Mongodb Project

prepared by:

Feras Abdusalam Gsiea 191207

Recruiting agency system

Introduction to Recruiting Agency System:

A recruiting agency serves as a vital bridge between job seekers and employers, streamlining the hiring process for both parties. The primary objective of such an agency is to match qualified candidates with job openings that align with their skills, experience, and career aspirations, while simultaneously helping employers find the right talent to meet their business needs.

Scenario:

Scenario: Streamlining Recruitment with the Recruiting Agency

TechCorp, a fast-growing software company, collaborates with a company like **CareerConnect**, a recruiting agency, to find a skilled **Backend Developer** for their team. Using the **Recruiting Agency Management System**, the process is managed efficiently from job posting to hiring, with **recruiters** handling candidate interviews on behalf of the employer.

- **Job Posting:**
  TechCorp submits a job listing to CareerConnect, specifying the position, required skills (Node.js, MongoDB, and Express.js), salary range, and location. The system stores the details and makes the job available to job seekers.

- **Candidate Application:**
  Jane Doe, a registered job seeker with expertise in backend development, finds the job posting and applies through the system. Her application is recorded in the database.

- **Application Review by Recruiter:**
  A CareerConnect recruiter, John Smith, reviews Jane's application. After confirming her skills match the requirements, John shortlists her for the next stage and updates the application status to "shortlisted."

- **Recruiter-Conducted Interview:**
  John schedules a virtual interview with Jane to assess her technical and interpersonal skills. The interview details—date, time, and mode—are recorded in the system, and Jane receives a notification. After the interview, John provides feedback:

    - **Rating:** 4.5/5
    - **Comments:** "Excellent technical expertise and great communication skills."

- The feedback is stored in the system for TechCorp's review.

- **Employer Decision:**
  Based on the recruiter's feedback, TechCorp decides to hire Jane Doe. The recruiter updates the application status to "hired," and the job listing is marked as filled.

db collections :

1. `job_postings` Collection

Purpose:

Stores job listings submitted by employers (e.g., TechCorp). These job postings are made available to candidates for applications.

Document Structure:

- `_id`: Unique identifier for the job posting (ObjectId).
- `employer`: Name of the employer (String).
- `position`: Job title (String).
- `required_skills`: List of skills required for the job (Array of Strings).
- `salary_range`: Salary range for the job (Object with `min` and `max` fields).
- `location`: Job location (String).
- `status`: Current status of the job posting (`open` or `filled`) (String).
- `posted_date`: Date the job was posted (ISODate).

2. `candidates` Collection

Purpose:
Stores information about job seekers who apply for jobs through the system.

Document Structure:

- `_id`: Unique identifier for the candidate (ObjectId).
- `name`: Full name of the candidate (String).
- `email`: Email address of the candidate (String).
- `skills`: List of skills the candidate possesses (Array of Strings).
- `resume`: Link to the candidate's resume (String).
- `registration_date`: Date the candidate registered on the platform (ISODate).

3. `recruiters` Collection

Purpose:
Stores information about recruiters who manage the hiring process for job postings.

Document Structure:

- `_id`: Unique identifier for the recruiter (ObjectId).
- `name`: Full name of the recruiter (String).
- `email`: Email address of the recruiter (String).
- `assigned_jobs`: List of job postings the recruiter is handling (Array of ObjectIds referencing `job_postings`).

4. `applications` Collection

Purpose:
Tracks applications submitted by candidates for specific job postings.

Document Structure:

- `_id`: Unique identifier for the application (ObjectId).
- `job_id`: Reference to the job posting (ObjectId referencing `job_postings`).
- `candidate_id`: Reference to the candidate (ObjectId referencing `candidates`).
- `application_date`: Date the application was submitted (ISODate).
- `status`: Current status of the application (`applied`, `shortlisted`, `rejected`, or `hired`) (String).
- `recruiter_id`: Reference to the recruiter handling the application (ObjectId referencing `recruiters`).

5. `interviews` Collection

Purpose:
Stores details about interviews conducted by recruiters for shortlisted candidates.

Document Structure:

- `_id`: Unique identifier for the interview (ObjectId).
- `application_id`: Reference to the application (ObjectId referencing `applications`).
- `interview_date`: Date and time of the interview (ISODate).
- `mode`: Mode of the interview (`virtual` or `in-person`) (String).
- `recruiter_id`: Reference to the recruiter conducting the interview (ObjectId referencing `recruiters`).
- `feedback`: Feedback from the recruiter (Object with `rating` and `comments` fields).

