



MDB100

# Introduction to MongoDB & Atlas

MongoDB Database and Security



# Topics we cover

Why a new Database?

MongoDB Terminology

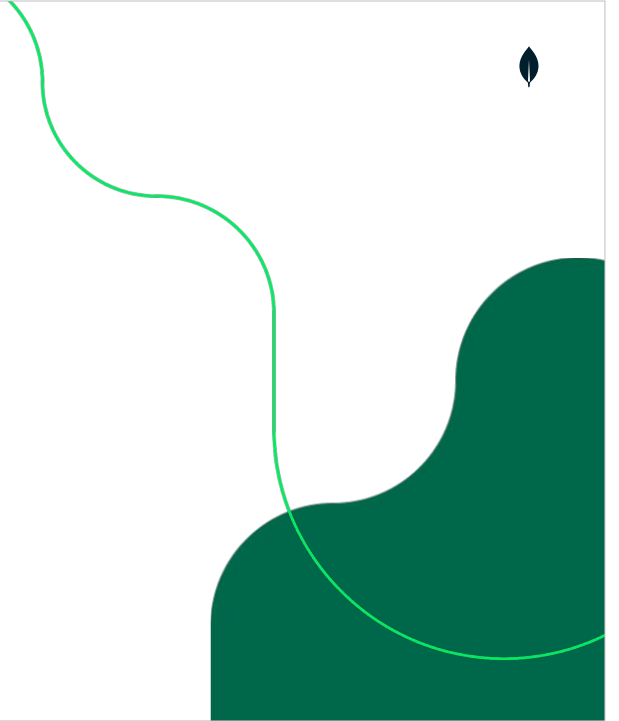
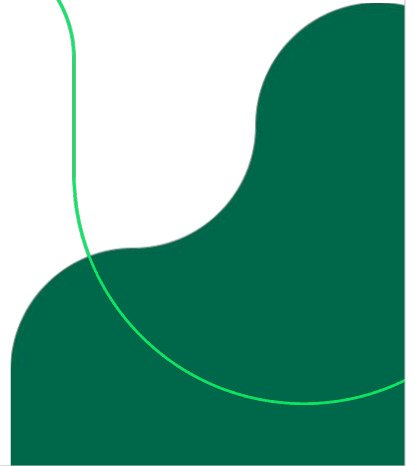
Benefits of MongoDB

When to use MongoDB

MongoDB and RDBMSs

The MongoDB Atlas Platform

# Why a New Database?





# Why a new Database?

The world is changing

Data

Volume, Velocity, Variety

Time

Iterative, Agile, Short Cycles

Risk

Always-on, Global, Scale

Cost

Open Source, Cloud, Pay by Usage

5

The way we write software has changed:

- **Data** - is now 'Big Data.' It is high in volume, it arrives faster; and even in a single system, it can have various forms and can change quickly.
- **Time** - Companies want software delivered fast and changed as soon as it has arrived - the current manner of Software Development is Agile and Rapid.
- **Risk** - 20 years ago, mainframes and specialized hardware were costly and were needed to provide the always-on capability. Now every system needs 100% uptime even while maintenance goes on.
- **Cost** - Flexible pay-as-you-go cloud services, with the ability to scale up, are the new norm. There has also been a shift to open source.



# MongoDB

- Modern Document-model database.
- Designed to back the modern-day business applications:
  - Developer and Operations oriented
  - Easy to scale horizontally
  - Business Critical
  - Lessons learned from 50 years of RDBMS

What two RDBMS functions are hard to do efficiently in a distributed system?

MongoDB was established in 2007 (as 10gen) by three entrepreneurs who had previously developed some of the internet's first advertising and shopping networks (DoubleClick)

The company originally started as a Platform-as-a-Service architecture based entirely on developing open source components.

They were unable to source a suitable database (frustration with the limitations of available databases at the time), so they built their own.

The company developed a document-oriented DB known today as MongoDB.

This was popular, so initially, they focussed on being a database company, returning to the cloud and then BaaS later.

It is a highly available and horizontally scalable database designed to be the primary database for modern business applications, focusing on the developer and DevOps audiences.

Fifty years of RDBMS has not been forgotten. The creators of MongoDB wanted to retain as much "Database" behavior as possible but as a distributed system.

Question - What two fundamental RDBMS features are hard to do efficiently in a distributed system?



# Relational Data Model

EmpID	Name	Dept	Title	Manager	Payband
9950	Dunham, Justin	500	1500	6531	C

EmpBenPlanID	EmpFK	PlanFK
1	9950	100
2	9950	200

PlanID	BenFK	Plan
100	1	PPO Plus
200	2	Standard

BenID	Benefit
1	Health
2	Dental

TitleID	Title
1500	Product Manager

DeptID	Department
500	Marketing

7

Answer:

- Joins - These are slow in a distributed system due to network hops
- Transactions - For similar reasons but also reliability is also an issue

A good distributed system needs a way to avoid these where possible:

- Only update one record in a table at a time (using SQL terminology for now, but this equates to document in a collection in MongoDB)
- Why? - Don't need to communicate with other servers

Typical RDBMS application schema depicted above - "One record" may consist of 6 tables - 9 rows which would cause a problem in a distributed system.



# Relational - Denormalization

EmpID	Name	Dept	Title	Manage	Payband
9950	Dunham, Justin	Marketing	Product Manager	6531	C

EmpBenPlanID	EmpFK	BenType	Plan
1	9950	Health	PPO Plus
2	9950	Dental	Standard

We can reduce the number of tables hence reduce the number of calls between servers by denormalizing data.

Denormalizing might seem like a backward step for those from an RDBMS background, but in reality, it is often utilized for performance reasons. With MongoDB, this is not considered bad practice.

