Math ECG 2. 2023-2024

Mathématiques Appliquées - F. Gaunard
http://frederic.gaunard.com
ENC Bessières, Paris 17e.

Chapitre 17. Bases de données. Requêtes SQL.

Avant-propos

L'administration, les banques, les assurances, les secteurs de la finance utilisent des **bases de données**, systèmes d'informations qui stockent dans des fichiers les données nombreuses qui leur sont nécessaires.

Une base de données relationnelle permet d'organiser, de stocker, de mettre à jour et d'interroger des données structurées volumineuses utilisées simultanément par différents programmes ou différents utilisateurs. Un logiciel, le **système de gestion de bases de données** (SGBD), est utilisé pour la gestion (lecture, écriture, cohérence, actualisation...) des fichiers dans lesquels sont stockées les données.

L'accès aux données d'une base de données relationnelle s'effectue en utilisant un langage informatique qui permet de sélectionner des données spécifiées par des formules de logique, appelées **requêtes** d'interrogation et de mise à jour.
L'objectif est de présenter une description applicative des bases de données en langage de requêtes **SQL** (*Structured Query Language*).

Ce chapitre se décompose en deux parties; une première partie (plus théorique, sous forme d'un cours), présente le vocabulaire des bases de données ainsi que les opérations et les manipulations que l'on pourra faire sur une BDD. Une deuxième partie, sous forme d'une activité (avec un exemple très concret, une enquête inspirée par les *Panama papers*) permet de se familiariser avec la syntaxe du langage SQL.

On remercie **Louis Merlin** (Alexandre Dumas, St Cloud) pour le partage d'une grande partie de ce document.

1 Bases de données. Vocabulaire.

1.1 Tables de relations

L'objet de base que nous manipulerons tout au long de ce cours est appelé *relation*.

Définition

Une **relation** est un tableau de données (penser à un fichier Excel). Chaque ligne représente un objet et tous les objets sont de même nature.
Une **base de donnée** (parfois abrégée en BDD) est un ensemble de relations.

☞ Même si le tableur Excel est la bonne image à garder en tête, nous verrons plus loin que, dans la pratique, les données sont stockées dans un fichier texte .csv (pour *comma separated values*), plus léger et moins riche graphiquement que le tableur.

Exemple

Construisons la table de relation des étudiants de cette prépa, en indiquant comme caractéristiques les dates de naissance, la ville de provenance et le choix des langues LVA et LVB.

Les étudiants de Bessières				
identifiant	**date de naissance**	**ville de provenance**	**LVA**	**LVB**
1	21/06/2003	Paris 18	Anglais	Espagnol
2	17/11/2004	Asnières	Espagnol	Anglais
3	24/03/2003	Paris 17	Anglais	Italien
4	11/10/2001	Paris 16	Arabe	Anglais
5	03/01/2002	Saint-Ouen	Anglais	Espagnol
6	03/01/2002	Paris 18	Anglais	Allemand
...	...	...	...	...

☞ Dans une relation, l'ordre des lignes n'a pas d'importance.

☞ Il est très fréquent de mélanger les termes définis par le modèle relationnel et les termes propres au vocabulaire des bases de données relationnelles. Ainsi

Définition

- Une *relation* est aussi appelée une **table**.
- Les termes *ligne*, *n-uplet*, *enregistrement* ou *vecteur* sont tous synonymes.
- De même les colonnes sont souvent appelées des **attributs**.
- Le **schéma** est l'ensemble des attributs d'une relation.
- Le *domaine* désigne l'ensemble des valeurs que peuvent prendre un attribut.

☞ On utilisera alors le terme *relation* ou *table* pour désigner la même chose au sein de cette activité.

1.2 Les clés

Reprenons l'exemple précédent des étudiants de notre prépa. Il est tout à fait possible que deux étudiants aient exactement la même date de naissance et étudient les mêmes langues. Dans une table, il est très important d'identifier de **manière unique** chaque ligne. C'est pour résoudre ce problème d'identification qu'interviennent les clés.

Définition

Une **clé** est un groupe minimum d'attributs caractérisant chaque n-uplet de manière unique.

