Projet JPA:

Depot GitHub: fait
réperoire conception: fait (à competer)
interrogation BDD -> Swing (fait pour les tests: Classes éxécutables)
Projet de type Maven: fait
Dependencies: ajoutées dans le fichier Pom.xml pour importer les dependances du projet

dependencies pour les informations de liaisons/utilisation avec la BDD: fait
Base de données(BDD): fait(bémol-1: La base de donnée FOOT a été créée avec le code SQL dans la fenetre de xampp. Code SQL: CREATE DATABASE IF NOT EXIST FOOT)
Tables des la BDD: Créées avec le code JPA/JPQL depuis l'interface (VerificationBase. Cette intrerface propose deux champs de texte pour saisir les chemins absolus de la base de donnée dans xampp et le chemin absolu du dossier contenant les fichiers.csv.)
//
Fichiers.csv:

results.csv = liste des rencontres avec les resultats
goalscoers.csv = liste des buteurs
shootouts.csv = complement d'information lorsque les matchs se sont joués avec tir aux buts

Results:
date -> Date date du match
home_team -> varChar nom de l'equipe hote
away_team -> varChar nom de l'equipoe invitée
home_score -> int score de l'equipe hote
away_score -> int score de l'equipe invitée
tournament -> VarChar nom du tournois
city -> Ville dans laquelle le match a eu lieu
country -> Pays dans lequel le match a eu lieu
neutral -> boolean indique si le match sets joue en lieux/terrain neutre

Shootouts:
date -> Date date du match
home_team -> varChar nom de l'equipe hote
away_team -> varChar nom de l'equipe invitée
winner -> varChar nom equipe vainqueur de l'epreuve de tir aux buts
first_shooter -> varChar nom equipe qui a commencée les tirs aux buts

Goalsscorers:
date -> date date du match
home_team -> varChar nom de l'equipe hote
away_team -> varChar nom de l'equipe invitée
team -> varChar nom de l'equipe du buteur
scorer -> varChar nom identité buteur
minute -> int minute a laquelle le but est marqué
own_goal -> boolean but contre son camp
penalty -> boolean tir sur penalty

org.mariadb.jdbc=pas utilisé pour ce projet.(pilotes téléchargés depuis mvn Repository.)
repertoireconception a la racine du projet pour inclure le dossier de conception au format .pdf
Repertoire: src/main/resources -> créer un fichier config.properties (pour la connexion a la BDD mySql)
persistence.xml dans un repertoire META-INF(pour la persistence des données)=OK dans projet
vérifier le nom de la persistence-unit: (ici) config1

//
DIAGRAMMES:

RESULTS: liste de rencontres avec resultats
entete: date, home_team, away_team, home_score, away_score, tournament, city, neutral
La classe a été pensée pour indiquer à JPA que ce sont des entités. Les types de données sont des type Java, car en plus d'être utilisées par JPA(constructeur sans arguments, getters, setters , méthode toString(), dans le procéssus de copie des données en base de données), ces classes permettent l'utilisation pour la gestion des données dans la base de donnée.)

```
@Entity
@Table(name="Results")
public class Results {
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
private int id;
private LocalDate date;
private String homeTeam;
private String awayTeam;
private int homeScore;
private int awayScore;
//**private Tournois tournois;
private String tournois;
private String nomVille;
private String country;
private boolean neutre;
//
```

A noter les annotations qui figurent: @Entity signale une Entité, @Table signale une Table, @id signal un id (identifiant unique de type Integer/int), @ signale la gestion automatique incrémentée de l'id.)

SHOOTOUTS: infos si tirs au buts
entete: date, home_team,away_team, winner, first_shooter

```
@Entity
@Table(name="ShootOut")
public class Shootout {


@Id
```

@GeneratedValue(strategy=GenerationType.***IDENTITY***)

private int id;

private LocalDate date;

private String homeTeam;

private String awayTeam;

private String winner;

private String firstShooter;

A noter les annotations qui figurent: @Entity signale une Entité, @Table signale une Table, @id signal un id (identifiant unique de type Integer/int), @ signale la gestion automatique incrémentée de l'id.)

