

Bourrez les gens de données incombustibles, gorgez-les de «faits», qu'ils se sentent gavés, mais absolument «brillants» côté information. Ils auront l'impression de penser, ils auront le sentiment du mouvement tout en faisant du sur-place.

Ray Bradbury - Fahrenheit 451

L'intuition est l'art, spécifique à l'esprit humain, d'élaborer une réponse correcte à partir de données qui, par elles-mêmes, sont incomplètes, voire trompeuses.

Isaac Azimov - L'aube de Fondation

4

Bases de données

Extrait du programme

THÈME : BASE DE DONNÉES

Contenus :

Modèle relationnel : relation, attribut, domaine, clef primaire, clef étrangère, schéma relationnel.

Capacités attendus :

Identifier les concepts définissant le modèle relationnel.

Commentaires :

Ces concepts permettent d'exprimer les contraintes d'intégrité (domaine, relation et référence).

Contenus :

Base de données relationnelle.

Capacités attendus :

Savoir distinguer la structure d'une base de données de son contenu.

Repérer des anomalies dans le schéma d'une base de données

Commentaires :

La structure est un ensemble de schémas relationnels qui respecte les contraintes du modèle relationnel.

Les anomalies peuvent être des redondances de données ou des anomalies d'insertion, de suppression, de mise à jour.

On privilégie la manipulation de données nombreuses et réalistes.

Contenus :

Langage SQL : requêtes d'interrogation et de mise à jour d'une base de données.

Capacités attendus :

Identifier les composants d'une requête.

Construire des requêtes d'interrogation à l'aide des clauses du langage SQL : SELECT, FROM, WHERE, JOIN.

Construire des requêtes d'insertion et de mise à jour à l'aide de : UPDATE, INSERT, DELETE.

Commentaires :

On peut utiliser DISTINCT, ORDER BY ou les fonctions d'agrégation sans utiliser les clauses GROUP BY et HAVING.

Contenus :

Système de gestion de bases de données relationnelles.

Capacités attendus :

Identifier les services rendus par un système de gestion de bases de données relationnelles :

persistance des données, gestion des accès concurrents, efficacité de traitement des requêtes, sécurisation des accès.

Commentaires :

Il s'agit de comprendre le rôle et les enjeux des différents services sans en détailler le fonctionnement.



b) 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 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 en bande dessinée Disney. Effectuez quelques requêtes.

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

Exercice 2 (Indian Food)

Nous allons travailler sur le fichier `indian_food.csv` provenant du site de bases de données ouvertes kaggle. Vous pouvez télécharger le document sur **e-lyco**.

1. Télécharger le fichier CSV et ouvrir-le avec votre 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.

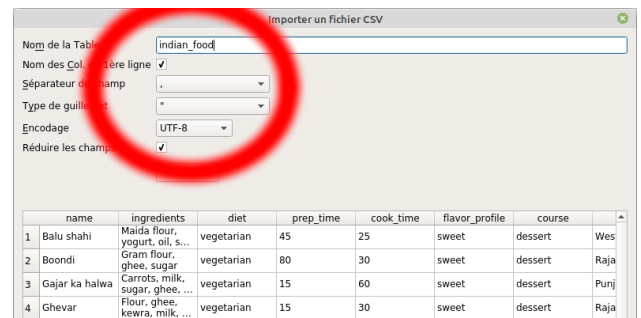
Exercice 3 (Indian Food)

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



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';
```

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

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 ingréd
    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";
```

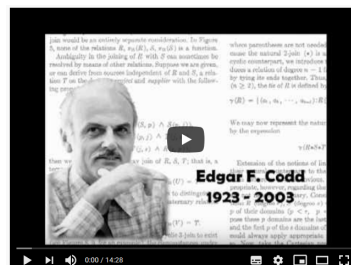
5. Pour chercher le nombre de **plat végétarien** ne contenant pas de **lait** :

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

c) Une vidéo introductive

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

<https://www.youtube.com/watch?v=pqoIBiM2AvE>



I. Modèle relationnel

a) Un problème introductif

Soit **SpeedyMcQueen** une agence de location de voitures de **Gothan City**. Toutes les informations nécessaires à la gestion du parc automobile et des clients se trouvent dans un tableur...

