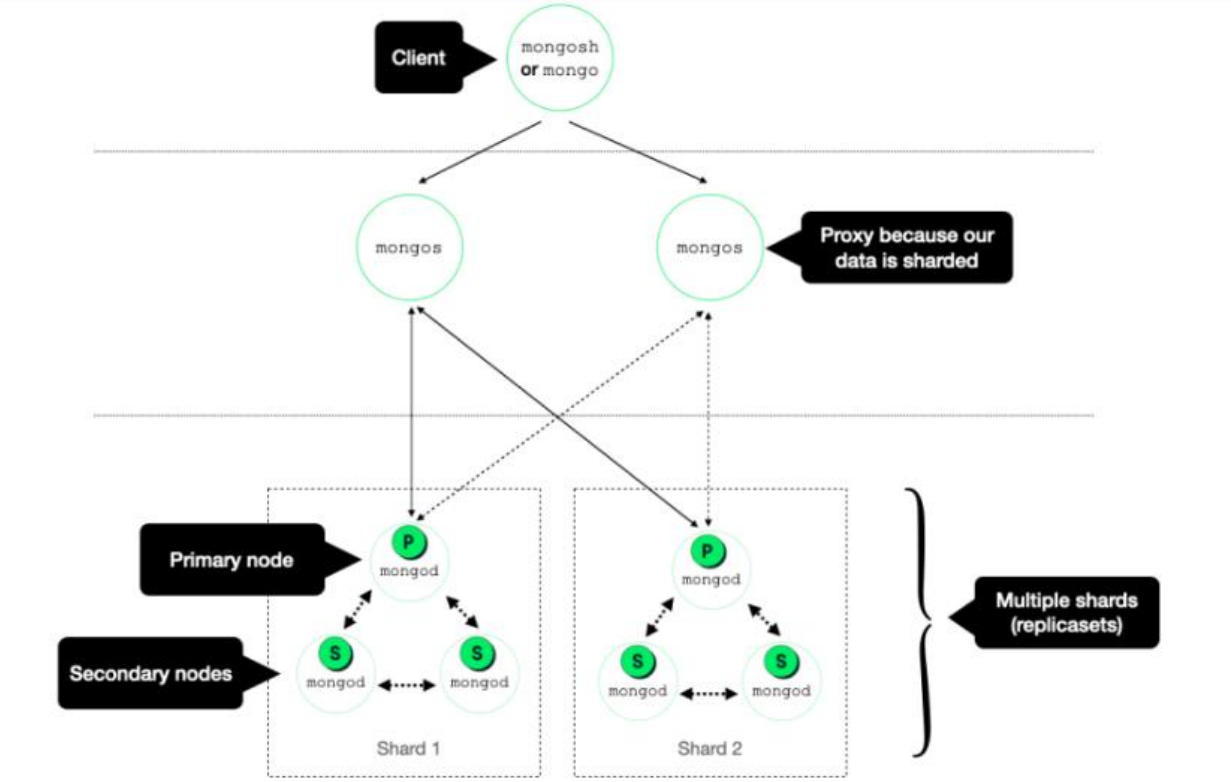
MongoDB



Terminology Breakdown

- You run **mongod** to start your server.
- The MongoDB server listens for connections from clients on port **27017**, and stores data in the */data/db* directory when you use **mongod**.
- MongoDB Atlas runs **mongod** for you, so you don't need to run the server yourself.
- **mongo** is the shell or the client. It's a JavaScript interface that you can use to interact with the MongoDB server (**mongod**).
- As of June 2020, it was superseded by the new Mongo Shell, **mongosh**
- **mongosh** has improved syntax highlighting, command history and logging.
- the shell, whichever you use, is just a way to communicate with your database cluster.
- **mongo** is a proxy that sits between the client application (mongo/mongosh) and a sharded database cluster, that is multiple mongod replica sets.

In earlier versions:

- **mongod** to initiate
- **mongo** to open shell

in newest version (using shell):

- only **mongosh**

Refer MongoDB CRUD

```
> help
> show dbs
> use shopDB
```

Show current db

```
> db
```

Create – 'products' here is the collection name

```
> db.products.insertOne({_id: 1, name: "pen", price: 1.20 })
> db.products.insertOne({_id: 2, name: "pencil", price: 0.80 })
```

Show all collections in the current db

```
> show collections
```

Find all entries in the 'products' collection

```
> db.products.find()
```

Finding a specific query using query operators

```
> db.products.find({name: "pen"})
> db.products.find({price: {$gt: 1}})
```

Projections – return only a specific field. 1 for yes and 0 for no. _id always comes by default

```
> db.products.find({_id: 1}, {name: 1, _id: 0})
```

