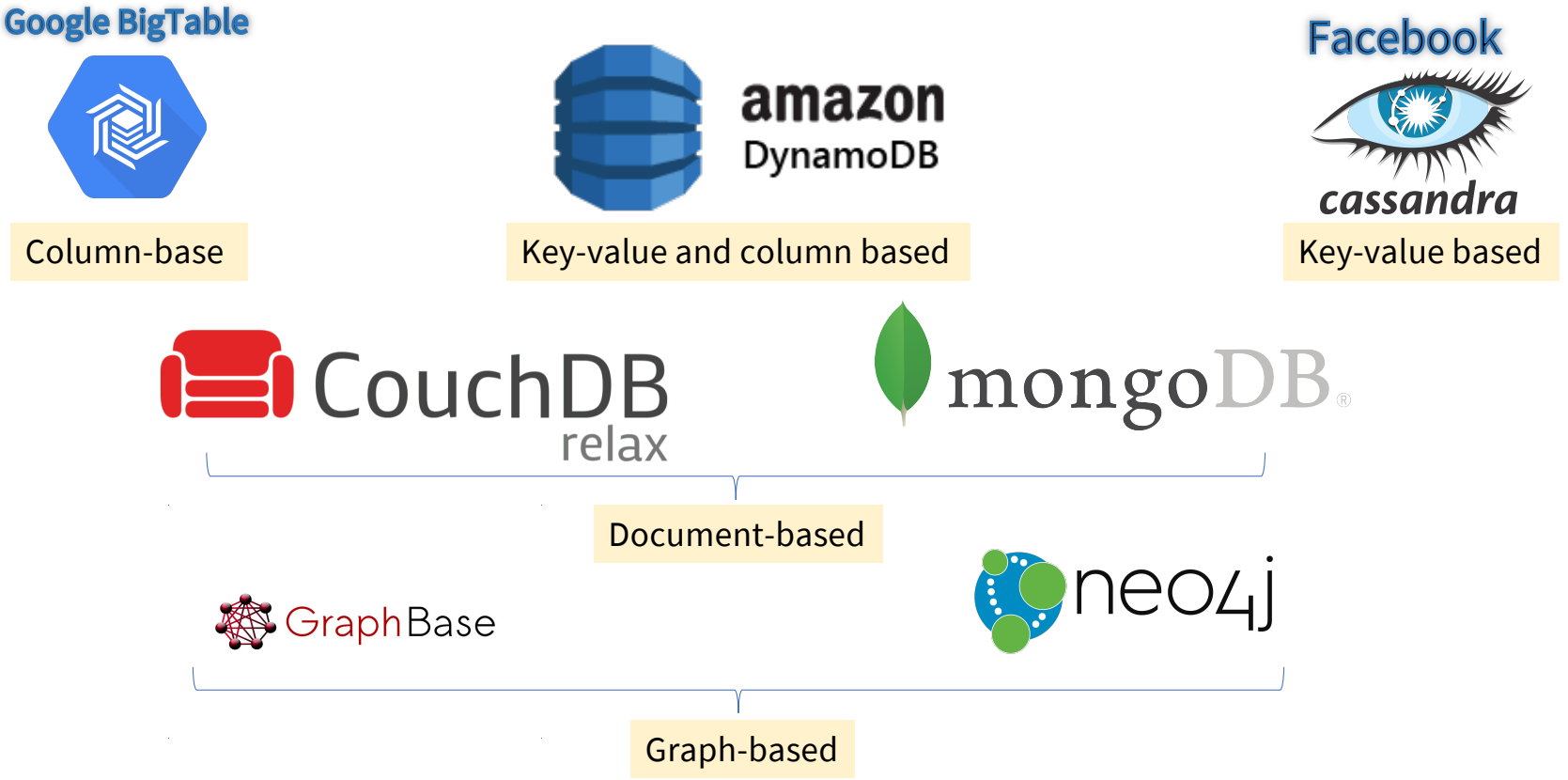
Introduction to NoSQL Systems

Emergence of NoSQL Systems

- The need by organisations to **store vast** amount of data, for example, emails, tweets, posts, updates, pictures, etc.
- SQL **service overload**, such as **powerful query language**, **concurrency control**, etc. which most of these applications do not need.
- Hence, Some of these **organisations** developed their own systems referred to as **NoSQL** systems to effectively manage these **vast** amount of data.

Introduction to NoSQL Systems

Emergence of NoSQL Systems (Cont.)