Queries

1. Increment a Field with `$inc`

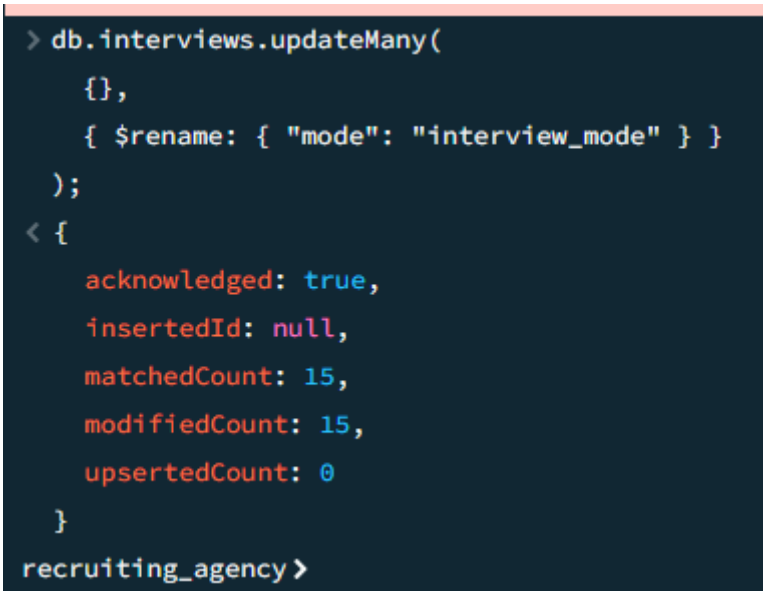
increment a field by a specified value increase the `min` salary of a job posting by $5000:

```
db.job_postings.updateOne(
  { position: "Backend Developer" },
  { $inc: { "salary_range.min": 5000 } }
);
```

2. Rename a Field with `$rename`

Rename the mode field to interview_mode in the interviews collection.

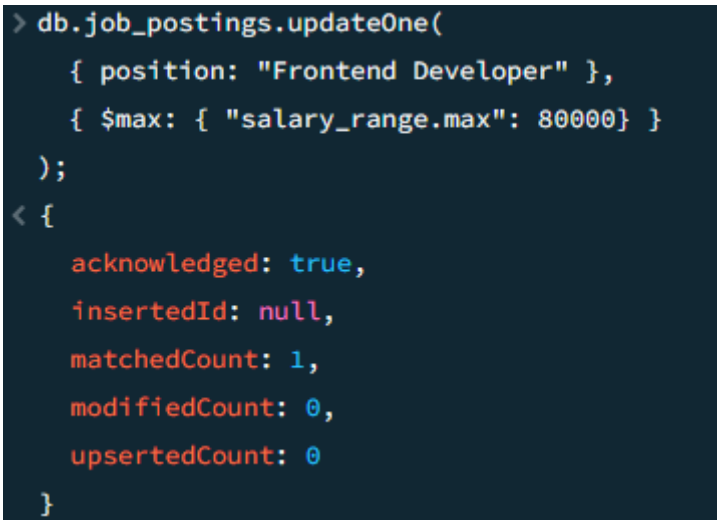
```
db.interviews.updateMany(
  {},
  { $rename: { "mode": "interview_mode" } }
);
```



3. Update a Field to the Maximum Value with `$max`

Update the max salary of the "Frontend Developer" job to $80,000 if it's currently lower.

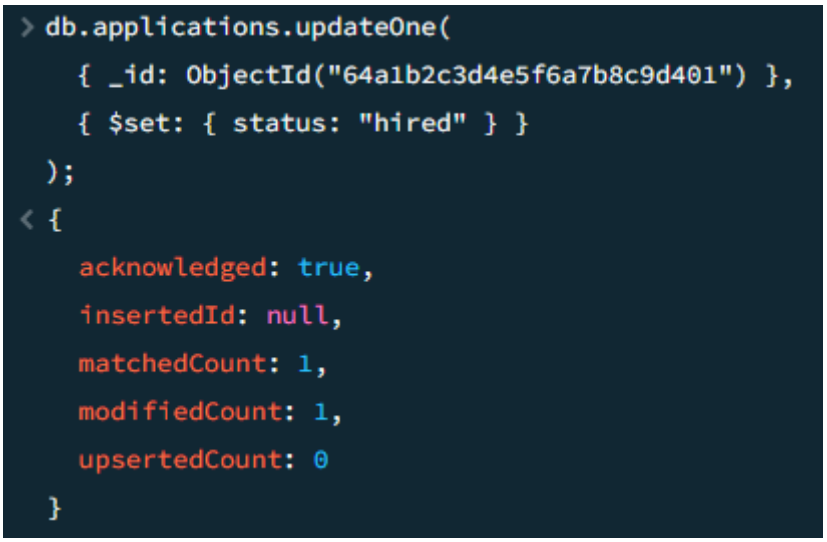
```
db.job_postings.updateOne(
  { position: "Frontend Developer" },
  { $max: { "salary_range.max": 80000} }
);
```



