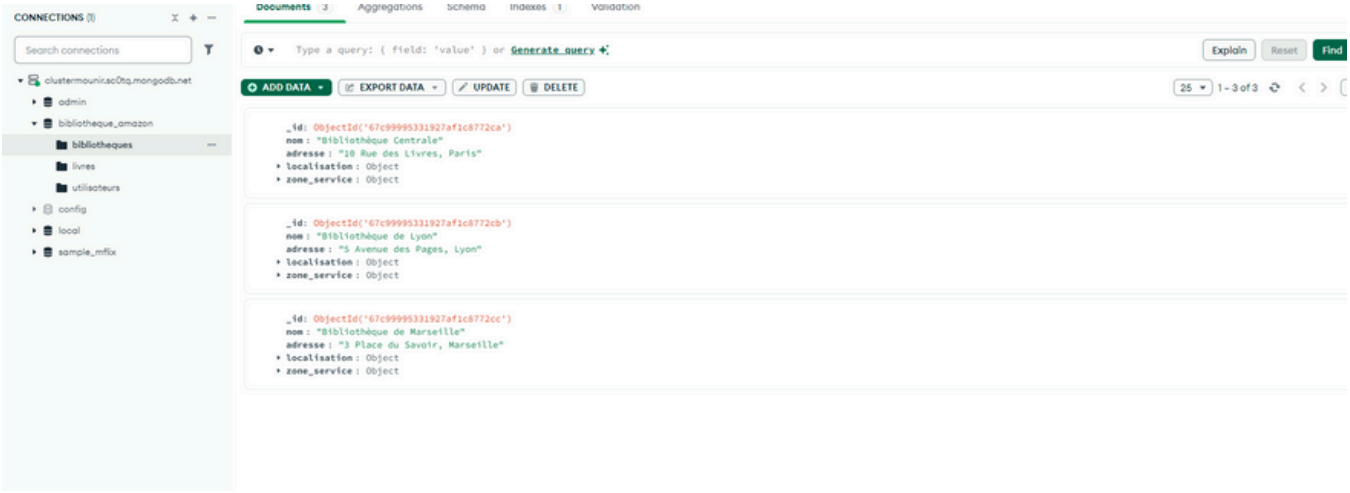
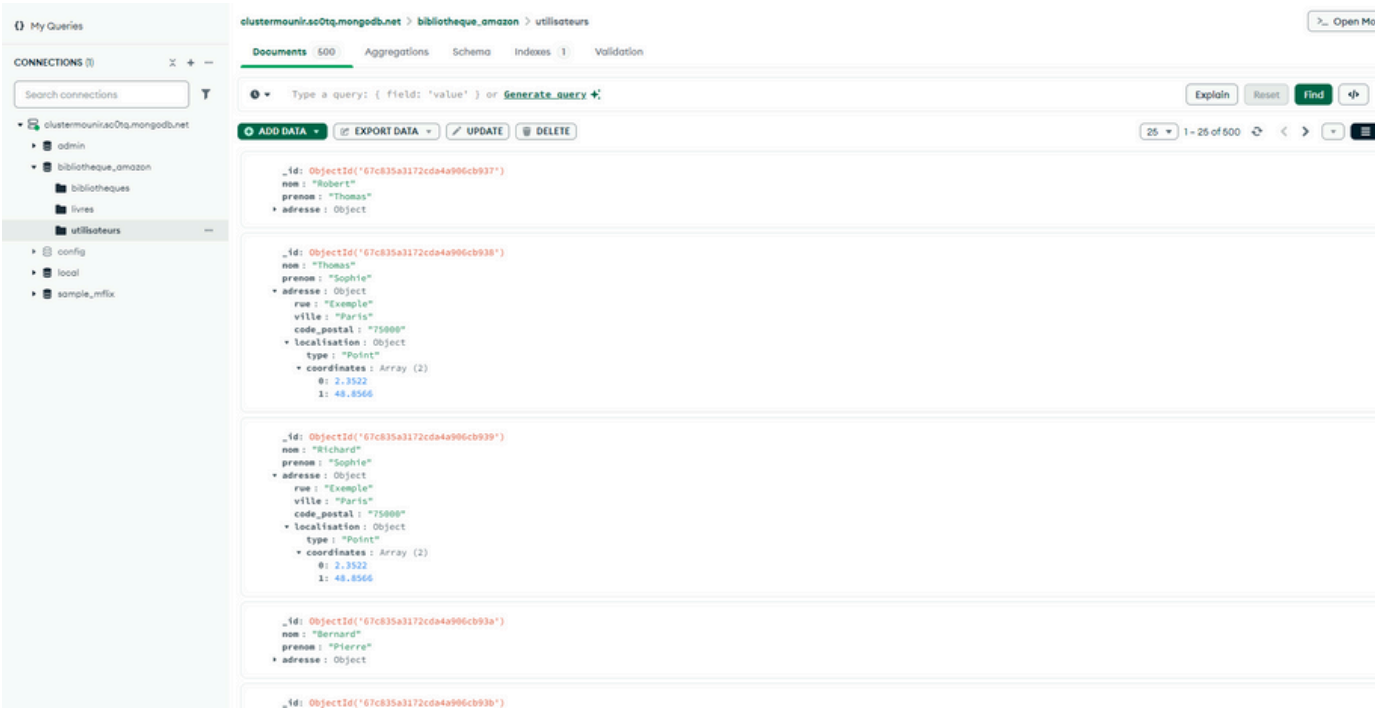
PRÉSENTATION BDD





