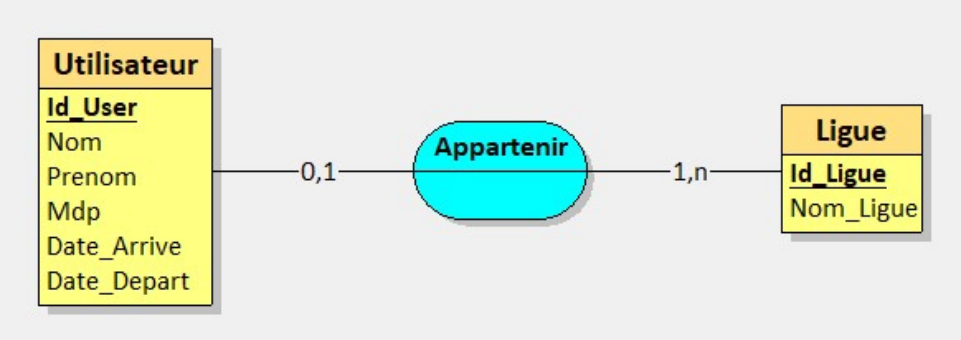
Documentation Du projet Personnel

Sommaire :

Base de données

MCD



Un utilisateur appartient à une seule ligue, sauf s'il est Root, dès lors il n’appartient à aucune ligue.

Une ligue est composée de plusieurs utilisateurs, et il est possible de gérer les utilisateurs, ainsi que leurs données.

Script :

```sql
DROP TABLE IF EXISTS employe;
DROP TABLE IF EXISTS ligue;


CREATE TABLE ligue(
    Id_Ligue INT NOT NULL AUTO_INCREMENT,
    Nom_Ligue VARCHAR(40),
    PRIMARY KEY (Id_Ligue)
)ENGINE = INNODB;


CREATE TABLE employe(
    User_Id INT NOT NULL AUTO_INCREMENT,
    Id_Ligue INT,
    Nom VARCHAR(25),
    Prenom VARCHAR(25),
    Mdp VARCHAR (50),
    Date_Arrivee DATE,
    Date_Depart DATE,
    Mail VARCHAR(25),
    PRIMARY KEY (User_Id),
    FOREIGN KEY (Id_Ligue) REFERENCES Ligue (Id_Ligue)
)ENGINE = INNODB;
```

Les id d'employe et ligue sont auto-incrémenté.

```sql
ALTER TABLE employe
DROP CONSTRAINT FK_EMP_LIG;
ALTER TABLE employe  ADD CONSTRAINT FK_EMP_LIG FOREIGN KEY (Id_Ligue) REFERENCES ligue (Id_Ligue);
```

Contrainte assurant le lien entre employe et ligue.

Code Java

```java
public class Employe implements Serializable, Comparable<Employe>
{
    private static final long serialVersionUID = 4795721718037994734L;
    private String nom, prenom, password, mail;
    private Ligue ligue;
    private GestionPersonnel gestionPersonnel;
    private LocalDate dateArrivee = LocalDate.of(0000, 01, 01);
    private LocalDate dateDepart = LocalDate.of(0000, 01, 01);
    private int id;
```

Constructeur de la classe employe dans laquelle nous avons rajouté les dates d'arrivée et de départ ,au format compris par MYSQL, d'un employe.

```java
//Getter de DateArrivee
public LocalDate getDateArrivee() {
    return dateArrivee;
}
//Getter de DateDepart
public LocalDate getDateDepart() {
    return dateDepart;
}

//Setter pour datearrivee
public void setDateArrivee(LocalDate dateArrivee) throws ExceptionArrivee{
        if( (dateDepart != null) && (dateArrivee.isBefore(dateDepart) ) )
        {
            throw new ExceptionArrivee();
        }
        this.dateArrivee = dateArrivee;

}

//Setter pour datedepart
public void setDateDepart(LocalDate dateDepart) throws ExceptionDepart {
    if( (dateArrivee != null) && (dateDepart.isAfter(dateArrivee) ) )
    {
        throw new ExceptionDepart();
    }
    this.dateDepart = dateDepart;
}
```

Ajout des setter et getter pour les dates ainsi que gestion des exceptions, s'activant lors d'une saisie non conforme au cahier des charges ou simplement impossible.

```java
package personnel;

public class ExceptionArrivee extends Exception {
        public ExceptionArrivee()
        {
            System.out.println("Exception ExceptionArrivee has been raised...");
        }
        @Override
        public String toString()
        {
          return "La date d'arrivée ne peut pas etre avant la date de départ ";
        }


}
```

Class exception pour la date arrivée

```java
package personnel;

public class ExceptionDepart extends Exception{
        public ExceptionDepart()
        {
            System.out.println("Exception ExceptionDepart has been raised...");
        }
        @Override
        public String toString()
        {
          return "La date de départ ne peut pas etre avant la date d'arrivée ";
        }


}
```

Class exception pour la date départ

```java
Option editerEmploye(Employe employe)
{
        Menu menu = new Menu("Gérer le compte " + employe.getNom(), "c");
        menu.add(afficher(employe));
        /*menu.add(changerNom(employe));
        menu.add(changerPrenom(employe));
        menu.add(changerMail(employe));
        menu.add(changerPassword(employe)); */

        //ajout des options pour modifier et supprimer un employé
        menu.add(modifierEmploye(employe));
        menu.add(GererLesDates(employe));
        menu.add(supprimerEmploye(employe));
        menu.addBack("q");
        return menu;
}
```