GOALSCORERS: liste des buteurs -> objets buts, joueurs, equipes,
entete: date, home_team,away_team,team(equipe buteur), scorers(Joueur), minute,own_goal,penalty

@Entity

@Table(name="GoalScorers")

public class Goalscorers {

@Id

@GeneratedValue(strategy=GenerationType.***IDENTITY***)

private int id;

private LocalDate date;

private String homeTeam;

private String awayTeam;

private String team;

//**private Joueur scorer;

private String scorer;

private String minute;

private boolean ownGoal;

private boolean penalty;

A noter les annotations qui figurent: @Entity signale une Entité, @Table signale une Table, @id signal un id (identifiant unique de type Integer/int), @ signale la gestion automatique incrémentée de l'id.)

Entites: Match, Buts, Joueurs, Equipes, Ville, Tournois
Result = liste des rencontres-> Matchs, Equipes, Villes, Tournois
Goalscorers = liste des buts et buteurs -> Buts, Equipes,
Shootouts = infos tirs au but -> Equipe, Match

Tables dans : C://xampp/mysql/data/Nom_Base->Verfier si existe
Result: pour copier le fichier .csv dans une table a créer
date -> date (Localdate)
home_team -> String/varChar
away_team -> String/varChar
home_score -> int

away_score -> int
tounament -> String/ varChar
city -> String
neutre -> boolean

Shootout:
date -> date(Localdate)
home_team -> String/varChar
away_team -> String/varChar
winner -> String/varChar
first_shooter -> String/varChar

Goalscorers:
date -> date(Localdate)
home_team -> String/varChar
away_team -> String/varChar
team -> String/varChar
scorer -> String/varChar
minute -> int
own_goal -> boolean
penalty -> boolean

Match: **
id -> int auto_increment
Date -> Date (format ISO)
nomHome -> varChar (home.nom,away.nom)
nomAway -> varChar
scoreHome -> Integer
scoreAway -> Integer
tournois -> (nom)varChar
ville -> (nom)varChar
pays -> (nom) varChar
neutre -> boolean
** pensée avant étude des données contenues dans les fichiers, mais inutilisée par la suite. JPA permet d'utiliser des données de type java, pour les charger dans les tables SQL.

Equipes:
id -> int
nom -> varChar
Home -> boolean
Away -> boolean
Ville -> Ville.nom, Ville.pays
joueurs -> String/varChar

Joueurs: **
id -> int auto increment
nom -> varChar
prenom -> varChar
equipe -> equipe.nom
** pensée avant étude des données contenues dans les fichiers, mais inutilisée par la suite. JPA permet d'utiliser des données de type java, pour les charger dans les tables SQL. Remplacé par un type String contenant le nom et le prénom du joueur.

Tournois: **
id -> int auto increment
date -> date(ISO)
nom -> varChar
ville -> Ville.nom, Ville.pays, Ville.neutral
participants -> <Equipe>Equipe.nom
** pensée avant étude des données contenues dans les fichiers, mais inutilisée par la suite. JPA permet d'utiliser des données de type java, pour les charger dans les tables SQL. Remplacé par un ype string contenant le nom du tournois.

Buts:
id -> int auto increment
date -> Date(LocalDate format ISO)
homeTeam -> String/varChar
awayTeam-> String/varChar
scorers -> String/varChar
Own_goal -> boolean
Penalty -> boolean

Ville: **
id -> int auto increment
nom -> varChar
country -> varChar
neutral -> boolean
<Equipe> equipe
** pensée avant étude des données contenues dans les fichiers, mais inutilisée par la suite. JPA permet d'utiliser des données de type java, pour les charger dans les tables SQL.

6 entites:processus/relations

Match: **
constructeur()
constructeur(args...)
Getters/Setters
toString()
recherche par id(id)
recherche par date()
recherche par score()
recherche par tournois()
recherche par nom_ville(Ville.nom)
recherche par nom_pays(Ville.nom_pays)
recherche par homeAway(Villehome,Villeaway)
afficherScore(VilleHome,VilleAway)
modifier_match(id)
supprimer_match()id

** pensée avant étude des données contenues dans les fichiers, mais inutilisée par la suite. JPA permet d'utiliser des données de type java, pour les charger dans les tables SQL.

