LES BASES DE DONNEES

Sources : Wikipedia, cours de F. Duchateau, LIRIS, S. Kaci Université Montpellier 2, B. Mermet, Université le Havre,

1. Le modèle relationnel

   Les informations et les données sont collectées depuis longtemps par les états, ou par diverses organisations. Le stockage des données a évolué des fiches papier aux fiches cartonnées, puis aux fichiers informatiques, et enfin aux bases de données. Comment faire pour construire un bon modèle pour une base de données ? Modéliser, c'est définir un monde abstrait qui coïncide avec une partie de l'apparence du réel. Bien modéliser, c'est faire que ce monde abstrait soit structuré, performant, et accessible facilement. Dans ce paragraphe, on va se concentrer sur le niveau logique de la représentation des données : le modèle relationnel.

   a. Un exemple

      Un club de sports dispose des données suivantes :

Prénom	Nom	Sport	Niveau	Adresse	Date dernière compétition	Classement dernière compétition	Code sport
Colette	Mavallée	Pétanque	débutant	Montpellier	aucune	0	P1
Berthe	Mavallée	Pétanque	national	Montpellier	01/01/2020	1	P1
Janine	Tutor	Fléchettes	Régional	Saint-Clément	15/12/2019	5	F2
Amélie	Diodeuor	Aquaponey	Départemental	Saint-Gély	25/08/2020	2	A2
Hans	Kimkonzern	Fléchettes	Débutant	Saint-Gély	aucune	0	F2
Janine	Tutor	Aquaponey	Confirmé	Saint-Clément	31/05/2020	8	A2

   b. Premier modèle : fichier informatique de type tableur (Open Office, Excel, etc.)

      Dans le modèle relationnel, ces données sont alors modélisées sous la forme de la *relation* (dite aussi *table*)

      Club = { *Prénom* `String` , *Nom* `String` , *Sport* `String` , *Adresse* `String` , *DateC* `Date` , *Classement* `Int` , *Code_Sport* `String` }

      On garde les données exactement comme elles sont présentées ci-dessus. Les anomalies suivantes peuvent se produire :

      - Redondance lorsqu'un homonyme s'inscrit.
      - Modification : si le code sport de Pétanque devient P3 en première ligne, alors il est nécessaire de modifier d'autres lignes
      - Suppression : des informations dépendant d'autres informations. Par exemple, supprimer un sport demande aussi de supprimer son code.
      - Insertion : insérer un nouvel enregistrement demande la connaissance de toutes les informations qui lui sont liées. Un nouvel inscrit doit immédiatement choisir un sport.

   c. Deuxième modèle : trois relations

      La base de données est constituée de trois relations *Inscrit* , *Sport* , *Compétition*.

      Les schémas de ces relation s sont

      Inscrit = { *Identifiant_I* `Int`, *Prénom* `String` , *Nom* `String` , *Adresse* `String` }

      Sport = { *Code_Sport* `String`, *Sport* `String` , *Horaire* `String`}

      Pratiquant = { *Identifiant_I* `Int`, *Code_Sport* `String` , *Niveau* `String` }

      Compétition = { *Identifiant_I* `Int`, , *DateC* `Date` , *Code_Sport* `String` , *Classement* `Int` }

Cours de Frédéric Mandon sous licence Creative Commons BY NC SA, http://creativecommons.org/licenses/by-nc-sa/3.0/fr/.

