TD BASES DU LANGAGE SQL
REQUETES SUR TABLES

1. Introduction

Le langage SQL (Structured Query Language, langage de requête structurée) est un langage servant à créer, modifier et interroger des bases de données relationnelles. Nous allons en explorer quelques rudiments de l'interrogation des bases données dans ce TD. Ce TD prolonge et complète la partie du cours de 1ère sur les données en table. Il n'est pas inutile de relire le cours de l'an dernier sur ce sujet (par exemple ici http://www.maths-info-lycee.fr/donnees_1ere.html).

2. Préparation

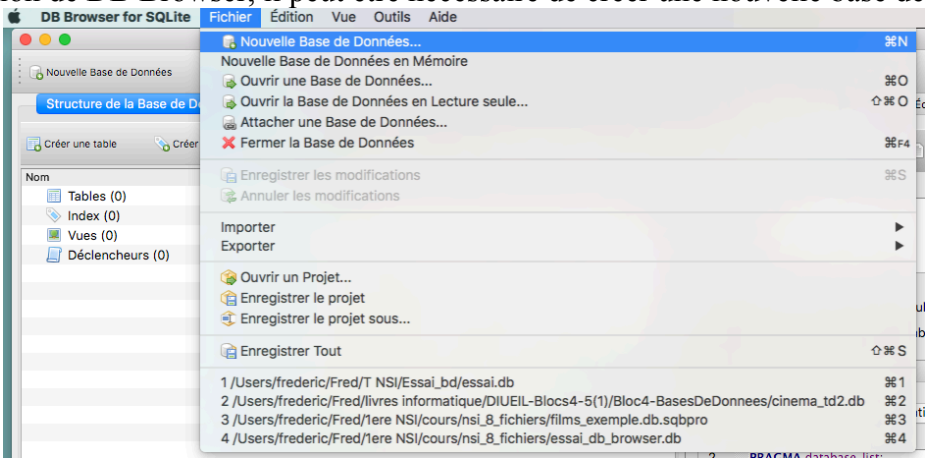
On va utiliser un gestionnaire de bases de données. Si vous ne vous en êtes jamais servi, le plus simple est d'utiliser DB Browser for SQLite, conçu pour l'apprentissage et le test. On le trouve dans EduPython (version 2.7 et suivantes, peut-être dans la 2.6). Si vous n'utilisez pas EduPython, DB Browser for SQLite se télécharge à https://sqlitebrowser.org/dl/ . *Remarque* : SQLlite est très permissif, certaines requêtes sont admises par SQLite mais pas par d'autres versions de SQL, notamment pas par celles utilisées « en vrai ».

On va importer des tables de données à partir de fichiers au format csv (rappel sous forme de question : que signifie csv ?). Télécharger les fichiers de données sur les films ici : http://www.maths-info-lycee.fr/programmes/donnees_films.zip . Les données que nous allons utiliser dans ce TP sont de taille moyenne, il y a à peu près 5000 films dans le fichier « films.csv », 5000 réalisateurs dans le fichier « realisateurs.csv », et 100 000 acteurs dans le fichier « distribution.csv ». Vous pouvez éventuellement ouvrir le fichier films.csv, dans un éditeur de textes, pour en voir le contenu.

Lancez DB Browser for SQLite. Les captures d'écrans ci-après sont faites sous Mac OsX, l'interface a un aspect différent mais est identique sous tous les systèmes.

