Jeu 421 - Livrable 1

PROJET DEVELOPPEMENT WEB

Courtenay Rebecca & Nguyen Duc Anh
M1 BIG DATA | 2021-2022

Table des matières

1. Base de données

a. Signification des colonnes dans chaque tableau

4 tables dans la base de données :

- Game
- Player
- Game_player
- Game_round

i. Game

Stocker les informations de chaque partie.

id	Identifiant de la partie
num_game	Nom de la partie (e.g : PartieDeJoueurX)
num_charge	Nombre de tour de la phase charge
num_decharge	Nombre de tour de la phase décharge

ii. Player

Informations personnelles du joueur et ses statistiques de jeu.

id	Identifiant du joueur
username	Le pseudo (unique) du joueur
password	Mot de passe du joueur
age	Age du joueur
sexe	Sexe du joueur
ville	Ville du joueur
nb_game	Nombre total de parties jouées par le joueur
nb_win	Nombre total de fois que le joueur a gagné
mean_win	Nombre de victoires moyennes du joueur : nombre de victoire/nombre total de partie jouées
mean_jet_charge	Nombre moyen de jetons en fin de phase de charge : nombre total de jetons de fin de phase de charge/nombre total de parties jouées
mean_jet_decharge	Nombre moyen de jetons en fin de phase de décharge : nombre total de jetons de fin de phase de décharge/nombre total de parties jouées

iii. Game_player

Les parties auxquelles le joueur a participé et les statistiques de chaque joueur de la partie.

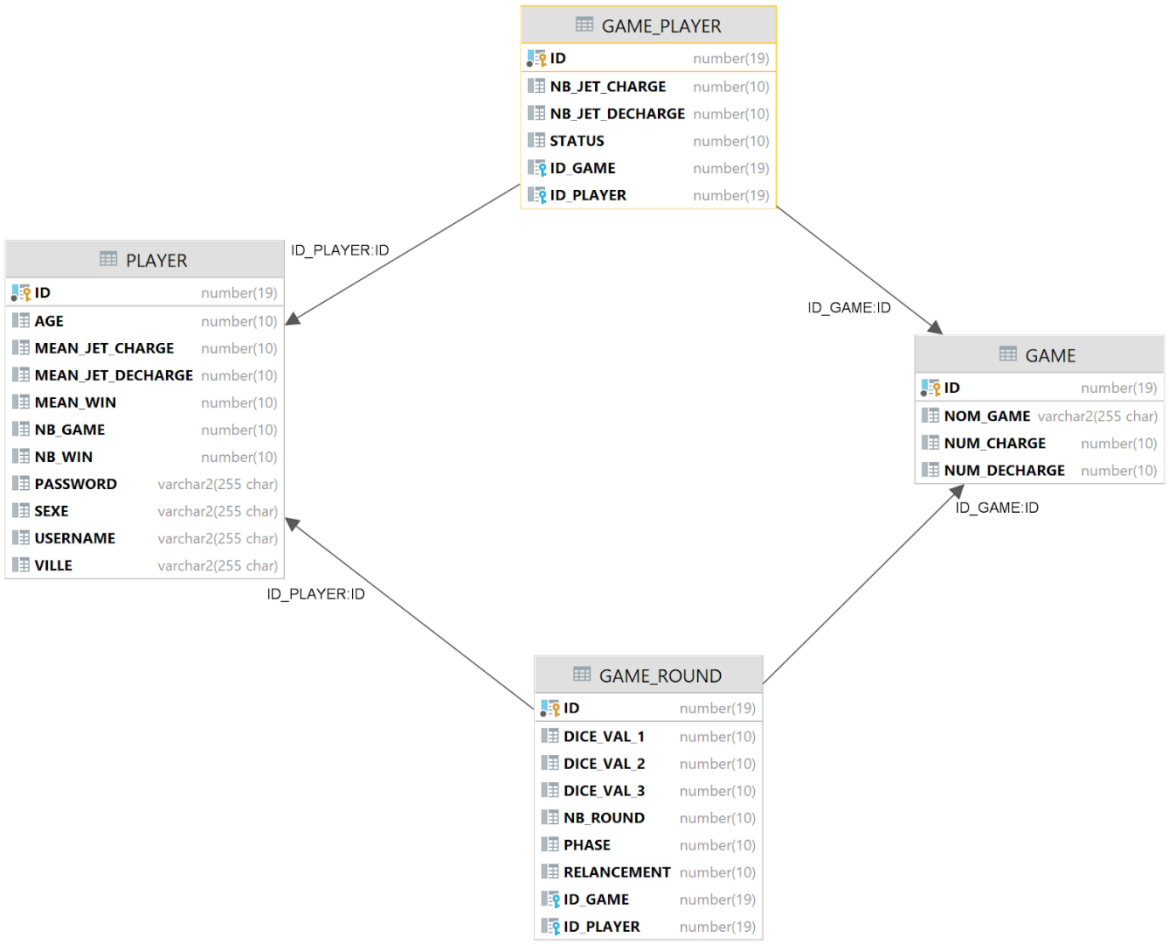
id	Identifiant du joueur de la partie
id_player	Identifiant du joueur – clé étrangère [lien avec la table ***player***]
id_game	Identifiant de la partie – clé étrangère [lien avec la table ***game***]
nb_jet_charge	Nombre de jetons du joueur à la fin de la phase de charge
nb_jet_decharge	Nombre de jetons du joueur à la fin de la phase de décharge
status	0 : perdu – 1 : gagné

