Institut Paul Lambin

BAC 2 Informatique de gestion

SQL

Super-Heros-Y-Ena-La-Dedans

Auteurs :
Christopher Sacré
Damien Meur

Professeur :
D. Grolaux

12 décembre 2016

Table des matières

1 Introduction

Pour l'unité d'enseignement de Gestion des données et plus précisément pour le cours de SQL, il nous a été demandé de réaliser une application en PostgreSQl et Java. Cette application comprend 2 acteurs principaux : les agents du SHYELD qui disposent d'un application Java pour :

- Disposer des informations des super-héros et d'ajouter ceux-ci en cas d'absence dans la base de données
- Ajouter des repérages de super-héros
- Ajouter des combats et les participations qui y sont liés
- Remarquer la mort d'un super-héros

; ainsi qu'une application centrale, elle aussi programmée en Java pour l'interface utilisateur, elle permet :

- Inscrire des agents
- Supprimer des agents
- Disposer d'informations sur la perte de visibilité de super-héros
- Remarquer la mort d'un super-héros
- Lister des zones de conflits
- Relever l'historique des repérages d'un agent
- Disposer de statistiques de gestions tels que : le classement de victoires/défaites des super-héros ou le classement du nombre de repérages effectués par les agents.

Nous expliquerons dans ce rapport certains de nos choix d'implémentations pour réaliser ce projet.

2 Remarques globales

2.1 Diagramme de structures des données

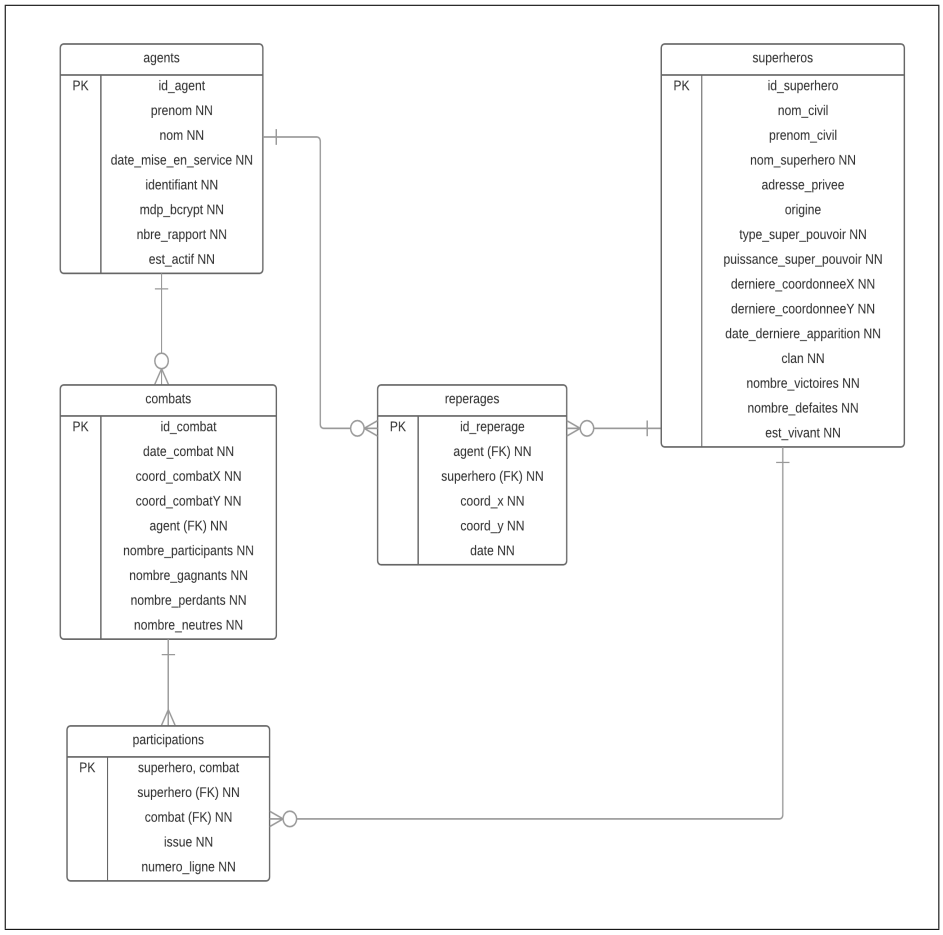

FIGURE 1 – Page d'accueil du bug

2.2 Description du diagramme de structures des données

Nous avons fait le choix de dénormaliser les champs suivants :
- nbre_rapport (table agents)
- nombre_participants (tables combats)
- nombre_gagnants (table combats)
- nombre_perdants (tables combats)
- nombre_neutres (table combats)
- nombre_victoires (table superheros)
- nombre_defaites (table superheros)
- date_derniere_apparition (table superheros)
- derniere_coordonneeX (table superheros)
- derniere_coordonneeY (table superheros)

Encryption Le mot de passe des agents (indispensable pour leur permettre de se connecter à l'application) est encrypté grâce à une clé *BCRYPT*.