	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	Batman	85 rue de la Barre	oui	750,00 €	05/08/20	08/08/20	oui	batmobile modèle 1950	250€ / jour	peinture violette	essence		oui
3	Batman	85 rue de la Barre	oui	1 500,00 €	10/08/20	12/08/20	oui	batmobile modèle 2000	500€ / jour	peinture noire, 6 roues	hydrogène	roue AD arrachée	non
4	Robin	85 rue de la Barre	oui	750,00 €	10/08/20	12/08/20	oui	batmobile modèle 1990	250€ / jour	peinture noire	sans plomb	moteur explosé	non
5	Catwoman	49 rue des mystères	non	2 500,00 €	13/08/20	22/08/20	oui	Jeep	250€ / jour	4 x 4	diesel	siège griffé	non
6	Batman	85 rue de la Barre	oui	3 000,00 €	14/08/20	25/08/20	oui	batmobile modèle 1950	250€ / jour	peinture violette	essence		oui
7	The Joker	3 boulevard Foch	non		20/08/20		non	Limousine	1500€ / jour	hyper Deluxe	diesel		non
8	Le Pingouin	12 place du Ralliement	non	1 500,00 €	20/08/20	25/08/20	oui	Jeep	250€ / jour	4 x 4	diesel	inondée	non
9	Batman	85 rue de la Barre	oui	1 000,00 €	30/08/20	01/09/20	oui	batmobile modèle 2000	500€ / jour	peinture noire, 6 roues	hydrogène	siège griffée	non
10	Robin	85 rue de la Barre	non		30/08/20		non	batmobile modèle 2010	750€ / jour	look impressionnant	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 **Batman** 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 clientèle, 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, **Catwoman** a empruntée une **Jeep** du 13 au 22 août et le Pingouin aussi du 20 au 25 août. Manifestement, il y a plusieurs **Jeep** dans le parc automobile de cette agence, mais on ne les distingue pas. On retrouve le même problème avec la **batmobile modèle 2000**. La première a été rendue le 12 août avec une roue arrachée. Elle n'est pas disponible et pourtant **Batman** reprend ce modèle le 30 août.
- Une remarque supplémentaire, nous ne connaissons pas quel est l'ensemble du parc automobile disponible dans cette agence. Saviez vous, entre autres, qu'il y a deux **batmobile modèle 2020** qui utilisent uniquement l'eau comme carburant????

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 proposer 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 la 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
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

b) 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.

c) Contrainte de domaine

Pour chaque **attribut** d'une **relation** on 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) flottant(e) ;
- **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
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

Remarque

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**)*

d) 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 6 personnages différents qui ont porté le costume de Robin⁵. Donc cet attribut seul ne suffira pas. Souvent on rajoute un deuxième attribut comme l'**adresse**. La plupart du temps, le couple **nom**, **adresse** suffit à identifier une personne de manière unique. Malheureusement pour nous, tous les Robin vivent (ou ont vécu) chez Bruce Wayne⁶.

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**⁷.

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	
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	

Remarque

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

- *Client*(numéro client Integer, nom String, adresse String)
- *Modèle*(nom String, caractéristiques String, prix de la location Float, carburant String)
- *Location*(numéro location Integer, nom du client String, voiture String, 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)

5. 4 garçons et 2 filles. [https://fr.wikipedia.org/wiki/Robin_\(comics\)](https://fr.wikipedia.org/wiki/Robin_(comics))

6. au 85 rue de la Barre.

7. 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.

e) 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	
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	