Introduction to NoSQL Systems

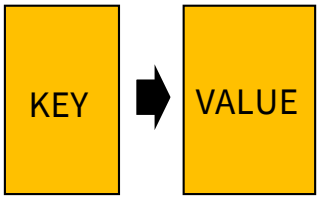
Characteristics Of NoSQL Systems

- NoSQL Characteristics Related to Distributed Database and Distributed Systems
  - Scalability (horizontal scalability)
  - Availability, Replication and Eventual Consistency
  - Replication Model (master-slave replication & master-master replication)
  - Sharding of files
  - High-Performance Data Access
- NoSQL Characteristics Related to Data Models and Query Language
  - Not Requiring a Schema (Schemaless)
  - Less Powerful Query Language
  - Versioning

Introduction to NOSQL Systems

Category of NoSQL Systems

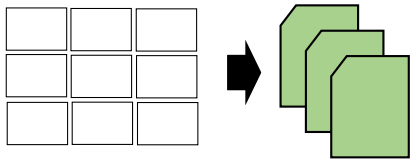
Key-value based NoSQL



Example: Riak, Redis server, Scalaris

Possess a simple data model based on fast access key to value associated with the key. The value can be record or object or document or complex data structure. Works best for shopping cart contents, etc.

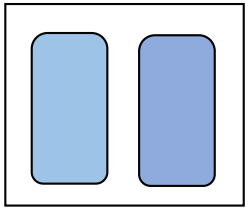
Document-based NoSQL



Example: **MongoDB**, CouchDB, OrientDB

Stores data in form of documents using well known data format, such as JSON(JavaScript object Notation) accessible via unique document "id". Mostly used for CMS, blogging, eComerse, etc.

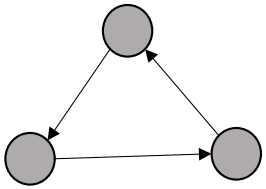
Column-based NoSQL



Example: BigTable, Cassandra, Hbase

Partition table by column into column family, where each column is stored in its own file. Widely used for for managing large data storage, such as data warehouse, etc.

Graph-based NoSQL



Example: Neo4J, InfoGrid, FlockDB

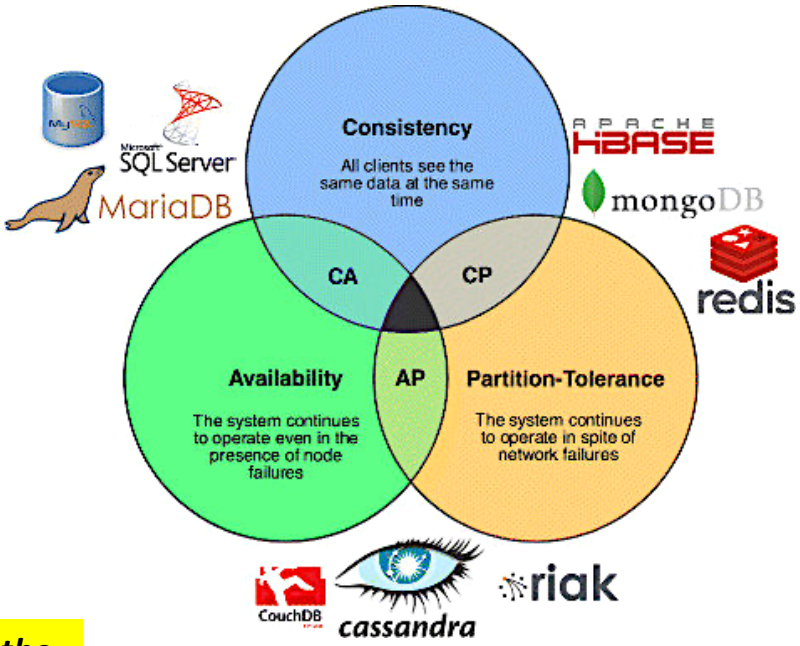
Stores entities as nodes and relation between entities as edges, where every node and edge has a unique identifier and can be traversed using path expression. Mostly used for social network, etc.

The CAP Theorem

Consistency, Availability and Partition Tolerance

- **Consistency** implies all nodes should contain same copies of replicated data item visible for various transactions.
- **Availability** implies that the system should be consistently available for read and write operations.
- **Partition tolerance** implies that the system should be continually available in situations where the system is partition by network fault

Note: The cap theorem states that it is not possible to guarantee all the desirable properties – consistency, availability and partition tolerance



Ref []

The CAP Theorem

Eventual Consistency

- In a NoSQL distributed database, a **weaker** consistency is often acceptable, while guaranteeing **availability** and **partition** tolerance - where the database will **eventually be consistent**.

- Hence the **BASE** property is used instead of the **ACID** property in centralised database.

