GDWFSCAUBDDEXAIII1A

Ceci est un modèle de copie. N'oubliez pas de renseigner vos prénom/nom, ainsi que le nom et le lien vers le projet.

Vous pouvez bien sûr agrandir les cadres pour répondre aux questions sur la description du projet si nécessaire.

Prénom : Jérémy
Nom : SANANIKONE

ATTENTION ! PENSEZ À RENSEIGNER VOS NOM ET PRÉNOM DANS LE TITRE DE VOS FICHIERS / PROJETS !

Nom du projet : Gestion de la base de données d'un cinéma

Lien Github du projet : https://github.com/Anothays/STUDI---BDD.git

Description du projet

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions. Dans cette rubrique, le jury cherche à voir comment vous procédez : comment vous organisez votre travail, comment vous réalisez concrètement la tâche ou l'opération pas à pas.
   Utiliser un langage professionnel. Employez le « je », car vous parlez en votre nom. Vous pouvez écrire au temps présent.

Cette évaluation m'a permis de faire le point sur mes compétences en conception de bases de données. J'ai opté pour la méthode Merise pour cet exercice, abordée notamment dans cet ouvrage qui m'a beaucoup aidé.

La conception (organisation des idées)

La première étape se concentre autour de l'énoncé. Je commence par prendre connaissance du sujet avec une première lecture. Ensuite, je mets en évidence les informations qui me paraissent importantes à la conception de la base de données. Je le fais parfois au stylo sur une feuille de papier ou bien avec un système d'annotations directement sur le document numérique. Concrètement, j'essaie de repérer les éléments qui seront traduits en entités dans le modèle conceptuel de données, ainsi que les cardinalités entre ces entités. Cette première étape me sert donc de brouillon au MCD.

La conception (les diagrammes)

La deuxième étape est la réalisation du MCD, MLD (modèle logique de données), MPD (modèle physique de données) de la méthode Merise. Grâce au éléments relevés dans la première étape, j'ai distingué les entités suivantes :

- **MovieTheater** qui représente un complexe cinématographique, c'est à dire un établissement de l'enseigne.
- **Room** qui représente une salle de projection. Chaque salle est liée à un établissement.
- **Movie** qui représente représente un film.
- **ProjectionEvent** qui représente une projection.
- **Ticket** qui représente un billet d'entrée. Un billet est acheté/réservé par un utilisateur sur l'application web, ou via l'intermédiaire d'une borne présente dans l'établissement. Les billets ne sont pas nominatifs. Un utilisateur peut donc très bien acheter un billet et faire en sorte qu'il soit utilisé par quelqu'un d'autre.
- **Tariff** qui représente tous les choix tarifaires permis par l'enseigne.
- **User** qui représente un utilisateur. Ce peut être un administrateur, un employé, ou un client.
- **Role** qui liste les différents rôles accordants plus ou moins de privilèges aux utilisateurs de l'application. Cette entité permet ainsi de discriminer les différents acteurs (les clients, employés, administrateurs).

Après avoir établie le MCD, je passe au modèle logique de données dans lequel j'explicite les clés étrangères (en vert dans le document joint).

Enfin, je termine par le modèle physique de données, dans lequel je renseigne les contraintes (PRIMARY KEY, FOREIGN KEY, UNIQUE), ainsi que le typage des données. J'ai laissé les liaisons entre les tables pour des raisons de lisibilité, bien que celles-ci ne soient plus nécessaires à ce stade. Quelques précisions concernant deux contraintes :

- La contrainte UNIQUE dans la table Room permet d'éviter d'avoir plusieurs salles avec le même nom au sein d'un même établissement.
- La contrainte UNIQUE dans la table ProjectionEvent permet d'éviter d'avoir deux séances au même moment dans la même salle.