Equipe: **
constructeur()
constructeur(args...)
Getters/Setters
toString()
recherche par id(id)
rechercher par nom(Equipe.nom)
rechercher par ville(Equipe.Ville.nom)
rechercher par pays(Equipe.Ville.pays)
afficher_joueur(Joueurs.liste)
calculerVictoire(Equipe.nom)
modifier_equipe(id)
supprimer_equipe(id)
** pensée avant étude des données contenues dans les fichiers, mais inutilisée par la suite. JPA permet d'utiliser des données de type java, pour les charger dans les tables SQL. Remplacé par une donnée de type String.

Joueurs: **
constructeur()
constructeur(args..)
Getters//Setters
toString()
recherche par id(id)
recherche par equipe(Equipe.nom)
recherche par equipe(Equipe.nom,date)
recherche par ville(Ville.nom)
recherche par pays(Ville.pays)
modifier(id)
supprimer(id)
** pensée avant étude des données contenues dans les fichiers, mais inutilisée par la suite. JPA permet d'utiliser des données de type java, pour les charger dans les tables SQL. Remplacé par une donnée de type string.

Tournois: **
constructeur()
constructeur(args...)
Getters//Setters
toString()
recherche par id(id)
recherche par date(date)
recherche par nom(Tournois.nom)
recherche par ville(Ville.nom)
modifier(id)
supprimer(id)
** pensée avant étude des données contenues dans les fichiers, mais inutilisée par la suite. JPA permet d'utiliser des données de type java, pour les charger dans les tables SQL. Remplacé par une donnée de type string.

Ville: **
constructeur()
constructeur(args...)

Getters//Setters
toString()
recherche par id(id)
recheche par nom(Ville.nom)
recherche par pays(Ville.pays)
recherche par neutralite(Ville.neutral)
modifier(id)
supprimer(id)
** pensée avant étude des données contenues dans les fichiers, mais inutilisée par la suite. JPA permet d'utiliser des données de type java, pour les charger dans les tables SQL. Remplacé par une donnée de type string.

Buts: Utilisé par le type Goalscorers
constructeur()
constructeur()
Getters//Setters
toString()
recherche par id(int id)
recherche par Date(Date date)
recherche par Statut(Ville.home, Ville.Away)
recherche par Equipe(Equipe.nom)
recherche par butteur(Joueur.nom,Joueur.prenom,Joueur.equipe.nom)
recherche de but(Date date, But.Own_goal/But.Penalty
modifier(id)
supprimer(id)

Faire le diagramme des classes, attributs et fonctions membres()
Faire le diagramme des entites-relations: cardinalités, liens entre les entites-> cle primaires et cle secondaires.

<u>Cardinalités:</u>
1 match se deroule dans N tournois
N match composent 1 Tournois
1 match se deroule dans 1 Ville
1 Ville accueille N match
1 match est composé de N(2) Equipes
1 Equipe joue N Match
1 Match produis 0 a N Buts
1 equipe marque de 1 à N Buts
1 Equipe est composé de N Joueurs (11+remplacants)
1 Joueur marque de 0 a N Buts
1 Ville possède de 1 à N Equipes(ex: Barcelone 2 equipes, ou Equipe Homme,Equipe femmes)
1 Tournois engage N equipes
1 tournois est composé de N match dans N Villes
1 match donné utilise N Equipe
1 Match comprend de 0 à N Buts
1 But est marqué par 1 Joueur
1 Ville

<u>Requetes demandées:</u>

Afficher les N meilleurs butteurs de tous les temps -> recherche et tri sur l'attribut scorers de la table But et calcul du nombre de buts marqués.

Afficher les N meilleurs butteurs d'une competition donnée -> rechercheParNom dans la table Tournois pour obtenir le nom des differentes equipes et calculer le nombre de Buts de chaque Joueurs.

Afficher les meilleurs butteurs d'une equipe donnée -> rechercheParNom dans la table But pour calculer le nombre de buts des Joueurs de l'equipe.

