NoSQL Databases - MongoDB

Agenda

- What is NoSQL databases
- Different kinds of NoSQL databases
- What is MongoDB?
- Overview of MongoDB.
- MongoDB’s key features
- MongoDB’s core server and tools.
- Installing MongoDB
- Use cases and production deployment
- Data Types, Schema Design and Data Modelling

Why NoSQL?

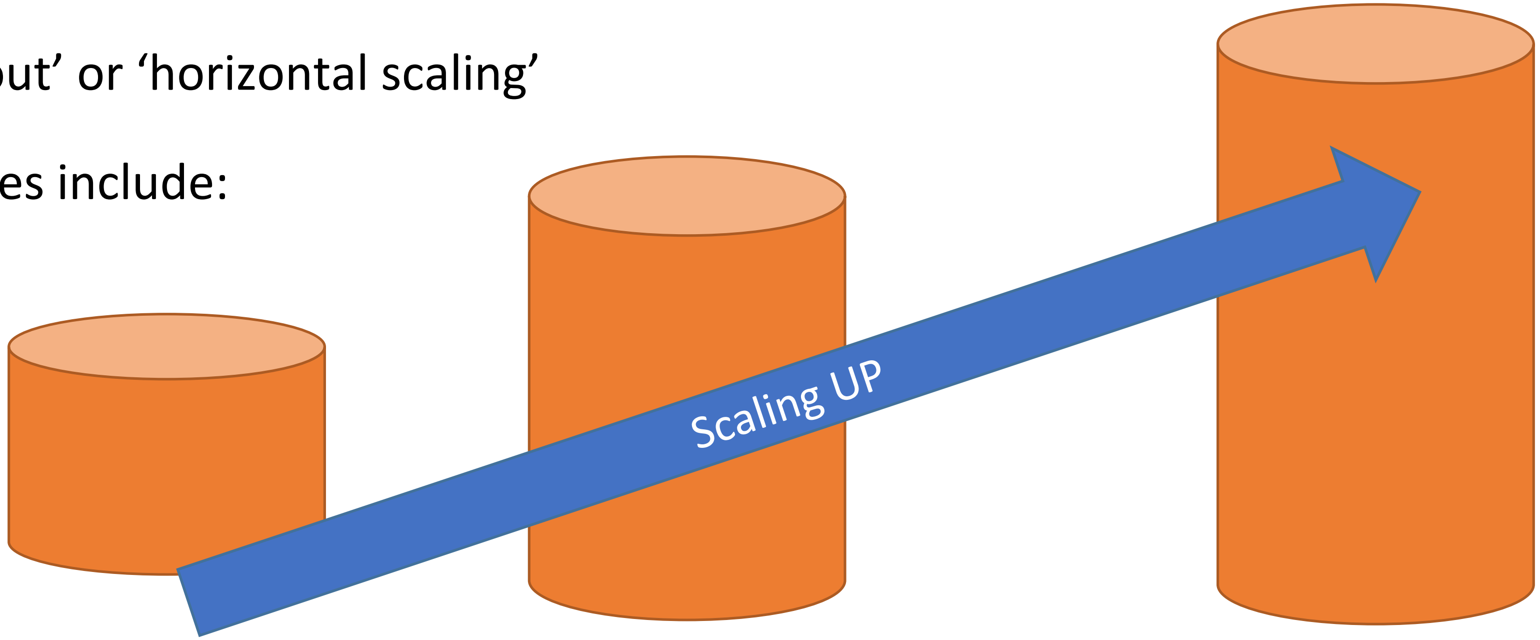
- Relational databases → mainstay of business
- Web-based applications caused spikes
- explosion of social media sites (Facebook, Twitter) with large data needs
- Example of such data : Personal user information, geo location data, social graphs , user generated contents , machine-logging data, sensor generated data etc
- rise of cloud-based solutions such as Amazon S3 (simple storage solution)