Les défauts identifiés ci-dessus disparaissent au moins partiellement, et la gestion globale des données est bien plus simple : plus de redondance, des tables plus simples, insertion/modification/suppression plus simples également.

d. Les différents niveaux de la conception d'une base de données.
Le modèle utilisé pour la base de données est indépendant de l'implémentation qui en est faite en machine. L'implémentation va utiliser une structure de données qui peut être variable pour un même modèle. On utilise presque systématiquement pour l'implémentation d'une base de données un système de gestion de base de données (SGBD, cf. ci-après pour plus de détails).
Plus précisément, il y a trois étapes dans la modélisation :
- Le niveau conceptuel, fortement abstrait, indépendant des SGBD. On utilise très souvent les diagrammes entité/association (pas très compliqué, mais pas au programme de terminale, on ne peut pas tout faire)
- Le niveau logique, qui se rapproche du type de SGBD choisi. Ici on se limite au modèle relationnel.
    - On détermine les entités (*exemple* : *Inscrit, Sport, Compétition*)
    - On détermine les schémas des entités : quels en sont les attributs, de quel type / dans quel domaine est chacun de ces attributs ?
    - On détermine les contraintes sur la base de données, c'est-à-dire l'ensemble des propriétés logiques qui doivent être respectées (*exemple* : on ne peut pas inscrire à une *Compétition* un individu non *Inscrit*)
- Le niveau physique : la structure de données et/ou le modèle de création des données pour un SGBD particulier. On l'aborde ci-après avec la requête SQL CREATE et la traduction des contraintes

e. Les contraintes
Les contraintes d'intégrité sont des règles permettant de garantir la cohérence des données lors de la mise à jour de la base.

i. Contraintes d'entité
La contrainte d'entité permet de s'assurer que chaque enregistrement de la relation est unique. Comme on l'a vu ci-dessus, la donnée d'un nom et prénom ne suffit pas vu l'existence d'homonymes. Par ailleurs rajouter l'adresse en plus n'est pas une solution pratique, vu que la personne peut déménager. C'est pour cela que l'on rajoute parfois un identifiant unique : numéro de sécurité sociale, numéro de candidat pour le bac.
Remarques :
- l'existence de numéros distincts pour la même personne, suivant les domaines, se justifie pour des raisons sociétales. En effet un unique numéro pour tout permettrait un croisement de fichiers très intrusif pour la vie privée (personne n'a besoin de savoir que M. X, numéro de sécurité sociale *xx*, a profité de son congé maladie pour exploser son record à Candy Crush où il serait identifié sous le même numéro)
- Un couple, plus généralement un *n*-uplet, peut servir d'identifiant unique. Si l'on crée une base de données des élèves de terminale NSI du Lycée Jean Jaurès, le couple (*Nom* , *Prénom*) vous identifie bien individuellement.
- En théorie, toutes les relations doivent avoir une clé primaire. En pratique, il arrive que certaines n'en aient pas, on crée dans ce cas un index pour rechercher un enregistrement dans la table (recherche dichotomique).

L'attribut ou l'ensemble d'attributs permettant l'identification de l'enregistrement est appelé **clé primaire** de la relation, et est noté dans le schéma par soulignement
Exemple :

Cours de Frédéric Mandon sous licence Creative Commons BY NC SA, http://creativecommons.org/licenses/by-nc-sa/3.0/fr/.

Inscrit = { *Identifiant_I* `Int`, *Prénom* `String` , *Nom* `String` , *Adresse* `String` }
Sport = { *Code_Sport* `String`, *Sport* `String` , *Horaire* `String`}
Pratiquant = { *Identifiant_I* `Int`, *Code_Sport* `String` , *Niveau* `String` }
Compétition = { *Identifiant_I* `Int`, , *DateC* `Date` , *Code_Sport* `String` , *Classement* `Int` }

ii. Contraintes de référence
Les relations *Pratiquant* et *Compétition* font référence à des attributs d'autres tables. Lorsqu'on enregistre une nouvelle ligne dans ces tables, on vérifie que les attributs référencés existent : ce sont des **clés étrangères**. On les note dans le schéma par un soulignement en pointillés.
Pratiquant = { *Identifiant_I* `Int`, *Code_Sport* `String` , *Niveau* `String` }

Compétition = { *Identifiant_I* `Int`, , *DateC* `Date` , *Code_Sport* `String` , *Classement* `Int` }