- BASE: **B**asically **A**vailable, **S**oft state, **E**ventual consistency
  - **B**asically **A**vailable means the DDB is available following the CAP theorem
  - **S**oft state means the system's state may change even transaction execution
  - **E**ventual constancy implies that the system will become consistent over time

Document-Based NoSQL Systems: MongoDB

- **MongoDB** is a **general purpose**, **document-based**, **distributed database** built for **modern application** developers and for the **cloud** era.

- **MongoDB** is developed by **MongoDB Inc** and licensed under the Server Side Public License (SSPL).

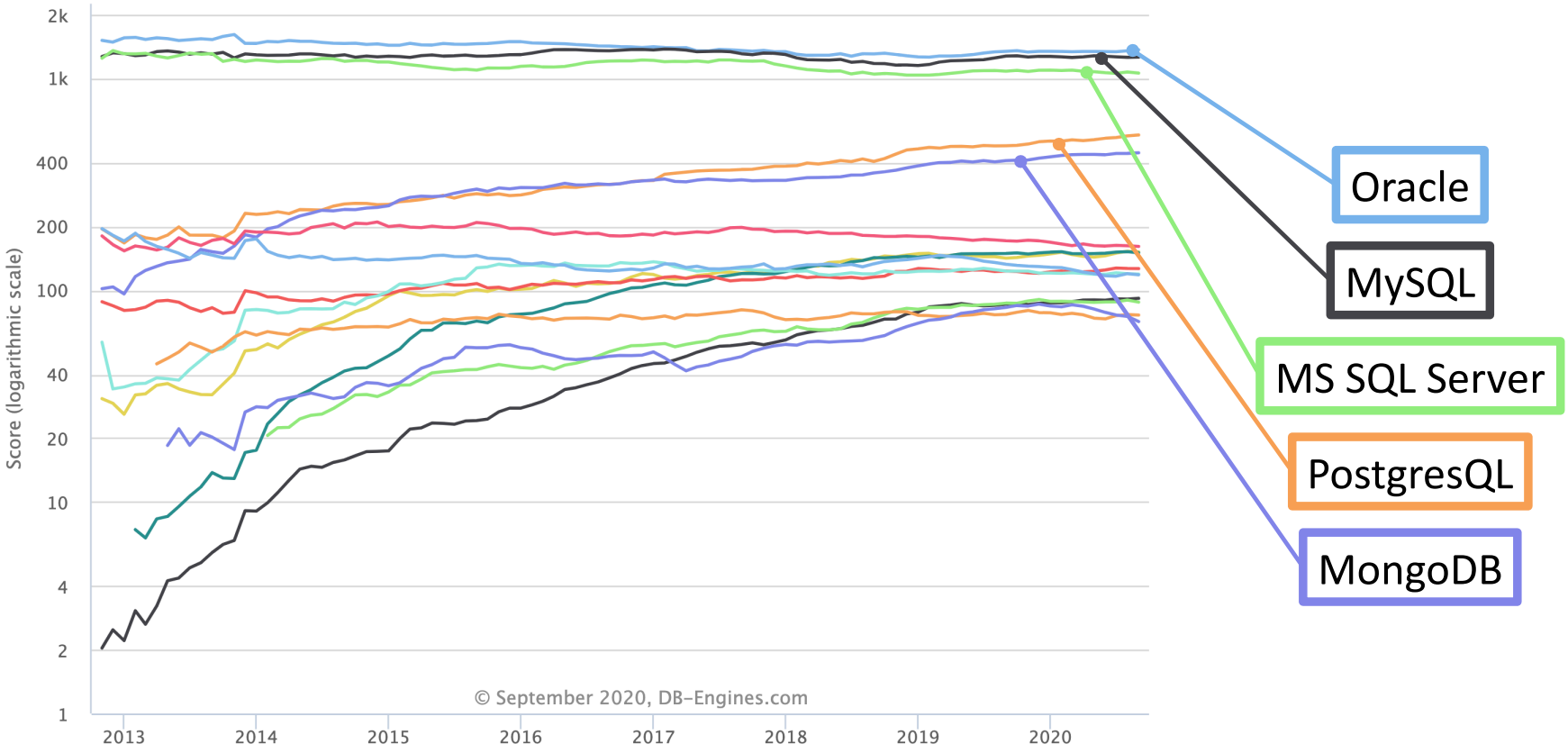
DB Systems Rankings

358 systems in ranking, September 2020

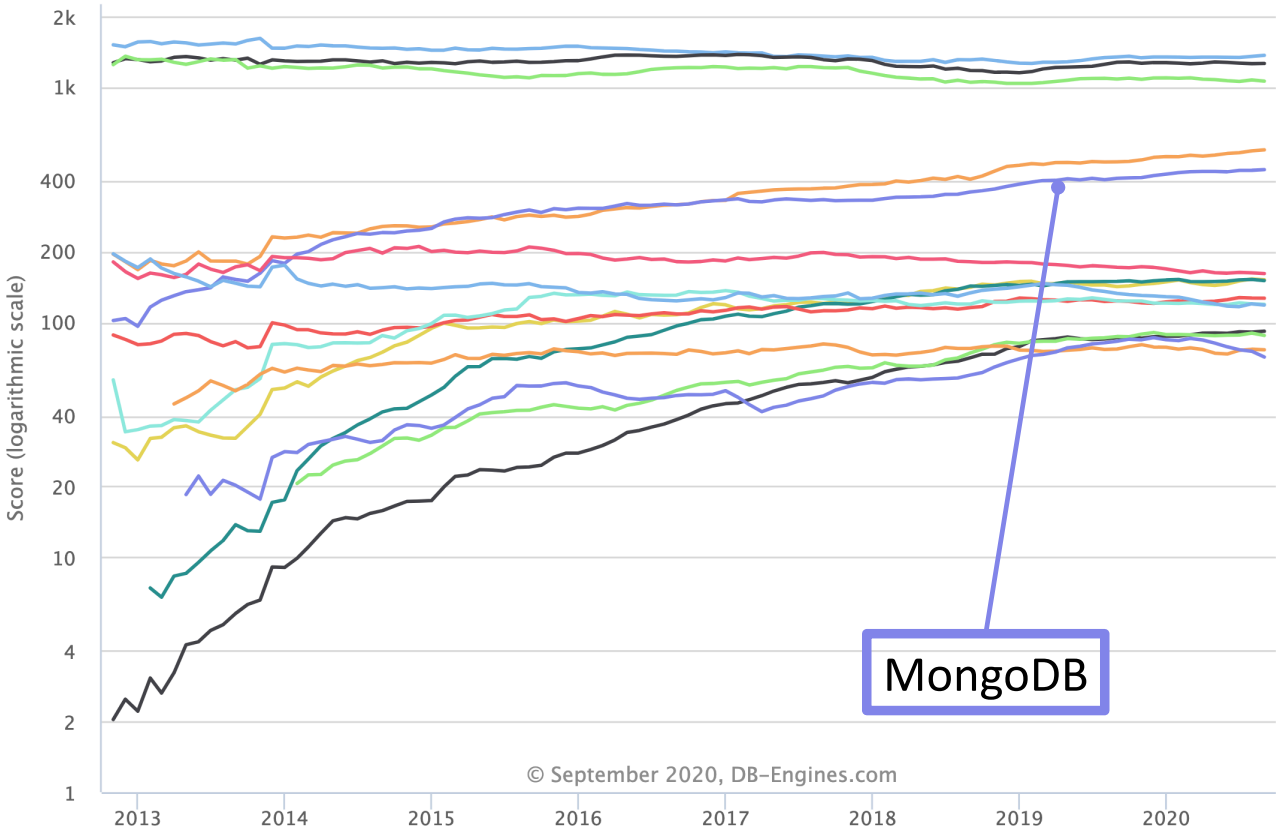
Rank			DBMS	Database Model	Score		
Sep 2020	Aug 2020	Sep 2019			Sep 2020	Aug 2020	Sep 2019
1.	1.	1.	Oracle	Relational, Multi-model	1369.36	+14.21	+22.71
2.	2.	2.	MySQL	Relational, Multi-model	1264.25	+2.67	-14.83
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1062.76	-13.12	-22.30
4.	4.	4.	PostgreSQL	Relational, Multi-model	542.29	+5.52	+60.04
5.	5.	5.	MongoDB	Document, Multi-model	446.48	+2.92	+36.42
6.	6.	6.	IBM Db2	Relational, Multi-model	161.24	-1.21	-10.32
7.	7.	↑ 8.	Redis	Key-value, Multi-model	151.86	-1.02	+9.95
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	150.50	-1.82	+1.23
9.	9.	↑ 11.	SQLite	Relational	126.68	-0.14	+3.31
10.	↑ 11.	10.	Cassandra	Wide column	119.18	-0.66	-4.22

https://db-engines.com/en/ranking

DB Rankings 2013 - 2020



DB Rankings 2013 - 2020



- Popularity
- High performance
- High availability
- Horizontal Scalability
- Rich query language
- Support for Multiple Storage Engines

