Projet : Annuaire de spectacles

1. Introduction

Le projet consiste à reproduire une partie d'un annuaire des spectacles parisiens.

On admet que l'on possède une liste de lieux de spectacles. Ces lieux peuvent avoir une ou plujsieurs salles.

Ces salles programment des spectacles pendant une durée déteminée avec des représentations à une date et une heure déteminée. Pour simplifier, les horaires ne changent pas selon les jours :o)

Ces lieux sont aussi géolocalisés. On voudrait pouvoir, en théorie, afficher dans la page un petit plan d'accès à l'adresse de ce lieu. Cela n'est pas demandé ici :o)

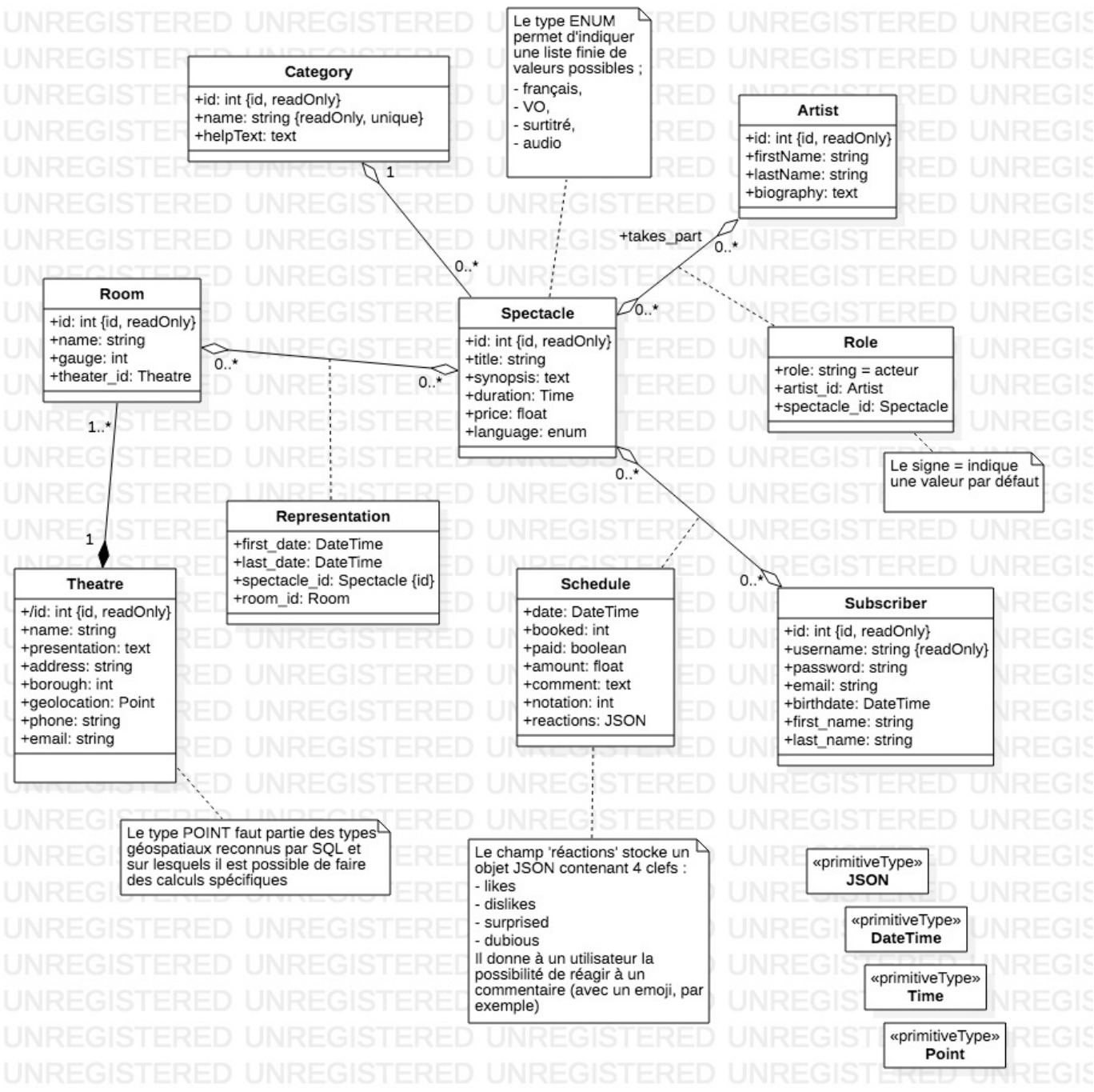
Les spectacles ont un titre ainsi qu'un résumé de l'action et on possède sur eux une liste de parties prenantes (acteurs, metteur en scène, des auteurs, de musiciens,...).

