Évaluation :

Créer et administrer une base de données

Référence : GDWFSCAUBDDEXAIII2A

Lien github: <https://github.com/Thioni/EvalBDD>

Diagramme de classe :

J’utilise pour créer mon diagramme de classe UML le site [drawio](https://app.diagrams.net/)

La relation entre la classe *“user”* et la classe *“ticket”* par exemple étant une **one-to-many** dans la mesure où un utilisateur peut acheter zéro ou plusieurs tickets.

Les classes “C*omplex*” et *“MovieScreening*” ont elles une relation **many-to-many** puisqu’un complexe aura une ou plusieurs projections, tandis qu’un même film pourra être projeté dans un ou plusieurs de ces complexes.

On a également une relation **One-to-one** entre les classes *“Administrator”* et *“MovieScreening”* puisqu’un seul administrateur peut se voir attribué un complexe pour lequel il pourra ajouter des projections.

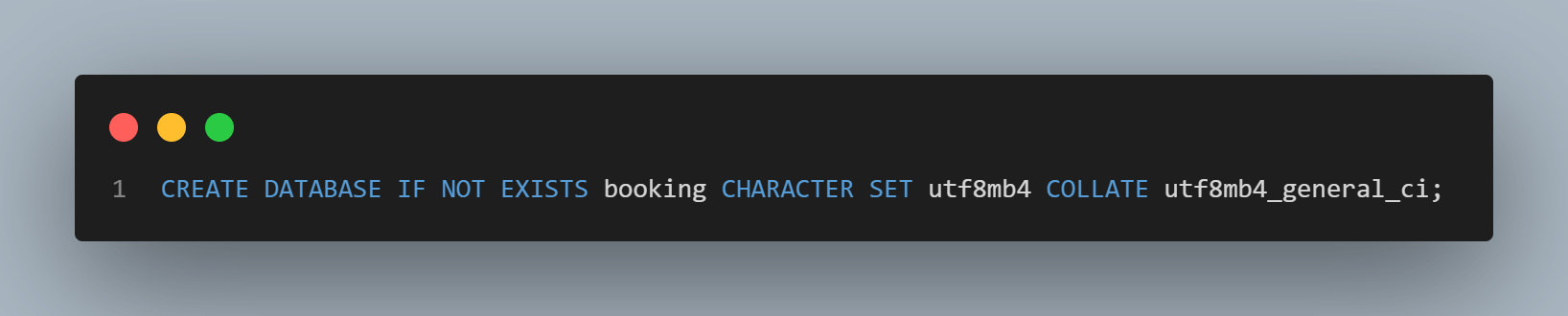
La cardinalité **0..1** représente le fait qu’un administrateur au plus peut avoir cette fonction, mais que certains administrateurs ne pourront ajouter de séances nulle part.

On trouve également sur ce diagramme une relation de composition entre les classes *“Hall”* et *“Complex”* puisque les salles de projections ne peuvent exister que si les complexes qui les accueillent existent eux même.

Au niveau des méthodes, on a dans la classe *"Administrator”* une méthode “addScreening()” qui servira à ajouter des projections, et dans la classe “*PaymentMethod*” une méthode “confirmationMail()” qui servira à envoyer un mail de confirmation après réservation d’une place.

Les tables :