iv. Game_round

Informations de charque tour (round) d'une partie pour chaque joueur.

id	Identifiant du tour
id_player	Identifiant de la partie – clé étrangère [lien avec la table ***player***]
id_game	Identifiant de la partie – clé étrangère (lien avec la table ***game***]
dice_val_1	Valeur du premier dé
dice_val_2	Valeur du deuxième dé
dice_val_3	Valeur du troisième dé
relancement	Nombre de fois qu'il y a un relancement
nb_round	Numéro du tour
phase	0 : charge – 1 : décharge

b. UML Diagramme



c. Commandes SQL

```
create table GAME (
    ID           NUMBER(19) not null primary key,
    NOM_GAME     VARCHAR2(255 char),
    NUM_CHARGE   NUMBER(10),
    NUM_DECHARGE NUMBER(10)
);
```

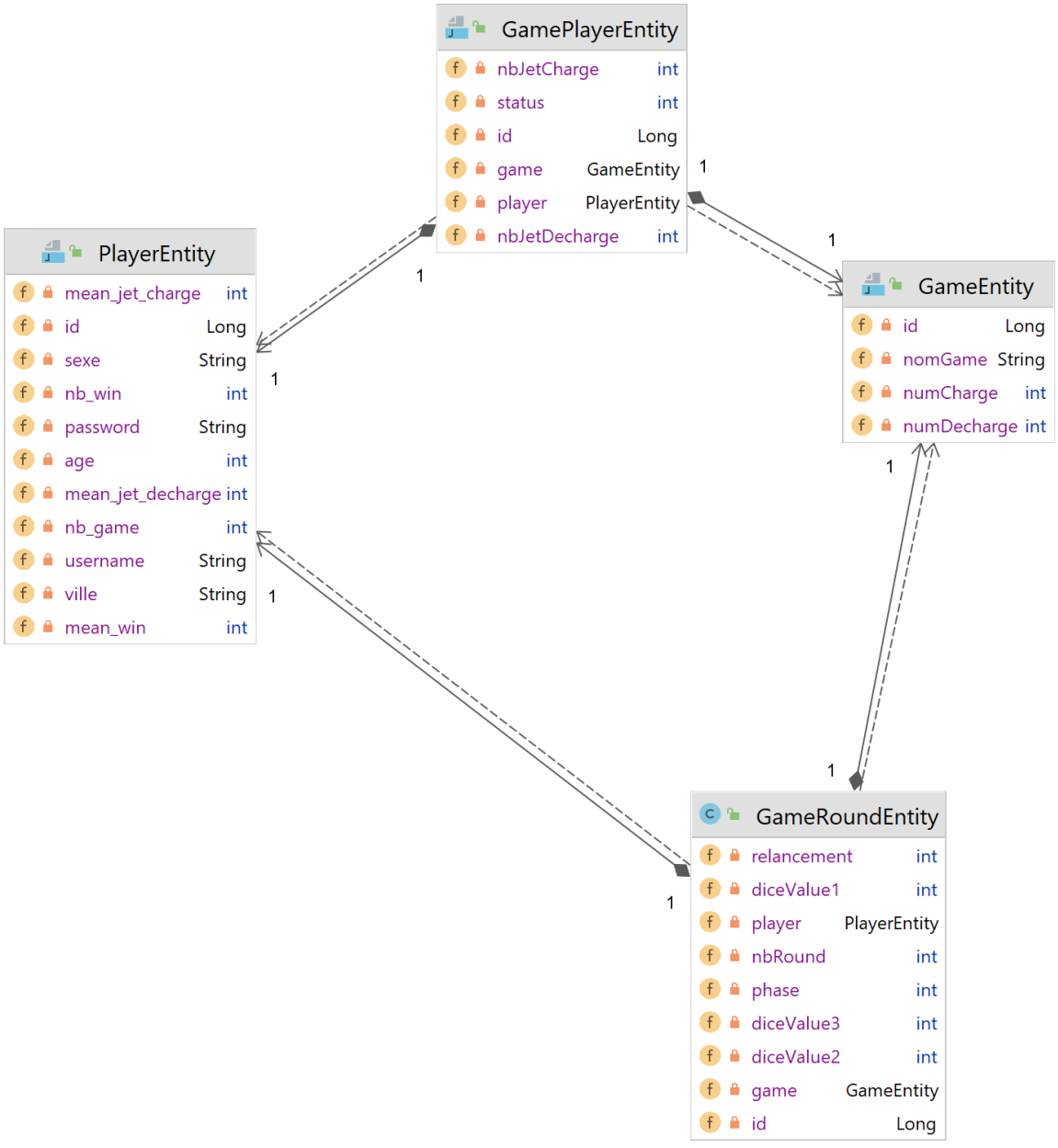
```
create table PLAYER (
    ID                NUMBER(19) not null primary key,
    AGE               NUMBER(10),
    MEAN_JET_CHARGE   NUMBER(10),
    MEAN_JET_DECHARGE NUMBER(10),
    MEAN_WIN          NUMBER(10),
    NB_GAME           NUMBER(10),
    NB_WIN            NUMBER(10),