Les spectacles sont classés par catégories ; celles-ci ne sont pas définies *a priori*, mais sont répertoriées dans une table particulière.

En tant qu'utilisateur, je peux m'inscrire sur le site, ce qui me permettrait de réserver des places et de commenter les spectacles. Je ne peux commenter les spectacles que si je les ai vus. Je peux aussi leur attribuer une note. Par ailleurs il est possible pour quelqu'un de réagir à un commentaire, en choisissant une des options pré-déterminées (`like` (je suis d'accord), `dislike` (je ne suis pas d'accord), `surprised` (je suis surpris), `dubious` (je suis dubitatif)).

Je peux aussi réserver pour un spectacle via le site, à condition qu'il reste des places disponibles pour la représentation qui m'intéresse.

1.1. Schéma de la base



1.2. Précisions sur le schéma

- Dans la table `spectacle`, la colonne `language` indique en quelle langue est le spectacle. Soit il est en français, soit s'il est dans une autre langue, on indique quel est le mode de traduction. La valeur est énumérée (colonne de type ENUM) ;

- Dans la table `schedule` (réservations), la colonne `paid` indique si les places ont réellement été payées ou sont juste dans le panier ; `amount` devrait être calculé en fonction du prix (`price`) et du nombre de places réservées (`booked`) ;

- Dans la table `schedule`, la `notation` attribuée à un spectacle est un entier entre 0 et 5 ; les `reactions` sont une possibilité pour une personne quelconque de réagir au

commentaire d'un spectateur, comme les emojis sur Discord, par exemple ;

- Dans la table `role`, les rôles ne sont pas considérés comme énumérés, mais une liste normalisée devrait malgré tout être établie pour éviter les erreurs (SQL est conçu pour être optimal lorsque la cohérence des données est assurée) ;
- Dans la table `theatre`, la localisation du théâtre est représentée sous forme d'un type géospatial SQL : **Point** ; donner une valeur à cette colonne se fait simplement sous la forme `Point(<longitude>, <latitude>)`. De nombreuses fonctions sont disponibles en SQL pour ce type de données ; la colonne `borough` représente l'arrondissement, en termes parisiens.

2. Cas d'utilisation

2.1. Création

1. Créer un utilisateur (s'inscrire)
2. Associer un artiste à un spectacle
3. Créer un spectacle
4. Effectuer une réservation pour un spectacle donné

Rappel :

```
INSERT INTO <table>
(<liste, des, champs>)
VALUES
(<liste, des, valeurs)
```

2.2. Mise à jour

1. Modifier le mot de passe d'un utilisateur
2. Ajouter une place à une réservation si le paiement n'a pas encore été effectué et modifier le montant à payer
3. Pour tous les spectacles ayant lieu un jour donné, augmenter le prix de 10% (les réservations déjà payées ne sont pas concernées)

Rappel :

```
UPDATE <table>
SET [<colonne> = <valeur>]+
WHERE <conditions>
```

2.3. Consultation

1. Afficher la liste des lieux de spectacle
2. Afficher les spectacles par arrondissement
3. Afficher les salles par arrondissement (borough)
4. Afficher les spectacles en cours pour une catégorie donnée
5. Afficher le nombre de spectacles par catégorie
6. Afficher les nombre moyen de places réservées (au total) par les personnes inscrites
7. Afficher la distribution statistique des réservations de places (c'est-à-dire le nombre de personnes ayant réservé au total un certain nombre de places)
8. Afficher le taux de remplissage des salles par spectacle, trié par ordre décroissant
9. Faire la liste des artistes qui ont participé avec un artiste donné à au moins deux spectacles différents
10. Afficher les liste de tous les metteurs en scène qui sont travaillé dans un théâtre donné
11. Afficher les trois catégories de spectacles pour lesquelles le plus de places ont été réservées
12. Recommandation : proposer à un personne X des spectacles qu'elle pourrait aimer, proposition basée sur les spectacles vus par les gens qui ont vu des spectacles en commun avec X
13. Afficher la liste des théâtres, triée par la note moyenne obtenue par les spectacles qui s'y sont joués
14. Existe-t-il des artistes ayant tenu au moins trois fonctions différentes (dans différents spectacles) ?
15. Afficher la liste des recettes par spectacle et par ordre décroissant
16. Y a-t-il des spectacles qui ont affiché complet parmi ceux qui ne se jouent plus ?
17. Quels sont les artistes préférés des spectateurs, c'est-à-dire ceux ayant participé aux spectacles les mieux notés en moyenne ?

