Bases de données – TP en SQLite

Bibliographie

[1] Tutoriel d'installation de sqlite sous windows

https://www.sqlitetutorial.net/download-install-sqlite

[2] Documentation sqlite : sqlite :

www.sqlitetutorial.net

[3] Tutoriel de prise en main de sqlite :

doc.ubuntu-fr.org/sqlite

[4] Tutoriel de SQL, en ligne autocorrigé mais en anglais :

w3schools.com

[5] Tutoriel d'auto-apprentissage de l'AEIF, en ligne autocorrigé:

https://nreveret.forge.aeif.fr/exercices_bdd

https://e-nsi.forge.aeif.fr/exercices_bdd/

[6] Cours en ligne : Introduction aux BD, SQL

https://www.pierre-giraud.com/php-mysql-apprendre-coder-cours/introduction-sql-mysql

Le langage SQL

Le modèle relationnel de Codd a été rapidement admis comme modèle définitif pour les bases de données. La première version de **langage d'interrogation structuré (SQL - *Structured Query Language)*** a été développé chez IBM en 1970 par Donald Chamberlain et Raymond Boyce. En 1979, Relational Software, Inc. (actuellement Oracle Corporation) présenta la première version commercialement disponible de **SQL**, rapidement imité par d'autres fournisseurs.

SQL se décompose en 5 parties, à savoir :

- ✓ **Ordres LDD** (langage de **définition des données**, ou **DDL**, *Data Definition Language*) : permet de modifier la structure de la base de données.
- ✓ **Ordres LMD** (langage de **manipulation des données**, ou **DML**, *Data Manipulation Language*) : permet de consulter / modifier le contenu de la base de données.
- ✓ Ordres LCD (langage de **contrôle des données**, ou **DCL**, *Data Control Language*) : permet de gérer les privilèges, c'est-à-dire les utilisateurs et les actions qu'ils peuvent entreprendre.
- ✓ **Ordres LCT** (langage de **contrôle des transactions**, ou **TCL**, *Transaction Control Language*) : permet de gérer les transactions, c'est-à-dire rendre atomique divers ordres enchaînés en séquence.
- ✓ ***SQL procedural*** : PSM (Persistent Stored Module), CLI (Call Level Interface), Embedded SQL, … qui est un ensemble d'outils pour que SQL s'interface avec des langages hôtes.

Nous ne verrons ici que des exemples simples pour les deux premières parties

SQL et MYSQL

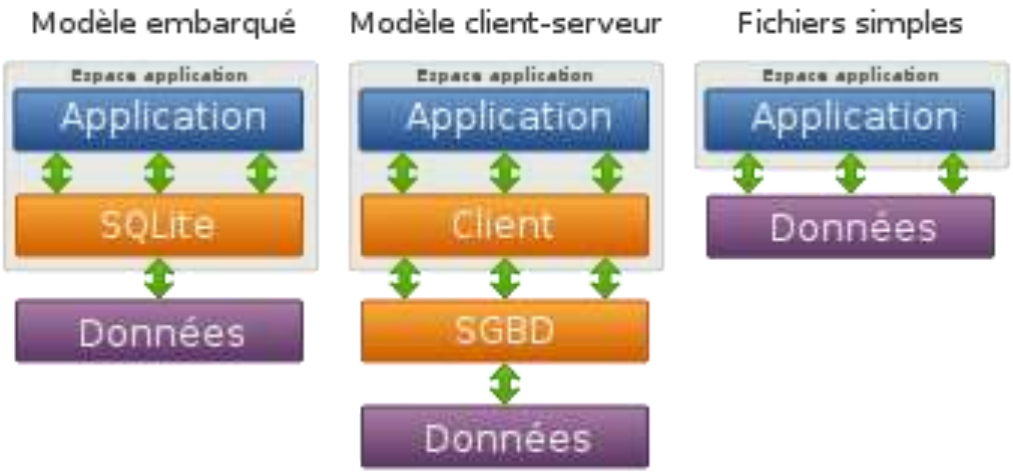
MySQL est un système de gestion de bases de données relationnelles (SGBDR). Il est distribué sous une double licence libre (ou GPL pour General Public Licence) et propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés

au monde, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle, Informix et **Microsoft SQL Server**. MySQL est un serveur de bases de données relationnelles SQL. MySQL fonctionne sur de nombreux systèmes d'exploitation différents, incluant Linux, Mac OS X et Windows. Les bases de données sont accessibles en utilisant bons nombres de langages de programmation (C, PHP, Python …). MySQL fait partie du quatuor **LAMP : Linux, Apache, MySQL, PHP**. Il appartient également à ses variantes **WAMP** (Windows).