The focus should be on the business requirements and optimizing a schema for this.

In this example using RDBMS, we cannot denormalize this 1:N relationship any further to create a single record \* - which would negate the need for joins, plus make the data more comfortable to use and digest

\* without complicated and messy delimited methods such as using delimited lists



# Document Model - Arrays

EmpID	Name	Dept	Title	Manage	Payband	BenType	Plan
9950	Dunham, Justin	Marketing	Product Manager	6531	C	Health	PPO Plus
						Dental	Standard

The example above depicts how we would ideally want to store this data in RDBMS, but it is not possible.

With the document model, we can not only store scalar types such as integer, float, text, etc., but we can also store embedded data - which equates to how we would ideally want to store the above (a table inside a table)

We have two of these special Container Types - Array and Nested Document. And Array is a field with multiple values - a Nested Document is fields that has sub fields of its own.

Data is co-located in-memory, on-disk, and logically on the same hardware.

This explicit parent-child storage mechanism improves retrieval and query performance and simplifies our application queries and updates.

Parent-child relationships are common within database applications; hence an improved approach to data storage and retrieval is often gladly adopted amongst developers.





# Document Model Representation - JSON

```
{
  "_id" : 9950,
  "pay_band": "C",
  "employee_name": "Dunham, Justin",
  "department" : "Marketing",
  "title" : "Product Manager, Web",
  "report_up": "Neray, Graham",
  "benefits" : [
    { "type" : "Health",
      "plan" : "PPO Plus" },
    { "type" : "Dental",
      "plan" : "Standard" }
  ]
}
```

10

An example document with the nested parent-child relationships described previously is shown above.

Note how this visualization is easier to digest than looking at such relationships in tabular form.

The above is JSON, which is human readable. We don't store JSON in MongoDB as this is not performant. We use BSON. Similar to viewing data as CSV in Oracle, but this is not how it is stored.

The field names above will also be stored as data, which creates some overhead. But compression in MongoDB greatly reduces the space required to store field names repeated across documents.

Missing fields have no cost in MongoDB, so sparse tables are cheaper.



# Document Storage - BSON

```
{
  "_id" : 9950,
  "pay_band": "C",
  "employee_name": "Dunham, Justin",
  "department" : "Marketing",
  "title" : "Product Manager, Web",
  "report_up": "Neray, Graham",
  "benefits" : [
    { "type" : "Health",
      "plan" : "PPO Plus" },
    { "type" : "Dental",
      "plan" : "Standard" }
  ]
}
```

```
\x315\x00\x00\x00
\x01
_id\x00
\x01\x00\x00\x00
0x26de\x00
\x02
pay_band\x00
\x02\x00\x00\x00
C\x00
\x02
employee_name\x00
\x02\x00\x00\x00
Dunham, Justin\x00
.
.
\x00
```

11

BSON is not JSON - <https://www.mongodb.com/basics/bson>

JSON is a text format that requires character by character parsing (like CSV) and provides fewer data types (integer, string, etc.)

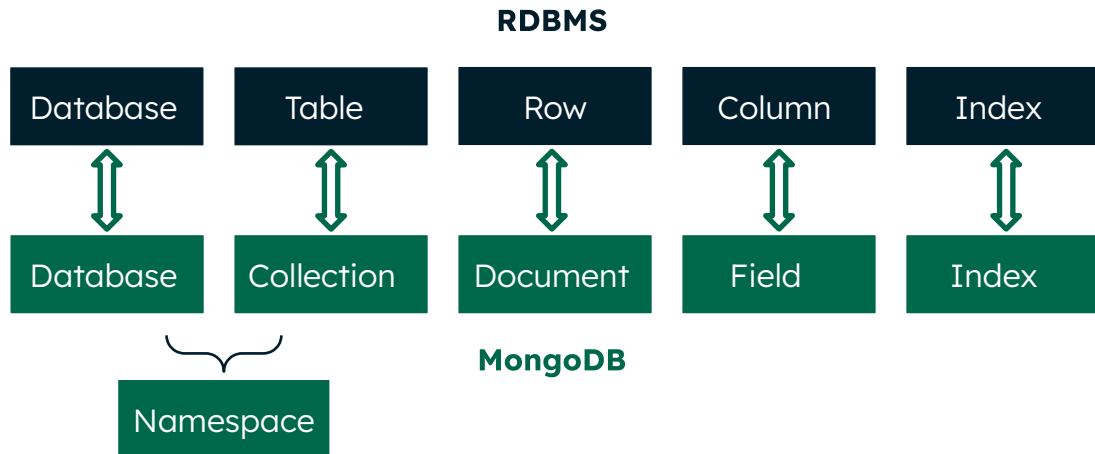
BSON is serialized, binary objects with data types, lengths (variable-length strings)

BSON is faster to read, traverse, and contains more information, and isn't subject to faking/injection

BSON is similar to how we used to save objects to disk by serializing them to a stream of bytes in 1990s/2000s windows applications



# Terminology



12

We explained that a Document is like a richer, more structured version of a Row with Repeating fields and Depth

Apart from that concepts are very similar to any other database, Documents are grouped together in collections (Tables) which should have a similar if not identical structure.

Collections are grouped together in Databases - where they relate to the same application or business "Thing". Like a "Schema" in Oracle or a "Database" in MySQL and MSSQL

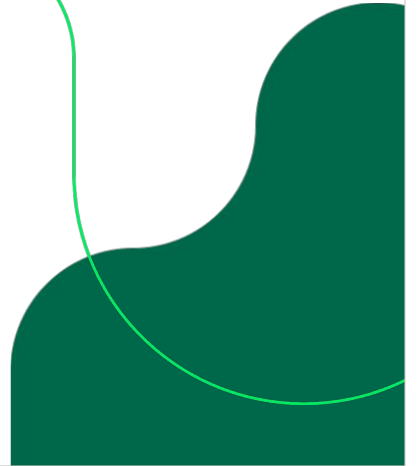
Inside a Document you have Fields - and we haven't mentioned this yet, on fields you can add indexes to speed up querying.