Remarque : lors de la création de la base de données, on crée d'abord les tables sans clés étrangères, puis les tables avec clés étrangères. En effet le SGBD vérifie la cohérence au fur et à mesure de la construction (et même s'il ne le fait pas, il vaut mieux être cohérent soi-même…).

iii. Contraintes de domaine
Les contraintes de domaine concernent les attributs. On les a exprimées ici sous forme de type « générique », `String`, `Int` ou `Date`. On dispose d'autres types, `Float`, `Boolean`, `Time`, … qui dépendent en fait du SGB utilisé. On verra dans le TD sur SQL qu'en effet ces contraintes s'expriment légèrement différemment. On peut également préciser qu'une valeur doit être non nulle. Par exemple lorsque l'on crée un nouvel adhérent, son adresse pourrait ne pas être immédiatement renseignée, mais pas ses nom et prénom. rajouter d'autres spécificités, notamment avec les contraintes utilisateur.

iv. Contraintes utilisateur
Ce sont des contraintes spécifiques qui ne rentrent pas dans les catégories précédentes. Par exemple, on peut préciser que le niveau est forcément dans la liste (débutant, confirmé, départemental, régional, national, international, intergalactique), qu'un numéro de téléphone portable débute par +33 suivi par un 6 ou un 7, puis est suivi de 8 chiffres, qu'un émail est de la forme : une chaîne de caractères sans @, un seul caractère @, au moins un caractère, un point, et encore au moins un autre caractère.

f. Exercices
- Aller sur la page d'accueil de votre site préféré où vous avez un identifiant de connexion, et proposez un modèle relationnel correspondant à l'usage de ce site (vous pouvez, voire vous serez obligé, de pas mal simplifier !)
- Supposons que vous regardiez des séries (supposition gratuite, je suis certain que vous préférez travailler). Créer un modèle relationnel pour faire une base de données concernant vos séries préférées.
- Exercices p 295-296 du livre

2. Système de gestion de bases de données (SGBD)
a. Historique
Avant les SGBD, les données étaient enregistrées dans des fichiers informatiques, comme celui présenté dans le premier modèle ci-dessus (au 1a et b). L'expression même de « base de données » n'était pas inventé. La gestion des données était rendue complexe par :
- manque de sécurité, une personne ayant accès à tout le fichier et non seulement une partie
- incohérence : comment vérifier lorsque l'on modifie une ligne que certaines de ces données ne figurent pas dans d'autres lignes

Cours de Frédéric Mandon sous licence Creative Commons BY NC SA, http://creativecommons.org/licenses/by-nc-sa/3.0/fr/.

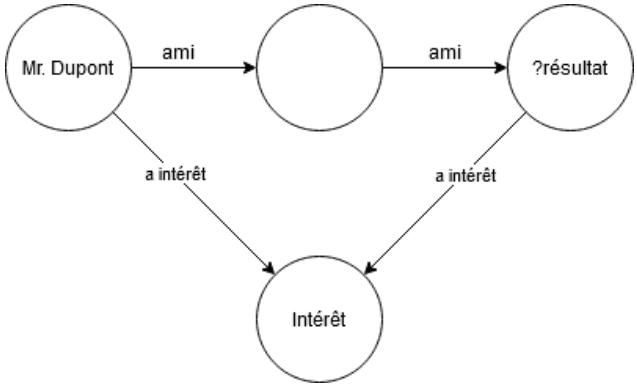
- redondance des information (coût de stockage important, à une époque où la mémoire était chère)
- difficulté d'accès aux données : nécessité d'écrire un nouveau programme pour les requêtes non prévues.
- Gestion de l'intégrité à intégrer dans chaque programme
- Concurrence d'accès : si deux modifications sont demandées en même temps, seule l'une d'entre elle était prise en compte.