    PASSWORD          VARCHAR2(255 char),
    SEXE              VARCHAR2(255 char),
    USERNAME          VARCHAR2(255 char),
    VILLE             VARCHAR2(255 char)
);
```

```
create table GAME_PLAYER(
    ID              NUMBER(19) not null primary key,
    NB_JET_CHARGE   NUMBER(10),
    NB_JET_DECHARGE NUMBER(10),
    STATUS          NUMBER(10),
    ID_GAME         NUMBER(19) constraint FKV8HALAQWCYL88W2XUG4YIG9W
references GAME,
    ID_PLAYER       NUMBER(19) constraint FK7QIWTWN8YGWWJRRRHB0OC0EFQ
references PLAYER
);
```

```
create table GAME_ROUND(
    ID         NUMBER(19) not null primary key,
    DICE_VAL_1 NUMBER(10),
    DICE_VAL_2 NUMBER(10),
    DICE_VAL_3 NUMBER(10),
    NB_ROUND   NUMBER(10),
    PHASE      NUMBER(10),
    RELANCEMENT NUMBER(10),
    ID_GAME    NUMBER(19) constraint FK38XTR1JS4A61TX6SDSV97OQG9
references GAME,
    ID_PLAYER  NUMBER(19) constraint FK7W6VVRFV5BRNRWLYFXOYHDXST
references PLAYER
);
```

2. POJO

Le diagramme de classes UML des POJO.