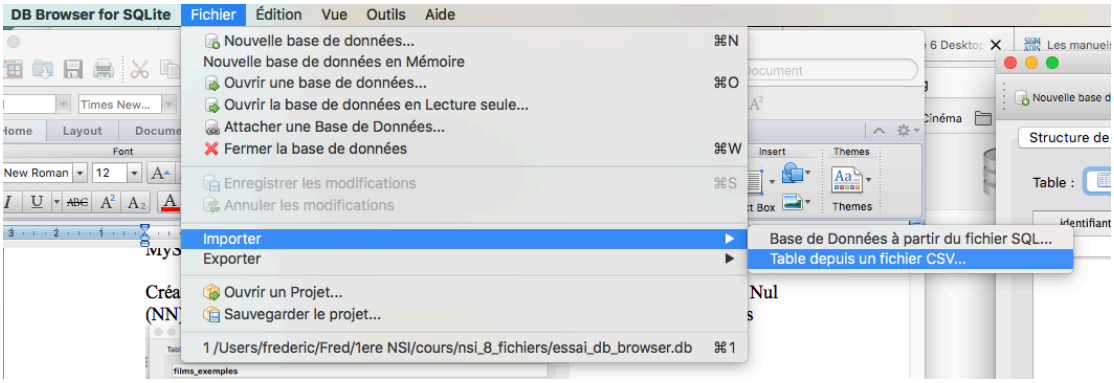
Importation d'un fichier

a. Essayez de faire directement l'import du fichier csv en passant au b. ci-dessous. Si l'import est impossible, en grisé dans le menu, faites alors l'étape préalable décrite dans ce paragraphe. En effet, suivant la version de DB Browser, il peut être nécessaire de créer une nouvelle base de données :



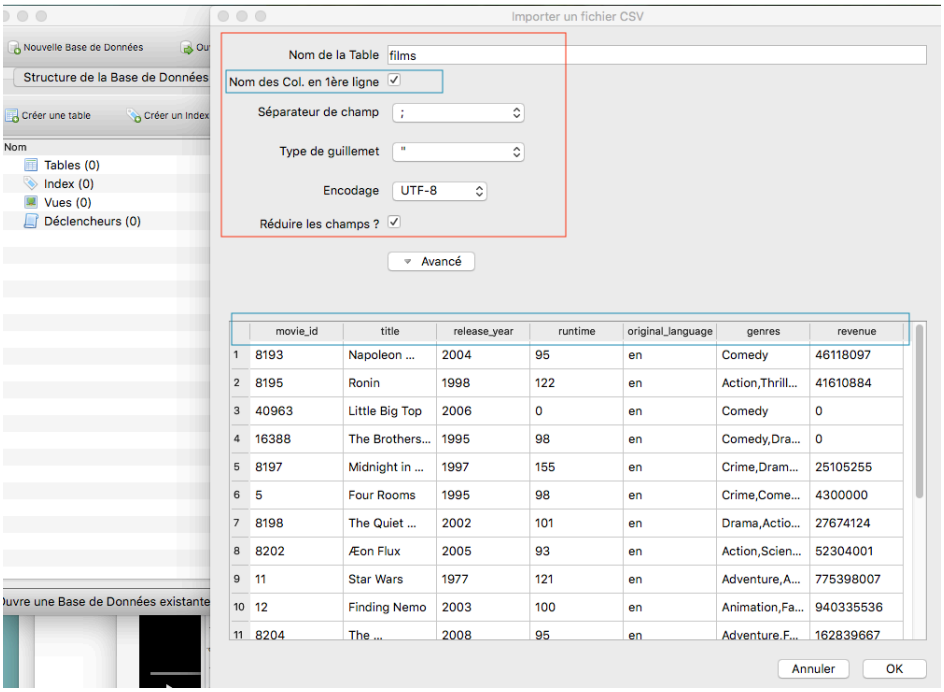
Créez votre base de données dans un répertoire bien identifié ; vous pouvez le nommer base_de_donnees par exemple. Donnez comme nom à la base de données bdfilms, ou tout autre nom clair. Annulez ensuite la fenêtre suivante (éditer la définition de la table)

b. Puis importez le fichier « films.csv »



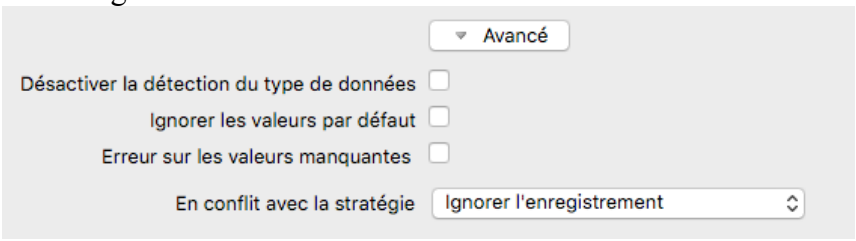
Faites attention aux éléments suivants :