La plupart des SGBD comme MySQL sont basés sur un modèle client-serveur. C'est-à-dire que la base de données se trouve sur un serveur qui ne sert qu'à ça, et pour interagir avec cette base de données, il faut utiliser un logiciel "client" qui va interroger le serveur et transmettre la réponse que le serveur lui aura donnée. Le serveur peut être installé sur une machine différente du client ; c'est souvent le cas lorsque les bases de données sont importantes.

Par conséquent, lorsqu'on installe un SGBD basé sur ce modèle, on installe en réalité deux choses (au moins) : le serveur et le client. Chaque requête (insertion/modification/lecture de données) est faite par l'intermédiaire du client. Jamais, on ne discute directement avec le serveur mais par l'intermédiaire de requête SQL.

Par contre, **SQLite** est une bibliothèque C qui implémente un moteur de base de données sans serveur. Une base de données SQLite est implémenté par un simple fichier. A l'heure actuelle, SQLite est l'un des moteurs de base de données les plus simples à installer, à manipuler et à administrer, ce qui en fait un très bon candidat pour l'utilisation dans un site web ou dans un smartphone.



1 TP1 Prise en main de SQLite dans un terminal

- ✓ Sous public_html/xxéInfoOSLogi5TT, crée un dossier **xx**_BD, un sous-dossier tp1 et ouvre un terminal linux.

1.1 Commandes SQLite

- ✓ **Lance le terminal SQLite**

Dans un **terminal linux**, lance la commande suivante avant de taper les commandes propres à SQLite :

```
sqlite3
```

« **sqlite>** » indique que tu es maintenant dans le "terminal" SQLite comme ici :

```
SQLite version 3.6.22
Enter ".help" for instructions
Enter SQL statements terminated with a " ;"
sqlite>
```

Toute commande SQLite commence par un point « . » et sera précédé de son curseur afin de bien distinguer les commandes SQLite des autres commandes.
Tu dois taper les commandes surlignées, les autres sont des explications complémentaires

- ✓ **Liste les commandes SQLite**

```
sqlite> .help
```

- ✓ **Quitte le terminal SQLite**

```
sqlite> .quit
```

1.2 Les commandes simples de manipulation de BD

1. Créer une base - ouvrir une base
Dans ton **terminal linux**, tape la commande suivante ;

```
sqlite3 livres.db
```

Si la base livres.db n'existe pas déjà, elle sera créée (tu peux vérifier qu'un fichier livres.db a bien est créé). Cette commande, non seulement crée ou ouvre la base, mais place le curseur immédiatement dans l'environnement terminal de SQLite. Toute commande tapée par la suite concernera cette base..

2. Détruire une base

Il suffit de détruire le fichier correspondant dans le terminal linux

```
rm livres.db
```

3. Créer une table : CREATE TABLE

Il faut avoir ouvert ou créé une base avant de taper la commande SQL suivante.

Exemple de création d'une table

```
sqlite> CREATE TABLE bandeDessinee (id int PRIMARY KEY, titre text, auteur text, resume text, num real, datecreation int);
```

Si …> apparait après avoir tapé la commande, c'est qu'il manque tout simplement le ; à la fin de la requête. Ajoute-le juste après le …> ; et valide.
Les requêtes SQL sont tapées en MAJUSCULES.
Les variables sont tapées en minuscules ; si elles sont composées de plusieurs mots, on peut les séparer par des « _ » ou mettre la 1ère lettre en majuscule (exemple : bande_dessinee ou bandeDessinee)

Syntaxe de création d'une table

```
CREATE TABLE tableName (
    colonne1 datatype,
    colonne2 datatype,
    colonne3 datatype,
    …
);
```

Création d'une table avec des contraintes

Principales contraintes SQL :

- **NOT NULL** : assure qu'une colonne ne peut pas avoir une valeur nulle c'est-à-dire qu'un champ ne peut pas être vide
- **PRIMARYKEY** : assure que tous les champs ne sont pas vides et ont des valeurs différentes. La clé primaire identifie chaque enregistrement de manière unique.
- **FOREIGNKEY** : identifie un enregistrement d'une autre table.