La notion de base de données tente de minimiser ces problèmes, et les logiciels Systèmes de Gestion de Base de Données suivent cette philosophie. L'idée des bases de données a été lancée en 1960 dans le cadre du programme Apollo.
Les premiers SGBD proposaient, dans les années 65-70, des modèles hiérarchiques. Les données étaient enregistrées dans des structures ressemblant aux dictionnaires de Python. Ces données étaient liées entre elles par des pointeurs, formant des listes ou des arbres (d'où la hiérarchie). Cette organisation commence à se détacher du format des fichiers.
Le modèle relationnel apparaît en 1970, inventé par Edgar frank Codd. Suit immédiatement la création du langage SQL par IBM. La première version commerciale de SQL sort en 1979.
S'ensuit une généalogie compliquée (voir https://fadace.developpez.com/sgbdcmp/story/). Cette généalogie complexe fait que les différentes versions de SQL ne sont pas forcément compatibles entre elles.

Dans les années 1990 et 2000 apparaissent d'autres modèles, de type objet et objet-relationnel, XML pour des données très variées (documents par exemple), NoSQL (très rapide mais ne vérifiant pas certains critères) et NewSQL (combinant rapidité et respect des critères ACID, cf. ci-après).
Plus en détail, le XML est une relique du web 1.0 des années 90, toujours très utile pour les petites bases données avec des échanges normalisés sur le web.
Le modèle objet permet, comme son nom l'indique, d'utiliser des objets comme données. On peut utiliser des versions de SQL pour en faire. Dans ce cas, le SGBD traduira le modèle objet en modèle relationnel en fait : comme précisé ci-après, l'utilisateur du SGBD n'a pas à se soucier de la manière dont sont traduites ses requêtes.
Le NoSQL permet d'éviter des requêtes SQL complexes.
Prenons l'exemple d'un réseau social. Monsieur Dupont se sent seul donc il cherche des amis, mais il ne veut pas n'importe qui comme ami, il cherche des amis d'amis qui ont au moins un intérêt en commun avec lui.
Voici la requête correspondante en pseudo SQL :

```
SELECT ami_resultat.*
FROM personnes p_dupont
INNER JOIN a_interet interet_dupon ON p_dupont.id=interet_dupont.personne_id
INNER JOIN a_ami a_ami1 ON p_dupont.id=a_ami1.personne_id
INNER JOIN personnes ami1 ON a_ami1.ami_id=ami1.id
INNER JOIN a_ami a_ami2 ON ami1.personne_id=a_ami2.personne_id
INNER JOIN personnes ami_resultat ON a_ami2.ami_id=ami_resultat.id
WHERE p_dupont.nom ='Monsieur Dupont'
AND EXISTS(
   SELECT *
   FROM a_interet a_interet_ami_res
   WHERE a_interet_ami_res.personne_id=ami_resultat.id
   AND a_interet_ami_res.interet_id=interet_dupont.interet_id
)
```


Cours de Frédéric Mandon sous licence Creative Commons BY NC SA, http://creativecommons.org/licenses/by-nc-sa/3.0/fr/.



