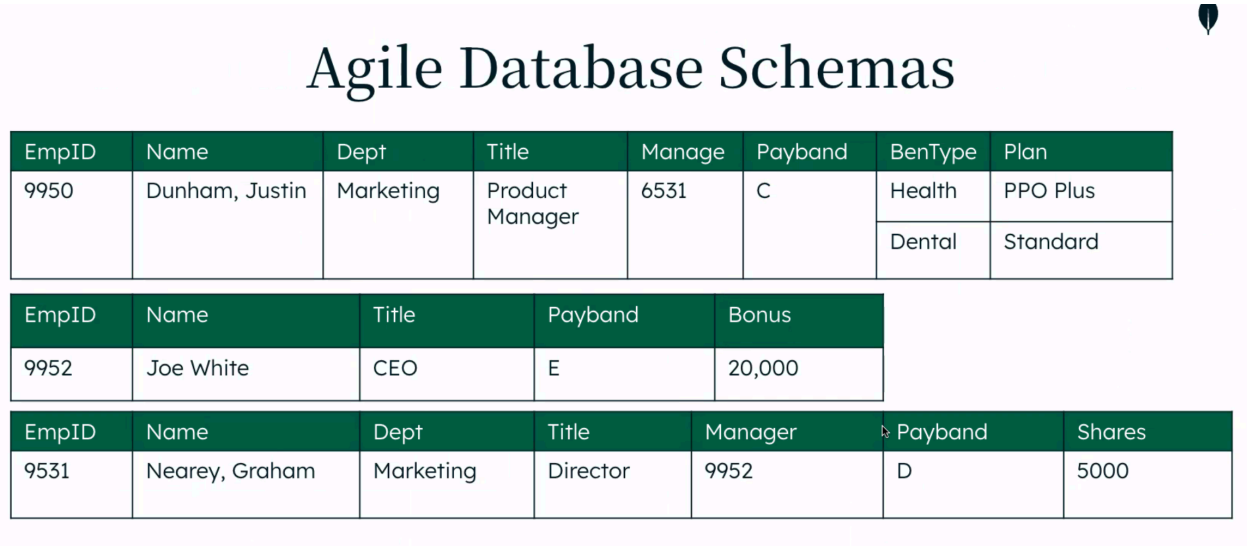
MDB100

M100 Introduction to mongodb and atlas, security

- object is the first class citizen in mongodb
- Bson is binary format of json, so it can adapt different data format
- agregate pipeline



Availability and Scalability

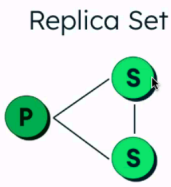
High Availability via Replica Sets

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

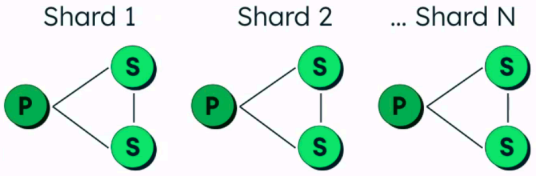
Scale via Shards

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

Compression of data



High availability - Replication



Horizontal scalability - Sharding

read and write default all to primary

shards is for horizontal scaling(can increase write throughput)

sharding read and write in mongodb

-downside is query need to chose the correct shard to be fast

Enterprise Tooling

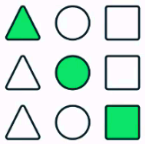
Enterprise Management Tools:

- 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.

When to use MongoDB

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

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



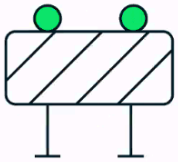
When you want to **store larger data structures together**

- Large Arrays
- Text Fields
- Binary Data

Things to be conscious of

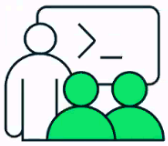
Easy to build with no training.

- 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.



- 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.

