EVALUATION D'ENTRAINEMENT

Créer et administrer une base de données

Côme Chevallier
Référence de l'évaluation : GDWFSCAUBDDEXAIII2A

Nom du projet : Gestion de base de données Cinéma
Lien GitHub :

Dans le cadre de ma formation de Web Développeur avec la plateforme en ligne Studi, j'ai souhaité réaliser l'évaluation d'entrainement (Créer et administrer une base de données) afin de me situer au niveau de mes compétences acquises.

Après la prise en compte des consignes et des attendus, j'ai décidé de procéder comme suit :

- Concevoir le modèle conceptuel de données ainsi que le modèle physique de données avec la méthode Merise (https://fr.wikipedia.org/wiki/Merise_(informatique))
- Ecrire l'ensemble des commandes avec le langage SQL et tester la viabilité des différentes lignes de commandes via l'outil en ligne DB Fiddle (https://www.db-fiddle.com/)
- Mettre en place mon serveur local (sur environnement MacOS) afin de procéder à la mise en route de la base données et vérifier le bon fonctionnement de l'ensemble de la procédure (CRUD : Create, Read, Update, Delete).

La conception du modèle conceptuel de données

Avant de commencer à écrire l'ensemble des commandes en langage SQL, il était essentiel de définir correctement les besoins de ce client (responsable d'un complexe de cinémas) et d'établir un modèle conceptuel de données (MCD) viable.
Ainsi, au travers de la FAQ fournie, j'ai pu distinguer 8 entités. Il était évidemment possible d'aller plus loin dans la définition des entités, mais j'ai préféré rester sur la demande initiale du client (quitte à devoir revoir le modèle selon ses différents retours).
Ainsi, il apparaissait évident que les instances suivantes devaient être mises en place :

Pour la partie cinéma :

- une entité complexe (telles les grands noms du cinéma UGC, Pathé Gaumont etc…)
- une entité cinéma (qui correspond au cinéma « physique », bâtiment que l'on peut trouver sur nos territoires)
- une entité salle (où seront diffusées les films)

Pour la partie film :

- une entité film qui comprendra l'ensemble des films pouvant être diffusés
- une entité séance qui comprendra à la fois le film et la salle pour la projection

Pour la partie vente :

- une entité client (personne physique assistant à la séance)
- une entité prix des billet (pour la configuration des billets car notre client souhaite pouvoir mettre à disposition des tarifications particulières)
- une entité ticket (qui permet de lier le client à une séance au travers d'un prix d'achat)

Pour chaque entité il conviendra alors de choisir les éléments essentiels. Après le retour de notre client, nous pourrons, au besoin, ajouter des informations.

Les différentes liaisons entre les entités sont établies au travers du MCD (en pièces jointes au présent document).

Une fois le MCD établi (entité et liaisons), il convenait alors de mettre en place le modèle physique de données qui sera la dernière étape avant l'écriture des lignes de commandes en SQL.
A ce stage de la conception, les clés de liaison (clé étrangère) apparaissent dans les entités ainsi que le tapage des propriétés (voir MPhD en pièces jointes).

L'écriture en langage de programmation

Les modèles de données étant établis, l'écriture en langage SQL peut commencer.
Il convient à ce moment de la conception de procéder par étape (en suivant le CRUD) :

1. CREATE

- création de la base de données (CREATE DATABASE). La commande CHARACTER SET permet de définir une modèle de caractère à utiliser pour la base de données (compatibilité des accents européens et français, smileys etc…). La commande IF NOT EXISTS permet d'éviter de créer une nouvelle base données si celle-ci existe déjà sur le serveur.