Ecriture du code SQL

La troisième étape consiste en l'écriture du code SQL qui découle du MPD. J'utilise l'outil en ligne dB Fiddle pour tester mon code https://www.db-fiddle.com/

- Je commence par créer le script de création de la base de donnée avec ses tables et contraintes. La première commande SQL de ce fichier est la suivante : **DROP DATABASE IF EXISTS `Cinema`;** elle me permet de supprimer la base de données si elle existe déjà. Cette commande ne me sert seulement qu'en phase de test (elle m'évite de devoir supprimer à la main la base avant de la recréer). Je prends donc soin de supprimer ou commenter cette commande du script avant de livrer le travail au client.

```
1   # Création de la base de données
2   # DROP DATABASE IF EXISTS `Cinema`;
    ▷ Execute
3   CREATE DATABASE IF NOT EXISTS `Cinema` CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
    ▷ Execute
4   USE `Cinema`;
```

- Je crée les différentes tables définies dans le MPD en prenant soin de préciser le moteur de stockage à utiliser (ici innoDB, qui est le moteur par défaut avec mysql 8). Dans l'exemple ci-dessous, l'attribut options représente un tableaux de valeurs sauvegardées en tableau json. Ces valeurs apportent des précisions sur la séance concernée (standard, IMAX, 3D, 4DX).

```
31  CREATE TABLE `ProjectionEvents` (
32      `id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
33      `id_room` INT NOT NULL,
34      `id_movie` INT NOT NULL,
35      `date` DATETIME NOT NULL,
36      `options` JSON,
37      `language` VARCHAR(30),
38      UNIQUE (`id_room`, `date`)
39  )engine=INNODB;
```

- Je déclare les contraintes de clés étrangères après avoir crée les tables, plutôt que de les créer directement dans celles-ci. Cela me permet de créer les tables dans l'ordre que je souhaite sans générer d'erreur.

```
74  ALTER TABLE `Roles_Users` ADD FOREIGN KEY (`id_user`) REFERENCES `Users`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
75  ALTER TABLE `Roles_Users` ADD FOREIGN KEY (`id_role`) REFERENCES `Roles`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
76  ALTER TABLE `Tickets` ADD FOREIGN KEY (`id_projectionEvent`) REFERENCES `ProjectionEvents`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
77  ALTER TABLE `Tickets` ADD FOREIGN KEY (`id_tariff`) REFERENCES `Tariffs`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
78  ALTER TABLE `Tickets` ADD FOREIGN KEY (`id_user`) REFERENCES `Users`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
79  ALTER TABLE `ProjectionEvents` ADD FOREIGN KEY (`id_room`) REFERENCES `Rooms`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
80  ALTER TABLE `ProjectionEvents` ADD FOREIGN KEY (`id_movie`) REFERENCES `Movies`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
81  ALTER TABLE `ProjectionEvents` ADD CONSTRAINT UNIQUE (`id_room`, `date`);
82  ALTER TABLE `Rooms` ADD FOREIGN KEY (`id_movieTheater`) REFERENCES `MovieTheaters`(`id`) ON DELETE CASCADE ON UPDATE CASCADE;
83  ALTER TABLE `Rooms` ADD CONSTRAINT UNIQUE (`name`, `id_movieTheater`);
```

Les clauses **ON DELETE CASCADE ON UPDATE CASCADE** permettent de faire en sorte qu'une mise à jour / suppression de la clé primaire d'une ligne, déclenche la mise à jour / suppression des lignes correspondantes des tables liées. Par exemple, en modifiant l'id d'une salle de la valeur 1 à 100, la clé étrangère id_room initialement égale à 1 passera à 100 dans toutes les lignes concernées de la table ProjectionEvents.