Challenges with RDBMS

- Hooking RDBMS to web-based application becomes trouble
- Developers begin to front RDBMS with memcache or integrate other caching mechanisms within the application (ie. Ehcache)
- As datasets grew, the simple memcache/MySQL model (for lower-cost startups) started to become problematic
- Hence, developers look forward to improve existing applications and develop new applications which can meet the needs of Big Data

Issue with Scaling Up

- Not possible to store and query when the dataset is just too big
- RDBMS were not designed to be distributed
- Began to look at multi-node database solutions
- Known as ‘scaling out’ or ‘horizontal scaling’
- Different approaches include:
- Master-slave
- Sharding



Different RDBMS Cluster Approach

Master-Slave

- Master handles all the write request because it must maintain the locks to guarantee roll back in case of failure.
- All reads goes to the replicated slave databases
- Reads from slave may be inconsistent as writes may not have been propagated down

Partition or sharding

- Scales well for both reads and writes
- Not transparent, application needs to be partition-aware
- Can no longer have relationships/joins across partitions
- Loss of referential integrity across shards

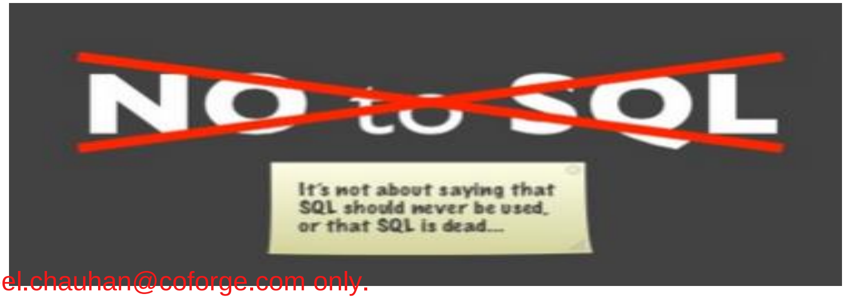
Other ways to scale RDBMS

- Multi-Master replication
- INSERT only, not UPDATES/DELETES
- No JOINs, thereby reducing query time
- This involves de-normalizing data
- In-memory databases

What is NoSQL?

Stands for Not Only SQL

- The term NOSQL was given by Carl Strozzi in 1998 to name his file-based database
- It was again re-introduced by Eric Evans when an event was organized to discuss open source distributed databases
- Eric clarified later that NoSQL does not mean “No to SQL”
- NoSQL means seeking alternatives for the relational databases specially for those use cases for which RDBMS are a bad fit.


~~NO to SQL~~
It's not about saying that SQL should never be used, or that SQL is dead...

Features of NoSQL

- non-relation based , no primary and foreign key relationship
- Either don’t require schema or provide flexible schema
- data are replicated to multiple nodes, data are partitioned too
- down nodes easily replaced
- no single point of failure
- horizontal scalable
- open-source software
- massive write performance
- fast key-value access

Types of NoSQL databases

1. Key-value
   Example: DynamoDB, Voldermort, Scalaris

2. Document-based
   Example: MongoDB, CouchDB

3. Column-based
   Example: BigTable, Cassandra, Hbase

4. Graph-based
   Example: Neo4J, InfoGrid

- “No-schema” is a common characteristics of most NOSQL storage systems
- Provide “flexible” data types

Benefits of NoSQL databases

- **High Scalability** : Ability to execute more and more queries and store more and more data without having any upper limit.
- **High Availability :** Ability to run Read/Write queries even when some servers are down

Disadvantage of NoSQL databases

- Don’t fully support relational features
- Normalization can’t be used, so No JOIN Queries
- no referential integrity constraints
- Non-Availability of SQL query language
- More programing is needed to work with these DBs
- NoSQL don’t follow and provide ACID properties
- They provide fewer guarantees by following CAP theorem
- No easy integration with applications that needs JDBC or ODBC drivers

