Agence bancaire

Auteurs:
Roméo David AMEDOMEY
Mohamed ELYSALEM
Tymotheus Igor Cyryl CIESIELSKI
Eliel WOTOBE
Prof : M. Gabriel MOPOLO

Description du sujet

Notre projet consiste à concevoir la base de données NoSQL Redis pour une agence bancaire et faire le mapping en java.

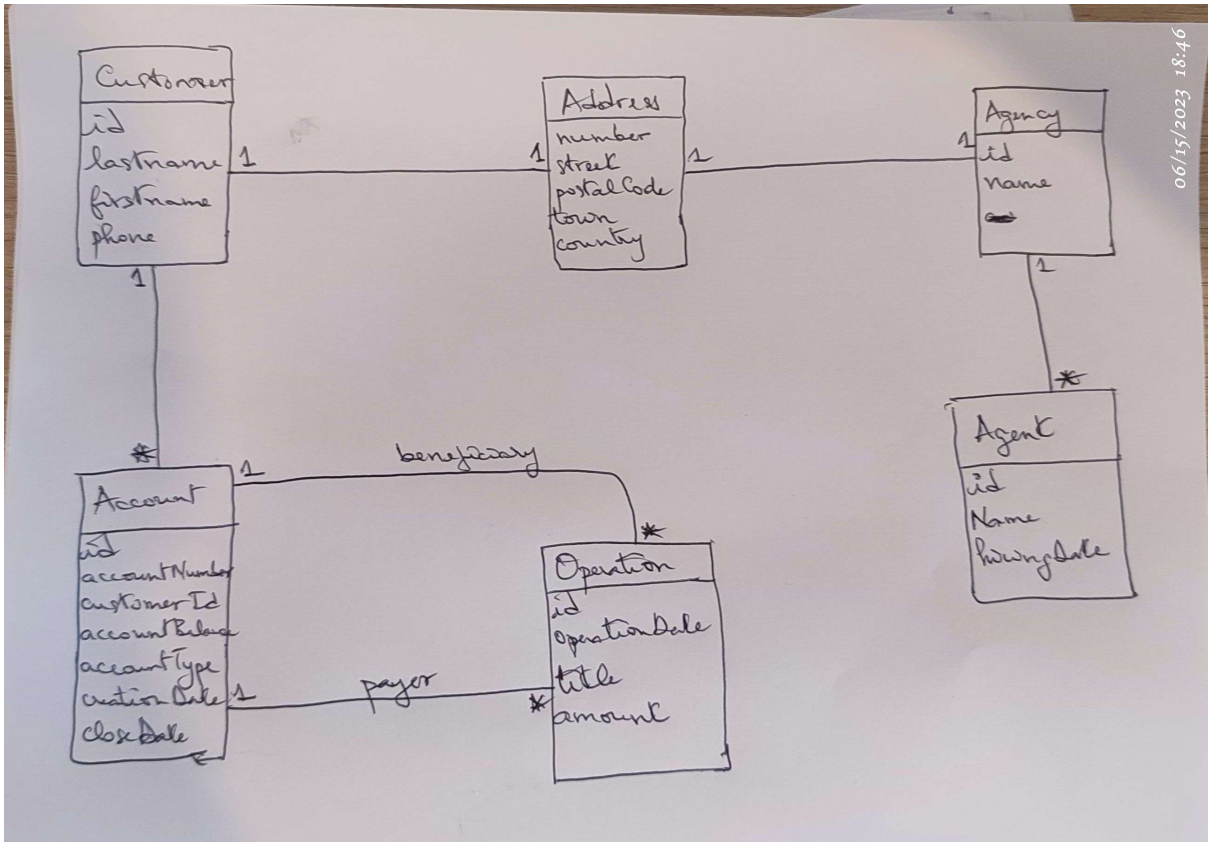
Cette application est conçue pour stocker les information des clients d'une agence bancaire et leurs comptes. Les utilisateurs pourront dans le cadre de la gestion des comptes, rechercher un client, rechercher ses comptes, effectuer des transactions tenir à jour ses informations. Il est possible d'effectuer des opérations compte à compte.