Fichier schéma > cinema_studi_schema.sql

```sql
/* ON CREE LA BASE DE DONNEES / AU PREALABLE ON VERIFIE QU'ELLE N'EST PAS DEJA EXISTANTE */
CREATE DATABASE IF NOT EXISTS theaterDB CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

```

- Création des différentes tables (en fonction des entités définies dans le MCD puis MPhD). La commande engine=innodb en fin de création permet de préciser au SGBD quel type de moteur de stockage utilisé afin d'optimiser les performances. Chaque clé primaire des tables sera basé sur une valeur unique non nulle auto-incrémentée. La commande IF NOT EXISTS permet d'éviter de créer une nouvelle table si celle-ci existe déjà dans la base de données

```sql
CREATE TABLE IF NOT EXISTS theater_complex (
  complex_id INTEGER AUTO_INCREMENT PRIMARY KEY NOT NULL,
  complex_name VARCHAR(60) NOT NULL
) engine=INNODB;
```

2. READ

Une fois les tables crées, il convient d'insérer des données de test afin de s'assurer que les liens entre les tables sont bonnes.
On va donc utiliser la commande INSERT INTO. La clé primaire étant auto-incrémentée, il n'est donc pas nécessaire de la renseigner.

```
/* ON INSERE DES DONNEES */
INSERT INTO client (client_ln, client_fn, client_email, client_password)
VALUES
('Bradman', 'Arturo', 'abradman0@google.com', '$2y$10$88wgseBRHdEP2XEqoJDZO.pxjO2xngg/FOAvPVpvFPEzF9/fFEUGm'),
('Baynes', 'Roley', 'rbaynes1@t-online.de', '$2y$10$bG2y7QKTR3kOd.3eNU9PG.yqdhzY9IJTa7i23voPG3fX6MtUMwBlW'),
('Siaspinski', 'Bethena', 'bsiaspinski2@time.com', '$2y$10$z/Vgxww6xvl9gosRva8rR.j7ZNgfrnZJ.yCeh5q1e/CwSEYP3UP/C'),
('Humbee', 'Nan', 'nhumbee3@dell.com', '$2y$10$NSNGsVoKtHjkTyww5nl20O6K/Mjl0nWbcU5t/rN/Lbe39OCgBcS8e'),
('Garfitt', 'Angelique', 'agarfitt4@blogger.com', '$2y$10$riv7QfpVCekXXbGMEsadnuYqcl7Qy9TYMQn.DYcgdkhofqnr5vqAO'),
('Algy', 'Cecily', 'calgy5@biglobe.ne.jp', '$2y$10$pyRG/dz6RV/aFbSXQfQwMuyLM59.nbpE2rACuViN4J4icKuIuLfSK'),
('Cavaney', 'Nevin', 'ncavaney6@google.de', '$2y$10$ZRWjydK89xwtZT0Ra6.l7uyrlVL8EKdRjNJsT7zK2PHQoW6amHuHm'),
('Hylands', 'Teddie', 'thylands7@samsung.com', '$2y$10$qB7HSkr.trwzyBZGloCdnOVM1lA.Azvkngiv4pbv420Id941eqHfO'),
('McCowen', 'Ketti', 'kmccowen8@fema.gov', '$2y$10$7ORhzK8dAh5iECmcWke4p.jLFWaiSLUjycna4Kj7cMOiI9aF8fbG6'),
('McQueen', 'Janean', 'jmcqueen9@altervista.org', '$2y$10$g73WPAhTb8wajyunfKK7ZupCNrt8xW1svTPW9/13iPxj16D8c2dRm'),
('Woolmington', 'Robbert', 'rwoolmingtona@infoseek.co.jp', '$2y$10$2PSZEeY2t1pnnRqQbVV4hOYEnE9kq5cozthTDGjuOVqCyNk1Vmk7S'),
('Shovelton', 'Giuseppe', 'gshoveltonb@virginia.edu', '$2y$10$fWLffyXNb/kbRRGmBtKXnODnulhL/yxpE/JZ.kfiJq/zdX0UPQ2QG'),
('Brownill', 'Cindie', 'cbrownillc@photobucket.com', '$2y$10$7mHUVua.Wqj.aIHukvI9geVvb71yMW/of2SG6FoSeDbo1R9Hhtxam'),
('Dufaur', 'Kylie', 'kdufaurd@wikia.com', '$2y$10$0dZ.X0T6rOqHdqTJRb9/JOF5Ji4jDXJmGfX7EMmxGI3Se5JuGSXDm'),
('Spenley', 'Norrie', 'nspenleye@csmonitor.com', '$2y$10$pNOCBDq0xEXlqz1VioJxreQ97IfugMpX5paaTPcbGWvMIZES.Hp32'),
('Simondson', 'Lani', 'lsimondsonf@biglobe.ne.jp', '$2y$10$wmLMlkv8wVBBA3VnUfWA/utnHtQrELFcVJWtohedvXQrmM0bh4OQe'),
('Halsey', 'Robbyn', 'rhalseyg@blogger.com', '$2y$10$vyLPlvpP6JLm.Oi78zQObOunCVpmIWvHt3NOxoYbOJ1E1/c6S9sAq'),
('Paschek', 'Cloris', 'cpaschekh@bizjournals.com', '$2y$10$nOl8TllUY1UoNfWguX31F.DG1JhVjAqOHmKFa9In4kph6dNfRKEmS'),
('Izen', 'Olenolin', 'oizeni@woothemes.com', '$2y$10$ocpiOalKY4ZJLuEbevgksubNinp39k7aVTJGS3L38qX/EqneLcnmK'),
('Gonneau', 'Clarice', 'cgonneauj@csmonitor.com', '$2y$10$Ns/S4hXzO6MA0hKAK2d.guqiyAWZ/yx5b5wz/qwjvqdfM07mFdEpG'),
('Quinlan', 'Marshall', 'mquinlank@bigcartel.com', '$2y$10$MA4hYmH3nYU1xrqN..mW1OL36Bx7qtS9.8xxxBUjmBLXVVlGypVCa'),
('Witchard', 'Mordecai', 'mwitchardl@disqus.com', '$2y$10$IxC.WzLRzzXpmzA49c4W6ueT5RPJUkc56pc//ZG6SXHgPd.dFndcO'),
('Visco', 'Martita', 'mviscom@oracle.com', '$2y$10$rU6T2VpSSfykF/xyGjhHbeZNOl5DpwKiXpoMa9QHgTsoizLgwXp2O'),
('Vittet', 'Cymbre', 'cvittetn@symantec.com', '$2y$10$3Xq9PdEbfodvhvPOEzEl8OuAKOUYVmUPEnhMm7aGMjCkkAs5xsi2m'),
('Houtby', 'Mikey', 'mhoutbyo@utexas.edu', '$2y$10$3oIfZmiKuuFnaxiAXkAeg.Kac0WOkuLuYjZjLae4hSfNX0yE7XNGC');
```