Suppression des données Pour éviter d'avoir des participations qui ne soient liées à aucun super-héros et à aucun agent, nous avons utilisé un système de drapeau (*est_vivant* & *est_actif*).

2.3 Vérification des données primaires

Lors de la création des tables, nous avons veillé à effectuer quelques vérifications essentielles (d'autres plus avancées à base de *TRIGGER* seront détaillées dans les sections 3 et4). Pour la table superheros :
- Aucune chaîne de caractère ne doit être vide
- La puissance d'un super pouvoir doit être comprise entre 1 et 10.
- Les coordonnées doivent être comprises entre 1 et 100.
- Le nombre de victoires et défaites doivent être supérieur ou égal à 0.
- La date de dernière apparition doit être inférieure ou égale à la date actuelle du serveur.
- Un clan ne peut être que 'M' (Marvelle) ou 'D' (Décé).

Pour la table agents :
- Les chaînes de caractères ne peuvent pas être vides.
- Les date de mise en service doit être inférieure ou égale à la date du serveur.
- Le nombre de rapports doit être supérieur ou égal à 0.

Pour la table combats :
- La date du combat doit être inférieure ou égale à la date du serveur.
- Les coordonnées doivent être comprises entre 0 et 100.
- Le nombre de participants doit être supérieur ou égal au nombre de perdants et de gagnants (il est possible d'avoir des participations neutres à un combat) et supérieur ou égal à 0.
- Le nombre de gagnants, de perdants et de participations neutres pour un combat doivent être supérieur ou égal à 0.

Pour la table participations :

— L'issue d'un combat vaut 'N' (neutre) par défaut. L'issue d'un combat dépend doit valoir 'N', 'P' (perdant) ou 'G' (gagnant). Un type a été créé à cet effet.

— Le numéro de ligne est plutôt ici à titre informatif et peut être utilisé afin de trier l'affichage d'un combat.

Pour la table repérages

— Les coordonnées doivent être comprises entre 0 et 100.

— La date doit être inférieur (ou égale) à la date du serveur.

2.4 Affichage des données au sein de la console JAVA

Actuellement nous avons d'utiliser la classe DBTablePrinter afin d'afficher les résultats de la plupart de nos requêtes au sein de console. Cela permet notamment d'avoir un affichage clair du résultat. D'autres requêtes ne renvoyant qu'un tuple d'une colonne pour résultat sont quant à elle affichées directement dans la console sans passer par la classe DBTablePrinter (résultat des insertions dans la base de données, ...).

2.5 Gestion des Transactions lors de l'ajout d'un combat

L'ajout d'un combat se fait au sein d'une transaction reprenant le combat en tant que tel ainsi que les participations qui y sont rattachées. Cela permet notamment d'effectuer plusieurs tests afin d'empêcher l'insertion d'un combat invalide.

