5. Bases de données

Introduction

5.0.1 to Excel or not to Excel ?

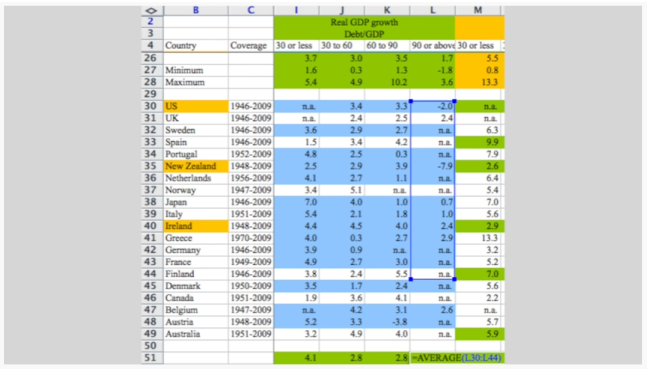
La plupart des gens et des sociétés utilisent un tableur [1] pour gérer leurs données et s'en sortent *très bien*. Très bien, dans le sens où c'est un outil disponible et faussement simple que l'on pense maîtriser. Voici deux articles à lire

5.0.1.1 Finance, espionnage, génétique : ces mauvaises manipulations d'Excel qui ont conduit à des désastres

Cet article est édifiant, mais la plupart des erreurs signalées correspondent plus à des erreurs de manipulation ou de méconnaissance du logiciel Excel®.
Quel que soit le système choisit, il est limité, il peut y avoir des erreurs de manipulation ou des erreurs de programmation. Malgré tout, il démontre quelque chose d'important. On croit connaître Excel® et on aime bien se reposer sur un outil que l'on croit connaître, mais si on veut réellement travailler sur des données sérieuses, mieux vaut faire appel à un spécialiste [2].

1. Le plus connu étant Excel®, pour ne pas le nommer...
2. qui je l'espère utilisera un autre outil !

5.0.1.2 Les politiques d'austérité, à cause d'une erreur sous Excel ?



	B	C	I	J	K	L	M
2			Real GDP growth				
3			Debt/GDP				
4	Country	Coverage	30 or less	30 to 60	60 to 90	90 or above	30 or less
26			3.7	3.0	3.5	1.7	5.5
27	Minimum		1.6	0.3	1.3	-1.9	0.8
28	Maximum		5.4	4.9	10.2	3.6	13.3
29							
30	US	1946-2009	n.a.	3.4	3.3	-2.0	n.a.
31	UK	1946-2009	n.a.	2.4	2.5	2.4	n.a.
32	Sweden	1946-2009	3.6	2.9	2.7	n.a.	6.3
33	Spain	1946-2009	1.5	3.4	4.2	n.a.	9.9
34	Portugal	1952-2009	4.8	2.5	0.3	n.a.	7.9
35	New Zealand	1948-2009	2.5	2.9	3.9	-7.9	2.6
36	Netherlands	1956-2009	4.1	2.7	1.1	n.a.	6.4
37	Norway	1947-2009	3.4	5.1	n.a.	n.a.	5.4
38	Japan	1946-2009	7.0	4.0	1.0	0.7	7.0
39	Italy	1951-2009	5.4	2.1	1.8	1.0	5.6
40	Ireland	1948-2009	4.4	4.5	4.0	2.4	2.9
41	Greece	1970-2009	4.0	0.3	2.7	2.9	13.3
42	Germany	1946-2009	3.9	0.9	n.a.	n.a.	3.2
43	France	1949-2009	4.9	2.7	3.0	n.a.	5.2
44	Finland	1946-2009	3.8	2.4	5.5	n.a.	7.0
45	Denmark	1950-2009	3.5	1.7	2.4	n.a.	5.6
46	Canada	1951-2009	1.9	3.6	4.1	n.a.	2.2
47	Belgium	1947-2009	n.a.	4.2	3.1	2.6	n.a.
48	Austria	1948-2009	5.2	3.3	-3.8	n.a.	5.7
49	Australia	1951-2009	3.2	4.9	4.0	n.a.	5.9
50							
51			4.1	2.8	2.8	=AVERAGE(L30:L44)	

Les politiques d'austérité : à cause d'une erreur Excel ?

Un dernier exemple d'erreur de manipulation (mais pas que [3]) d'un fichier Excel® aux conséquences dramatiques.

5.0.2 Des requêtes

Intéressons nous maintenant à l'utilisation des données. Le but est de stocker un maximum d'informations pour ensuite effectuer ce qu'on appelle une requête, autrement dit, on interroge cette base de données pour obtenir des informations précises selon certains critères.

Nous allons commencer par faire quelques requêtes.

exercice 5.1 Allez sur le site `https://inducks.org`, (International Network for Disney-Universe Comic Knowers and Sources) une énorme base de données open source sur les publications de bandes dessinées Disney. Effectuez quelques requêtes. ■

Nous allons maintenant essayer d'effectuer quelques requêtes *simples* à l'aide d'un tableur.

exercice 5.2 — Indian Food. Nous allons travailler sur le fichier `indian_food.csv` provenant du site de bases de données ouvertes kaggle. Le document est dans le dossier compagnon de cours que avez téléchargé sur l'espace habituel.

1. Télécharger le fichier CSV et l'ouvrir avec son tableur préféré. Analyser le document (contenu, descripteur, ...)
2. Chercher tous les plats provenant de la région **East**.
3. Chercher tous les plats provenant de la région **East** en n'affichant que le **nom du plat** et **ses ingrédients**.
4. Compter le nombre de **plat végétarien**.
5. Chercher le nombre de **plat végétarien** ne contenant pas de **lait**.

■

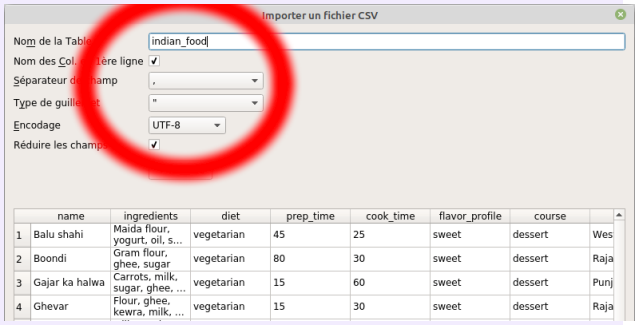
Vous avez normalement dû avoir quelques difficultés pour effectuer quelques-unes de ces requêtes qui semblent pourtant relativement simples... Essayons maintenant d'utiliser un autre outil : **SQL** [4].
Le langage **SQL** est un langage que nous allons découvrir ensemble tout au long de ce chapitre.