Exemple de commande

```
CREATE TABLE orders (
    orderID int NOT NULL,
    orderNumber int NOT NULL,
    personID int,
    PRIMARY KEY (orderID),
    FOREIGN KEY (personID) REFERENCES persons (personID)
);
```

4. Lister les tables

```
sqlite> .tables
```

5. Insérer des enregistrements dans la table : INSERT

Insère les enregistrements suivants :

```
sqlite> INSERT INTO bandeDessinee VALUES (1, 'Tintin au Congo', 'Hergé', 'Tintin est au Congo.', 5.0, NULL);
sqlite> INSERT INTO bandeDessinee VALUES (2, 'le nid des marsupilamis', 'Franquin', 'Un reportage incroyable', 6.0, date('now'));
sqlite> INSERT INTO bandeDessinee VALUES (3, 'la déesse', 'Moebius', 'une aventure géniale', 7.0, strftime("%Y-%m-%d %H:%M:%S",'now','localtime'));
```

Syntaxe de l'insertion d'un nouvel enregistrement

```
INSERT INTO nomTable (colonne1, colonne2, colonne3, …)
VALUES (valeur1, valeur2, valeur3, …) ;
```

ou pour une mise à jour de tous les enregistrements :

```
INSERT INTO nomTable VALUES (valeur1, valeur2, valeur3, …) ;
```

6. Visualiser le contenu de la table : SELECT

```
sqlite> SELECT * FROM bandeDessinee;
```

Résultat

```
1 | tintin au Congo | Hergé | Tintin est au Congo. | 5.0|
2 | le nid des marsupilamis | Franquin | Un reportage incroyable | 6.0 | 2011-02-03
3 | la déesse | Moebius | une aventure extraordinaire | 7.0 | 2011-02-03 18:36:25
```

- ✓ **Visualiser les titres des bandes dessinées créées en février**

```
SELECT titre FROM bandeDessinee WHERE strftime ('%m', date_creation) = '02';
```

Résultat

```
le nid des marsupilamis
la déesse
```

Syntaxe d'une requête SELECT

```
SELECT colonne1, colonen2, …
FROM nomTable
WHERE condition
```

7. Effacer un enregistrement dans la table: DELETE

```
sqlite> DELETE FROM bandeDessinee WHERE id = 3;
```

Pense à visualiser ta table après chaque manipulation.

Syntaxe pour effacer un enregistrement

```
DELETE FROM nomTable WHERE condition
```

8. Ajouter une colonne à la table : ALTER TABLE

```
sqlite> ALTER TABLE bandeDessinee ADD "éditeur" TEXT;
```

Syntaxe pour ajouter une colonne

```
ALTER TABLE nomTable
ADD nomColonne type ;
```

9. Modifier / mettre à jour un enregistrement à la table : UPDATE

```
sqlite> UPDATE bandeDessinee SET éditeur = 'Casterman' WHERE id = 1 ;
```

Syntaxe pour modifier un enregistrement

```
UPDATE nomTable
SET colone1=valeur1, colone2=valeur2, …
WHERE condition ;
```

10. Modifier le nom d'une table

```
sqlite> ALTER TABLE bandeDessinee RENAME TO 'bd';
```

11. Supprimer une table : DROP

```
sqlite> DROP TABLE nomTable
```

1.3 Sauvegarder / Ouvrir une table au format SQL

- ✓ Copie la table au format SQL

Quitte sqlite3 puis sauvegarde la BD en format SQL avec la commande suivante dans le terminal linux :

```
sqlite3 livres.db .dump > livres.sql
```

- ✓ Renomme la base originale avant de procéder à la récupération de la base :

```
mv livres.db livres_save.sb
```

- ✓ Re-crée la base de données

```
sqlite3 livres.db
```

- ✓ Assure-toi qu'elle est vide

```
sqlite> .tables
```

- ✓ Lis la base de données à partir du fichier sql

```
sqlite> .read livres.sql
```

- ✓ Fais une requête pour vérification

```
sqlite> .tables
sqlite> SELECT * FROM bd ;
```

1.4 Modifier le format de sortie

Pour rendre l'affichage plus lisible les résultats de ta requête, tu peux :

- Afficher le nom des colonnes / Changer l'aspect des colonnes
- Modifier le séparateur
- Modifier la largeur des colonnes
- Modifier la sortie en code html

Pour plus d'informations, consulte le site internet [2]. Ici nous nous contenterons de :

- Présenter les résultats d'une requête sous forme de tableau
- Définir une largeur de colonne.