Ajout de la gestion des dates dans le dialogue en ligne de commande.

```java
//Modification de la methode changerAdministrateur pour qu'elle change l'administrateur d'une ligue
private List<Employe> changerAdministrateur(final Ligue ligue)
{
    return new List<>("Changer l'administrateur", "c",
            () -> new ArrayList<>(ligue.getEmployes()),
            (index, element) -> {ligue.setAdministrateur(element);}
            );
}
```

Option permettant de changer l’administrateur d’une ligue

```java
package jdbc;

public class Credentials

{
    private static String driver = "mysql";
    private static String driverClassName = "com.mysql.cj.jdbc.Driver";
    private static String host = "localhost";
    private static String port = "3306";
    private static String database = "personnel";
    private static String user = "superuser";
    private static String password = "root";

    static String getUrl()
    {
        return "jdbc:" + driver + "://" + host + ":" + port + "/" + database +"?zeroDateTimeBehavior=CONVERT_TO_NULL&serverTimezone=UTC";

    }

    static String getDriverClassName()
    {
        return driverClassName;
    }

    static String getUser()
    {
        return user;
    }

    static String getPassword()
    {
        return password;
    }
}
```

Class Credentials contenant les informations de connexion à la Base de données local (sur MySQL).

```java
package jdbc;

import java.sql.DriverManager;

public class JDBC implements Passerelle
{
    Connection connection;

    public JDBC()
    {
        try
        {
            Class.forName(Credentials.getDriverClassName());
            connection = DriverManager.getConnection(Credentials.getUrl(), Credentials.getUser(), Credentials.getPassword());
        }
        catch (ClassNotFoundException e)
        {
            System.out.println("Pilote JDBC non installé.");
        }
        catch (SQLException e)
        {
            System.out.println(e);
        }
    }
```

Fonction se trouvant dans JDBC et initiant la connexion à la base de données

```java
@Override
public int insert(Ligue ligue) throws SauvegardeImpossible
{
    try
    {
        PreparedStatement instruction;
        instruction = connection.prepareStatement("insert into ligue (Nom_Ligue) values(?)", Statement.RETURN_GENERATED_KEYS);
        instruction.setString(1, ligue.getNom());
        instruction.executeUpdate();
        ResultSet id = instruction.getGeneratedKeys();
        id.next();
        return id.getInt(1);
    }
    catch (SQLException exception)
    {
        exception.printStackTrace();
        throw new SauvegardeImpossible(exception);
    }
}
```

Fonction insert ligue, récupérant le résultat de la requête SQL effectuée, sur l'application, afin d'insérer la ligue créée sur l'application, dans la base de données.

```java
@Override
public void deleteLigue(Ligue ligue) throws SauvegardeImpossible
{
    try
    {
        System.out.println("Ca marche");
        PreparedStatement instruction;
        instruction = connection.prepareStatement("delete from ligue where Id_Ligue = ?", Statement.RETURN_GENERATED_KEYS);
        instruction.setInt(1, ligue.getID());
        instruction.executeUpdate();
    }
    catch (SQLException exception)
    {
        exception.printStackTrace();
        throw new SauvegardeImpossible(exception);
    }
}
@Override
public void deleteEmploye(Employe employe) throws SauvegardeImpossible
{
    try
    {
        System.out.println("Ca marche taggle");
        PreparedStatement instruction;
        instruction = connection.prepareStatement("delete from employe where User_Id = ?", Statement.RETURN_GENERATED_KEYS);
        instruction.setInt(1, employe.getID());
        instruction.executeUpdate();
    }
    catch (SQLException exception)
    {
        exception.printStackTrace();
        throw new SauvegardeImpossible(exception);
    }
}
```

Fonctions de suppression d'une ligue et d'un employé, lors de la suppression sur la console, les éléments seront également supprimés sur MySQL.

```java
@Override
public int insert(Employe employe) throws SauvegardeImpossible
{
    try {
        Date dateArriveeSQL = Date.valueOf(employe.getDateArrivee());
        Date dateDepartSQL = Date.valueOf(employe.getDateDepart());
        PreparedStatement instruction;
        instruction = connection.prepareStatement("insert into employe (Nom, Prenom, Mdp, Date_Arrivee, Date_Depart, Mail) values(?, ?, ?, ?, ?, ?)"
        instruction.setString(1, employe.getNom());
        instruction.setString(2, employe.getPrenom());
        instruction.setString(3, employe.getPassword());
        instruction.setDate(4, dateArriveeSQL);
        instruction.setDate(5, dateDepartSQL);
        instruction.setString(6, employe.getMail());
        instruction.executeUpdate();
        ResultSet id = instruction.getGeneratedKeys();
        id.next();
        return id.getInt(1);
    }
    catch (SQLException exception)
    {
        exception.printStackTrace();
        throw new SauvegardeImpossible(exception);
    }
}
```

Fonction d'insertion d'un employé dans la base de donnée MySQL, depuis l'application.

```java
@Override
public int update(Employe employe) throws SauvegardeImpossible
{
    try
    {
        PreparedStatement instruction;
        instruction = connection.prepareStatement("update utilisateur set Nom = ?, Prenom =
        instruction.setString(1, employe.getNom());
        instruction.setString(2, employe.getPrenom());
        instruction.setString(3, employe.getPassword());
        instruction.setString(4, employe.getDateArrivee().toString());
        instruction.setString(5, employe.getDateDepart().toString());
        instruction.executeUpdate();
        ResultSet id = instruction.getGeneratedKeys();
        id.next();
        return id.getInt(1);
    }
    catch (SQLException exception)
    {
        exception.printStackTrace();
        throw new SauvegardeImpossible(exception);
    }
}
```

Fonction de modification d’un employé depuis l’application, qui modifiera également la base de données MySQL

Schéma des chemins de la console :