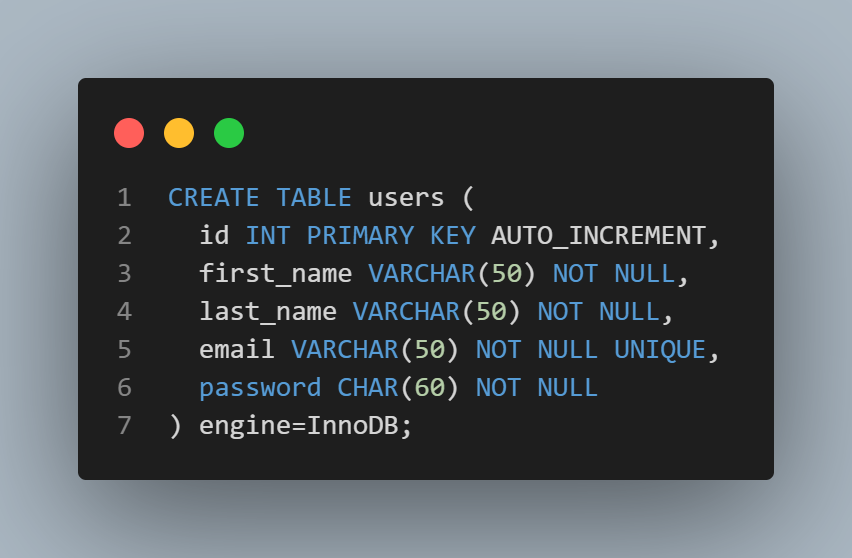
En premier lieu je créer la base de données elle-même :



La base de données est ici créée en précisant l’utilisation de l’encodage **utf8mb4** afin de pouvoir utiliser les différents types d’alphabets, voire au besoin des emojis.

Par sécurité, la création des tables se fait en utilisant ici le moteur de stockage transactionnel **innoDB** afin d’éviter ou de limiter d’éventuels problèmes de traitement des requêtes en cas de problème technique.

Les colonnes sont créées en utilisant les types définis dans le diagramme, les champs **string** devenant ici des **VARCHAR.**



Pour les champs “*email*” des tables “*users*” et “*administrators*” j’ajoute la contrainte **UNIQUE** afin de n’avoir qu’une seule création de compte possible par adresse mail.

Clefs primaires :

Pour toutes les clefs primaires j’utilise une clef artificielle afin d’avoir la garantie qu’elles ne trouvent jamais modifiées par de potentiels changements apportés aux tables.

Celles-ci seront auto-incrémentées et disposeront donc automatiquement d’un identifiant unique et qui ne pourra être nulle, la contrainte **NOT NULL** étant ici utilisée.

Clefs étrangères :

Elles sont définies en fonctions des cardinalités établies dans le diagramme de classe, la majorité d’entre elles seront de type **one-to-many** et recevront donc une clef étrangère.

Par exemple, la relation entre la classe *“user”* et la classe *“ticket”* étant une **one-to-many**, la classe *“ticket”* étant du côté multiple de la cardinalité inclut une clef étrangère nommé ici “*customer*”.

On trouve cependant une relation **many-to-many** entre les tables *“movie\_screening”* et *“complexes”* puisqu'un ou plusieurs complexes sont susceptibles de projeter le même film, et que chaque complexe a aussi la possibilité d’avoir un même film projeté dans plusieurs salles.

Pour la table “*halls*” représentant les salles de projection, la clef étrangère liée à la table “*complexes*” reçoit également l’instruction “**ON DELETE CASCADE**”. En cas de suppression d’un complexe de la base de données, les salles de projections qui y sont liées seront également automatiquement supprimées.

Remplissage des tables :