Les données insérées nous pouvons utiliser des requêtes de sélection en utilisant la commande SELECT FROM.
La liaison entre les tables se fera grâce à la commande JOIN ON.
La condition WHERE permettra quant à elle de conditionner l'affichage des données.
Il est également possible d'utiliser la clause ORDER BY afin de grouper des données et la condition HAVING qui permet de conditionner l'affichage sur la clause ORDER BY.

```
SELECT
f.film_title AS 'FILM',
r.room_number AS 'SALLE',
r.room_seats AS 'PLACES EN SALLE',
r.room_seats - COUNT(t.event_id) AS 'PLACES DISPONIBLES',
COUNT(t.event_id) AS 'TICKETS ACHETES'
FROM event e
JOIN ticket t ON e.event_id = t.event_id
JOIN room r  ON e.room_id = r.room_id
JOIN film f ON e.film_id = f.film_id
GROUP BY t.show_id;
```

3. UPDATE

La force des bases de données relationnelles est de pouvoir mettre à jour certains enregistrements, voire certaines valeurs de manière rapide, simple et sécurisée.
On utilisera alors la commande UPDATE … SET.
Dans cet exemple on affiche un client en particulier :

```
USE theaterDB;

SELECT *
FROM client
WHERE client_id = 2;
```

client_id	client_fn	client_ln	client_email	client_password
2	Roley	Baynes	rbaynes1@t-online.de	$2y$10$bG2y7QKTR3kOd.3eNU9PG.yqdhzY9lJTa7i23voPG3fX6MtUMwBlW

Puis on le met à jour et on l'affiche de nouveau grâce à une requête SELECT.

```
UPDATE client
SET client_ln = 'ROBERT'
WHERE client_id = 2;

SELECT *
FROM client
WHERE client_id = 2;
```

client_id	client_fn	client_ln	client_email	client_password
2	Roley	ROBERT	rbaynes1@t-online.de	$2y$10$bG2y7QKTR3kOd.3eNU9PG.yqdhzY9lJTa7i23voPG3fX6MtUMwBlW

Les données étant liées, mettre à jour une donnée dans une table, mettra à jour les données liées dans une autre table.

