No Sql:

MongoDb

MERN

M - MongoDB

SQL - Structured Query database

Show dbs

use first class

db

first .db. dropDatabase ()

Show Collections

db. createCollection ("users")

db. users. insert ({"batch": "web-15"})

db. users. find ()

db. users. insert ({"class": "mongodb intro"})

db. users. find ()

Manoj kumar

Page No.:
Date : / /

Creating - database

use web15 ↵

show dbs ↵

db ↵

Collection

.db.createCollection("Students") ↵

show Collections ↵

Document

.db.Students.insert({"name": "prashant"})

db.Students.insert({"name": "Amir"}); ↵

More than 1 line insert

db.Students.insertMany([{"name": "Visal"}
{"name": "Avinash"}]) ↵

Show all insert data

db.Students.find({}) ↵

find / show

.db.students.find({}).pretty() ↵

db.Student.find({"city": "pune"}).
pretty(); ↵

db.student.findOne({"age": 23}) ↵

Page No.:
Date : / /

Update

db.students.update({"name": "prashant"}, {"age": 25}) ←

यह Replace कर देगा

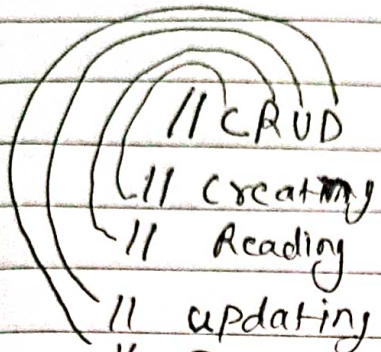
db.students.update({"name": "prasant"}, {$set: {"age": 25}})

Delete / Remove

db.students.remove({"name": "John"}) ←

1st

Mongo introduction Live



get

post

get

put or patch

deleting

Mongo ↵

Show dbs ↵

* Create data base / or select Database

use test ↵

* Check which data base i am using.

→ .db ↵

* Drop and Remove data base

db.dropDatabase() ↵

* Show Collections inside test

show Collection ↵

* Create Collection

db.createCollection("users") ↵

* Insert documents

db.users.insert({"batch": "web_15"}) ↵

Show

db.users.find() ↵

Page No.:
Date : / /

Creating :-

① Create a Database :-

→ use web15

* Show data base :-

→ show dbs ↵

db ↵

* Collection If collection is not present then create it

How to create Collection :-

db.createCollection ("students") ↵

* How to Show Collection :-

→ Show Collections ↵

* How to Create Collection & Document :-

How to insert data

db.students.insert({"name": "prashant"}) ↵

db.students.insert({"name": "Amir"}) ↵

* How to insert many document in one time :-

db.students.insertMany([{"name": "vishal"},{"name": "Avinash"}]) ↵

Manoj Kumar

Page No.:
Date : / /

How to read that :-

{ : db.students.find({}) ↵

Insert many information

db.students.insert({"name":"sai", "age":24,
"city":"pune"});

☆ How to show clean format all data

db.students.find({}).pretty() ↵

☆ find through some criteria:

db.students.find({"age":23}).pretty() ↵

☆ find where city is pune

→ db.students.find({"city":"pune"}).
pretty() ↵

☆ find only one data same age person data

db.students.findOne({"age":23}) ↵

Update, Modify, Change

☆ Replace the data

~~db.update~~

db.students.update({"name":"prashant"},{
"age":25}) ↵

Page No.:
Date : / /

* Update the data:

db.student.update({"name": "Amir"}, {$set: {"age": 25}}) ↵

* add city and country any object

db.students.update({"name": "Ranjith"},
{$set: {"city": "pune", "country": "india"}}) ↵

How to update many :-

db.students.updateMany({"age": 23}, {$set: {"eligible": true}}) ↵

* How to update Every one

db.students.updateMany({}, {$set: {"Country": "India"}});

Update / Remove

Note: { Updating and deleting is Destructive → once the change is done, you can't undo it. }

db.students.remove({"name": "John"}) ↵

Check

db.students.find({}).pretty() ↵

db.help()
db.student.help()

Manoj Kumar

Page No.:
Date : / /

Document को Delete filed को Delete करना है —

db.students.remove({"name":"Ranjith"},
{$unset:{"eligible":true}})

<, <=, !=, ==, >, >=
lt, lte, ne, eq, gt, gte

find who age is 25

db.students.find({"age":{$eq:25}})
.pretty()

db.Students.find({"budget":{$eq:15}})
gte

81 3/2021

Page No.:
Date : / /

Mongo Advanced (Live)

Mockaroo.com for dummy data download

import "download files" files

go to → Mongodb compass → Select database → Create Collection → Select Collection → import → chose type → JSON → Select files → import.

