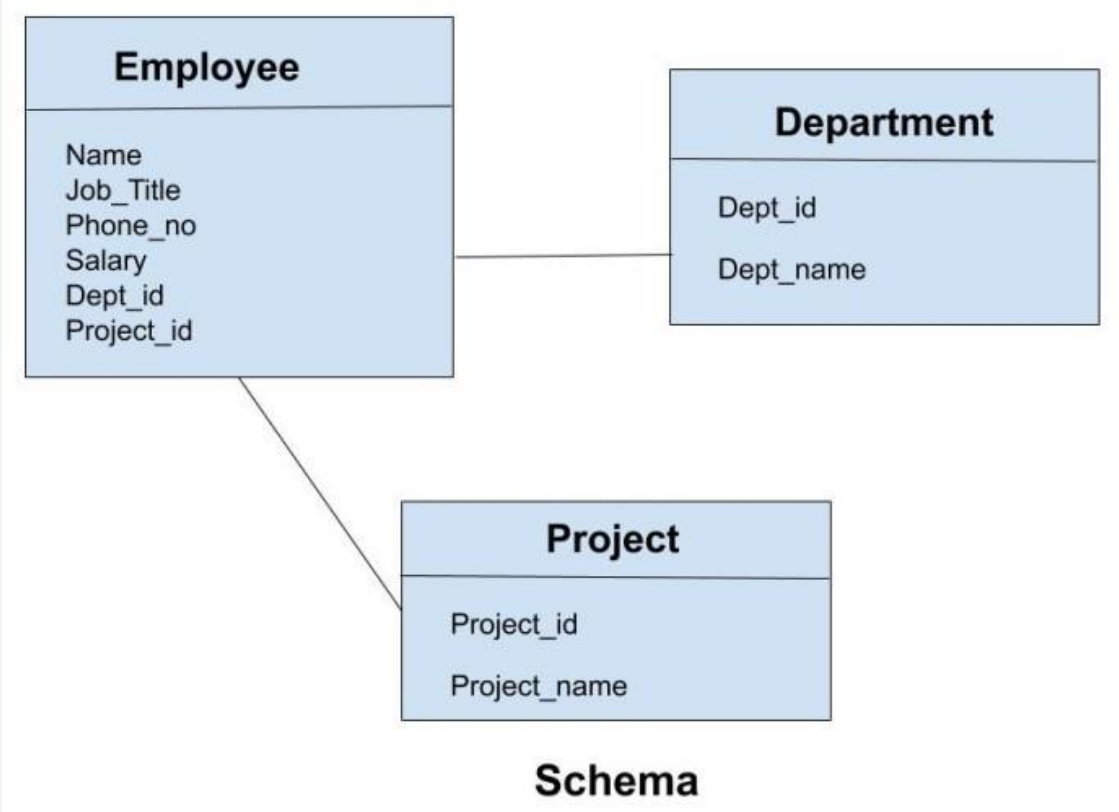
MODULE 5

DATABASE SCHEMA

- A database schema is an abstract design that represents the storage of your data in a database.
- It describes both the organization of data and the relationships between tables in a given database.
- Developers plan a database schema in advance so they know what components are necessary and how they will connect to each other.



Schema

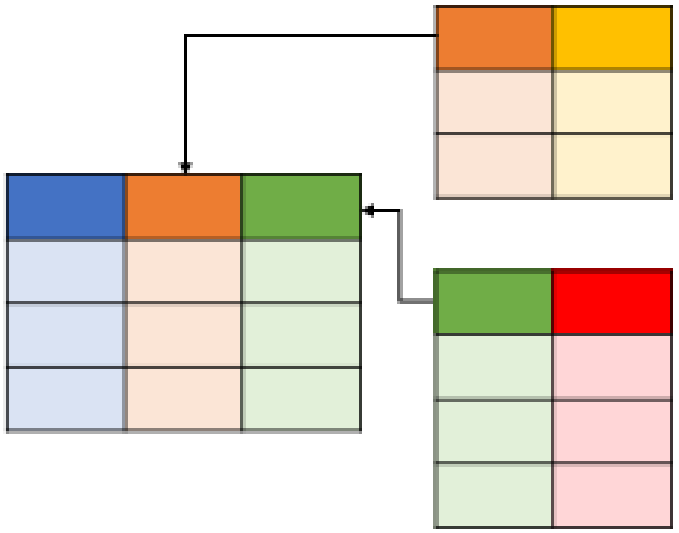
NOSQL DATABASE

- (aka "not only SQL") are non-tabular databases and store data differently than relational tables.
- store data in a format other than relational tables.
- come in a variety of types based on their data model.
- The main types are
  - document,
  - key-value,
  - wide-column, and
  - graph.
- They provide flexible schemas and scale easily with large amounts of data and high user loads.

NOSQL DATABASES - NEED

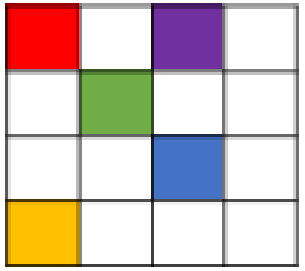
- As storage costs rapidly decreased, the amount of data that applications needed to store and query increased.
- This data came in all shapes and sizes — structured, semi-structured,— and defining the schema in advance became nearly impossible.
- NoSQL databases allow developers to store huge amounts of unstructured data, giving them a lot of flexibility.