4. Set a Field with $set

Update the status of Alice Johnson's application to "hired".

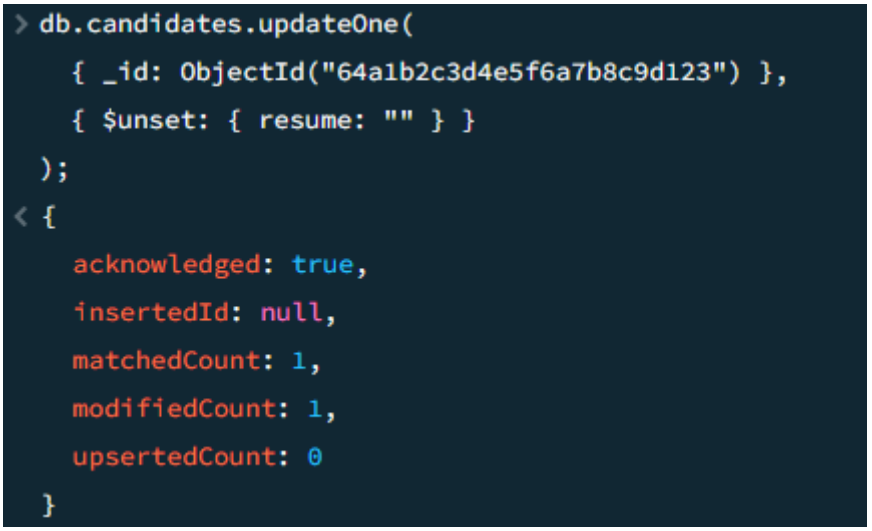
```
db.applications.updateOne(
  { _id: ObjectId("64a1b2c3d4e5f6a7b8c9d401") },
  { $set: { status: "hired" } });
```



5. Unset a Field with $unset

remove the resume field from a candidate's document.

```
db.candidates.updateOne(
  { _id: ObjectId("
64a1b2c3d4e5f6a7b8c9d123") },
  { $unset: { resume: "" } }
);
```



6. Add an Element to an Array with $push

add a new skill to a candidate’s skills array.

```
db.candidates.updateOne(
  { “name”: “Bob Smith”},
  { $push: { skills: "TypeScript" } }
);
```

```
> db.candidates.updateOne(
    { "name": "Bob Smith"},
    { $push: { skills: "TypeScript" } }
  );
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
>
```

7. Remove an Element from an Array with $pull

remove "CSS" from a candidate’s skills array.

```
db.candidates.updateOne(
  { "name":"Bob Smith" },
  { $pull: { skills: "CSS" } }
);
```

```
> db.candidates.updateOne(
    { "name":"Bob Smith" },
    { $pull: { skills: "CSS" } }
  );
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

8.Update Multiple Fields with `$set`

update both the `status` and `application_date` of an application.

db.applications.updateOne(
{ _id: ObjectId("64a1b2c3d4e5f6a7b8c9d402") },
{ $set: { status: "rejected", application_date: ISODate("2023-10-25T00:00:00Z") } }
);

```
> db.applications.updateOne(
    { _id: ObjectId("64a1b2c3d4e5f6a7b8c9d402") },
    { $set: { status: "rejected", application_date: ISODate("2023-10-25T00:00:00Z") } }
  );
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