Document-Based NoSQL Systems: MongoDB

Terminology and Concepts: SQL and MongoDB

SQL Terms/ Concepts	MongoDB Terms/ Concepts
database	database
table	collection
row	document
column	field
index	index
table joins	$lookup
primary key	primary key

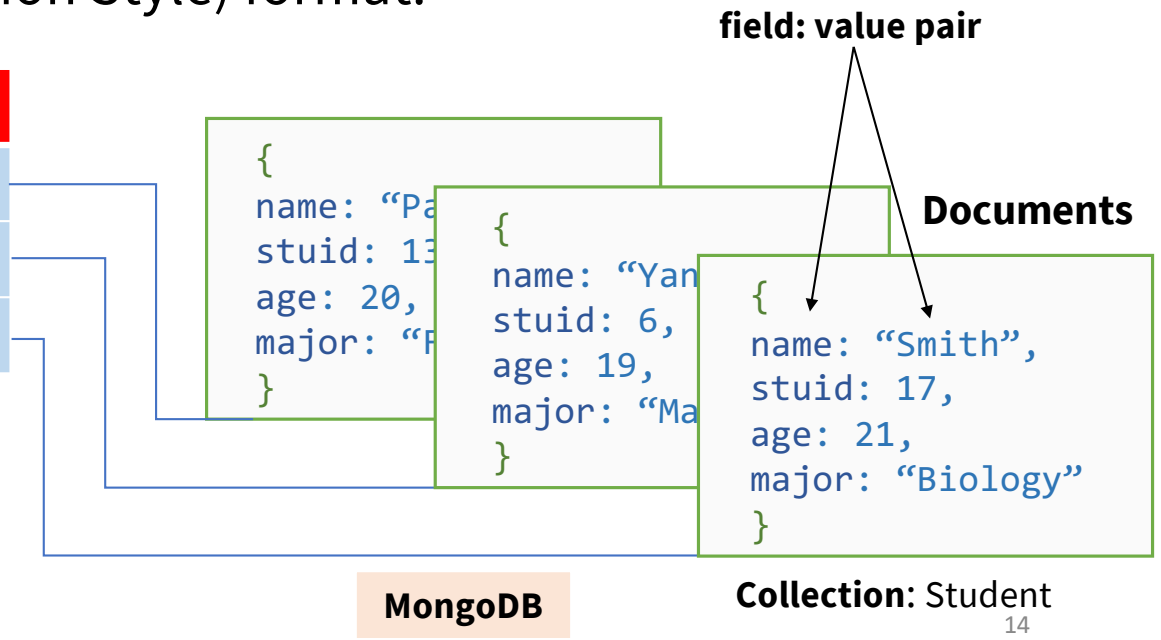
MongoDB Database and Collection

In MongoDB, **database** holds **collection** of documents which are in **JSON-style** (JavaScript Object Notation Style) format.

name	stuid	age	major
Smith	17	21	Biology
Yang	6	19	Maths
Patel	13	20	French

SQL

Table: Student



MongoDB Database and Collection

Explicitly create a Database

```
db.createCollection(<name>, <options>)

db.createCollection("myNewDB")
```

The `myNewDB` explicitly creates a new database if it does not already exist.

Create a Collection

```
db.myNewCollection1.insertOne({name: "Patel"})
```

The `myNewCollection` creates a new collection if it does not already exist.

Create Database and Collection

```
use myNewDB

db.myNewCollection2.insertOne({name: “Smith”})
```

The `insertOne()` operation creates both the database `myNewDB` and the collection `myNewCollection` if they do not already exist.

MongoDB Document

BSON and JSON

✍In MongoDB, **database** holds **collection** of documents which are in **JSON-style** (JavaScript Object Notation Style) format.