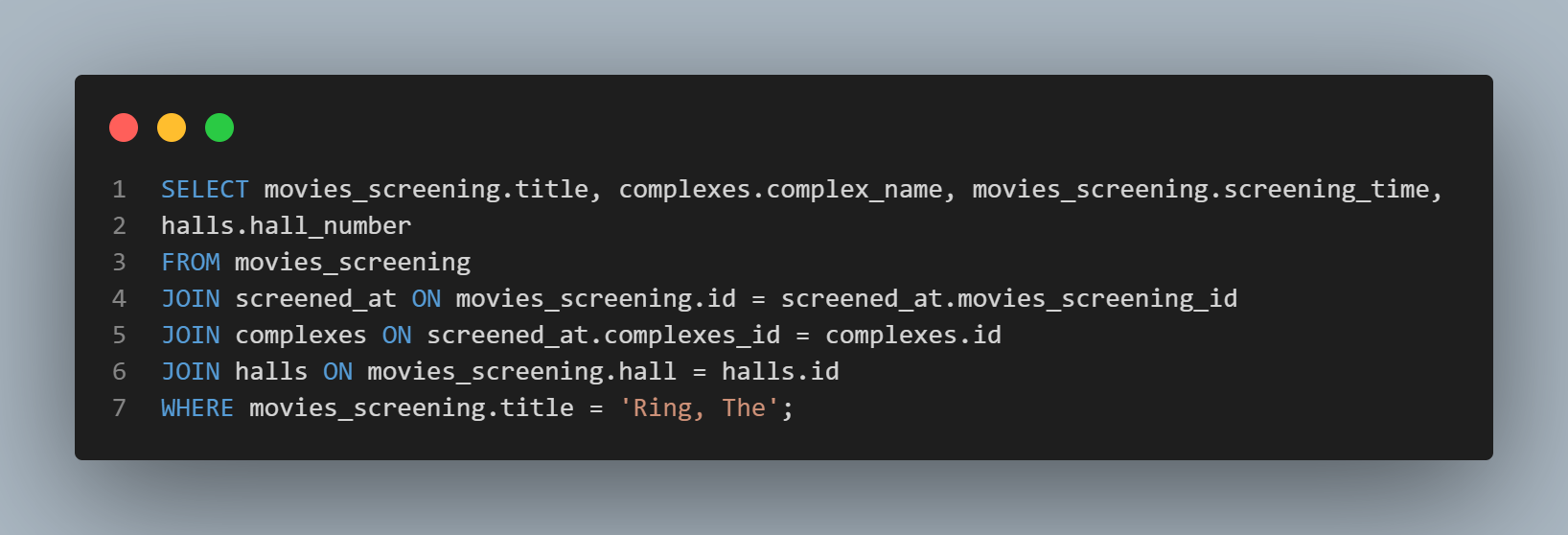
Je fais ici appel au site [mockaroo](https://www.mockaroo.com/) pour générer la plus grande partie des informations fictives.

Les champs *“ID”* n’apparaissent pas dans le code SQL puisqu’ils sont définis comme auto-incrémentés.

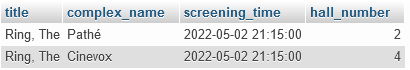
Mots de passe :

Les mots de passes des utilisateurs et des administrateurs sont ici cryptés avec bcrypt <https://www.bcrypt.fr/> , raison pour laquelle les champs *“password”* des tables *“users”* et *“administrators”* ont une valeur fixe de 60 caractères, bcrypt nous fournissant un résultat qui sera toujours d’une longueur de 60 caractères.

Au niveau des projections, on trouve deux projections simultanées du même film “The Ring”, on peut vérifier le fait que ces deux projections sont des entités à part entières en testant par exemple avec l’instruction SQL :

