AGENCE BANCAIRE

RAPPORT TP MONGODB PRESENTE PAR

ROMEO DAVID AMEDOMEY

MOHAMED ELYSALEM

TYMOTHEUS IGOR CYRYL CIESIELSKI

Eliel WOTOBE

Sous la supervision de M. GABRiel MOPOLO

2022/2023

* 1. Choix du Sujet

1. **Description du sujet** :

Ce sujet consiste à fournir une version informatisée d’une agence bancaire

2- **Dictionnaire de données** :

* ENTITE : Compte
* ATTRIBUTS :
* \_id : identifiant unique du compte (entier)
* balance : solde du compte (nombre décimal)
* created\_at : date de création du compte (date)
* closed\_at : date de clôture du compte (date)
* type : type du compte (énumération)

* ENTITE : Gestionnaire
* ATTRIBUTS :
* \_id : identifiant unique du Gestionnaire(entier)
* name : nom du Gestionnaire (chaîne de caractère)
* dateHired : date d’embauche du manager (date)

* ENTITE : Client
* ATTRIBUTS :
* \_id : identifiant unique du Client (entier)
* lastName : nom du Client (chaîne de caractères)
* firstName : prénom du Client (chaîne de caractères)
* phone : téléphone du Client (entier)
* address : adresse du Client (Objet)

* ENTITE : Agence
* ATTRIBUTS :
* \_id : identifiant du document(entier)
* name : nom de l’Agence (chaîne de caractère)
* address : adresse de l’Agence (Objet)

* ENTITE : Operation
* ATTRIBUTS :
* \_id : identifiant unique de l’Opération (entier)
* label : libellé de l’opération (chaîne de caractère)
* date : date de l’opération (date)
* debitAmount : montant débité lors de l’opération (nombre décimal)
* creditAmount : montant crédité lors de l’opération (nombre décimal)

**3- Description textuelle des associations** :

Une agence dispose au minimum 1 gestionnaire et au maximum plusieurs gestionnaires

Une agence dispose au minimum 1 compte et au maximum plusieurs comptes.

Un compte appartient à 1 et une seule agence.

Un gestionnaire appartient à 1 et une seule agence.

Un client dispose au minimum 1 compte et au maximum n comptes.

Un compte appartient à un et un seul client.

Un compte dispose au minimum 1 opération et au maximum n opérations.

Une opération ne peut que s’effectuer sur 1 et 1 seul compte

**Définition du modèle Entité-Association Merise**

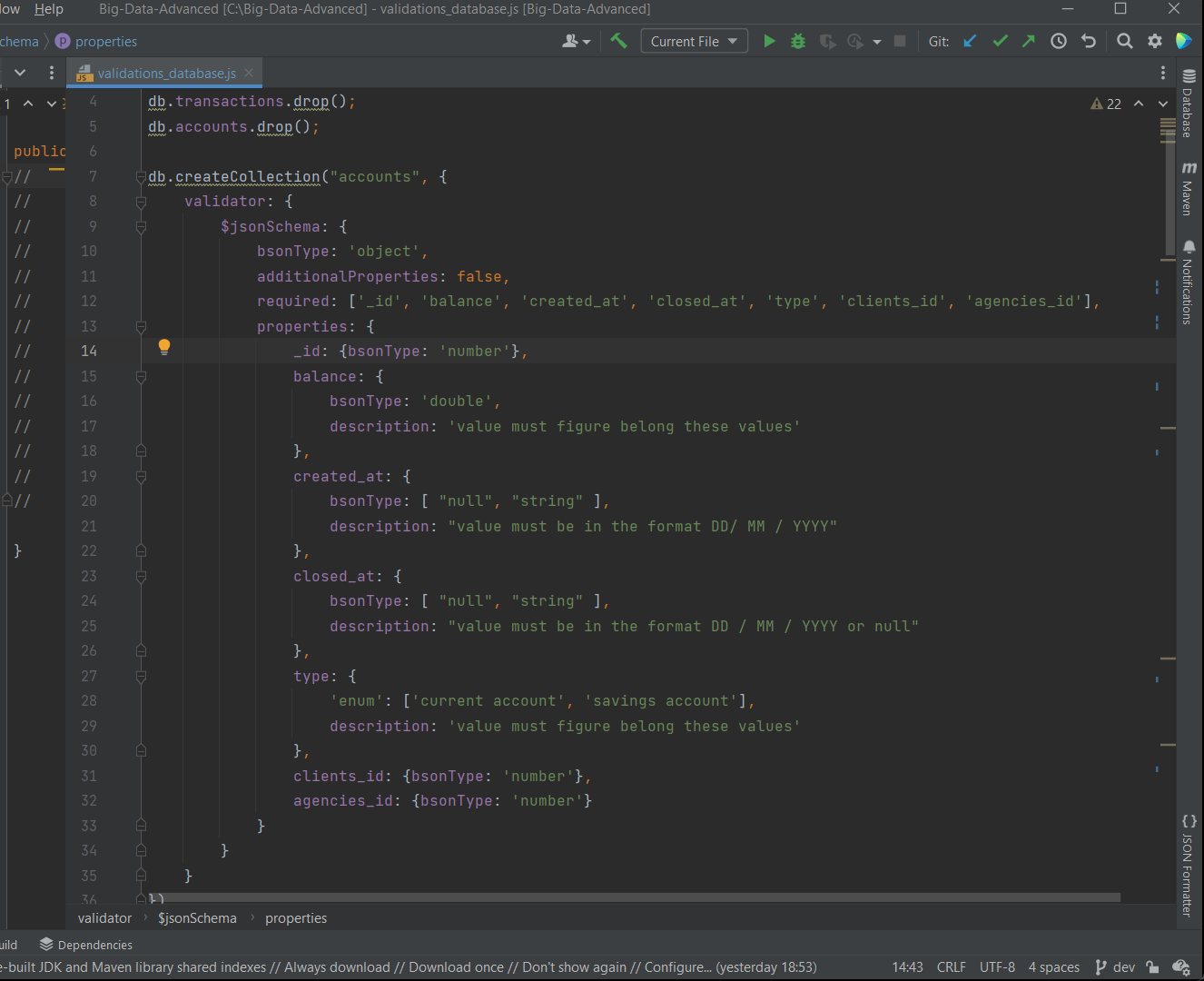
Une image contenant texte, nombre, capture d’écran, Police