Who are users of NoSQL

- All the e-commerce companies such as Flipkart, Amazon, Walmart etc use
- NoSQL database for storing huge volume of data and large amount of request from user.
- All the Cab aggregator companies such as OLA and UBER
- The mobile app companies like Kobo and Playtika
- Consumer appliances companies such as LG, Samsung etc use NoSQL for IOT use cases
- NOSQL has been used by some of the mobile gaming companies like, electronic arts, zynga and tencent for Social Gaming use cases

Tradeoff in Cluster Databases

ACID

A DBMS is expected to support “ACID transactions,” processes that are:
Atomicity: either the whole operation is performed, or none is
Consistency: Whenever clients reads the data it will be consistent
Isolation: one operation or transaction at a time by holding locks
Durability: once data is committed, DBs will safely keep this data even in the case of process crash

CAP

Consistency: all the nodes on cluster has the same copies
Availability: cluster always accepts reads and writes even if some nodes are down
Partition tolerance: guaranteed properties are maintained even if cluster gets divided into 2 or more partitions because of network failures.

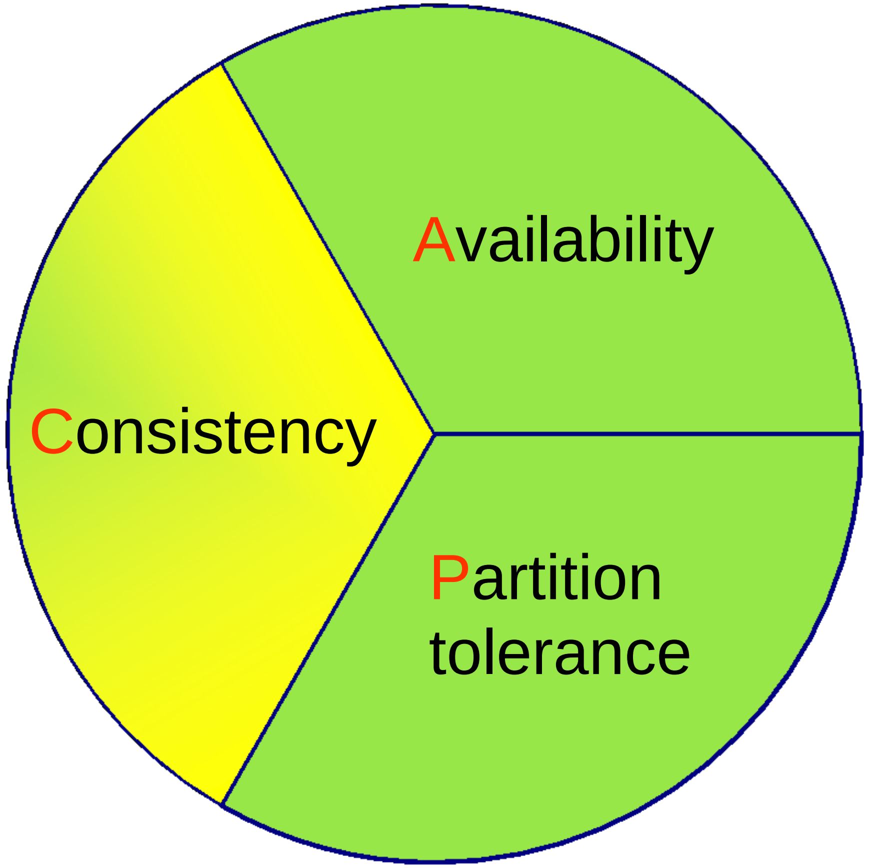
CAP Theorem

Brewer’s CAP Theorem:

- For any distributed cluster, it is “impossible” to guarantee simultaneously all of these three properties
- You can have at most two of these three properties for any distributed shared data system
- Large Cluster will “partition” at some point due to network failures :

  If database is designed to provide partition tolerance, then Database by default will either provide C(Consistency) or A(Availability) but not the both .

CAP Theorem?