- nom des colonnes en 1ère ligne. C'est le cas dans ce fichier, dans la 1ère ligne de l'exemple ci-dessous, il y a « movie_id », « title », etc.
- Séparateur de champ : souvent c'est « ; », mais parfois c'est « , » ou tabulation. Dans le fichier fourni c'est bien le « ; »
- Type de guillemets : laissez celui qui est proposé
- Encodage : ici utf-8 (rappel sous forme de question : qu'est-ce que l'utf-8 ?).



	movie_id	title	release_year	runtime	original_language	genres	revenue
1	8193	Napoleon ...	2004	95	en	Comedy	46118097
2	8195	Ronin	1998	122	en	Action,Thrill...	41610884
3	40963	Little Big Top	2006	0	en	Comedy	0
4	16388	The Brothers...	1995	98	en	Comedy,Dra...	0
5	8197	Midnight in ...	1997	155	en	Crime,Dram...	25105255
6	5	Four Rooms	1995	98	en	Crime,Come...	4300000
7	8198	The Quiet ...	2002	101	en	Drama,Actio...	27674124
8	8202	Æon Flux	2005	93	en	Action,Scien...	52304001
9	11	Star Wars	1977	121	en	Adventure,A...	775398007
10	12	Finding Nemo	2003	100	en	Animation,Fa...	940335536
11	8204	The ...	2008	95	en	Adventure,F...	162839667

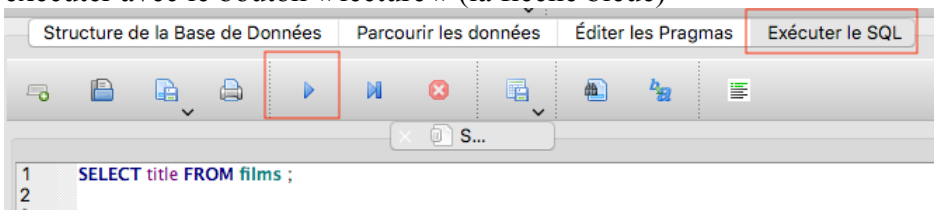
- En cas d'erreur entraînant un abandon de l'import, aller dans les réglages avancés et choisir l'option « ignorer l'enregistrement » :



c. L'onglet « parcourir les données » vous permet de vérifier que les données sont bien importées dans la table.
Nous verrons ultérieurement qu'il y a des contraintes à définir et respecter sur les données. Dans les tables fournies, ces contraintes sont déjà vérifiées.

d. Vocabulaire.
Une table de données est une **relation**, les colonnes sont les **attributs** de la relation.
Exemple : dans la relation *film* précédente, l'attribut *runtime* donne la durée des films

3. Exploitation des données avec une seule table.
   a. Projection
   La projection est la sélection de certaines colonnes d'une table
   SELECT <liste d'attribut(s)> **FROM** <liste de table(s)>
   Les mots-clés peuvent être en majuscules ou en minuscules, l'usage fait qu'on les tape en majuscules en général. On finit les lignes par un point-virgule.
   Exemple :
   - Aller dans l'onglet « exécuter le SQL » ;
   - taper SELECT title FROM films ;
   - exécuter avec le bouton « lecture » (la flèche bleue)



S'il y a plusieurs tables avec les mêmes noms d'attributs :
SELECT films.title FROM films
Sélectionner plusieurs attributs :
SELECT title , release_year FROM films
Supprimer les doublons :
SELECT **DISTINC**T release_year FROM films ;
Sélectionner toutes les colonnes :
SELECT * FROM films ;

   b. Ordonner les résultats
   Exemples :
   - SELECT title, release_year FROM films **ORDER BY** release_year ;
   - SELECT title, release_year FROM films ORDER BY release_year **DESC** ;
   - SELECT title, release_year FROM films ORDER BY release_year DESC , title ASC ;

   Par défaut, le tri est ascendant (dans l'ordre numérique ou lexicographique croissant)

   c. Restriction et sélection
   L'objectif est d'afficher certaines lignes, qui vérifient un critère donné. On utilise le mot-clé **WHERE**.
   Exemples :
   - SELECT title FROM films WHERE release_year = 1953 ;
   - SELECT * FROM films WHERE title = "Batman" ;

   Opérateurs :
   On dispose des opérateurs de comparaison usuels, qui fonctionnent aussi bien avec les données numériques, les chaînes, ou les dates (que nous verrons ultérieurement).
   - Égalité = ; différence <> ; supérieur >, inférieur <, supérieur ou égal >=, inférieur ou égal <=
   - **AND** ; **OR** ; **NOT**
   - **IN** permet de tester l'appartenance à une liste
     SELECT title FROM films WHERE release_year IN (1953, 1963,1973) ;

Complément : sélectionner des lignes qui commencent (ou contiennent ou finissent) par un caractère donné :

- SELECT * FROM films WHERE title **LIKE** "ab%" ;
- Donner la requête qui permet de trouver les films finissant par « ab », ainsi que ceux contenant « ab ».

d. Exercices 1.