3. Les scientifiques ne sont pas tous d'honnêtes personnes. On pourra rigoler en regardant cet article sur une étude prouvant que la chloroquine protège des accidents de trottinette, parue dans un journal scientifique « très sérieux ».
`https://www.francetvinfo.fr/sante/maladie/coronavirus/chloroquine/comment-une-etude-fumeuse-sur-les-trottinettes-e4073359.html`

4. SQL pour *Structured Query Language*, en français *langage de requête structurée*.

exercice 5.3 — Indian Food II. Ouvrir le fichier `indian_food.csv` avec le logiciel *DB Browser for SQLite*.
Une fenêtre s'ouvre.
Vérifier que vous avez bien les options suivantes :

- « Nom des Col en 1ère ligne » : ✓
- « Séparateur de champ » : ,
- « Type de guillemet » : "
- « Encodage » : UTF-8



	name	ingredients	diet	prep_time	cook_time	flavor_profile	course	
1	Balu shahi	Maida flour, yogurt, oil, s...	vegetarian	45	25	sweet	dessert	Wes
2	Boondi	Gram flour, ghee, sugar	vegetarian	80	30	sweet	dessert	Raja
3	Gajar ka halwa	Carrots, milk, sugar, ghee, ...	vegetarian	15	60	sweet	dessert	Punj
4	Ghevar	Flour, ghee, kewra, milk, ...	vegetarian	15	30	sweet	dessert	Raja

Vous allez pouvoir faire les quelques requêtes suivantes.
AVERTISSEMENT : le but de cet exercice n'est pas d'apprendre le langage **SQL** mais de découvrir avec quelle facilité on peut effectuer des requêtes. On reviendra sur ce langage plus tard.

1. Allez dans l'onglet « Exécuter le SQL ».
   Comme vous allez le constater, pour afficher des commentaires en **SQL** il suffit de mettre deux tirets.

2. Pour chercher tous les plats provenant de la région **East** :

```
5  --2) plats venant de l'East
6  SELECT *
7    FROM indian_food
8    WHERE region = 'East';
```

3. Pour chercher tous les plats provenant de la région **East** en n'affichant que le **nom du plat** et **ses ingrédients** :

```
10 --3) uniquement le nom et les ingré
       dients des plats venant de l'
       East
11 SELECT name, ingredients
12   FROM indian_food
13   WHERE region = 'East';
```

4. Pour compter le nombre de **plat végétarien** :

```
15 --4) nombre de plats végétariens
16 SELECT COUNT(*)
17   FROM indian_food
18   WHERE diet = "vegetarian";
```

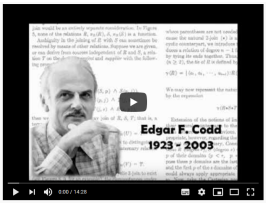
5. Pour chercher le nombre de **plat végétarien** ne contenant pas de **lait** :

```
20 --5) nombre de plats végétariens ne
       contenant pas de lait
21 SELECT COUNT(*)
22   FROM indian_food
23   WHERE (diet = "vegetarian")
24   AND (ingredients NOT LIKE '%Milk%
       ');
```

■

5.0.3 Une vidéo introductive

Pour finir cette introduction, voici une vidéo qui va vous présenter tout ce que nous allons voir ensemble dans ce chapitre.



5.1 Modèle relationnel

5.1.1 Un problème introductif

Soit « `Les voitures de De Funès` » une agence de location des plus belles voitures utilisées dans les meilleurs film de Louis de Funès. Toutes les informations nécessaires à la gestion des clients se trouvent dans un tableur (disponible dans le dossier à télécharger)

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	client	adresse	a payé	montant	début location	fin location	voiture rendue	modèle voiture	Prix de la location	caractéristiques	carburant	réparations à prévoir	voiture disponible
2	Louis de Funès	85 rue de la Barre	oui	150,00 €	05/08/22	08/08/22	oui	Citroën 2CV	50€ / jour	peinture bleue	essence		oui
3	Louis de Funès	85 rue de la Barre	oui	1 000,00 €	10/08/22	12/08/22	oui	Citroën DS 21 Pallas	500€ / jour	peinture noire avec un bateau sur le toit	essence	roue AD arrachée	non
4	Olivier de Funès	85 rue de la Barre	oui	800,00 €	10/08/22	12/08/22	oui	Cadillac DeVille 1964	400€ / jour	convertible	sans plomb	moteur explosé	non
5	Guy Grosso	1 boulevard Foch	non	2 250,00 €	13/08/22	22/08/22	oui	Citroën Méhari	250€ / jour	4 x 4 vert	diesel	siège griffé	non
6	Louis de Funès	85 rue de la Barre	oui	550,00 €	14/08/22	25/08/22	oui	Citroën 2CV	50€ / jour	peinture bleue	essence		oui
7	Michel Modo	3 boulevard Foch	non		20/08/22		non	Soucoupe volante	1500€ / jour	trop top	hydrogène		non
8	Olivier de Funès	85 rue de la Barre	non	1 250,00 €	20/08/22	25/08/22	oui	Citroën Méhari	250€ / jour	4 x 4 vert	diesel	inondée	non
9	Louis de Funès	85 rue de la Barre	oui	1 000,00 €	30/08/22	01/09/22	oui	Citroën DS 21 Pallas	500€ / jour	peinture noire avec un bateau sur le toit	essence	siège griffée	non
10	Olivier de Funès	85 rue de la Barre	non		30/08/22		non	Citroën DS	750€ / jour	voiture volante	électricité		oui

On peut faire les remarques suivantes :

- Il y a une erreur de saisie sur la cellule **M10**. En effet, la voiture n'a pas encore été rendue. Elle ne peut donc pas être disponible. On devrait pouvoir éviter ces erreurs de saisie en automatisant (programmation) certaines informations.
- Il y a beaucoup d'informations. Certaines sont redondantes.
  Par exemple, le client `Louis de Funès` n'a qu'une seule adresse. Ce n'est finalement pas si nécessaire de la répéter à chaque fois. Même chose pour les voitures (prix de la location, caractéristiques, carburant).
- Si vous travaillez dans cette agence, en fonction du service pour lequel vous travaillez, vous recevez un nombre important d'informations qui ne vous sont pas utiles.
  En effet, si vous vous occupez de l'accueil de la clientèle, vous avez juste besoin de savoir :
  — quel est le nom du client,
  — si la voiture qu'il désire emprunter est disponible,
  — s'il rend la voiture qu'il a emprunté,
  — si cette voiture est en état, sinon on renseigne les réparations à prévoir,
  — s'il souhaite payer immédiatement ou plus tard.
  Si vous vous occupez du service comptable, vous avez juste besoin de savoir :
  — quel est le nom du client,
  — s'il n'a pas payé depuis un moment,
  — s'il n'a pas rendu le véhicule depuis un moment,
  — quel est son adresse pour le contacter.
  Si vous êtes dans le garage, vous avez juste besoin de savoir :
  — s'il y a des réparations à prévoir,
  — sur quelle voiture.
- On constate aussi qu'il manque quelques informations. En effet, `Guy Grosso` a empruntée une `Citroën Mehari` du 13 au 22 août et `Olivier de Funès` aussi du 20 au 25 août. Manifestement, il y a plusieurs `Citroën Mehari` dans le parc automobile de cette agence, mais on ne les distingue pas. On retrouve le même problème avec la `Citroën DS 21 Pallas`. La première a été rendue le 12 août avec une roue arrachée. Elle n'est pas disponible et pourtant `Louis de Funès` reprend ce modèle le 30 août.
- Une remarque supplémentaire, nous ne savons pas quel est l'ensemble du parc automobile disponible dans cette agence. Saviez vous, entre autres, qu'il y a une `Chenard et Walcker T4 Torpédo` essence de 1925 et deux `Chevrolet Impala SS` décapotable de 1965, essence également ? ? ? ?