9. Update Nested Fields

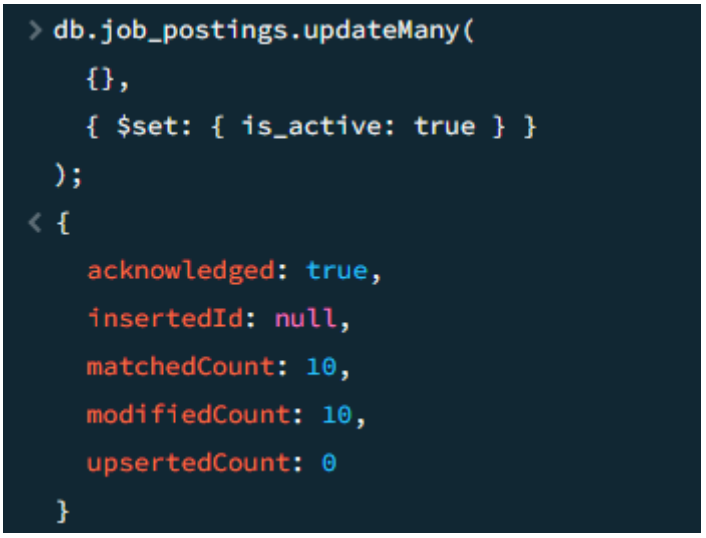
edit feedback comment for Alice johnson
db.interviews.updateOne(
{ _id: ObjectId("64a1b2c3d4e5f6a7b8c9d501") }, // Alice Johnson
{ $set: { "feedback.comments": "Outstanding performance in the technical interview." } }
);

```
> db.interviews.updateOne(
    { _id: ObjectId("64a1b2c3d4e5f6a7b8c9d501") }, // Alice Johnson
    { $set: { "feedback.comments": "Outstanding performance in the technical interview." } }
  );
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

10.Update All Documents with `$set`

add a new field is_active to all job postings and set it to true

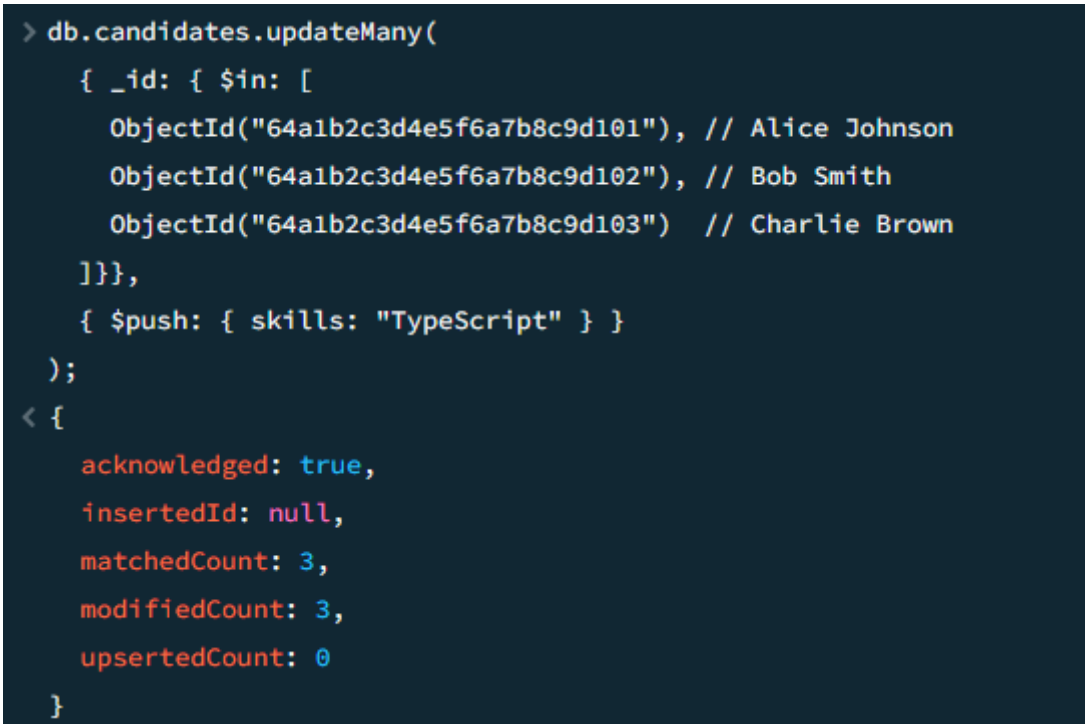
```
db.job_postings.updateMany(
  {},
  { $set: { is_active: true } }
);
```



11.Update Multiple Candidates with $push

use the $push operator to add a new skill like "TypeScript" to the skills array for multiple candidates.