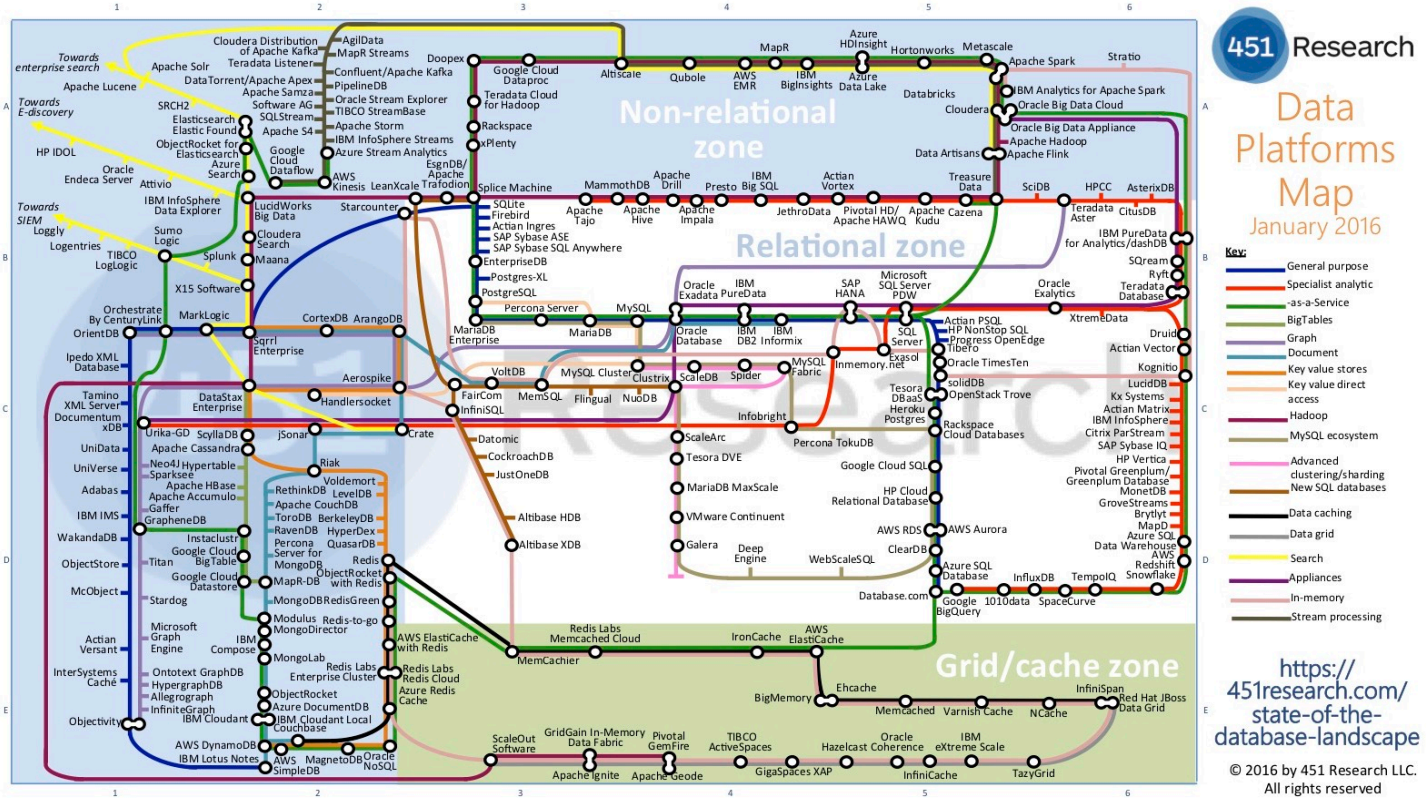
Et voici la requête en pseudo sparql (langage de requêtage sur des graphes inspiré de la syntaxe SQL). On se contente de « suivre les flèches » du graphe :

```
SELECT ?ami_resultat
WHERE ?Dupont nom "Monsieur Dupont"
   ?Dupont a_interet ?interet
   ?Dupont a_ami ?ami1
   ?ami1 a_ami ?ami_resultat
   ?ami_resultat a_interet ?interet
```

Il n'y a pas photo sur la complexité respective des deux requête (et encore il manque un DISTINC quelque part dans la requête SQL)

Pour autant qu'on puisse le savoir, les chiffres étant difficiles à trouver, Oracle domine le marché des SGBD actuellement (48% du parc en 2019), suivi par MySQL, qui est un produit… d'Oracle à l'origine ! Et qui est en progression rapide, très demandé. Le troisième sur le podium est Microsoft SQL Server. Ces logiciels sont conçus pour gérer de très grosses bases de données, avec une architecture client/serveur. Les deux premiers sont également chers et réservés aux grandes organisations.

Le schéma suivant donne la filiation des différents SGBD en 2016.