AJOUT DES COORDONNÉES

J’ai ajouté des coordonnées géographiques aux utilisateurs en structurant l’adresse avec un format GeoJSON.

```
db.utilisateurs.updateMany({}, {
  $set: {
    adresse: {
      rue: "Exemple",
      ville: "Paris",
      code_postal: "75000",
      localisation: {
        type: "Point",
        coordinates: [2.3522, 48.8566] // Longitude, Latitude
      }
    }
  }
});
```

J’ai ensuite créé une nouvelle collection “bibliotheques” avec 3 bibliothèques, en incluant leur nom et adresse, leur localisation en tant que point et une zone de service définie par un Polygon.

```
db.bibliotheques.insertMany([
  {
    nom: "Bibliothèque Centrale",
    adresse: "10 Rue des Livres, Paris",
    localisation: { type: "Point", coordinates: [2.35, 48.85] },
        zone_service: { type: "Polygon", coordinates: [[[2.34,48.84], [2.36,48.84], [2.36,48.86], [2.34,48.86], [2.34,48.84]]] }
  }
]);
```

CRÉATION INDEX GÉOSPATIAUX

J’ai ajouté des index géospatiaux pour optimiser les requêtes basées sur la distance car sans cet index, MongoDB ne pourrait pas effectuer de recherche par distance et renverrait une erreur.
Cet index permet d’exploiter les fonctions $near, $geoWithin et $geoIntersects.

```
db.utilisateurs.createIndex({ "adresse.localisation": "2dsphere" });
db.bibliotheques.createIndex({ "localisation": "2dsphere" });
```