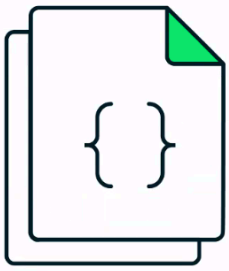
- You can update parts of records without retrieving them first.
- You can compute aggregates and summaries on the server.
- Schemas matter and can be rigidly enforced by the server.
- All values have a defined data type (String, Integer, Date, etc.).
- You can join data together for querying and retrieval.
- You can query with SQL using the BI connector add-on.

What is different in MongoDB

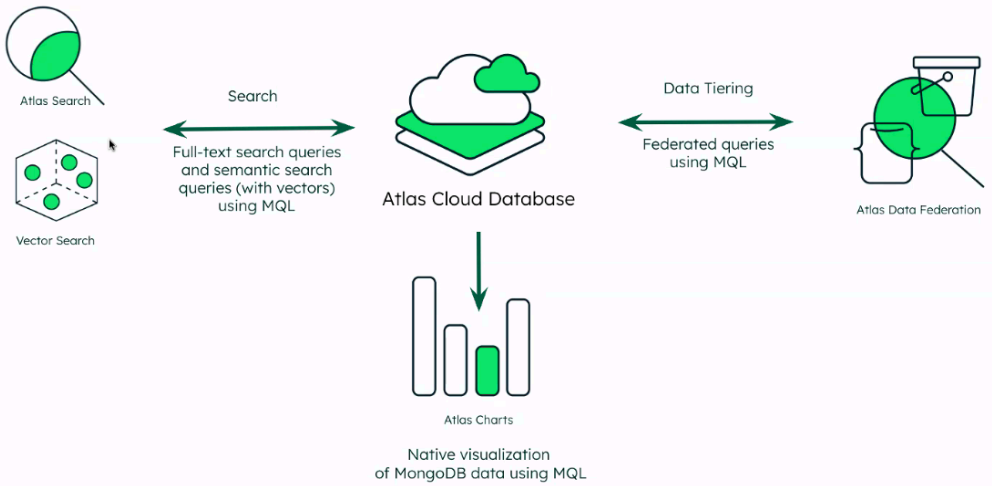
You can Query with SQL but normally don't
- Interaction is from code using Object-based APIs
- Rather than constructing SQL Strings
- SQL is used only to enable third-party BI tools.