Description générée automatiquement

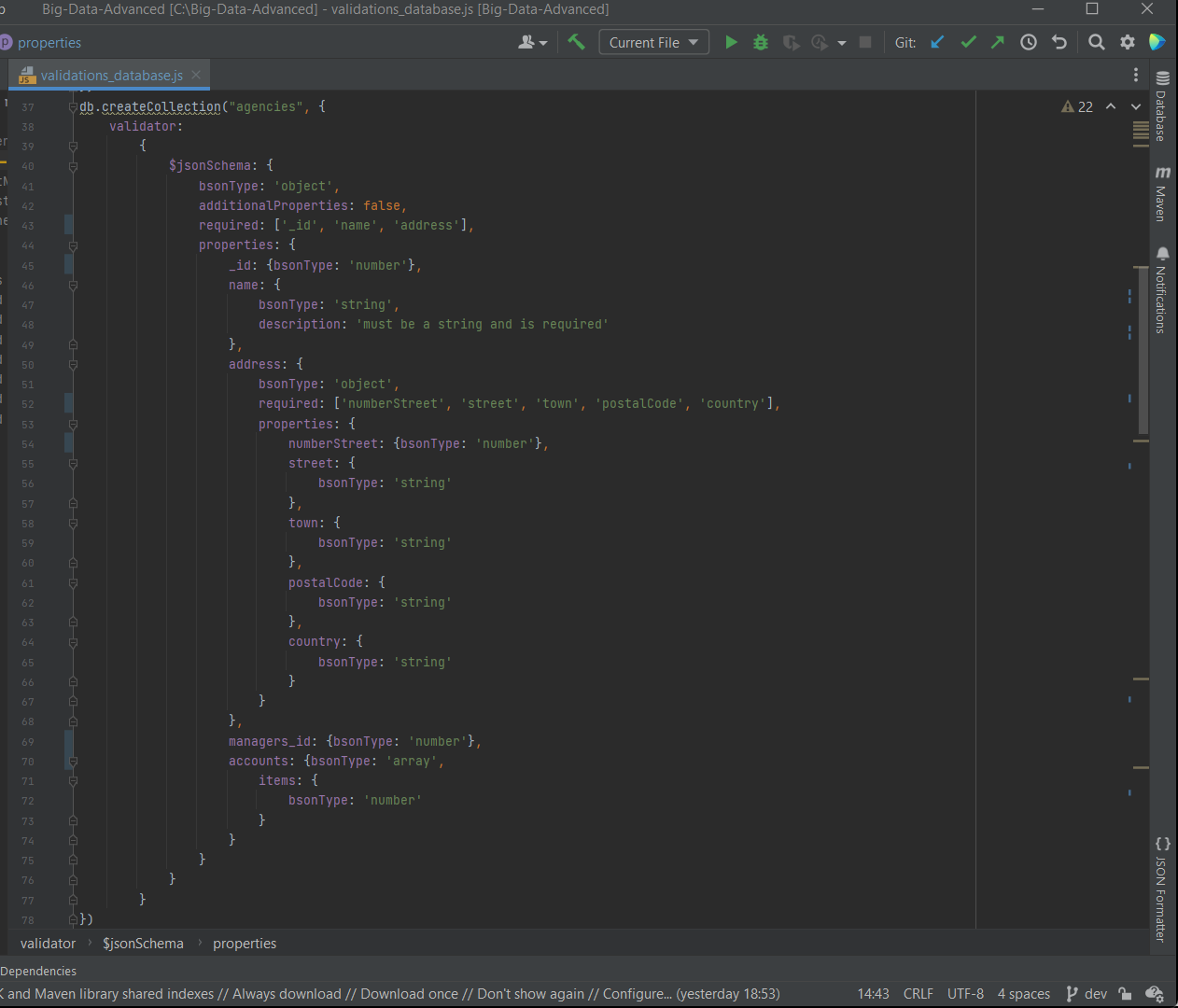
3. **Conversion du MCD MERISE en des objets MONGODB ou du moteur nosql de votre choix et des classes java**