Aux vues de toutes ces remarques, on peut se dire que ce modèle *à plat*, avec un seul tableau n'est pas le meilleur modèle. On a en réalité besoin de plusieurs **relations**, décrites par leurs **attributs**. On peut par exemple proposé ce modèle avec 4 relations :

- une relation **Client** avec uniquement les caractéristiques des clients,
- une relation **Modèle** avec uniquement les caractéristiques des différents modèles de voiture,
- une relation **Location** avec uniquement le nom du client, la voiture, la date d'entrée et la date de sortie du véhicule, si le montant de location a été payé. Nous n'avons pas besoin du montant de la location car on peut le calculer lorsque l'on connaît la date d'arrivée, la date de retour et le prix de la location.
- une relation **Voiture** avec uniquement l'état des voitures du parc automobile.

Nous pourrions ainsi avoir ces 4 relations :

Client
nom
prénom
adresse

Modèle
nom
caractéristiques
prix de la location
carburant

Location
nom du client
voiture
début de la location
fin de la location
a payé

Voiture
modèle
réparation à prévoir
disponible

5.1.2 Contraintes d'intégrité

Les **contraintes d'intégrité** sont des propriétés logiques, préservées à tout instant par la base de données et qui garantissent la cohérence des données. Autrement dit, une contrainte d'intégrité est un invariant de la base de données. On distingue 4 grandes catégories de contraintes d'intégrité jouant des rôles complémentaires.

5.1.3 Contrainte de domaine

Pour chaque **attribut** d'une **relation** ont va lui attribuer un **domaine** parmi, entre autres, les domaines suivants :

- `String`, une chaîne de caractères ;
- `Int`, un entier ;
- `Boolean`, une valeur booléenne `True` ou `False` ;
- `Float`, un nombre (à virgule) flottante ;
- `Date`, une date au format `jour/mois/année` ;
- `Time`, un instant au format `heure:minute:seconde`.

Le choix du domaine doit permettre

- de représenter exactement et sans perte d'information toutes les valeurs possibles pour un attribut ;
- de limiter autant que possible la saisie de valeurs illégales ou mal formées.

Si nous revenons à notre exemple, nous avons les **contraintes de domaines** suivantes :

Client	Domaine
nom	`String`
prénom	`String`
adresse	`String`

Modèle	Domaine
nom	`String`
caractéristiques	`String`
prix de la location	`Float`
carburant	`String`

Location	Domaine
nom du client	`String`
voiture	`String`
début de la location	`Date`
fin de la location	`Date`
a payé	`Boolean`

Voiture	Domaine
modèle	`String`
réparation à prévoir	`String`
disponible	`Boolean`

Rmq On peut aussi représenter les **relations** sous la forme de tuples.

- *Client(nom `String`, adresse `String`)*
- *Modèle(nom `String`, caractéristiques `String`, prix de la location `Float`, carburant `String`)*
- *Location(nom du client `String`, voiture `String`, début de la location `Date`, fin de la location `Date`, a payé `Boolean`)*
- *Voiture(modèle `String`, réparation à prévoir `String`, disponible `Boolean`)*

5.1.4 Contrainte d'entité

La **contrainte d'entité** permet de s'assurer que chaque élément d'une relation est **unique** et identifie une **entité** de manière unique et non ambiguë. Par exemple dans la relation `Client`, on doit choisir un (ou plusieurs) attributs pour caractériser de manière unique chaque clients de la base. On pourrait choisir l'attribut `nom`, mais chacun sait qu'il y a plusieurs clients qui portent le même nom `de Funès`, `Louis` et `Olivier`. Donc cet attribut seul ne suffira pas. Souvent on rajoute un deuxième attribut comme le `prénom`. La plupart du temps, le couple `nom`, `prénom` suffit à identifier une personne de manière unique. Mais rien ne nous assure qu'il n'existe pas un autre Louis de Funès.
La solution consiste alors à rajouter un nouvel attribut, un `numéro de client`.
L'attribut qui permet d'identifier chaque entité d'une relation de manière unique s'appelle une **clé primaire**.
Pour la relation `Modèle`, l'attribut `nom` peut être une très bonne **clé primaire**.
Pour la relation `Location`, on devra rajouter un nouvel attribut `numéro location` comme **clé primaire**[5].
Pour la relation `Voiture`, on devra rajouter un nouvel attribut `numéro voiture` comme **clé primaire** pour différencier toutes les voitures.

Notre schéma devient alors :

Client	Domaine	Clé
numéro client	`Integer`	Primaire
nom	`String`	
prénom	`String`	
adresse	`String`	

Modèle	Domaine	Clé
nom	`String`	Primaire
caractéristiques	`String`	
prix de la location	`Float`	
carburant	`String`	

Location	Domaine	Clé
numéro location	`Integer`	Primaire
nom du client	`String`	
voiture	`String`	
début de la location	`Date`	
fin de la location	`Date`	
a payé	`Boolean`	

Voiture	Domaine	Clé
numéro voiture	`Integer`	Primaire
modèle	`String`	
réparation à prévoir	`String`	
disponible	`Boolean`	

Rmq Dans la représentation en tuple, la clé primaire est soulignée.

- *Client(<u>numéro client `Integer`</u>, nom `String`, adresse `String`)*
- *Modèle(<u>nom `String`</u>, caractéristiques `String`, prix de la location `Float`, carburant `String`)*
- *Location(<u>numéro location `Integer`</u>, nom du client `String`, voiture `String`, début de la location `Date`, fin de la location `Date`, a payé `Boolean`)*
- *Voiture(<u>numéro voiture `Integer`</u>, modèle `String`, réparation à prévoir `String`, disponible `Boolean`)*

5. Une combinaison `nom du client`, `voiture`, `début location` devrait aussi donner une bonne **clé primaire**, mais c'est tout de même plus simple de donner un numéro unique à chaque location.

5.1.5 Contrainte de référence

Les **clefs primaires** ne servent pas uniquement à distinguer les entités de manière unique. Elles permettent aussi à servir de référence dans une autre relation.
Par exemple dans la relation `Location`, l'attribut `nom du client` doit faire référence à la relation `Client`. Pour lier ces deux relations, on va donc remplacer l'attribut `nom du client` par la **clé primaire** de la relation `Client`. L'attribut `numéro de client`, **clé primaire** de la relation `Client` crée le lien avec la **relation** `Location`. Il est alors une **clé étrangère** de la relation `Location`.
De la même manière, dans la relation `Location`, l'attribut `voiture` peut être remplacé par la **clé primaire** de la relation `Voiture`. Il est alors une **clé étrangère** de la relation `Location`.
Enfin, dans la relation `Voiture`, l'attribut `modèle` peut être remplacé par la **clé primaire** de la relation `Modèle`. Il est alors une **clé étrangère** de la relation `Voiture`.