SQL DATABASES

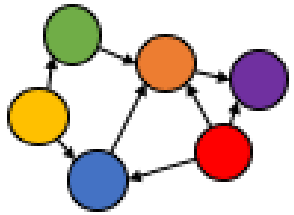


Relational

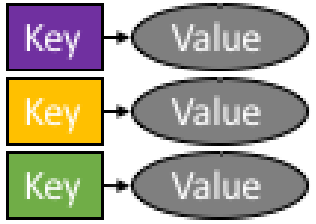
NoSQL DATABASES



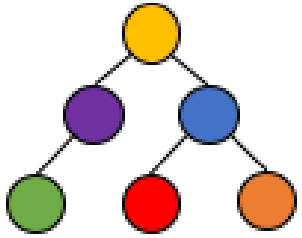
Column



Graph



Key-Value



Document

NOSQL VS RELATIONAL DATABASES

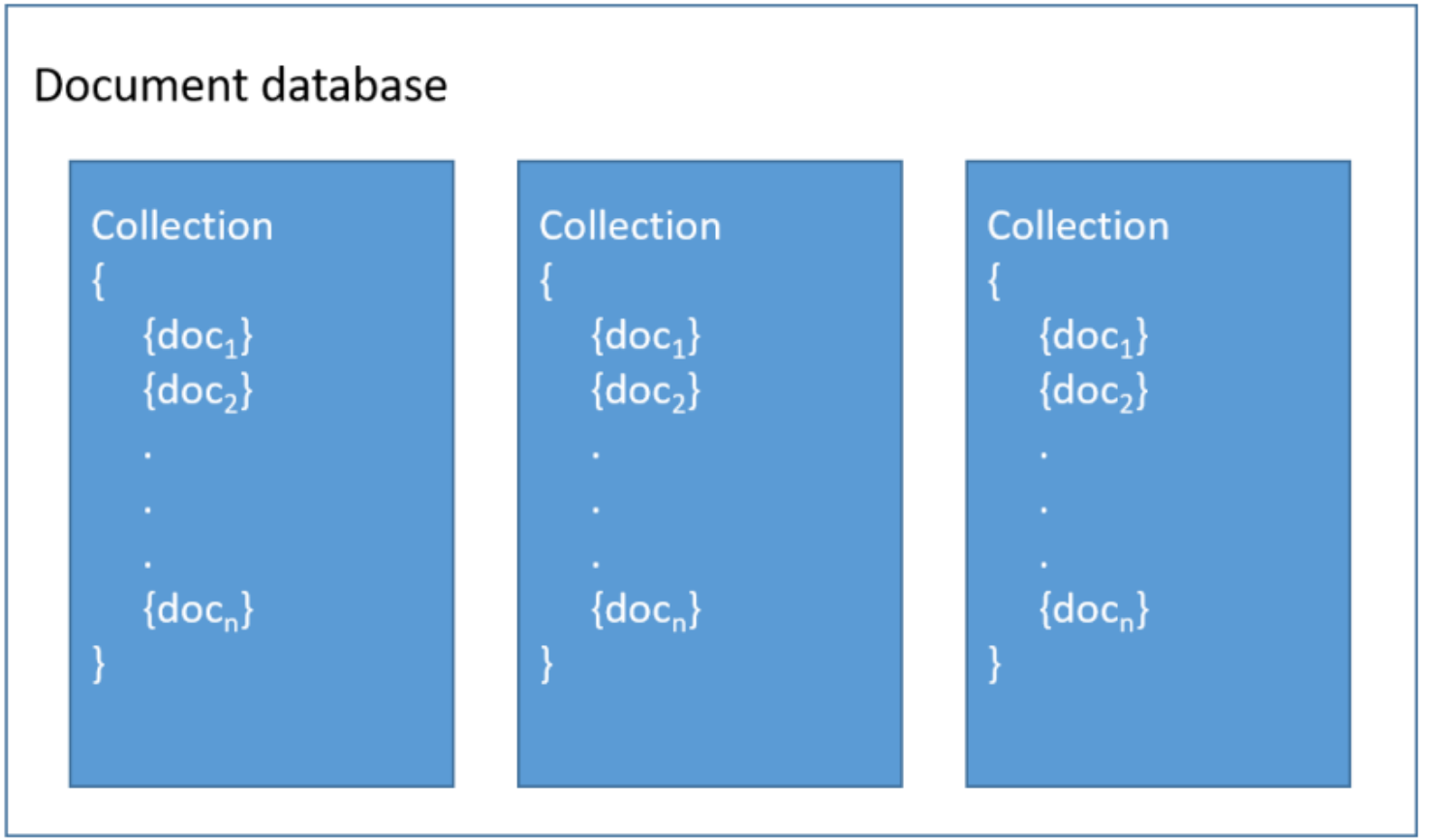
Relational databases

- Use tables/rows/columns
- Need a predefined schema / complicated to change
- slow queries when joining multiple tables
- vertically scalable
- Typically closed source

NoSQL

- Originally non-SQL/non-relational
- Not only SQL
- Non-relational databases
- Don't use tables/rows/columns
- Schema-less/easy changes
- Fast queries
- Horizontally scalable
- Open source

DOCUMENT DATABASE - OVERVIEW



Documents -> rows

Collections -> tables

DOCUMENTS

- Set of key-value pairs
- **Keys**: strings
- **Values**: numbers, strings, booleans, arrays or objects
- **Schemaless**: no need to specify the structure
- **Formats**: JSON, BSON, YAML, or XML

Document Queries

All the users who live in New York and like hiking

All the users older than 40

User's data by user_id

Documents - JSON format

```
{
  "user_id": 512,
  "name": "Carol",
  "last_name": "Harper",
  "email": "carolharper@datazy.com",
  "address": {
    "street": "123 Sesame Street",
    "city": "New York City",
    "state": "New York",
    "country": "USA"
  },
  "hobbies": [
    "hiking",
    "painting"
  ]
}
```

DOCUMENTS – POLYMORPHIC MODEL

```
{
  "user_id": 512,
  "name": "Carol",
  "last_name": "Harper",
  "email": "carolharper@datazy.com",
  "address": {
    "street": "123 Sesame Street",
    "city": "New York City",
    "state": "New York",
    "country": "USA"
  },
  "hobbies": [
    "hiking",
    "painting"
  ]
}
```

```
{
  "user_id": 513,
  "name": "Benjamin",
  "last_name": "Lieberman",
  "email": "benjaminlieberman@datazy.com",
  "date_of_birth": "07/04/1984",
  "hobbies": [
    "reading"
  ]
}
```

COLLECTIONS

- Sets of documents
- Store the same type of entities
- Organize documents and collections by thinking about the queries

POPULAR DOCUMENT DATABASES

PROS AND CONS OF DOCUMENT DATABASES

Advantages

- Flexibility
  - Don't need to predefine the schema
  - Documents can vary over time
- intuitive for developers
  - Natural way to work
  - JSON is human-readable
  - Less Coding
  - simpler and faster development
  - Easier for new developers
- Horizontal scalability
  - Sharding

Limitations

- more responsibility
  - care about data in the application code (check required email)
  - Care about redundant data (modify duplicated name)

WHEN TO USE DOCUMENT DATABASES

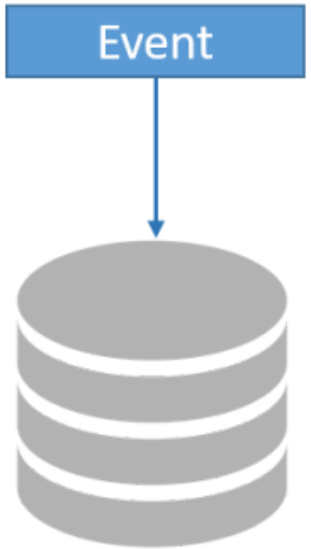
Suitable cases

- **Catalogs**
  - E-commerce web sites/applications store product information
  - Different attributes between the products
  - Embed related information


```
{
  "product_id": 879,
  "name": "Fashion shirt",
  "category": {
    "category_id": 15,
    "name": "Tops & t-shirts",
    "type": "Shirt"
  }
}
```