Commandes

```
.header on
.mode column
.width 20 20 20 20 20
```

Exemple de résultat (obtenu après avoir tapé la commande 1.2.6) :

```
id    titre               auteur    resume                 num  date_creation  editeur
----  ------------------  --------  ---------------------  ---  -------------  ----------
1     tintin au congo     hergé     Tintin est au congo.   5.0                 casterman
2     le nid des mars     franquin  Un reportage incroya   6.0  2021-02-12
```

Tu pourrais écrire ces commandes dans le terminal linux mais pour que les modifications soient prises en compte à chaque démarrage, tu peux créer un fichier **.sqliterc** sous /home/user puis les copier dedans.
Attention, les fichiers qui commencent par un point sont des fichiers cachés ; pour les voir, rajoute l'option **–la** à la commande **ls** dans le terminal linux :

Commandes linux

```
cd
```

```
ls -la
gedit .sqliterc &
```

- ✓ **Historique des commandes sqlite**

A la fin de l'exercice, tu pourras lire les commandes SQLite que tu as tapées en éditant le fichier **.sqlite_history** sous /home/user.

2 TP2 - Création et manipulations simples de BD : filmographie

Dans ce TP, tu vas voir les requêtes basiques pour créer et manipuler une base de données sous SQLite.
A la fin de la séance, tu dois rendre un script SQL (un fichier **xx_tp2.sql**) avec les requêtes SQL qu'il t'est demandé d'écrire ainsi que l'historique des requêtes du TP **xx_tp2_readme.sql**.
Dans cet exercice, tu vas créer une base de données suivante :

films (f_id entier, titre texte)

acteurs (a_id entier, nom texte, prenom texte)

filmographie (fg_fk_acteur_id entier, fg_fk_film_id entier)

Règles d'écriture et de travail

Pour rappel, films, acteurs, filmographies sont des tables
Pour chaque table, on liste en () leurs attributs (id, titre etc …) en précisant leur type (int, text …)
Les attributs soulignés sont des clés primaires (Primary Key) ou secondaires/étrangère (Foreign Key).
Préfixe les attributs avec la 1ère lettre de l'entité quand les noms peuvent prêter à confusion (ex : f_id, a_id).
Indique dans le nom de l'attribut s'il s'agit d'une clé étangère (ex : fg_fk_acteur_id qui va faire le lien avec l'identifiant de la table acteurs)
Utilise des noms d'attributs clairs, et si 2 entités ont le même attributs, rajoute en préfixe l'initiale de l'entité.
Evite les majuscules, privilégie l'underscore « _ »
Ecris chaque requête dans un fichier sql que tu liras dans le terminal sqlite avec la commande .read

Requêtes sur la table Films

1. Crée et ouvre une BD xx_tp2.db.

   Crée et ouvre un fichier films.sql dans lequel tu écriras les requêtes des questions suivantes.

   Tu peux tester tes requêtes en lisant le fichier sql :

```
sqlite> .read films.sql
```

2. Ecris une requête pour créer la table Films avec la colonne f_id de type entier (en tant que **clé primaire**) et la colonne titre de type texte (non null).

Si tu lis plusieurs fois le fichier sql, une erreur risque d'être générée car on ne peut pas créer une table déjà existante. Dans ce cas, avant de la créer, il faut l'effacer en écrivant la commande suivante dans films.sql :

```
sqlite> DROP TABLE IF EXISTS film
```

3. Ajoute à la table films les titres de films suivants, avec leurs clés respectives :

   {(1, "Les évadés"), (2, "Le parrain"), (3, "La vie de Pi")}

4. Ecris une requête pour afficher tous les éléments de la table Films.

Résultat

```
1|Les évadés
2|Le parrain
3|La vie de Pi
```

5. Ecris une requête pour ajouter les titres de films suivants. . Que se passe-t-il pour la dernière ?

   {(4, "Chocolat"), (5, "Scarface"), (6, "Rango")}, {(4, "Bienvenu chez les ch'tis),

