TD/TP R3.07
SQL dans un langage de prog
1. JDBC

2025/2026

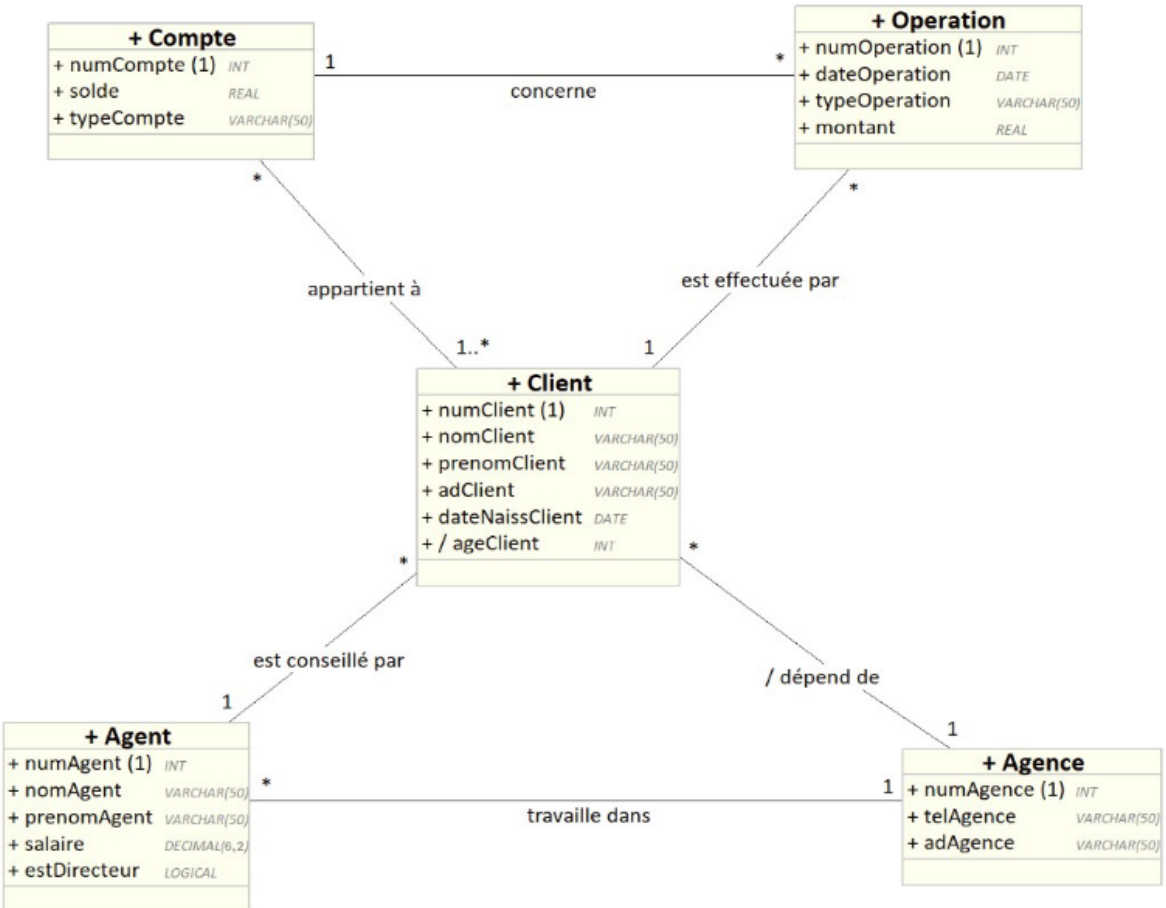
1 Introduction

Dans ce TD/TP, vous allez développer une application Java qui interagit avec une base de données relationnelle en utilisant JDBC. Vous devrez également dockeriser votre application et la base de données.

2 Prérequis

— Connaissances de base en Java.
— Docker installé sur votre machine et une bonne connexion Internet.
— L'IDE IntelliJ IDEA de Jetbrains.