Cours de Frédéric Mandon sous licence Creative Commons BY NC SA, http://creativecommons.org/licenses/by-nc-sa/3.0/fr/.

b. Fonctionnalités

Un SGBD doit permettre la création, la modification et l'accès aux données de manière sécurisée. Plus précisément :

- Création de la base de données
- Puis création des relations (tables) de la base, du moins pour un SGBD relationnel comme on en utilise dans ce cours
  - Avec respect des attributs et contraintes d'intégrité
  - Avec respect des contraintes de cohérence
- Suppression de données dans les tables, ou de tables en entier
- Mise à jour des données
- Interrogation des données avec un langage spécifique de requêtes, SQL étant le plus connu
- Sécurité des données :
  - tous les utilisateurs n'ont pas accès à toutes les données
  - respect des règles de cohérence, notamment lorsque plusieurs requêtes sont effectuées à la suite
  - pas de modification simultanée des mêmes données par plusieurs utilisateurs
  - conservation des données en cas de panne matérielle
- Tout ceci en étant transparent vis-à-vis de la structure des données. L'utilisateur –ou le programme- faisant une requête n'a pas besoin de savoir comment sont organisées les données en mémoire.
- *Remarque* : pour cette raison, SQL est un langage déclaratif (du moins en partie). On ne dit rien sur la manière dont le résultat doit être obtenu, contrairement à ce que l'on fait en Python par exemple. On *déclare* simplement : « donne-moi toutes les données qui vérifient… »

Une remarque sur les transactions.

On reprend l'exemple d'introduction. Lorsqu'un des sports est supprimé, il faut également supprimer toutes les entrées de la table « pratiquant », vu les contraintes de clé étrangère. Imaginons que suite à une panne (matérielle ou logicielle) seule la première requête soit exécutée : la base de données se retrouve dans un état incohérent. Utiliser une **transaction**, qui est un type particulier de requête, permet de s'assurer que toutes les requêtes sont exécutées avant de changer l'état de la base de données, ce qui en assure la cohérence. Pour plus de détail voir p 331-334 du livre.

c. Propriétés ACID

Source Wikipédia

Atomicité, Cohérence, Isolation, Durabilité : ce sont les propriétés qui garantissent qu'une transaction informatique est exécutée de façon fiable

- Atomicité : cette propriété assure qu'une transaction se fait au complet ou pas du tout, cf. la remarque précédente
- Cohérence : cette propriété assure que chaque transaction amènera le système d'un état valide à un autre état valide. En particulier les contraintes d'intégrité doivent être vérifiées (mais pas seulement)
- Isolation : les transactions s'exécutent comme si elles étaient seules sur le système. Si par exemple une transaction complexe T1 s'exécute en même temps qu'un autre transaction complexe T2, alors T1 ne peut pas accéder à un état intermédiaire de T2. T1 et T2 doivent produire le même résultat qu'elles soient exécutées simultanément ou successivement.
- Durabilité : les transactions sont « gravées dans le marbre ». Une fois qu'une transaction a eu lieu, la base de données reste dans l'état modifié, même suite à une panne d'électricité.
  Remarque du professeur : jusqu'à présent, la durabilité maximale des supports de sauvegarde en informatique est de 10 ans (contre 400 sans trop de difficultés pour le papier, ne parlons pas du « marbre gravé »). Ce critère de durabilité est donc à l'échelle de l'utilisation des bases de données, et non dans un temps historique.

Cours de Frédéric Mandon sous licence Creative Commons BY NC SA, http://creativecommons.org/licenses/by-nc-sa/3.0/fr/.

d. Exercices SGBD p 342

3. Le langage SQL

Voir le TP pour les détails.