6. Ecris une requête pour afficher tous les éléments de la table Films.

Résultat

```
1| Les évadés
2| Le parrain
3| La vie de Pi
4| Chocolat
5| Scarface
6|Rango
```

7. Ecris une requête pour afficher tous les titres de films.

Résultat

```
Les évadés
Le parrain
La vie de Pi
Chocolat
Scarface
Rango
```

Création de la table acteurs

8. Crée et ouvre un fichier acteurs.sql dans lequel tu écriras les requêtes des questions suivantes.

9. Ecris une requête pour créer la table acteurs avec la colonne a_id de type entier (en tant que clé primaire) et les colonnes nom et prenom de types texte (non null).

10. Ecris une requête pour ajouter les acteurs suivants :

    {Johnny Deep, Al Pacino, Suraj, Sharma}.

Requêtes de recherche dans la table acteurs

11. Ecris une requête qui permet de lister le nom des acteurs.

Résultat

```
Deep
Pacino
Sharma
```

Création de la table filmographie

12. Crée et ouvre un fichier filmographie.sql dans lequel tu écriras les requêtes des questions suivantes.

13. Ecris une requête pour créer la table filmographie.

14. A la suite de la définition des champs de la table, ajoute les contraintes d'intégrité : **fg_fk_acteur_id** et **fg_fk_film_id** sont des **clés étrangères** correspondant aux identifiants des acteurs (a_id) et des films (f_id).

Code pour les contraintes sur la clé étrangère acteur :

```
CONSTRAINT fk1 FOREIGN KEY (fg_fk_acteur_id) REFERENCES a_id
```

Attention, pour utiliser les contraintes de clé étrangère, à chaque connexion, il faut les activer avec la commande suivante :

```
sqlite> PRAGMA foreign_keys=ON ;
```

15. Ecris les requêtes pour remplir la table filmographie.

    Deep a joué dans Chocolat et Rango, Al Pacino dans Le parrain et Scarface et Sharma dans La vie de Pi.

Code pour « Deep a joué dans Chocolat »:

INSERT INTO filmographie VALUES (1,4)

16. Crée un script **xx_tp2.sql** décrivant ta BD.

Livre les 4 fichiers sql.

3 TP3 - Requêtes SQL avancées : filmographie

Dans ce TP, tu vas voir des requêtes SQL de manipulation de BD. A la fin de la séance, tu dois rendre tous les fichiers avec les requêtes SQL ainsi que l'historique des requêtes du TP **xx_tp3_readme.sql**.

- ✓ Crée et ouvre une BD **xx_tp3.db**
- ✓ Ecris dans le fichier de requêtes SQL **tp3_create.sql** les requêtes de création des tables films, acteurs, filmographie ci-dessous.

Tu peux vérifier ta requête dans le terminal sqlite avec la commande suivante :

```
sqlite> .read tp3_create.sql
```

Pense à vérifier que les tables ont bien été créées.

- ✓ Télécharge du drive le fichier **tp3_insert.sql** contenant les requêtes des enregistrements ci-dessous et lance-le.

Pense à vérifier que les enregistrements sont bien dans chacune des tables.

Film (f_i, titre, realisateur, annee)

Acteur (a_id, nom)

Filmographie fg_fk_acteur_id, fg_fk_film_id, role, salaire)

Les attributs soulignés sont des clés primaires ou secondaires.

Tables des films

f_id	titre	realisateur	annee
1	Les évadés	Darabont	1994
2	Le parrain	Coppola	1972
3	Le parrain 2	Coppola	1974
4	L'odyssée de Pi	Ang Lee	2013
5	Chocolat	Hallstrom	2000
6	Scarface	De Palma	1983
7	Rango	Verbinski	2011

Tables des acteurs

a_id	nom
1	Johnny Deep
2	Al Pacino
3	Suraj Sharma
4	Brad Pitt
5	Edward Norton

Tables des filmographies

fg_fk_acteur_id	fg_fk_film_id	role	salaire
1	5	Roux	5000
1	7	Rango	10000
2	2	Michael Corleone	10000
2	3	Michael Corleone	20000
2	6	Tony Montana	15000
3	4	Pi	20000

- ✓ Crée un fichier tp3_setup.sql reprenant l'ensemble des requêtes de création de la BD.