4. DELETE

Enfin en cas de besoin, le client pourra supprimer certaines données (en cas d'erreur de mise à jour, de suppression d'un film etc…).
Les données étant liées, supprimer une données « parente » supprimera également les données enfants :

- supprimer un cinéma, supprimera également toutes les salles associées ainsi que les séances créées.
- supprimer un film, supprimera toutes les séances associées à ce film. (On pourrait alors y mettre une condition pour les séances déjà passées afin de garder un historique => DELETE FROM WHERE avec une jonction sur les séances pour conditionner cette suppression).

Toutefois, certaines données parentes n'enclencheront pas la suppression des données enfants . J'ai choisi ici de ne pas supprimer les cinémas (entité physique) en cas de suppression d'un complexe (entité non physique). Toutefois dans le cadre d'une base de données ne concernant qu'un seul type de complexe (si la base de données ne concerne par exemple que les cinémas PATHE GAUMONT), il conviendra alors de supprimer tous les cinémas n'appartenant pas à la marque.
Le retour du client nous permettra de mettre à jour la base de données selon ses besoins propres.

```
USE theaterDB;

SELECT COUNT (e.event_id) AS 'Nombre de séance', f.film_title AS 'Film'
FROM film f
JOIN event e ON e.film_id = f.film_id
WHERE f.film_id < 4
GROUP BY f.film_title;
```

Nombre de séance	Film
4	Vertigo
2	The Innocents
2	The Deer Hunter

L’aspect sécurité

Les données étant accessibles en clair dans une base de données, un principe de sécurité veut que les informations sensibles (dans notre cas, les mots de passe de nos clients pour leur compte) soit hashées (c’est à dire soit traitées par un algorithme spécifique afin de rendre leur lecture impossible.
Ce travail se fera à priori (avant l’insertion en base de données) par l’application. Au moment de la validation des données, l’application se chargera de hasher la données sensible puis transmettra l’ensemble des données dans la base de données.
Dans notre base de données actuelles, les mots apparaissent donc directement hashé via l’algorithme BCrypt (https://www.bcrypt.fr/ pour le faire en direct).

Le client a également demandé la possibilité d’avoir des comptes administrateurs pour certaines personnes au siège principal (accès intégral à la base de données) et des comptes pour les gestionnaires de cinémas, pour mettre en place leurs séances (écriture dans la table event).
J’ai pour cela procéder à la création de 3 rôles :

- app_gest qui à toutes les autorisations
- app_write qui à les droits d’écriture (SELECT, INSERT, UPDATE, DELETE) sur la table event
- app_read qui à les droits de lecture sur l’ensemble de la base de données theaterDB

Une fois les rôles créés, il suffit alors de créer des utilisateurs (CREATE USER) avec un mot de passe (IDENTIFIED BY), auxquels nous affectons des droits (GRANT [nom du rôle] TO [nom de l’utilisateur].

```
CREATE ROLE IF NOT EXISTS 'app_gest', 'app_write', 'app_read';
GRANT ALL ON theaterDB.* TO 'app_gest';
GRANT SELECT, INSERT, UPDATE, DELETE ON theaterDB.event TO 'app_write';
GRANT SELECT ON theaterDB.* TO 'app_read';

CREATE USER IF NOT EXISTS 'GBadmin' IDENTIFIED BY 'GBadminPassword';
CREATE USER IF NOT EXISTS 'WRadmin' IDENTIFIED BY 'WRadminPassword';
CREATE USER IF NOT EXISTS 'RDadmin' IDENTIFIED BY 'RDadminPassword';

GRANT 'app_gest' TO 'GBadmin';
GRANT 'app_read' TO 'RDadmin';
GRANT 'app_write', 'app_read' TO 'WRadmin';
```