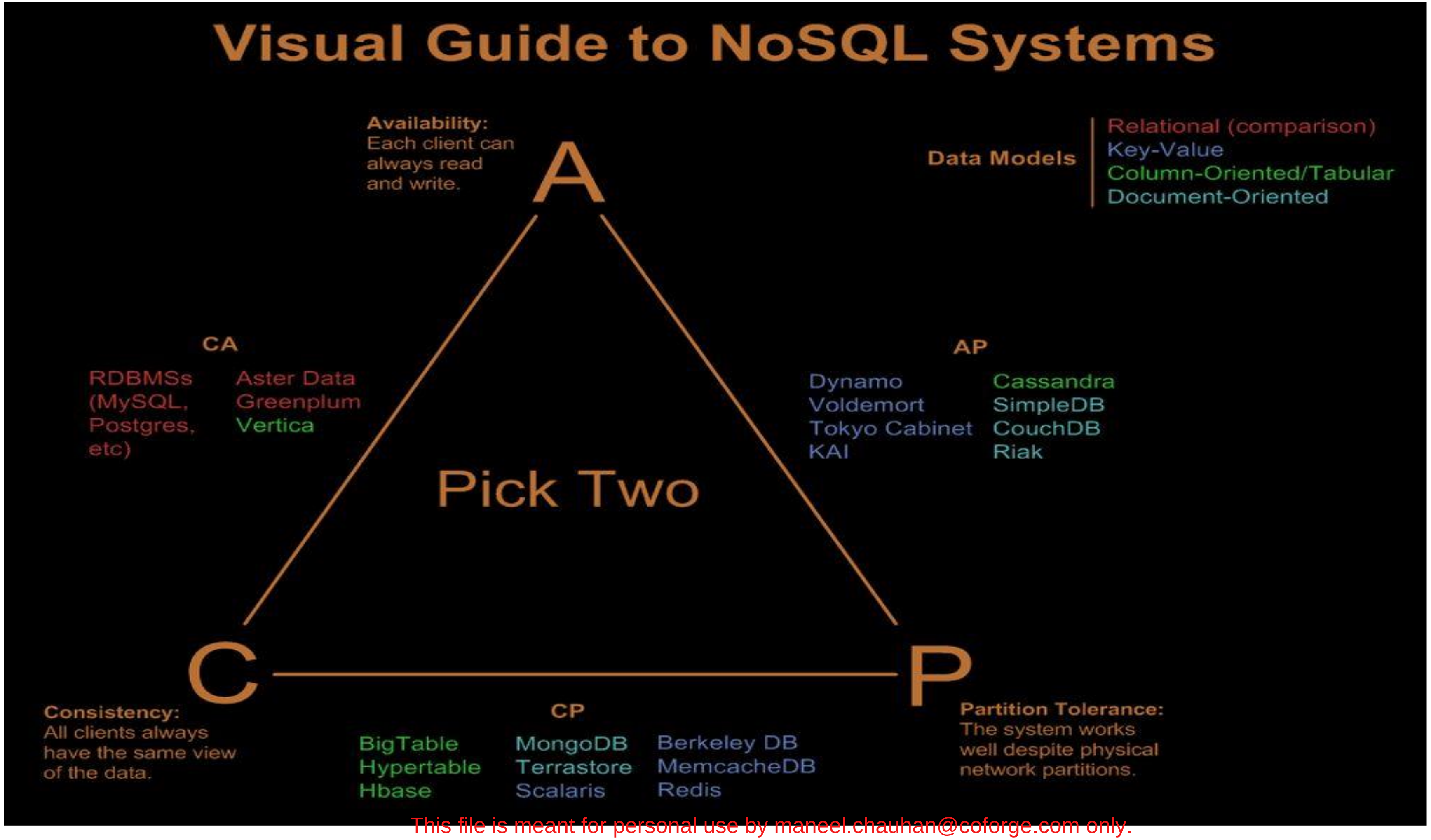
Consistency

All client always have the same view of the data

2 types of consistency:

1. Strong consistency – ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability)
2. Weak consistency – BASE (**B**asically **A**vailable **S**oft-state **E**ventual consistency)

CAP



Visual Guide to NoSQL Systems

Availability: Each client can always read and write.

