

COMPTE RENDU SQL-

Table des matières

Rappel du projet	3
Consignes	3
Structuration du projet	4
Mon organisation	4
Qu'est-ce que le SQL	4
Définition	4
Sqlite	4
Définition	4
Le nouveau schéma	5
Introduction.....	5
Récupération de l'ancienne base de données	5
Création base de données	6
Création des tables	6
Précisions de commande	7
Les clés	7
Clé Primaire.....	7
Clé Étrangère	7
Supprimer les tables	9
La migration	10
Première méthode.....	10
Méthode finale	10
Les requêtes.....	13
La première	13
La deuxième	14
La troisième	15
La quatrième.....	16
Le script Bash.....	17
Le menu.....	17
Le script Python	18
Introduction.....	18
Le menu	18
Choix de nom.....	19
Création base de données	21

Rappel du projet

Consignes

Le but du projet était de savoir migrer des informations sur une nouvelle base de données.
Le sujet est donc :

« Vous avez été contacté par TeamLikwid pour migrer leur ancienne base de données vers leur nouveau schéma. Ils veulent que vous :

- Créez un script .sql qui créera le nouveau schéma de base de données
- Créez un deuxième script qui transférera les données de l'ancienne base de données vers la nouvelle.

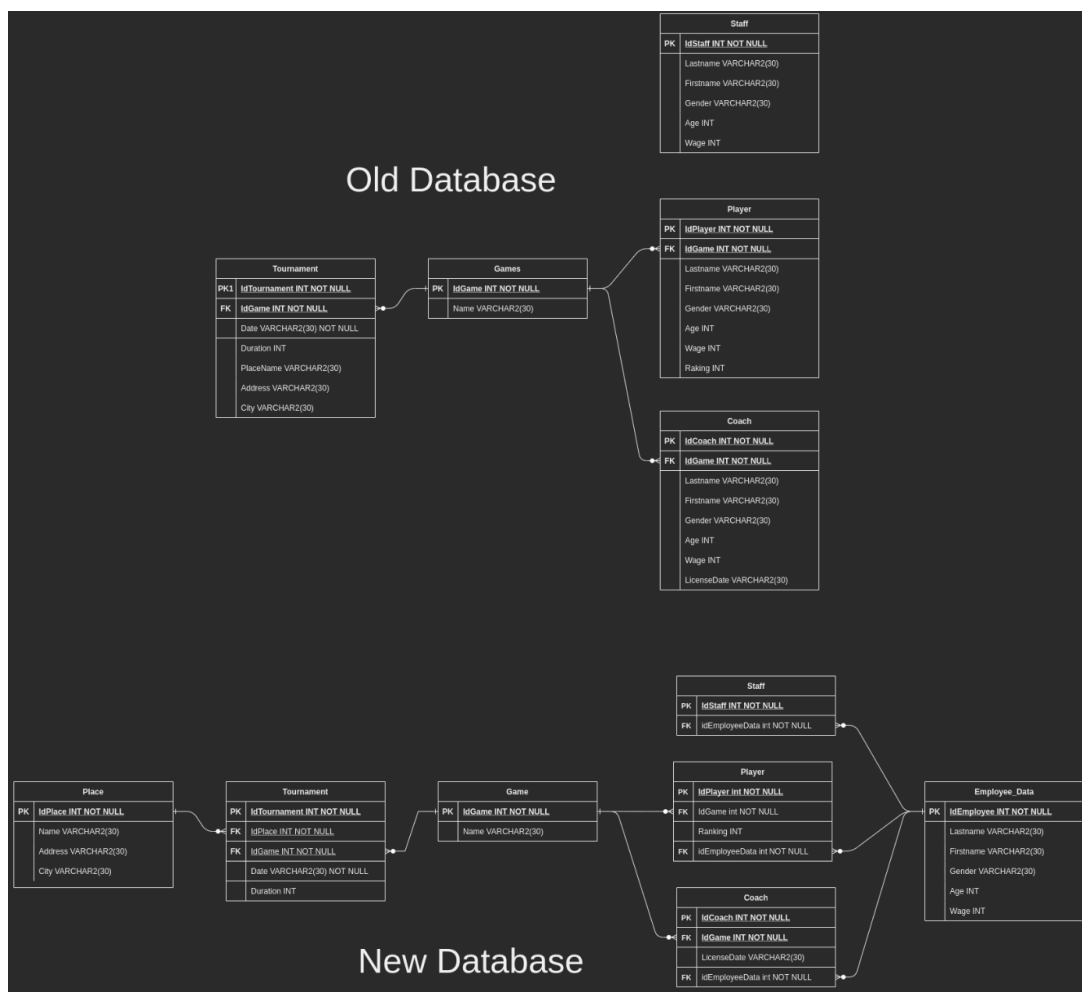
De plus, pour s'assurer qu'ils peuvent toujours accéder à leurs données, ils souhaitent que vous montriez les requêtes + les résultats pour les éléments suivants : - Listez tous les tournois pour un nom de jeu donné - Étant donné un nom de jeu, récupérez le salaire moyen des joueurs - Liste tous les tournois par lieu - Obtenez le nombre de joueurs par sexe. »

Un schéma d'aide était également fourni.

Il m'a permis de visualiser la forme de l'ancienne base de données ainsi que celle de la nouvelle.

Enfin, il fallait récupérer le fichier contenant l'ancienne base de données.

Une fois tout cela analysé, il était enfin temps de commencer !