Export de la base de données

Il convient aussi de prévoir, en cas de soucis majeur avec la base de données, un script qui permettra de sauvegarder la base de données à un instant T.
Avec MySQL, un outil est mis à disposition : mysqldump.
Aussi avec une ligne de commande spécifique, il est possible d’exporter la base de données en fichier sql : « mysqldump -u [nom de l’utilisateur] -p [nom de la base de données à exporter] > [chemin du fichier à exporter] ».

Utilisant le système MacOS, la ligne de commande en local donne ceci :

```
/usr/local/mysql/bin/mysqldump -u root -p theaterDB > saveDB2.sql
```

L'insertion du mot de passe de l'utilisateur se fera via un prompt dans l'utilitaire de commande. Ne jamais mettre le mot de passe dans la ligne de commande.

Import de la base de données

A l'inverse, lorsque nous avons besoin de restaurer la base de données, il suffira, au préalable de supprimer la base de données (DROP DATABASE), de la recréer (CREATE DATABASE) puis en se connectant dessus, importer le ficher créer avec mysqldump.

```
/usr/local/mysql/bin/mysql -u root -p theaterDB < ~/Desktop/saveDB2.sql
```

A noter que les données USER et ROLE créées au préalable ne sont pas affectées lors de la suppression de la base de données.

Les cas d'utilisations

Pour permettre au client d'observer le bon fonctionnement de sa base de données, je propose, grâce aux cas d'utilisations de mettre en place des requêtes (de sélection, d'insertion, de suppression ou de modification). Ces cas seront bien évidemment non exhaustifs mais simplement à titre d'exemple.
En fonction donc du rôle de chaque utilisateurs, ses utilisations vont être plus ou moins différentes.
Je distingue 3 utilisateurs :

- l'administrateur global (qui doit être à gestionnaire de l'enseigne/ complexe de cinémas)
- le gérant d'un cinéma
- le client d'un cinéma

Le client va vouloir :

- afficher les films à l'affiche
- sélectionner un film et voir les séances de ce film pour la journée choisie avec les places restantes
- choisir la séance et une tarification et valide son billet

Le gérant du cinéma va vouloir :

- afficher le nombre de billets vendus dans ses salles pour une date spécifique (ou plage de dates)
- afficher la liste des ses clients avec leur adresse mail (et le nombre de billets acheté par client, trié par CA)
- ajouter une séance (date et film)
- supprimer une séance

L'administrateur global va vouloir :

- afficher le CA total de toutes les salles de cinéma sur une plage de date données
- ajouter un cinéma
- ajouter des salles de cinéma à un cinéma
- afficher tous les clients du complexe et leur email pour de la communication

Ces requêtes sont en pièces jointes dans le dossier 'Cas d'utilisations'.

Moyens et outils utilisés

Afin de pouvoir mettre en place cette évaluations, j'ai utilisé les éléments suivants :

Visual Studio Code comme outil de développement (IDE).
Le site DBFiddle (https://www.db-fiddle.com/) m'a permis de tester mes lignes de commande SQL en direct.
Le site officiel de mySQL m'a permis de mettre en place mon serveur local mySQL pour la mise en place de la base de données (https://dev.mysql.com/downloads/mysql/).
Le site BCrypt (https://www.bcrypt.fr/) m'a permis de hasher facilement les mots de passe à insérer en BDD.
Le site LucidChart (https://lucid.app/) m'a permis de mettre en place facilement la partie graphique du MCD et MPhD.
Le site GitHub (https://github.com/) m'a permis de mettre en ligne les fichiers de ma base de données.

Contexte

Ce site a été développé dans le cadre d'une évaluation d'entrainement avec un sujet proposé par la plateforme d'étude Studi.

Le live d'explication fourni par la plateforme m'a permis de mettre à plat et de clarifier les attentes sur ce projet.