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



The MongoDB Atlas Platform



Useful MongoDB Commands

Show all databases in the cluster

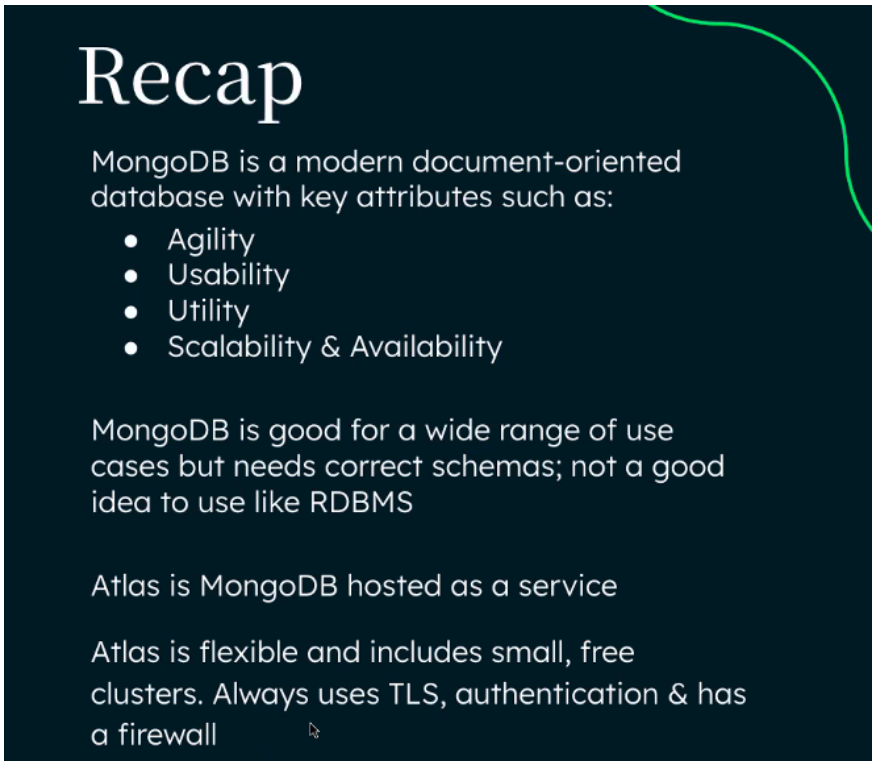
Check the **current database**

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>
```



curl ifconfig.me

mongodb connectionstring

srv format and direct connect format

srv have tls and direct to primary

Basic Database CRUD Interactions

	Single Document	Multiple Documents
Create	`insertOne(doc)`	`insertMany([doc,doc,doc])`
Read	`findOne(query, projection)`	`find(query, projection)`
Update	`updateOne(query,change)`	`updateMany(query,change)`
Delete	`deleteOne(query)`	`deleteMany(query)`

Order of operations in insertMany()

insertMany() can be ordered or unordered.

Ordered (default) stops on first error.

Unordered reports errors but continues; can be reordered by the server to make the operation faster.

```
MongoDB> let friends = [
       {_id: "joe" },
       {_id: "bob" },
       {_id: "joe" },
       {_id: "jen" }
]

MongoDB> db.collection1.insertMany(friends)
{ errmsg : "E11000 duplicate key error ...",
nInserted : 2 }

MongoDB> db.collection2.insertMany(friends,{ordered:false})
{ errmsg : "E11000 duplicate key error ...",
nInserted : 3 }

MongoDB> db.collection1.find()
{ _id : "joe" }
{ _id : "bob" }

MongoDB> db.collection2.find()
{ _id : "joe" }
{ _id : "bob" }
{ _id : "jen" }
```

23

https://www.mongodb.com/developer/products/mongodb/schema-design-anti-pattern-case-insensitive-query-index/

case insensitive search

MongoDB

4.10 Replace case insensitive $regex query - [Priority 2]

As $regex queries cannot efficiently utilise case-insensitive indexes, running a case-insensitive query using $regex with the i option may end up examining all index keys, resulting in a slower query than one that fully utilises an index.

```
//SAMPLE REGEX QUERY

db.persons.find({first_name: { $regex: /Harriet/i} })
```

It is recommended that a case-insensitive index (meaning it has a collation strength of 1 and 2) be created first, thereafter run a case-insensitive query with the same collation as the index. A collation defines the language-specific rules that MongoDB will use for string comparison. Indexes can optionally have a collation with a strength that ranges from 1 to 5. Collation strengths of 1 and 2 both give you case-insensitivity. For more information on the differences in collation strengths.

```
//SAMPLE CASE INSENSITIVE INDEX

db.persons.createIndex(
  { first_name: 1 },
  {
    collation: {
      locale: "en",
      strength: 2
    }
  }
)
```

A query that is run with the same collation as a case-insensitive index will return case-insensitive results. Since these queries are covered by indexes, they execute very quickly.

```
//SAMPLE CASE INSENSITIVE QUERY

db.persons.find({first_name: "Harriet"}).collation( { locale: 'en',
strength: 2 } )
```

recommend to use collation

Projection: choosing the fields to return

Find operations can include a **projection** parameter.

Projections only return a subset of each document.

Projections include/exclude a set of fields.

```
MongoDB> db.customers.insertOne({
       _id : "ann@gmail.com",
       name: "Ann", orders: [], spend: 0,
       lastpurchase: null
})

MongoDB> db.customers.findOne({ name: "Ann" })
{  _id : "ann@gmail.com",
  name : "Ann",
  orders : [], spend: 0, lastpurchase: null }

MongoDB> db.customers.findOne({ name:"Ann" },{name:1, spend:1})
{ _id : "ann@gmail.com", name : "Ann", spend : 0 }

MongoDB> db.customers.findOne({ name:"Ann" },{name:0, orders:0})
{ _id : "ann@gmail.com", spend : 0, lastpurchase : null }

MongoDB> db.customers.findOne({ name:"Ann" },{name:0, orders:1})
MongoServerError: "Cannot do inclusion on field orders in
exclusion projection"

MongoDB> db.customers.findOne({ name:"Ann" },{_id: 0, name:1})
{ name : "Ann" }
```

can not mix 0 and 1 (except _id filed can do so)

Fetch multiple documents using find()

find() returns a cursor object rather than a single document

We fetch documents from the cursor to get all matches

mongosh fetches and displays 20 documents from the cursor object.