Exemple

Dans notre exemple précédent, l'attribut **identifiant** est une clé ("*l'étudiant qui a pour identifiant 2*" permet de désigner un étudiant sans aucune ambiguïté).

En revanche, la **date de naissance** n'est pas une clé car il y a deux étudiants qui sont nés le 03/01/2002. Le groupe d'attributs {**date de naissance**, **LVB**} est une clé.

Exercice 1. On considère la relation **conserves** décrite par la table ci-dessous.

code barre	produit	marque	prix unitaire	quantité	peremption
3083680020763	Maïs	Budget+	1,19	12	2026
3017800207901	Maïs	GourmetBio	2,69	9	2025
3038359008504	Tomates	PizzaLovers	2,99	6	2025
303087439871	Haricots Verts	Budget+	1,89	10	2025
3093439818758	Pêches	GourmetBio	2,99	6	2030
3023249825326	Pois chiches	Budget+	0,99	4	2026

(1) Le groupe d'attributs {**code barre**} est-il une clé pour notre relation ?
(2) Le groupe d'attributs {**code barre**, **produit**} est-il une clé pour notre relation ?
(3) Le groupe d'attributs {**produit**, **marque**, **prix**} est-il une clé pour notre relation ?
(4) Le groupe d'attributs {**produit**} est-il une clé pour notre relation ?
(5) Le groupe d'attributs {**marque**, **prix**} est-il une clé pour notre relation ?

☞ Ainsi, comme c'est le cas dans notre exemple précédent, plusieurs attributs peuvent être des clés. Or, dans une base de données, nous voulons nous mettre d'accord une fois pour toute sur l'une d'entre elles.

Définition

Dans une relation, les groupes d'attributs qui constituent des clés sont appelés des *clés candidates*. Dans la pratique, il sera indispensable d'en choisir une et nous l'appellerons la **clé primaire**.
En anglais, clé primaire se dit Primary Key et s'abrège en **PK**.

☞ Dans le logiciel de gestion de base de données que nous utiliserons, nous verrons que lors de la création d'une table de données dans un fichier .csv, il est **indispensable** de désigner au préalable une clé primaire.

À retenir!

Le choix d'une clé primaire pourrait très bien être arbitraire (parmi les clés candidates). Cependant, on peut prendre en compte certains critères. En effet, une clé primaire est généralement choisie de façon à ce qu'elle soit simple, c'est-à-dire qu'elle ne contienne le moins d'attributs possibles.

De plus, on préfère généralement les attributs "basiques" (par exemple des entiers ou des chaînes de caractères courtes).

☞ Parfois aucune des clés candidates ne contient un nombre raisonnable d'attributs et ne semble "simple". On peut alors fabriquer une clé simple :

Définition

Une **clé artificielle** est un attribut que l'on ajoute à la relation. Cet attribut n'a pas de réelle signification dans le domaine que l'on modélise et sa seule fonction est d'identifier de manière unique les n-uplets de la relation.

Exemple

Reprenons encore une fois la liste des étudiants en prépa ECG de Bessières. Le premier attribut **identifiant** est une clé artificielle.

☞ Pour des questions de performance (en termes de temps de calcul), les clés non artificielles ne sont souvent pas optimisées lorsqu'il est demandé au SGBDD de retrouver une ligne dans une table.

☞ Un bon programmeur prend en fait l'habitude d'utiliser comme clés primaires des clés artificielles.

Une base de données est en fait bien souvent constituée de plusieurs tables ; et ces tables ne sont pas indépendantes.

Dans notre exemple fil rouge de la base de données des étudiants de notre prépa, nous pouvons nous intéresser à la ville de provenance des étudiants et renseigner dans une relation séparée certaines caractéristiques des villes qui y décrivent l'ambiance de travail. Même si cette seconde table est d'un intérêt indépendant, ces deux relations sont liées car chaque étudiant provient d'une des villes décrites dans le seconde table.

Villes autour de l'ENC Bessières			
ville (PK)	**temps de trajet max**	**avantages**	**inconvénients**
Paris 17	5	proche	cher
Asnières	20	calme et proche	pas de bibliothèque
Saint-Ouen	10	proche	Bruyant
Paris 16	35	sport au Bois	super cher
Paris 18	20	possib. de res. univ.	pas facile pour le sport
...	...	...	...