Pour chacun de ces exercices, vous copierez la solution dans le document « exercices ».
Il peut être utile d'aller dans l'onglet « explorer les données » pour retrouver les noms des attributs.

- Donner la liste des titres et recettes des films ayant réalisé plus de 1000000000 $ de recettes.
- Donner la liste de toutes les données sur les films réalisés entre 1918 et 1939. On proposera deux requêtes, la deuxième utilisera **BETWEEN** ; vous en devinerez la syntaxe.
- Donner la liste des films de langue anglaise, française, espagnole ou allemande.
- Donner la liste de toutes les langues différentes de réalisation.
- Donner la liste des titres et des langues des films qui n'ont pas été tournés en anglais (« en » pour english), en les classant par langue.
- Donner la liste des titres, année de sortie et durée des films de plus de 3 heures, classée par durée décroissante
- Donner la liste des titres et années de sortie des films d'action.
- Donner la liste des titres et années de sortie des films d'action dont le titre comporte « Man ».

4. Attributs calculés

On peut créer de nouvelles colonnes, en calculant un résultat à partir d'une colonne existante.
Remarquez la différence entre les deux requêtes suivantes

- SELECT title, revenue/1000000 FROM films WHERE revenue > 1000000000;
- SELECT title, revenue/1000000.0 FROM films WHERE revenue > 1000000000;

Pour plus de clarté, on peut renommer une colonne avec **AS** :

- SELECT title, revenue/1000000 AS recette_en_m$ FROM films WHERE revenue > 1000000000;

On peut créer de nouvelles colonnes à partir de colonnes existantes :

- SELECT title, revenue/runtime AS recette_par_mn FROM films WHERE revenue > 1000000000;

5. Fonctions d'agrégation

Ces fonctions servent à calculer une valeur à partir de plusieurs lignes :

- Somme **SUM**, moyenne **AVG**, minimum **MIN**, Maximum **MAX**
- Compter un nombre de lignes **COUNT**

Exemple :

- SELECT AVG(revenue) AS recette_moyenne FROM films;
- SELECT COUNT(*) AS nb_films, AVG(revenue) AS recette_moyenne, MIN(revenue) AS recette_min , MAX(revenue) AS recette_max FROM films;

Remarque :
Le dernier exemple est une requête assez longue, qui est peu lisible sous cette forme. On peut l'écrire sur plusieurs lignes :

```
SELECT COUNT(*) AS nb_films,
AVG(revenue) AS recette_moyenne,
MIN(revenue) AS recette_min ,
MAX(revenue) AS recette_max
FROM films;
```

Testez et concluez (et rédigez votre conclusion !) : SELECT title , COUNT(*) as nb_films FROM films

On peut combiner ces opérations de calcul avec une sélection :

- SELECT COUNT(*) as nb_films_fr FROM films WHERE original_language = "fr" ;
- SELECT COUNT(DISTINCT original_language) as nb_langues_vieux
  FROM films
  WHERE release_year < 1960;

6. Exercices 2

Pour chacun de ces exercices, vous copierez la solution dans le document « exercices ».

- Donner la durée moyenne des films.
- Trouver les titres des films de durée anormalement courte.
- Reprendre la première question en excluant les films précédents. *Remarque* : dans une base de données, il est possible qu'il y ait des erreurs. Le travail de l'informaticien n'est pas de corriger ces erreurs, mais de les signaler, et éventuellement de faire le travail demandé en ne tenant pas compte des valeurs fausses.
- On suppose que le prix moyen d'un billet est de 8 $. Calculer le nombre d'entrées des films les plus rentables (recette supérieure ou égal à 500000000 $).
- Donner la recette totale des films de 2015.
- Donner la durée moyenne, maximale et minimale des films les plus rentables (recette supérieure ou égal à 500000000 $).
- Donner les titres et recette par minute des films les plus rentables (recette supérieure ou égal à 500000000 $), classée par recette par minute décroissante.

