SEILER
Luc

MIGRATION DE BASE DE DONNÉES SQLITE

LIEN DU REPO : https://ytrack.learn.ynov.com/git/LSEILER/Project_SQL

IDE utilisés :

- DataGrip pour les scripts .sql et tests avec SQLite
- Visual Studio Code pour Python et test lancement des .sql

Pour faire la migration, j'ai d'abord fait tous les scripts .sql et les tests à la main (avec les tables une par une), avant de tout automatiser avec un autre script en Python.

J'ai commencé par télécharger la base de données « tpb2.sql » et je l'ai mise dans un dossier appelé « backup », si jamais des erreurs se produisent lors du projet.

1-CREATION DES NOUVELLES TABLES

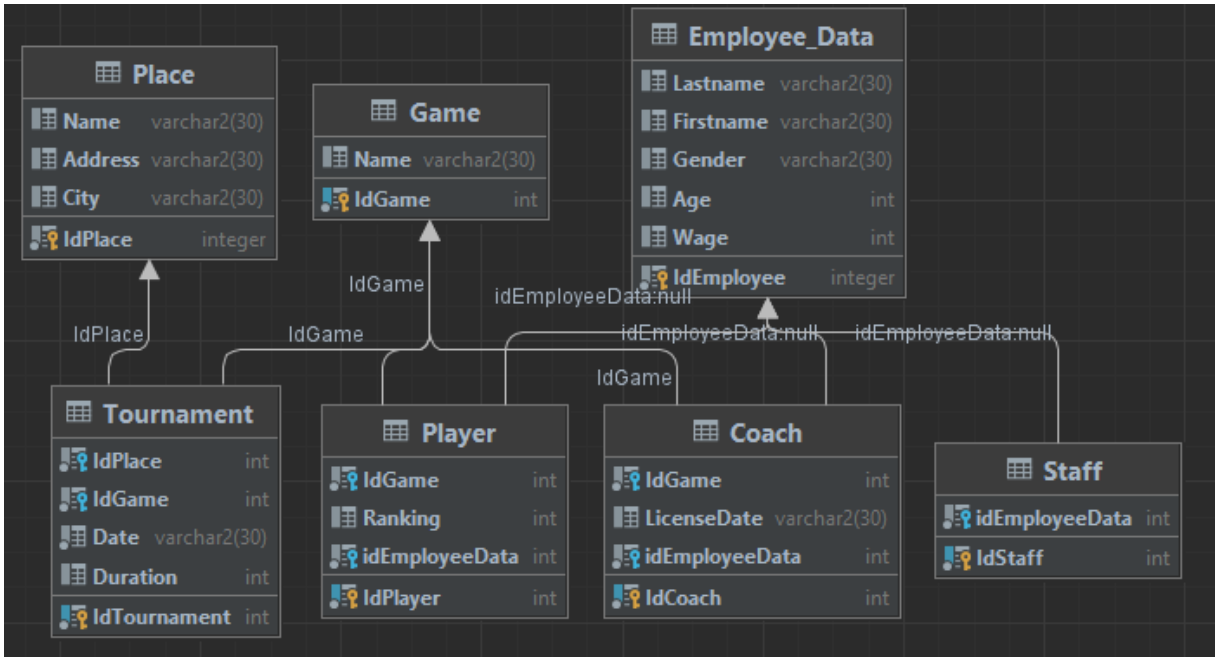
J'ai ensuite créé la base de données appelée ici « teamLikwid.db », puis j'ai créé un fichier .sql, qui va contenir le code permettant la création du nouveau schéma voulu.

Comme par exemple la nouvelle table Tournament :

```
CREATE TABLE Tournament
(
    IdTournament INT PRIMARY KEY NOT NULL,
    IdPlace INT NOT NULL,
    IdGame INT NOT NULL,
    Date VARCHAR2(30) NOT NULL,
    Duration INT,
    FOREIGN KEY (IdPlace) REFERENCES Place(IdPlace),
    FOREIGN KEY (IdGame) REFERENCES Game(IdGame)
);
```

J'ai choisi de mettre les clés IdPlace et IdEmployee en « AUTOINCREMENT », car elles n'existent pas dans le nouveau schéma. Il faut qu'elles puissent s'incrémenter toutes seules lors de leur création.

Voici le résultat sur le diagramme de DataGrip après la création de toutes les tables :



2-LA MIGRATION

La migration entière est stockée dans un seul script .sql.

D'abord, nous devons attacher la première base de données « tpb2.db » au script pour pouvoir travailler avec (en renseignant le chemin entier de l'ancienne).

J'ai choisi de commencer par la migration de la table 'Game', car elle a la même structure que l'ancienne, en faisant des INSERT et SELECT.

```
ATTACH DATABASE 'C:\Users\lucky\Ynov\Projets\Project_SQL\tpb2.db' AS old;

/*INSERT INTO Tournament (IdTournament, IdGame, Date, Duration)
SELECT idTournament, IdGame, Date, Duration
FROM old.tournament;*/

/*MIGRATE GAME TABLE*/
INSERT OR IGNORE INTO Game(IdGame, Name)
SELECT IdGame, Name
FROM old.game
```

Avec cette requête, les données ont été migrées, le schéma était le même.