Check Comand promt

db ↵

Show Collection

db.User.count() ↵ // How many items available.

db.user.find({}).pretty() ↵

// find only one first data

db.users.findOne({}) ↵

More than two Condition then used it

Manoj Kumar

Page No.:
Date : / /

- Count

It is used to count data.

db.users.count() ↵

both condition is satisfied the execute.

And finding which document there both condition is true.

```
let num = 5;
if(num > 5 && num % 5 == 0)
  {
    console.log(true)
  }
else
  {
    console.log(false);
  }
```

① db.users.find({$and:[{gender:"Male"},{id:{$lt:20}}]}).pretty() ↵

② db.users.find({$and:[{gender:"Female"},{id:{$gte:20}}]}).pretty() ↵

// 3 Condition Check Throu And

③ db.users.find({$and:[{gender:"MALE"},{id:{$gt:20}},{id:{$lt:30}}]}).pretty() ↵

② OR किसी भी Condition के Satisfied होने पर Execute होगा

db.users.find({$or:[{gender:"MALE"},{id:{$lt:10}}]}).pretty() ↵

Page No.:
Date : / /

③ Not operator

```
db.users.find({gender:{$not:{$eq:"Female"}}}).pretty() ←
```

② `db.users.find({id:{$not:{$gte:10}}}).pretty() ←`

③ `db.users.find({gender:{$not:"MALE"}})←`

NOR → AND + OR = NOR

दोनों condition fail होगा है तो print करो.
All condition is false - those element is get

```
db.users.find({$Nor:[{gender:"male"},{
        id:{$gte:10}}]}).pretty() ←
```

In — जितनी वैल्यू एरे में available है उनको show करेगा है तो.

```
db.users.find({gender:{$in:["male","Female"]}}).pretty() ←
```

Mongo Advanced Live part 2

```
db ←
web15
Show Collections
db.users.find({}).pretty() ←
```

Mongo Session

Page No.:
Date : / /

Select

केवल नाम देखना है।

```
db.users.find({}, {first_name: 1}).pretty()
```

All the document id less then ten only name show

```
db.users.find({id: {$lte: 10}}, {first_name: 1})
```

```
db.users.find({id: {$lte: 10}}, {first_name: 1, _id: 0}).pretty()
```

0 से शुरू होता है।

Sort

Ascending order 1

```
db.users.find({}).sort({id: 1}).pretty()
```

descending order -1

```
db.users.find({}).sort({first_name: 1}).pretty()
db.users.find({}, {first_name: 1, second_name: 1, _id: 0}).sort({first_name: 1}).pretty()
```

Limit

दिए गए 5 ही डाटा देखना है।

```
db.users.find({}, {first_name: 1}).limit(4).pretty()
```

```
db.users.find({gender: "female"}, {first_name: 1, id: 0}).limit(5).pretty()
```

SKIP

```
db.users.find({}, {id: 1}).skip(5).limit(5).pretty()
```

यदि किसी का second name हो

db.users.find ({ Second name : { $exists : true } }).

db. users. find (gender : "male" }). limit (10). pretty ( )

Connet mongoDB data base with express

Compass , atlast) Rm x- (It is used to delete all file using terminal)

Terminal vs Code

⇒

npm init -y ↵

npm i express ↵

npm i mongoose ↵ // It is help to Conect Database mongos

npm i Nodemon -g ↵

pakcey . json में script ["server" : "nodemon server . js"]

```
Const express = require ("express");
Const mongoose = require("mongoose");
Const app= express();

app.get ("/users" (req, res) => {
        return res. send ("users")
    })

app. listen( 5000, () => {
        Console. log ("listining at 5000")
    })
```

Check browser :- localhost : 5000/users

Nodemon . index . js

Manoj kumar

// Connect MongoDB

Const ConnectDB = () => {
// Mongoose.Connect("mongodb://127.0.0.1:27017/
Mongoose.Connect("mongodb://localhost:27017/abis");
// mongoose.Connect(url);

// Create Schema - basicaly a structor in our data base

Const usesSchema = Mongoose.schema({
id: Number,
first_name: String,
last_name: String,
email: String,
gender: String,
ip_address: String,
})

// model - which collection refers to like users student
// users - user

Const User = mongoose.model("user", usesSchema)
// db.user

app.get("/users", async(req, res) => {
// Const user = User.find({}).pretty()
Const userData = await User.find({}).lean()
.exec()
return res.send(userData)
console.log(userData)
})

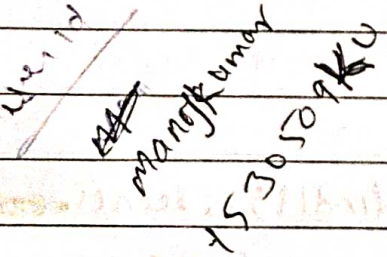
// ...skip().limit().sort().pretty()