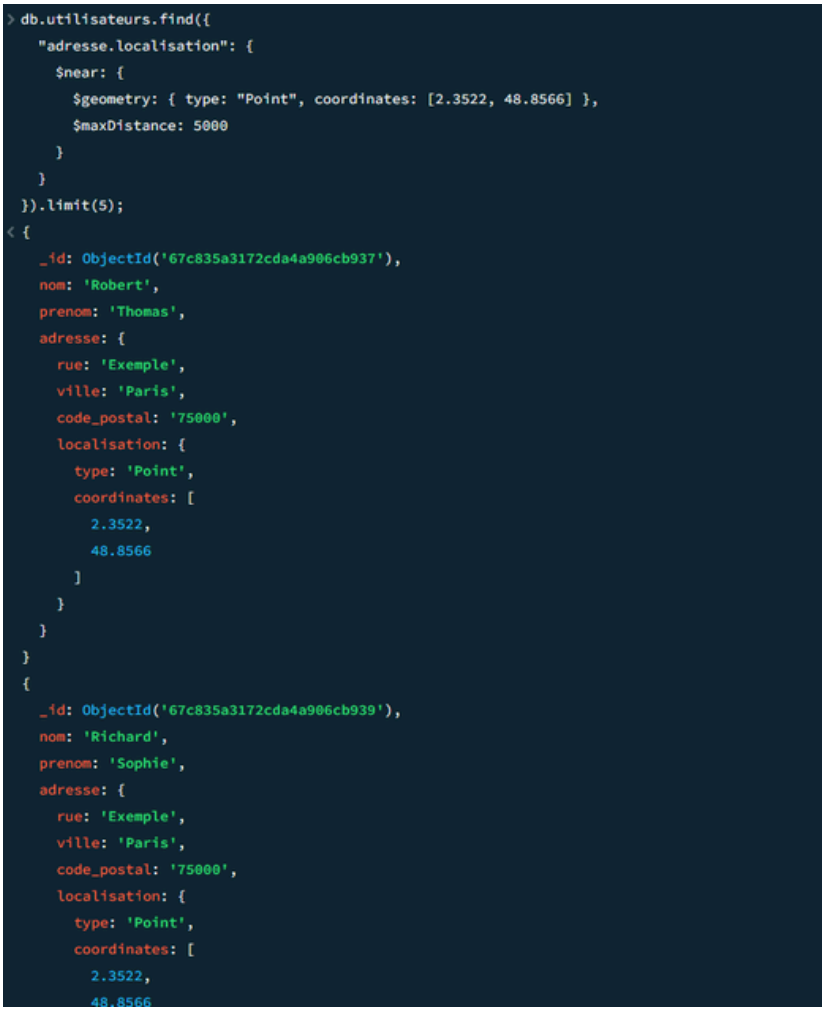
LES REQUÊTES GÉOSPATIALES

1. **Trouver les 5 utilisateurs les plus proches d'un point donné**

J'ai recherché les 5 utilisateurs les plus proches du centre de Paris dans un rayon de 5 km.

```
db.utilisateurs.find({
  "adresse.localisation": {
    $near: {
      $geometry: { type: "Point", coordinates: [2.3522, 48.8566] },
      $maxDistance: 5000
    }
  }
}).limit(5);
```

Grâce à $near, MongoDB trie automatiquement les résultats par distance croissante. Voici un petit aperçu du résultat :



```
> db.utilisateurs.find({
    "adresse.localisation": {
      $near: {
        $geometry: { type: "Point", coordinates: [2.3522, 48.8566] },
        $maxDistance: 5000
      }
    }
  }).limit(5);
< {
    _id: ObjectId('67c835a3172cda4a906cb937'),
    nom: 'Robert',
    prenom: 'Thomas',
    adresse: {
      rue: 'Exemple',
      ville: 'Paris',
      code_postal: '75000',
      localisation: {
        type: 'Point',
        coordinates: [
          2.3522,
          48.8566
        ]
      }
    }
  }
  {
    _id: ObjectId('67c835a3172cda4a906cb939'),
    nom: 'Richard',
    prenom: 'Sophie',
    adresse: {
      rue: 'Exemple',
      ville: 'Paris',
      code_postal: '75000',
      localisation: {
        type: 'Point',
        coordinates: [
          2.3522,
          48.8566
```

LES REQUÊTES GÉOSPATIALES

2.Trouver la bibliothèque la plus proche d'un utilisateur

J'ai recherché la bibliothèque la plus proche d'un utilisateur se trouvant à Lyon.

```
db.bibliotheques.find({
  "localisation": {
    $near: {
      $geometry: { type: "Point", coordinates: [4.8357, 45.7640] }
    }
  }
}).limit(1);
```

Cette requête permet d'orienter un utilisateur vers la bibliothèque la plus accessible.
Voici un petit aperçu du résultat :

```
>_MONGOSH
> db.bibliotheques.find({
    "localisation": {
      $near: {
        $geometry: { type: "Point", coordinates: [4.8357, 45.7640] }
      }
    }
  }).limit(1);
< {
    _id: ObjectId('67c99995331927af1c8772cb'),
    nom: 'Bibliothèque de Lyon',
    adresse: '5 Avenue des Pages, Lyon',
    localisation: {
      type: 'Point',
      coordinates: [
        4.8357,
        45.764
      ]
    },
    zone_service: {
      type: 'Polygon',
      coordinates: [
        [
          [
            4.82,
            45.75
          ],
          [
            4.85,
            45.75
          ],
          [
            4.85,
            45.77
          ],
          [
            4.82,
            45.77
          ],
```