On a :

Client	Domaine	Clé
numéro client	`Integer`	Primaire
nom	`String`	
prénom	`String`	
adresse	`String`	

Modèle	Domaine	Clé
nom	`String`	Primaire
caractéristiques	`String`	
prix de la location	`Float`	
carburant	`String`	

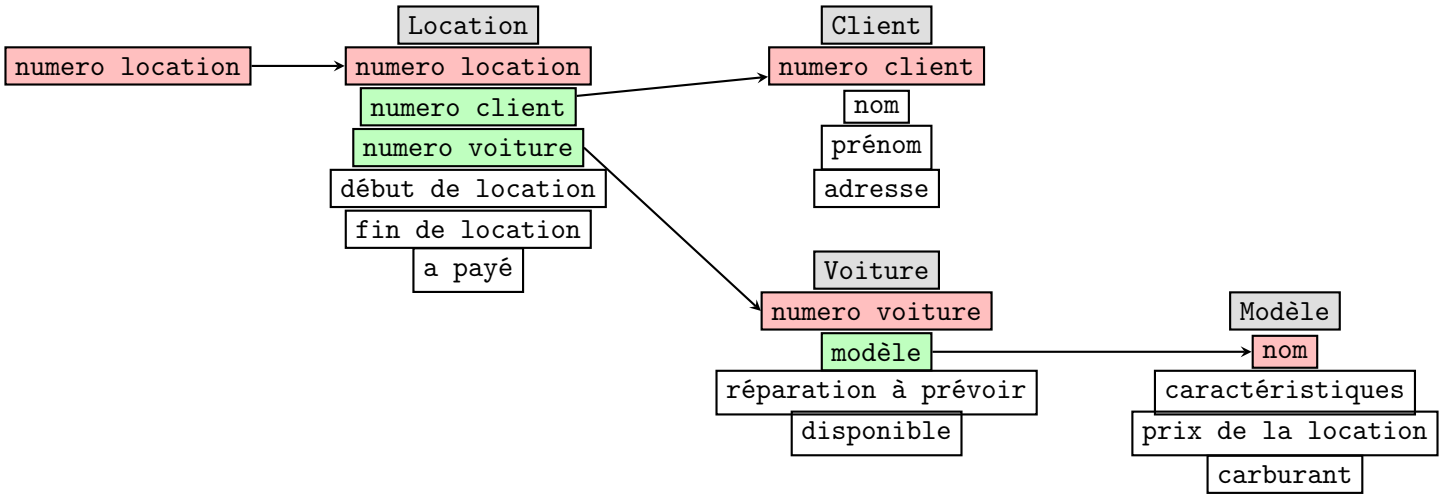
Location	Domaine	Clé
numéro location	`Integer`	Primaire
numéro client	`Integer`	Étrangère(Client)
numéro voiture	`Integer`	Étrangère(Voiture)
début de la location	`Date`	
fin de la location	`Date`	
a payé	`Boolean`	

Voiture	Domaine	Clé
numéro voiture	`Integer`	Primaire
modèle	`String`	Étrangère(Modèle)
réparation à prévoir	`String`	
disponible	`Boolean`	

Rmq Dans la représentation en tuple, la clé étrangère est soulignée en pointillé.

- *Client(numéro client `Integer`, nom `String`, adresse `String`)*
- *Modèle(nom `String`, caractéristiques `String`, prix de la location `Float`, carburant `String`)*
- *Location(numéro location `Integer`, numero client `Integer`, numéro voiture `Integer`, début de la location `Date`, fin de la location `Date`, a payé `Boolean`)*
- *Voiture(numéro voiture `Integer`, modèle `String`, réparation à prévoir `String`, disponible `Boolean`)*

Maintenant que nous avons nos **clefs étrangères**, pour une location, à partir du `numéro location` on a accès à la relation `Location`, puis au `numéro client` et donc à la relation `Client` et ainsi au `nom`, au `prénom` et à l'`adresse` du client... etc ...



5.1.6 Contraintes utilisateurs

Pour finir, il reste les **contraintes utilisateurs** aussi appelées les **contraintes métiers**. Ce sont toutes les contraintes restantes.
Par exemple pour la relation `Client`, les attributs `nom` et `adresse` ne doivent pas être nuls.
Pour la relation `Location`, entres autres, l'attribut `début de la location` doit être antérieur à l'attribut `fin de la location`.
Pour la relation `Modèle`, le `prix de la location` ne doit pas être négatifs.
... etc ...

5.1.7 Attention!!!!

On remarquera peut-être plus tard qu'avec le langage SQL certaines de ces contraintes ne sont pas obligatoires, notamment les contraintes d'entité et de référence. Par contre si on n'essaye pas de respecter ces différentes contraintes on risque d'avoir une base peu fonctionnelle voire inutilisable.
La première chose à faire avant de créer une base de données est donc de bien réfléchir à son modèle relationnel en respectant les contraintes d'intégrité.

5.1.8 Exercices

Ces exercices sont issues du livre NUMÉRIQUE ET SCIENCES INFORMATIQUES - TERMINALE *de Thibault Balabonski, Sylvain Conchon, Jean-Christophe Filliâtre et Kim Nguyen*

exercice 5.4 On souhaite modéliser un annuaire téléphonique simple dans lequel chaque personne (identifiée par son nom et son prénom) est associée à son numéro de téléphone.
Proposer une modélisation relationnelle de cet annuaire. ■

exercice 5.5 On considère la solution proposée à l'exercice précédent. Dire si chacun des ensembles est une relation valide pour le schéma `Annuaire`.

1. `{}`
2. `{('Titi', 'Toto', '0123456789')}`
3. `{('Titi', 'Toto', '0123456789'), ('Doe', 'John', '0123456789')}`
4. `{('Titi', 'Toto', '0123456789'), ('Titi', 'Toto', '9876543210')}`
5. `{('Titi', 'Toto', '0123456789'), ('Doe', 'John')}`
6. `{('Titi', 'Toto', 0123456789)}`

■

exercice 5.6 Donner la modélisation relationnelle d'un bulletin scolaire. Cette dernière doit permettre de mentionner :

- des élèves, possédants un numéro d'étudiant alphanumérique unique
- un ensemble de matières fixées, mais qui ne sont pas données
- au plus une note sur 20, par matière et par élève.

On prendra soin de préciser toutes les contraintes utilisateurs qui ne peuvent êtres inscrites dans les schémas des relations. ■

exercice 5.7 On considère la solution proposée à l'exercice précédent. Dire si chacun des ensembles est une relation valide pour le schéma de la base de données du bulletin de notes.

1. ```
   Eleve = {}
   Matiere = {}
   Note = {}
   ```

2. ```
   Eleve = {('Titi', 'Toto', 'AB56789')}
   Matiere = {('NSI',0), ('Sport',1)}
   Note = {('AB56789',1,17)}
   ```

3. ```
   Eleve = {('Titi', 'Toto', 'AB56789')}
   Matiere = {('NSI',0)}
   Note = {('AB56789',1,17)}
   ```

4. ```
   Eleve = {('Titi', 'Toto', 'AB56789')}
   Matiere = {('NSI',0)}
   Note = {('AB56789',0,17), ('AB56789',0,18)}
   ```