	IdGame	Name
1	1	SuperSmashBros
2	2	Tekken 7
3	3	StreetFighter 6
4	4	LeagueOfLegends
5	5	CS:GO
6	6	Apex

En deuxième, j'ai choisi la table « Place », car la table « Tournament » a besoin d'une clé secondaire « IdPlace », qui n'existe pas dans la nouvelle base. Il y aurait eu des erreurs d'ID si l'on fait la table Tournament. Il faut d'abord créer la clé primaire de Place (en AUTOINCREMENT), pour qu'on puisse l'insérer dans Tournament par la suite, et l'utiliser pour la migration d'autres tables.

```sql
/*MIGRATE PLAGE TABLE*/
INSERT OR IGNORE INTO Place (Name, Address, City)
SELECT PlaceName, Address, City
FROM old.tournament
GROUP BY Address;
```

Lors de la migration de la table « Place », l'IdPlace va s'incrémenter tout seul. Et comme plusieurs adresses étaient utilisées dans Tournament, j'ai dû les regrouper « GROUP BY » pour ne pas avoir de doublons.

	IdPlace	Name	Address	City
1	1	home	1 Rue cassée	BrequinVille
2	2	Arenes	12 Parsec	YouVille
3	3	Stadium	3 Rue des champs	ParisVille
4	4	Stade	33 Rue du stade	StadeVille

Comme les clés secondaires dont la table « Tournament » a besoin (IdPlace et IdGame) sont créées, on peut maintenant migrer cette table sans erreurs d'Id.

Les données à recupérer de « Tournament » étant dans plusieurs tables, il a fallu utiliser un « JOIN » (ici l'IdPlace).

```sql
/*MIGRATION TOURNAMENT TABLE*/
INSERT OR IGNORE INTO Tournament
    (
    IdTournament,
    IdPlace,
    IdGame,
    Date,
    Duration
    )
SELECT IdTournament, IdPlace, IdGame, Date, Duration
FROM old.tournament
INNER JOIN Place p
ON p.Name = old.tournament.PlaceName;
```

Maintenant, on peut faire les autres tables. Mais il est préférable de commencer par la table Employee_Data, afin de créer l'ID et ne pas avoir de problèmes (dans la même logique que la table Tounament, comme c'est cette table qui contient toutes les informations sur le Staff, Player et Coach).

Pour ça, il faut migrer les données des 3 tables Staff, Player et Coach une par une :

```sql
/*MIGRATION FROM PLAYER TO EMPOYEE*/
INSERT OR IGNORE INTO Employee_Data
    (
    Lastname,
    Firstname,
    Gender,
    Age,
    Wage
    )
SELECT DISTINCT Lastname, Firstname, Gender, Age, Wage
FROM old.Player
GROUP BY Lastname, Firstname, Gender, Age, Wage;

/*MIGRATION FROM COACH TO EMPLOYEE*/
INSERT OR IGNORE INTO Employee_Data
    (
    Lastname,
    Firstname,
    Gender,
    Age,
    Wage
    )
SELECT Lastname, Firstname, Gender, Age, Wage
FROM old.Coach
GROUP BY Lastname, Firstname, Gender, Age, Wage;

/*MIGRATION FROM STAFF TO EMPLOYEE*/
INSERT OR IGNORE INTO Employee_Data
    (
    Lastname,
    Firstname,
    Gender,
    Age,
    Wage
    )
SELECT Lastname, Firstname, Gender, Age, Wage
FROM old.Staff
GROUP BY Lastname, Firstname, Gender, Age, Wage;
```

Voici le résultat près la migration des trois tables dans Employee_Data :

	IdEmployee	Lastname	Firstname	Gender	Age	Wage
1	1	Boyer	Callie	male	29	220000
2	2	Candice	Nuts	female	26	100000
3	3	Connor	Corey	other	29	240000
4	4	Hahn	Michael	other	27	110000
5	5	Hood	Wendy	female	24	240000
6	6	Kop	Vlad	male	22	150000
7	7	Lowery	Alastair	male	22	160000
8	8	Pacheco	Emma	female	26	130000
9	9	Roman	Jackson	male	20	110000
10	10	Sykes	Roisin	other	17	160000
11	11	Tour	Ro	male	21	120000
12	12	Ward	Suzanne	female	23	260000
13	13	Atkins	Karl	male	27	270000
14	14	Barrera	Jessica	female	34	340000
15	15	Camacho	Trinity	female	26	260000
16	16	Figueroa	Arjun	male	28	280000
17	17	Noble	Ricky	male	28	280000
18	18	Wood	Jimmy	male	33	330000
19	19	Abdirahman	Turner	female	38	100000
20	20	Aidan	Shannon	female	39	150000
21	21	Crystal	Taylor	male	43	250000
22	22	Martha	Howell	female	26	190000

Les données de tous les employés sont maintenant dans une seule table.