Suitable cases

- **Event logging**
  - Types of events:
    - User logging
    - Product purchase
    - Errors
    - ...
  - Sharding by:
    - Time
    - Type of event
    - ...




```
{
  "type": "info",
  "message": "user_logged",
  "user_id": 551,
  ...
}
```

Suitable cases

- **User profiles**

```
{
  "user_id": 512,
  "name": "Carol",
  "last_name": "Harper",
  "email": "carolharper@datazy.com",
  "address": {
    "street": "123 Sesame Street",
    "city": "New York City",
    "state": "New York"
  },
  ...
}
```

- Information may vary
- Document flexibility

Suitable cases

- **Content management systems**
  - Blogs, video platforms, etc.
  - Users' content
    - Comments
    - Images
    - Videos
    - ...


```
{
  "id": 458,
  "url": "myblog/datazy.com",
  "title": "How to write a blog entry at Datazy",
  "tags": [
    "Datazy",
    "Blog"
  ],
  "last10comments": [
    { "name": "Eliza", "comment": "Great!" },
    { "name": "Eric", "comment": "Thank you!"}
  ]...
}
```

Suitable cases

- **Real-time analytics**
  - Page views, unique visitors...
  - Easy to store the information

```
{
  "_id": "1000241",
  "hour": "Sat Jun 12 2021 16:40:00 GMT+0200 (EST)",
  "site": "datazy",
  "uniques": 5,
  "pageviews": 15,
  ...
}
```

UNSUITABLE CASES

- Very structured data
- Always have consistent data

WHAT IS MONGODB? - OVERVIEW

- MongoDB is a popular document database.
- can be installed locally or hosted in the cloud.
- It stores data in a type of JSON format called BSON.
- A record in MongoDB is a document, which is a data structure composed of key value pairs similar to the structure of JSON objects.

MONGODB - FEATURES

- MongoDB Query Language (MQL)
- db.users.find({"address.zipcode" : "423201"})
- Native drivers for programming languages : C#, Java, Python, Scala etc.
- Indexes on any field
- Joins in queries
- scale horizontally
- Replication (50 copies of our data)

MONGODB- PRODUCTS

- **MongoDB Compass:**
  - Free GUI
  - Explore schema, create queries visually...
- **MongoDB Atlas:**
  - Cloud service
  - AWS, Azure, Google Cloud
- **MongoDB Enterprise Advanced:**
  - Run MongoDB in our infrastructure

- **MongoDB Atlas Lake:**
  - Query and analyze data
  - AWS S3 and MongoDB Atlas
  - MQL
- **MongoDB Charts:**
  - Visualizations of the data
- **Realm Mobile Database:**
  - Store data locally on iOS or Android

LOCAL VS. CLOUD DATABASE

- MongoDB can be installed locally, which will allow you to host your own MongoDB server on your hardware.
  - You can download and use the MongoDB open source Community Server on your hardware for free.
- MongoDB Atlas is a cloud database platform.
  - This is much easier than hosting your own local database.
  - Sign up for a free MongoDB Atlas account to get started.

INSTALLING MONGODB

- MongoDB Community edition is a free, open-source database that is a popular option for powering modern applications
- Download the installer.
  - Download the MongoDB Community .msi installer from the following link:

    https://www.mongodb.com/try/download/community?tck=docs_server
  - In the Version dropdown, select the version of MongoDB to download.
- Run the MongoDB installer.
  - For example, from the Windows Explorer/File Explorer:
  - Go to the directory where you downloaded the MongoDB installer (.msi file).
  - Double-click the .msi file.
  - In the Platform dropdown, select Windows.
  - In the Package dropdown, select msi.
  - Click Download.
- Select **Install MongoD as a Service**

MongoDB Atlas
MongoDB Enterprise Advanced
MongoDB Community Edition
MongoDB Community Server
MongoDB Community Kubernetes Operator
Tools
Mobile & Edge

access to advanced functionality such as auto-scaling, serverless instances (in preview), full-text search, and data distribution across regions and clouds. Deploy in minutes on AWS, Google Cloud, and/or Azure, with no downloads necessary.

Give it a try with a free, highly-available 512 MB cluster.

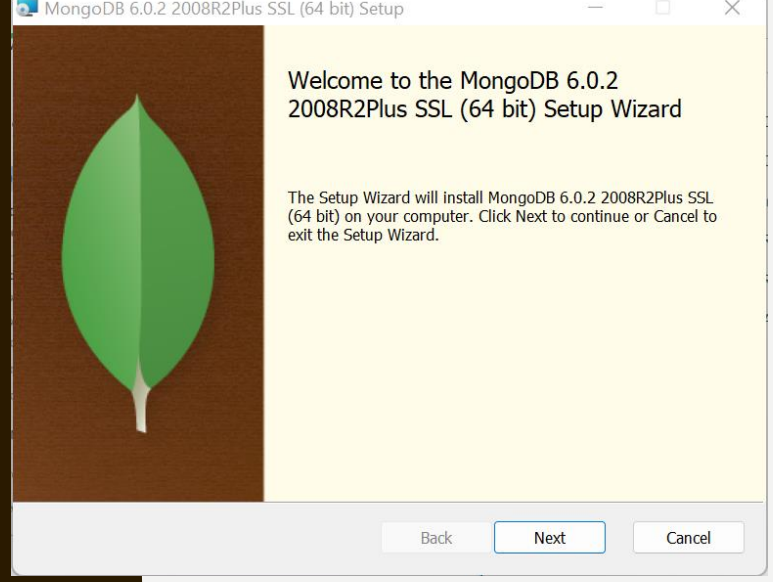
Version
6.0.2 (current)