5. ```
   Eleve = {('Titi', 'Toto', 'AB56789')}
   Matiere = {('NSI',0), ('Sport',1)}
   Note = {('AB56789',0,17), ('AB56789',1,17)}
   ```

■

exercice 5.8 Modéliser des informations sur les départements français. Pour chaque département on veut stocker son nom, son code, son chef-lieu et la liste de tous les départements voisins.
Attention, les codes de département sont tous des nombres, sauf la Corse du Sud et la Haute Corse qui ont respectivement les codes 2A et 2B. Les départements d'Outre-Mer ont un code à 3 chiffres (de 971 à 976).
Proposer une contrainte utilisateur permettant d'éviter la redondance d'information dans la liste des voisins. ■

exercice 5.9 Proposer une modélisation pour un réseau de Bus. Cette dernière doit être suffisamment riche pour permettre de générer, pour chaque arrêt de Bus du réseau, une fiche horaire avec tous les horaires de passage de toutes les lignes de Bus qui desservent l'arrêt. ■

exercice 5.10 On considère deux relations *R(a Integer, b Integer, c Integer)* et *S(a Integer, e Integer)* où l'attribut `a` de `S` est une clé étrangère faisant référence à `a` de `R`.
Dire si les affirmations suivantes sont vraies ou fausses, en justifiant.

1. Les `a` de `R` sont tous deux à deux distincts.
2. Les `b` de `R` sont tous deux à deux distincts.
3. Les `a` de `S` sont tous deux à deux distincts.
4. Les `e` de `S` sont tous deux à deux distincts.
5. `S` peut être vide alors que `R` est non vide.
6. `R` peut être vide alors que `S` est non vide.

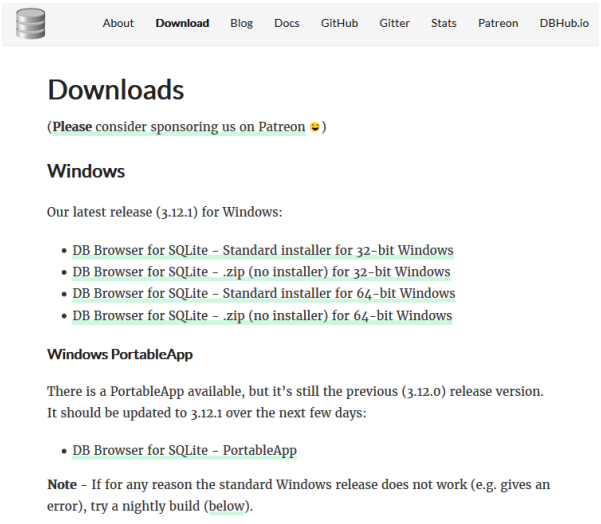
■
   ```

5.2 Le langage SQL

Ce qui a été vu dans la première partie est un modèle mathématiques permettant de raisonner sur des données tabulées. Il doit ensuite être mis en œuvre par un logiciel particulier, un **système de gestion de bases de données**, **SGBD** en abrégé. L'écrasante majorité des SGBD utilisent un langage du type **SQL** (*Structured Query Language*, langage de requête structuré). Ce dernier permet d'envoyer des ordres au SGBD de deux natures différentes :

- les mises à jour permettant la création de relations, l'ajout d'entité dans ces dernières, leur modification et leur suppression ;
- les requêtes permettant de récupérer les données répondant à des critères particuliers.

Nous allons dans la suite utiliser une variation du langage **SQL**, **SQLite** avec le logiciel SQLiteBrowser. Le Grand Sage vous demande donc d'installer la version portable sur votre clef USB.
https://sqlitebrowser.org/dl/



Si vous souhaitez revoir comment fonctionne ce logiciel, il existe un petit tutoriel proposé par David Roche, professeur de NSI à Grenoble :
https://pixees.fr/informatiquelycee/n_site/nsi_term_bd_sql.html

5.2.1 Quelques remarques générales

- En SQL, on ne fait aucune différence entre les minuscules et les majuscules, mais par convention, toutes les instructions sont écrites en `MAJUSCULE` et tous les attributs en `minuscule`.
- On ne doit pas utiliser de caractères spéciaux et l'espace est simulé par un `_`.
- Les retours à la ligne ne sont pas nécessaires, mais par convention, on passe à la ligne après chaque couple **mot-clef / champ d'application**.
- On termine chaque instruction par un point virgule `;`
- Les commentaires sont précédés de deux tirets `--`

5.2.2 Les mises à jour

Nous allons reprendre notre exemple de l'agence de location de voiture **Les voitures de De Funès** pour appliquer toutes les instructions **qu'il faut connaître**.

5.2.2.1 Création d'une table

Si nous reprenons notre **relation** `Modele`, nous pouvons créer la première **table** :

```
CREATE TABLE Modele
(nom TEXT PRIMARY KEY ,
caracteristiques TEXT,
prix_location REAL,
carburant TEXT);
```

Plusieurs remarques :

- pour identifier la clef primaire, il suffit d'indiquer **PRIMARY KEY** sur l'attribut correspondant
- les domaines les plus souvent utilisés sont **INTEGER** pour entier, **REAL** pour un flottant, **TEXT** pour une chaîne de caractères de taille quelconque, **VARCHAR(n)** pour une chaîne d'au plus n caractères, **CHAR(n)** pour une chaîne d'exactement n caractères, **DATE** pour une date au format **'AAAA-MM-JJ'**, **TIME** pour une heure au format **'hh :mm :ss'** et enfin **BOOLEAN** pour un booléen qui peut être alors **'T'** ou **'F'**.

Pour les relations contenant une **clef étrangère**, il faut préciser la provenance de cette clef avec le mot **REFERENCES** puis le nom de la table et entre parenthèses, le nom de l'attribut.

```
CREATE TABLE Voiture
(numero_voiture INTEGER PRIMARY KEY,
modele TEXT REFERENCES Modele(nom),
prevoir_reparations TEXT,
disponible BOOLEAN);
```

Rmq Comme la table `Voiture` contient une clef étrangère appartenant à la table `Modele`, il faut bien évidemment avoir créer en premier la table `Modele` pour ensuite faire **REFERENCES** à cette table.

exercice 5.11 Créer les autres tables `Client` et `Location` ■

Rmq Pour la table `Client`, on peut demander à ce que la clef primaire s'auto-incrémente :

```
CREATE TABLE Client
(numero_client INTEGER PRIMARY KEY AUTOINCREMENT,
...
);
```

5.2.2.2 Insertion dans une table

Pour insérer de nouvelles entités dans une table. Les valeurs de chaque entité doivent être alors données dans le même ordre que celui des attributs lors de la création de la table.

```
INSERT INTO Modele VALUES
('Citroën 2CV', 'peinture bleue', 50, 'essence'),
('Citroën DS 21 Pallas', 'peinture noire avec un bateau sur le toit', 500, '
    essence');
```

On peut aussi spécifier les différents attributs de la table. On peut alors modifier l'ordre des attributs. Cette écriture est aussi utile si on a la clef primaire qui s'autoincrémente, il ne faut dans ce cas pas préciser sa valeur.

```
INSERT INTO Modele(nom, caracteristiques, carburant, prix_location) VALUES
('Cadillac DeVille 1964', 'convertible', 'sans plomb', 400);
```

Vous devez respecter la contrainte d'entité et donc ne pas donner deux attributs identiques sur la clef primaire à deux entités différentes :

```
INSERT INTO Modele VALUES
('Citroën DS 21 Pallas', 'peinture noire', 300, 'essence');
```

On obtient le message d'erreur suivant :

```
L'exécution s'est terminée avec des erreurs.
Résultat : UNIQUE constraint failed: modele.nom
À la ligne 1 :
INSERT INTO Modele VALUES
('Citroën DS 21 Pallas', 'peinture noire', 300, 'essence');
```

exercice 5.12 Remplir les 4 tables avec des entités correspondants aux fichiers CSV donné précédemment. ■

5.2.2.3 Suppression de tables

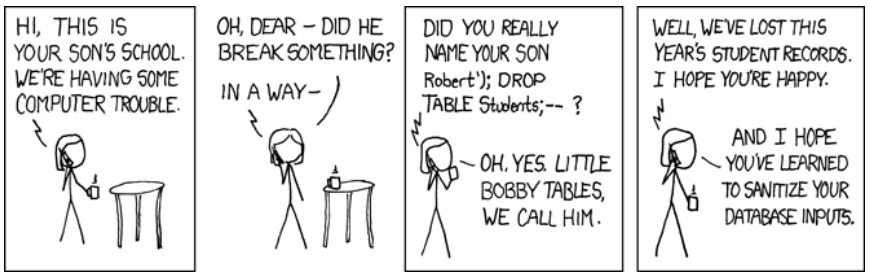
On peut supprimer des tables :

```
DROP TABLE Voiture;
```

Par contre, on ne peut pas supprimer une table si sa clef primaire est utilisée comme une clef secondaire dans une autre table :

```
DROP TABLE Client;
```

devrait, renvoyer un message d'erreur...