Ainsi, si on veut connaître le temps de trajet quotidien d'un étudiant, on doit déjà connaître sa ville de provenance, puis regarder dans la table des villes, quel est le temps de trajet pour venir à Bessières.

☞ Retrouver une ligne dans une table, c'est justement le rôle des clés !

Définition

Un attribut d'une table qui est une clé pour une autre table de la base de données s'appelle une **clé étrangère**.

En anglais, clé étrangère se dit Foreign Key et s'abrège en **FK**.

Exemple

Dans notre liste d'étudiant, l'attribut **ville de provenance** n'est pas une clé (plusieurs étudiants peuvent venir de la même ville). Cependant, cet attribut fait quand même référence à la clé primaire d'une autre table de la base de données, celle qui décrit les conditions de travail dans les villes voisines de L'ENC. La ville de provenance est donc une clé étrangère dans la relation des étudiants.

À retenir!

C'est en général une mauvaise idée de regrouper tous les attributs dans une même table. En effet,

- Si le temps de trajet pour aller de l'ENC à une ville voisine change (par exemple avec les nouvelles lignes du Grand Paris), il faudrait modifier l'information dans toutes les lignes correspondantes de la table (pour tous les étudiants qui habitent dans cette ville). On risque d'en oublier et de perdre la cohérence de la base de données. On cherche à éviter la **redondance des données**.

- Si on souhaite ajouter les caractéristiques d'une nouvelle ville proche de l'ENC, mais qu'aucun étudiant n'habite dans cette ville, on ne peut pas.

Définition

Le processus qui consiste à créer de nouvelles tables pour éviter la redondance s'appelle la **normalisation**.

☞ Dans la suite de ce cours, nous nous intéresserons uniquement à la *manipulation* des données, et nous n'entrerons pas dans les détails de la *création* de données et de leur *stockage*. Les stratégies de normalisation sont donc hors programme en ECG.

1.3 Les tables d'association

Considérons la situation suivante. On dispose de la relation des **écoles** ci-dessous

Écoles de commerce		
Identifiant (PK)	**Nom de l'école**	**Rang**
1	HEC	1
2	ESSEC	2
3	ESCP	3
4	EDHEC	4
5	EMLyon	4 ex aequo
...	...	...

(Observons que nous avons pris pour **PK** une clé articifielle, le rang n'étant pas une bonne PK car il peut y avoir des ax-aequo.)

C'est maintenant la fin de l'année dans notre prépa et nous voulons ajouter à notre base de données les informations sur les différentes admissibilités de nos étudiants dans les différentes écoles. Nous pouvons donc ajouter une colonne dans notre table d'étudiants avec l'attribut **admissible à** qui est une FK vers la table **écoles**.

Et si un étudiant est admissible dans 2 écoles, comment fait on ?

On pourrait mettre deux colonnes avec les attributs **admissible à 1** et **admissible à 2** qui seraient toutes les deux des clés étrangères vers la table **écoles**. Mais ce raisonnement n'est pas très bon : on ne peut pas savoir à l'avance dans combien d'écoles un étudiant va être admissible.

La stratégie inverse ne fonctionne pas non plus. On pourrait envisager de mettre dans la table **écoles** un attribut **étudiant admissible** qui serait une clé étrangère vers la table **étudiants**. Mais cela voudrait dire qu'une école ne peut sélectionner qu'un seul étudiant admissible !

☞ Ce problème motive la définition ci-dessous.

Définition

Une **table d'association** est une relation qui contient comme attributs au moins deux clés étrangères vers d'autres relations de la base de données.

Exemple

Construisons donc la table d'association des étudiants et des écoles dans lesquelles ils sont admissibles.

Nous pouvons renseigner efficacement les différentes admissibilités des étudiants de la prépa, sans nous limiter, ni en nombre d'écoles par étudiant, ni en nombre d'étudiants admissibles par école.

Admissibilités	
Étudiant	**École**
1	1
1	2
1	4
2	4
2	5
...	...

☞ Chacun des attributs de cette table est FK mais la PK est constituée des deux attributs {**Étudiant, École**}.