Finally in MongoDB - we refer to the combination of a Database and Collection as a Namespace.

We will mix and match these through this training to ease in the new terminology.

**Note:** A field may also be referred to as a key.

# Benefits of MongoDB





# Agile Database Schemas

EmpID	Name	Dept	Title	Manager	Payband	BenType	Plan
9950	Dunham, Justin	Marketing	Product Manager	6531	C	Health	PPO Plus
						Dental	Standard

EmpID	Name	Title	Payband	Bonus
9952	Joe White	CEO	E	20,000

EmpID	Name	Dept	Title	Manager	Payband	Shares
9531	Nearey, Graham	Marketing	Director	9952	D	5000

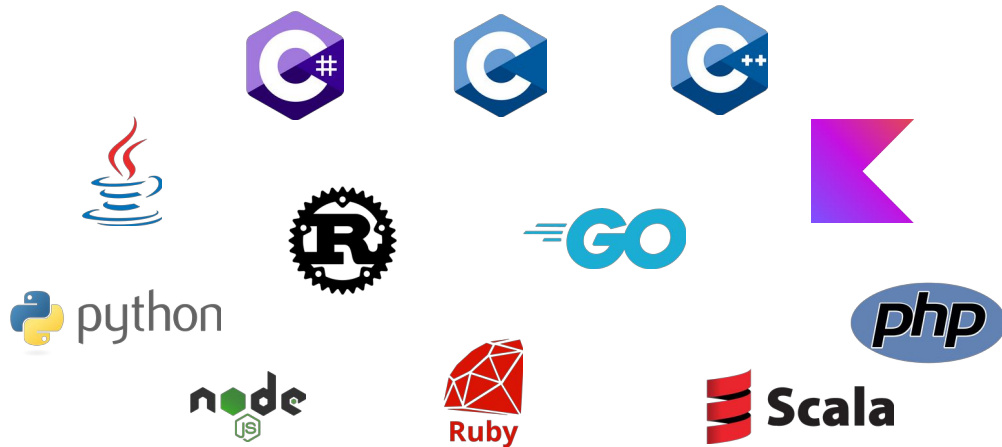
The MongoDB document approach means that not all documents in a collection (records/tables in SQL language) need to have exactly the same fields.

Logically they should be variants of the same type of record, and consistency should be enforced to a degree. However, this flexibility is advantageous.

No need to store missing fields, and there is no cost for adding new ones. The database server doesn't have a list of expected fields by default. This is efficient if you need to store sparse data where in an RDBMS you may have a lot of null values.

Every document in MongoDB already logically contains every possible legal field name. It's just it has a value of null by default unless you set it to something else.

# Multiple Language Support



15

When designing MongoDB, careful consideration was given about how to interact with the data. Traditional SQL was observed as being inadequate

- Using SQL as the internal computer transfer mechanism is inefficient as it requires parsing, which is expensive and errors prone including injection attacks.
- SQL was not designed as an interprocess design mechanism but as a UI (That's why it's basically English), and it's a terrible fit, but ODBC/JDBC use it.
- SQL needs to be standard - and the SQL standard doesn't work for documents. Better no SQL than SQL-ish

Idiomatic Driver Libraries instead:

- MongoDB uses BSON to communicate and utilizes an object-based API in the form of drivers.
- Drivers that work with native data types (Maps, Dictionaries, Objects) are available for many popular programming languages as well as idiomatic ones for other less common languages
- For testing/debugging, we have an interactive JSON REPL (Read Eval Print Loop) environment. This allows interaction with the database in real-time, rather than having to compile code every time you run a query

Official Drivers for: C, C#, C++, Go, Kotlin, Java, NodeJS, PHP, Python, Ruby, Rust, Scala



# Native Clients

Command-line shell for interacting directly with database

Compass as a GUI to access the database

```
> db.collection.insertOne({
  product:"MongoDB",
  type:"Document Database"
})

> db.collection.findOne()
{
  "_id": ObjectId("5106c1c2fc629bfe52792e86"),
  "product": "MongoDB",
  "type": "Document Database"
}
```

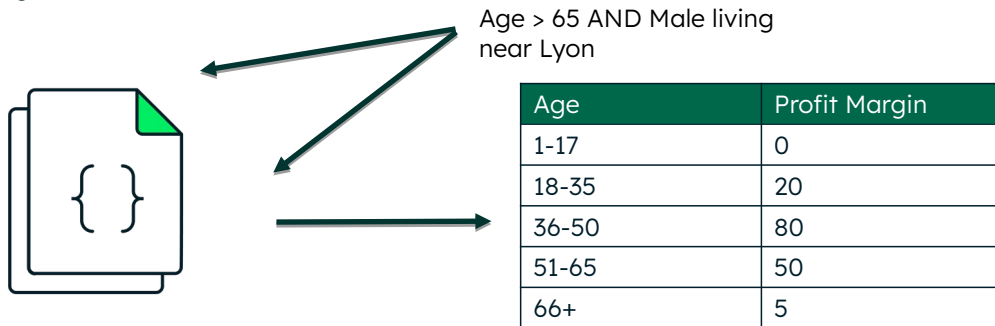


# Full Database Functionality

Complex Indexed Queries

Smart Edits

Aggregation Framework



17

MongoDB provides agility, scalability, and performance without sacrificing the functionality of relational database (like full index support and rich queries)

It utilizes a planner to select what indexes might optimize query performance. Indexes are used to limit the data that needs to be scanned.