Source xkcd :https://xkcd.com/327/

5.2.2.4 Respect des contraintes d'intégrité

- Pour la **contrainte de domaine**, nous l'avons vu, c'est spécifié au moment de la création de la table.
- De même, pour la **contrainte d'entité**, nous avons vu qu'il suffit d'écrire **PRIMARY KEY** après l'attribut correspondant.
  Si on utilise plusieurs attributs pour former la clef primaire, on peut le spécifier à la fin de la création de la table :

```
CREATE TABLE Ami
(nom TEXT,
prenom TEXT,
adresse TEXT,
nu_tel TEXT,
PRIMARY KEY (nom, prenom));
```

- A nouveau, pour la **contrainte de référence**, il suffit d'écrire **REFRENCE** puis la table, puis entre parenthèses le nom de la clé primaire utilisée comme clef étrangère.
- Pour les **contraintes utilisateurs**, on peut :
  — préciser qu'un champ ne doit pas être vide en écrivant **NOT NULL**.

```
CREATE TABLE Modele
(nom TEXT PRIMARY KEY ,
caracteristiques TEXT NOT NULL,
prix_location REAL NOT NULL,
carburant TEXT NOT NULL);
```

  Évidemment, la clef primaire a pour but d'identifier de manière unique chaque entité, le champ correspondant ne peut pas être vide, ce n'est donc pas la peine de le préciser.
  — demander à vérifier au moment de la saisie certaines contraintes avec le mot **CHECK** suivi d'une formule booléenne.
  Par exemple, le prix de la location ne doit pas être négatif...

```
CREATE TABLE Modele
(nom TEXT PRIMARY KEY ,
caracteristiques TEXT NOT NULL,
prix_location REAL NOT NULL,
carburant TEXT NOT NULL,
CHECK (prix_location>0));
```

exercice 5.13 Réécrire les différentes tables en rajoutant des contraintes utilisateurs. ■

Rmq Pour modifier ou supprimer une entité, le code ressemble à celui d'une requête. Nous le verrons donc à ce moment-là.

5.2.3 Les requêtes

Une fois la base créée il ne reste plus qu'à l'interroger.
La structure générale est la suivante :

```
SELECT attribut1, attribut2, ...
FROM nom de la table
WHERE expression_booléene;
```

Autrement dit :

- SELECT pour demander les attributs souhaités ;
- FROM pour préciser la table contenant ces attributs ;
- WHERE pour préciser la condition que les attributs doivent respecter.

■ **Example 5.1** Pour obtenir la liste des différents modèles et leur prix de location, il suffit d'écrire :

```
SELECT nom, prix_location
FROM Modele;
```

Mais si vous avez des moyens financiers « limités », vous pouvez demander les modèles dont le prix de location se situe entre 100 et 400 euros par jour :

```
SELECT nom, prix_location
FROM Modele
WHERE prix_location >= 100 AND prix_location <= 400;
```

On obtient le résultat suivant :

nom	prix_location
Cadillac DeVille 1964	400.0
Citroën Méhari	250.0

■

5.2.3.1 Sur la clause SELECT

Si on veut tous les attributs de la table il suffit d'utiliser le symbole * qui signifie ALL. Ainsi la requête suivante permet d'obtenir toutes les informations sur tous les modèles diesel.

```
SELECT *
FROM Modele
WHERE carburant = 'diesel';
```

On obtient le résultat :

nom	caracteristiques	prix_location	carburant
Citroën Méhari	4 x 4 vert	250.0	diesel

5.2.3.2 Sur la clause `WHERE`

La clause `WHERE` doit être suivi d'une expression booléenne à l'aide des opérateurs <, <=, >, >=, = et **<>**, des opérateurs arithmétiques +, -, *, /, %, de constantes, de noms d'attributs, d'opérateurs logiques `AND`, `OR` et `NOT` et enfin d'opérateurs spéciaux tels `LIKE`[6].
Par exemple, si on veut tous les différents modèles de Citroën, on procède ainsi :

```
SELECT nom
FROM Modele
WHERE nom LIKE '%citroën%';
```

On obtient le résultat :

nom
Citroën 2CV
Citroën DS
Citroën DS 21 Pallas
Citroën Méhari

5.2.3.3 Tri et suppression des doublons (limite hors-programme mais vu au BAC)

Si on souhaite obtenir les noms des différents modèles ainsi que leur prix de location du moins cher au plus cher, on rajoute à la fin la clause `ORDER BY` avec le mot-clef `ASC` pour ascendant (plus petit au plus grand) ou `DESC` pour descendant (du plus grand au plus petit).

```
SELECT nom, prix_location
FROM Modele
ORDER BY prix_location ASC;
```

On obtient le résultat :

nom	prix_location
Citroën 2CV	50.0
Citroën Méhari	250.0
Cadillac DeVille 1964	400.0
Citroën DS 21 Pallas	500.0
Chevrolet Impala SS	600.0
Citroën DS	750.0
Soucoupe volante	1500.0
Chenard et Walcker T4 Torpédo	1500.0

6. Cet opérateur est très certainement hors-programme. On doit l'utilise pour savoir si une chaîne de caractères est inclues dans une autre. On peut lui associer le % qui dans une chaîne signifie « joker » il peut donc être remplacé par n'importe quelle chaîne.

Maintenant si on ne souhaite obtenir que les prix de location, on ne met que l'attribut `prix_location` mais on obtiendra 2 fois la valeur `1500.0`. Pour éviter les répétitions, on précise `DISTINCT` devant l'attribut.

```
SELECT DISTINCT prix_location
FROM Modele
ORDER BY prix_location ASC;
```

On obtient le résultat :

prix_location
50.0
250.0
400.0
500.0
600.0
750.0
1500.0

5.2.3.4 Fonction d'agrégation (limite hors-programme mais vu au BAC)

On peut rajouter à la clause `SELECT` ce que l'on appelle des **fonctions d'agrégation** qui permettent d'appliquer une fonction à l'ensemble des valeurs. En voici quelques-unes :

- La fonction `COUNT` permet de compter tous les éléments satisfaisant une requête.
  Par exemple pour compter le nombre de modèles, on peut procéder ainsi :

```
SELECT COUNT(*)
FROM Modele;
```

- La fonction `SUM` permet d'effectuer la somme de tous les éléments satisfaisant une requête.
  Par exemple pour compter le total de toutes les prix de location (ce qui n'a aucun intérêt), on peut procéder ainsi :

```
SELECT SUM(prix_location)
FROM Modele;
```

- La fonction `AVG` permet d'effectuer la moyenne de tous les éléments satisfaisant une requête.
  Par exemple pour obtenir le prix moyen d'une location, on peut procéder ainsi :

```
SELECT AVG(prix_location)
FROM Modele;
```

- La fonction `MIN` (respectivement `MAX`) permet d'obtenir la valeur minimale (respectivement maximale) de tous les éléments satisfaisant une requête.
  Par exemple pour obtenir le prix minimum d'une location, on peut procéder ainsi :

```
SELECT MIN(prix_location)
FROM Modele;
```

exercice 5.14 Effectuer les requêtes suivantes :

1. La liste des voitures disponibles.
2. Le nombre de voitures disponibles.
3. La liste des voitures avec leur numéro ayant des réparations à prévoir. On affichera également les réparations à prévoir.
4. La liste des modèles non diesel.
5. La liste des modèles non diesel avec leur prix de location par ordre décroissant.
6. La liste des voitures avec leur caractéristiques de peinture noire.