Structuration du projet

Mon organisation

Il était important pour moi de me « décortiquer » le travail pour que tout soit plus clair. Pour cela, je me suis servie d'un outil, « Trello ». Il permet de pouvoir créer toutes sortes de tableaux qui permettent une meilleure organisation.

Celui-ci m'a permis de pouvoir découper mon travail en petites actions à effectuer. De ce fait, j'ai pu voir toutes les tâches à faire ainsi que mon avancée. Cela m'a beaucoup aidé à visualiser les différentes actions effectuées.

Je me suis également servie de l'IDE VS Code, il propose de nombreuses fonctionnalités qui m'ont été utiles. J'ai donc ensuite installé une extension me permettant de faire du SQLite.

Qu'est-ce que le SQL

Définition

Nous allons d'abord commencer par définir simplement le SQL.

Le SQL (Structured Query Language) est un langage de programmation utilisé pour gérer les bases de données relationnelles. Il permet de créer, modifier et interroger des bases de données. Avec SQL, vous pouvez effectuer des opérations telles que la sélection, l'insertion, la mise à jour et la suppression de données.

SQLite

Définition

Il y a plusieurs raisons pour lesquelles on utilise SQLite :

- Il est disponible gratuitement (open-source), ce qui signifie qu'il est libre d'utiliser et de distribuer, par conséquent, il est accessible à un grand nombre de développeurs.
- Il est léger et ne nécessite ni configuration ni maintenance, ce qui le rend idéal pour les applications qui ont un stockage et une mémoire limités.
- Il est populaire pour les applications mobiles car il est intégré à plusieurs systèmes d'exploitation mobiles, notamment iOS et Android.
- Il est facile à utiliser et peut gérer des quantités importantes de données, ce qui en fait un choix idéal pour les applications qui ont besoin de stocker et d'interroger de grandes quantités de données.

En résumé, SQLite est un choix populaire pour les applications qui ont besoin d'un système de gestion de bases de données léger, stable et facile à utiliser.

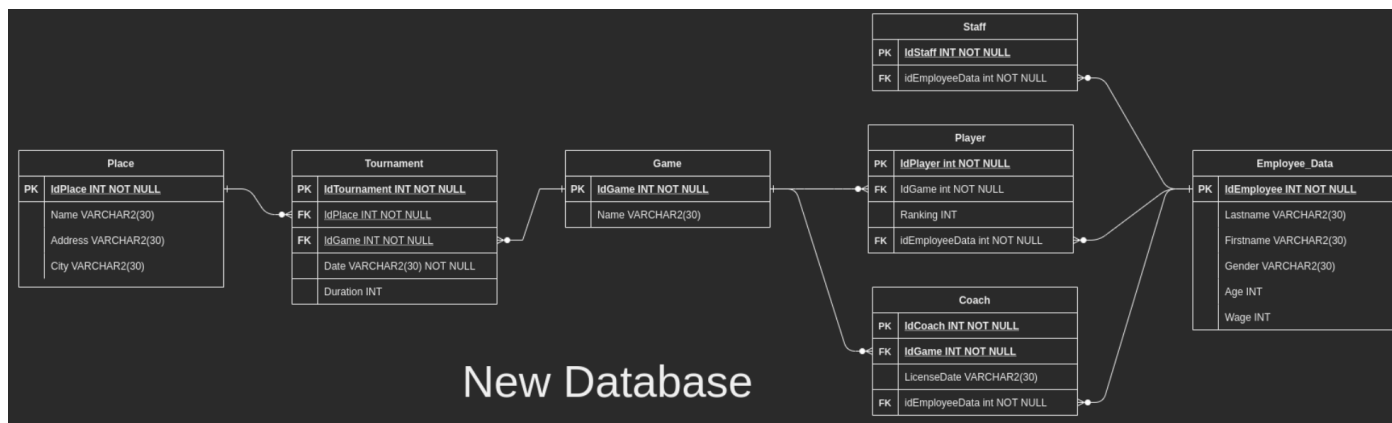
Le nouveau schéma

Introduction

Pour créer une nouvelle base de données, Il y a plusieurs étapes à respecter.

La première est tout simplement de créer le nouveau schéma qui nous permettra par la suite de transférer les données de l'ancienne base à la nouvelle.

Un modèle de schéma nous a été fournis, il faut donc le respecter :



Récupération de l'ancienne base de données

Pour faciliter mon travail de connexion avec les bases de données, j'ai installé sql tools sur VsCode.

C'est une extension qui permet la gestion de base de données et qui permet de gagner en facilité et en temps.

Dans le sujet, le fichier de l'ancienne base de données nous a été fourni.

J'ai donc récupéré le fichier du sujet et je l'ai téléchargé.