This differs from some NoSQL DBs where you can only query on predefined patterns.

MongoDB behaves very much like a typical RDBMS database in that it also edits smartly - i.e., the server makes changes directly instead of fetching the record back to the client, modifying it, and pushing it back (like many NoSQL databases do, which causes race conditions and locking problems)

SQL databases have always been able to summarise and aggregate on the server-side and simply return raw data. Early on, like other vendors, MongoDB adopted Map-reduce to achieve this, but it is complicated and slow, so it was changed to the aggregation pipeline.

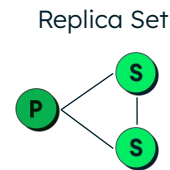
The aggregation pipeline has extended MongoDB query capability to allow complex in-database computation and aggregation to provide summarised data. It is declarative, rather than writing functions in Map-Reduce, which is more like SQL.



# Availability and Scalability

## High Availability via Replica Sets

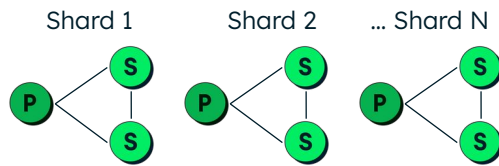
- Multiple copies of all data
- Different hosts / locations
- Continuous replication



**High availability** - Replication

## Scale via Shards

- Partition data over multiple Replica Sets
- Provides unlimited hardware scaling



**Horizontal scalability** - Sharding

## Compression of data

Allowing scalability was one of the driving factors for creating MongoDB, along with the ability to run over multiple physical machines to take advantage of more RAM, CPU, and Disk.

A prerequisite for this is 'High Availability' because the more machines you deploy, the greater the risk for failure. If configured correctly, MongoDB can survive losing up to 50% of its servers and keep working (with caveats). A server can be taken down, upgraded, maintained, and replaced without downtime to the cluster.

The concept of sharding is synonymous with table partitioning. If deploying scaling, auto sharding can be utilized to spread data across multiple clusters. This can dynamically resize partitions and move data accordingly.

MongoDB compresses data on disk (up to about 50%) to utilize space and minimize storage costs.



# Enterprise Tooling

## Enterprise Management Tools:

- Atlas - MongoDB Database as A Service
- Ops Manager - Monitor/Alert/Manage/Backup your own servers.
- Cloud Manager - Ops Manager hosted in the cloud.
- Kubernetes Operator to deploy in containers.
- Terraform Provider from Hashicorp.



Working at scale requires good enterprise management tooling, which we have in the form of Atlas, MongoDB as a service and Ops Manager & Cloud Manager (Hosted Ops Manager) to Manage your own hardware.

These tools let you Monitor, Deploy, Upgrade Live , and Backup, etc when you supply the hardware and, for Atlas even the hardware is provided for you and scaled to fit.

In addition we have integration with Kubernetes, and Terraform to ensure MongoDB can be deployed in these environments.

# When to use MongoDB

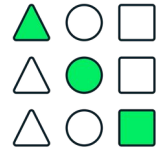




# When to use MongoDB

When you need **high-speed access to complex objects**

- Atomic partial updates
- Fast retrieval
- Secondary indexes
- Aggregation capabilities



When you want to **store larger data structures together**

- Large Arrays
- Text Fields
- Binary Data



21

Imagine writing the backend for a complex system such as a computer game where the player has created a whole environment with buildings, characters, etc.

How would you store, query, and analyze that data?

MongoDB is optimized for storing and querying complex data objects.

Partial updates allow multiple processes to effectively update the same document simultaneously where there is no overlap of changes.

The aggregation pipeline is a mature and powerful tool for querying data in a fast, efficient, and easy to understand manner.

MongoDB can store larger text objects, and blobs inline with the data and not overspill pages with their own API's



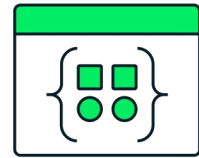
# When to use MongoDB (cont.)

When you value **rapid development**

- Interaction by Objects
- Rich functionality

Where you need to store **structures of varying shapes.**

- Application-defined schemas
- Heterogeneous schemas



22

MongoDB encourages application-defined schemas and performance targeted at the most utilized application queries.

## Case study: Metlife

Integration of 70 separate administrative systems to allow for a consolidated customer view

Migrated from SQL to document model, providing a readable JSON format

Chose MongoDB because of the ability of NoSQL to ingest structured, semi-structured, and unstructured information without requiring tedious, expensive, and time-consuming database-mapping or extract, transform and load (ETL) processes.

Allowed for team independence in the ETL process in the form of a UI

Moved from pilot to roll out in 90 days

Leading internet companies release new software versions every half hour or less. Used well, MongoDB can assist in enabling such rapid evolution of live systems.



# When to use MongoDB (cont.)

When we have **large data volumes**.

- Data volumes growing
- Growth potentially unlimited.
- No big up-front payment.



When we want **distributed data**.

- Worldwide low latency.
- Legal obligations for locations.
- Real-time distribution.



23

Other reasons to use MongoDB.

Large data volumes for applications such as:

- IoT (internet of things)
- Gaming
- Smart devices
- Social media
- Website interactions / purchases
- Transaction history

When data growth is unpredicted or is growing at a high velocity

When you want distributed data:

Global application usage requiring localized data for speed / legal reasons

Use of 'Zone Sharding' to distribute data across different hardware depending on data age, relevance, etc



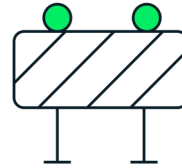
# Things to be conscious of

Easy to build with no training.

- Easy to get wrong.
- Performance can suffer
- Issues can arise too late!