Listing 1 – Utilisation de transaction au sein de notre application

```
public int ajouterCombat(Combat combat, ArrayList<Participation>
    participations) {
     int id = -1;
     try {
        connexionDb.setAutoCommit(false);
        System.out.println(combat.getAgent());
        try {
           tableStatement.get("ajoutComb").setDate(1,
                combat.getDateCombat());
           tableStatement.get("ajoutComb").setInt(2,
                combat.getCoordCombatX());
           tableStatement.get("ajoutComb").setInt(3,
                combat.getCoordCombatY());
           tableStatement.get("ajoutComb").setInt(4,
                combat.getAgent());
           tableStatement.get("ajoutComb").setInt(5,
                combat.getNombreParticipants());
           tableStatement.get("ajoutComb").setInt(6,
                combat.getNombreGagnants());
```

```
            tableStatement.get("ajoutComb").setInt(7,
                combat.getNombrePerdants());
            tableStatement.get("ajoutComb").setInt(8,
                combat.getNombreNeutres());
            try(ResultSet rs =
                tableStatement.get("ajoutComb").executeQuery()) {
               while(rs.next()) {
                  id = Integer.valueOf(rs.getString(1));
               }
               for(Participation participation : participations){
                  participation.setCombat(id);
                  ajouterParticipation(participation);
               }
            }
            connexionDb.commit();
         } catch (SQLException se) {
            System.out.println("Le combat n'a pas pu etre ajoute");
            if(!participations.isEmpty()){
               System.out.println("Les partcipations ont tout de
                    meme ete rajoutee en tant que reperages");
               for(Participation participation : participations){
                  ajouterReperage(new Reperage(combat.getAgent(),
                      participation.getSuperhero(),
                      combat.getCoordCombatX(),
                      combat.getCoordCombatY(),
                      combat.getDateCombat()));
               }
               id = -2;
            }
         }
      } catch (SQLException se) {
         try {
            connexionDb.rollback();
         } catch (SQLException e) {
            System.out.println("Le retour en arriere n'a pas pu
                etre effectue");
            id = -1;
         }
      } finally {
         try {
            connexionDb.setAutoCommit(true);
            return id;
         } catch (SQLException e) {
            System.out.println("La fermeture de la connexion n'a
                pas pu etre effectue");
            return -1;
         }
      }
   }
```

```
public int ajouterParticipation(Participation participation) {
   try {
      tableStatement.get("ajoutPa").setInt(1,
          participation.getSuperhero());
      tableStatement.get("ajoutPa").setInt(2,
          participation.getCombat());
      tableStatement.get("ajoutPa").setString(3,
          String.valueOf(participation.getIssue()));
      try(ResultSet rs =
          tableStatement.get("ajoutPa").executeQuery()) {
         while(rs.next()) {
           return Integer.valueOf(rs.getString(1));
         }
         return -1;
      }
   } catch (SQLException se) {
      se.printStackTrace();
      System.out.println("La participation n'a pas pu etre
          ajoutee");
      return -1;
   }
}
```

2.6 Utilisation de VIEWS au sein de notre code

Pour certaines opérations ne nécessitant aucune donnée, nous avons préféré utiliser des vues plutôt qu'une procédure à proprement parler, c'est notamment le cas pour :

Listing 2 – Obtenir les informations sur tous les héros n'ayant pas été aperçu depuis plus de dix jours

```
DROP VIEW IF EXISTS shyeld.perte_visibilite;

CREATE VIEW shyeld.perte_visibilite AS
SELECT DISTINCT sh.id_superhero, sh.nom_superhero,
    sh.date_derniere_apparition, sh.derniere_coordonneeX,
    sh.derniere_coordonneeY
FROM shyeld.superheros sh
WHERE (date_part('year', age(sh.date_derniere_apparition)) >= 1
   OR date_part('month', age(sh.date_derniere_apparition)) >= 1
   OR date_part('day', age(sh.date_derniere_apparition)) > 15)
   AND sh.est_vivant = TRUE
ORDER BY sh.nom_superhero;
```

Listing 3 – Lister les zones potentielles de conflit

```
DROP VIEW IF EXISTS shyeld.zone_conflit;
```

```
CREATE VIEW shyeld.zone_conflit AS
SELECT DISTINCT s.derniere_coordonneeX, s.derniere_coordonneeY
FROM shyeld.superheros s, shyeld.superheros s1
WHERE s.clan <> s1.clan
AND ((s.derniere_coordonneeX = s1.derniere_coordonneeX AND
    s.derniere_coordonneeY = s1.derniere_coordonneeY)
   OR (s.derniere_coordonneeX = s1.derniere_coordonneeX + 1 AND
       s.derniere_coordonneeY = s1.derniere_coordonneeY)
   OR (s.derniere_coordonneeX = s1.derniere_coordonneeX - 1 AND
       s.derniere_coordonneeY = s1.derniere_coordonneeY)
   OR (s.derniere_coordonneeX = s1.derniere_coordonneeX AND
       s.derniere_coordonneeY = s1.derniere_coordonneeY + 1)
   OR (s.derniere_coordonneeX = s1.derniere_coordonneeX AND
       s.derniere_coordonneeY = s1.derniere_coordonneeY - 1))
AND EXTRACT(DAY FROM NOW() - s.date_derniere_apparition) <= 10
AND EXTRACT(DAY FROM NOW() - s1.date_derniere_apparition) <= 10
ORDER BY s.derniere_coordonneeX;
```

Listing 4 – Obtenir un classement des héros en fonction de leurs victoires

```
DROP VIEW IF EXISTS shyeld.classementVictoires;

CREATE VIEW shyeld.classementVictoires AS
SELECT s.nombre_victoires, s.id_superhero, s.nom_superhero
FROM shyeld.superheros s
WHERE s.est_vivant='TRUE'
ORDER BY s.nombre_victoires DESC;
```