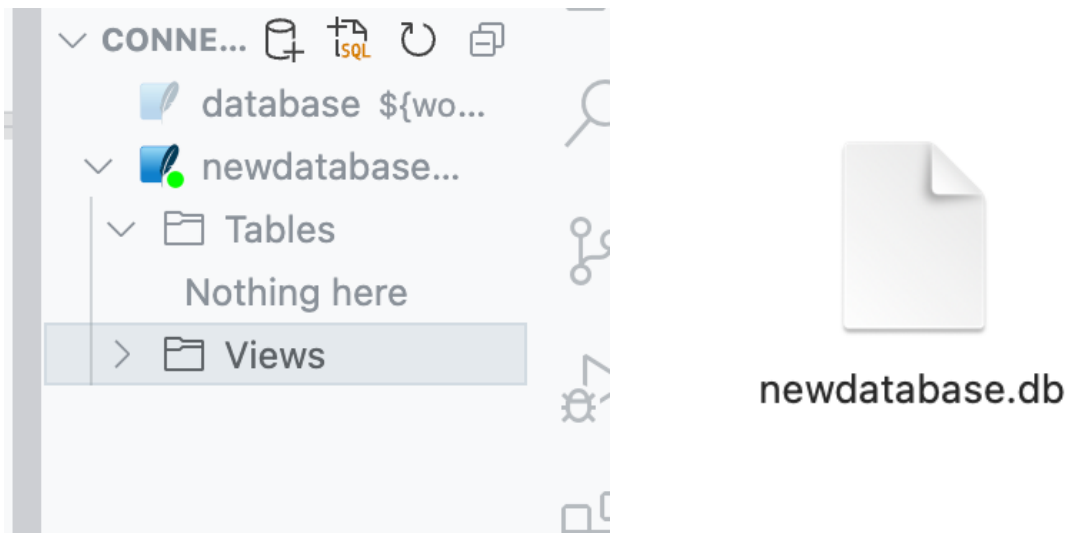
Je l'ai renommé pour que le nom est plus de sens pour moi lorsque que je travaillerais dessus.

On configure ensuite l'interface qui nous permet d'interagir avec la base de données, on peut directement faire cela via VsCode.

Création base de données

Pour commencer, il faut d'abord créer la nouvelle base de données.

Pour cela, il faut créer un fichier .db avec le nom qui nous convient. On fait donc exactement la même manipulation qu'avec l'ancienne base de données récupérée.



On obtient donc une base de données vide.

Création des tables

On peut donc ensuite créer les tables de la base de données.

Pour cela, j'ai créé un script SQL se nommant : createtable.sql.

J'ai donc commencé par créer les tables une par une pour avoir exactement le schéma demandé comme ci-dessus.

Pour le moment, je créer toutes les tables de mon schéma, vidée. Je les remplirais en suivant en migrant les informations de l'ancienne base, vers la nouvelle.

La commande qui m'a permis de créer mes tables est la commande : « CREATE TABLE »

Si on prend l'exemple de la table « Place », voici ce que ça donne :

```
CREATE TABLE IF NOT EXISTS Place (  
    IdPlace INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,  
    Name VARCHAR2(30),  
    Address VARCHAR2(30),  
    City VARCHAR2(30)  
);
```

On remarque donc bien la commande « CREATE TABLE » qui me permet donc comme on le remarque ci-dessus de créer la première table à gauche de mon schéma, « Place ».

Précisions de commande

Juste en suivant, j'ai rajouté la commande « IF NOT EXIST », qui me permet de ne pas avoir d'erreur. Sans cela, une erreur serait apparue en m'informant que la table existe déjà.

Cela m'a permis de Créer la table seulement si elle n'a pas été créé avant, et de pouvoir continuer d'écrire mon script sans erreur.

On remarque aussi, « INTEGER » qui me permet de notifier à l'IDE que je veux seulement des nombres entiers à cet endroit-là.

On lui indique aussi que l'on ne veut pas de valeurs nulles lorsque la table sera créée : « NOT NULL ».

Les clés

Clé Primaire

Si nous continuons à « avancer » dans la commande, nous remarquons une clé primaire.

Une clé primaire est utilisée pour créer des relations entre les tables.

Pour indiquer une clé primaire, il suffit de mettre :

« PRIMARY KEY AUTOINCREMENT ».

J'ai rajouté auto-incrémente, car cela permet de spécifier que cette contenant une clé primaire sera incrémentée automatiquement.

Clé Étrangère

Si nous prenons, une autre table, la table Coach.