```
MongoDB> for(let x=0;x<200;x++) {
       db.taxis.insertOne({ plate: x })
}

MongoDB> db.taxis.find({})
{ _id : ObjectId("609b9aaccf0c3aa225ce9116"), plate : 0 }
{ _id : ObjectId("609b9aaccf0c3aa225ce9117"), plate : 1 }
...
{ _id : ObjectId("609b9aaccf0c3aa225ce9129"), plate : 19 }
Type "it" for more

MongoDB> it
{ _id : ObjectId("609b9aaccf0c3aa225ce912a"), plate : 20 }
{ _id : ObjectId("609b9aaccf0c3aa225ce912b"), plate : 21 }
...
{ _id : ObjectId("609b9aaccf0c3aa225ce913d"), plate : 39 }

MongoDB> db.taxis.find({ plate: 5 })
{ _id : ObjectId("609b9aaccf0c3aa225ce911b"), plate : 5 }
```

Using Cursors

Here, we store the result of find to a variable.

We then manually iterate over the cursor.

The query is not actually run until we fetch results from the cursor.

```
MongoDB> let mycursor = db.taxis.find({})

MongoDB> while (mycursor.hasNext()) {
       let doc = mycursor.next();
       printjson(doc)
}
{ _id : ObjectId("609b9aaccf0c3aa225ce9117"), plate : 0 }
{ _id : ObjectId("609b9aaccf0c3aa225ce9118"), plate : 1 }
   ...
{ _id : ObjectId("609b9aaccf0c3aa225ce91dd"), plate : 199 }

MongoDB> let mycursor = db.taxis.find({}) // No Output

MongoDB> mycursor.forEach( doc => { printjson(doc) })

//This does nothing - does not even contact the server!
MongoDB> for(let x=0;x<100;x++) {
       let c = db.taxis.find({})
}
```

skip and limit

not suggest to do pagination

recommend way to do is below

Cursors Pagination

```
db.collectionA.aggregate([
    { $match: { platform: 'P1' }},
    { $sort: { _id: 1 }},
    { $limit: 10000},
    { $project: { _id: 1, fieldA: 1, fieldB: 1 }}
]);
```

Then to get subsequent batch

```
const lastId = ObjectId("123123123");

db.collectionA.aggregate([
    { $match: { _id: { $gt: lastId }, platform: 'P1' }},
    { $sort: { _id: 1 }},
    { $limit: 10000},
    { $project: { _id: 1, fieldA: 1, fieldB: 1 }}
]);
```

Sorting Results

Use **sort()** cursor modifier to retrieve results in a specific order

Specify an object listing fields in the order to sort and sort direction

```
MongoDB> let rnd = (x)=>Math.floor(Math.random()*x)

MongoDB> for(let x=0;x<100;x++) {
db.scores.insertOne({ride:rnd(40),swim:rnd(40),run:rnd(40)})}

//Unsorted
MongoDB> db.scores.find({},{_id:0})
{ ride : 5, swim : 11, run : 11 }
{ ride : 0, swim : 17, run : 12 }
{ ride : 17, swim : 2, run : 2 }

//Sorted by ride increasing
MongoDB> db.scores.find({},{_id:0}).sort({ride: 1})
{ ride : 0, swim : 38, run : 10 }
{ ride : 1, swim : 37, run : 37 }
{ ride : 1, swim : 30, run : 20 }