Listing 5 – Obtenir un classement des héros en fonction de leurs défaites

```
DROP VIEW IF EXISTS shyeld.classementDefaites;

CREATE VIEW shyeld.classementDefaites AS
SELECT s.nombre_defaites, s.id_superhero, s.nom_superhero
FROM shyeld.superheros s
WHERE s.est_vivant='TRUE'
ORDER BY s.nombre_defaites DESC;
```

Listing 6 – Obtenir un classement des héros en fonction de leurs repérages

```
CREATE VIEW shyeld.classementReperages AS

SELECT a.nom, a.prenom, count(r.id_reperage) as "reperages"
FROM shyeld.agents a, shyeld.reperages r
WHERE a.id_agent = r.agent
AND a.est_actif = TRUE
GROUP BY a.id_agent
```

```
ORDER BY count(r.id_reperage) DESC;
```

Listing 7 – Afficher le contenu de la table shyeld.agents

```
DROP VIEW IF EXISTS shyeld.affichageAgents;

CREATE VIEW shyeld.affichageAgents AS

SELECT a.*
FROM shyeld.agents a;
```

Listing 8 – Afficher le contenu de la table shyeld.combats

```
DROP VIEW IF EXISTS shyeld.affichageCombats;

CREATE VIEW shyeld.affichageCombats AS

SELECT c.*
FROM shyeld.combats c;
```

Listing 9 – Afficher le contenu de la table shyeld.reperages

```
DROP VIEW IF EXISTS shyeld.affichageReperages;

CREATE VIEW shyeld.affichageReperages AS

SELECT r.*
FROM shyeld.reperages r;
```

3 Remarques application SHYELD

3.1 SQL

3.1.1 Triggers nécessaires à la dénormalisation

Etant donné que certains champs sont dénormalisés (voir section 2.2), nous avons utilisés des triggers pour les tenir à jour.

La vérification des données d'un combat se fait par exemple via la fonction *shyeld.verificationAuthenticiteCombat()*. Le déclenchement de ce trigger nous a posé soucis car les combats et les participations sont ajoutés au sein d'une transaction. Nous avons dès lors ajouté l'option "DEFERRABLE INITIALLY DEFERRED" dans la déclaration de notre trigger. Il est à noter que le nombre de rapports de l'agent est incrémenté directement depuis la fonction de création de combat.

Listing 10 – Trigger de vérification d'un combat

```
CREATE OR REPLACE FUNCTION
    shyeld.verificationAuthenticiteCombat() RETURNS TRIGGER AS $$
DECLARE
   _participation RECORD;
   _superhero RECORD;
   _compteurMarvelle integer := 0;
   _compteurDece integer := 0;

BEGIN
   IF NOT EXISTS (SELECT * FROM shyeld.participations p WHERE
       p.combat = NEW.id_combat) THEN RAISE foreign_key_violation;
   END IF;
   FOR _participation IN SELECT p.* FROM shyeld.participations p
       WHERE p.combat = NEW.id_combat LOOP
      IF 'M' = (SELECT s.clan FROM shyeld.superheros s WHERE
          s.id_superhero = _participation.superhero) THEN
         _compteurMarvelle := _compteurMarvelle + 1;
      ELSE
         _compteurDece := _compteurDece +1;
      END IF;
      UPDATE shyeld.combats SET nombre_participants =
          nombre_participants + 1 WHERE id_combat =
          _participation.combat;
      IF 'G' = _participation.issue THEN
         UPDATE shyeld.combats SET nombre_gagnants =
             nombre_gagnants + 1 WHERE id_combat =
             _participation.combat;
      ELSIF 'P' = _participation.issue THEN
         UPDATE shyeld.combats SET nombre_perdants =
             nombre_perdants + 1 WHERE id_combat =
             _participation.combat;
      ELSE
         UPDATE shyeld.combats SET nombre_neutres = nombre_neutres
             + 1 WHERE id_combat = _participation.combat;
      END IF;
   END LOOP;
   IF (_compteurDece <= 0 OR _compteurMarvelle <= 0) THEN
      RAISE 'Deux clans adverses sont requis pour le meme combat';
   END IF;
   RETURN NEW;

END;
$$  LANGUAGE plpgsql;

CREATE CONSTRAINT TRIGGER trigger_authenticiteCombat AFTER INSERT
    ON shyeld.combats
DEFERRABLE INITIALLY DEFERRED
```

```
FOR EACH ROW
EXECUTE PROCEDURE shyeld.verificationAuthenticiteCombat();
```

Lors de la création d'un combat, chaque participation entraîne la création automatique d'un repérage. Si le combat n'est pas valide (il n'y a pas au moins un héros de chaque clan), la transaction sera annulée (ROLLBACK). En cas de combat invalide (inexistance d'au moins deux participations de deux héros de clans adverses), les repérages liés aux transactions sont tout de même effectués.