3 Diagramme de la base de données



4 Description de la base de données

Le schéma de la base de données est le suivant :

— **Compte** : numCompte (INT), solde (REAL), typeCompte (VARCHAR).

— **Client** : numClient (INT), nomClient (VARCHAR), prenomClient (VARCHAR), adClient (VARCHAR), dateNaissClient (DATE), ageClient (INT).

— **Opération** : numOperation (INT), dateOperation (DATE), typeOperation (VARCHAR), montant (REAL).

— **Agent** : numAgent (INT), nomAgent (VARCHAR), prenomAgent (VARCHAR), salaire (DECIMAL), estDirecteur (LOGICAL).

— **Agence** : numAgence (INT), telAgence (VARCHAR), adAgence (VARCHAR).

5 Mise en place de la base de données avec Docker

Vous allez utiliser Docker pour créer une instance PostgreSQL contenant cette base de données.

5.1 Étape 1 : Créer un docker-compose.yml pour la base de données

Créez un fichier `docker-compose.yml` pour lancer un conteneur PostgreSQL avec la base de données initialisée.

```yaml
version: '3.8'
services:
  db:
    image: postgres:18
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: banque_db
    ports:
      - "5432:5432"
    volumes:
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
```

5.2 Étape 2 : Script SQL d'initialisation

Créez un fichier `init.sql` pour initialiser la base de données avec les tables et quelques données.

```sql
CREATE TABLE IF NOT EXISTS Agence (
    numAgence INT PRIMARY KEY,
    telAgence VARCHAR(50),
    adAgence VARCHAR(100)
);

CREATE TABLE IF NOT EXISTS Agent (
    numAgent INT PRIMARY KEY,
    nomAgent VARCHAR(50),
    prenomAgent VARCHAR(50),
    salaire DECIMAL(10, 2),
    estDirecteur BOOLEAN,
    numAgence INT REFERENCES Agence(numAgence)
);

CREATE TABLE IF NOT EXISTS Client (
    numClient INT PRIMARY KEY,
    nomClient VARCHAR(50),
    prenomClient VARCHAR(50),
    adClient VARCHAR(100),
    dateNaissClient DATE,
    ageClient INT,
    numAgent INT REFERENCES Agent(numAgent)
);

CREATE TABLE IF NOT EXISTS Compte (
    numCompte INT PRIMARY KEY,
    solde REAL,
```

```
    typeCompte VARCHAR(50)
);

CREATE TABLE IF NOT EXISTS Operation (
    numOperation INT PRIMARY KEY,
    dateOperation DATE,
    typeOperation VARCHAR(50),
    montant REAL,
    numCompte INT REFERENCES Compte(numCompte)
);
```

6 Tâches à réaliser

Vous devez implémenter les méthodes Java suivantes en utilisant JDBC pour interagir avec la base de données. Lancez IntelliJ IDEA sur votre PC et créez un projet Java avec Maven comme système de build.

6.1 Méthodes à implémenter

Voici quelques méthodes que vous devez implémenter dans une classe `BanqueDAO` :

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class BanqueDAO {
 private Connection connection;

 public BanqueDAO(Connection connection) {
  this.connection = connection;
 }

 // Methode pour se connecter au serveur de donnees
 public static Connection getConnection() throws SQLException {
  // TO COMPLETE
 }

 // Methode pour ajouter une agence
 public void ajouterAgence(Agence agence) throws SQLException {
  // TO COMPLETE
 }

 // Methode pour ajouter un agent
 public void ajouterAgent(Agent agent) throws SQLException {
  // TO COMPLETE
 }

 // Methode pour ajouter un client
 public void ajouterClient(Client client) throws SQLException {
  // TO COMPLETE
 }

 // Methode pour obtenir un client par son numero
 public Client obtenirClientParNum(int numClient) throws SQLException {
  // TO COMPLETE
 }

 // Methode pour ajouter une operation a un compte
 public void ajouterOperation(Operation operation) throws SQLException {
  // TO COMPLETE
 }

 // Methode pour obtenir le solde d'un compte
 public double obtenirSoldeCompte(int numCompte) throws SQLException {
  // TO COMPLETE
 }