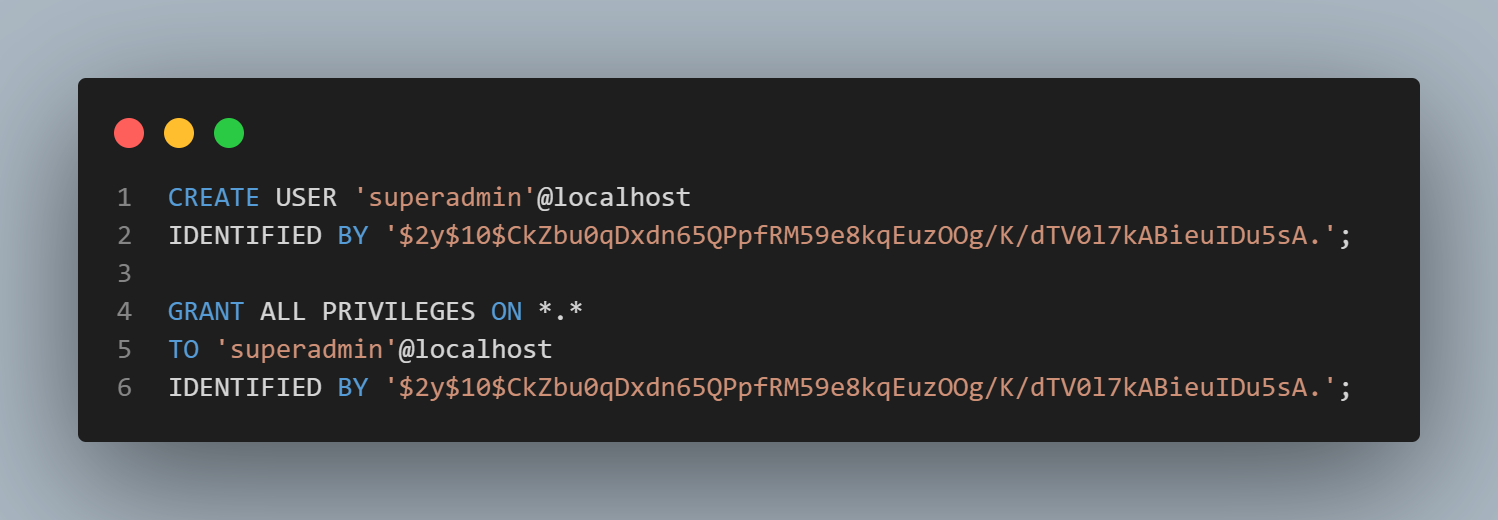


Celle-ci nous permet ici de rechercher dans la base de données un film dont le titre est “The Ring” et d’afficher à la fois le complexe de projection ainsi que l’horaire de celle-ci.



Comptes utilisateurs :

Au niveau des comptes administrateurs, on a un administrateur qui, conformément au souhait du client, aura une vue sur tout en lui donnant les accès



On aura également un administrateur par complexe ayant la possibilité d’ajouter des projections dans son complexe attitré et qui sera identifié grâce à la clef étrangère “*manage”* qui référence le champ *“id”* de la table “*complexes”*.

Export et import des données :

- L’export des données se fera avec la commande de sqldump



Ce qui me permet d’exporter un fichier SQL nommé ici “booking”.

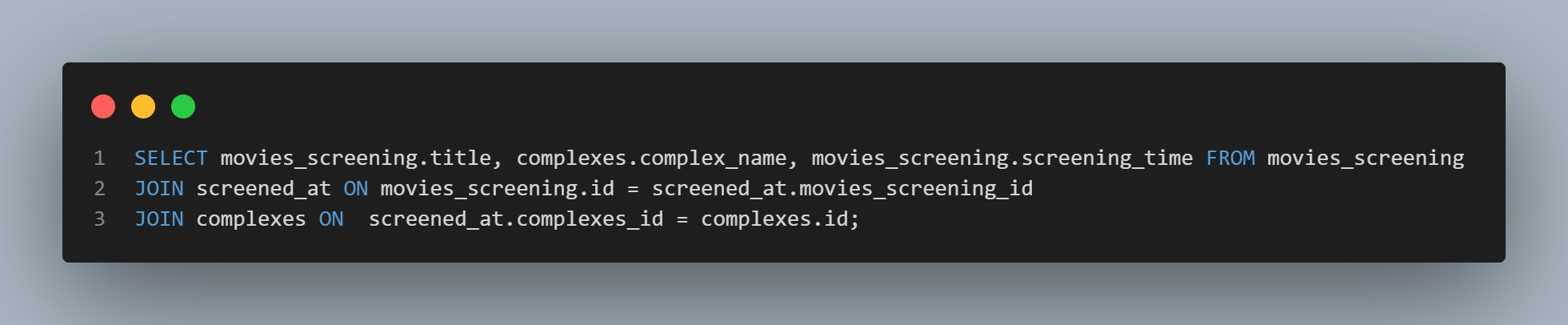
Pour importer ces données, j’utiliserai la commande :



Vérifications :

Pour tester la validité et la cohérence de la base de données j’utilise ici phpMyAdmin et y insert différentes requêtes.