```
db.candidates.updateMany(
  { _id: { $in: [
    ObjectId("64a1b2c3d4e5f6a7b8c9d101"), // Alice Johnson
    ObjectId("64a1b2c3d4e5f6a7b8c9d102"), // Bob Smith
    ObjectId("64a1b2c3d4e5f6a7b8c9d103")  // Charlie Brown
  ]}},
  { $push: { skills: "TypeScript" } }
);
```



12.Add a Unique Skill to Multiple Candidates ($addToSet)

add a unique skill if the skill already exists it wont be added again

```
db.candidates.updateMany(
  { _id: { $in: [
    ObjectId("64a1b2c3d4e5f6a7b8c9d101"), // Alice Johnson
    ObjectId("64a1b2c3d4e5f6a7b8c9d102"), // Bob Smith
    ObjectId("64a1b2c3d4e5f6a7b8c9d103")  // Charlie Brown
  ]}},
  { $addToSet: { skills: "Docker" } }
);
```

```
> db.candidates.updateMany(
    { _id: { $in: [
      ObjectId("64a1b2c3d4e5f6a7b8c9d101"), // Alice Johnson
      ObjectId("64a1b2c3d4e5f6a7b8c9d102"), // Bob Smith
      ObjectId("64a1b2c3d4e5f6a7b8c9d103")  // Charlie Brown
    ]}},
    { $addToSet: { skills: "Docker" } }
  );
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 3,
    modifiedCount: 3,
    upsertedCount: 0
  }
```

13. Multiply Salary by 10% Using `$mul`

Increase the max salary of a job posting by 10%

```
db.job_postings.updateOne(
  { position: "Backend Developer" },
  { $mul: { "salary_range.max": 1.1 } }
);
```

```
db.job_postings.updateOne(
  { position: "Backend Developer" },
  { $mul: { "salary_range.max": 1.1 } } // Multiply by 1.1 (10% increase)
);
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

14. Set a Minimum Salary Using `$min`

Update the min salary of a job posting to $50,000 **only if the current value is lower**.

```
db.job_postings.updateOne(
  { position: "Frontend Developer" },
  { $min: { "salary_range.min": 50000 } }
);
```

15. Add a Timestamp Using `$currentDate`

Add a last_updated field to track when a job posting was last modified.

```
db.job_postings.updateOne(
  { position: "Data Engineer" },
  {
    $set: { status: "closed" },
    $currentDate: { last_updated: true }
  }
);
```

```
> db.job_postings.updateOne(
    { position: "Data Engineer" },
    {
      $set: { status: "closed" },
      $currentDate: { last_updated: true }
    }
  );
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

16. Update Nested Array Elements with `arrayFilters`

Correct a typo in a candidate's skills array

```
db.candidates.updateOne(
  { "skills": "Javascript" },
  { $set: { "skills.$[element]": "JavaScript" } },
  { arrayFilters: [ { "element": "Javascript" } ] }
);
```