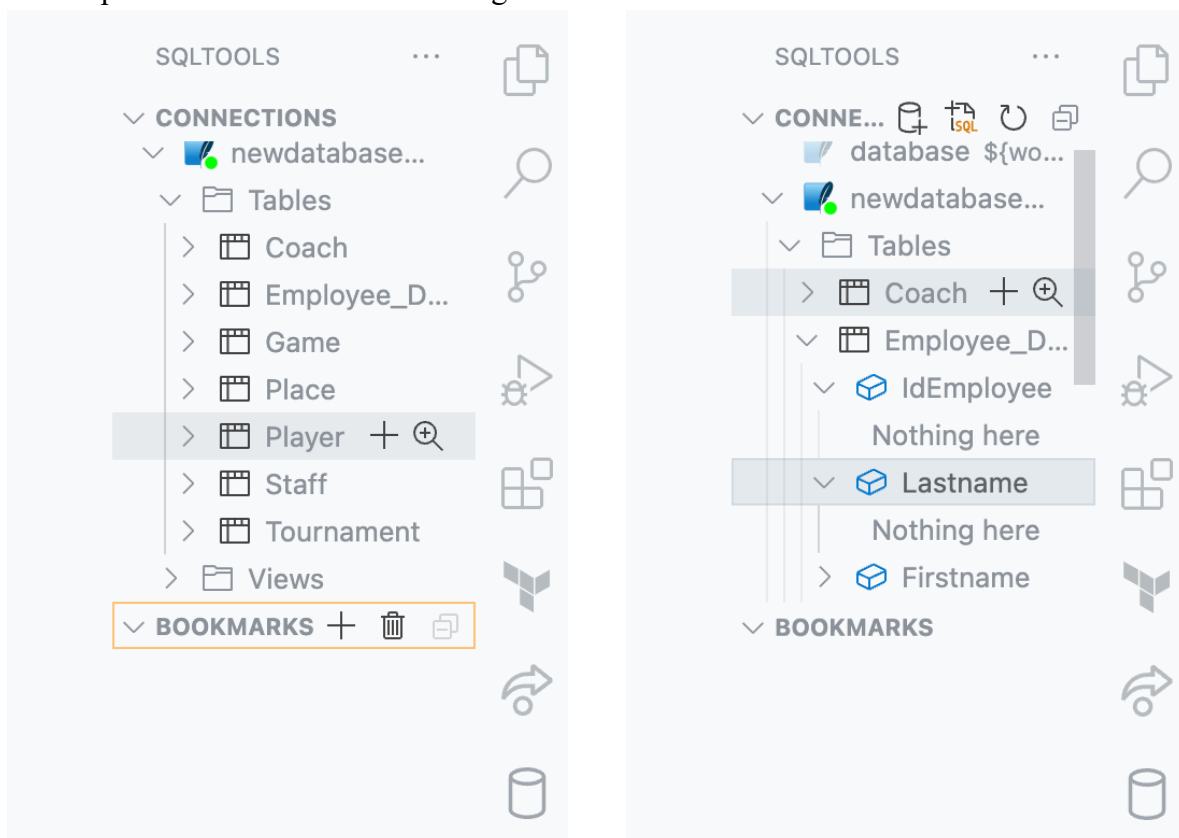
Nous remarquons que dans celle-ci nous avons une clé étrangère.

```
CREATE TABLE IF NOT EXISTS Coach (
    IdCoach INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    IdGame INTEGER NOT NULL,
    LicenseDate VARCHAR2(30),
    idEmployeeData INTEGER NOT NULL,
    FOREIGN KEY (IdGame) REFERENCES Game(IdGame),
    FOREIGN KEY (idEmployeeData) REFERENCES Employee_Data(IdEmployee)
);
```

Une clé étrangère, également appelée clé secondaire, est une contrainte de base de données utilisée pour créer des relations entre les tables d'une base de données.

Elle permet de faire référence à une clé primaire dans une autre table.

La par exemple, pour la table coach, elle fait référence à la table Employee_Data. On peut donc définir une clé étrangère en utilisant la commande "FOREIGN KEY" lors de la



création de la table comme effectuer ci-dessus.

Une fois toutes les tables créées, me voici donc avec mon schéma de ma nouvelle base de données.

Si l'on regarde dans la nouvelle base de données on s'aperçoit que les tables ont bien été créées mais qu'il n'y a rien à l'intérieur.

C'est normal puisque nous avons seulement créé les tables sans rien mettre à l'intérieur.

C'est dans la prochaine étape que nous allons remplir la nouvelle base de données ainsi que les tables qu'elle contient en migrant les données de l'ancienne base de données à la nouvelle.

Supprimer les tables

J'ai créé un script qui me permet de supprimer. Toutes mes tables.
Je l'ai nommé deletetable.sql

Je l'ai principalement utilisé pour tester mon code.

Pour supprimer mes tables j'ai utilisé la commande « DROP TABLE ».
Cette commande permet de supprimer définitivement une table d'une base de données.
Cela supprime en même temps les éventuels index, trigger, contraintes et permissions associées à cette table.

Cela m'a donc permis de pouvoir tester toutes les autres commandes sans erreurs.
Comme par exemple, les requêtes ou la migration.

```
▷ Run on active connection | ≡ Select block
DROP TABLE IF EXISTS Game;

DROP TABLE IF EXISTS Tournament;

DROP TABLE IF EXISTS Place;

DROP TABLE IF EXISTS Employee_Data;

DROP TABLE IF EXISTS Staff;

DROP TABLE IF EXISTS Player;

DROP TABLE IF EXISTS Coach;
```

J'ai également utilisé la commande « IF EXIST » qui me permet de supprimer les tables si elles n'existent pas.
Il permet de tester si la table existe ou non avant d'effectuer la commande.

La migration

Première méthode

J'ai d'abord commencé par faire une première méthode, je me suis ensuite rendu-compte que celle-ci ne serait pas la plus optimale.

Elle m'a cependant permis de visualiser les données à récupérer de l'ancienne base de données.

Le but de cette première technique était de récupérer les données pour ensuite les migrer sur la nouvelle base de données.

Voici à quoi ressemble mon script de base :

```
▷ Run on active connection | ≡ Select block
SELECT PlaceName, Address, City FROM Tournament;

SELECT IdTournament, Date, Duration FROM Tournament;

SELECT * FROM Game;

SELECT IdStaff FROM Staff;
```

J'ai simplement fait des « SELECT » avec toutes les informations donc j'avais besoins pour chaque nouvelle table de ma nouvelle base de données.

J'ai donc pu me rendre compte des données à transfère sur la nouvelle base de données. Ce script m'a donc servi de brouillon.

Méthode finale

Ce script est donc le script final qui m'a permis de migrer les données d'une base à l'autre. C'est un script que j'ai tout simplement nommé : migration.sql

J'ai d'abord lié les deux bases de données entres elles grâce à cette commande :

```
ATTACH DATABASE 'projectSQL.db' AS old_db;
ATTACH DATABASE 'newdatabase.db' AS new_db;
```

Elle me permet d'associer le fichier de base à mon nouveau fichier.