- Une fois les tables crées, j'insère les fixtures.

```
66  INSERT INTO `Users` (`name`, `lastname`, `email`, `password`, `phoneNumber`) VALUES
67  ('John', 'Doe', 'JohnDoe@JD.fr','$2y$13$yttGHVF.y1IKVYgeqxp3We2AQZtTMCeCrPzEjPQ/8KxrAPv5hICGy', '0123456789'),
68  ('Émile', 'Guerin', 'EmileGuerin@EG.com', '$2y$13$yttGHVF.y1IKVYgeqxp3We2AQZtTMCeCrPzEjPQ/8KxrAPv5hICGy', '0123456789'),
69  ('Suzanne', 'Henry', 'Suzanne@Henry.com', '$2y$13$yttGHVF.y1IKVYgeqxp3We2AQZtTMCeCrPzEjPQ/8KxrAPv5hICGy', '0123456789'),
70  ('Célina', 'Leger', 'Celina@leger.com', '$2y$13$yttGHVF.y1IKVYgeqxp3We2AQZtTMCeCrPzEjPQ/8KxrAPv5hICGy', '0123456789')
71  ;
```

- Une fois les données insérées, il est possible de faire des requêtes pour questionner la base de données. Vous retrouverez dans le fichier **select.sql** des exemples de requêtes SELECT avec des jointures.

```
13  SELECT
14  Tariffs.title AS Tarif,
15  COUNT(Tickets.id) AS `Nombre de billets`,
16  SUM(Tariffs.price) AS TOTAL
17  FROM Tariffs
18  JOIN Tickets ON Tickets.id_tariff = Tariffs.id
19  GROUP BY Tarif;
```

- En ce qui concerne la mise à jour des données, un utilisateur pourra par exemple modifier la salle associée à une projection. Pour cela, il utilisera l'interface utilisateur de l'application. Celle-ci pourra directement interagir avec la base de données grâce à des requêtes préparées, ou alors grâce à l'utilisation d'un ORM (Object-Relational Mapping).

  Voici un exemple de requête UPDATE :

```
3  USE `Cinema`;
4
5  # Change la salle pour les projections initialement prévues en salle 1.
   ▷ Execute
6  UPDATE `ProjectionEvents` SET id_room = 7 WHERE id_room = 1;
```

- L'utilisateur qui aura les privilèges d'un CINEMA_SUPER_ADMIN aussi la possibilité de supprimer des films du catalogue. Une suppression d'un film aura pour conséquences de supprimer les projections associées. Cette suppression en cascade est permise par la clause **ON DELETE CASCADE** dans la déclaration des contraintes expliquées précédemment. Voici une telle requête :

```
1  USE `Cinema`;
2
3  # Supprime un film du catalogue
   ▷ Execute
4  DELETE FROM Movies WHERE title = 'One Piece';
```

Le déploiement en local

Une fois le code SQL écrit, je teste le déploiement en local de la base de données. Pour cela j'utilise le programme mysql 8.0 directement avec le terminal en utilisant la commande suivante : **mysql -u root -p** (le chemin de mysql est renseigné au préalable dans le path de l'ordinateur). J'utilise aussi phpMyAdmin pour visualiser rapidement les résultats, lancé au préalable avec la commande **php -S localhost:8080**. Cette dernière commande me permet de démarrer le serveur web intégré à php 8.

En ce qui concerne la sécurité

En ce qui concerne la sécurité, je m'assure tout d'abord que les mots de passe enregistrés soient bien hashés. Normalement, le hashage doit être effectué par l'application en amont. Comme il s'agit d'un exercice sur les bases de données, j'ai utilisé ici l'algorithme BCRYPT directement sur le web. Je fais aussi en sorte que l'entropie des mots de passe soit forte, selon les recommandations de la CNIL.

Je prévois aussi des commandes Shell pour exporter/ré-importer la base de données en cas de besoin. Ces commandes sont enregistrée dans les fichier **CinemaExport.sh** et **CinemaImport.sh**.

Pour exporter, j'utilise mysqldump en exécutant la commande suivante :
/usr/local/mysql-8.0.34-macos13-x86_64/bin/mysqldump -u root -p Cinema > CinemaExport.sql

Pour importer ce sera cette commande :
/usr/local/mysql-8.0.34-macos13-x86_64/bin/mysql -u root -p Cinema < CinemaExport.sql

Le mot de passe d'accès au SGBD doit être renseigné après avoir lancé l'une des commandes.