Listing 11 – Trigger de création automatique de repérage en cas d'insertion d'une participation

```
CREATE TRIGGER trigger_reperageCombat AFTER INSERT ON
    shyeld.participations
FOR EACH ROW
EXECUTE PROCEDURE shyeld.reperageCombat();

--- MAJ champ date apparition et dernieres coordonnees de la
    table shyeld.superheros ---
CREATE OR REPLACE FUNCTION
    shyeld.miseAJourDateCoordonneeReperage() RETURNS TRIGGER AS $$
BEGIN
   IF NOT EXISTS (SELECT s.* FROM shyeld.superheros s WHERE
       s.id_superhero = NEW.superhero) THEN
     RAISE foreign_key_violation;
   END IF;

   UPDATE shyeld.superheros SET date_derniere_apparition =
       NEW.date, derniere_coordonneeX = NEW.coord_x,
    derniere_coordonneeY = NEW.coord_y WHERE id_superhero =
        NEW.superhero;
     RETURN NEW;

    EXCEPTION
      WHEN check_violation THEN RAISE EXCEPTION 'trigger date -
          coord superhero incorrect';
END;
$$LANGUAGE plpgsql;



CREATE TRIGGER trigger_reperage_coordonnee AFTER INSERT ON
    shyeld.reperages
FOR EACH ROW
EXECUTE PROCEDURE shyeld.miseAJourDateCoordonneeReperage();
```

3.1.2 Droits d'accès aux données

Etant donné que l'application SHYELD est l'application centrale, elle sera exécutée grâce à l'utilisateur POSTGRESQL possédant la base de donnée, aucune restriction d'accès n'est effectuée pour cette partie de l'application.

3.2 Java

4 Remarques application Agent

4.0.1 SQL

4.0.2 Connexion Agent

Etant donné que les mots de passes sont salés, nous faisons la vérfication de ceux-ci en Java. La fonction s shyeld.check_connexion(varchar(255)) renvoie le mot de passe haché en BCRYPT pour faire la vérification au sein de l'applicaiton java (elle renvoie NULL en cas où 'identifiant ne correspond à aucun utilisateur.

4.0.3 Droits d'accès aux données

L'agent n'est pas considéré comme un utilisateur administrateur, nous avons donc minimisé ces droits d'accès à la base de données au minium. A noter que l'utilisateur administrateur est représenté ici par *dmeur15* et l'utisateur agent par *csacre15*.

Listing 12 – Droits d'accès à la base de données pour un agent

```
GRANT CONNECT
ON DATABASE dbdmeur15
TO csacre15;

GRANT USAGE
ON SCHEMA shyeld
TO csacre15;

GRANT SELECT ON
shyeld.combats,
shyeld.participations,
shyeld.reperages,
shyeld.superheros,
shyeld.agents
TO csacre15;

GRANT TRIGGER ON
shyeld.combats,
shyeld.participations,
shyeld.superheros,
shyeld.reperages
TO csacre15;
```

```
GRANT INSERT ON
shyeld.reperages,
shyeld.combats,
shyeld.participations,
shyeld.superheros
TO csacre15;

GRANT UPDATE ON
shyeld.superheros,
shyeld.combats
TO csacre15;

GRANT EXECUTE ON FUNCTION
shyeld.rechercherSuperHerosParNomSuperHero(VARCHAR),
shyeld.creation_superhero(varchar, varchar, varchar, varchar,
    varchar, varchar, integer, integer, integer,
    timestamp,varchar , integer, integer, boolean),
shyeld.creation_reperage(integer, integer, integer, integer,
    timestamp),
shyeld.creation_combat(timestamp, integer, integer, integer,
    integer, integer, integer, integer),
shyeld.creation_participation(integer, integer, varchar),
shyeld.supprimerSuperHeros(INTEGER),
shyeld.rechercherSuperHerosParNomSuperHero(varchar),
shyeld.verificationAuthenticiteCombat(),
shyeld.miseAJourDateCoordonneeReperage(),
shyeld.reperageCombat(),
shyeld.miseAJourNombreVictoiresDefaites()
TO csacre15;

GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA shyeld TO
    csacre15;
```

4.1 Java