3.1 **Spécification des modèles de documents à mettre dans chaque collection** :

**Compte :**



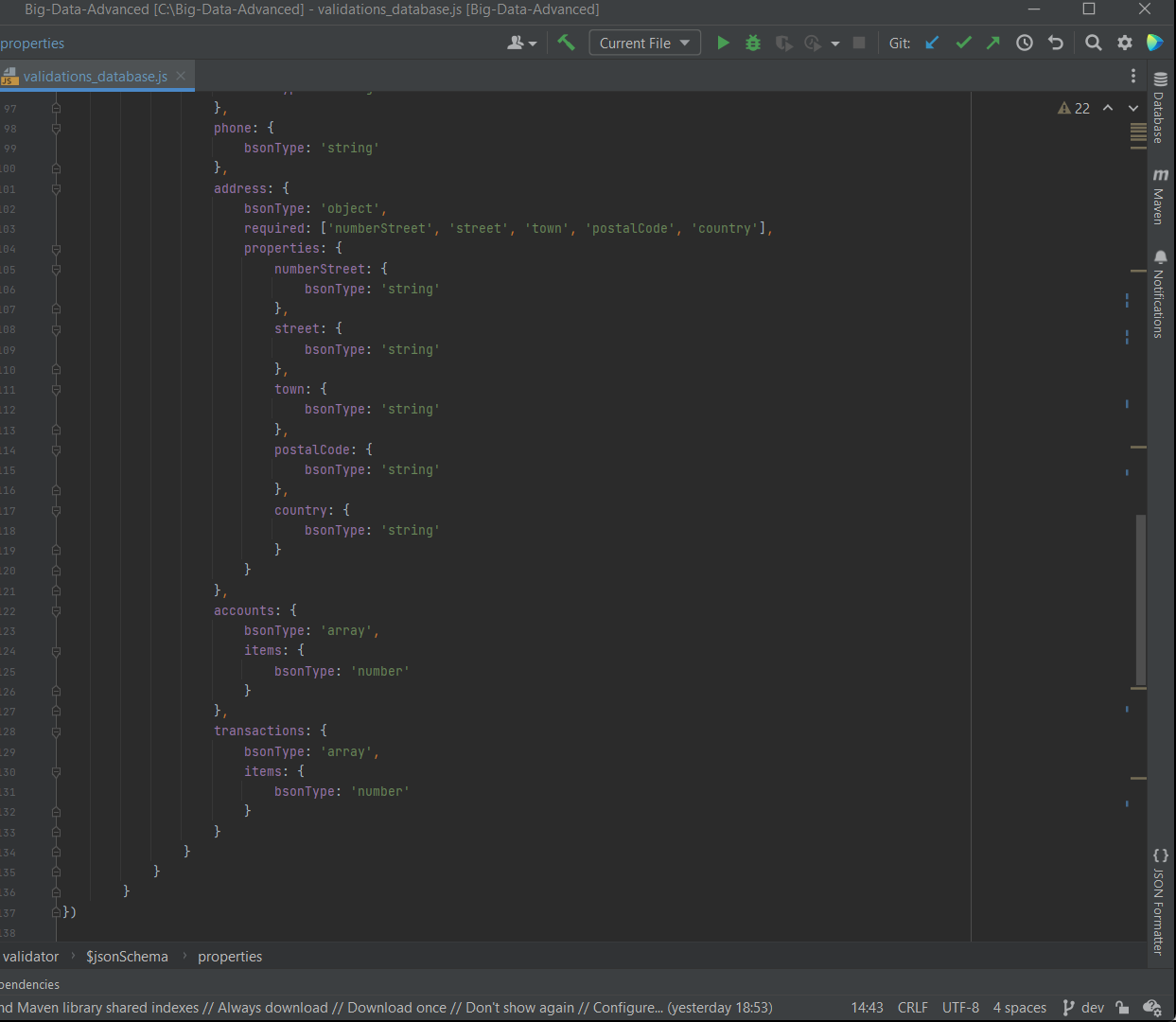
**Agence :**



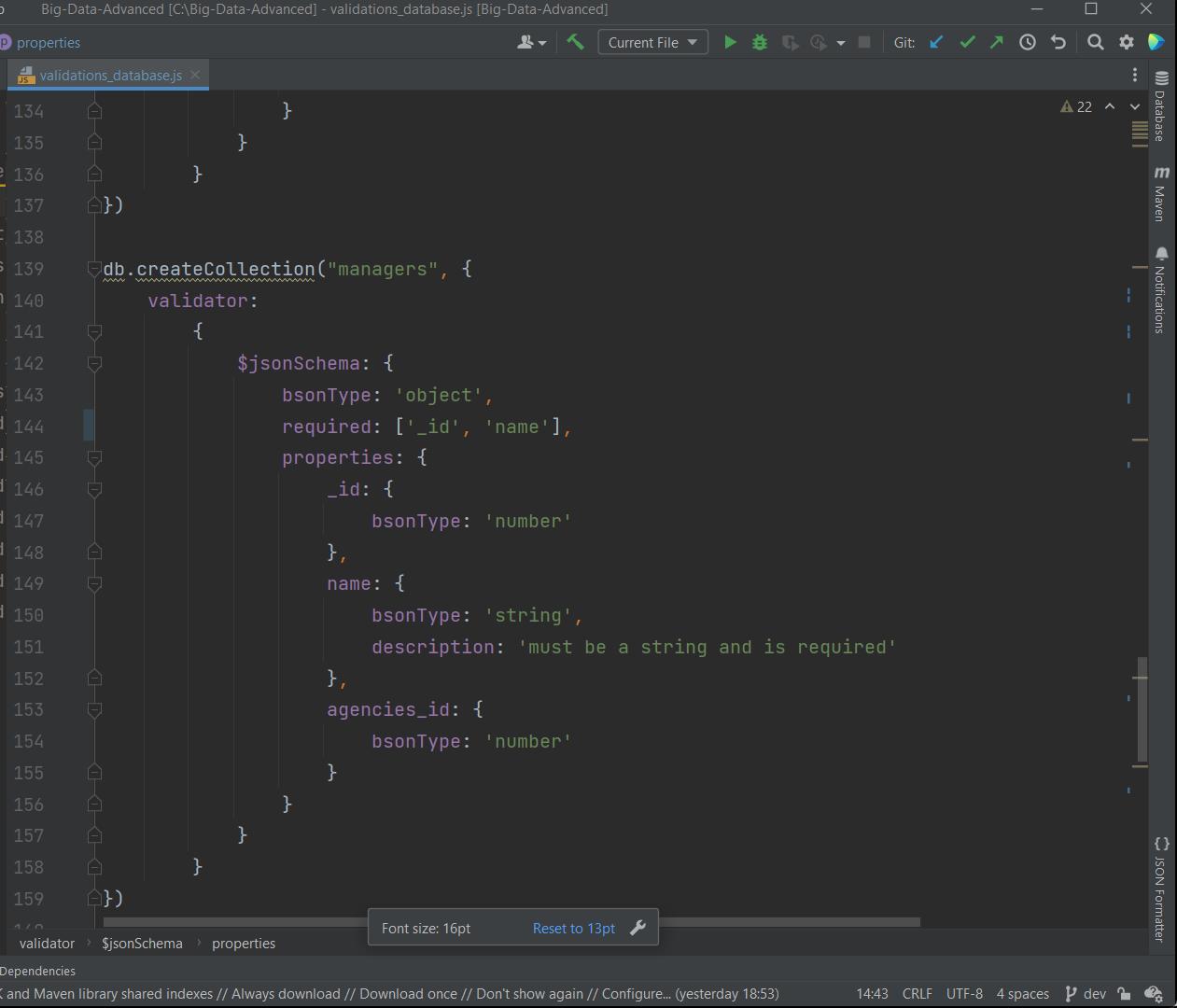
**Clients :**

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

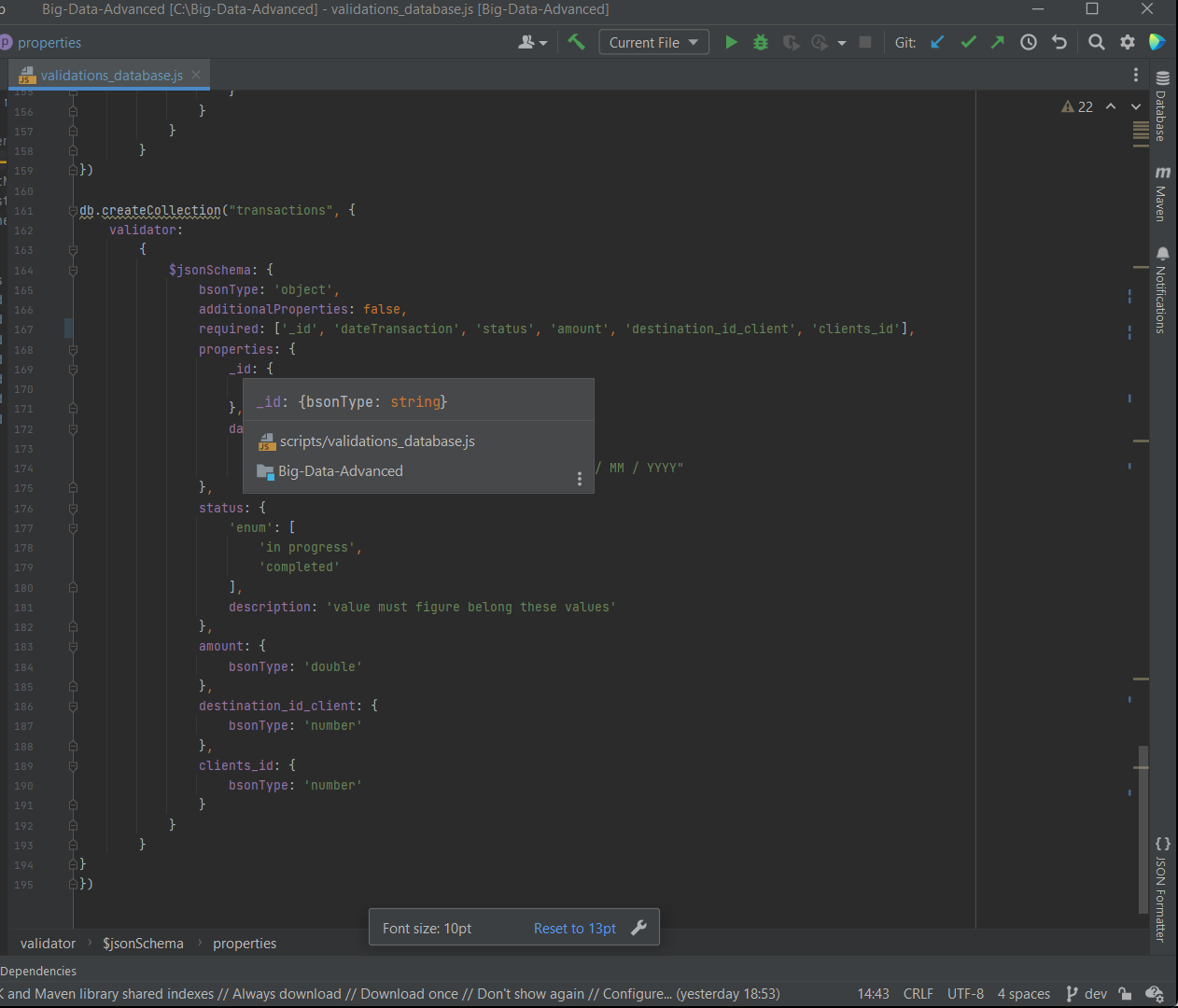
Description générée automatiquement



**Gestionnaire :**



**Transaction :**



Spécification des classes et méthodes java :

Classes : **Agence, Compte, Client, Gestionnaire, Transaction**

**Agence :**

**public void insertOneAgence(String nomCollection, Document agence);   
public void getAgenceById(String nomCollection, Integer agenceId);**

**public void getAgences(String nomCollection, Document whereQuery, Document projectionFields, Document sortFields);**

**public void updateAgences(String nomCollection, Document whereQuery,Document updateExpressions, UpdateOptions updateOptions)**

**public void deleteAgences(String nomCollection, Document filters)**

**Compte  :**

Méthodes CRUD :

**public void insertOneCompte(String nomCollection, Document compte);**

**public void insertManyComptes(String nomCollection, List<Document> comptes);**

**public void getCompteById(String nomCollection, Integer compteId);**

**public void getComptes(String nomCollection, Document whereQuery, Document projectionFields, Document sortFields);**

**public void updateComptes(String nomCollection,Document whereQuery,Document updateExpressions, UpdateOptions updateOptions);**