Consistency: All clients always have the same view of the data.

Partition Tolerance: The system works well despite physical network partitions.

Data Models
- Relational (comparison)
- Key-Value
- Column-Oriented/Tabular
- Document-Oriented

CA

RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica

AP

Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak

CP

BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis

Key Value Databases

- Key-value stores are the simplest NoSQL databases.
- They are like a dictionary, stores every item as an attribute name (or "key"), together with its value.
- Examples of key-value stores are Riak and Voldemort.
- Some key-value stores, such as Redis, allow each value to have a type, such as "integer", which adds functionality.
- Key-value NoSQL databases are ideal for database for lookup queries with extremely quick and optimized retrieval

Document Databases

{Name:“Michael", Address:“FlatNo 112,Waterfront Aparment ,NYK,USA”,Grandchildren: {Claire: "7", Barbara: "6", "Magda: "3", "Kirsten: "1", "Otis: "3", Richard: "1“} Phones: [“123-456-7890”, “234-567-8963”] }

Example: MongoDb,CouchBase etc

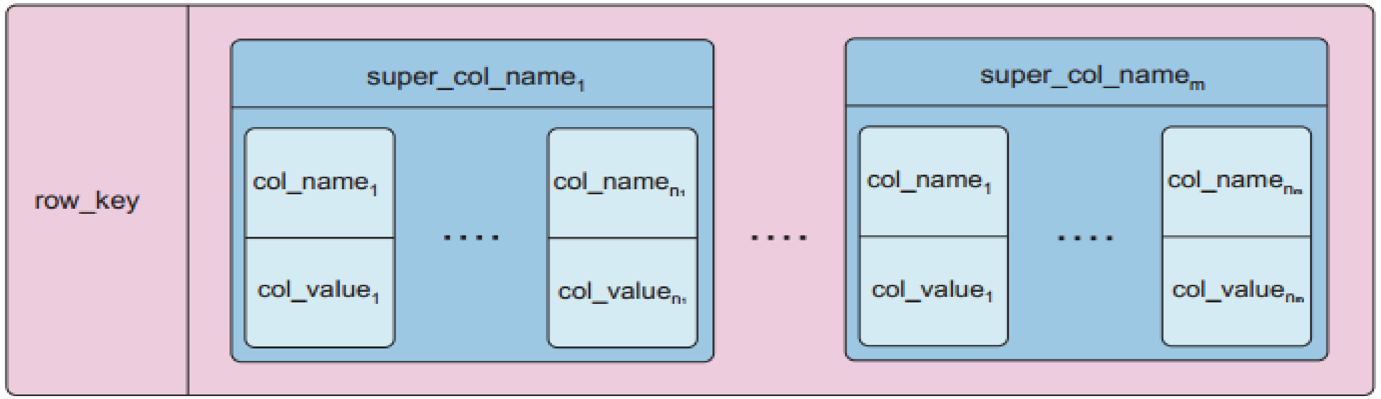
Columnar Databases

Based on Google’s BigTable paper, they are like column oriented relational databases (store data in column order)

Tables similarly to RDBMS, but handle semi-structured

Data model:

- Collection of Column Families
- Column family = (key, value) where value = set of **related** columns (standard, super)
- indexed by *row key*, *column key* and *timestamp*



row_key

super_col_name$_1$

col_name$_1$ col_value$_1$ …. col_name$_{n1}$ col_value$_{n1}$

….

super_col_name$_m$

col_name$_1$ col_value$_1$ …. col_name$_{nm}$ col_value$_{nm}$

Columnar Databases

- One column family can have variable numbers of columns
- Cells within a column family are sorted “physically”
- Very sparse, most cells have null values