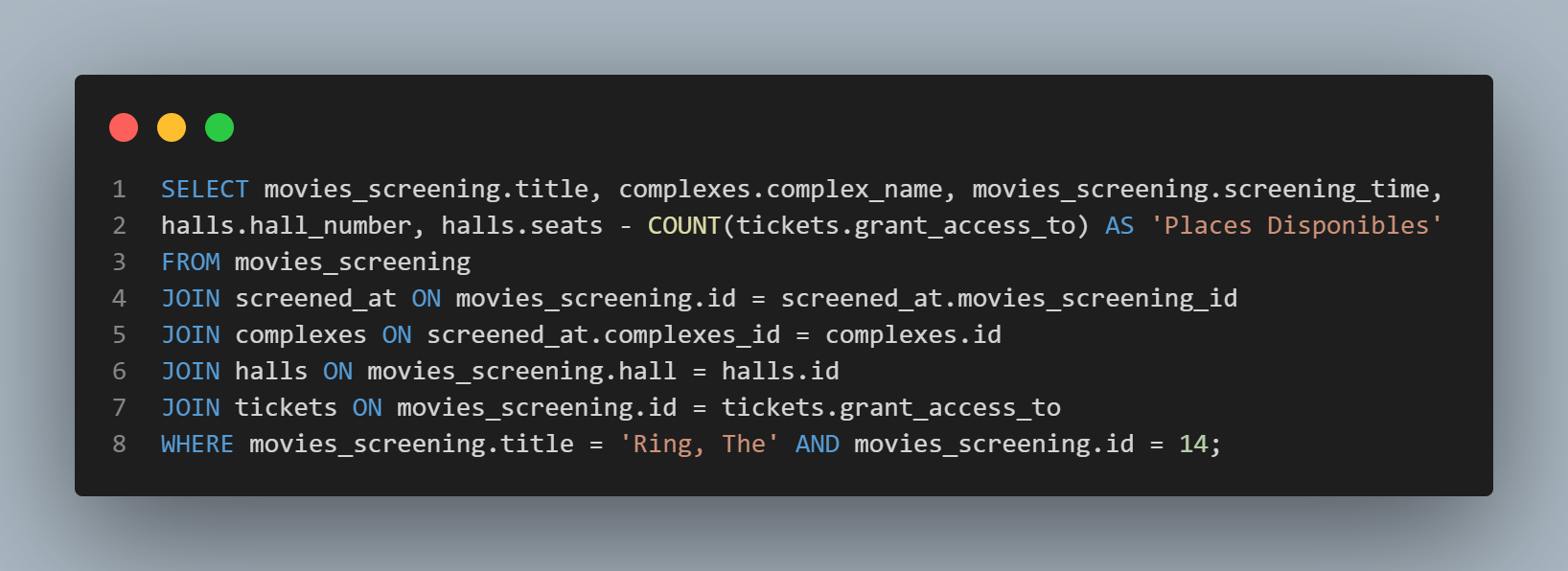
* Instruction permettant de voir les différentes séances :



Résultat de la requête :



* Instruction permettant d'afficher les informations sur le lieu de projection d’une séance en particulier, ainsi que le nombre de places disponibles :