LES REQUÊTES GÉOSPATIALES

3.Trier les bibliothèques par distance

J’ai utilisé $geoNear pour trier toutes les bibliothèques en fonction de leur distance par rapport à Paris.

```
db.bibliotheques.aggregate([
  {
    $geoNear: {
      near: { type: "Point", coordinates: [2.3522, 48.8566] },
      distanceField: "distance_km",
      spherical: true
    }
  }
]);
```

L’opérateur $geoNear est le plus précis pour mesurer les distances et renvoie la distance exacte. Voici un petit aperçu du résultat :

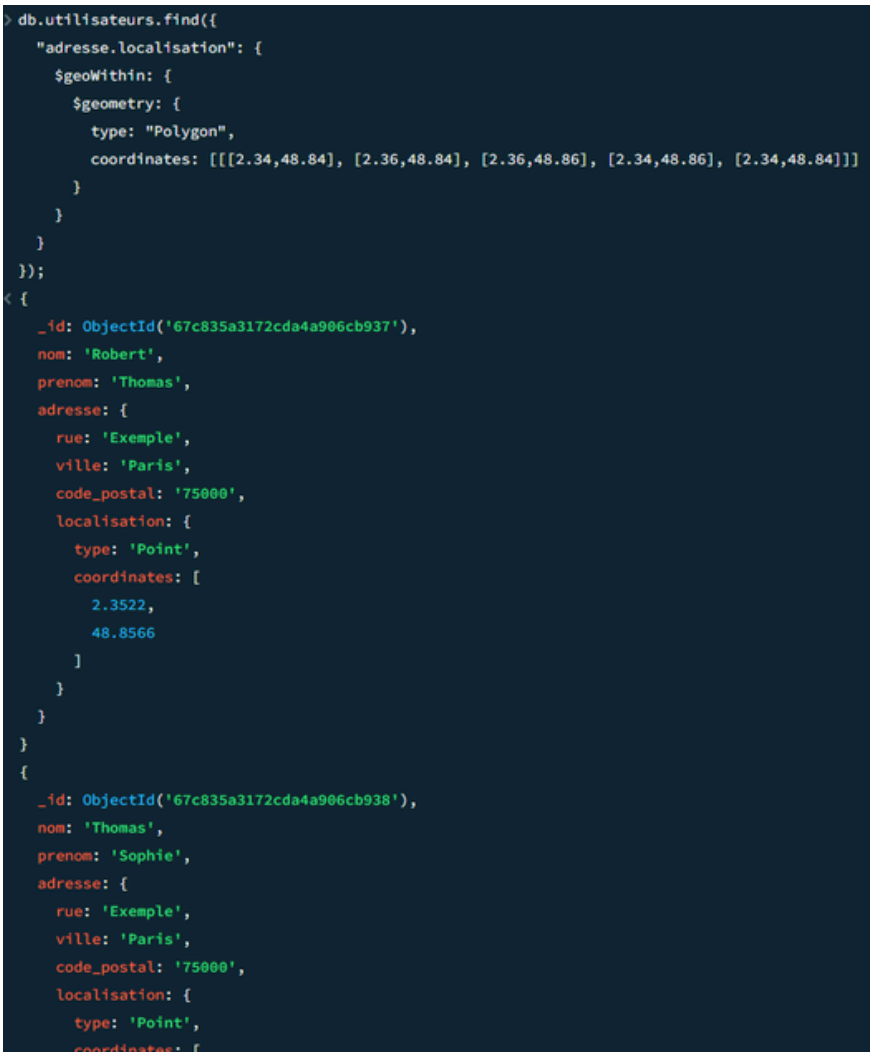
LES REQUÊTES GÉOSPATIALES

4.Trouver les utilisateurs dans une zone

J’ai recherché tous les utilisateurs situés dans une zone précise.

```
db.utilisateurs.find({
  "adresse.localisation": {
    $geoWithin: {
      $geometry: {
        type: "Polygon",
        coordinates: [[[2.34,48.84], [2.36,48.84], [2.36,48.86], [2.34,48.86], [2.34,48.84]]]
      }
    }
  }
});
```