Comparison: RDBMS vs column-based NOSQL

Query on multiple tables

RDBMS: must fetch data from several places on disk and glue together

Column-based NOSQL: only fetch column families of those columns that are required by a query (all columns in a column family are stored together on the disk, so multiple rows can be retrieved in one read operation data locality)

Example : Hbase, Cassandra, HyperTable etc

Graph Databases

- Focus on modeling the structure of data (interconnectivity)
- Scales to the complexity of data
- Inspired by mathematical Graph Theory (G=(E,V))

Data model:

(Property Graph) nodes and edges

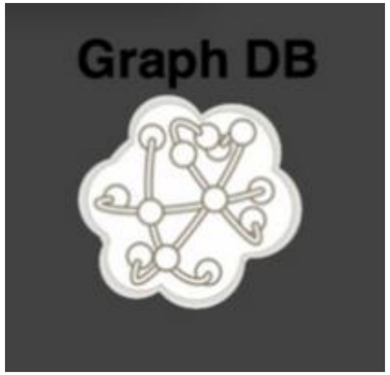
Nodes may have properties (including ID)

Edges may have labels or roles

- Single-step vs path expressions vs full recursion

Example:

Neo4j, FlockDB, Pregel, InfoGrid …



What is MongoDB?

- MongoDB is a document-oriented database
- MongoDB replaces the concept of a “row” with a more flexible model, the “document.”
- MongoDB stores data in the form of BSON(binary form of JSON)
- Document-oriented approach allows to store complex hierarchical relationships with a single record in the form of nested JSON
- This approach suits application developers of modern object-oriented languages as Java, Java Script based framework(AngularJS, RectJS), C++ etc
- MongoDB is also schema-free: it is not necessary to define collection attributes before writing data into it.
- This provide developers a lot of flexibility to handle evolving data models.

MongoDB Use Cases

- Personalization
- Mobile
- Internet of things
- Real time Analytics
- Web Applications
- Content Management
- Catalog
- Single View

What is JSON?

- JSON : JSON (JavaScript Object Notation) is a lightweight data-interchange format.It is easy for humans to read and write.

- JSON is a text format and language independent, It also uses object concept that are familiar to programmers of different object oriented languages such as Java,C#,C++ etc

- JSON supports all the basic data types you'd expect: numbers, strings, and boolean values, as well as arrays and hashes

- Document databases such as MongoDB use JSON documents in order to store records, just as tables and rows store records in a relational database

- A JSON database returns query results that can be easily parsed, with little or no transformation, directly by JavaScript and most popular programming languages – reducing the amount of logic you need to build into your application layer

Example of JSON

```
{
        "_id" : 1,
        "name" : { “GreatLearning"},
        "customers" : [ “Genpact", “Accenture", “Wipro", “Infosys" ],
        “courses" : [
                {       "name"   : “Data Science with SAS",
                            "domain" : “Statistics and Analytics "
                },
                { "name"   : “Big Data Specialization",
                  "domain" : “Hadoop eco-system analytics"
                }
        ]
}
```