1:29

```
app.get("/users", async (req, res) =>
{
const userData = await User.find({}, { first_name: 1, _id: 0 }).limit(10).
lean().exec();
console.log(userData)
return res.send(userData)
})

app.listen(5000, async() => {
try {
await connectDB();
}
catch {
console.log("Something went wrong")
}
console.log("Listening at 5000")
})
```

If we have our mongoDb data base online then download mogodb Atlas



9/3/2021

Manoj Kumar

Page No.:
Date : / /

Express Relationship

There are 3 type of relationship:-

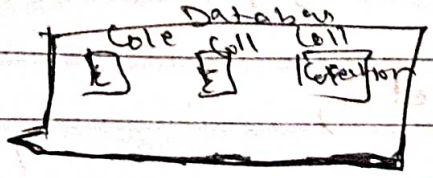
(1) one to one :- Here 1 document of 1 Collection is related to 1 document of another Collection like user and students Collection where 1 user can be 1 student and 1 Student also can be 1 user only.

(2) One to Many :- Here 1 document of 1 Collection is related to Many documents of another Collection like users and posts Collections where 1 user can write Many posts but 1 post can be written by 1 user only.

(3) Many to Many :- Here Many documents of 1 Collection are related to Many documents of another Collections like posts and tags Collections (like in StackOverflow) when 1 post can have Many tags and 1 tag can also belong to Multiple posts.

Note:

Database is a set of collection, collection is a set of document



npm ; nodemon ; g

Page No.:
Date : / /

npm init -y ↵
npm i express mongoose ↵

Now create index.js file →

```
Const express = require("express");
Const Mongoose = require("Mongoose");

Const app= express();

Const Connect = () =>{
    return Mongoose.Connect(
    "...itles Connection Pass here");
};

// User Schema
// Step1: Creating the Schema.

Const userSchema = new mongoose.Schema({
    firstName: {type: String, require: true},
    LastName: {type: String, require: false},
    email: {type: String, require: true, unique: true},
    password: {type: String, require: true,},
}, {timestamps: true});

// Step2: Creating the model

Const User= Mongose.mode("user", userSchema); // user= users.
// user.find() → // db.user.find()

app.listen(5000, async () => {
    try {
        await Connect();
    } catch(err) {
        console.log(err);
    }
```

// user.find() // db.user.find
// user.create() // db.create.find

// POST SCHEMA

// Step1:- Create the Schema

```
Const postSchema = new mongoose.Schema({
        {
        title: {type: String, required: true},
        body: {type: String, required: true},
        },
        {
        timestamps: true    // createdAT, updateAT
        }
    });
```

// Step2: Creating the model

```
Const post = mongoose.model("post", postSchema);
                    // post => posts
```

// Comment Schema

// Step1:- Create the Schema

```
Const CommentSchema = new mongoose.Schema({
        {
        body: {type: String, required: true},
        },
        {
        timestamps: true,
        } );
```

// Step2:- Create the model

```
Const Comment = Mongoose.model("comment",
                    CommentSchema);
                    // Comment => Comments.
```

Page No.:
Date : / /

```
Const express = require("express");
Const Mongoose = require("Mongoose");

Const app = express();
app.use(express.json());          // It is mendratory
                                  // for add data

Const Connect = () => {
    return Mongoose.Connect("mongodb+srv://
    dhval:dhaval_123@cluster0.1juvz.mongodb.net/
    web15-atlas?retryWrites=true&w=majority");
};

// USER SCHEMA
// Step1:- creating the Schema

Const UserSchema = new Mongoose.Schema(
    {
    firstName: {type: String, required: true},
    lastName: {type: String, required: false},
    email: {type: String, required: true, unique:
                                        true},
    password: {type: String, required: true},    //
},
{
    VersionKey: false,
```

```
    timestamps : true,      // createdAT, updatedAt
  }
  );
)
// Step 2: Creating the model

Const User = Mongoose.model("user", userSchema);
                              // user => users

// POST SCHEMA
// Step 1:- Creating the schema

Const postschema = new mongoose.schema(
      {
  title : { type: String, required: true },
  body : { type: String, required: true },
  userId: {
        type: Mongoose.schema.Type.objectId,
        ref : "user",
        required: true,
      },
    },
    {
    versionkey : false,
    timestamps: true,    // CreatedAT, updateAT
    }
);

// Step 2:- Creating the Model

Const post = Mongoose.model("post", postschema);
                              // post => posts
```

// COMMENT Schema
// Step 1 :- Creating the schema

```
Const CommentSchema = new mongoose.Schema(
    {
    body : { type : String , required : true },
    postId : {
            type : mongoose.schema.Type.ObjectId,
            ref : "post",
            required : true,
        },
    },
    {
    versionKey : false,
    timestamp : true,
    }
    );
```