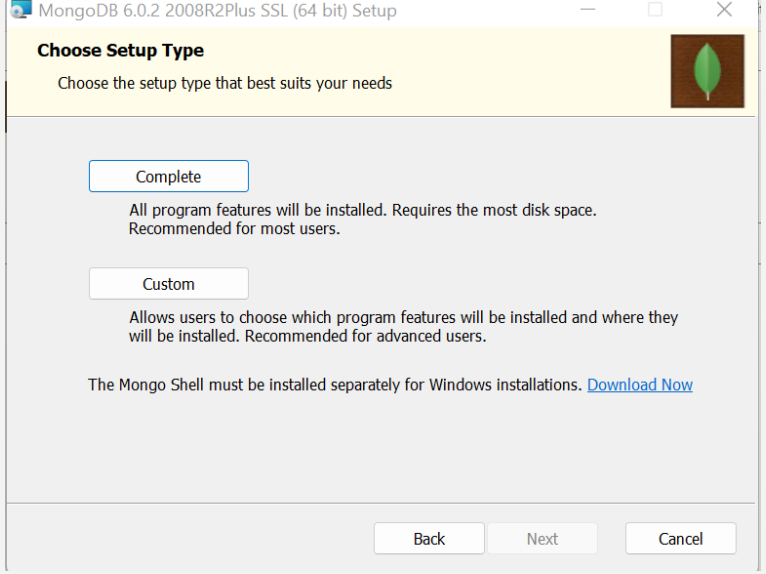
Platform
Windows

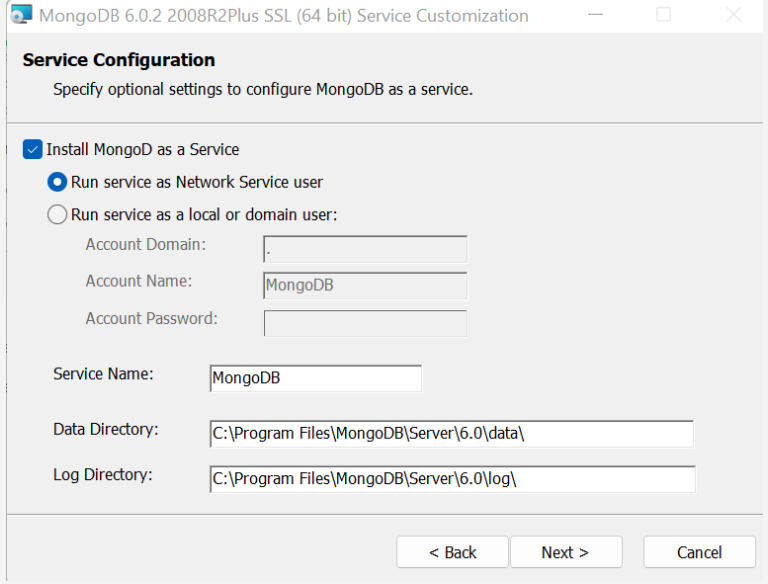
Package
msi

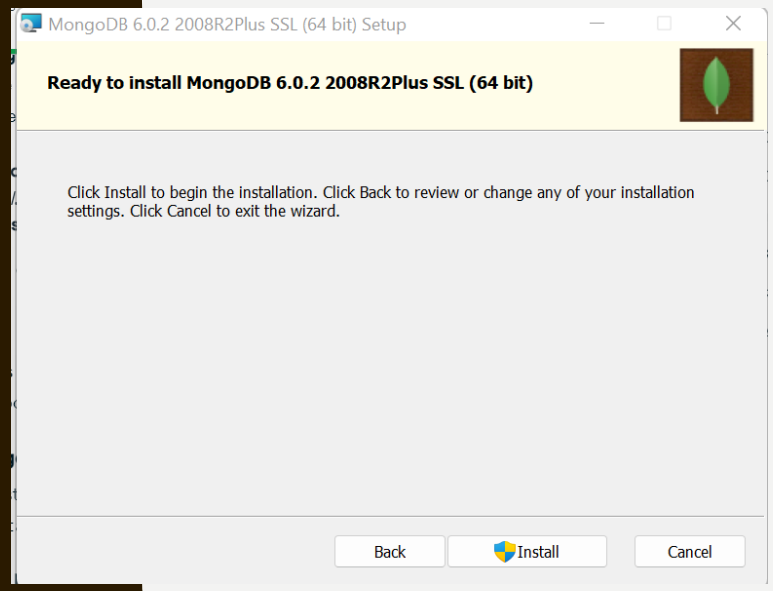
Download

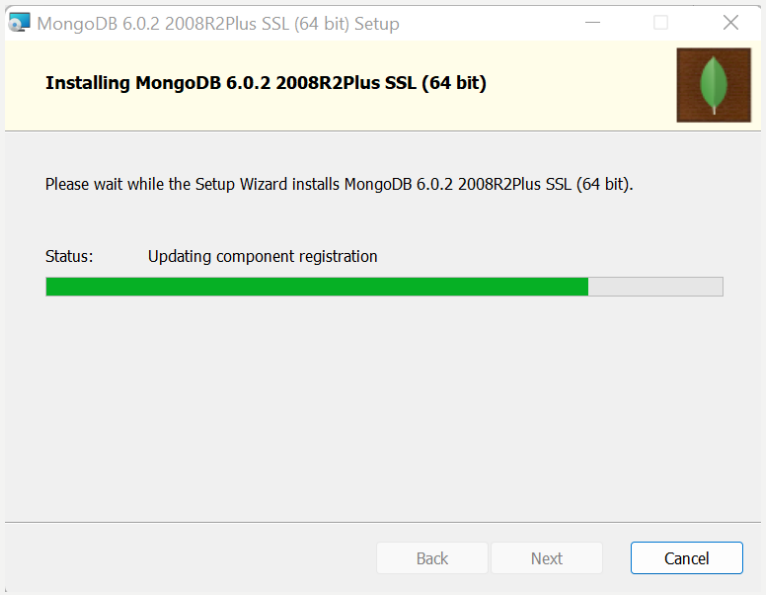
More Options

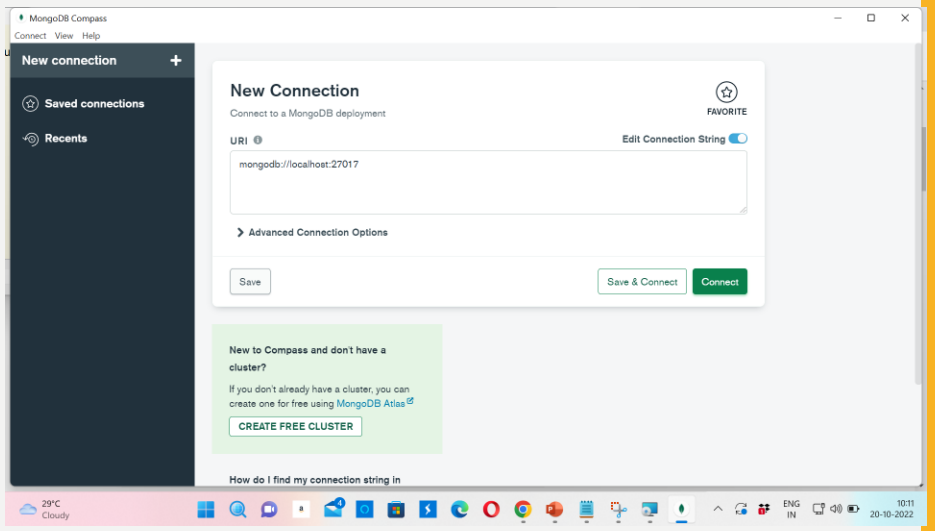

MongoDB 6.0.2 2008R2Plus SSL (64 bit) Setup
Welcome to the MongoDB 6.0.2 2008R2Plus SSL (64 bit) Setup Wizard
The Setup Wizard will install MongoDB 6.0.2 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.
Back
Next
Cancel