■

5.2.4 Jointure

Pour le moment nous n'avons effectuer des requêtes que sur une unique table, mais la plupart des informations les plus intéressantes sont à cheval sur plusieurs tables. Par exemple, pour la location n°6, la voiture n'a toujours pas été rendue, mais qu'elle est cette voiture manquante ? Nous allons devoir faire ce qu'on appelle une **jointure** entre la table `Location` et la table `Voiture` avec la clause `JOIN` suivi de la table à lier puis `ON` suivi du lien entre les tables.
Le lien entre les tables provient des clefs étrangères.

Voici la requête :

```
SELECT modele
FROM Voiture
JOIN Location ON Voiture.n_voiture = Location.n_voiture
WHERE Location.n_location = 6;
```

On remarquera que pour éviter toute confusion sur le nom des attributs, on précise la table d'où provient l'attribut avec la syntaxe `Table.attribut`.

exercice 5.15 Effectuer les requêtes suivantes :

1. La location n°16 n'a pas été rendu. Quel est le nom et l'adresse du client.
2. Le nom et l'adresse de tous les clients qui n'ont pas rendu leur voiture de location.
3. Le nom le prénom et l'adresse de tous les clients qui n'ont pas encore payé.
4. Pour aider le client, afficher le prix des locations par ordre croissant et le modèle des voitures disponibles.
5. Le prix des locations des voitures disponibles par ordre croissant sans répétitions.
6. Pour aider le garagiste, le numéro de la voiture, le modèle, les caractéristiques et le carburant de toutes les voitures qui sont en attente de réparation.
7. On souhaite avoir la liste des numéros de location ainsi que le modèle de la voiture louée pour toutes les locations.
8. On souhaite avoir la liste des numéros de location ainsi que le modèle de la voiture louée pour toutes les locations qui ont débutées entre le 8 et 15 août 2022.

■

5.2.5 Modification des données

La structure pour modifier des données ressemble à celle des requêtes.

5.2.5.1 Suppression de lignes

Finalement, la voiture n°4 est complètement détruite. On va la supprimer de la table `Voiture` avec la clause `DELETE FROM`.

```
DELETE FROM Voiture
WHERE n_voiture = 4;
```

5.2.5.2 Mise à jour

`Olivier De Funès` a rendu la voiture pour la location n°9 le 3 septembre 2022. On va modifier la table `Location` avec la clause `UPDATE` suivie de `SET`.

```
UPDATE Location
SET fin_loc = '2022-09-03'
WHERE n_location = 9;
```

5.3 Le sujet 0

Vous pouvez retrouver l'ensemble du sujet 0 à l'adresse suivante : https://eduscol.education.fr/media/3962/download.

Cet exercice porte sur les bases de données relationnelles et le langage SQL.

L'énoncé de cet exercice utilise les mots du langage SQL suivant :

`SELECT`, `FROM`, `WHERE`, `JOIN`, `INSERT INTO`, `VALUES`, `COUNT`, `ORDER BY`.

Dans un lycée imaginaire, les données relatives aux élèves de secondes sont regroupées dans un fichier nommé `seconde_lyc.csv`. Un extrait de son contenu est représenté figure 1.

num_eleve	nom	prenom	datenaissance	langue1	langue2	option	classe
133310FE	ACHIR	Mussa	01/01/2005	anglais	espagnol		2A
156929JJ	ALTMEYER	Yohan	05/05/2005	allemand	anglais	théâtre	2D
500633KH	BELEY	Thibaut	05/05/2005	anglais	espagnol		2A
911887GA	BELEY	Marie	05/05/2005	anglais	espagnol		2A
906089JJ	BELEY	Manon	10/01/2005	anglais	allemand		2E
488697GA	CAILLE	Marie	30/03/2004	italien	anglais		2D
193514FB	CHARPENTIER	Jules	26/12/2005	espagnol	anglais		2C
321188FA	CLAUDEL	Benjamin	09/09/2005	espagnol	anglais		2E
081282GF	EISEN	Carla	23/06/2004	anglais	allemand		2A
026946KB	EL AYAR	Amir	11/09/2005	anglais	arabe	cinéma	2D
108303KG	GEHIN	Arthur	26/02/2005	allemand	anglais		2D
057934BK	GROSJEAN	Alexandre	09/11/2005	anglais	espagnol		2C
571113KE	HENRY	Paul	12/03/2005	allemand	anglais		2E
488820DE	JACQUEY	Marc	13/11/2005	anglais	italien		2D
024810CE	JULIANO	Alberto	21/04/2005	anglais	espagnol		2C
249992EJ	KLEIBER	Gusti	20/02/2005	anglais	espagnol	cinéma	2E
492698AF	LACOUR	Julie	06/04/2005	italien	anglais		2D
026454FA	LARBI	Nourdine	14/07/2005	espagnol	anglais		2C
309341GD	LEFZA	Yasmina	26/11/2005	espagnol	anglais		2E
076725HD	MARTIN	Victor	13/03/2005	anglais	espagnol		2A
815183CB	NGUYEN	Ngong	16/03/2005	anglais	espagnol		2D
094002FC	PELTIER	Romane	14/06/2005	allemand	anglais		2D
321262HD	RENAULT	Zoé	06/08/2005	anglais	espagnol	latin	2E
075421AK	ROTH	Ursule	03/01/2005	anglais	allemand		2A
121001CK	SERHANI	Sabrina	01/09/2005	italien	anglais		2D
538965DJ	TUDJANE	Yourk	31/01/2005	espagnol	anglais		2D
389873GC	VIALET	Priscille	28/02/2005	espagnol	anglais		2C
980306CA	WADE	Marcelin	03/05/2005	allemand	anglais		2E
807158DH	WENGER	Alexandre	20/08/2005	allemand	anglais		2A
666702FA	YAMAN	Elamine	23/04/2005	anglais	arabe		2D

Extrait du fichier `seconde_lyc.csv`

Pour les besoins de l'organisation du lycée, le chef d'établissement exploite la base de données par des requêtes en langage `SQL`. Il a pour cela créé une table (ou relation) SQL dénommée `seconde` dans son système de gestion de bases de données dont la structure est la suivante :

`seconde`
`num_eleve` (clef primaire)
`langue1`
`langue2`
`option`
`classe`

L'attribut `num_eleve` est un entier, les autres sont des chaines de caractère (le type `CHAR`).

Question 1

1. Dans le modèle relationnel, quel est l'intérêt de l'attribut `num_eleve`.
2. Écrire une requête SQL d'insertion permettant d'enregistrer l'élève `ACHIR Mussa` dans la table `seconde`. Les informations relatives à cet élève sont données dans la ligne 1 du fichier `seconde_lyc.csv`.