```
> db.candidates.updateOne(
    { "skills": "Javascript" },
    { $set: { "skills.$[element]": "JavaScript" } },
    { arrayFilters: [ { "element": "Javascript" } ] }
  );
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 0,
    modifiedCount: 0,
    upsertedCount: 0
  }
```

Aggregation operations

17. Single-Purpose Aggregation: Average Interview Rating per Job

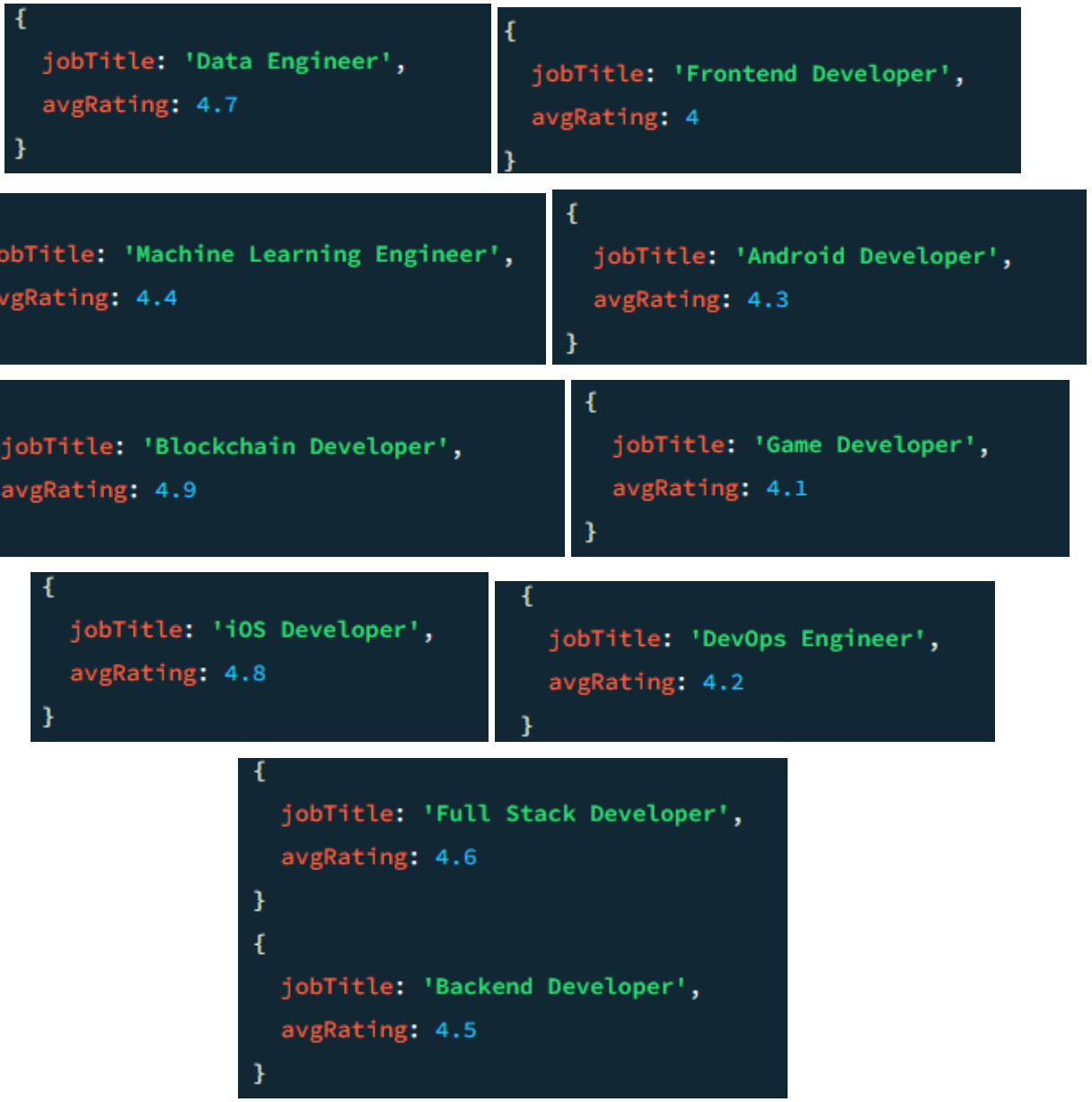
Calculate the average interview rating for each job posting.

```
db.interviews.aggregate([
 {
  $lookup: {
   from: "applications",
   localField: "application_id",
   foreignField: "_id",
   as: "application"
  }
 },
 { $unwind: "$application" },
 {
  $lookup: {
   from: "job_postings",
   localField: "application.job_id",
   foreignField: "_id",
   as: "job"
  }
 },
 { $unwind: "$job" },
 {
  $group: {
   _id: "$job.position",
   avgRating: { $avg: "$feedback.rating" }
  }
 },
 {
  $project: {
   _id: 0,
   jobTitle: "$_id",
   avgRating: { $round: ["$avgRating", 2] }
  }
 }
]);
```

```
> db.interviews.aggregate([
    {
      $lookup: {
        from: "applications",
        localField: "application_id",
        foreignField: "_id",
        as: "application"
      }
    },
    { $unwind: "$application" },
    {
      $lookup: {
        from: "job_postings",
        localField: "application.job_id",
        foreignField: "_id",
        as: "job"
      }
    },
    { $unwind: "$job" },
    {
      $group: {
        _id: "$job.position",
        avgRating: { $avg: "$feedback.rating" }
      }
```

```
  },
  {
    $project: {
      _id: 0,
      jobTitle: "$_id",
      avgRating: { $round: ["$avgRating", 2] }
    }
  }
]);
{
```

Result:


{
jobTitle: 'Data Engineer',
avgRating: 4.7
}
{
jobTitle: 'Frontend Developer',
avgRating: 4
}
{
jobTitle: 'Machine Learning Engineer',
avgRating: 4.4
}
{
jobTitle: 'Android Developer',
avgRating: 4.3
}
{
jobTitle: 'Blockchain Developer',
avgRating: 4.9
}
{
jobTitle: 'Game Developer',
avgRating: 4.1
}
{
jobTitle: 'iOS Developer',
avgRating: 4.8
}
{
jobTitle: 'DevOps Engineer',
avgRating: 4.2
}
{
jobTitle: 'Full Stack Developer',
avgRating: 4.6
}
{
jobTitle: 'Backend Developer',
avgRating: 4.5
}

18. Pipelining Aggregation: Candidates with Matching Skills for a Job

Calculate the average interview rating for each job posting.

there are no required skills because there are no real jobs so there will be no output because no matching will happen