 // Methode pour obtenir toutes les operations d'un compte
 public List<Operation> obtenirOperationsParCompte(int numCompte) throws SQLException {
  // TO COMPLETE
```

```
 }

 // Methode pour obtenir tous les clients conseilles par un agent
 public List<Client> obtenirClientsParAgent(int numAgent) throws SQLException {
  // TO COMPLETE
 }
}
```

6.2 Classes métier

Voici les classes métiers que vous devez utiliser :

```
public class Client {
    private int numClient;
    private String nomClient;
    private String prenomClient;
    private String adClient;
    private java.sql.Date dateNaissClient;
    private int ageClient;
    private int numAgent;

    public Client(int numClient, String nomClient, String prenomClient, String adClient,
    java.sql.Date dateNaissClient, int numAgent) {
        this.numClient = numClient;
        this.nomClient = nomClient;
        this.prenomClient = prenomClient;
        this.adClient = adClient;
        this.dateNaissClient = dateNaissClient;
        // this.agentClient = ...
        this.numAgent = numAgent;
    }

    // Getters et setters. A completer
}

public class Compte {
    private int numCompte;
    private double solde;
    private String typeCompte;

    public Compte(int numCompte, double solde, String typeCompte) {
        this.numCompte = numCompte;
        this.solde = solde;
        this.typeCompte = typeCompte;
    }

    // Getters et setters. A completer
}

public class Operation {
    private int numOperation;
    private java.sql.Date dateOperation;
    private String typeOperation;
    private double montant;
    private int numCompte;

    public Operation(int numOperation, java.sql.Date dateOperation, String typeOperation,
     double montant) {
        this.numOperation = numOperation;
        this.dateOperation = dateOperation;
        this.typeOperation = typeOperation;
        this.montant = montant;
    }

    // Getters et setters. A completer
}

public class Agent {
    private int numAgent;
    private String nomAgent;
    private String prenomAgent;
    private double salaire;
    private boolean estDirecteur;
```

```
60
61      // Constructeurs, getters et setters
62  }
63
64  public class Agence {
65      private int numAgence;
66      private String telAgence;
67      private String adAgence;
68
69      // Constructeurs, getters et setters
70  }
```

7 Dockerisation de l'application

7.1 Étape 1 : Créer un Dockerfile pour l'application Java

Créez un fichier `Dockerfile` pour votre application Java.

```
# Utilisation d'une image Maven pour construire l'application
FROM maven:3.9.6-eclipse-temurin-17 AS build
WORKDIR /app
COPY pom.xml .
RUN mvn dependency:go-offline
COPY src ./src
RUN mvn package

# Utilisation d'une image JRE pour executer l'application
FROM eclipse-temurin:17-jre-jammy
WORKDIR /app
COPY --from=build /app/target/jdbc.example-1.0.0.jar app.jar
CMD ["java", "-jar", "app.jar"]
```

7.2 Étape 2 : Mettre à jour le docker-compose.yml

Ajoutez un service pour votre application dans le fichier `docker-compose.yml`.

```
services:
  app:
    build: .
    depends_on:
      - db
    ports:
      - "8080:8080"
    environment:
      DB_URL: jdbc:postgresql://db:5432/banque_db
      DB_USER: user
      DB_PASSWORD: password
```

Mettez à jour le `docker-compose.yml` pour ne pas exposer le port du serveur Postgres et créez un réseau virtuel Docker entre les deux services pour qu'ils puissent interagir.

8 Livrables

- Un dépôt Git contenant votre code source.
- Une archive Zip avec les sources et fichiers de configuration :
  - Un fichier `docker-compose.yml` et un `Dockerfile` pour votre application.
  - Un fichier `init.sql` pour initialiser la base de données.
- Un rapport (fichier Markdown) expliquant les choix techniques et les difficultés rencontrées.

9 Conseils

- Utilisez des requêtes préparées pour éviter les injections SQL.
- Gérez correctement les exceptions SQL.
- Testez chaque méthode individuellement avant de passer à la suivante.