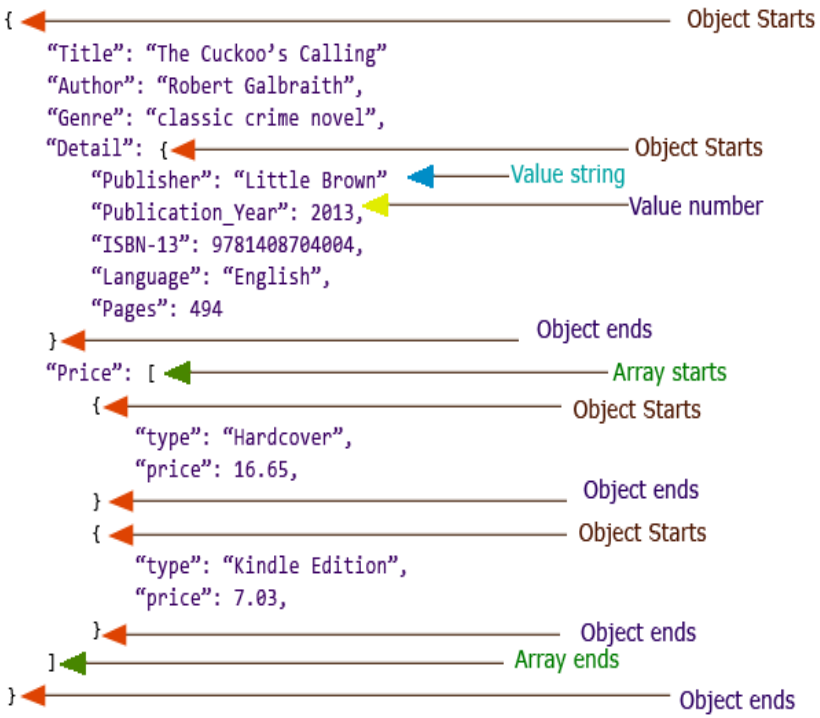
✍**JSON** is an open, **human** and **machine-readable** standard to transmit data objects consisting of **attributes-value pairs**.

✍MongoDB represents JSON documents in **binary-encoded** format called **Binary JSON** (BSON - ***[bee · sahn]***) behind the scene.

MongoDB Document

JavaScript Object Notation Structure: Example

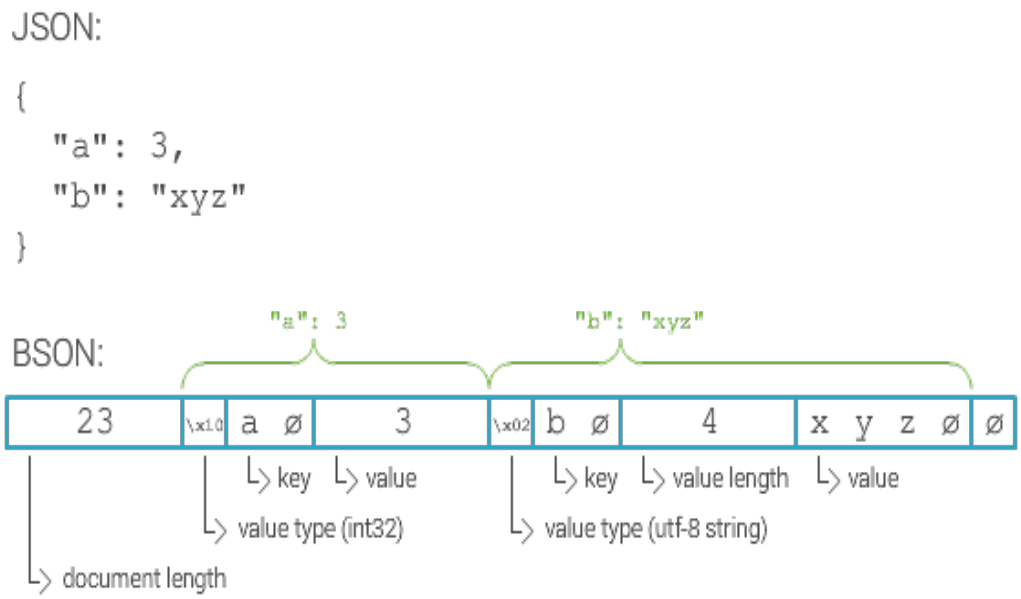
- Easy for **humans** to write and read, and easy for **computers** to parse and generate
- **Objects** can be nested
- Built on
  - name/value pairs
  - Ordered list of values



MongoDB Document

Binary JSON (BSON)

- **Binary-encoded** serialization of **JSON**-like docs
- Goals:
  - **Lightweight** (Keeping spatial over-head to a minimum)
  - **Traversable** (stores the length of values to find that specific key)
  - **Efficient** (decoding and encoding)



MongoDB Document

Binary JSON (BSON) (Cont.)

In BSON:

- Data is represented in **field / value** pairs
- A field/value pair consists of a field name followed by a colon, followed by a value:

Example:

```
name: “Joseph"
```

- Fields are separated by commas

Example:

```
name: “Joseph”, “course”: “Databases”, score: 80
```

- Curly braces hold objects (documents)

Example:

```
{name: “Joseph”, “course”: “Databases”, score: 80}
```

MongoDB Document

Binary JSON (BSON) (Cont.)