Remarque

☞ Rien ne nous empêche d'ajouter des attributs dans notre table d'association (par exemple la date de passage pour les oraux dans la situation précédente) : tous les attributs de la table d'association ne sont donc pas nécessairement des clés étrangères pour une relation de la base de données.

Cela pose d'ailleurs la question du choix de la clé primaire dans une table d'association... mais c'est une autre histoire.

2 Algèbre relationnelle

Dans la partie précédente, nous avons vu comment représenter des données. Nous passons maintenant à la partie manipulation. Comme convenu au début de ce cours, nous décrivons tout d'abord de manière abstraite les différentes opérations que l'on peut appliquer à une base de données, puis nous verrons comment cela se concrétise dans la section suivante.

2.1 Projection et Restriction

Définition

La **projection** consiste à sélectionner les attributs que l'on souhaite et éliminer les autres.

Exemple

Si l'on s'intéresse uniquement aux dates de naissance des étudiants de la prépa, on peut faire une projection pour ne conserver que l'attribut **date de naissance** et on obtient alors la table suivante.

identifiant	Date de naissance
1	21/06/2003
2	17/11/2004
3	24/04/2003
4	11/10/2001
5	03/01/2002
6	03/01/2002

Nous voudrions maintenant faire la même opération mais sur les lignes, c'est-à-dire ne conserver que certaines des lignes de notre relation. C'est légèrement plus compliqué car les colonnes de notre table portent un nom mais pas les lignes. Les lignes d'une base de données sont amenées à varier car les informations contenues dans la table varient au cours du temps.

Comme les lignes à garder ou à enlever sont *dynamiques*, il nous faut donc une condition qui nous permette de restreindre les lignes.

Définition

Une **restriction** dans une table de données est une condition binaire (de type vrai ou faux) qui porte sur un ou plusieurs des attributs de la relation.

Exemple

La restriction de la table Étudiants sous la condition LVA='Anglais' produit la table suivante:

Les étudiants de Bessières				
identifiant	**date de naissance**	**ville de provenance**	**LVA**	**LVB**
1	21/06/2003	Paris 18	Anglais	Espagnol
3	24/03/2003	Paris 17	Anglais	Italien
5	03/01/2002	Saint-Ouen	Anglais	Espagnol
6	03/01/2002	Paris 18	Anglais	Allemand
...	...	...	...	...

Exercice 2. Quelle table obtient-on à partir de la table **conserves** de l'Exercice 1 après restriction sous la condition peremption <2026 ?

2.2 Union, Différence, Intersection

Définition

L'**union** de deux relations R_1 et R_2 de même schéma est une nouvelle relation, également de même schéma, qui contient l'ensemble des lignes de R_1 et R_2

☞ Attention, dans la pratique avec le langage SQL, l'union de deux tables qui contiennent une ligne en commun produira une table avec la même ligne répétée.
Mais ce n'est pas très grave, il existe une commande pour éliminer les doublons d'une table.

Définition

La **différence** de deux relations R_1 et R_2 de même schéma donne une relation R_3 de même schéma qui contient toutes les lignes de R_1 qui ne sont pas dans R_2.

À retenir!

Attention, si l'opération de réunion est commutative, ce n'est pas le cas de l'opération de différence : l'ordre compte!

Définition

L'**intersection** de deux relations R_1 et R_2 de même schéma donne une relation R_3 de même schéma qui contient toutes les lignes qui sont à la fois dans R_1 et dans R_2.

2.3 Produit cartésien

Au conseil de classe, chaque professeur doit donner une appréciation sur chaque étudiant de la prépa. Pour cela, nous disposons d'une relation qui contient les informations sur les professeurs de la classe (pour simplifier, nous n'entrerons pas dans le détail du choix des options).

identifiant prof	**Nom du professeur**
prof1	M. Barroux
prof2	M. Gaunard
prof3	M. Perl
prof4	Mme Soutet
prof5	M. Vallée

et (une projection et restriction du) tableau des étudiants

identifiant étudiant
1
2
3

Pour pouvoir indiquer toutes les appréciations dans une relation, nous devons former le *produit cartésien* des deux tables des enseignants et des étudiants.