7. Jointures (exploitation des données avec plusieurs tables).

Importez les tables realisateurs.csv et distribution.csv dans la base de donnée actuelle (n'en créez pas une nouvelle).

Les jointures sont utilisées pour combiner les informations de plusieurs tables, à l'aide des clés étrangères. Plusieurs rédactions sont possibles, en exemple ici pour donner le réalisateur de chaque film :

- SELECT title , director FROM films , realisateurs
  WHERE films.movie_id = realisateurs.movie_id;

- SELECT title , director FROM films **INNER JOIN** realisateurs
  ON films.movie_id = realisateurs.movie_id;

- SELECT films.title , realisateurs.director FROM films INNER JOIN realisateurs
  ON films.movie_id = realisateurs.movie_id;

Remarques :

- La troisième rédaction a la préférence de votre professeur.
  - Par rapport à la première, elle a l'avantage de bien distinguer les tables des attributs et de différencier le mot-clé WHERE du mot-clé ON. Cette première pratique est obsolète. Par ailleurs, l'usage du mot-clé JOIN dans les deux autres rédactions permet de préciser le type de jointure. En terminale, on ne fait que des jointures internes, mais dans le TD de 1ère vous avez fait d'autres types de jointures.
  - Par rapport à la deuxième, la troisième rédaction respecte l'usage qui est de préciser la table des attributs à utiliser, ce qui se fait lorsqu'il y a plusieurs tables en jeu.
- Dans les deux rédactions, on précise bien sur quel attribut on fait la jointure, avec en plus l'identifiant de la relation : « films.movie_id » est l'attribut movie_id de la relation films. Il est possible que les attributs aient des noms différents dans les tables, ce n'est pas un problème.
- On peut utiliser les diverses opérations vues avant (et aussi en complément ci-après) lors d'une jointure :

- SELECT title , director
  FROM films INNER JOIN realisateurs
  ON films.movie_id = realisateurs.movie_id
  WHERE release_year = 1970;

8. Exercices 3

Pour chacun de ces exercices, vous copierez la solution dans le document « exercices ».
Je vous conseille de vérifier le nom des attributs avant de vous lancer dans les requêtes (onglet « explorer les données »). Respectez également scrupuleusement les orthographes des noms donnés ci-dessous.

- Donner la liste des acteurs ayant tourné dans des films français. La requête renverra le nom de l'acteur et le titre du film.
- Donner la liste des films dans lesquels a tourné Sigourney Weaver, classée par année de sortie.
- Donner la liste des acteurs du film "Amélie" (titre anglais de « Amélie Poulain »).
- Donner la liste des acteurs ayant tourné sous la direction de Ridley Scott.
- Donner la liste par ordre alphabétique des acteurs dont les films comiques ont fait plus de 2000000000 $ de recettes.
- Donner la liste des acteurs ayant tourné sous la direction de Ridley Scott, ainsi que les titres des films dans lesquels ils ont tourné.
- Donner la liste des acteurs ayant tourné sous la direction de Wong Kar-wai dans des films en Cantonais (l'abréviation pour Cantonais est… zh…)

9. Complément (hors programme) : regroupement de lignes

Cette partie est certes hors programme, mais elle n'est pas bien difficile et permet de regrouper en une seule ligne des informations sur des données ayant un caractère commun.
Exemple : on souhaite trouver le nombre de films par langage original ; on utilise **GROUP BY**.

- SELECT original_language , count(*) as nombre_films
  FROM films
  GROUP BY original_language;

Exemple 2 : on souhaite trouver le nombre de films par langage original, ainsi que les recettes totales générées, en excluant les films de langue anglaise ; on utilise GROUP BY et **HAVING**. *Remarque* : cet exemple doit aussi fonctionner avec WHERE. WHERE est exécuter avant GROUP BY, HAVING après.