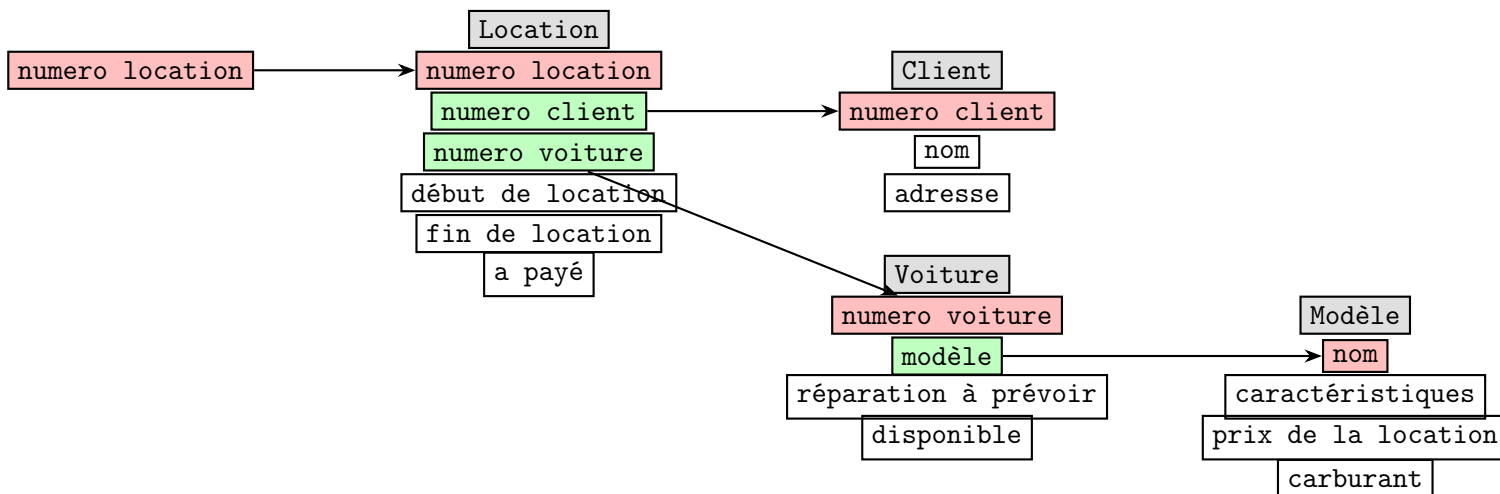
Voiture	Domaine	Clé
numéro voiture	Integer	Primaire
modèle	String	Étrangère(Modèle)
réparation à prévoir	String	
disponible	Boolean	

Remarque

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, numéro 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** et à l'**adresse** du client... etc ...



f) 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`, entre 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 ...

g) Exercices

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

Exercice 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

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

- `{}`
- `{('Titi', 'Toto', '0123456789')}`
- `{('Titi', 'Toto', '0123456789'), ('Doe', 'John', '0123456789')}`
- `{('Titi', 'Toto', '0123456789'), ('Titi', 'Toto', '9876543210')}`
- `{('Titi', 'Toto', '0123456789'), ('Doe', 'John')}`
- `{('Titi', 'Toto', 0123456789)}`

Exercice 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 être inscrites dans les schémas des relations.

Exercice 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. <code>Eleve = {}</code>
<code>Matiere = {}</code>
<code>Note = {}</code> | 4. <code>Eleve = {('Titi', 'Toto', 'AB56789')}</code>
<code>Matiere = {('NSI',0)}</code>
<code>Note = {('AB56789',0,17), ('AB56789',0,18)}</code> |
| 2. <code>Eleve = {('Titi', 'Toto', 'AB56789')}</code>
<code>Matiere = {('NSI',0), ('Sport',1)}</code>
<code>Note = {('AB56789',1,17)}</code> | 5. <code>Eleve = {('Titi', 'Toto', 'AB56789')}</code>
<code>Matiere = {('NSI',0), ('Sport',1)}</code>
<code>Note = {('AB56789',0,17), ('AB56789',1,17)}</code> |
| 3. <code>Eleve = {('Titi', 'Toto', 'AB56789')}</code>
<code>Matiere = {('NSI',0)}</code>
<code>Note = {('AB56789',1,17)}</code> | |

Exercice 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 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 10

On considère deux relations $R(\underline{a \text{ Integer}}, b \text{ Integer}, c \text{ Integer})$ et $S(\underline{a \text{ Integer}}, e \text{ Integer})$ 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.

II. 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 le langage **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/>



The screenshot shows the SQLiteBrowser website's 'Downloads' section. At the top, there is a navigation bar with links: About, Download, Blog, Docs, GitHub, Gitter, Stats, Patreon, and DBHub.io. Below the navigation bar, the 'Downloads' heading is followed by a note: '(Please consider sponsoring us on Patreon 🍷)'. Under the 'Windows' section, it states 'Our latest release (3.12.1) for Windows:' and lists four download options: 'DB Browser for SQLite - Standard installer for 32-bit Windows', 'DB Browser for SQLite - .zip (no installer) for 32-bit Windows', 'DB Browser for SQLite - Standard installer for 64-bit Windows', and 'DB Browser for SQLite - .zip (no installer) for 64-bit Windows'. Below this, the 'Windows PortableApp' section mentions that a PortableApp is available but is the previous (3.12.0) release version, and it should be updated to 3.12.1. It lists one option: 'DB Browser for SQLite - PortableApp'. A 'Note' at the bottom states: 'If for any reason the standard Windows release does not work (e.g. gives an error), try a nightly build (below)'.