//Sorted by swim increasing then ride decreasing
MongoDB> db.scores.find({},{_id:0}).sort({swim: 1, ride: -1})
{ ride : 31, swim : 0, run : 14 }
{ ride : 11, swim : 0, run : 14 }
{ ride : 30, swim : 1, run : 34 }
{ ride : 21, swim : 1, run : 3 }
```

db.companies.find(
{number_of_employees:{$lt:200}},
{number_of_employees:1, name:1, _id: 0}
).sort({number_of_employees:-1}).limit(1)

$ne is range operator

$gt

$gte

$lt

$ne

$in [,] array elments less than 200 elements is equal oprator, if move than 200 than is range operator

Boolean operator

AND,OR,NOR,NOT

$or:[{},{}]

use double quote when use nested query

db.people.find({"address.zipcode":"10005"})

array query operators

$all (the order doesnt matter, but need to have all elements)

$size

$elemMatch

Array specific query operators

MongoDB has operators designed specifically for querying against arrays.

These are as follows:

- $all
- $size
- $elemMatch

Why is there no $any operator?

```
MongoDB> db.fun.insertOne({
      "name": "John",
      hobbies: ["cars","robots","gardens"]
})
{acknowledged:true }

MongoDB> db.fun.find({
      hobbies: { $all: ["robots","cars"] }
})
{ _id : ObjectId("..."), name : "John", hobbies : [
"cars", "robots", "gardens" ] }

MongoDB> db.fun.find({
      hobbies: { $all: ["robots", "cars", "bikes"] }
}) //No result as bikes is not in the array

MongoDB> db.fun.find({ hobbies : { $size : 3}})
{ _id : ObjectId("..."), name : "John", hobbies : [
"cars", "robots", "gardens" ] }

MongoDB> db.fun.find({ hobbies : { $size : 4 } })
```

8

ElemMatch

$elemMatch matches documents that contain an array field with at least one element that matches the specified query criteria.

```
MongoDB> db.gamescore.insertMany([
{ "_id": "player1", "results": [{ "game": "pacman",
"score": 10 },
      { "game": "pong", "score": 5 }]
   },
{ "_id": "player2", "results": [{ "game": "pacman",
"score": 5 },
      { "product": "pong", "score": 7 }]
   }
])

{acknowledged:true }

MongoDB> db.gamescore.find({
 results: { $elemMatch:{ game: "pacman", score: {$gt:5}}
}})

{ _id : 'player1', results: [...] }
```

9

db.gamescore.insertMany([

{ "_id": "player1", "results": [{ "game": "pacman", "score": 10 },

{ "game": "pong", "score": 5 }]

},

{ "_id": "player2", "results": [{ "game": "pacman", "score": 5 },

{ "product": "pong", "score": 7 }]

}

])

db.gamescore.find({ results: { $elemMatch: { product: "pong", score: {$gte : 7}}}})

comparing two fields in the same documents

```
//Movies where rotten tomatoes rates it higher than imdb
MongoDB> db.movies.find({
   $expr: { $gt: [ "$tomatoes.viewer.rating" ,"$imdb.rating" ] }
})
```

update

use modifiedCount to check the update is success

$set

The $set operator

$set - sets the value of a field to an explicit absolute value

Use dot notation to set a field in an embedded document

Setting a field to an object replaces the existing value entirely

```
{ $set  :
  {
    length: 10,
    width: 10,
    shape: "square",
    coordinates: [3,4]
  }
}
```