MongoDB 6.0.2 2008R2Plus SSL (64 bit) Setup
Choose Setup Type
Choose the setup type that best suits your needs
Complete
All program features will be installed. Requires the most disk space. Recommended for most users.
Custom
Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.
The Mongo Shell must be installed separately for Windows installations. Download Now
Back
Next
Cancel


MongoDB 6.0.2 2008R2Plus SSL (64 bit) Service Customization
Service Configuration
Specify optional settings to configure MongoDB as a service.
Install MongoD as a Service
Run service as Network Service user
Run service as a local or domain user:
Account Domain:
Account Name:
MongoDB
Account Password:
Service Name:
MongoDB
Data Directory:
C:\Program Files\MongoDB\Server\6.0\data\
Log Directory:
C:\Program Files\MongoDB\Server\6.0\log\
< Back
Next >
Cancel


MongoDB 6.0.2 2008R2Plus SSL (64 bit) Setup
Ready to install MongoDB 6.0.2 2008R2Plus SSL (64 bit)
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel


MongoDB 6.0.2 2008R2Plus SSL (64 bit) Setup
Installing MongoDB 6.0.2 2008R2Plus SSL (64 bit)
Please wait while the Setup Wizard installs MongoDB 6.0.2 2008R2Plus SSL (64 bit).
Status: Updating component registration
Back
Next
Cancel


MongoDB Compass
Connect View Help
New connection
Saved connections
Recents
New Connection
Connect to a MongoDB deployment
FAVORITE
URI
Edit Connection String
mongodb://localhost:27017
Advanced Connection Options
Save
Save & Connect
Connect
New to Compass and don't have a cluster?
If you don't already have a cluster, you can create one for free using MongoDB Atlas
CREATE FREE CLUSTER
How do I find my connection string in

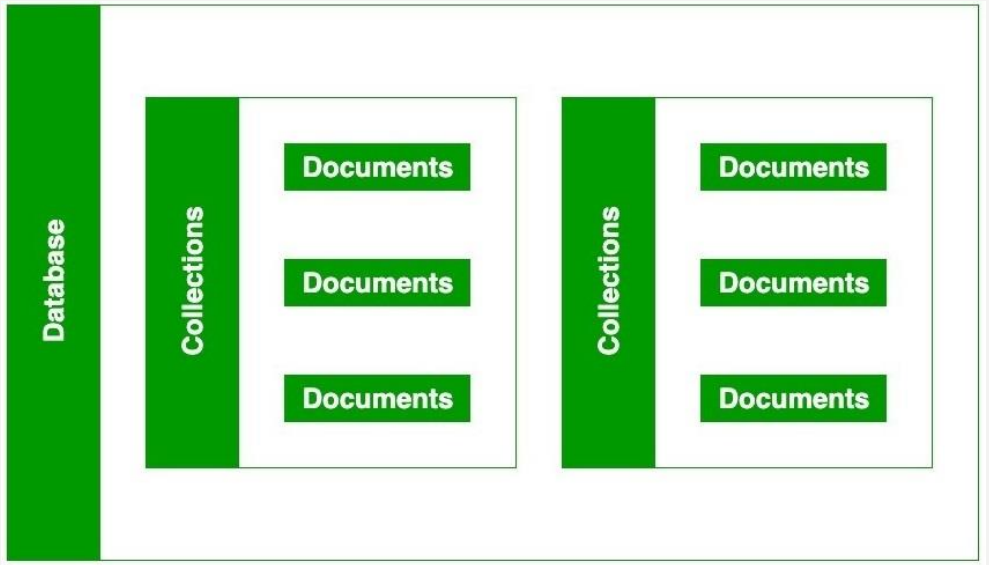
MONGODB COMPASS

- offers additional functionality like data visualization and performance profiling as well as offering CRUD (create, read, update, delete) access to data, databases, and collections.

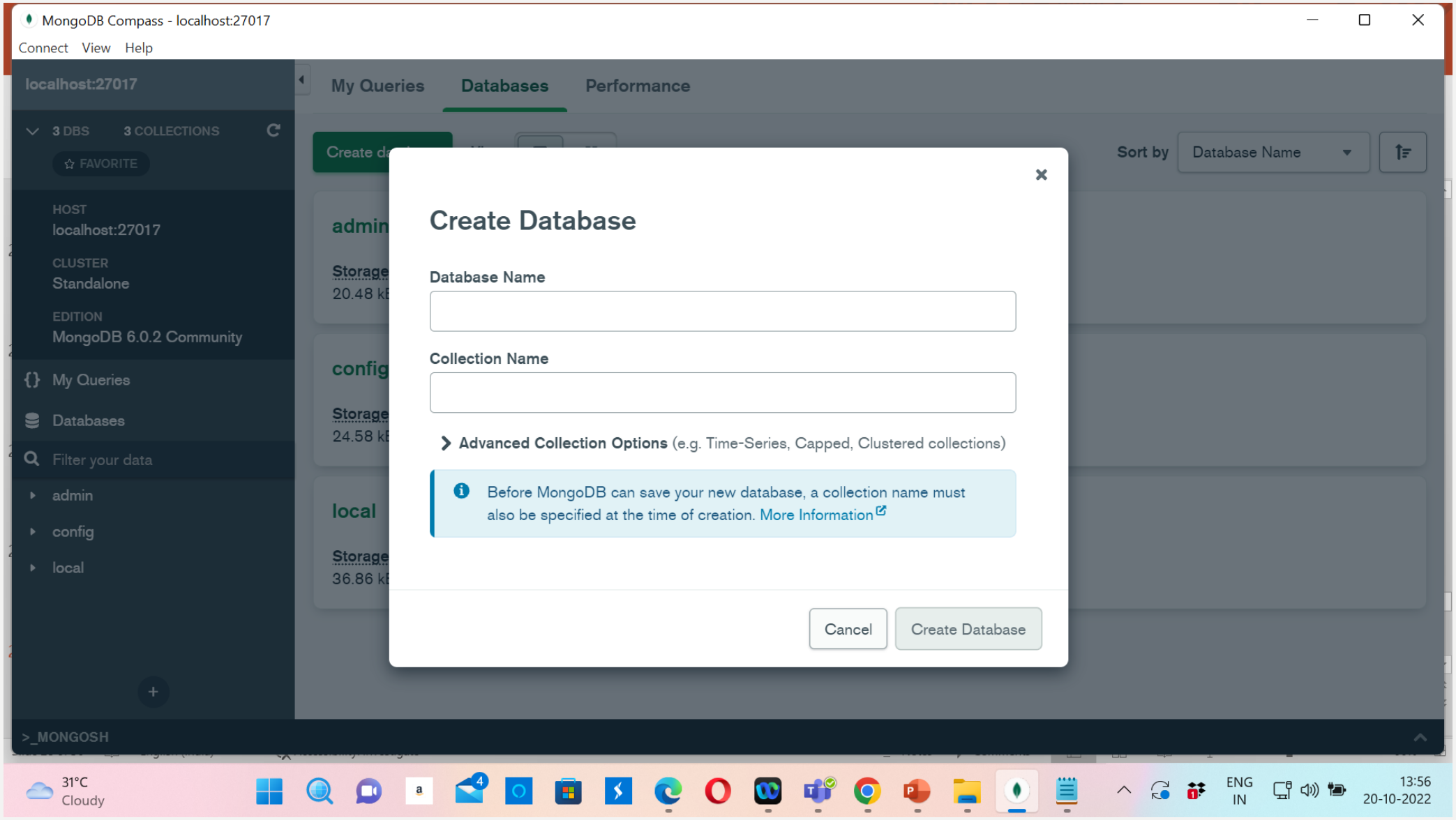
HOW MONGODB WORKS?