Spécification, Analyse et conception

Dictionnaire des données

Entité	Propriété	Description	Type	Contraintes
Customer	id	Identifiant du client	int	Unique
	lastname	Le nom du client	String	
	firstname	Les prénoms du client	List of string	
	phone	Le numéro portable du client	String	
	address	L'adresse du client	Address	
Agency	Id	Identifiant de l'agence	Int	Unique
	Name	La désignation de l'agence	String	
	Address	Adresse de l'agence	Address	
Operation	Id	Identifiant de l'opération	Int	Unique
	operationDate	Date de création de l'opération	Date	
	Title	Description de l'opération	String	
	Amount	Le montant de l'operation		
	payerAccount	Identifiant du compte débité	String	
	beneficiaryAccount	Identifiant du compte crédité	String	
Account	Id	Identifiant du compte	Int	Unique
	accountNumber	Numéro de compte	String	Unique
	customerId	Identifiant du proprietaire du compte	Int	
	accountBalance	Le solde du compte	Int	
	accountType	Le type du compte	String	
	creationDate	Date de création	Date	

	closeDate	Date de fermeture du compte	Date	
Agent	Id	Identifiant de l'agent	Int	Unique
	Name	Nom de l'agent	String	
	bankId	Identifiant de l'agence	Int	
	hiringDate	Date d'embauche	Date	



Règles de gestion

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.

Conversion merise en objet Redis

Étant donné que le moteur de base de données Redis est de type clé/valeur, nous avons opté pour le mapping des objets avec des classes java sérialisable. Toutes les entités sont donc représentées par des classes que l'on peut transformer en objet json ou créer à partir d'objet json. Cet objet json est stocker en base de données sous forme de chaine de caractère.

Quant aux associations, lorsqu'on rencontre une association one to one, on préfère stocker l'association en tant que sous model. Pour reste, il a été géré comme du relationnel.

Toutes les classes ont un service associé qui permet de gérer les opérations en base de données. Nous avons principalement les méthodes suivantes :

- save : pour enregistrer en base de données
- getAll : pour récupérer toutes les valeurs d'un type donné
- getById : pour récupérer par l'identifiant
- updateValueForAll : pour mettre à jour une valeur pour tout (updateMany)
- updateById : pour mettre à jour les valeurs pour un seul
- delete : pour supprimer un élement connaissant sont id.

Nous avons utilisé une convention de nommage qui associe le nom de la classe avec l'identifiant pour de l'élément pour l'enregistrer en base : c'est son identifiant. A titre d'exemple, pour le client avec l'identifiant 1, sont identifiant en base est customer:1. « customer: « est commun à tous les clients. Pour la recherche en base de données, nous avons un index de type redisearch par type. Toujours pour le client, on a : idx_customer. C'est cet index qui nous permet d'aller vite dans la recherche et de pouvoir rechercher par un champ sans avoir à tout faire dans le programme.

Complément sur le moteur redis

1. Modèles de données supportés : Redis est un moteur de base de données NoSQL qui prend en charge plusieurs modèles de données. Les principaux modèles de données pris en charge par Redis sont les suivants :
    - Paires clé-valeur : Redis est souvent utilisé comme une simple base de données clé-valeur, où chaque clé est associée à une valeur. Les valeurs peuvent être de différents types tels que chaînes, listes, ensembles, hachages et ensembles ordonnés.
    - Listes : Redis prend en charge les opérations sur les listes, telles que l'ajout d'éléments en tête ou en queue, la récupération d'éléments par index, la modification d'éléments existants, etc.
    - Ensembles : Redis permet de stocker des ensembles non ordonnés d'éléments uniques. Il fournit des opérations pour effectuer des unions, des intersections et des différences entre ensembles, ainsi que pour ajouter ou supprimer des éléments d'un ensemble.
    - Hachages : Redis offre la possibilité de stocker des paires clé-valeur dans des hachages. Les hachages sont utiles lorsque vous avez besoin de stocker des structures de données plus complexes, telles que des objets ou des enregistrements.

3. Ensembles ordonnés : Redis permet de stocker des ensembles d'éléments avec une valeur associée à chaque élément. Les ensembles ordonnés sont triés en fonction de la valeur associée, ce qui permet de récupérer rapidement les éléments selon différents critères.

4. Architecture du moteur NoSQL (Redis) : L'architecture de Redis repose sur un modèle client-serveur.

Dans cette architecture, l'application communique avec Redis via un client Redis. Le serveur Redis gère la logique de stockage et de récupération des données, ainsi que les opérations spécifiques à Redis telles que la manipulation des clés et des valeurs.

Redis utilise une architecture en mémoire, ce qui signifie que les données sont stockées principalement en mémoire vive, offrant des performances de lecture et d'écriture extrêmement rapides. Les données peuvent également être persistées sur le disque

Génération automatique de données

Des données générées automatiquement sont présents dans le dossier ressource du projet (json). Le chargement des données est effectué dans les tests.