DBAs need to be trained and certified.

- Developers do traditional DBA tasks.
- But DBAs have tasks too.
- Forget them at your peril.
- Or use DBaaS like Atlas.



Easy to get wrong - Although a powerful tool, your applications will suffer in the long run if it's adopted using bad practices. Training and good design are important!

Because something is free and easy to understand the basics of does not mean it's simple to master

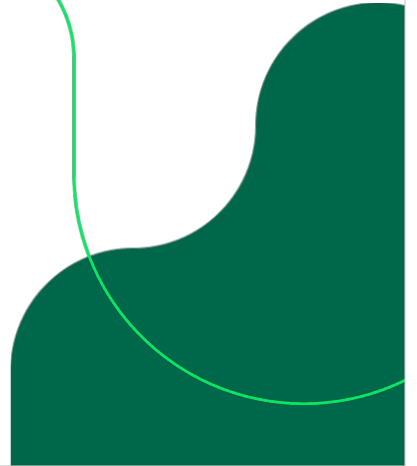
Optimization of schema design for most utilized application queries is crucial to long term application performance. Changes to schema design are not impossible, but better to try and get this right from the beginning.

The understanding of using efficient indexing is also crucial to application performance.

Developers should understand how the database works as well as DBAs as they often perform traditional DBA tasks.

Use Atlas for a simplified approach to running and maintaining MongoDB.

# MongoDB and RDBMSs







# With an RDBMS

Data is stored in an efficient binary form on the disk.

You can index and query any field or set of fields.

You can perform atomic updates to one or more records.

You can update parts of records without retrieving them first.

You can compute aggregates and summaries on the server.

Schemas matter and are rigidly enforced by the server.

All fields have a defined data type (String, Integer, Date, etc.).

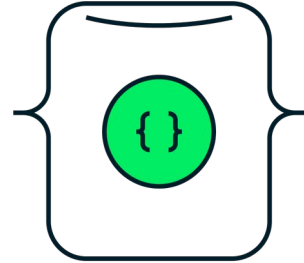
You join data together for querying and retrieval.

You can query with SQL.



# With MongoDB

Data is stored in an efficient binary form on the disk.  
You can index and query any field or set of fields.  
You can perform atomic updates to one or more records.  
You can update parts of records without retrieving them first.  
You can compute aggregates and summaries on the server.  
Schemas matter and **can be** rigidly enforced by the server.  
All **values** have a defined data type (String, Integer, Date, etc.).  
You **can** join data together for querying and retrieval.  
You can query with SQL **using the BI connector** add-on.



27

Dispelling some myths that some people may have heard about MongoDB - Some features such as schema enforcing are optional, not missing.

You don't need to store every field for each record. If you just omit it, it's null and consumes no space.

Normally a given field has a single data type that can be enforced or not, depending on your requirements.

Data is stored as serialized binary information as BSON and not as JSON. MongoDB is not a JSON database any more than Oracle is a CSV database.

The BI connector is an add-on for MongoDB Enterprise that provides a MySQL Compatible proxy that makes MongoDB pretend to be a MySQL server and provides read-only SQL access.



# What is different in MongoDB

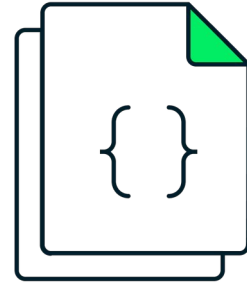
You can Query with SQL but normally don't

- Interaction is from code using Object-based APIs
- Rather than constructing SQL Strings
- SQL is used only to enable third-party BI tools.

Documents/Objects are first-order data types

- Data modeling/Schema design is done differently
- Dynamic schemas can be used if desired

The primary key field is always called `_id`



28

SQL can be used to query MongoDB, but this is not common. MongoDB uses a more performant query language that involves sending serialized objects over the wire to describe data and give instructions.

Injection attacks are a concern with SQL as they convert pseudo-English to computer language.

No parsing is required (as with SQL), which reduces server CPU usage.

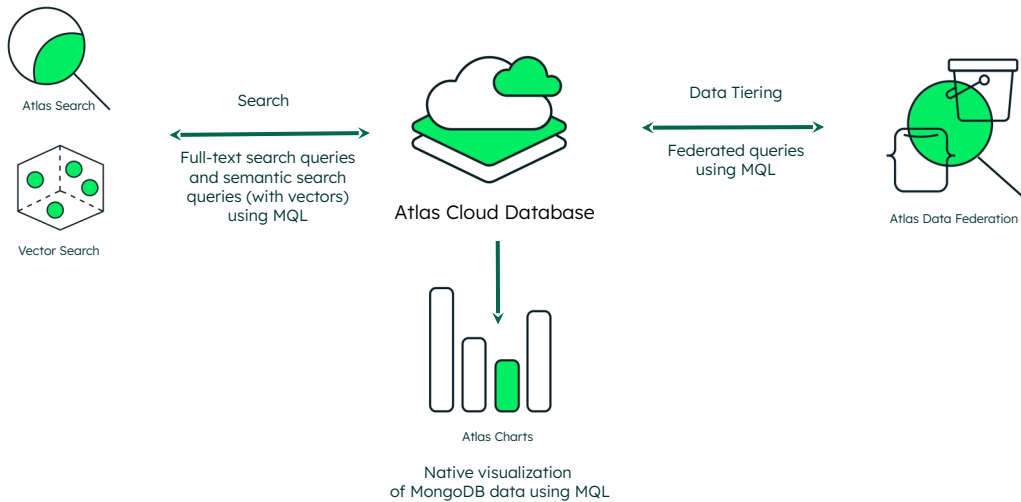
Good schema design is still paramount with MDB. Just populating with data and utilizing a dynamic schema is possible but may cause performance issues further down the line when querying.