L’opérateur $geoWithin permet de filtrer les utilisateurs qui habitent dans une zone de service. Voici un petit aperçu du résultat :



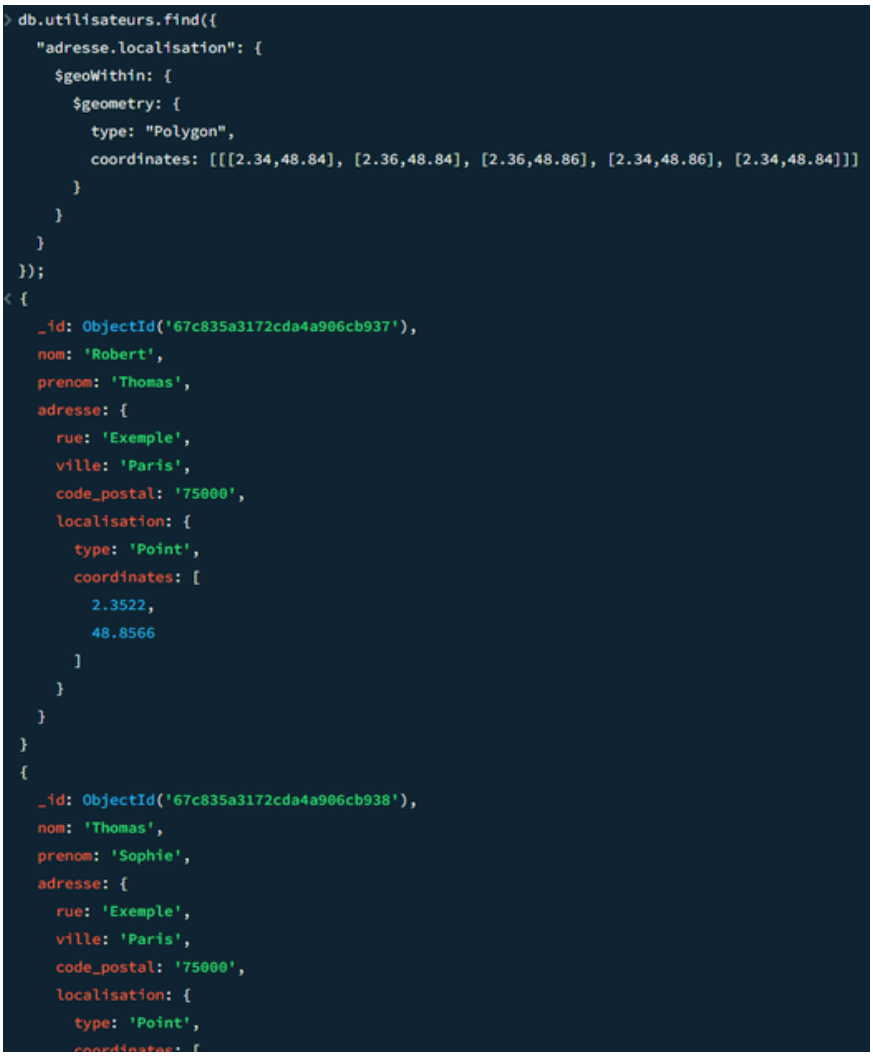
LES REQUÊTES GÉOSPATIALES

4.Trouver les utilisateurs dans une zone

J’ai recherché tous les utilisateurs situés dans une zone précise.

```
db.utilisateurs.find({
  "adresse.localisation": {
    $geoWithin: {
      $geometry: {
        type: "Polygon",
        coordinates: [[[2.34,48.84], [2.36,48.84], [2.36,48.86], [2.34,48.86], [2.34,48.84]]]
      }
    }
  }
});
```

L’opérateur $geoWithin permet de filtrer les utilisateurs qui habitent dans une zone de service. Voici un petit aperçu du résultat :



CONCLUSION

Grâce à ce TP, j'ai appris à manipuler les requêtes géospatiales dans MongoDB et à stocker et structurer des données géographiques avec GeoJSON, utiliser des index pour optimiser les requêtes et effectuer des recherches de proximité, de tri et de zone avec $near, $geoWithin, $geoNear.