```
db.job_postings.aggregate([
 {
  $match: { position: "Backend Developer" }
 },
 {
  $lookup: {
   from: "candidates",
   let: { requiredSkills: "$required_skills" },
   pipeline: [
    {
     $match: {
      $expr: {
       $gte: [
        { $size: { $setIntersection: ["$skills", "$$requiredSkills"] } },
        3
       ]
      }
     }
    }
   ],
   as: "qualifiedCandidates"
  }
 },
 {
  $project: {
   _id: 0,
   jobTitle: "$position",
   qualifiedCandidates: {
    $map: {
     input: "$qualifiedCandidates",
     as: "candidate",
     in: {
      name: "$$candidate.name",
      matchingSkills: {
       $setIntersection: ["$$candidate.skills", "$required_skills"]
      }
     }
```

```
      }
    }
  } }
]);
```

```
test> db.job_postings.aggregate([
        {
          $match: { position: "Backend Developer" }
        },
        {
          $lookup: {
            from: "candidates",
            let: { requiredSkills: "$required_skills" },
            pipeline: [
              {
                $match: {
                  $expr: {
                    $gte: [
                      { $size: { $setIntersection: ["$skills", "$$requiredSkills"] } },
                      3
                    ]
                  }
                }
              }
            ],
            as: "qualifiedCandidates"
          }
        },
        {
```

```
    {
      $project: {
        _id: 0,
        jobTitle: "$position",
        qualifiedCandidates: {
          $map: {
            input: "$qualifiedCandidates",
            as: "candidate",
            in: {
              name: "$$candidate.name",
              matchingSkills: {
                $setIntersection: ["$$candidate.skills", "$required_skills"]
              }
            }
          }
        }
      }
    }
  ]);
```

19. Map-Reduce: Recruiter Performance (Applications Handled & Avg Rating)

Calculate the number of applications handled and average interview rating per recruiter.

```
var map = function() {
  var key = this._id; // Recruiter ID
  var value = {
    count: 1,
    totalRating: this.interviews.feedback.rating || 0
  };
  emit(key, value);
};


var reduce = function(key, values) {
  var result = { count: 0, totalRating: 0 };
  values.forEach(function(value) {
    result.count += value.count;
    result.totalRating += value.totalRating;
  });
  return result;
};
```

```
var finalize = function(key, reducedValue) {
  return {
    totalApplications: reducedValue.count,
    avgRating: reducedValue.totalRating / reducedValue.count
  };
};


db.recruiters.mapReduce(
  map,
  reduce,
  {
    out: "recruiter_performance",
    finalize: finalize,
    query: {},
    scope: { interviews: db.interviews.find().toArray() }
  }
);
```

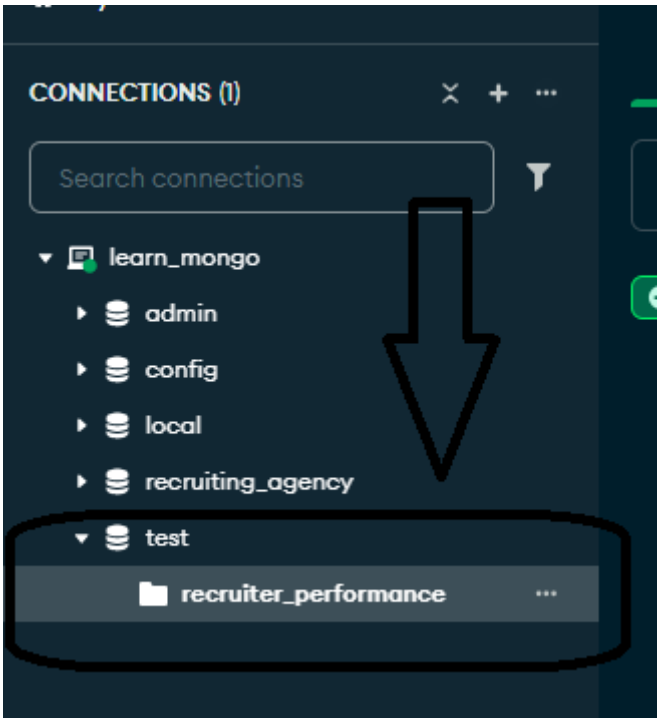
```
> var map = function() {
    var key = this._id; // Recruiter ID
    var value = {
      count: 1,
      totalRating: this.interviews.feedback.rating || 0
    };
    emit(key, value);
  };


  var reduce = function(key, values) {
    var result = { count: 0, totalRating: 0 };
    values.forEach(function(value) {
      result.count += value.count;
      result.totalRating += value.totalRating;
    });
    return result;
  };
```