**public void deleteComptes(String nomCollection, Document filters);**

**Client :**

Méthodes CRUD :

**public void insertOneClient(String nomCollection, Document client);**

**public void getClients(String nomCollection, Document whereQuery, Document projectionFields, Document sortFields);**

**public void updateClients(String nomCollection,Document whereQuery,Document updateExpressions, UpdateOptions updateOptions);**

**public void deleteClients(String nomCollection, Document filters);**

**Gestionnaire :**

Méthodes CRUD :

**public void insertOneGestionnaire(String nomCollection, Document gestionnaire);**

**public void getGestionnaireById(String nomCollection, Integer gestionnaireId);**

**public void getGestionnaires(String nomCollection,Document whereQuery, Document projectionFields,Document sortFields);**

**public void updateGestionnaires(String nomCollection,Document whereQuery,Document updateExpressions, UpdateOptions updateOptions);**

**public void deleteGestionnaires(String nomCollection, Document filters);**

**Transaction :**

Méthodes CRUD :

**public void insertOneTransaction(String nomCollection, Document transaction);**

**public void insertManyTransactions(String nomCollection, List<Document> transactions);**

**public void getTransactionById(String nomCollection, Integer transactionId);**

**public void getTransactions(String nomCollection, Document whereQuery, Document projectionFields, Document sortFields);**

**public void updateTransactions(String nomCollection,Document whereQuery,Document updateExpressions, UpdateOptions updateOptions);**

**public void deleteTransactions(String nomCollection, Document filters);**

4.  Compléments sur le moteur NoSQL MongoDB

**Modèles de données supportées** :

Le moteur NoSQL MongoDB supporte les données suivantes :

**Modèle de donnée orienté documents :**

MongoDB est souvent utilisé comme une base de données orientée documents. Dans ce modèle, les données sont stockées sous forme de documents BSON (Binary JSON) flexibles, qui sont similaires aux objets JSON. Les documents peuvent contenir des paires clé-valeur, des tableaux, des sous-documents, etc. C'est un modèle de données très flexible et adapté à un large éventail de cas d'utilisation.

Exemple :

// Exemple de document BSON représentant un utilisateur

{

  \_id: ObjectId("60c930217e5c3d482862f1b3"),

  name: "John Doe",

  age: 30,

  email: "john.doe@example.com",

  address: {

    street: "123 Main Street",

    city: "New York",

    country: "USA"

  },

  interests: ["football", "music", "reading"]

}

**Modèle de données orienté colonnes :**

MongoDB prend également en charge le modèle de données orienté colonnes avec le moteur de stockage WiredTiger. Ce modèle permet de stocker les données de manière colonne par colonne plutôt que document par document.

Voici un exemple :

Supposons que nous avons une collection "employees" contenant des documents représentant des employés. Chaque document contient des informations telles que le nom, le département, le salaire et la date d'embauche de l'employé. Pour simuler un modèle de données orienté colonnes, nous pouvons organiser les informations en utilisant des champs distincts pour chaque colonne :

// Exemple de document BSON représentant un employé avec un modèle de données orienté colonnes

{

  \_id: ObjectId("60c930217e5c3d482862f1b3"),

  name: "John Doe",

  department: "IT",

  salary: 50000,

  hireDate: ISODate("2022-01-01")

}

Cependant, il est important de noter que cette approche a certaines limites et n'est pas aussi performante que des bases de données spécifiquement conçues pour le modèle de données orienté colonnes d’où' l’utilisation des moteur noSQL specialisés dans les modèles orientés colonnes comme : Apache Cassandra ou Apache HBase.

**Modèles de données orienté graphes :**

MongoDB offre des fonctionnalités de stockage et de requête de données orientées graphes à travers l'extension GraphLookup et les opérations $graphLookup et $lookup. Cela permet de représenter et de naviguer dans des données de graphes. Voici un exemple de requête graphique pour trouver les amis d'un utilisateur :

db.users.aggregate([

  {

    $match: { \_id: ObjectId("60c930217e5c3d482862f1b3") }

  },

  {

    $graphLookup: {

      from: "users",

      startWith: "$friends",

      connectFromField: "friends",

      connectToField: "\_id",

      as: "friends"

    }

  }

])

**Modèle de données orienté clé-valeur :**

MongoDB peut être utilisé pour stocker des données orientées clé-valeur en utilisant des documents BSON. Voici un exemple d'utilisation de MongoDB pour stocker des paires clé-valeur :