- MongoDB is structured on a client-server model where a server daemon accepts connections from clients and processes database actions from them. The server must be running for clients to connect and interact with databases.

- Data storage under MongoDB is different from traditional databases. A record in MongoDB is a document (a data structure composed of field and value pairs, similar to JSON objects) and documents are stored in collections (analogous to tables in RDBMS).



CREATING A MONGODB DATABASE



INSERT FIRST DATA

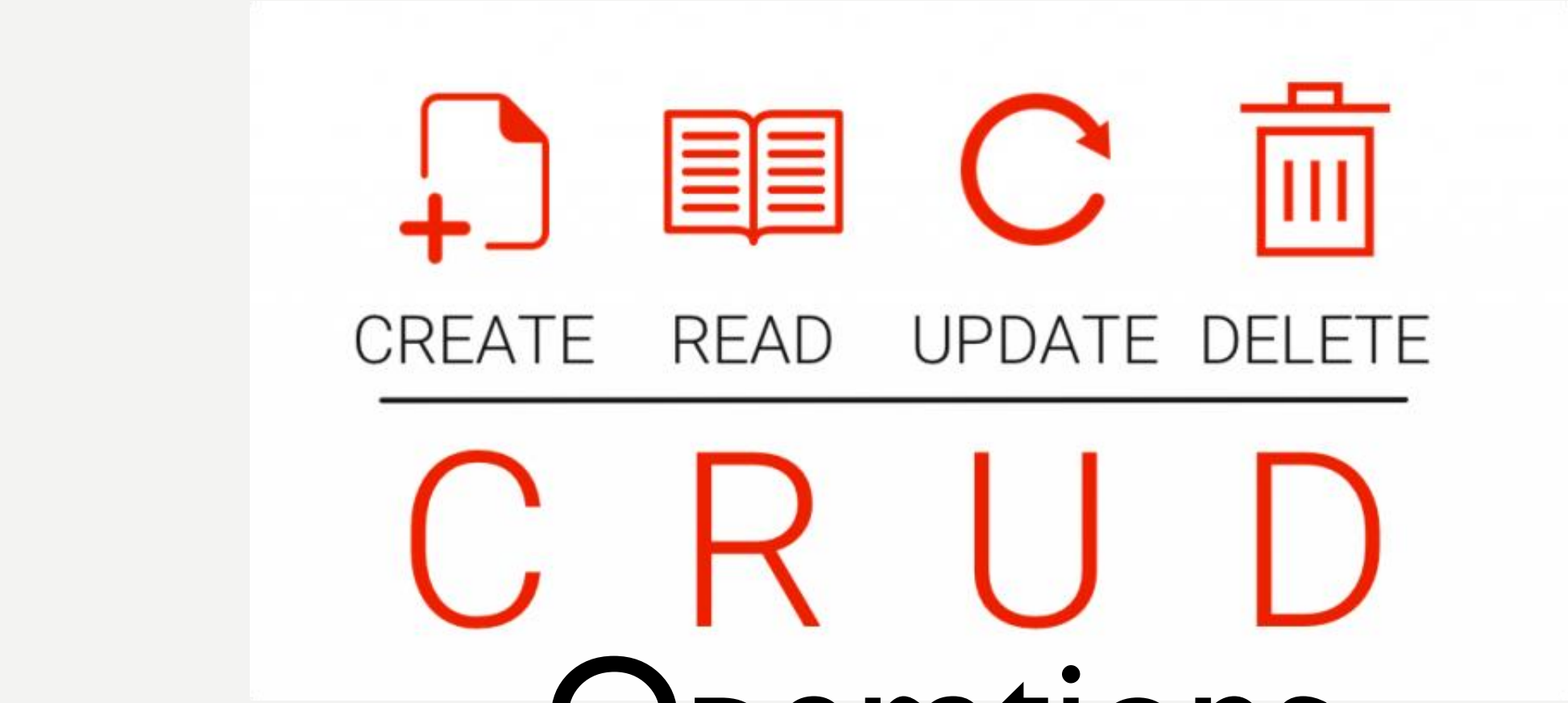
- Create a collection with name books and insert the following data

```
[
{ "_id" : 8752, "title" : "Divine Comedy", "author" : "Dante", "copies" : 1 },
{ "_id" : 7000, "title" : "The Odyssey", "author" : "Homer", "copies" : 10 },
{ "_id" : 7020, "title" : "Iliad", "author" : "Homer", "copies" : 10 },
{ "_id" : 8645, "title" : "Eclogues", "author" : "Dante", "copies" : 2 },
{ "_id" : 8751, "title" : "The Banquet", "author" : "Dante", "copies" : 2 }
]
```

Check out editing, deleting, and find options available on compass

MONGOSH

- MongoDB Shell is the quickest way to connect to (and work with) MongoDB. Easily query data, configure settings, and execute other actions with this modern, extensible command-line interface
- Can be downloaded from
  - https://www.mongodb.com/try/download/shell2
- Extract the zip file and update the path



Operations

https://www.mongodb.com/docs/manual/crud/

RELATIONSHIP BETWEEN DATABASE, COLLECTION AND A DOCUMENT

Database	humanResourcedb		
Collections	employees	addresses	departments
Documents	{ "firstName": "John", "lastName": "King", "email": "john.king@abc.com", "salary": "33000" }	{ "street": "H12", "house": "33", "city": "New York", "country": "USA" }	{ "name": "Technical", "totalEmployees": "100", }

MONGOSH COMMANDS

```
db.posts.insertOne({
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
})
```

- **show dbs** – displays all the databases
- **use <database name>** - to change or create to a database
- **db.createCollection(<collection name>)** – to create a collection
- **Insertion**
  - **db.collection_name.insertOne(object)** – to insert an object into a collection
  - **db.collection_name.insertMany()** can be used to insert multiple objects
- **find()** and **findOne()** – to find and select data from MongoDB collection
  - db.collection_name.find()
  - db.collection_name.findone()
  - db.collection_name.find({category:"News"})

MANGOSH COMMANDS

- Updation
  - updateOne() – update the first document that is found matching the provided query.
  - updateMany() - update all documents that match the provided query.
- Deletion
  - deleteOne() method will delete the first document that matches the query provided.
  - deleteMany() method will delete all documents that match the query provided.