```
var finalize = function(key, reducedValue) {
  return {
    totalApplications: reducedValue.count,
    avgRating: reducedValue.totalRating / reducedValue.count
  };
};


db.recruiters.mapReduce(
  map,
  reduce,
  {
    out: "recruiter_performance",
    finalize: finalize,
    query: {},
    scope: { interviews: db.interviews.find().toArray() }
  }
);
```

```
{ result: 'recruiter_performance', ok: 1 }
```

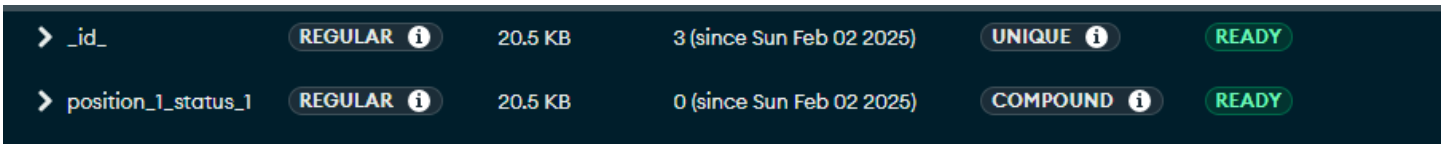


Indexes on the database

1.job posting collection:

compound index on **position** and **status :**

db.job_postings.createIndex({ position: 1, status: 1 });

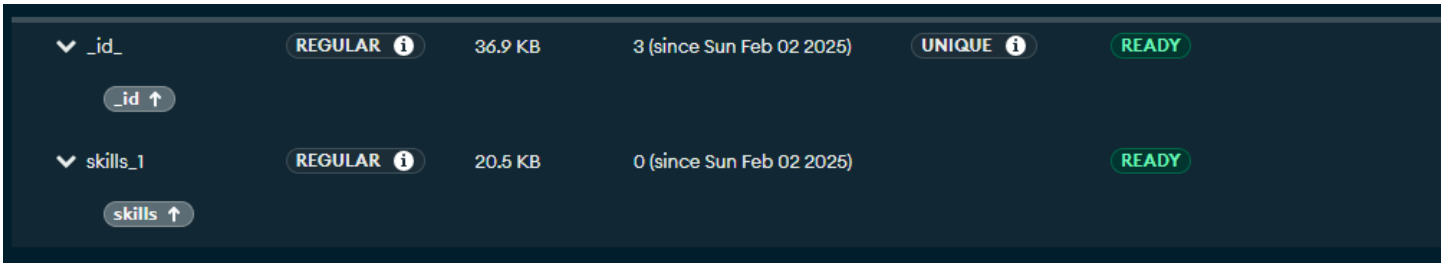


purpose:
Quickly find open jobs by position

2.candidates collection:

multikey index on **skills:**

db.candidates.createIndex({ skills: 1 });

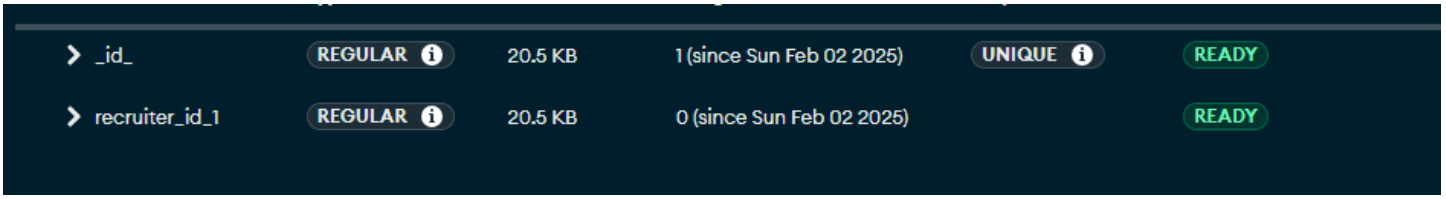


purpose:
Find candidates with specific skills

3.interviews collection:

index on **recruiter_id:**

db.interviews.createIndex({ recruiter_id: 1 });



purpose:
Find all interviews conducted by a recruiter.