Les 3 1ères lignes améliorent l'affichage des résultats des requêtes et peuvent être insérées dans le fichier **.sqliterc** sous /home/user (voir les explications p2)

```
.header on
.mode column
.width 20 20 20 20 20
PRAGMA foreign_keys=ON ;

.read tp3_create.sql
.tables
.read tp3_insert.sql

-- Afficher le contenu des 3 tables séparément
SELECT
   *
FROM
   acteurs;

SELECT
   *
FROM
   films;

SELECT
   *
FROM
   filmographie;
```

- ✓ Tu peux aussi rajouter au fichier tp3_setup.sql une requête pour afficher une sorte de table « virtuelle » qui reprend les enregistrements des 3 tables.

L'idée est ici de joindre les 3 tables dans la même requête par le code suivant :

```
-- Afficher 1 table "virtuelle" reprenant les 3 tables
SELECT
  *
FROM
  acteurs, films, filmographie
WHERE
  fg_fk_acteur_id = a_id
    AND
      fg_fk_film_id = f_id;
```

- ✓ **Code les requêtes suivantes, chacune devant être sauvegardées dans un fichier xx_tp3-N.sql** (N : numéro de la requête ci-dessous). Tu peux vérifier ta requête dans le terminal sqlite avec la commande suivante :

```
sqlite> .read xx_tp3-N.sql
```

1. (*) La liste de tous les films où a joué Johnny Deep.

Pour cela, tu peux procéder par étapes :

- ✓ Trouver l'identifiant de Johnny Deep

```
sqlite>
SELECT
  a_id
FROM
  acteurs
WHERE
  nom  = 'Johnny Deep';
```

Résultat: lecture de la table des acteurs : a_id = 1

```
sqlite> 1
```

✓ Lister tous les films avec a_id = 1

```
sqlite >
SELECT
   titre
FROM
   films, filmographie
WHERE
   f_id=fg_fk_film_id
      AND fg_fk_acteur_id =1;
```

Résultat: lecture de la table des filmographies : f_id = 5 et 7

```
sqlite> Chocolat
Rango
```

✓ lister tous les films où a joué Johnny Deep

```
SELECT
   titre
FROM
   films, filmographie
WHERE
   f_id=fg_fk_film_id
      AND fg_fk_acteur_id = (SELECT
         a_id
      FROM
         acteurs
      WHERE
         nom  = 'Johnny Deep');
```

2. **(*) La liste des années où Johnny Deep a joué dans un film, ainsi que son rôle dans ce film.**

   Pour cela, tu peux procéder par étapes :

   - ✓ Trouver l'identifiant de Johnny Deep (a_id = 1)
   - ✓ Lister les rôles avec a_id = 1
   - ✓ Lister les années avec a_id = 1
   - ✓ Lister les rôles et les années avec a_id = 1
   - ✓ Lister les rôles et les années où Johnny Deep a joué

3. **(*) La liste de tous les films réalisés par le réalisateur du film "Le parrain". La requête ne doit pas contenir le nom de ce réalisateur.**

   Pour cela, tu peux procéder par étapes :

   - ✓ Trouver le réalisateur du parrain (Coppola)
   - ✓ Lister les films de Coppola
   - ✓ Lister les films réalisés par le réalisateur du films « Le parrain »

4. Re-écris les requêtes des tests 1 et 2 en utilisant un seul SELECT pour chacun.

5. La liste des films qui commencent par la chaîne de caractères "Le" ou contiennent la chaîne de caractères "de". (SELECT … WHERE … LIKE … %)

6. La liste des films ordonnée par année de réalisation, dans l'ordre descendant. (SELECT … ORDER BY … DESC)

7. Le nombre d'acteurs ayant joué dans le film "L'odyssée de Pi". (SELECT COUNT)

8. La liste des noms d'acteurs qui n'ont joué dans aucun film. (SELECT NOT IN)

9. La liste des noms d'acteurs ayant joué dans au moins dans un film avec la moyenne des salaires qu'ils ont touché sur tous leurs films. Nomme la colonne "Moyenne". Exemple : Si Johnny Deep a joué dans 2 films, et a touché 2 salaires s_1 et s_2, la moyenne sera $(s_1+s_2)/2$. (SELECT AVG)