Afficher les equipes qui ont gagnés le plus de match en % -> calculer le nb de match et le nb de victoires(home et away)

Afficher les match entre 2 equipes données et afficher le % de victoires des 2 Equipes

Persistence des données: persistense.xml, dans un projet Maven

```
<persistence-unit name="Nom de la base de donnée" transaction-type="RESOURCE-LOCAL">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider<:provider>
        <properties>
                <property name = "jakarta.persitence.jdbc.url" value =
        "jdbc:mysql://localhost:numero port/nom base">
                </property>
                <property>
                        <property name = "jakarta.persitence.jdbc.user" value ="root"/>
                </property>
                <property>
                        <property name = "jakarta.persitence.jdbc.password" value =""/>
                </property>
        </properties>
```

Création d'un EntityManger:

```
EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory(config1);
EntityManager em = entityManagerFactory.createEntityManager();
EntityTransaction transaction = em.getTransaction();
transaction.begin();
// Création, Mises à jour, insertions et suppressions ici
transaction.commit();
```

Inserer une donnée: (une entrée)

```
EntityManager em = entityManagerFactory.createEntityManager();
Results RS = new Results(); //utilisation dans la copie d'un fichier .csv, avec un constructeur sa sargument, et chargement des des données dans le sattributs de la classe.
RS.setDate(LocalDate.parse(copie[0].toString()));//LocalDate
RS.setHomeTeam(copie[1].toString());
RS.setAwayTeam(copie[2].toString());
RS.setHomeScore(Integer.parseInt(copie[3].toString()));
```

```
RS.setAwayScore(Integer.parseInt(copie[4].toString()));
RS.setTournois(copie[5].toString());
RS.setNomVille(copie[6].toString());
RS.setCountry(copie[7]);
RS.setNeutre(Boolean.parseBoolean(copie[8].toString()));
em.persist(h); (conservé dans le contexte de persistence)
```

Meme requete avec JPQL: (les types de donnnées ont été conservé pour l'intégrité des données.)

```
Results RS = new Results();
RS.setDate(LocalDate.parse(txtf1.getText().toString()));//LocalDate
RS.setHomeTeam(txtf2.getText().toString());
RS.setAwayTeam(txtf3.getText().toString());
RS.setHomeScore(Integer.parseInt(txtf4.getText().toString()));
RS.setAwayScore(Integer.parseInt(txtf5.getText().toString()));
RS.setTournois(txtf6.getText().toString());
RS.setNomVille(txtf7.getText().toString());
RS.setCountry(txtf8.getText().toString());
RS.setNeutre(Boolean.parseBoolean(txtf9.getText().toString()));
em.persist(RS);
```

Recuperer une instance:

Dans ce cas j'ai utilisé un query , avec une requete SQL et des paramètres nommés(JPQL, pour faire le lien entre JPA/Hibernate et les tables SQL.)

```
Query query = em.createQuery("SELECT rs FROM Results rs WHERE rs.date= :date");
query.setParameter("date",LocalDate.parse(txtf1.getText().toString()));
list = query.getResultList();
Results RS = new Results();
if(list.size()>1) {
FA = new FenetreAffichage();
iter = list.iterator();
while(iter.hasNext()) {
ligne = iter.next().toString();
FA.text.append(ligne+"\n");
}
FA.setVisible(true);
```

(ici , le terme FA est relatif à une instance d'une claqse utilistaire FenetreAffichage qui sert à afficher plusieurs lignes de texte(ou instance d'un objet avec la méthode toString(). Cet element

utilise aussi des JscrollPane horizontaux et verticaux.)

-> revoie null si l'instance n'est pas trouvée.
Recherche pa requete: requete sql
(bemol-2: il est vrai que j'aurais due utiliser des requetes preparées dans des classes indépendantes e réutilisables, ainsi qu'une classe de connéxion pour éviter les répétitions des paramètres des EntityManager et transactions.)
Pour les recherches , modifications et suppréssions, les type de retour des recherches , ont étés pensées selon trois cas: Pas de résultats de recherche, un cas de résultat unique, et le cas avec plusieurs réponses, qui sont affichées dans un composant affichant les différents réponses sur chaque ligne, avec l'identifiant unique (id) en début de ligne. L'identifiant est requis pour les requetes de modification ou suppréssion.(agissant comme un premier filtre.)