3. Lors de l'insertion de l'élève `ALTMEYER Yohan` (ligne 2 du fichier `seconde_lyc.csv`), une erreur de saisie a été commise sur la première langue, qui devrait être `allemand`. Écrire une requête SQL de mise à jour corrigeant les données de cet élève.

Question 2 On suppose maintenant que la table `seconde` contient les informations issues de la figure 1 (ni plus, ni moins, même si la figure 1 n'est qu'un extrait du fichier `seconde_lyc.csv`).

1. Quel est le résultat de la requête

   ```
   SELECT num_eleve FROM seconde ;
   ```
   ?
2. On rappelle qu'en SQL, la fonction d'agrégation `COUNT()` permet de compter le nombre d'enregistrements dans une table.

   Quel est le résultat de la requête

   ```
   SELECT COUNT(num_eleve) FROM seconde ;
   ```
   ?
3. Écrire la requête permettant de connaître le nombre d'élèves qui font allemand en `langue1` ou `langue2`.

Question 3 Le chef d'établissement souhaite faire évoluer la structure de sa base de données. Pour ce faire, il créé une nouvelle table `eleve` dont la structure est la suivante :

eleve
num_eleve (clef primaire, clef étrangère de la table seconde)
nom
prenom
datenaissance

Là encore, l'attribut `num_eleve` est un entier, les autres sont des chaines de caractère (le type `CHAR`).

1. Expliquer ce qu'apporte l'information `clef étrangère` pour l'attribur `num_eleve` de cette table en termes d'intégrité et de cohérence.
2. On suppose la table `eleve` correctement créée et complétée. Le chef d'établissement aimerait lister les élèves (nom, prénom, date de naissance) de la classe 2A.

   Écrire la commande qui permet d'établir cette liste à l'aide d'une jointure entre `eleve` et `seconde`.

Question 4 Proposer la structure d'une table `coordonnees` dans laquelle on pourra indiquer, pour chaque élève, son adresse, son code postal, sa ville, son adresse mail. Préciser la clef primaire et/ou la clé étrangère en vue de la mise en relation avec les autres tables.

CORRECTION DU SUJET 0 :

Avertissement : Il y a une erreur dans l'énoncé puisqu'il est écrit que le numéro d'identification des élèves `num_eleve` est un entier, alors que `1333310FE` ne peut clairement pas l'être.
Dans le corrigé, on prendra pour `num_eleve` le numéro proposé sans les 2 dernières lettres soit `1333310`.

Question 1

1. L'attribut `num_eleve` est un numéro unique attribué à chaque élève car c'est une clef primaire. Elle permet d'identifier de manière unique chaque élève de seconde dans ce lycée.
2.

```
1 INSERT INTO seconde VALUES
2 (133310, 'anglais', 'espagnol', '', '2A');
```

3.

```sql
UPDATE seconde
SET langue1='allemand'
WHERE num_eleve = 156929;
```

Question 2

1. On obtient tous les numéros d'identification de tous les élèves de seconde de ce lycée.
2. On obtient le nombre d'élèves de seconde dans ce lycée.
3.

```sql
SELECT COUNT(*)
FROM seconde
WHERE langue1 = 'allemand' OR langue2 = 'allemand';
```

Question 3

1. En terme d'intégrité, `num_eleve` est une clé érangère, donc une clé primaire d'une autre table, ce qui permet d'avoir des entités uniques pour chaque élève de seconde du lycée.
   En terme de cohérence, `num_eleve` est une clé étrangère ce qui permet de relier les deux tables.
2.

```sql
SELECT nom, prenom, datenaissance
FROM eleve
JOIN seconde ON seconde.num_eleve = eleve.num_eleve
WHERE seconde.classe = '2A';
```

Question 4

coordonnees
num_eleve (clef primaire, clef étrangère de la table seconde)
adresse
code_postal
adresse_mail

5.4 Des exercices en lignes

exercice 5.16 Des requêtes simples sur le prix Nobel.
https://sqlzoo.net/wiki/SELECT_from_Nobel_Tutorial
Les corrections :
https://github.com/jisaw/sqlzoo-solutions/blob/master/select-from-nobel.sql ■

exercice 5.17 Des requêtes avec les fonctions d'agrégation `SUM` et `COUNT` sur des données de population mondiale (beaucoup de questions hors-programme).
https://sqlzoo.net/wiki/SUM_and_COUNT
Les corrections :
https://github.com/jisaw/sqlzoo-solutions/blob/master/sum-and-count.sql ■

exercice 5.18 Des requêtes avec des jointures sur l'Euro 2012.
https://sqlzoo.net/wiki/The_JOIN_operation
Les corrections :
https://github.com/jisaw/sqlzoo-solutions/blob/master/join.sql ■

5.5 Mini-Projet : Création et gestion d'une base de données en Python

Pour ce mini-projet nous allons utiliser un fichier contenant une liste de mots français provenant du site **eduscol** : https://eduscol.education.fr/186/liste-de-frequence-lexicale

On découvre que l'on peut utiliser le langage SQL en Python. Il y a juste à bien gérer la notion de curseur.

```
from pathlib import Path
import sqlite3

##########################################
# lecture des données du fichier réception sous la forme d'une chaîne de caractères
##########################################

with open("listes_mots.csv", "r") as fichier:
    abcd = fichier.read()

abcd = abcd.split('\n')
abcd.pop()
#print(abcd)


##########################################
# Création de la base
##########################################

fichier_base = "mots.sq3"
if not Path(fichier_base).is_file():
    base = sqlite3.connect(fichier_base)
    curseur = base.cursor() # le curseur garde en mémoire un certain nombre d'actions à exécuter
    curseur.execute("CREATE TABLE Mot(id INTEGER PRIMARY KEY, mot TEXT, nature TEXT, frequence INTEGER, longueur INTEGER)")
    for i in range(1,len(abcd)) :
        ligne = abcd[i].split(';')
        curseur.execute('INSERT INTO Mot VALUES (?,?,?,?,?)', (i-1, ligne[2], ligne[0], int(ligne[1]), len(ligne[2])))
    base.commit() # toutes les actions stockées dans le curseur sont exécutées
    curseur.close()
    base.close()

##########################################
# requête
##########################################

base = sqlite3.connect(fichier_base)
curseur = base.cursor()
curseur.execute("SELECT * FROM Mot WHERE longueur = 4")
for ligne in curseur :
    print(ligne)
base.commit()
curseur.close()
base.close()
```

On remarque que l'intérêt d'écrire un programme en Python réside dans le traitement de l'information. On a par exemple pu rajouter la longueur du mot.

On pourrait aussi imaginer la création d'une base de données sur des familles de nombres (les nombres premiers, ou les nombres de Fibonacci) afin d'interroger cette base en vue de faire de belles découvertes mathématiques.

5.6 Mini-Projet : Films de requins tueurs

La page wikipedia https://fr.wikipedia.org/wiki/Liste_de_films_de_requins_tueurs référence (presque ?) tous les films de requins tueurs. On souhaite créer une plateforme dans laquelle des fans de films de requins tueurs s'inscrivent et renseignent les films qu'ils ont déjà vus ainsi que les films qu'ils souhaitent voir prochainement. Les fans ayant des films en commun pourraient alors être mis en relation.

1. Proposer un système relationnel pertinent.

2. Créer la base de données en utilisant le langage SQL directement ou à partir d'un programme écrit en Python.