Base de donnée **Présentation**

Alex Schlageter A2

Sommaire

01

Mise en place

Mise en place

```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]: aschlag2
Mot de passe pour l'utilisateur aschlag2 :
psql (14.1)
Attention : l'encodage console (850) diffère de l'encodage Windows (1252).
            Les caractères 8 bits peuvent ne pas fonctionner correctement.
            Voir la section « Notes aux utilisateurs de Windows » de la page
            référence de psql pour les détails.
Saisissez « help » pour l'aide.

postgres=> \dt
            Liste des relations
 SchÚma |    Nom     | Type  | PropriÚtaire
--------+------------+-------+--------------
 public | agence     | table | aschlag2
 public | categorie  | table | aschlag2
 public | client     | table | aschlag2
 public | contrat    | table | aschlag2
 public | facture    | table | aschlag2
 public | marque     | table | aschlag2
 public | modele     | table | aschlag2
 public | type       | table | aschlag2
 public | vehicule   | table | aschlag2
 public | ville      | table | aschlag2
(10 lignes)


postgres=>
```

Affichage des tables de la base sur le PSQL

1. Installation de PostgreSQL au TP1
2. Création de la base via le Shell SQL (PSQL)
3. Connexion à la base via IntelliJ IDEA
4. Remplissage de la base avec des tables à l'aide d'un script :

```
36  create table Agence
37  (
38      idAgence serial primary key,
39      nbEmployes int not null,
40      idVille INT,
41      CONSTRAINT fk_ville
42          FOREIGN KEY(idVille)
43              REFERENCES ville(idVille)
44  );
```

Clé primaire (serial)

Autres attributs

Clé étrangère (ici une agence est situé dans une ville ⇩)

Exemple de code pour la table "Agence"

```
3   create table Ville
4   (
5       idVille serial primary key,
6       nomVille varchar not null,
7       nombreHabitants int not null
8   );
```

Exemple de code pour la table "Ville"

Mise en place

5. Remplissage des tables avec des données de test :

```
-- insert Agence
insert into agence values (0, 2, 0);
insert into agence values (1, 78, 1);
insert into agence values (2, 41, 2);
insert into agence values (3, 22, 3);
insert into agence values (4, 25, 4);
```

Exemple de code pour l'insertion des données dans la table Agence ...

- Avec l'id de l'agence, le nombre d'employés et l'id de la ville

```
-- Insert Ville
insert into ville values (0, 'Altkirch', 6000);
insert into ville values (1, 'Mulhouse', 100000);
insert into ville values (2, 'Strasbourg', 300000);
insert into ville values (3, 'Colmar', 70000);
insert into ville values (4, 'Saint-louis', 95000);
```

... Et pour la table Ville

- Avec l'id de la ville, le nom et le nombre d'habitants

Mise en place

6. Implémentation de la fonction de connexion dans notre code :

```java
// Connexion à la bdd
static Connection connexion() throws SQLException {
    String url = "jdbc:postgresql://localhost/postgres";
    Properties props = new Properties();
    props.setProperty("user","aschlag2");
    props.setProperty("password","2204");
    Connection conn = DriverManager.getConnection(url, props);
    return conn;
}
```

Serveur/nom de la base

- Cette fonction sera appelé à chaque action que l'on effectue sur la base :
  - Opérations CRUD
  - Différentes requêtes du projet

02

Entités

Entités

Chaque table de la base possède sa classe qui hérite de la classe abstraite “Entity”. (Exemple avec la classe Agence ci-contre)

Ces classes ont les attributs de la table correspondante, différents constructeurs, ainsi que les getter et setter qui nous aideront lors de l’établissement du CRUD. Il y’a également un toString qui nous permettra de mieux mettre en évidence les résultats de nos tests.

```java
public class Agence extends Entity {
    private int idAgence;
    private int nbEmployes;
    private Ville ville;

    public Agence() { this( id: 0); }
    public Agence(int id) { this(id,  nbEmployes: 0); }
    public Agence(int id, int nbEmployes) { this(id, nbEmployes,  ville: null); }
    public Agence(int id, int nbEmployes, Ville ville) {
        super();
        this.idAgence = id;
        this.nbEmployes = nbEmployes;
        this.ville = ville;
    }
    public int getIdAgence() { return idAgence; }
    public void setIdAgence(int id) { this.idAgence = id; }
    public int getNbEmploye() { return nbEmployes; }
    public void setNbEmployes(int nbEmployes) { this.nbEmployes = nbEmployes; }
    public Ville getVille() { return ville; }
    public void setVille(Ville ville) { this.ville = ville; }

    @Override
    public String toString() {
        return "Agence{" +
                "idAgence=" + idAgence +
                ", nbEmployes=" + nbEmployes +
                ", ville=" + ville +
                '}';
    }
}
```