// Exemple de document BSON stockant des paires clé-valeur

{

  \_id: ObjectId("60c930217e5c3d482862f1b3"),

  data: {

    key1: "value1",

    key2: 42,

    key3: true

  }

}

**Architecture du moteur NoSQL :**

Organisé en SHARD

Une SHARD contient un serveur maître et des serveurs esclave

Une instance MongoDB comprend 1 ou plusieurs Bases de Données

Une base de données comprend 0, 1 ou plusieurs collections

Une collection contient 0,1 ou plusieurs documents

Méthode de partitionnement :

MongoDB utilise un mécanisme de partitionnement appelé "sharding" pour répartir les données sur plusieurs nœuds dans un cluster MongoDB. Voici comment MongoDB réalise son partitionnement :

1. Sharding Key :

* MongoDB utilise une clé de partitionnement, également appelée "sharding key", pour déterminer comment les données sont réparties entre les nœuds du cluster.
* La clé de partitionnement est généralement basée sur un ou plusieurs champs dans les documents MongoDB.
* MongoDB prend en charge différents types de clés de partitionnement, tels que les clés basées sur des valeurs uniques, des plages de valeurs, des hachages, etc.

1. Shards :

* Les données sont réparties en plusieurs "shards" (nœuds de stockage) dans le cluster.
* Chaque shard contient un sous-ensemble des données de la base de données.

1. Config Servers :

* MongoDB utilise également des "config servers" pour stocker les métadonnées relatives au partitionnement, telles que les informations sur les chunks de données et la configuration du cluster.
* Les config servers permettent au système de suivre l'emplacement des données et de gérer le partitionnement.

1. Chunking :

* Les données sont divisées en "chunks" (morceaux) en fonction de la clé de partitionnement.
* Chaque chunk correspond à une plage de valeurs de la clé de partitionnement.
* Les chunks sont distribués entre les shards en fonction des plages de valeurs qu'ils couvrent.

1. Balancing :

* MongoDB surveille la distribution des chunks entre les shards et effectue un rééquilibrage automatique si nécessaire.
* Le rééquilibrage s'assure que les chunks sont répartis de manière équitable entre les shards pour éviter les déséquilibres de charge.

1. Queries :

* Lorsqu'une requête est exécutée, MongoDB détermine quel shard contient les données requises en fonction de la clé de partitionnement.
* Les requêtes peuvent être dirigées vers un seul shard ou propagées à plusieurs shards si nécessaire.

**Méthode de réplication :**

La méthode de réplication dans MongoDB est basée sur un modèle maître-esclave (Master-Slave). Voici comment se présente la méthode de réplication dans MongoDB :

1. Nœud maître (Primary) :

* Le nœud maître est le nœud principal responsable de la gestion des opérations d'écriture (insertions, mises à jour, suppressions) et de la coordination de la réplication.
* Le nœud maître accepte les opérations d'écriture et les propage aux autres nœuds de la réplication.

1. Nœuds esclaves (Secondaries) :

* Les nœuds esclaves sont des répliques du nœud maître.
* Les nœuds esclaves reçoivent les opérations d'écriture provenant du nœud maître et les appliquent localement pour maintenir une copie à jour des données.

1. Opérations de réplication :

* Le nœud maître enregistre toutes les opérations d'écriture dans son journal d'opérations (oplog).
* Les nœuds esclaves lisent le journal d'opérations du nœud maître et appliquent les opérations dans le même ordre pour maintenir la cohérence des données.

1. Élection d'un nouveau nœud maître :

* Si le nœud maître devient indisponible, les nœuds esclaves peuvent élire un nouveau nœud maître parmi eux.
* L'élection est basée sur un algorithme de vote et garantit la continuité des opérations d'écriture et de lecture.

1. Réplication synchrone :

La réplication dans MongoDB offre une haute disponibilité et une tolérance aux pannes en assurant la redondance des données. Si le nœud maître devient indisponible, un nœud esclave peut prendre sa place et continuer à fournir un accès en lecture et en écriture aux données. De plus, les nœuds esclaves peuvent être utilisés pour effectuer des lectures supplémentaires afin de répartir la charge et d'améliorer les performances.

MongoDB propose également d'autres fonctionnalités de réplication avancées, telles que la réplication en cascade, la réplication géographique, la réplication avec des tags, qui permettent de répondre à des exigences spécifiques en matière de disponibilité, de performance et de localisation des données.