- SELECT original_language ,
  count(*) as nombre_films ,
  sum(revenue)
  FROM films
  GROUP BY original_language HAVING original_language <> "en";

Testez et concluez (et rédigez votre conclusion !) :

- SELECT title, original_language , count(*) as nombre_films
  FROM films
  GROUP BY original_language;

Exercices complémentaires (4) :

a. Donner la durée moyenne des films par année.
b. Donner la liste des films les plus rentables par nationalité (on affichera la langue d'origine et les recettes).
c. Donner le nombre de films sortis après 1965, chaque année et par nationalité. Dans un deuxième temps, on triera en plus par année de sortie (ou par ordre alphabétique si c'est déjà par année de sortie)
d. Donner la liste des films les plus rentables par nationalité (on affichera la langue d'origine et les recettes). On n'affichera pas les nationalités dont les recettes des films sont égales à 0.

e. Donner la liste des films les plus rentables par nationalité (on affichera le titre, la langue d'origine et les recettes).
Remarque : avec une requête « un peu pourrie » en SQLite, on obtient facilement la réponse. Cette requête n'est pas acceptée dans des versions plus rigoureuses de SQL. Réfléchir au pourquoi de « un peu pourrie ». Avec les outils du paragraphe suivant, on peut répondre rigoureusement à cette question.
f. Trouver les titres de films qui sont présents plusieurs fois dans la relation « films ».
Puis donner les réalisateurs de ces films
g. Donner la liste des réalisateurs ayant tourné dans plusieurs langues.

10. Complément (probablement hors programme) : requêtes imbriquées.

La requête suivante permet de trouver tous les films sortis la même année que Star Wars :

- SELECT title from films
  WHERE films.release_year =
  (SELECT release_year from films where title = "Star Wars");

Exercice :

- Répondre en une seule requête à l'exercice 9f.
- Reprendre la question : « donner la liste des films les plus rentables par nationalité (on affichera le titre, la langue d'origine et les recettes) ».
  On utilisera à la fois requêtes imbriquées et auto-jointure. Une auto-jointure s'écrit par exemple :

```
SELECT f1.movie_id , f2.title
FROM films as f1 INNER JOIN films as f2
ON f1.movie_id = f2.movie_id
```

11. Créer une relation

L'objectif de cette partie est de créer une nouvelle relation (table), afin de pouvoir enregistrer les films vus. *Les connaissances de ce paragraphe ne peuvent pas être demandées au bac.*

a. Échauffement.
Aller dans l'onglet « structure de la base de données », clic droit sur la table « films », puis « modifier une table ».
Le code SQL de la création de la table (**CREATE**) s'affiche dans la fenêtre qui s'ouvre. Commentez ce code en vous inspirant de la partie du cours portant sur les contraintes d'intégrité et de construction des bases de données. Notamment, pensez-vous que cette relation est bien construite ? Qu'y manque-t-il ? Dans la partie interface graphique de cette fenêtre, cochez les cases permettant de mieux construire la table, et observez les modifications conséquentes du code. Vous pouvez également copier et coller dans un document annexe les requêtes SQL

On va en conséquence recréer notre base de données, mais proprement cette fois-ci ! Vous pouvez détruire la base courante, après avoir quitté DB Browser for SQLite (c'est mieux que de simplement effacer la table, il y a des erreurs bizarres avec ce logiciel…pour changer…). Relancez le logiciel.
La requête SQL propre pour la relation « films » est :

```
CREATE TABLE `films` (
`movie_id`      INTEGER NOT NULL UNIQUE,
`title`   TEXT,
`release_year` INTEGER,
`runtime`       INTEGER,
`original_language`    TEXT,
`genres`        TEXT,
`revenue`       INTEGER,
PRIMARY KEY(`movie_id`)
);
```

Recréer aussi les relations « realisateurs » et « distribution ». Quelques remarques :

- Ne pas importer les fichiers csv avant d'avoir crée la base en entier.
- La clé primaire pour la table « réalisateurs » est le couple (« movie_id » , « actor »). Pourquoi ?
- Si vous essayez de faire de même pour la table « distribution », il y aura un souci lors de l'import des données. Ne pas modifier, on en aura besoin ultérieurement, mais explorer les options de l'interface afin que l'import fonctionne (il y a plusieurs possibilités)

- SQL n'est pas sensible à la casse. Si cela vous amuse, vous pouvez écrire iDeNtIfIaNt, SQL reconnaîtra « identifiant ».
- Pour détruire une table, la requête est **DROP TABLE** *nom_de_la_table*.