Ensuite, on peut migrer les données de ces trois tables, sans avoir de problèmes pour l'ID de l'employer, une par une aussi. Pour cela, j'ai migré les données de la table « Coach » pas exemple, en déterminant si le nom, je prénom, et l'âge de l'ancienne table sont trouvées dans la table «Employee_Data ».

Et voici le résultat de la migration de chacune des tables.

	IdStaff	idEmployeeData
1	1	21
2	2	19
3	3	20
4	4	22

	IdPlayer	IdGame	Ranking	idEmployeeData
1	1	1	68	11
2	2	6	13	6
3	3	4	43	2
4	4	4	17	4
5	5	5	50	5
6	6	2	205	9
7	7	3	288	3
8	8	2	216	10
9	9	4	218	1
10	10	5	85	8
11	11	1	273	12
12	12	6	192	7

	IdCoach	IdGame	LicenseDate	idEmployeeData
1	1	1	2023-01-09T09:41:09Z	15
2	2	2	2023-02-27T09:41:09Z	13
3	3	3	2023-03-23T09:41:09Z	18
4	4	4	2023-04-24T09:41:09Z	14
5	5	5	2023-05-08T09:41:09Z	16
6	6	6	2023-06-05T09:41:09Z	17

Une fois toutes les données migrées, cela veut dire que le fichier de migration .sql fonctionne. Ensuite, j'ai fait un script Python qui va automatiser les tâches (création de la BDD, des tables, migration…).

3- LE SCRIPT PYTHON

Choix de Python car il est connu pour le travail avec des bases de données. Il aura pour but d'automatiser toute la migration suivant ces étapes :

J'ai créé les différentes étapes une par une, découpées en fonctions.

1- Le backup : créer un backup de la base de données source, en travaillant avec les librairies « os » et « shutil ». La fonction va créer un nouveau dossier « old-database-backup », copier le fichier .db de l'ancienne base et le mettre dans ce dossier.
2- 'a voir si on garde) La création de la nouvelle base de donnée, en créant une nouvelle connexion avec sqlite3.
3- Créer les différentes tables afin d'avoir le même schéma souhaité, en lisant et en exécutant le tout premier script .sql.
4- La migration de données, en exécutant aussi un autre script .sql, qui va migrer les données de la base source vers la nouvelle de l'étape 2.
5- Les requêtes. J'ai mis les 4 requêtes finales dans différents fichiers (eux-mêmes dans un dossier à part). La fonction va donc exécuter la requêtes SQL de chacun des fichier se trouvant dans le dossier. Cela permettra de s'adapter, si d'autres tests sont ajoutés.

(Plus de détails du script Python dans le README de git)

4- LES 4 REQUETES DE TEST

Voici les 4 requêtes demandées ainsi que le résultat obtenu :

"List every tournament for a given game name"

```
SELECT IdTournament,
       g.Name,
       Date,
       Duration
FROM Tournament
INNER JOIN Game g
    ON g.idGame = tournament.IdGame
```

	IdTournament	Name	Date	Duration
1	1	SuperSmashBros	2022-12-01T09:00:00Z	7
2	2	Tekken 7	2022-11-05T09:00:00Z	12
3	3	LeagueOfLegends	2022-10-10T09:00:00Z	2
4	4	StreetFighter 6	2022-09-15T09:00:00Z	3
5	5	SuperSmashBros	2022-08-20T09:00:00Z	2

"Given a game name, retrieve the average wage of players"

```
SELECT Name, AVG(Wage) AS 'AverageWage'
FROM Employee_Data
INNER JOIN Player P
    ON Employee_Data.IdEmployee = P.idEmployeeData
INNER JOIN Game G
    ON G.IdGame = P.IdGame
GROUP BY Name
```

	Name	AverageWage
1	Apex	155000
2	CS:GO	185000
3	LeagueOfLegends	143333.33333333334
4	StreetFighter 6	240000
5	SuperSmashBros	190000
6	Tekken 7	135000

“List all tournament by place”

```sql
SELECT IdTournament,
       P.City,
       P.Address,
       Date,
       Duration
FROM Tournament
INNER JOIN Place P
    ON P.IdPlace = Tournament.IdPlace
```

	IdTournament	City	Address	Date	Duration
1	1	BrequinVille	1 Rue cassée	2022-12-01T09:00:00Z	7
2	2	ParisVille	3 Rue des champs	2022-11-05T09:00:00Z	12
3	3	YouVille	12 Parsec	2022-10-10T09:00:00Z	2
4	4	StadeVille	33 Rue du stade	2022-09-15T09:00:00Z	3
5	5	BrequinVille	1 Rue cassée	2022-08-20T09:00:00Z	2

“Get the number of players by gender”

```sql
SELECT Gender, COUNT(IdPlayer) AS 'NbrPlayers'
FROM Player
INNER JOIN Employee_Data ED
    ON ED.IdEmployee = Player.idEmployeeData
GROUP BY Gender;
```

	Gender	NbrPlayers
1	female	4
2	male	5
3	other	3