Update Operations

```
> db.products.updateOne({_id: 1}, {$set: {stock: 32}})
> db.products.updateOne({_id: 2}, {$set: {stock: 12}})
```

Delete Operations

```
> db.products.deleteOne({_id: 2})
```

Relationships

```
> db.products.insert(
      {
      _id: 3,
      name: “rubber”,
      price: 1.30,
      stock: 43,
      reviews: [
            {
                  authorname: “Sally”,
                  rating: 5,
                  review: “best rubber ever”
            }
            {
                  authorname: “Paul”,
                  rating: 4,
                  review: “Fantastic”
            }
      ]
      }
)
```

Relationships

```
{
	_id: 1,
	name: “pen”,
	price: 1.20,
	stock: 32
}

{
	_id: 2,
	name: “pencil”,
	price: 0.80,
	stock: 12
}

{
	orderNumber: 3243,
	productsOrdered: [1, 2]
}
```

Mongoose

1. Install mongoose

```
npm install mongoose
```

2. Require mongoose

```
const mongoose = require("mongoose");
```

3. Connect mongoose. Setup database (dbname - wikiDB)

```
mongoose.connect("mongodb://localhost:27017/wikiDB");
```

4. Initialize schema (where a document has 2 fields: title and content)

```
const articleSchema = new Schema({
  title: {
    type: String,
    required: true
  },
  content: {
    type: String,
    required: true
  }
});
```

5. Setup collection named articles

```
const Article = mongoose.model("Article", articleSchema);
```

Create

```
const <constantName> = new <ModelName> ({
    <fieldName> : <fieldData>,
    ...
});
<constantName>.save();
```

(postman -> helps test APIs and send requests. View and create dbs using Studio 3T mongodb GUI)

Example: *app.post and Create -> analogous (RESTful API)*

```
app.post("/articles", function(req, res){

  // console.log(req.body.title);
  // console.log(req.body.content);

  const newArticle = new Article({
    title: req.body.title,
    content: req.body.content
  });

  newArticle.save(function(err){
    if(!err){
      res.send("successfully added a new article");
    } else {
      res.send(err);
    }
  });
});
```

Reading from Database

```
<ModelName>.find({conditions}, function(err, results){
      // use the found results docs
});
```

Find all documents

```
<ModelName>.find({}, function(err, results){
      // use the found results docs
});
```

Example: *app.get and Read -> analogous (RESTful API)*

```
app.get("/articles", function(req, res){
      Article.find({}, function(err, foundArticles){
            // console.log(foundArticles);
            if(!err){
                  res.send(foundArticles);
            } else {
                  console.log(err);
            }
      })
});
```

Delete

```
<ModelName>.deleteMany(
      {conditions},
      Function(err){
      }
);
```

Delete all documents

```
<ModelName>.deleteMany(
      Function(err){
      }
);
```

Example: *app.delete and Delete -> analogous (RESTful API) -> use postman to send delete req*

```
app.delete("/articles", function(req, res){
  Article.deleteMany(function(err){
    if(!err){
      res.send("successfully deleted all articles");
    } else {
      console.log(err);
    }
  })
});
```

Read/ Find One Document

```
<ModelName>.findOne(
      {conditions},
      Function(err, result){
            // use the found result
      }
);
```

Update Document

```
<ModelName>.replaceOne(
      {conditions},
      {updates},
      {overwrite: true}
      function(err, results){}
);
```

```
<ModelName>.updateOne(
      {conditions},
      {updates},
      function(err, results){}
);
```

Delete One Document

```
<ModelName>.deleteOne(
      {conditions},
      Function(err){}
);
```

Read, Update, Delete Single Entries

```
app.route("/articles/:articleTitle")

.get(function(req, res){
// if url = localhost:3000/article/JSON
// req.params.articleTitle = "JSON"
Article.findOne(
      {title: req.params.articleTitle},
      function(err, foundArticle){
            if(foundArticle){
                  res.send(foundArticle);
            } else {
                  res.send("No matching article found.");
            }
      });
})

.put(function(req, res){
Article.replaceOne(
      {title: req.params.articleTitle},
      {title: req.body.title, content: req.body.content},
      {overwrite: true},
      function(err){
            if(!err){
                  res.send("successfully updated article");
            } else {
                  console.log(err);
      }
      }
);
})
```

```
.patch(function(req, res){
  Article.updateOne(
    {title: req.params.articleTitle},
    //{$set: {title: "", content: ""}}
    {$set: req.body},
    function(err){
      if(!err){
        res.send("Successfully updated article");
      } else {
        res.send(err);
      }
    }
  );
})

.delete(function(req,res){
  Article.deleteOne(
    {title: req.params.articleTitle},
    function(err){
      if(!err){
        res.send("Successfully deleted article");
      } else {
        res.send(err);
      }
    }
  );

});
```