Modifier une instance:

```
EntityManagerFactory entityManagerFactory =
Persistence.createEntityManagerFactory("config1");//Foot dans persistence.xm

EntityManager em = entityManagerFactory.createEntityManager();

EntityTransaction transaction = em.getTransaction();

transaction.begin();//ouverture de la transaction

//

Query query = em.createQuery("UPDATE Results rs SET rs.date= :newdate WHERE rs.date=
:olddate");

query.setParameter("newdate",(LocalDate.parse(txtf1.getText())));

query.setParameter("olddate",(LocalDate.parse(RSmodif.getDate().toString())));

query.executeUpdate();

query = em.createQuery("UPDATE Results rs SET rs.homeTeam= :newhomeTeam WHERE
rs.homeTeam= :oldhomeTeam");

query.setParameter("newhomeTeam",(txtf2.getText()));

query.setParameter("oldhomeTeam",RSmodif.getHomeTeam().toString());

query.executeUpdate();

query = em.createQuery("UPDATE Results rs SET rs.awayTeam= :newawayTeam WHERE
rs.awayTeam= :oldawayTeam");

query.setParameter("newawayTeam",(txtf3.getText()));

query.setParameter("oldawayTeam",RSmodif.getAwayTeam().toString());

query.executeUpdate();

query = em.createQuery("UPDATE Results rs SET rs.homeScore= :newhomeScore WHERE
rs.homeScore= :oldhomeScore");

query.setParameter("newhomeScore",(Integer.parseInt(txtf4.getText())));

query.setParameter("oldhomeScore",RSmodif.getHomeScore());

query.executeUpdate();

query = em.createQuery("UPDATE Results rs SET rs.awayScore= :newawayScore WHERE
rs.awayScore= :oldawayScore");
```

```
query.setParameter("newawayScore",(Integer.parseInt(txtf5.getText())));
query.setParameter("oldawayScore",RSmodif.getAwayScore());
query.executeUpdate();
query = em.createQuery("UPDATE Results rs SET rs.tournois= :newtournois WHERE
rs.tournois= :oldtournois");
query.setParameter("newtournois",(txtf6.getText()));
query.setParameter("oldtournois",RSmodif.getTournois().toString());
query.executeUpdate();
query = em.createQuery("UPDATE Results rs SET rs.nomVille= :newnomVille WHERE
rs.nomVille= :oldnomVille");
query.setParameter("newnomVille",(txtf7.getText()));
query.setParameter("oldnomVille",RSmodif.getNomVille());
query.executeUpdate();
query = em.createQuery("UPDATE Results rs SET rs.country= :newcountry WHERE
rs.country= :oldcountry");
query.setParameter("newcountry",(txtf8.getText()));
query.setParameter("oldcountry",RSmodif.getCountry());
query.executeUpdate();
query = em.createQuery("UPDATE Results rs SET rs.neutre= :newneutre WHERE rs.neutre=
:oldneutre");
query.setParameter("newneutre",(Boolean.parseBoolean(txtf9.getText())));
query.setParameter("oldneutre",RSmodif.isNeutre());
query.executeUpdate();
//
transaction.commit();//validation de la transaction
em.close();//fermeture du flux
```

Utilisation de l'identifiant unique(id), afin de cibler une seule instance. Il est a noter que j'ai utilisée une instance de l'objet Result afin de conserver les données de la recheche par identifiant unique qui affiche les attributs dans les champs de l'interface, qui servent à modifier de 0 à plusieurs valeurs d'attributs. La modification remplacera les données de l'objet (mémorisées = oldData))par celles contenues dans les champs (newData).

Supprimer une instance:

```
Query query = em.createQuery("DELETE FROM Results rs WHERE rs.id= :id");
query.setParameter("id",(Integer.parseInt(txtf10.getText().toString())));
query.executeUpdate();
transaction.commit();//validation de la transaction
```