Rappel (format le plus fréquent) :

```
SELECT <colonnes>
FROM <relation>
INNER JOIN <relation> ON <critère>
-- quelquefois plus simplement NATURAL JOIN <relation>
WHERE <conditions>
-- notamment WHERE <col> IN (<relation>) AS <alias>
GROUP BY <colonnes>
HAVING <conditions>
ORDER BY <critères>
LIMIT <nombre>
```

2.4. Suppression

1. Supprimer une réservation connaissant le spectateur et le spectacle
2. Annuler un spectacle

   Rappel :

   ```
   DELETE FROM <table>
   WHERE <condition>
   ```

2.5. Bonus

Voici quelques requêtes qui peuvent poser des problèmes que nous n'avons pas eu le temps de voir en cours, mais auxquelles pous pouvez réfléchir.

1. **INSERT** : Créer un spectacle avec les dates des représentations et y associer une salle.

   Indice : Ce n'est pas possible avec une seule requête 😄 ; une solution est décrite ici avec une **transaction**, assez proche d'un commit **git** (la transaction n'est valildée que si tout s'est bien passé).

2. **DELETE** : Supprimer un compte utilisateur (subscriber). Quels problèmes cela pose-t-il ?

   Indice : Si l'on supprime un utilisateur, que deviennent les clefs étrangères de la table des réservations ?

3. **UPDATE** : Ajouter une réaction au commentaire d'un utilisateur.

   Indice : Utiliser les fonctions JSON de MySQL

4. **SELECT** : Trouver les trois théâtres les plus proches du point de géolocalisation (48.3 2.21).

Indice : La fonction `ST_Distance_Sphere` est dédiée à ce genre de calcul

5. **SELECT** : Trouver les commentaires qui ont obtenu plus de 5 « likes ».

   Indice : Utiliser la recherche JSON de MySQL

6. **SELECT** : Trouver dans les synopsis des spectacles ceux qui correspondent le mieux à une sélection de mots.

   Indice : Il faut d'abord créer un index `FULLTEXT` sur la colonne `synopsis` avant d'utiliser la clause `MATCH`

3. Contraintes et Rendu

Le sujet du devoir est bien entendu SQL. Il n'est pas demandé d'écrire un application complète qui réaliserait la gestion des données.

La présentation des résultats peut être tout à rudimentaire. En revanche, vous ferez attention à ce que votre code évite les failles de sécurité les plus gorssières (comme les injections SQL) .

Beaucoup de questions étant liées à des références variables — une requête doit pouvoir être paramétrée et non figée, vous privilégierez l'écriture des requêtes sous forme de fonctions PHP, que vous rassemblerez au sein de classes spécifiques.

Vous aurez à rendre ;

1. Une archive contenant le code de votre « application ».
2. Un export de votre base de données

   N.B. La question de la soutenance n'a pas encore été tranchée à ce jour avec Brontis

4. Conseils

1. Repartez de l'architecture vue en cours. Initialiser votre projet avec **Composer** vous facilitera la vie par la suite.

```
composer init
```

2. Si vous voulez engendrer vos propres données factices, vous pouvez utiliser la bibliothèque **FakerPHP**. Pour installer une bibliothèque dans votre application, utilisez la sous-commande `require` de Composer :

```
composer require fakerphp/faker
```

3. MySQL peut manipuler des données au format JSON (même si cela n'est pas vraiment conforme à l'« esprit » SQL). Vous trouverez toute les ressources à propos des fonctions JSON dans le chapitre de la documentation MySQL :

- Référence des fonctions JSON

4. Pour répondre à la question (bonus 😄) du devoir, vous pouvez regarder la documentation de MySQL sur les types des données géospatiales. En particulier, il existe une fonction `ST_Distance_Sphere` qui permet de calculer la distance entre deux points géolocalisés.

- Fonctions géométriques pour MySQL

5. Parmi les ressources peu évoquées de SQL, existe aussi la recherche « en plein texte », bien que celle-ci soit moins performante qu'un outil dédié comme **Elasticsearch**. Vous pouvez regarder ce type de recherche dans la documentation.

- Recherche en plein texte avec MySQL

5. Ressources

- Packagist
- FakerPHP

6. Post-scriptum

Je suis ouvert à toutes les questions, les remarques et les suggestions ; n'hésitez donc pas à m'interpeler.

Vous pouvez me joindre directement à mon adresse personnelle :

michel.cadennes@sens-commun.fr