```
{ $set  :
  {
    "schoolname" : "Valley HS",
    staff: { principal: "jones" },
    "address.zip" : 90210
  }
}
```

$unset

$unset:{field:””} set value to 1 or true or empty string

$inc. increase

$mul. multiple

$max/$min depends on current value

$max and **$min** can modify a field depending on its current value.

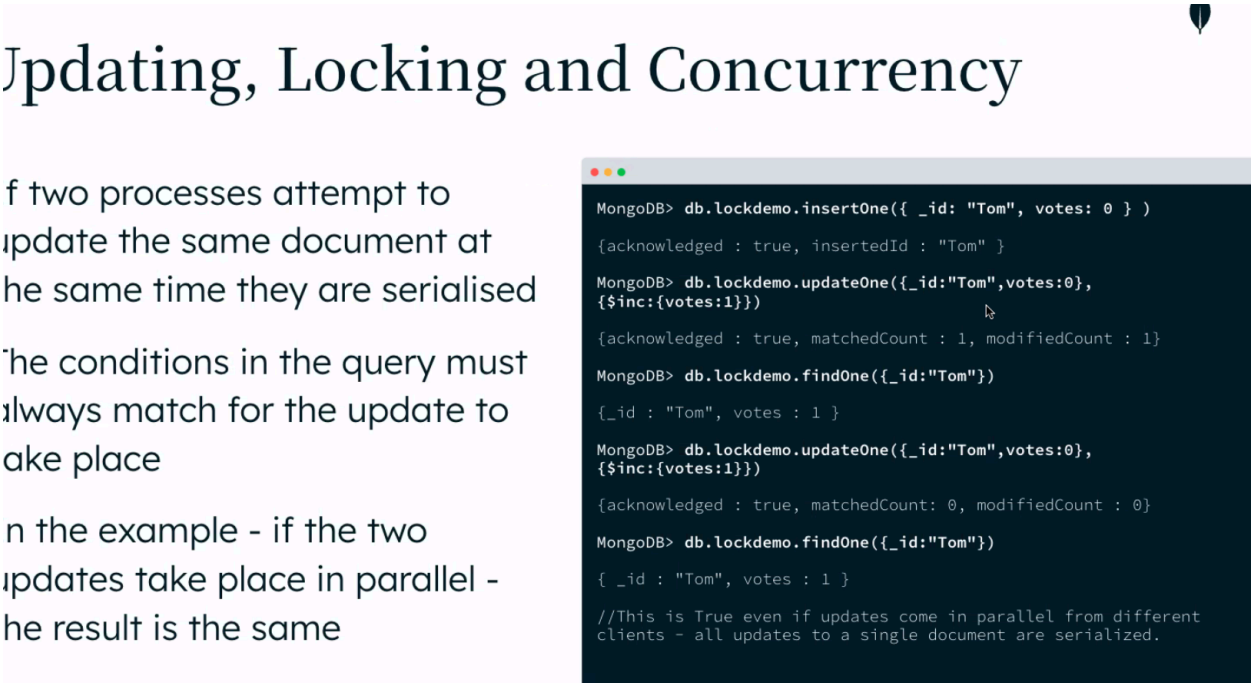
Only update the field if the value given is larger (or smaller) than the current value.

```
MongoDB> db.gamescores.insertOne({name: "pacman", highscore:
10000 })

{acknowledged : true, insertedId : ObjectId("...")}

//This finds the record but does not change it as 9000 < 10000
MongoDB> db.gamescores.updateOne({name:"pacman"},{$max: {
"highscore": 9000}})

{acknowledged : true, matchedCount : 1, modifiedCount : 0}

//This finds and changes highscore as 12000 > 10000
MongoDB> db.gamescores.updateOne({name:"pacman"},{$max: {
"highscore": 12000}})