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

a) 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 en 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 –

b) Les mises à jour

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

Création d'une table

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

```
1 CREATE TABLE Modele
2 (nom TEXT PRIMARY KEY ,
3 caracteristiques TEXT,
4 prix_location REAL,
5 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 **INT** 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'exactly 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 parenthèse, le nom de l'attribut.

```
1 CREATE TABLE Voiture
2 (numero_voiture INT PRIMARY KEY,
3 modele TEXT REFERENCES modele(nom),
4 prevoir_reparations TEXT,
5 disponible BOOLEAN);
```

Remarque

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 11

Créer les autres tables *Client* et *Location*

Remarque

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

```
1 CREATE TABLE Cleint
2 (numero_client INT PRIMARY KEY AUTO_INCREMENT,
3 ...
4 );
```

Insertion dans une table

Pour insérer de nouvelles entités dans une table :

```
1 INSERT INTO Modele VALUES
2 ('batmobile modèle 1950', 'peinture violette', 250, 'essence'),
3 ('batmobile modèle 1990', 'peinture noire', 250, 'sans plomb');
```

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

```
1 INSERT INTO Modele VALUES
2 ('batmobile modèle 1950', 'peinture noire, 6 roues', '500', 'hydrogène');
```

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
```

```
('batmobile modèle 1950', 'peinture noire', 250, 'essence');
```

Exercice 12

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

Suppression de tables

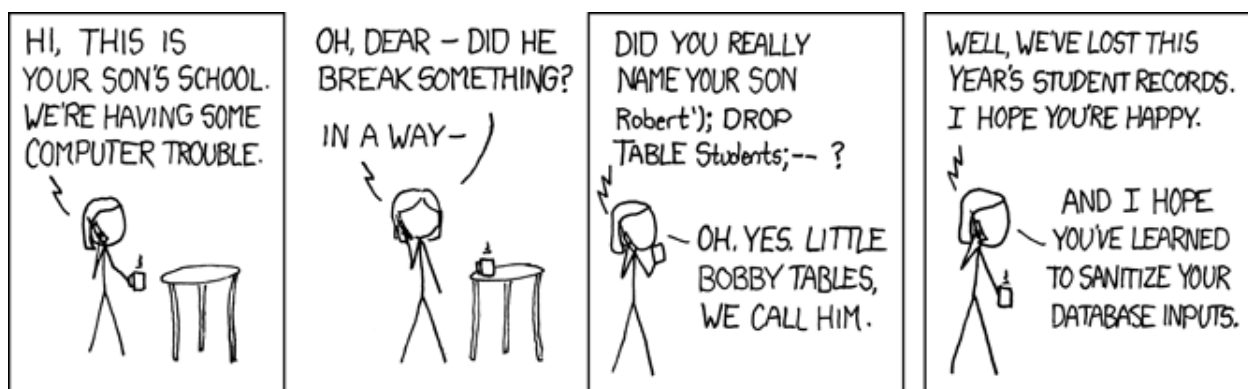
On peut supprimer des tables :

```
1 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 :

```
1 DROP TABLE Client;
```

devrait, renvoyer un message d'erreur...



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

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 :

```
1 CREATE TABLE Ami
2 (nom TEXT,
3 prenom TEXT,
4 adresse TEXT,
5 nu_tel TEXT,
6 PRIMARY KEY (nom, prenom));
```

- A nouveau, pour la **contrainte de référence**, il suffit de d'écrire **REFERENCE** 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**.

```
1 CREATE TABLE Modele
2 (nom TEXT PRIMARY KEY ,
3 caracteristiques TEXT NOT NULL,
4 prix_location REAL NOT NULL,
5 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...

```
1 CREATE TABLE Modele
2 (nom TEXT PRIMARY KEY ,
3 caracteristiques TEXT NOT NULL,
4 prix_location REAL NOT NULL,
5 carburant TEXT NOT NULL,
6 CHECK (prix_location>0));
```

Exercice 13

Réécrire les différentes tables en rajoutant des contraintes utilisateurs.