Cela permet de connecter les deux bases de données ensemble.

Je peux donc maintenant commencer la migration de mes données.

```
INSERT INTO new_db.Player(IdGame, Ranking, IdEmployeeData)
SELECT new_db.Game.IdGame, Ranking, new_db.Employee_Data.IdEmployee
FROM old_db.Player, new_db.Game, new_db.Employee_Data
WHERE old_db.Player.IdGame = new_db.Game.IdGame AND old_db.Player.Lastname = new_db.Employee_Data.Lastname;
```

Pour cela j'ai utilisé la commande « INSERT INTO » qui me permet d'insérer de nouvelles données dans une de mes tables.

Il faut d'abord spécifier le nom de la base de données dans laquelle je veux insérer mes données (ici ce sera « new_db »), suivit du nom de la table dans laquelle je veux insérer les nouvelles données.

Il faut ensuite renseigner la liste des colonnes entre parenthèse juste après le nom de la table. Je fais ensuite un « SELECT » pour sélectionner les colonnes que je souhaite récupérer. J'utilise ensuite « FROM » qui me permet d'indiquer les tables dans lesquelles je souhaite prendre les informations.

La commande « WHERE » est une condition qui me permet lier les données de l'ancienne table aux données de la nouvelle.

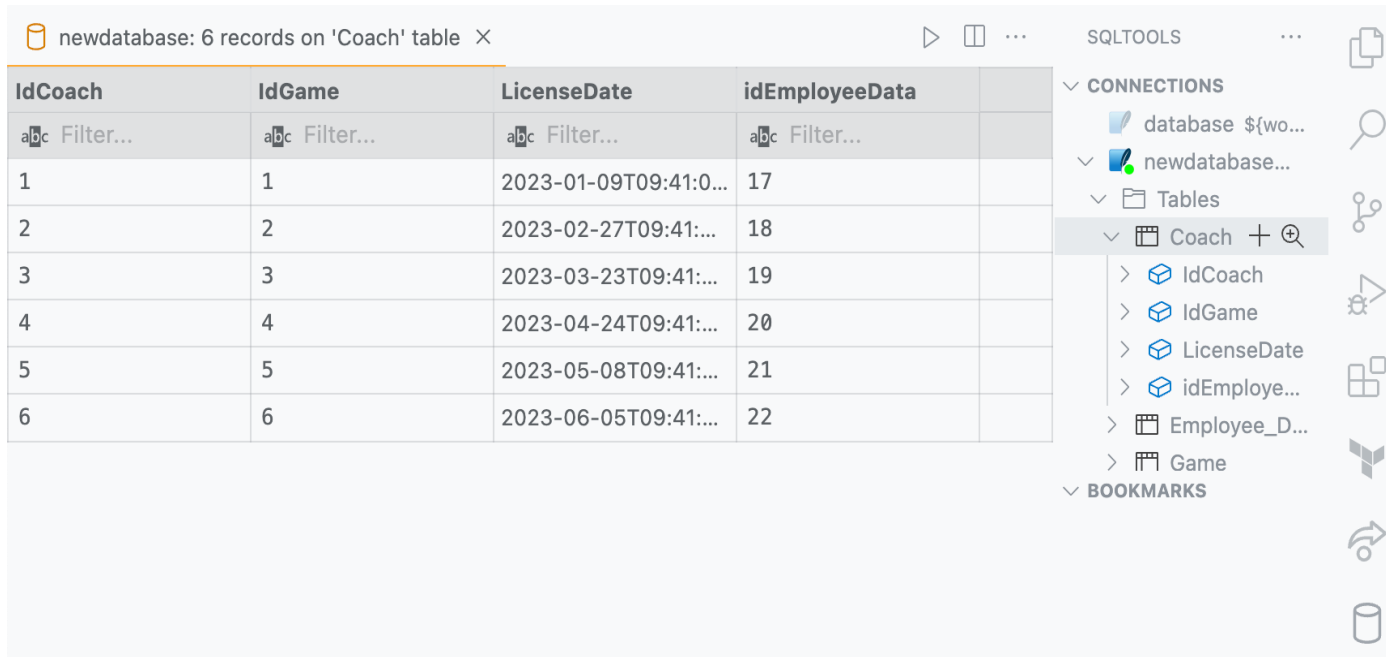
Pour cela on trouve une valeur qui est la même dans l'ancienne et la nouvelle table, ce qui nous permet de les mettre en communs.

Je répète ensuite toutes ses étapes pour toutes les tables de ma nouvelle base de données de qui me permet de la remplir des données nécessaires.

Une fois tout cela effectuer, toutes les données auront bien était migrer sur la nouvelle base de données.

On peut donc vérifier et s'apercevoir que les données ont bien migrer de l'ancienne base à la nouvelle.

Les données ne sont pas effacées le l'ancienne base, cela permet simplement de les copier et de les déplacer.



The screenshot shows a database management interface. At the top, a tab indicates 'newdatabase: 6 records on 'Coach' table'. Below this is a table with 5 columns: 'IdCoach', 'IdGame', 'LicenseDate', 'idEmployeeData', and an empty column. The table contains 6 rows of data. To the right of the table is a sidebar with a tree view under 'CONNECTIONS' and 'BOOKMARKS'. The 'CONNECTIONS' section shows a tree structure: 'database \${wo...}' (expanded), 'newdatabase...' (expanded), 'Tables' (expanded), 'Coach' (expanded), and then 'IdCoach', 'IdGame', 'LicenseDate', 'idEmployee...', 'Employee_D...', and 'Game'.