MongoDB will automatically create a unique `_id` when one is not supplied. This is GUID and can be used as a primary key if no others are used.

# The MongoDB Atlas Platform



# The MongoDB Atlas Platform



30

MongoDB Atlas is a fully-managed, integrated data layer designed for modern applications and modern operating models. At its heart, Atlas delivers the latest versions of MongoDB, which provides foundational features such as

- An intuitive and flexible data model
- MongoDB Query Language (MQL) for building nearly any workload
- Transactional guarantees at a global scale
- Unique data distribution capabilities

Atlas Search gives you full text search powered by Lucene on your Atlas hosted databases.

Atlas data lake allows you to store data cheaply in Amazon S3 in JSON, CSV or Parquet format and query them like a MongoDB database.

- This include auto migrating data from Atlas to S3 as it gets older
- And federated queries over Atlas and Data Lake

MongoDB Charts

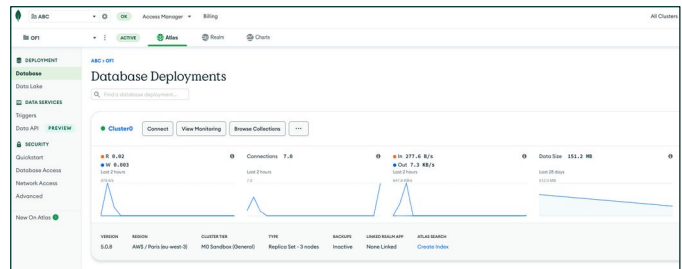
- Rich, document-centric, visualization functionality provided by MongoDB Charts



# Atlas Managed Clusters

MongoDB Atlas is MongoDB as a Service.

- Highly available, scalable clusters in AWS/GCP/Azure
- Global availability and scaling
- Development and Business focus
- Manage Database infrastructure



31

M0 clusters are free sandbox replica set clusters. You can deploy one M0 cluster per Atlas project. If you do not have an account / corporate login, you can create one: <https://www.mongodb.com/cloud>

We are going to configure a 3-node replica set for this training.

A Node is synonymous to an instance or a single MongoDB server.

A Cluster is a group of nodes.

# Launch an Atlas cluster





# Setting-up MongoDB Atlas

We will set up an Atlas Free Tier cluster

- 3 Nodes in different datacenters

- Highly available

- 512 MB of storage

- Secure by default

- Using TLS network encryption

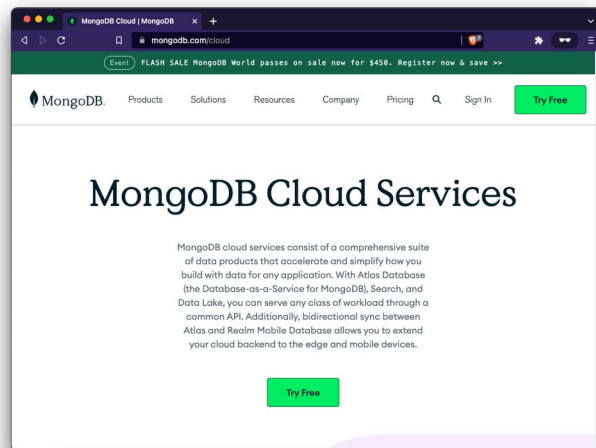
Atlas free tier is on a shared server.

Some functionalities are not available/limited (throughput, number of connections, data transfer limits, monitoring, alerting, API access, etc.)

It is technically large enough to run most small business applications and is free for life.



# Demo



34

One of the students will share their screen and the instructor will show the following steps: (Other students should follow along as the Instructor shows each step)

1. Signup and login to your lab environment
2. Sign Up to cloud.mongodb.com.
  - a. Use personal email id and not your company email address.
  - b. Note down the password
  - c. If the participant already has an account created then the steps maybe slightly different. Assist such students in setting up the free tier cluster.
3. Setup a free tier cluster
  - a. Use defaults for setup
  - b. Create a DB user with the following credentials: Username: admin Password: qwerty123
  - c. Load sample data. Depending on the workflow (new account or existing account) the user may or maynot be presented the option to load sample data during cluster creation. Address both situations.
  - d. Add lab VM public IP to network access list in Atlas
4. Connect to the new cluster using mongosh from the lab VM

# Essential commands





# Useful MongoDB Commands

**Show all databases** in the cluster

Check the **current database**

**Set db** to the **test** database

Check the current database again

**Show the collections** in the database

```
Atlas [primary] myFirstDatabase> show dbs
admin   377 kB
local  13.8 GB

Atlas [primary] myFirstDatabase> db
myFirstDatabase

Atlas [primary] myFirstDatabase> use test
switched to db test

Atlas [primary] test> db
test

Atlas [primary] test> show collections

Atlas [primary] test>
```

36

**show dbs** is a helper - it's equivalent to a small piece of javascript code to list the databases.

Some commands:

- **db** - this is a database object and is used at the beginning of most commands to run against a DB. Executing just db without any other methods, gives you the current database name.
- **show dbs** - shows available databases
- **use <database>** - selects database to use. Note that it does not need to already exist
- **show collections** - show collections available in the currently selected database. If there are no collections in a database, the command gives an empty response.



