```
Mongoose.connect("mongodb://localhost/Exam", {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
Const db = mongoose.connection;

Db.on("error", console.error.bind(console, "MongoDB connection error:"));
Db.once("open", () => {
  Console.log("Connected to MongoDB");
});

Const examSchema = new mongoose.Schema({
  Name: String,
  Email: String,
  City: String,
  examDate: Date,
});

Const Exam = mongoose.model("Exam", examSchema);

App.use(bodyParser.json());

App.post("/api/addExamData", (req, res) => {
  Const { name, email, city, examDate } = req.body;

  Const newExam = new Exam({ name, email, city, examDate });

  newExam.save((err) => {
    if (err) {
      console.error("Error inserting exam data:", err);
      res.status(500).json({ error: "Error inserting exam data" });
    } else {
      Console.log("Exam data inserted successfully");
      Res.json({ message: "Exam data inserted successfully" });
    }
  });
});

App.listen(port, () => {
  Console.log(`Server is running on port ${port}`);
});
```

This example sets up an HTML form with validation rules, a frontend JavaScript file (`form.js`) to handle form submission and send data to the server, and a Node.js backend (`server.js`) to receive the data and insert it into a MongoDB collection named "Exam." Make sure to adjust the MongoDB connection URL and schema as needed for your environment.

32----

To perform the mentioned tasks using Node.js and Mongoose with the given employee collection, you can follow these steps:

1. Set up Mongoose and connect to your MongoDB database.

```javascript
```

```
Const mongoose = require(‘mongoose’);
Mongoose.connect(‘mongodb://localhost/main’, { useNewUrlParser: true, useUnifiedTopology: true });

Const db = mongoose.connection;
Db.on(‘error’, console.error.bind(console, ‘MongoDB connection error:’));
Db.once(‘open’, () => {
  Console.log(‘Connected to MongoDB’);
});

// Define the Employee schema
Const employeeSchema = new mongoose.Schema({
  Name: String,
  Age: Number,
  Position: String,
  Salary: Number,
});

Const Employee = mongoose.model(‘Employee’, employeeSchema);
```

2. Insert the provided data into the “main” collection:

```javascript
Const initialData = [
  // … The provided employee data …
];

Employee.insertMany(initialData, (err) => {
  If (err) {
    Console.error('Error inserting data:', err);
  } else {
    Console.log('Data inserted successfully');
  }
});
```

3. Perform the requested queries:

```javascript
// (1) Update or insert a document with age 43 and position "Senior Manager"
Employee.updateOne(
  { age: 43, position: 'Senior Manager' },
  { $set: { experience: 17 } },
  { upsert: true },
  (err, result) => {
    If (err) {
      Console.error('Error updating/inserting document:', err);
    } else {
      Console.log('Document updated/inserted successfully');
    }
  }
);

// (2) Find the employee with the highest salary
Employee.findOne().sort({ salary: -1 }).exec((err, employee) => {
  If (err) {
    Console.error('Error finding employee:', err);
  } else {
    Console.log(`Employee with highest salary: ${employee.name}, Position: ${employee.position}`);
  }
});

// (3) Count documents where name contains “ric”
Employee.countDocuments({ name: /ric/ }, (err, count) => {
  If (err) {
    Console.error(‘Error counting documents:’, err);
  } else {
    Console.log(`Number of documents with “ric” in name: ${count}`);
  }
});

// (4) Increase the salary of employees with salary less than 45000 by 10%
Employee.updateMany({ salary: { $lt: 45000 } }, { $mul: { salary: 1.1 } }, (err, result) => {
  If (err) {
    Console.error(‘Error updating salaries:’, err);
  } else {
    Console.log(`Salaries updated for ${result.nModified} employees`);
  }
});

// (5) Find positions where name has 4 or 5 letters
Employee.distinct(‘position’, { name: /^[a-zA-Z]{4,5}$/ }, (err, positions) => {
  If (err) {
    Console.error(‘Error finding positions:’, err);
  } else {
    Console.log(`Positions with names containing 4 or 5 letters: ${positions}`);
  }
```

~~~
});
```
~~~

Make sure to adjust the MongoDB connection URL and schema as needed for your environment. This code sets up the Mongoose schema, inserts initial data, and performs the requested queries on the “main” collection.