IdCoach	IdGame	LicenseDate	idEmployeeData	
1	1	2023-01-09T09:41:0...	17	
2	2	2023-02-27T09:41:...	18	
3	3	2023-03-23T09:41:...	19	
4	4	2023-04-24T09:41:...	20	
5	5	2023-05-08T09:41:...	21	
6	6	2023-06-05T09:41:...	22	

Pour finir, j'utilise la commande « DETACH DATABASE »

```
DETACH DATABASE old_db;  
DETACH DATABASE new_db;
```

Cette commande me permet de détacher et de dissocier les deux bases de données. Elle déconnectera donc les deux bases.

Nous avons donc un nouveau schéma de base de données ainsi que toutes les informations qu'elle contient.

Les requêtes

La première

Il était ensuite demandé de faire plusieurs requêtes pour s'assurer que l'on a toujours accès aux données.

Il était demandé de faire 4 requêtes :

- Listez tous les tournois pour un nom de jeu donné
- Avec un nom de jeu donné, récupérez le salaire moyen des joueurs
- Liste tous les tournois par lieu
- Obtenez le nombre de joueurs par sexe

Si l'on prend la première requête :

« Listez tous les tournois pour un nom de jeu donné »

Voici la commande que j'ai utilisé pour faire cette requête :

```
SELECT * FROM Tournament INNER JOIN Game ON  
Tournament.IdGame= Game.IdGame  
WHERE Game.Name = 'SuperSmashBros';
```

J'ai donc sélectionné toute la table des tournois, ce qui me permet d'avoir la liste de tous les tournois.

J'ai ensuite fait la commande « INNER JOIN » qui me permet de lier des tables entre elles. Dans ce cas j'ai lié la table « Tournament » et la table « Game ».

Le « WHERE » me permet dans ce cas-là de donner une condition précise. Par exemple, quand ma colonne « Name » de ma table « Game » est égale à « SuperSmashBros ».

Cela affiche donc les tournois quand le nom de mon jeu est égal à « SuperSmashBros » dans ce cas-là.

On obtient donc ce résultat :

IdTournament	IdPlace	IdGame	Date	Duration	Name	
1	1	1	2022-12-01T09:00:0...	7	SuperSmashBros	
2	5	1	2022-12-01T09:00:0...	7	SuperSmashBros	
6	1	1	2022-08-20T09:00:...	2	SuperSmashBros	
7	5	1	2022-08-20T09:00:...	2	SuperSmashBros	

La deuxième

Si l'on prend maintenant la deuxième requête :
« Avec un nom de jeu donné, récupérez le salaire moyen des joueurs »

Pour récupérer le salaire moyen des joueurs, j'ai utilisé la commande AVG qui me permet de calculer la valeur moyenne de la colonne entrée entre parenthèse.

J'ai ensuite utilisé un alias « AS » qui me permet de renommer temporairement ma table.

Cela devient donc bien plus lisible.

J'ai donc récupéré le salaire moyen de ma table « Employee_Data »

J'ai continué par faire la commande « INNER JOIN » pour lier la table « Employee_Data » et « Player ».

```
SELECT AVG(Wage) AS 'Average wage' FROM Employee_Data
INNER JOIN Player
ON Employee_Data.IdEmployee = Player.IdEmployeeData
INNER JOIN Game ON Player.IdGame = Game.IdGame
WHERE Game.Name = 'Apex'
```

Le « WHERE » me permet la aussi de donner une condition précise. Par exemple, quand ma colonne « Name » de ma table « Game » est égale a « Apex ».

Cela affiche donc le salaire moyen des joueurs quand le nom du jeu est égal à « Apex » dans ce cas-là.

On obtient donc ce résultat :

Average wage
abc Filter...
155000

La troisième

Si l'on prend maintenant la troisième requête :
« Liste tous les tournois par lieu »

Pour cette requête, j'ai fait deux exemples, qui pour moi rentrer dans la consigne.

Voici le premier :

J'ai sélectionné tous les tournois. J'ai ensuite lié la table « Tournament » avec la table « Place ».

```
SELECT * FROM Tournament
INNER JOIN Place
ON Place.IdPlace = Tournament.IdPlace;
```

J'obtiens donc tous les tournois par tous les lieux :

IdTournament	IdPlace	IdGame	Date	Duration	Name	Address	City
abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...
1	1	1	2022-12-01T09:00:0...	7	home	1 Rue cassée	BrequinVille
2	5	1	2022-12-01T09:00:0...	7	home	1 Rue cassée	BrequinVille
3	2	2	2022-11-05T09:00:0...	12	Stadium	3 Rue des champs	ParisVille
4	3	4	2022-10-10T09:00:0...	2	Arenes	12 Parsec	YouVille
5	4	3	2022-09-15T09:00:...	3	Stade	33 Rue du stade	StadeVille
6	1	1	2022-08-20T09:00:...	2	home	1 Rue cassée	BrequinVille
7	5	1	2022-08-20T09:00:...	2	home	1 Rue cassée	BrequinVille

La deuxième méthode que j'ai utilisé pour cette même requête est quasiment la même que la première.

J'ai fait exactement la même chose en rajoutant un « WHERE ».