Définition

Le **produit cartésien** de deux relations R_1 et R_2 est une table qui contient toutes les combinaisons possibles des lignes de R_1 et des lignes de R_2 et qui contient les colonnes de R_1 ainsi que les colonnes de R_2.

Exemple

Dans l'exemple entamé en début de sous-section, le produit cartésien des relations professeurs et étudiants renvoie la table suivante:

identifiant prof	Nom du professeur	identifiant étudiant
prof1	M. Barroux	1
prof1	M. Barroux	2
prof1	M. Barroux	3
prof2	M. Gaunard	1
prof2	M. Gaunard	2
prof2	M. Gaunard	3
prof3	M. Perl	1
prof3	M. Perl	2
prof3	M. Perl	3
prof4	Mme Soutet	1
prof4	Mme Soutet	2
prof4	Mme Soutet	3
prof5	M. Vallée	1
prof5	M. Vallée	2
prof5	M. Vallée	3

à laquelle il est ensuite facile d'ajouter un attribut **appréciation**.

2.4 Jointure

La *jointure* est le concept fondamental de l'algèbre relationnelle. Jusqu'à présent, nous avons vu que les clés étrangères (FK) servent à lier des relations. Mais nous ne savons pas encore comment exploiter ces liaisons. C'est justement le but de l'opération de jointure.

Si nous voulons connaître le temps de trajet pour venir à l'ENC d'un étudiant de la prépa, nous devons regarder d'abord dans la table des étudiants sa ville de provenance, puis aller chercher le temps de trajet dans la table des villes voisines. La jointure sert à coller les deux tables pour faire apparaître l'information sur une seule table. Plus précisément,

Définition

On considère deux relations R_1 et R_2 et on suppose que dans la relation R_1, le groupe d'attributs G est une clé externe (FK) pour la relation R_2. La **jointure** de R_1 et R_2 selon G est la table qui contient le même nombre de lignes que R_1 et les attributs de R_1 et de R_2. Chaque ligne est construite en commençant par la ligne de R_1, puis en ajoutant à sa suite la ligne de R_2 caractérisée par la valeur de sa clé dans G.

Le groupe d'attributs G s'appelle **la condition de jointure**.

Exemple

Reprenons notre relation préférée **étudiants**

étudiants				
identifiant (PK)	**date de naissance**	**ville de provenance (FK)**	**LVA**	**LVB**
1	21/06/2003	Paris 18	Anglais	Espagnol
2	17/11/2004	Asnières	Espagnol	Anglais
3	24/03/2003	Paris 17	Anglais	Italien
4	11/10/2001	Paris 16	Arabe	Anglais
5	03/01/2002	Saint-Ouen	Anglais	Espagnol
6	03/01/2002	Paris 18	Anglais	Allemand

et une relation **villes** (obtenue après une projection d'une relation ci-avant):

villes	
ville (PK)	**temps de trajet max**
Paris 17	5
Asnières	20
Saint-Ouen	10
Paris 16	35
Paris 18	20

La jointure de ces deux relations selon la condition ville de provenance = ville donne la table suivante

étudiants. identifiant	**étudiants. date de naissance**	**étudiants. ville de provenance**	**étudiants. LVA**	**étudiants. LVB**	**villes. ville**	**villes. temps de trajet max**
1	21/06/2003	Paris 18	Anglais	Espagnol	Paris 18	20
2	17/11/2004	Asnières	Espagnol	Anglais	Asnières	20
3	24/03/2003	Paris 17	Anglais	Italien	Paris 17	5
4	11/10/2001	Paris 16	Arabe	Anglais	Paris 16	35
5	03/01/2002	Saint-Ouen	Anglais	Espagnol	Saint-Ouen	10
6	03/01/2002	Paris 18	Anglais	Allemand	Paris 18	20

Exercice 3. Montrer que la jointure est l'enchaînement d'un produit cartésien et d'une restriction.

2.5 Agrégation

L'*agrégation* (qui n'est en fait pas une opération de l'algèbre relationnelle) est utilisée lorsqu'on veut faire un calcul qui porte sur plusieurs ligne d'une table. Elle sert par exemple à répondre à la question *Quel est le temps de trajet moyen des étudiants pour chaque LVA (ou chaque LVB)* ?