- **Embedded** document

Example:
```
{name: “Joseph”, courses: {course1: “databases”, course2: “programming”}}
```

The field **courses** holds an **embedded document** containing the fields **course1** and **course2**

- An **array** is stored in brackets []

Example:
```
{name: “Joseph”, courses: [“databases”, “programming”]}
```

- An **array** of embedded document contains documents embedded in the array

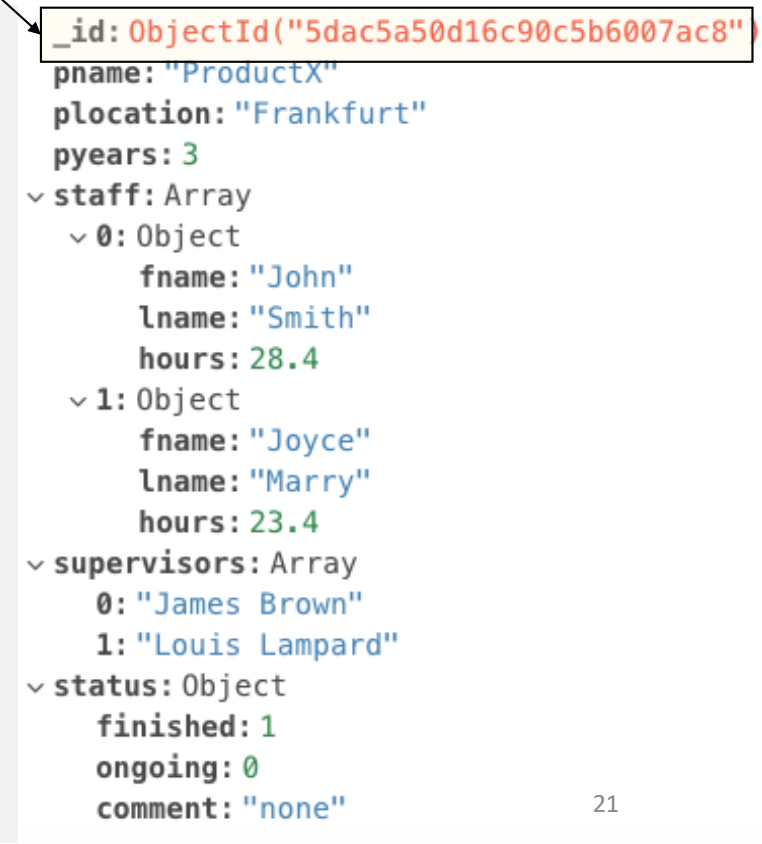
Example:
```
{name: “Joseph”, courses: [{course1: “databases”, course2: “programming”}]}
```

MongoDB Document

Document Structure: Example

```
db.projectCollection.insertOne(
{
   pname: "ProductX",
   plocation: "Frankfurt",
   staff: [
        {fname: "John",lname: "Smith", hours: 28.4},
        {fname: "Joyce", lname: "Marry", hours: 23.4}
        ],
   supervisors:["James Brown", "Louis Lampard"],
   status: {finished: 1, ongoing: 0, comment: "none"}
})
```

Holds the objectid



MongoDB Document

Document Structure: Example

```
db.projectCollection.insertOne(
{
   pname: "ProductX",
   plocation: "Frankfurt",
   pyears: 3
   staff: [
         {fname: "John",lname: "Smith", hours: 28.4},
         {fname: "Joyce", lname: "Marry", hours: 23.4}
         ],
   supervisors:["James Brown", "Louis Lampard"],
   status: {finished: 1, ongoing: 0, comment: "none"}
})
```

```
{ "_id" : ObjectId("5dabb758d16c90c5b6007ac4"),
"pname": "ProductX",
"plocation" : "Frankfurt",
“pyears”: 3
"staff" : [
{ "fname" : "John", "lname" : "Smith", "hours" : 28.4 },
{ "fname" : "Joyce", "lname" : "Marry", "hours" : 23.4 } ],
"supervisors" : [ "James Brown", "Louis Lampard" ],
"status" : { "finished" : 1, "ongoing" : 0, "comment" :
"none" } }
```

BSON

MongoDB Document

The _id Field

- Each document in a collection requires a unique **_id** field
- The **_id** field acts as a primary key to the collection
- If an inserted document omits the **_id** field, an **Objectid** is automatically generated for the **_id** field
- The **_id** field is always the first field in the document
- The **_id** field may contain BSON data type except arrays

MongoDB CRUD Operations

Query Operators

Name	Description
`$eq`	Matches values that are equal to a specified value.
`$gt`	Matches values that are greater than a specified value.
`$gte`	Matches values that are greater than or equal to a specified value.
`$in`	Matches any of the values specified in an array.
`$lt`	Matches values that are less than a specified value.
`$lte`	Matches values that are less than or equal to a specified value.
`$ne`	Matches all values that are not equal to a specified value.
`$nin`	Matches none of the values specified in an array.