b. Création de la table « films_vus »

Créer avec une requête SQL, et non avec l'interface graphique, la table « films_vu » dont les attributs sont :

- Movie_id, non nul . C'est une clé étrangère : on rajoute en fin de la requête
  FOREIGN KEY(`movie_id`) **REFERENCES** `films`(`movie_id`)
- Date. Le type est **DATE** et non INTEGER ou NUMERIC. Non nulle
- La clé primaire est le couple « movie_id » , « date ».
- Note. La note est un entier compris 1 et 5 (ou autres valeurs si vous souhaitez). Syntaxe
  Note INTEGER **CHECK** ((note >= 1) AND (note <=5))
- Version. Pour préciser la version du film, on peut mettre un champ de type VARCHAR(2) (2 caractères maximum, TEXT étant de longueur quelconque)
  Version VARCHAR(2) **CHECK** (version **IN** ('vo' , 'vf'))

Vérifiez que la table est bien créée, et que le code de création en est correct. Ceci peut se voir avec un clic droit sur « modifier la table » dans l'onglet « structure de la base de données ». Vous pouvez ensuite importer les fichiers csv. Si vous avez bien fait attention à donner le même nom à la relation qu'au fichier, avec les mêmes noms d'attributs, DB Browser vous proposera l'import dans les tables que vous venez de créer.

12. Insertion/modification/suppression d'enregistrements

Pour rajouter des enregistrements, on utilise

INSERT INTO table (liste d'attributs facultatives) **VALUES** liste de valeurs

Exemples :

- Si les données sont dans l'ordre des champs :
  INSERT INTO films_vus VALUES (348, '12-sept-1979', 5,'vo');
- Si les données sont dans le désordre ou manquantes :
  INSERT INTO films_vus ("movie_id", "note", "date")
  VALUES (78, 5,"15-sept-1982");
- *Tester, corriger et commenter* :
  - INSERT INTO films_vus VALUES (348, '12-sept-1979', 5,'vf');

Exercice :

- Insérer quelques lignes de votre choix, dont au moins deux avec des notes inférieures ou égales à 2. Mettre un ou plusieurs films sans « note ».
- Tester la requête SELECT movie_id FROM films_vus WHERE note = NULL
- Tester la requête SELECT movie_id FROM films_vus WHERE note IS NULL
- Conclure, et compter le nombre de films vus où la note n'est pas renseignée.

Pour supprimer des enregistrements, on utilise

DELETE FROM table **WHERE** condition

Exercice : supprimer les films vus dont la note est inférieure ou égale à 2.

Pour mettre à jour des enregistrements, on utilise

UPDATE table **SET** att1 = val1 , att2 = val2,… (facultatif **WHERE** condition)

Exercice : mettre à jour les enregistrements où la note n'est pas renseignée en mettant la valeur de votre choix par défaut.

13. Vérification de la cohérence des données ; améliorations à apporter.

Utiliser les méthodes du paragraphe "groupement de lignes" pour répondre aux questions suivantes, du moins pour ceux qui en ont le temps et/ou l'envie, ces notions n'étant pas au programme. Vous pouvez aussi récupérer les requêtes auprès du professeur pour les tester, ou encore utiliser les réponses qui sont données ci-après.

Y-a-t-il :

- Des identifiants de films identiques présents plusieurs fois dans la table des films ?
- Des couples acteurs/identifiants de films présents plusieurs fois dans la table des acteurs ?
- Idem avec la relation des réalisateurs.

Utiliser la méthode du paragraphe "requêtes imbriquées" pour répondre aux questions suivantes, avec la même remarque que pour les questions précédentes.

Y-a-t-il :

- des films sans réalisateurs ?
- Des films sans acteurs ?
- Des acteurs dans des films non référencés ?
- Idem avec la relation des réalisateurs.

Les réponses respectives sont oui, oui, non, oui, oui, oui, oui. Utiliser ces résultats, ainsi qu'une critique des requêtes CREATE, pour répondre à la question : quels sont les qualités et défauts des relations films, distribution et réalisateurs, d'un point de vue purement "conception de la base de données" ? Qu'auriez-vous proposé pour éviter ces défauts ? Vous pouvez aussi répondre à cette question en vous plaçant du côté utilisateur, qui n'a pas forcément les mêmes idées et besoins.