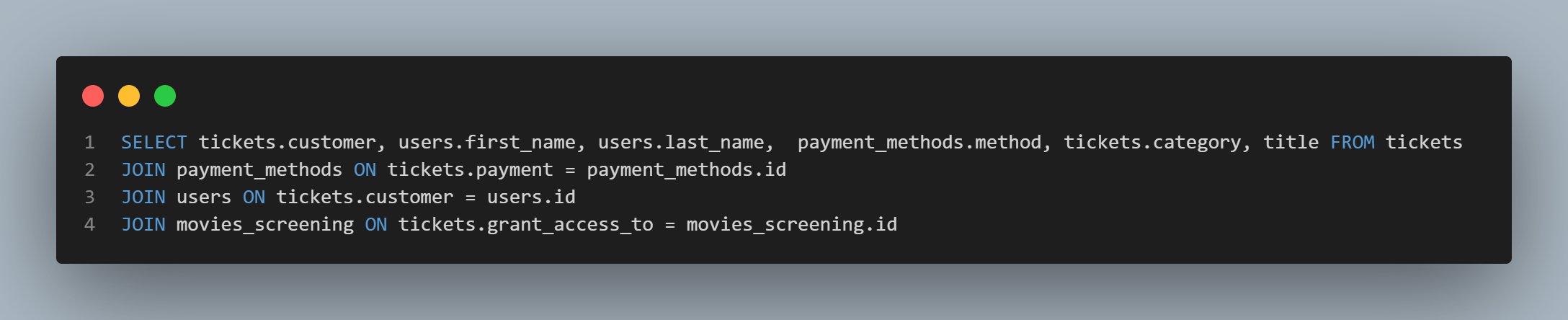


Pour cet exemple je vérifie combien de places reste pour la séance du film ‘The Ring’ du complexe Cinevox, le film disposant de deux séances simultanées sur deux complexes différents.

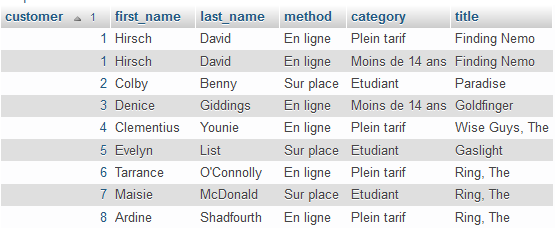
Résultat de la requête :



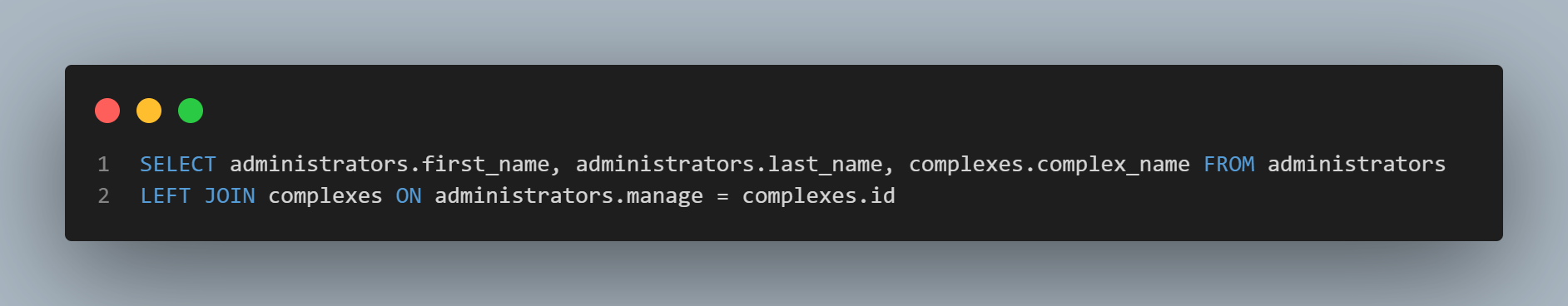
* Instruction permettant de voir la liste des utilisateurs avec la leur id client, nom, prénom, méthode de paiement, type de tarification et titre du film :



Résultat de la requête :



* Instruction permettant de vérifier quels administrateurs ont la possibilité d’ajouter des projections, et le cas échéant dans quel complexe.



J’utilise ici un LEFT JOIN au lieu d’un simple JOIN afin d’obtenir une liste de tous les administrateurs et non pas uniquement ceux ayant la gestion des projections d’un complexe.

Résultat de la requête :