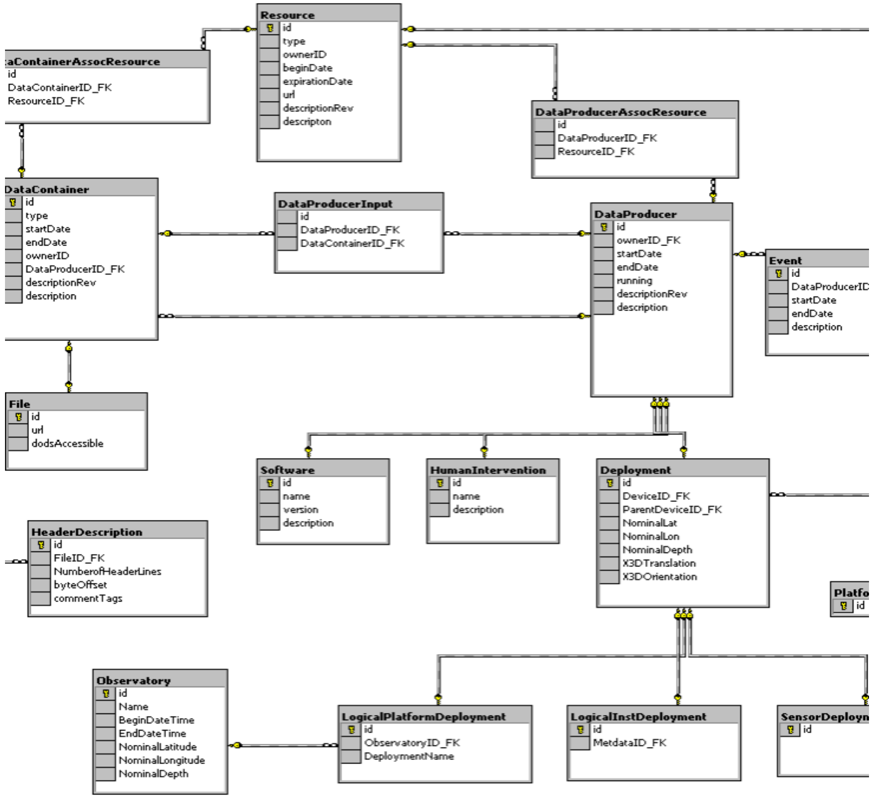
Why MongoDB stores data as BSON

- MongoDB stores data or document in binary-encoded format called BSON.
- BSON is a binary-encoded serialization of JSON-like documents.
- BSON supports the embedding of documents and arrays within other documents and arrays
- BSON also supports additional data types that are not part of the JSON spec such as BinData and Date data type
- BSON is designed to be lightweight , traversable and efficient

MongoDB Structure

RDBMS	MongoDB
Database	Database
Table, View	Collection
Row or Record	Document (JSON, BSON)
Column	Field
Index	Index
Join	Embedded Document
Foreign Key	Reference
Partition	Shard

MongoDB is easy to use



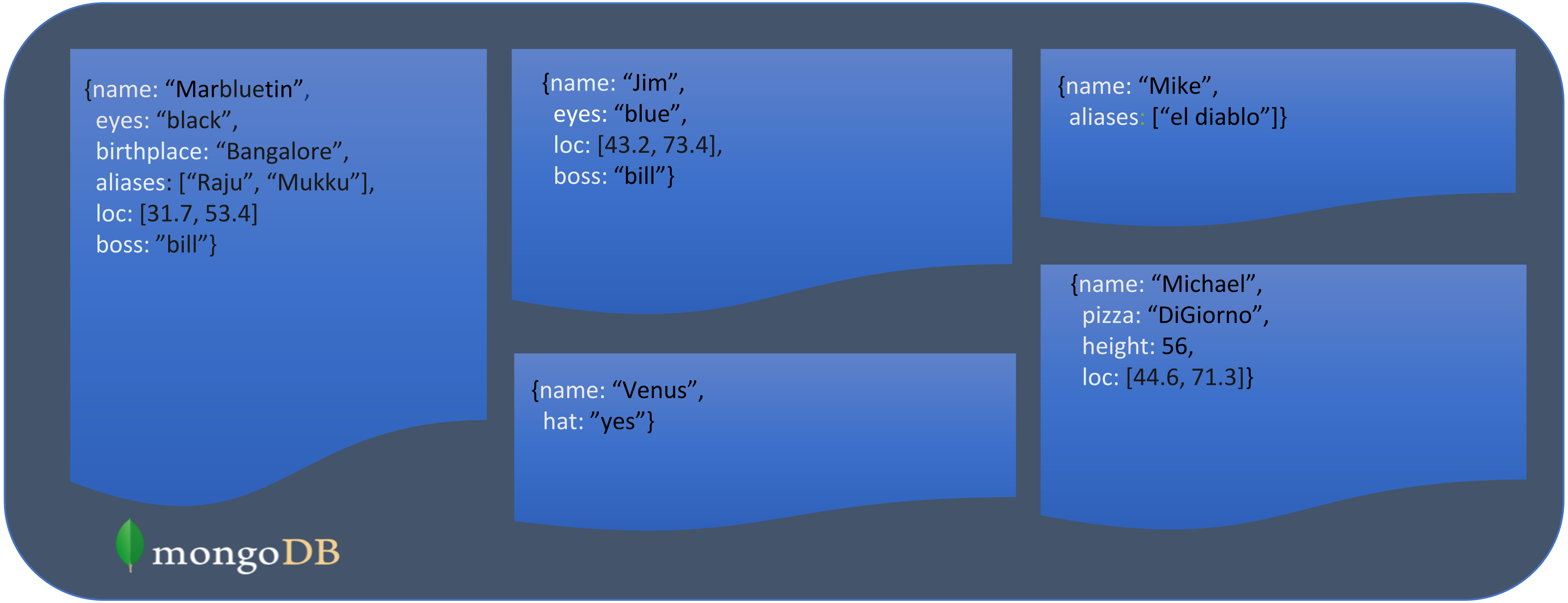
RDBMS ER Diagram

```
{
  title: ‘MongoDB’,
  contributors: [
    { name: ‘Mukesh Kumar’,
      email: ‘mukesh@greatlearning.com’ },
    { name: ‘Laxman L’,
      email: ‘laxman@gmail.com’ }
  ],
  model: {
    relational: false,
    awesome: true
  }
}
```