# Database Interaction

**Find all documents** in a collection

Create new document object **as variable**

**Insert a Document** in the collection

**Find Documents** based on a filter

```
Atlas [primary] test> db.employees.find({})

Atlas [primary] test> var employee = { "name" : "Jon", "hungry" : true, "title" : "director" }
{ name: 'Jon', hungry: true, title: 'director' }

Atlas [primary] test> db.employees.insertOne(employee)
{ acknowledged: true, insertedId: ObjectId("60efe274ca75939ea79feeea") }

Atlas [primary] test> db.employees.find({hungry:true})
[
  {
    _id: ObjectId("60efe274ca75939ea79feeea"),
    name: 'Jon',
    hungry: true,
    title: 'director'
  }
]

Atlas [primary] test> show collections
employees
```

37

Technically you are creating an object - and putting it in MongoDB as a Document, but Record is a generic term.

Some commands:

- **db** - this is a database object and is used at the beginning of most commands to run against a db
- **insertOne()** - this allows us to insert an object e.g.  
db.getCollection('test').insertOne({name:"bob"})
- **find()** - this allows us to query e.g. db.getCollection('test').find()

**Note:** The **employees** collection is automatically created when we execute the insert command.  
Demo:

1. db.employees.find({})
2. var employee = { "name" : "Jon", "hungry" : true, "title" : "director" } { name: 'Jon', hungry: true, title: 'director' }
3. db.employees.insertOne(employee)
4. db.employees.find({hungry:true})
5. show collections



# A JavaScript REPL

**mongosh** - JavaScript and NodeJS REPL



```
Atlas [primary] test> for(let i=0; i<10; i++) {print(i);}
0
1
2
3
4
5
6
7
8
9
Atlas [primary] test>
```

38

We can run JavaScript code snippets in mongosh. It is a fully functional JavaScript and NodeJS REPL for interacting with MongoDB deployments.

Note: REPL stands for Read Evaluate Print Loop, and it is a programming language environment (basically a console window) that takes single expression as user input and returns the result back to the console after execution.

The REPL session provides a convenient way to quickly test simple JavaScript code.

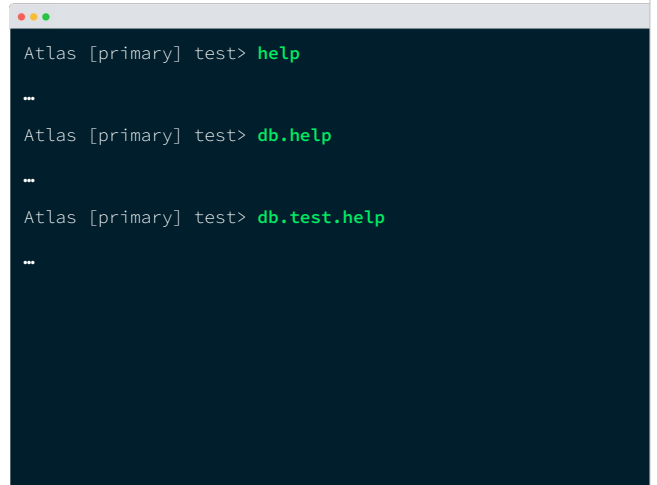


# Help!

help

db.help

db.<collection>.help



```
Atlas [primary] test> help
...
Atlas [primary] test> db.help
...
Atlas [primary] test> db.test.help
...
```

help - Shell help

db.help - Database class help

db.col.help - Collection class help

# Useful tools



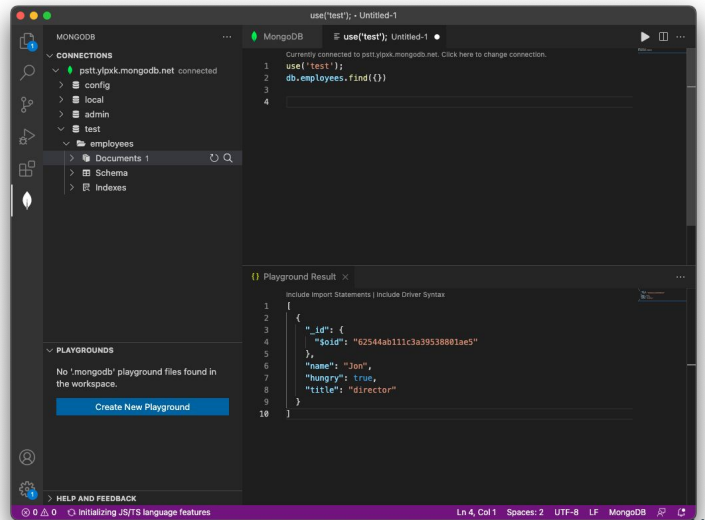


# VSCode Playground

MongoDB for VS Code

Visual Studio Marketplace

Intelligent Autocomplete





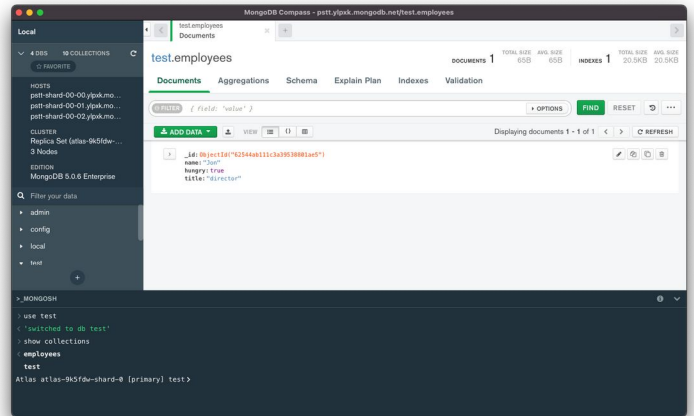


# Compass

GUI tool that includes:

- Aggregation Builder
- Mongosh
- Visual explain Plans

<https://www.mongodb.com/products/tools/compass>



You can download and install MongoDB Compass

# Quiz Time!





# #1. Select two features that are NOT part of the core MongoDB database

A

Aggregations

B

Transactions

C

Stored  
procedures

D

Field level  
updates

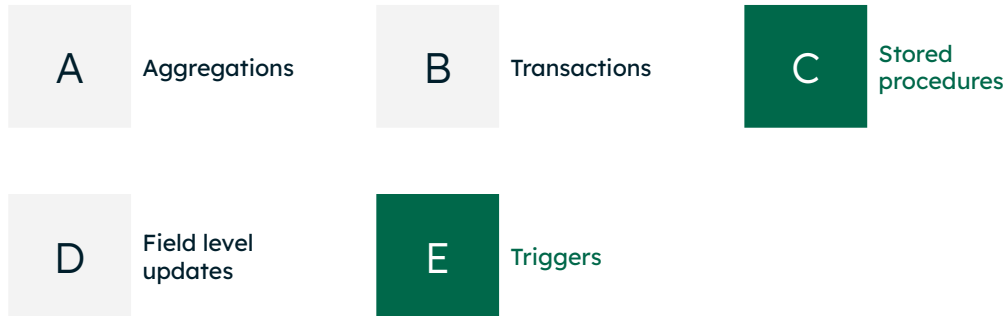
E

Triggers

Answer in the next slide.



# #1. Select two features that are NOT part of the core MongoDB database



45

MongoDB

- includes an aggregation framework
- can work with transactions
- allows you to update() and \$set fields in a document

MongoDB does not include any stored procedure

Triggers are a feature of App Services



## #2. Select three correct statements about MongoDB Documents

A

Stored as BSON on the disk

B

Often queried using BSON syntax

C

Homogeneous in a collection

D

Created with a primary key even if this is not provided

E

Distributed over a network when sharding

Answer in the next slide.



## #2. Select three correct statements about MongoDB Documents

A

Stored as BSON on the disk

B

Often queried using BSON syntax

C

Homogeneous in a collection

D

Created with a primary key even if this is not provided

E

Distributed over a network when sharding

MongoDB always stores data in BSON format on disk

Documents inside the same collection can have different structure, different fields, etc, so they are not identical in structure

Though when a document is created we don't need to specify an `_id` field, an `_id` is mandatory for any document

Documents from the same collection can be distributed in different clusters(sharding)



## #3. Select two core MongoDB functions

A

Create an Inner  
Join on  
documents

B

Define Foreign  
Key Constraints

C

Natively query  
data using SQL

D

Enforce the data  
type of  
individual fields

E

Enforce  
mandatory fields  
in a collection

Answer in the next slide.



## #3. Select two core MongoDB functions

A

Create an Inner Join on documents

B

Define Foreign Key Constraints

C

Natively query data using SQL

D

Enforce the data type of individual fields

E

Enforce mandatory fields in a collection

You can use \$lookup to gather data from other collections on a query but this is not the same as an inner join and it would affect performance, so it is not recommended

MongoDB stores data as BSON format that has a binary data type

It is possible to use SQL syntax to query data in MongoDB using specific connectors, like the BI connector (but not natively)

It is possible in a collection to force the specific fields and data types to define the document structure

The notion of foreign key constraints don't exist in MongoDB (ie, nothing is prevented from being changed/removed on a linked collection)





## #4. Select the components in the Atlas platform which use MQL (MongoDB Query Language)

A

Atlas Search

B

Atlas Vector  
Search

C

Atlas Data  
Federation

D

Atlas Cloud  
Database

E

Atlas MongoDB  
Charts

Answer in the next slide.



## #4. Select the components in the Atlas platform which use MQL (MongoDB Query Language)



Atlas Search



Atlas Vector  
Search



Atlas Data  
Federation



Atlas Cloud  
Database



Atlas MongoDB  
Charts

Refer to the MongoDB Atlas Platform diagram.



## #5. Select one statement which best describes MongoDB Atlas

A

Self-managed  
DB installed  
locally

B

Service to install  
a database on a  
server

C

Cloud provider  
used to deploy  
clusters

D

Managed  
database service  
on the cloud

E

Group of tools to  
work with a  
database

Answer in the next slide.



## #5. Select one statement which best describes MongoDB Atlas

A

Self-managed  
DB installed  
locally

B

Service to install  
a database on a  
server

C

Cloud provider  
used to deploy  
clusters

D

Managed  
database service  
on the cloud

E

Group of tools to  
work with a  
database

This question can be described in various ways but the most fitting answer from the provided options is D. It is a DaaS. The service itself can allow for installations and the ability to deploy using different cloud providers - but its still part of the service overall - it also contains a set of tools, but is not the tools directly.

# Recap

MongoDB is a modern document-oriented database with key attributes such as:

- Agility
- Usability
- Utility
- Scalability & Availability

MongoDB is good for a wide range of use cases but needs correct schemas; not a good idea to use like RDBMS

Atlas is MongoDB hosted as a service

Atlas is flexible and includes small, free clusters. Always uses TLS, authentication & has a firewall

54

- Data volume, velocity, and variety have increased substantially; software development cycles have decreased; requirements for always-on, scalable systems are increasingly table-stakes. The way software and hardware are purchased and consumed has changed a lot since the RDBMS was first invented.
- MongoDB is a document-oriented database for modern applications:
  - By using the document model, MongoDB makes it easier to map to modern programming paradigms, faster to retrieve data, and importantly makes it easier to build systems that scale across multiple machines to escape the limits of a single server. Documents are stored as binary in BSON, although often shown using JSON.
  - MongoDB is agile, usable, capable, and scalable. We integrate well with programming languages and don't sacrifice functionality for scale.
- MongoDB is a general-purpose database with a wide range of use cases and value to bring to all of them.
- Some ways of working with MongoDB are similar to an RDBMS; others require re-thinking to ensure that we make the most out of a horizontally scalable document-oriented database and avoid costly mistakes.