03

Contraintes

Contraintes

- Utilisation de la classe interface Dao :

Qui déclare les fonctions (abstraites) qui seront utilisées pour le CRUD.

```java
public interface Dao {
    Collection<Entity> findAll() throws DaoException;
    Entity findById(int id) throws DaoException;
    void create(Entity entity) throws DaoException;
    void update(Entity entity) throws DaoException;
    void delete(Entity entity) throws DaoException;

}
```

Class Dao

- Utilisation de la classe abstraite JdbcDao :

Qui implémente la classe précédente et qui prend une connexion dans son constructeur. (cela va nous permettre de créer une nouvelle classe pour chaque table qui va pouvoir faire la connexion avec la base)

```java
public abstract class JdbcDao implements Dao {
    protected Connection connection;
    public JdbcDao(Connection connection) {
        this.connection = connection;
    }

}
```

Class JdbcDao

Contraintes

- Classes d'implémentations Dao pour chaque table. (Exemple avec la classe AgenceDaoImpl ci-contre)

Fonction findAll pour afficher toutes les agences (fonction Read du cRud)

```
public class AgenceDaoImpl extends JdbcDao {
    private VilleDaoImpl villeDao;

    public AgenceDaoImpl(Connection connection) {
        super(connection);
        villeDao = new VilleDaoImpl(connection);
    }

    @Override
    public Collection<Entity> findAll() throws DaoException {
        Collection<Entity> agences = new ArrayList<>();
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery( sql: "SELECT * FROM agence");
            while (resultSet.next()) {
                Agence agence = new Agence();
                agence.setIdAgence(resultSet.getInt( columnLabel: "idAgence"));
                agence.setNbEmployes(resultSet.getInt( columnLabel: "nbEmployes"));
                Ville ville = (Ville) villeDao.findById(resultSet.getInt( columnLabel: "idVille"));
                agence.setVille(ville);
                agences.add(ville);
            }
        } catch (SQLException e) {
            throw new DaoException(e);
        }
        return agences;
    }
```

...

Contraintes

Autres fonctions :

Fonction findById qui permet d'afficher une agence particulière.

Fonction create qui permet de créer une agence.

```java
@Override
public Entity findById(int id) throws DaoException {
    Agence agence = null;
    String sqlReq = "SELECT * FROM agence WHERE idAgence = ?";
    PreparedStatement statement = null;
    try {
        statement = connection.prepareStatement(sqlReq);
        statement.setInt( parameterIndex: 1, id);
        ResultSet resultSet = statement.executeQuery();
        while (resultSet.next()) {
            agence = new Agence();
            agence.setIdAgence(resultSet.getInt( columnLabel: "idAgence"));
            agence.setNbEmployes(resultSet.getInt( columnLabel: "nbEmployes"));
            Ville ville = (Ville)
                    villeDao.findById(resultSet.getInt( columnLabel: "idVille"));
            agence.setVille(ville);
        }
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
    return agence;
}

@Override
public void create(Entity entity) throws DaoException {
    Agence agence = (Agence) entity;
    PreparedStatement statement = null;
    String sqlReq = "insert into agence(idAgence, nbEmployes) values (?, ?)";
    try {
        statement = connection.prepareStatement(sqlReq);
        statement.setInt( parameterIndex: 1, agence.getIdAgence());
        statement.setInt( parameterIndex: 2, agence.getNbEmploye());
        int res = statement.executeUpdate();
    } catch (SQLException e) {
        throw new DaoException(e);
    }
}
```

Contraintes

Dernières fonctions :

Fonction update qui permet de mettre à jour une agence.