{acknowledged : true, matchedCount : 1, modifiedCount : 1}

MongoDB> db.gamescores.find({})
{ _id : ObjectId("..."), name : "pacman", highscore : 12000 }
```

update concurrent

always good to include the older value in the query

(eg. vote:0)



mvcc

multi version

What is MVCC?

MVCC (Multi-Version Concurrency Control) is a concurrency control method that allows multiple users to read and write data without locking, ensuring consistency without blocking operations.

Does MongoDB Use MVCC?

✅ **Yes, MongoDB implements MVCC for read operations**, but with some differences from traditional relational databases like PostgreSQL.

How MVCC Works in MongoDB

1. **Reads Use a Snapshot View**

- When a read operation starts, it gets a consistent snapshot of the document.
- This snapshot does not block writes.
- Readers **never block writers**, and **writers never block readers**.

2. **WiredTiger Storage Engine (Since MongoDB 3.0)**

- MongoDB's **WiredTiger** storage engine uses **snapshot isolation**.
- Each operation sees a **stable version** of the data at the start of the query.

3. **Writes Use Copy-on-Write**

- Instead of modifying a document directly, MongoDB creates a **new version** of the document.
- The older version is still visible to ongoing reads.
- Once the write is complete, the new version becomes active.

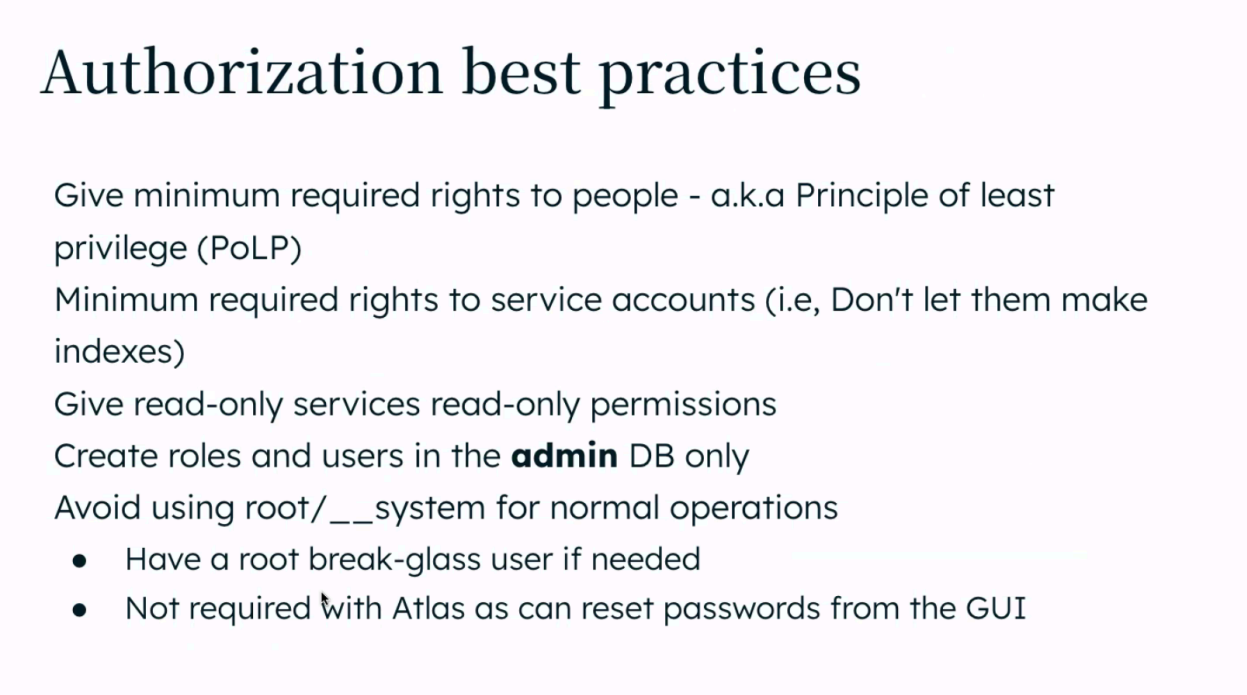
https://www.mongodb.com/blog/post/building-with-patterns-a-summary

design pattern

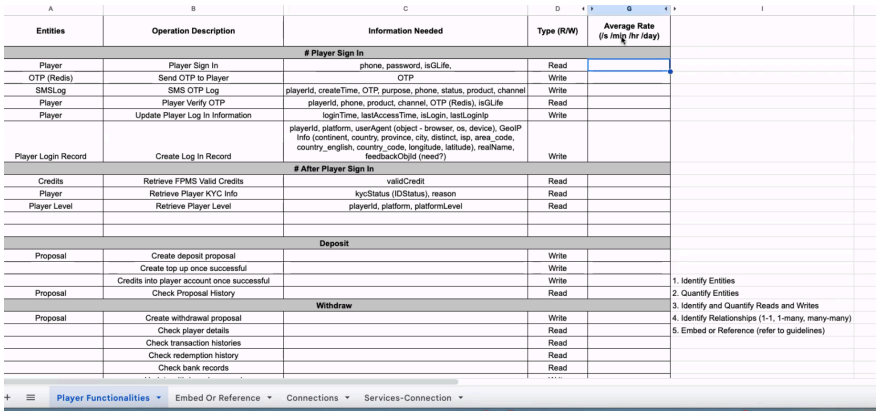
security

What to use and when?
Human users
● Should always have individual logins
● Centralized Identity Management (LDAP/Kerberos/OIDC) is preferable
● SCRAM-SHA is secure but not centralized, requires more maintenance

● X.509 needs certificates with passwords and either local or LDAP Authorization Software/Service users
● No human should know the credentials used by a service
● Kerberos is a great option for Windows Services
● On Atlas with AWS, IAM is available.
● On Atlas with Azure or Okta, OIDC is available.
● Can generate a random password (and associated user) at install time
● Store passwords securely at the application end (i.e, in an Enterprise Password MS)



data modelling guideline



Entities	Operation Description	Information Needed	Type (R/W)	Average Rate (/s /min /hr /day)	
# Player Sign In					
Player	Player Sign In	phone, password, isGLife,	Read		
OTP (Redis)	Send OTP to Player	OTP	Write		
SMSLog	SMS OTP Log	playerId, createTime, OTP, purpose, phone, status, product, channel	Write		
Player	Player Verify OTP	playerId, phone, product, channel, OTP (Redis), isGLife	Read		
Player	Update Player Log In Information	loginTime, lastAccessTime, isLogin, lastLoginIp	Write		