MongoDB CRUD Operations

Create Operations

✍Create or insert operations add **new document** to a **collection**

✍If the collection does not currently **exist**, the insert operations **create** it

✍The following **methods** are used to insert documents into a collection:

`db.collectionName.insertOne(<document>)`	Insert a **single** document into a collection
`db.collectionName.insertMany(<documents>)`	Insert a **multiple** document into a collection
`db.collectionName.insert(<documents>)`	Insert a **single** or **multiple** document into a collection

MongoDB CRUD Operations

Read Operations

✍Read operation retrieves documents from a collection – queries a collection for documents

✍The following **methods** are used to read document in a collection

```
db.collectionName.find(<query> <projection>)
```

query specifies selection critaria using query operators, and **projection** specifies the **field** to return in the **document**

```
db.projectCollection.find( { “pyears”: { $gt:3 } }, { staff:1, supervisors:1 })
```

Displays all **staff** and **supervisors** who **work** on or **supervise** projects that are **over three years**

```
db.projectCollection.find( {“staff.hours”: { $gt:18 }} ,{staff:1} )
```

Displays all **staff** who has worked for over **18** hours on any **project**

MongoDB CRUD Operations

Update Operations

✍Modify existing document in a collection

✍The following methods are used to update documents in a collection

```
db.collectionName.updateMany (<filter>, <update action>, <options>)
```

filter denotes the selection criteria for the update, **update action** implies the modification to apply, and **options** implies additional optional actions that can be assigned to the operation .

```
db.projectCollection.updateMany(
   { "pyears": { $gte: 5 } },
   {$set: { "status.finished": 1, "status.ongoing": 0}}
)
```

Updates the **finished** and **ongoing** status of all **projects** whose number of **years** is **greater** than or **equal** to **five** to **1** and **0** respectively

MongoDB CRUD Operations

Delete Operations

✍Delete operations remove documents from a collection.

✍The following methods are used to remove documents from a collection

```
db.collectionName.deleteMany (<filter>, <options>)
```

filter specifies deletion criteria using query operators, and **options** specifies additional optional actions that can be assigned to the operation.

```
db.projectCollection.deleteMany(
      { "status.finished" :1, "status.ongoing": 0}
)
```

Deletes all **projects** whose **finished** status is **1** and **ongoing** status is **0**

MongoDB CRUD Operations

More Operations Example

```
SELECT staff, supervisors
FROM projectCollection
WHERE pyears >=3 OR pyears <= 5
```

```
db.projectCollection.find(
{ $or: [{"pyears": { $gte:3 }},
{"pyears": { $lte:5 }} ]},
{ staff:1, supervisors:1 })
```

```
SELECT * FROM projectCollection
WHERE pyears >=5 AND
finished = 1 AND
Ongoing = 0
```

```
db.projectCollection.find(
{ $and: [{"pyears": { $gte:5 }},
{"status.finished": { $eq:1 }},
{"status.ongoing": { $eq:0 }} ]})
```

MongoDB CRUD Operations

Operations Output Sample 1

```
db.projectCollection.find(
{ $or: [{"pyears": { $gte:3 }},
{"pyears": { $lte:5 }} ]},
{ staff:1, supervisors:1 })
```

```
{
   "_id":ObjectId(   "5dac5a50d16c90c5b6007ac8"   ),
   "staff":[
      {
         "fname":"John",
         "lname":"Smith",
         "hours":28.4
      },
      {
         "fname":"Joyce",
         "lname":"Marry",
         "hours":23.4
      }
   ],
   "supervisors":[
      "James Brown",
      "Louis Lampard"
   ]
}
```

MongoDB CRUD Operations

Operations Output Sample 2

```
db.projectCollection.find(
{ $and: [{"pyears": { $gte:5 }},
{"status.finished": { $eq:1 }},
{"status.ongoing": { $eq:0 }} ]})
```

```
{
   "_id":ObjectId("5dac5bdcd16c90c5b6007aca"),
   "pname":"ProductY",
   "plocation":”London",
   "pyears":5,
   "staff":[
      {
         "fname":"Salah",
         "lname":"Hammed",
         "hours":89.4
      },
      {
         "fname":"Marry",
         "lname":"Fin",
         "hours":98.4
      }
   ],
   "supervisors":[
      "James Brown",
      "Louis Lampard"
   ],
   "status":{
      "finished":1,
      "ongoing":0,
      "comment":"none",
      "finish":1
   }
}
```