Fonction delete qui permet de supprimer une agence.

```java
@Override
public void update(Entity entity) throws DaoException {
    Agence agence = (Agence) entity;
    PreparedStatement statement = null;
    String sqlReq = "update agence set nbEmployes = ?, idVille = ? where idAgence = ?";
    try {
        statement = connection.prepareStatement(sqlReq);
        statement.setInt( parameterIndex: 1, agence.getNbEmploye());
        statement.setInt( parameterIndex: 2, agence.getVille().getIdVille());
        statement.setInt( parameterIndex: 3, agence.getIdAgence());

        int res = statement.executeUpdate();
    } catch (SQLException e) {
        throw new DaoException(e);
    }
}

@Override
public void delete(Entity entity) throws DaoException {
    Agence agence = (Agence) entity;
    PreparedStatement statement = null;
    String sqlReq = "delete from agence where idAgence = ?";
    try {
        statement = connection.prepareStatement(sqlReq);
        statement.setInt( parameterIndex: 1, agence.getIdAgence());
        int res = statement.executeUpdate();
    } catch (SQLException e) {
        throw new DaoException(e);
    }
}
```

04

Tests

Tests

Tests sur le CRUD avec la table Ville :

```java
public static void main(String[] args) throws DaoException, SQLException {
    // TEST CRUD
    VilleDaoImpl villeDao = new VilleDaoImpl(connexion());
    Ville v1 = new Ville( id: 5, nomVille: "Paris", nombreHabitants: 2100000);
    Ville v2 = new Ville( id: 5, nomVille: "Marseille", nombreHabitants: 900000);

    System.out.println("Création de Paris : ");
    villeDao.create(v1);
    System.out.println(villeDao.findAll());
    System.out.println(villeDao.findById(5));

    System.out.println("Modification de Paris en Marseille : ");
    villeDao.update(v2);
    System.out.println(villeDao.findById(5));

    System.out.println("Suppression de Marseille : ");
    villeDao.delete(v2);
    System.out.println(villeDao.findAll());
```

Code pour le test du CRUD

```
Création de Paris :
[Ville{idVille=0, nomVille='Altkirch', nombreHabitants=6000}
, Ville{idVille=1, nomVille='Mulhouse', nombreHabitants=100000}
, Ville{idVille=2, nomVille='Strasbourg', nombreHabitants=300000}
, Ville{idVille=3, nomVille='Colmar', nombreHabitants=70000}
, Ville{idVille=4, nomVille='Saint-louis', nombreHabitants=95000}
, Ville{idVille=5, nomVille='Paris', nombreHabitants=2100000}
]
Ville{idVille=5, nomVille='Paris', nombreHabitants=2100000}

Modification de Paris en Marseille :
Ville{idVille=5, nomVille='Marseille', nombreHabitants=900000}

Suppression de Marseille :
[Ville{idVille=0, nomVille='Altkirch', nombreHabitants=6000}
, Ville{idVille=1, nomVille='Mulhouse', nombreHabitants=100000}
, Ville{idVille=2, nomVille='Strasbourg', nombreHabitants=300000}
, Ville{idVille=3, nomVille='Colmar', nombreHabitants=70000}
, Ville{idVille=4, nomVille='Saint-louis', nombreHabitants=95000}
]
```

Résultats d'exécution

Tests

Tests sur les requêtes (1 exemple):
(Requête n°5 (Nombre de véhicules pour chaque marque))

```java
// R5
public static void nbVehiculesPourChaqueMarque() throws SQLException {
    connexion();
    Statement statement = connexion().createStatement();
    ResultSet resultSet = statement.executeQuery( sql: "SELECT Marque.nomMarque, COUNT(Vehicule.immatriculation) as nbVehicules " +
                                                  "FROM Vehicule " +
                                                  "INNER JOIN Marque ON Marque.idMarque = Vehicule.idMarque " +
                                                  "GROUP BY Marque.nomMarque;");
    System.out.println("Le nombre de véhicules pour chaque marque :");
    while (resultSet.next()) {
        String nomMarque = resultSet.getString( columnLabel: "nomMarque");
        int nbVehicules = resultSet.getInt( columnLabel: "nbVehicules");
        System.out.println("-----------------------------\n\tNom de la marque : "+ nomMarque + "\n\tNombre de véhicules : " + nbVehicules);
    }
    System.out.println("-----------------------------");
    connexion().close();
}
```

Fonction de la requête n°5

Toutes requêtes = même forme :

1) On appelle la connexion
2) On récupère les résultats de la requête SQL dans une variable de type ResultSet
3) On parcours la variable pour afficher ce que l'on souhaite

Tests

Résultat de l'exécution de la fonction précédente :

```
Le nombre de véhicules pour chaque marque :
----------------------------
    Nom de la marque : Ford
    Nombre de véhicules : 2
----------------------------
    Nom de la marque : Mercedes
    Nombre de véhicules : 1
----------------------------
    Nom de la marque : Peugeot
    Nombre de véhicules : 1
----------------------------
    Nom de la marque : BMW
    Nombre de véhicules : 1
----------------------------
    Nom de la marque : Volkswagen
    Nombre de véhicules : 1
----------------------------
```

FIN