MongoDB’s nested model

Schema Free

- MongoDB does not need any pre-defined data schema
- Every document could have different data!



```
{name: “Marbluetin”,
  eyes: “black”,
  birthplace: “Bangalore”,
  aliases: [“Raju”, “Mukku”],
  loc: [31.7, 53.4]
  boss: ”bill”}
```

```
{name: “Jim”,
  eyes: “blue”,
  loc: [43.2, 73.4],
  boss: “bill”}
```

```
{name: “Mike”,
  aliases: [“el diablo”]}
```

```
{name: “Venus”,
  hat: ”yes”}
```

```
{name: “Michael”,
  pizza: “DiGiorno”,
  height: 56,
  loc: [44.6, 71.3]}
```

How to scale database?

- Applications data are growing at an incredible pace
- Due to Rapid pace of digitalization ,Advances in sensor technology, increases in available bandwidth, and the popularity of smart devices
- where even small scale applications need to store more data than many databases were meant to handle
- A terabyte of data, once considered huge data, is now commonplace
- Application developers face a difficult decision: how should they scale their databases to make their application scalable

MongoDB is designed for scale out

- MongoDB is designed to scale out from the beginning.
- Using Sharding, it can split up data across multiple servers
- Sharding in MongoDB, can balance data and load across a cluster, redistributing documents automatically
- This helps developers to focus on programming the application, not scaling it
- When they need more capacity, they can just add new shards to the cluster

MongoDB Features

- Ad hoc queries
- Secondary Indexes
- Replication
- Auto-Sharding
- Querying
- Fast In-Place Updates
- Aggregation
- Capped Collection

MongoDB Compass

MongoDB Compass is a powerful GUI for querying, aggregating, and analyzing your MongoDB data in a visual environment.

Compass is free to use and source available, and can be run on macOS, Windows, and Linux.

Download compass

https://downloads.mongodb.com/compass/mongodb-compass-1.39.4-win32-x64.exe

Refer to the below link to know more about Compass:
https://www.mongodb.com/try/download/compass

https://www.mongodb.com/docs/compass/current/install/

MongoDB Compass

With Compass we can visually Explore the Data.

Some tasks which Compass can help us accomplish, such as importing and managing data from an easy-to-navigate interface are:

- Import your data
- Query your data
- Create aggregation pipelines
- Run commands in the shell

Thank You