// Step : 2 :- Creating the model

```
Const Comment = mongoose.model("Comment",
                    CommentSchema);   // Comment => Comments.
```

// CRUD operations

// get => getting data from the server
// post => adding data to the server
// put / patch => updating data in the server
// delete => deleting data from the server

Manoj kumar

npm run server

```
app.get("/users", async (req, res) => {
    try {
        const users = await User.find().lean()
                                .exec();
        return res.status(200).send({users: users});
    } catch(err) {
        .status(500)
        .send({message: "Something went wrong
                --- try agin later"});
    }
});

app.post("/users", async (req, res) => {
    try {
        const user = await User.create(req.
                                body);
        return res.status(201).send(user);
    }
    catch(err) {
        return res.status(500).send({
            message: err.message});
    }
});

// body => req.body
// url => req.params
// query string => req.query
// .lean() convert mongoose object into JSON object
// .exec()
```

If you want to find something that send into the body

Page No.:
Date : / /

```
app.get("/users/:id", async (req, res) => {
    try {
    Const user = await User.findByid(
          req.params.id).lean().exec();
// db.users.findone({_id: object('622173471b006569170')})
    return res.status(200).send(user);
    } catch(err) {
    return res.status(500).send({message: err.
                    message});
    } });

// If getting the single item then use it Id or any nam
app.patch("/users/:id", async (req, res) => {
    try {
    Const user = await User.findByIdAndupdate(
          req.params.id, req.body, {
          new: true,        // If you want to updated
    })                        value after paching the
                              value
    .lean()
    .exec();
    // db.users.update({_id: object('6228131 76')}, {set:
                  { req.body }})
    return res.status(200).send(user);
    } catch (err)
       {
      return res.status(500).send({message:
              err.message});
      }
    });
```

Manoj Kumar

// post CRUD

```
app.get("/posts", async (req, res) => {
    try {
        const posts = await post.find()
            .lean().exec();
        return res.status(200).send(posts);
    }
    catch (err) {
        return res.status(500).send(
        { message: err.message } )
    });

app.post("/posts", async (req, res) => {
    try {
        const post = await post.
            create(req.body);
        return res.status(200).send(
    } catch (err)   post);
        {
        return res.status(500).send
        ( { message: err.message } )
    })
```

// Single document get

```
app.get("/post/:id", async (req, res) => {
    try {
        const post = await post.findById(
            req.params.id).lean().exec();
        return res.send(post);
    } catch() {
```

Express MVC (Live lecture)

// COMMENT CRUD

```
app.get("/comments", async (req, res) => {
    try {
        Const Comment = await Comment.
            find().lean().exec();
        return res.status(200).Send(comment
    } Catch(err)
        {
        return res.status(500).Send({ message
            err.message });
    }
})
```

// Create Comment

```
app.post("/Comments"), async (req, res) => {
    try {
    Const Comment = await comment.Crea
    - te(req.body);

    return res.Status(201).Send(comment)
    } Catch(err) {
    return res.Status(500).send({ message:
        err.message });
});
```

Manoj kumar

Page No.:
Date : / /

7 Single Comment Patch

```
app.get("/comments/:id", async (req, res) => {
    try {
    Const comment = await Comment.findById(
        ~~req.body~~, req.params.id).lean().exec();
    return res.status(201).send(comment);
    } catch (err) {
    return res.status(500).send({
        message: err.message });
    }
});
```

//Update Comment

```
app.patch("/comments/:id", async (req, res) => {
    try {
    Const comment = await Comment.findByIdAndUpdate
        (req.params.id, req.body, { new:
        true }).lean().exec();
    return res.status(200).send(comment);
    } catch (err) {
    return res.status(500).send({ message:
        err.message });
    }
});
```

MONGO

Page No.:
Date : / /

// Delete Comment

```
app.delete("/comments/:id", async (req, res) =>
    {
    try {
    Const Comment = await Comment.findByIdAnd
        Delete(req.params.id).lean().exec();
    return res.status(200).send(comment);
    } catch(err) {
        return res.status(500).send({ message:
            err.message });
    }
});

app.listen(5000, async () => {
        try {
        await connect();
        } catch(err) {
        console.log(err);
        }
});
```

Mandi Kumar

Page No.:
Date : / /

MVC

populate

```
app.get("/posts", async (req, res) => {
    try {
  Const posts = await  post.find () —
            //.Populate ("userId")
    .populate ({
        path: "userId",
        select : { firstName: 1, email : 1, id: 0 },
    }) .lean() .exec();

    return res.send(200). send (posts);
    } catch(err) {
    return res. status (500). send ({
        message: err. message });
    } });
```