L'agrégation est donc une opération en deux étapes : une étape de *partitionnement* et une étape de calcul où on applique une fonction à chacun des blocs de la partition.

Définition

On considère une table et un groupe d'attributs.
Un **agrégat** est une partition des lignes d'une table selon les différentes valeurs que peuvent prendre les attributs
.
Dans cette situation, on dit que les attributs sont des **attributs de partitionnement**.

Une fois les agrégats formés, on leur applique une fonction d'agrégation.

Définition

Une **fonction d'agrégation** est une fonction définie sur les éléments de la partition (elle rend une unique valeur pour chaque bloc de la partition).

☞ Les exemples classiques de fonctions d'agrégation que nous manipulerons dans la pratique sont les fonctions de *décompte* (compter le nombre d'éléments d'un bloc), des fonctions de *moyenne*, de *minimum*, de *maximum*.

3 Commandes SQL (une enquête sur les *Panama Papers*)

Cette deuxième partie du chapitre est rédigée sous forme d'une activité (dont vous êtes le personnage principal). Nous allons y découvrir le langage SQL qui est construit pour dialoguer avec des bases de données relationnelles.

Nous n'entrerons pas (trop) dans les stratégies de création de données. Nous supposerons que la base de données est déjà construite et déjà remplie. Ce qui nous intéresse ici c'est d'interroger ces données.

☞ Avant de commencer, il faudra télécharger un interpréteur de commandes SQL (on a choisi SQLite) et la base de données que nous allons utiliser. On renvoie à l'annexe pour le détail.

3.1 Introduction : le contexte

En avril 2016, le journal allemand *Süddeutsche Zeitung* ainsi que le Consortium International des Journalistes d'Investigation (ICIJ) publient des documents confidentiels provenant d'un cabinet d'avocats panaméen.

Cette publication fait grand bruit à travers le monde, car les documents sur lesquels ont enquêté les journalistes du consortium international révèlent des informations sur plus de 214 000 sociétés offshores ainsi que le nom des actionnaires de celles-ci. C'est l'affaire des Panama Papers.

Si l'affaire a fait tant de bruit, c'est parce qu'elle dévoile un système complexe, massif et secret permettant à des entreprises ou à des particuliers de cacher de grosses sommes d'argent sur des comptes bancaires. Ces comptes sont généralement situés dans des pays où la législation est avantageuse, que ce soit en termes de secret bancaire, de taxation, ou de contrôle de la provenance de l'argent. Ce phénomène est appelé l'évasion fiscale.

Si ces pratiques sont souvent légales, l'opinion publique les voit en général d'un mauvais ?il. En effet, des sommes d'argents générées au sein d'un État donné (les bénéfices d'une entreprise par exemple) sont taxées par ce même État. Le fruit de cette taxation est redistribué (entre autres) aux services publics dont bénéficie la population du pays en question.

Cependant, l'évasion fiscale consiste à transférer les bénéfices du pays d'origine vers des pays à législation avantageuse (appelés les paradis fiscaux). Ainsi, les sommes d'argent générées dans un pays donné échappent en partie à l'impôt, et ne bénéficient donc plus aux populations locales. Dans certains pays, le montant estimé de l'évasion fiscale est égal ou supérieur au budget annuel de l'État, lorsqu'en parallèle leurs hôpitaux peinent à assurer les soins nécessaires (source : ICIJ).

Dans les *Panama Papers* se trouvaient par exemple les noms de plusieurs responsables politiques à travers le monde. Certains d'entre eux ont dû démissionner suite à la pression de l'opinion publique. Mais les *Panama Papers* ont aussi mis en lumière des moyens de financement de réseaux criminels ou terroristes.

En France, l'affaire a été révélée par 2 groupes de journalistes : Premières Lignes Production (qui a réalisé ce documentaire), et Le Monde.

3.2 La base de données

Les *Panama Papers* sont composés de près de 11,5 millions de documents (emails, courriers, contrats, etc.), pour un volume d'environ 2 Go. De ces documents écrits, l'ICIJ a tenté d'extraire les informations essentielles grâce à des algorithmes. Le résultat de cette extraction a été placé dans une base de données rendue publique.