IMPORTANT POINTS

- MongoDB reserves _id name for use as a unique primary key field that holds ObjectId type.
- A document field name cannot be null but the value can be.
- A document fields can be without quotation marks " " if it does not contain spaces, e.g. { name: "Steve"}, { "first name": "Steve"} are valid fields.
- Use the dot notation to access array elements or embedded documents.
- MongoDB supports maximum document size of 16mb.
- MongoDB collection can store documents with different fields. It does not enforce any schema.

UPDATE OPERATORS

- The following operators can be used to update fields:
  - $inc: Increments the field value
  - $rename: Renames the field
  - $set: Sets the value of a field
  - $unset: Removes the field from the document

- updateMany() : It update all documents in a collection with matching filter.
- updateOne() : It update only one top most document in a collection with matching filter.
- update() : By default, the update() method updates a single document. Include the option {multi : true} to update all documents that match the query criteria. Hence we can use it as both ways.

```
Syntax : db.collection.update(
<query>,
<update>,
{
upsert: <boolean>,
multi: <boolean>,
writeConcern: <document>,
collation: <document>
}
)
```

MONGODB QUERY OPERATORS

Comparison Operators

`$eq` : Values are equal
`$ne` : Values are not equal
`$gt` : Value is greater than another value
`$gte` : Value is greater than or equal to another value
`$lt` : Value is less than another value
`$lte` : Value is less than or equal to another value
`$in` : Value is matched within an array

Logical Operators

`$and` : Returns documents where both queries match
`$or` : Returns documents where either query matches
`$nor` : Returns documents where both queries fail to match
`$not` : Returns documents where the query does not match

PROJECTION

Project Fields to return from Query

- queries in MongoDB return all fields in matching documents by default.
- A projection document is used to restrict fields to return.
- db.collection.find() returns all fields in matching documents if you do not specify a projection document.
  - db.inventory.find()
  - db.inventory.find({ status: "A" })
- Return specific fields and the _id field only
  - A projection can explicitly include several fields by setting the <field> to 1 in the projection document.
  - db.inventory.find({ status: "A" }, { item: 1, status: 1 })
- supress _id Field
  - remove the _id field from the results by setting it to 0 in the projection
  - db.inventory.find({ status: "A" }, { item: 1, status: 1, _id: 0 })
- Return All But the Excluded Fields
  - db.inventory.find({ status: "A" }, { status: 0, instock: 0 })

EMBEDDED DOCUMENTS

- A document in MongoDB can have fields that hold another document. It is also called nested or embedded document.
- Example is shown right
  - Department and address field contains another document
  - the address field contains the phone field which holds a second level document.

Example: Embedded Document

```
{
    _id: ObjectId("32521df3f4948bd2f54218"),
    firstName: "John",
    lastName: "King",
    department: {
                _id: ObjectId("55214df3f4948bd2f8753"),
                name:"Finance"
            },
    address: {
        phone: { type: "Home", number: "111-000-000" }
    }
}
```

EMBEDDED DOCUMENTS

- An embedded document can contain upto 100 levels of nesting.
- Supports a maximum size of 16 mb.
- Embedded documents can be accessed using dot notation embedded-document.fieldname, e.g. access phone number using address.phone.number.

QUERYING THE EMBEDDED DOCUMENTS

- Matching an embedded or nested document
  - db.inventory.find({ size: { h: 14, w: 21, uom: "cm" } })
- query on nested field
  - db.inventory.find({ "size.uom": "in" })
- specify the match using query operator
  - db.inventory.find({ "size.h": { $lt: 15 } })

ARRAY

- Arrays can hold any type of data or embedded documents.
- A field in a document can hold array.
- Array elements in a document can be accessed using dot notation with the zero-based index position.
- The example document contains the skills field that holds an array of strings. To specify or access the second element in the skills array, use skills.1.

Example: MongoDB Document with an Array

```
{
    _id: ObjectId("32521df3f4948bd2f54218"),
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com",
    skills: [ "Angular", "React", "MongoDB" ],
}
```

QUERYING THE ARRAY

- Create a collection with the name : inventory2
- Insert the following data
  - db.inventory2.insertMany([
  - { item: "journal", qty: 25, tags: ["blank", "red"], dim_cm: [14, 21] },
  - { item: "notebook", qty: 50, tags: ["red", "blank"], dim_cm: [14, 21] },
  - { item: "paper", qty: 100, tags: ["red", "blank", "plain"], dim_cm: [14, 21] },
  - { item: "planner", qty: 75, tags: ["blank", "red"], dim_cm: [22.85, 30] },
  - { item: "postcard", qty: 45, tags: ["blue"], dim_cm: [10, 15.25] }
  -]);

QUERYING THE ARRAY

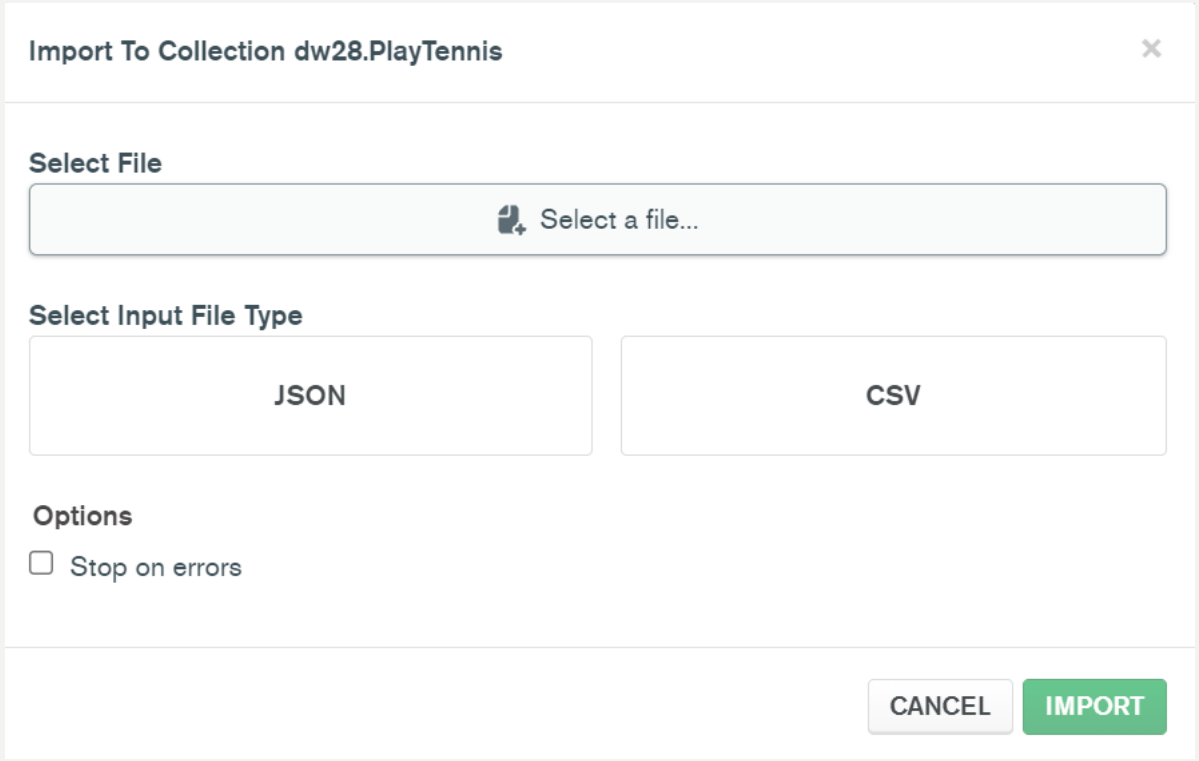
- Match an Array
  - queries for all documents where the field tags value is an array with exactly two elements, "red" and "blank", in the specified order:
  - db.inventory2.find({ tags: ["red", "blank"] })
  - to find an array that contains both the elements "red" and "blank", without regard to order or other elements in the array, use the $all operator:
  - db.inventory2.find({ tags: { $all: ["red", "blank"] } })
- Query an Array for an Element
  - To query if the array field contains at least one element with the specified value, use the filter { <field>: <value> } where <value> is the element value.
  - db.inventory2.find({ tags: "red" })
- To specify conditions on the elements in the array field, use query operators in the query filter document:
  - { <array field>: { <operator1>: <value1>, ... } }
  - db.inventory2.find({ dim_cm: { $gt: 25 } })