10. La liste des paires d'acteurs ayant le même salaire. Une paire d'acteurs ne doit pas se trouver deux fois dans ton résultat. (SELECT a1.nom, a2.nom FROM acteurs a1, acteurs a2 …)

11. Les salaires dans la base de données sont en dollars. Afficher ces salaires en dirhams. 1 dollar coûte aux environs de 9 dirhams. (SELECT … AS)

4 TP4 – S'entraîner sur une BD simple : copains de classe

L'objectif de ce tp est de créer la BD des **exercices 2 & 11** sur les copains de classe, d'y rentrer les données ci-dessous et d'écrire les requêtes demandées.

nom	prénom	sexe	date de naissance	téléphone	adresse	no postal	ville	canton
Ochon	Paul	H	1995-08-08	324661155	Place des Peupliers 3	2900	Porrentruy	JU
Ochon	Eric	H	1995-08-08	324661155	Place des Peupliers 3	2900	Porrentruy	JU
Gross	Jean	H	1995-03-24	324668341	La condemène	2950	Courgenay	JU
Fonfec	Sophie	F	1994-12-24	324711230	Rue du Général-Comman 26	2950	Courgenay	JU
Camé	Léon	H	1995-01-02	273956619	Rue de la Scierie 1	1965	Savièse	VS
Darc	Jeanne	F	1995-01-31	273224614	Rue de la Scierie 1	1950	Sion	VS
Sapin	Noëlle	F	1996-03-14	219635678	Promenade des Pêcheurs 6	1820	Montreux	VD
Tamère	Dominique	F	1995-01-08	324666391	Vieille Rue 2	2900	Porrentruy	JU
Maillard	Colin	H	1994-12-31	324669912	Route de Varandin 9	2905	Courtedoux	JU
Nord	Paul	H	1996-01-21	324661762	Route de Montancy 332	2903	Villars-sur-	JU

4.1 Création de la base de données

- ✓ Crée une base de données **copainsclasses.db**
- ✓ Ouvre le fichier **copainsclasses.ods** fourni dans le drive.

  Assure-toi que les cellules:

  - "dates de naissance" sont du type date
  - "téléphone" et "no postal" sont du type entier

  Sauvegarde le fichier sous **copainsclasses.cvs**

- ✓ A partir de **copainsclasses.cvs**, crée les **fichiers de requêtes SQL** suivants:
  - **create_copainsclasse.sql** contenant les requêtes de création des tables COPAINS, VILLES, HABITE

⇨ **Pense à respecter les règles d'écriture et de travail vues précédemment !**

Vérifie ton travail en affichant les tables

- **insert_copains.sql** contenant les requêtes des enregistrements dans la table COPAINS
  Vérifie ton travail en affichant le contenu de COPAINS
- **insert_villes.sql** contenant les requêtes des enregistrements dans la table VILLES
  Vérifie ton travail en affichant le contenu de VILLES
- **insert_habite.sql** contenant les requêtes des enregistrements dans la table HABITE

Vérifie ton travail en affichant le contenu de HABITE

- ✓ **Crée un fichier setup_copainsclasse.sql reprenant l'ensemble des requêtes de création de la BD ainsi que l'affichage des tables et de leur contenu.**

Remarque :

Les 3 1ères lignes améliorent l'affichage des résultats des requêtes et peuvent être insérées dans le fichier **.sqliterc** sous /home/user (voir les explications p2)

4.2 Requêtes de recherche dans la BD

Avec des commandes SQL, recherche dans la base de données de copainsclasses.db :

- Test1 (*) : les noms de famille de tous les Paul ;
- Test2 (*) : le numéro de téléphone de Sophie Fonfec ;
- Test3 : les noms et prénoms de tous ceux nés avant 1995 ;
- Test4 : les noms et prénoms de tous ceux qui sont nés en janvier 1995 ;
- Test5 (*) : les noms et prénoms de tous ceux qui habitent Porrentruy ;
- Test6 : le nombre de non-jurassiens ;
- Test7 (*) : les noms et prénoms de toutes les valaisannes (habitantes de Villars).