Ici, les attributs « **game** » et « **player** » dans ***GamePlayerEntity*** représente respectivement « **id_game** » et « **id_player** » dans ***game_player.***

De même, les attributs « **game** » et « **player** » dans ***GameRoundEntity*** représente respectivement « **id_game** » et « **id_player** » dans ***game_round.***

3. Description des méthodes

a. Fonction *createPlayer*

Cette fonction :

- Sera appelée lorsqu'un utilisateur créera un compte sur le site web
- Prendra en entrée un objet ***PlayerEntity***
- Encodera le mot de passe de l'utilisateur puis enregistrera les informations de l'utilisateur dans la table ***Player***

```
@Override
public void createPlayer(PlayerEntity player) {
    BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
    String encodedPassword = passwordEncoder.encode(player.getPassword());
    player.setPassword(encodedPassword);
    playerDAO.save(player);
}
```

b. Fonction *updatePlayerInfo*

Cette fonction :

- Sera appelé lorsque l'utilisateur modifiera ses informations personnelles
- Prend en entrée un objet ***PlayerEntity***
- Cherche l'utilisateur dans la base de données ***Player*** grâce au pseudo (**username**) de celui-ci afin de mettre à jour les informations de cet utilisateur, avec les nouvelles informations données, et les enregistrer dans la base de données [l'identifiant et le nom d'utilisateur reste toujours le même]

```
@Override
public void updatePlayerInfo(PlayerEntity player) {
    String username = player.getUsername();
    String password = player.getPassword();
    int age = player.getAge();
    String sexe = player.getSexe();
    String ville = player.getVille();

    PlayerEntity playerToUpdate = playerDAO.findByUsername(username);
    BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
    String encodedPassword = passwordEncoder.encode(password);

    playerToUpdate.setPassword(encodedPassword);
    playerToUpdate.setAge(age);
    playerToUpdate.setSexe(sexe);
    playerToUpdate.setVille(ville);
    playerDAO.save(playerToUpdate);
}
```

c. Fonction *updatePlayerStat*

Cette fonction :

- Sera appelé une fois que le joueur aura terminé une partie
- Prend en entrée le pseudo du joueur **username**
- Cherche l'utilisateur dans la base de données grâce à son pseudo **username** afin de récupérer ses statistiques (nombre de jetons à la fin de phase de charge et de décharge) de toutes les parties
- Calcul des informations telles que **meanWin**, **meanJetCharge**, **meanJetDecharge** et les enregistre dans la table ***Player*** selon le pseudo du joueur

```java
@Override
public void updatePlayerStat(String username){
    PlayerEntity playerToUpdate = playerDAO.findByUsername(username);
    Long playerId = playerToUpdate.getId();
    List<GamePlayerEntity> gamePlayerEntity = gamePlayerDAO.findByPlayer(playerId);
    int nbGame = gamePlayerEntity.size();
    int nbWin = 0;
    int totalNbJetCharge = 0;
    int totalNbJetDecharge = 0;
    for (GamePlayerEntity playerEntity : gamePlayerEntity) {
        int status = playerEntity.getStatus();
        int nbJetCharge = playerEntity.getNbJetCharge();
        int nbJetDecharge = playerEntity.getNbJetDecharge();
        if (status == 1) {
            nbWin += 1;
        }
        totalNbJetCharge += nbJetCharge;
        totalNbJetDecharge += nbJetDecharge;
    }
    int meanWin = nbWin/nbGame;
    int meanJetCharge = totalNbJetCharge/nbGame;
    int meanJetDecharge = totalNbJetDecharge/nbGame;
    playerToUpdate.setNbGame(nbGame);
    playerToUpdate.setNbWin(nbWin);
    playerToUpdate.setMeanWin(meanWin);
    playerToUpdate.setMeanJetCharge(meanJetCharge);
    playerToUpdate.setMeanJetDecharge(meanJetDecharge);
    playerDAO.save(playerToUpdate);
}
```

d. Fonction *updateGame*

Cette fonction :