Remarque

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

c) Les requêtes

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

```
1 SELECT attribut1, attribut2, ...
2 FROM nom de la table
3 WHERE expression_booléenne;
```

Exemple

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

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

Mais si vous ne voulez pas rouler avec une vieille épave et si en même temps vous avez des moyens limités, vous pouvez demander les modèles dont le prix de location se situe entre 500 et 1000 euros par jour :

```
1 SELECT nom, prix_location
2 FROM Modele
3 WHERE prix_location >= 500 AND
4       prix_location <= 1000;
```

On obtient le résultat suivant :

nom	prix_location
Batmobile modèle 2000	500.0
Batmobile modèle 2010	750.0
Batmobile modèle 2020	1000.0

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.

```
1 SELECT *
2 FROM Modele
3 WHERE carburant = 'diesel';
```

On obtient le résultat :

nom	caracteristiques	prix_location	carburant
Jeep	4 x 4	250.0	diesel
Limousine	hyper Deluxe	1500.0	diesel

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`⁸.

Par exemple, si on veut tous les différents modèles de Batmobile, on procède ainsi :

```
1 SELECT nom
2 FROM Modele
3 WHERE nom LIKE '%batmobile%';
```

On obtient le résultat :

Batmobile modèle 1950
Batmobile modèle 1990
Batmobile modèle 2000
Batmobile modèle 2010
Batmobile modèle 2020

Tri et suppression des doublons (limite hors-programme)

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).

```
1 SELECT nom, prix_location
2 FROM Modele
3 ORDER BY prix_location ASC;
```

On obtient le résultat :

nom	prix_location
Batmobile modèle 1950	250.0
Batmobile modèle 1990	250.0
Jeep	250.0
Batmobile modèle 2000	500.0
Batmobile modèle 2010	750.0
Batmobile modèle 2020	1000.0
Limousine	1500.0

8. Cet opérateur est très certainement hors-programme. On doit l’utiliser pour savoir si une chaîne de caractères est incluse 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 met que l'attribut `prix_location` on va obtenir 3 fois la valeur 250.0. Pour éviter les répétitions, on précise `DISTINCT` devant l'attribut.

```
1 SELECT DISTINCT prix_location
2 FROM Modele
3 ORDER BY prix_location ASC;
```

On obtient le résultat :

<u>prix_location</u>
250.0
500.0
750.0
1000.0
1500.0

Fonction d'agrégation (limite hors-programme)

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 :

```
1 SELECT COUNT(*)
2 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 :

```
1 SELECT SUM(prix_location)
2 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 :

```
1 SELECT AVG(prix_location)
2 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 :

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

Exercice 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.
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 de peinture noire.

d) Jointure

Pour le moment nous n'avons effectué 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 :

```
1 SELECT modele
2 FROM Voiture
3 JOIN Location ON Voiture.n_voiture = Location.n_voiture
4 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 15

Effectuer les requêtes suivantes :

1. Le nom et l'adresse du client n'a pas rendu la location n°9.
2. Le nom et l'adresse de tous les clients qui n'ont pas rendu leur voiture de location.
3. Le nom et l'adresse de tous les clients qui n'ont pas encore payé.
4. Pour aider le client, le prix des locations par ordre croissant et le modèle des voitures disponibles.
5. Le prix des locations 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 2020.

e) Modification des données

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

Suppression de lignes

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

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

Mise à jour

Robin a rendu la voiture n°10 le 3 septembre 2020. On va modifier la table **Location** avec la clause **UPDATE** suivie de **SET**.

```
1 UPDATE Location
2 SET fin_loc = '2020-09-03'
3 WHERE n_location = 9;
```

III. Le sujet 0

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

Exercice 4

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	ALTMAYER	Yohan	05/05/2005	allemand	anglais	théâtre	2D
500633KH	BELEY	Tribaut	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	ELAYAR	Amir	11/09/2005	anglais	arabe	cinéma	2D
108308KG	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 chaînes 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 **ALTMAYER 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 crée 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 chaînes de caractère (le type **CHAR**).

1. Expliquer ce qu'apporte l'information **clef étrangère** pour l'attribut **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.

```

1 UPDATE seconde
2 SET langue1='allemand'
3 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.

```

1 SELECT COUNT(*)
2 FROM seconde
3 WHERE langue1 = 'allemand'
4      OR langue2 = 'allemand';

```

Question 3

1. En terme d'intégrité, `num_eleve` est une clé étrangère, 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.

```

1 SELECT nom, prenom, datenaissance
2 FROM eleve
3 JOIN seconde ON seconde.num_eleve = eleve.num_eleve
4 WHERE seconde.classe = '2A';

```

Question 4

coordonnees
num_eleve (clef primaire, clef étrangère de la table <code>seconde</code>)
adresse
code_postal
adresse_mail

IV. Des exercices en lignes

Exercice 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 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 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>