**Montée en charge** :

La montée en charge sous MongoDB peut être réalisée en utilisant plusieurs techniques et fonctionnalités offertes par le système. Voici quelques-unes des approches couramment utilisées pour la montée en charge dans MongoDB :

1. Réplication :

* MongoDB prend en charge la réplication, ce qui permet de créer des répliques des données sur plusieurs nœuds (appelés membres) dans un ensemble de réplicas.
* Chaque membre peut traiter les lectures et les écritures, ce qui répartit la charge entre les nœuds et améliore la disponibilité des données.
* Lorsqu'un nœud principal échoue, un autre nœud est élu automatiquement pour devenir le nouveau nœud principal, assurant ainsi une haute disponibilité.

1. Partitionnement (Sharding) :

* MongoDB prend également en charge le partitionnement, ou sharding, qui permet de distribuer les données sur plusieurs serveurs (shards).
* Les données sont réparties en fonction d'une clé de partitionnement spécifiée, ce qui permet de répartir la charge de manière équilibrée entre les shards.
* Chaque shard gère une partie du jeu de données, ce qui augmente la capacité de stockage et de traitement globale du système.

1. Indexation :

* L'indexation efficace des données est essentielle pour les performances et la montée en charge dans MongoDB.
* En créant les bons index sur les champs les plus fréquemment utilisés dans les requêtes, vous pouvez améliorer les performances des opérations de recherche et de tri.
* Les index aident également à réduire la charge sur les ressources système, en accélérant la récupération des données.

1. Mise à l'échelle horizontale :

* MongoDB facilite la mise à l'échelle horizontale en ajoutant de nouveaux nœuds à mesure que la charge augmente.
* Vous pouvez ajouter de nouveaux serveurs pour augmenter la capacité de stockage et de traitement du système, répartissant ainsi la charge entre les nœuds supplémentaires.

1. Cache :

* MongoDB utilise des mécanismes de cache en mémoire pour stocker les données les plus fréquemment utilisées, ce qui réduit les accès disque et améliore les performances.
* Le cache de lecture (WiredTiger Cache) stocke les pages de données et d'index les plus souvent accédées en mémoire.
* Le cache d'agrégation (Aggregation Pipeline) stocke les résultats intermédiaires des opérations d'agrégation pour accélérer les requêtes complexes.

1. Optimisation des requêtes et de la modélisation des données :

* Une optimisation minutieuse des requêtes et de la modélisation des données peut contribuer à améliorer les performances et à éviter les goulets d'étranglement lors de la montée en charge.
* Cela peut inclure la dénormalisation des données, la sélection appropriée des types de données, la réduction du nombre de requêtes et l'utilisation de techniques avancées telles que l'agrégation et les indexes composites.

**Gestion du ou des caches mémoire :**

Dans MongoDB, la gestion du cache mémoire est réalisée à plusieurs niveaux pour améliorer les performances du système. Voici comment MongoDB gère ses caches mémoire :

1. Cache de lecture (WiredTiger Cache) :

* MongoDB utilise le moteur de stockage WiredTiger par défaut, qui comprend un cache de lecture en mémoire.
* Ce cache de lecture (parfois appelé cache de page) stocke les pages de données et les index les plus fréquemment utilisés en mémoire.
* Lorsqu'une requête nécessite des données, MongoDB vérifie d'abord si elles se trouvent dans le cache de lecture. Si c'est le cas, les données sont retournées rapidement sans accéder au disque.

1. Cache d'index :

* MongoDB maintient également un cache d'index en mémoire pour améliorer les performances des requêtes basées sur les index.
* Ce cache stocke les index les plus fréquemment utilisés, ce qui permet d'accélérer les opérations de recherche et de tri des données.

1. Cache d'agrégation (Aggregation Pipeline) :

* Lors de l'utilisation de l'Aggregation Pipeline de MongoDB pour effectuer des opérations d'agrégation complexes, MongoDB utilise un cache pour stocker les résultats intermédiaires.
* Ce cache d'agrégation permet de réduire la nécessité de recalculer les étapes de l'agrégation à chaque exécution de la requête.

1. Gestion automatique du cache :

* MongoDB gère automatiquement la taille et l'éviction du cache mémoire en fonction de la charge de travail et de la disponibilité de la mémoire système.
* Le cache est dimensionné en fonction de la mémoire disponible sur le système, et MongoDB ajuste dynamiquement la taille du cache en fonction des besoins.

Il est à noter que la configuration et la gestion du cache mémoire peuvent être personnalisées en fonction des besoins spécifiques de l'application et de l'environnement. MongoDB fournit des options de configuration pour ajuster la taille du cache, contrôler la durée de vie des éléments en cache, et surveiller les métriques de performances liées au cache.

En optimisant l'utilisation de la mémoire et en maintenant les données les plus fréquemment utilisées en cache, MongoDB améliore les performances en réduisant les accès disque et en réduisant les délais de réponse des requêtes.