- Sera appelé à la fin de chaque phase
- Prend en entrée le nom de la partie (**nomGame)**, nombre de tour dans la phase (**num)** et id d'une phase (**phase**) (0 : phase de charge & 1 : phase de décharge)
- Trouvera la partie en cours dans la base de données grâce au nom de la partie
- Mettre à jour, après chaque phase, le nombre de tour de cette phase dans la table ***Game***

```
@Override
public void updateGame(String nomGame, int num, int phase) {
    GameEntity gameToUpdate = gameDAO.findByNomGame(nomGame);
    if (phase == 0) {
        gameToUpdate.setNumCharge(num);
    } else {
        gameToUpdate.setNumDecharge(num);
    }
    gameDAO.save(gameToUpdate);
}
```

e. Fonction *updateGamePlayer*

Cette fonction :

- Sera appelée à la fin d'une phase
- Prend en entrée l'objet ***GameEntity, PlayerEntity,*** nombre de jetons que le joueur a après une phase (**nbJet**), id d'une phase (**id**) (0 : phase de charge, 1 : phase de décharge), statuts (**status**) (0 : perdu, 1 : gagné)
- Trouver la ligne qui contient les informations d'un des joueurs de la partie dans la table ***GamePlayer***
- Met à jour, en fonction de l'identifiant de la phase, le nombre de jetons que l'utilisateur obtient dans cette phase
- Met à jour, lorsqu'une partie est terminée, le statut indiquant si le joueur a gagné ou perdu

```
@Override
public void updateGamePlayer(GameEntity game, PlayerEntity player, int nbJet, int phase, int status) {
    GamePlayerEntity gamePlayerToUpdate = gamePlayerDAO.findByGameAndPlayer(game, player);
    if (phase == 0) {
        gamePlayerToUpdate.setNbJetCharge(nbJet);
    } else if (phase == 1) {
        gamePlayerToUpdate.setNbJetDecharge(nbJet);
    } else {
        gamePlayerToUpdate.setStatus(status);
    }

    gamePlayerDAO.save(gamePlayerToUpdate);
}
```

f. Fonction CRUD comme dans un DAO

i. createGameRound

Mettre à jour la table ***GameRound*** à chaque lancement de dé.

```
@Override
public void createGameRound(GameRoundEntity gameRound) { gameRoundDAO.save(gameRound); }
```

ii. createGamePlayer

Met à jour la table ***GamePlayer*** avant le lancement d'une partie.

```
@Override
public void createGamePlayer(GamePlayerEntity gamePlayer) { gamePlayerDAO.save(gamePlayer); }
```

iii. createGame

Met à jour la table ***Game*** avant le lancement d'une partie.

```
@Override
public void createGame(GameEntity game) { gameDAO.save(game); }
```

iv. findByNomGame

Trouver une partie par le nom de la partie.

```
@Repository("gameDAO")
public interface GameDAO extends JpaRepository<GameEntity, Long> {

    GameEntity findByNomGame(String nomGame);

}
```

v. findByPlayer

Trouver des lignes dans la table ***GamePlayer*** grâce à l'objet ***PlayerEntity***.

```
List<GamePlayerEntity> findByPlayer(PlayerEntity player);
```

vi. FindByUsername

Trouver la ligne dans la table Player grâce à pseudo du joueur.

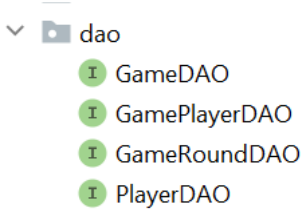
```
@Repository("playerDAO")
public interface PlayerDAO extends JpaRepository<PlayerEntity, Long> {
    PlayerEntity findByUsername(String username);
}
```

2. Files structure

Le programme est construit en utilisant *Spring Framework*.

Les fichiers POJO sont stockés dans le package *com.project.jeu421.entity.*

Les fichiers DAO sont stockés dans le package *com.project.jeu421.dao*.



Les fichiers contenant les méthodes d'accès aux données, qui font autre chose que du CRUD comme dans un DAO, sont situés dans le package *com.project.jeu421.service.*