Cela m'a permis de donner une condition précise. Quand ma colonne « Name » de ma table « Place » est égale à « Stadium ».

```
SELECT * FROM Tournament
INNER JOIN Place
ON Place.IdPlace = Tournament.IdPlace
WHERE Place.Name = 'Stadium'
```

Cela affiche donc la liste des tournois quand le nom est « Stadium ».

La quatrième

On prend donc la dernière requête :

« Obtenez le nombre de joueurs par sexe »

Pour récupérer le nombre de joueur par sexe, il faut d'abord compter le nombre de joueurs. Pour cela, j'ai utilisé la commande COUNT qui me permet de compter le nombre de ligne dans une table.

J'ai donc récupéré le nombre de joueur.

J'ai ensuite utilisé un alias « AS » qui me permet de renommer temporairement ma table.

Cela devient donc bien plus lisible.

J'ai continué par faire la commande « INNER JOIN » pour lier la table « **Player** » et « **Employee_Data** »

J'ai ensuite fait un « GROUP BY » qui me permet de trier le résultat.

Dans ce cas je veux trier par « Gender » pour avoir le nombre de joueurs par sexe.

```
SELECT COUNT(IdPlayer) AS 'Nbr Player by gender' FROM Player
INNER JOIN Employee_Data
ON Player.IdEmployeeData = Employee_Data.IdEmployee
GROUP BY Gender
```

On obtient donc ce résultat :

IdTournament	IdPlace	IdGame	Date	Duration	Name	Address	City
abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...
3	2	2	2022-11-05T09:00:0...	12	Stadium	3 Rue des champs	ParisVille

Nbr Player by gender	
abc Filter...	
4	
5	
3	

Le script Bash

J'ai créé un script Bash pour gagner en facilité et en rapidité.

Le script Bash va me permettre de raccourcir de nombreuses tâches répétitives et de gagner du temps.

J'ai un script interroge l'utilisateur sur la tâche qu'il veut effectuer.

J'ai proposé d'exécuter tout ce qui était demandé dans le sujet.

L'utilisateur peut donc :

- 1- Créer une base de données vide
- 2- Créer les tables de la base de données
- 3- Migrer les données de l'ancienne base de données vers la nouvelle
- 4- Créer et migrer la base de données
- 5- Récupérer des informations précises dans la base de données
- 6- Supprimer toutes les informations "

Le menu

```
#!/bin/sh
echo "
Veillez choisir l'option que vous voulez exécuter:

1- Créer une base de données vide
2- Créer les tables de la base de données
3- Migrer les données de l'ancienne base de données vers la nouvelle
4- Créer et migrer la base de données
5- Récupérer des informations précises dans la base de données
6 - Supprimer toutes les informations "
read choix
if [ $choix = 1 ]
then
    sqlite3 newdatabase.db < createdatabase.sql
    echo " Vous venez de creer une base de données vide"
fi
if [ $choix = 2 ]
then
    sqlite3 newdatabase.db < createtable.sql
    echo " Vous venez de creer les tables de la base de données"
fi
if [ $choix = 3 ]
then
    sqlite3 newdatabase.db < migration.sql
    echo " Vous venez de migrer les informations de l'ancienne base de données vers la nouvelle"
fi
if [ $choix = 4 ]
then
    sqlite3 newdatabase.db < createtable.sql
    sqlite3 newdatabase.db < migration.sql
    echo "Vous venez de creer et remplir la base de données, en migrant les informations de l'ancienne base de données vers la nouvelle "
fi
if [ $choix = 5 ]
then
    echo "Voici qu'elques informations plus précises:"
    sqlite3 newdatabase.db < queries.sql
fi
if [ $choix = 6 ]
then
    sqlite3 newdatabase.db < deletetable.sql
    echo "Vous venez de supprimer la base de données ainsi que toutes les informations qu'elle possède"
fi
```

▲ **Projet sql** sh script.sh

Veillez choisir l'option que vous voulez exécuter:

- 1- Créer une base de données vide
- 2- Créer les tables de la base de données
- 3- Migrer les données de l'ancienne base de données vers la nouvelle
- 4- Créer et migrer la base de données
- 5- Récupérer des informations précises dans la base de données
- 6 - Supprimer toutes les informations

Le script Python

Introduction

Je me suis vite aperçu que script Bash avait des limites dans son utilisation. Les fonctionnalités que je voulais ajouter à mon script n'étais pas disponible. J'ai donc décidé de créer un script python.

Celui-ci me permet donc d'échanger avec l'utilisateur pour avoir des informations supplémentaires.

Je suis donc parti sur la même base que mon script Bash, il y a seulement la syntaxe qui change puisque le langage est différent.