Au niveau de la terminale retenez les structures des requêtes :

- d'exploration des données, version programme de Terminale NSI

  SELECT *liste d'attributs* (et *fonctions d'agrégation*)
  FROM *table*
  INNER JOIN *table* ON *égalité d'attributs*
  WHERE *condition(s)*
  ORDER BY *attributs de tri* (chaque attribut suivi de DESC ou ASC)
- d'un ajout d'enregistrement

  INSERT INTO *table* (*liste d'attributs facultative*) VALUES *liste de valeurs*
- de modification d'enregistrement(s)

  UPDATE *table* SET *att1 = val1* , *att2 = val2*,… (WHERE *condition*)
- de suppression d'un enregistrement

  DELETE FROM *table* WHERE *condition*

Si vous voulez aller un peu plus loin retenez les structures des requêtes :

- d'exploration des données, version programme de Terminale NSI + compléments :

  SELECT *liste d'attributs* (et *fonctions d'agrégation*)
  FROM *table*
  INNER JOIN *table* ON *égalité d'attributs*
  WHERE *condition(s)*
  GROUP BY *attributs de regroupement*
  HAVING *condition (de groupes)*
  ORDER BY *attributs de tri* (chaque attribut suivi de DESC ou ASC)
- de création de table

  CREATE *table* (
  Attribut1 type de données, (contrainte(s))
  …
  PRIMARY KEY *attribut*
  FOREIGN KEY *attribut* REFERENCES *autre_table.attribut*
)
- de suppression de table

  DROP *table* (IF EXISTS)

4. Interaction SGBD / Python

L'utilisation de SQLite dans Python est très simple, il suffit d'importer la bibliothèque en question.

Code utilisant l'exemple du TP :

```
# Bibliothèque sqlite
import sqlite3
# Connexion à la base de données
connexion = sqlite3.connect('bdfilms.db')
# Exécution d'une requête, le nom "curseur" de la variable sera expliqué
en bac + 42
curseur = connexion.cursor()
curseur.execute ("SELECT title , release_year FROM films WHERE
original_language = 'zh'")
# Affichage des résultats de la requête
for element in curseur:
    print(element)
# Fermeture de la connexion
```


Cours de Frédéric Mandon sous licence Creative Commons BY NC SA, http://creativecommons.org/licenses/by-nc-sa/3.0/fr/.

```
connexion.close()
```

On peut ensuite travailler sur les données en Python, comme faire des graphiques, ou faire des calculs complexes, comme l'écart-type, ce que l'on ne peut pas faire en SQL.
Si l'on modifie la base de données, que ce soit en agissant sur les tables ou les enregistrement, alors il faut rajouter l'instruction commit().

```
# Insertion d'un nouvel enregistrement
connexion.execute ("INSERT INTO films_vus VALUES (679, '08-oct-1986', 4,'vo');")
connexion.commit()
```

Attaque par injection de code SQL

Attention à ne pas utiliser les propriétés de concaténation de chaîne dans un programme Python (ou autre) pour exécuter une requête SQL.
Exemple : réfléchissez à cette suite d'instructions

```
langage = " 'zh' ; DROP TABLE films"
curseur = connexion.execute ("SELECT title , release_year FROM films WHERE original_language = " + langage)
```

Cette attaque est appelée *injection de code SQL*. Pour l'éviter, on utilise une requête paramétrée, où (?) est remplacé par le paramètre précisé ensuite :

```
langage = 'zh'
curseur.execute ("SELECT title , release_year FROM films WHERE original_language = (?)",(langage,))
```

On peut mettre plusieurs paramètres dans une requête, les différents (?) seront remplacés par les paramètres précisés successivement.

Vous pouvez tester la requête paramétrée suivante, avec tentative d'injection de code, elle renverra une erreur :

```
langage = " 'zh' ; SELECT * FROM films_vus"
curseur.execute ("SELECT title , release_year FROM films WHERE original_language = " + langage)
```


Cours de Frédéric Mandon sous licence Creative Commons BY NC SA, http://creativecommons.org/licenses/by-nc-sa/3.0/fr/.