DATATYPES ALLOWED IN SCHEMAS / SCHEMA TYPES

- String
- Number
- Date
- Boolean
- Buffer
- ObjectId
- Mixed
- Array

FETCHING DATA FROM STRUCTURED FILES

- Data can also be updated from csv file



RELATIONSHIP BETWEEN DATA

- In MongoDB,
  - one-to-one,
  - one-to-many, and
  - many-to-many
- relations can be implemented in two ways:

1. Using embedded documents
2. Using the reference of documents of another collection

IMPLEMENTING RELATION

- Related data can be included using Embedded Document
- For example, you can include an address as an embedded document, as shown below.

```
db.employee.insertOne({
      _id: ObjectId("32521df3f4948bd2f54218"),
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com",
    salary: "33000",
    DoB: new Date('Mar 24, 2011'),
    address: {
                street:"Upper Street",
                house:"No 1",
                city:"New York",
                country:"USA"
            }
})
```

- Implement Relation using Reference is another way
- using the reference of the primary key field of documents of another collection.

```
db.address.insertOne({
      _id: 101,
    street:"Upper Street",
    house:"No 1",
    city:"New York",
    country:"USA"
})

db.employee.insertOne({
    firstName: "John",
    lastName: "King",
    email: "john.king@abc.com",
    salary: "33000",
    DoB: new Date('Mar 24, 2011'),
    address: 101
})
```

ONE TO ONE USING EMBED METHOD

```
{
    _id: "joe",
    name: "Joe Bookreader",
    address: {
                street: "123 Fake Street",
                city: "Faketon",
                state: "MA",
                zip: "12345"
              }
}
```

ONE TO ONE USING REFERENCE METHOD

```
// patron document
{
    _id: "joe",
    name: "Joe Bookreader"
}

// address document
{
    patron_id: "joe", // reference to patron document
    street: "123 Fake Street",
    city: "Faketon",
    state: "MA",
    zip: "12345"
}
```

ONE TO MANY

```
// patron document
{
   _id: "joe",
   name: "Joe Bookreader"
}

// address documents
{
   patron_id: "joe", // reference to patron document
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}

{
   patron_id: "joe",
   street: "1 Some Other Street",
   city: "Boston",
   state: "MA",
   zip: "12345"
}
```

```
{
   "_id": "joe",
   "name": "Joe Bookreader",
   "addresses": [
                 {
                   "street": "123 Fake Street",
                   "city": "Faketon",
                   "state": "MA",
                   "zip": "12345"
                 },
                 {
                   "street": "1 Some Other Street",
                   "city": "Boston",
                   "state": "MA",
                   "zip": "12345"
                 }
               ]
}
```

CONNECTING TO MONGODB FROM PYTHON

- PyMongo is a Python distribution containing tools for working with MongoDB, and is the recommended way to work with MongoDB from Python.
- To work with pymongo, first install pymongo distribution
  - pip install pymongo
- To check whether the library is properly installed or not, run
  - import pymongo
- Making a connection with a MongoClient
  - create a MongoClient instance to run the mongodb instance
  - from pymongo import MongoClient
  - client = MongoClient()

    (or)
  - client = MongoClient('localhost', 27017)

PYMONGO

- Getting the database names
  - client.list_database_names()
- Getting a database
  - A single instance of MongoDB can support multiple independent databases.
  - db = client.test_database

    (or)
  - db = client['test-database']
- Getting the collection names
  - db.list_collection_names()
- inserting documents into collection
  - insert_one() - used to insert a document.
  - insert_many() - used to insert many documents.
- Querying
  - find_one() - is used to get a single document
  - find() - returns all the documents
- count_documents() - to find how many documents match a query

```
>>> db.test.insert_one({'x': 1})
<pymongo.results.InsertOneResult object at 0x0000018A9EA06EE0>
>>> db.test.insert_one({'y': 2})
<pymongo.results.InsertOneResult object at 0x0000018A9D9F4B20>
>>> db.test.find_one()
{'_id': ObjectId('635e877308d42c347b2c4256'), 'x': 1}
>>> db.test.find()
<pymongo.cursor.Cursor object at 0x0000018A9E9A0CA0>
>>> for x in db.test.find():
...     print(x)
...
{'_id': ObjectId('635e877308d42c347b2c4256'), 'x': 1}
{'_id': ObjectId('635e878408d42c347b2c4257'), 'y': 2}
```

```
>>> db.inventory.count_documents({})
5
>>> db.inventory.count_documents({"status":"A"})
3
```

CHECK YOUR UNDERSTANDING

- MongoDB is a (relational / NoSQL) database.
- _____ is the command-line client of MongoDB Database? (Mongo Shell / Web application/ windows service)
- which command starts mongo shell? (mongo / mongod / mongosh / shell)
- Which command creates a new database? (create / switch / use / new)
- MongoDB collections are similar to ___ in RDBMS. (Views / Tables / Rows / Columns)
- Which method creates a new collection?
- MongoDB stores data as _____ document (JSON/ BSON/ XML/ YML)