Le menu

Voici donc le menu que j'ai créé :

```
def menu():
    print ("Veuillez choisir l'option que vous voulez exécuter:")
    print ("1- Créer une base de données vide")
    print ("2- Créer les tables de la base de données")
    print ("3- Migrer les données de l'ancienne base de données vers la nouvelle")
    print ("4- Créer et migrer la base de données")
    print ("5- Récupérer des informations précises dans la base de données")
    print ("6 - Supprimer toutes les informations")
    choice = input(" ")
    if choice == '1':
        name = input("veuillez choisir un nom pour créer la de base de données:")
        createdatabase(name)
        input("vous venez de creer une base de données vide")

    if choice == '2':
        name = input("veuillez choisir un nom de base de données:")
        print(" Le nom de votre base de données est :" + name)
        createschema(name)
        print(" Vous venez de creer les tables de la base de données")
    if choice == '3':
        newname = input ("veuillez choisir le nom de la base de données que vous voulez remplir ")
        oldname = input("veuillez choisir la base de données depuis laquelle prendre les données")
        migration(newname, oldname)

        print("Vous venez de migrer les informations de l'ancienne base de données vers la nouvelle")
    if choice == '4':
        name = input("veuillez choisir un nom de base de données:")
        print(" Le nom de votre base de données est :" + name)
        createschema(name)
        newname = input ("veuillez choisir le nom de la base de données que vous voulez remplir ")
        oldname = input("veuillez choisir la base de données depuis laquelle prendre les données")
        migration(newname, oldname)
        print("Vous venez de creer et remplir la base de données, en migrant les informations de l'ancienne base de données vers la nouvelle ")
    if choice == '5':
        print("Voici quelques informations plus précises:")
        recuperation()
    if choice == '6':
        delete()
        print("Vous venez de supprimer la base de données ainsi que toutes les informations qu'elle possède")
```

J'ai donc décidé d'améliorer les fonctionnalités déjà existantes en les modifiant. J'ai par exemple réussi à prendre les réponses de l'utilisateur et à m'en servir dans mon code. Cela permet de personnaliser mon programme.

L'utilisateur peut modifier celui-ci en fonction de ses besoins. Il peut donc changer un grand nombre de paramètres sans toucher une ligne de code.

L'utilisateur gagne donc du temps.

Choix de nom

Grâce à l'automatisation que j'ai effectué en python je peux maintenant demander à l'utilisateur de choisir le nom qu'il souhaite donner à sa base de données.

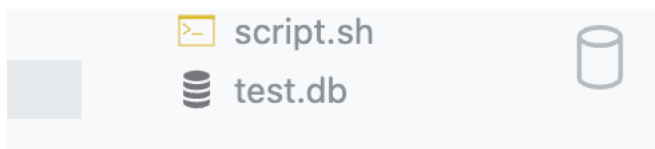
Il peut donc choisir le nom qui souhaite au moment de la création.

Une phrase s'affiche en lui demandant le nom qu'il souhaite mettre.

La base de données se crée donc avec le nom que l'utilisateur vient de rentrer.

Et une autre phrase s'affiche pour lui confirmer que la base de données a bien été créée.

```
o Projet sql python3 migration.py
Veillez choisir l'option que vous voulez exécuter:
1- Créer une base de données vide
2- Créer les tables de la base de données
3- Migrer les données de l'ancienne base de données vers la nouvelle
4- Créer et migrer la base de données
5- Récupérer des informations précises dans la base de données
6 - Supprimer toutes les informations
1
veuillez choisir un nom pour créer la de base de données:test.db
vous venez de creer une base de données vide
```



On remarque donc bien le fichier « test.db » qui vient de se créer.

En effet, il peut aussi choisir dans quelle base insérer les tables ainsi lors de la migration, la base de départ ainsi que la base finale.

Pour récupérer l'information entrée dans le terminal par l'utilisateur, j'ai utilisé des inputs.

Un input renvoie une valeur dont le type correspond à ce que l'utilisateur a entré.

Dans notre exemple, la variable « Name », « newname » ou même « oldname » contiendra donc une chaîne de caractères.

```

choice = input(" ")
if choice == '1':
    name = input("veuillez choisir un nom pour créer la de base de données:")
    createdatabase(name)
    input("vous venez de creer une base de données vide")

if choice == '2':
    name = input("veuillez choisir un nom de base de données:")
    print(" Le nom de votre base de données est :" + name)
    createschema(name)
    print(" Vous venez de creer les tables de la  base de données")
if choice == '3':
    newname = input ("veuillez chisir le nom de la base de données que vous voulez remplir ")
    oldname = input("veuillez choisir la base de données depuis laquelle prendre les données")
    migration(newname, oldname)

```

Pour intégrer ses variables dans mon script python j'ai utilisé des accolades. Elles me permettent donc de pouvoir intégrer mes variables python dans mon script SQL. Les variables me permettent donc de choisir ci-dessous par exemple le nom de ma base de données de base et le nom dans laquelle je veux migrer les informations.

```

def migration(newname, oldname):
    connexion= sqlite3.connect(newname)
    c = connexion.cursor()
    c.executescript('''
        ATTACH DATABASE '{}' AS old_db;
        ATTACH DATABASE '{}' AS new_db;

```

Et le point format qui est en lien avec les accolades me permet donc d'insérer les variables python dans mon script SQL. Cela permet d'adapter le script pour qu'il fonctionne.

```

DETACH DATABASE old_db;
DETACH DATABASE new_db;
'''
.format(oldname, newname)
)

```

Création base de données

J'ai fait en sorte que l'utilisateur puisse choisir l'option de créer sa base de données avec le nom qu'il souhaite.

J'ai donc inséré mon script SQL qui me permet de créer ma base de données. J'ai bien-sûr fait les modifications nécessaires pour que cela fonctionne dans mon script python.

```
def createdatabase(name):  
    connexion= sqlite3.connect(name)  
    c = connexion.cursor()  
    c.executescript(''' CREATE DATABASE IF NOT EXISTS {}  
        '''.format(name)  
    )  
    connexion.commit()  
    c.close()  
    connexion.close()
```

```
if choice == '1':  
    name = input("veuillez choisir un nom pour créer la de base de données:")  
    createdatabase(name)  
    input("vous venez de creer une base de données vide")
```