Player Login Record	Create Log In Record	playerId, platform, userAgent (object - browser, os, device), GeoIP Info (continent, country, province, city, distinct, isp, area_code, country_english, country_code, longitude, latitude), realName, feedbackObjId (need?)	Write		
# After Player Sign In					
Credits	Retrieve FPMS Valid Credits	validCredit	Read		
Player	Retrieve Player KYC Info	kycStatus (IDStatus), reason	Read		
Player Level	Retrieve Player Level	playerId, platform, platformLevel	Read		
Deposit					
Proposal	Create deposit proposal		Write		
	Create top up once successful		Write		
	Credits into player account once successful		Write		1. Identify Entities
Proposal	Check Proposal History		Read		2. Quantify Entities
Withdraw					3. Identify and Quantify Reads and Writes
Proposal	Create withdrawal proposal		Write		4. Identify Relationships (1-1, 1-many, many-many)
	Check player details		Read		5. Embed or Reference (refer to guidelines)
	Check transaction histories		Read		
	Check redemption history		Read		
	Check bank records		Read		

Player Functionalities | Embed Or Reference | Connections | Services-Connection

embed or reference

Guideline	Description	YES / NO	EMBED / REFERENCE	
Simplicity (简单)	Would keeping entities together lead to a simpler data model and code?	YES	EMBED	
Go Together (包含)	Do the entities have a "has-a", "contains" or similar relationship? e.g. a book has an author	YES	EMBED	
Query Atomicity (查询原子性)	Does the app query the entities together? e.g. do you query a book with author info together?	YES	EMBED	
Update Complexity (更新复杂度)	Are the entities updated together? do you update an author info together with book info?	NO	REFERENCE	
Archival (档案)	Should pieces of information be archived at the same time? an author may have multiple books	YES	EMBED	
Cardinality	Is there a high cardinality (current or growing) in child side of relationship? Unlikely that a list of authors for a book will change over time or become too long	NO	EMBED	Unlikely to go above 500
Data Duplication (数据重复)	Would data duplication be too complicated to manage and is undesired?	NO	EMBED	
Document Size	Would combined size of pieces of information take too much memory or transfer bandwidth?	NO	EMBED	
Document Growth (数据增长)	Would embedded piece grow without bound	NO	EMBED	Unlikely to go above 500
Workload	Are pieces of information written at different times in a write-heavy workload?	NO	EMBED	
Individuality (独立)	For child side of relationship, can entity exist by themselves without a parent? Can author (child) exist by itself without a book?	NO	EMBED	
		# EMBED	10	
		# REFERENCE	1	
	Data that is accessed together, should be stored together （一起查询的数据，应该存储在一起）			
	Sometimes a guideline can take priority over the others			

Convert to table