Je prévois aussi un script **user.sql** qui hiérarchise les utilisateurs de la base de données selon trois rôles :

- Le CINEMA_SUPER_ADMIN, qui possède tous les droits sur l'entièreté des données. Il peut donc opérer n'importe quelle opération de CRUD (Create, read, update, delete) sur n'importe qu'elle table.
- Le CINEMA_ADMIN, qui peut effectuer des opérations de crud sur les tables ProjectionEvents et Tickets.
- Le CINEMA_USER, qui peut effectuer des opérations de crud sur la table Tickets, et des opérations en lecture seule sur la table ProjectionEvents.

```
1  DROP USER IF EXISTS 'CINEMA_SUPER_ADMIN';
   Execute
2  DROP USER IF EXISTS 'CINEMA_ADMIN';
   Execute
3  DROP USER IF EXISTS 'CINEMA_USER';
4
   Execute
5  CREATE USER IF NOT EXISTS 'CINEMA_SUPER_ADMIN' IDENTIFIED BY 'SuperAdminCinemaPass';
   Execute
6  CREATE USER IF NOT EXISTS 'CINEMA_ADMIN' IDENTIFIED BY 'AdminCinemaPass';
   Execute
7  CREATE USER IF NOT EXISTS 'CINEMA_USER' IDENTIFIED BY 'UserCinemaPass';
8
   Execute
9  GRANT ALL ON `Cinema`.* TO 'CINEMA_SUPER_ADMIN';
   Execute
10 GRANT ALL ON `Cinema`.ProjectionEvents TO 'CINEMA_ADMIN';
   Execute
11 GRANT ALL ON `Cinema`.Tickets TO 'CINEMA_ADMIN';
   Execute
12 GRANT SELECT, INSERT, UPDATE, DELETE ON `Cinema`.Tickets TO 'CINEMA_USER';
   Execute
13 GRANT SELECT ON `Cinema`.ProjectionEvents TO 'CINEMA_USER';
```

Au niveau de l'application web, une première gestion des rôles permettra de distinguer différents types d'utilisateurs. C'est à cela que sert la table d'association **Roles_Users**. Cependant, les rôles que je crée avec le code SQL me permettent de contrôler les privilèges accordés aux personnes qui souhaiteraient manipuler directement la base, avec une interface telle que phpMyAdmin ou en ligne de commande. Ces rôles ajoutent donc une sécurité supplémentaire intervenant directement au niveau des données.

*Remarque : Comme précédemment, les lignes **DROP USER IF EXISTS** ne me servent qu'en phase de tests. Je prends soin de les commenter avant de livrer au client.*

2. Précisez les moyens utilisés. Expliquez tout ce dont vous avez eu besoin pour réaliser vos tâches : langages de programmation, frameworks, outils, logiciels, documentations techniques, etc...

J'ai utilisé les outils suivants :
- L'outils en ligne Excalidraw pour la réalisation de mes diagrammes.
- Visual Studio Code pour l'écriture du code SQL.
- L'outil en ligne dBFiddle pour tester rapidement du code SQL.
- PhpMyAdmin pour me permettre de visualiser rapidement le résultat des opérations effectuées
- Le serveur web intégré à PHP 8 pour me permettre de lancer PhpMyAdmin
- Mysql 8.0 utilisé en ligne de commande grâce au terminal de macOs Monterey : zsh.
- L'algorithme BCRYPT
- Github pour garder la traçabilité de mon travail au travers de différents commits.

3. Contexte. Les noms des organismes, entreprises ou associations, dans lesquels vous avez exercé vos pratiques
NB: Pour le cas des exercices et évaluations demandées sur la plateforme Studi, il s'agit de...Studi.

Cette évaluation d'entrainement a été faite sous la direction de l'organisme STUDI

4. Informations complémentaires (*facultatif*)