Cette base n'est pas exacte, elle contient par exemple beaucoup de doublons et de champs erronés.

Grossièrement, la BDD des *Panama Papers* contient des sociétés *offshores*. Celles-ci sont créées pour des bénéficiaires par des fournisseurs de services *offshores*. Des intermédiaires se chargent généralement de faire le lien entre les bénéficiaires et les fournisseurs de services *offshores*.

Il y a 4 tables principales dans la base de données.

(1) La table **entity**. C'est elle qui contient les sociétés *offshores*.
(2) La table **intermediary**, qui contient les intermédiaires.
(3) La table **address** qui contient les adresses de certaines sociétés intermédiaires.
(4) La table **officer**, contenant entre autres les bénéficiaires des sociétés.

Ces tables contiennent les données publiées par l'ICIJ, auxquelles ont été ajoutées quelques données **fictives spécialement pour ce T.P**, notamment la société *Big Data Crunchers Limited*. Elle a été créée de toutes pièces pour servir de fil rouge.

☞ Une société peut être domiciliée dans un pays, mais être enregistrée dans un autre. Dans ce cas, cette société répondra à la juridiction dans laquelle elle est enregistrée, même si son adresse officielle n'est pas dans cette juridiction.

Souvent, les termes *juridiction* et *pays* sont confondus. En général, les lois sont les mêmes à l'intérieur d'un même pays. Mais parfois, un pays possède plusieurs juridictions : c'est souvent le cas des états fédéraux, dans lesquels chaque état possède des lois différentes. Par exemple, l'état du Delaware aux USA est souvent considéré comme un paradis fiscal, car les lois y sont plus avantageuses pour les sociétés que dans les autres états des USA.

Un T.P dont vous êtes le héros

On propose de se mettre dans la peau d'un enquêteur qui enquête sur le financement d'un réseau criminel.

Plus précisément, on a au cours d'une enquête intercepté une facture émise par une mystérieuse société qui s'appelle *Big Data Crunchers Limited*. Sur cette facture, l'adresse de cette société n'est pas indiquée. On ne sait pas qui se cache derrière cette société, mais on pense qu'elle peut être une société écran. Une société écran ne se crée pas si facilement que cela. En général, il faut demander de l'aide à des services spécialisés. On les appellera ici des intermédiaires.

On va donc enquêter sur cette mystérieuse société, mais aussi sur les intermédiaires qui ont aidé à la créer, car on pense qu'il sera peut-être possible de les accuser de complicité.

3.3 Un peu de vocabulaire

Définition non mathématique

En économie, une **société** est la forme juridique la plus répandue des entreprises ; c'est un terme souvent utilisé pour désigner une entreprise. (Source : Wikipedia)

Définition non mathématique

Une **société écran** est une société fictive, créée pour dissimuler les transactions financières d'une ou de plusieurs autres sociétés. (Source : Wikipedia)

Définition non mathématique

Une société *offshore* est une société "extraterritoriale" en français. En pratique, il s'agit d'une société créée dans un pays dans lequel le bénéficiaire économique final n'est pas résident et qui est dirigée hors du pays dans lequel elle est immatriculée.
Elles sont souvent utilisées dans des pays où la fiscalité est avantageuse. La société *offshore* est une forme de société écran, qui présente toutes les caractéristiques d'une société réelle (elle est immatriculée par exemple), mais dont l'apparence ne correspond pas à la réalité.

Définition non mathématique

Un **intermédiaire** est dans la plupart des cas une personne ou un cabinet d'avocats agissant pour des clients recherchant un fournisseur de services *offshores* ou demandant la création d'une société *offshore*. (Source : ICIJ)

Définition non mathématique

Un Fournisseur de services *offshore* (en anglais : *offshore service provider* ou *agent*). C'est une société qui fournit des services dans une juridiction *offshore*, sur demande d'un client. Ces services peuvent être la création, l'enregistrement ou la gestion de sociétés offshores. (Source : ICIJ)

Définition non mathématique

Un **bénéficiaire** (en anglais : *beneficial owner* ou *beneficiary*) est la personne réellement propriétaire de la société. Dans le monde *offshore*, l'identité du bénéficiaire est souvent gardée secrète. (Source : ICIJ)

Remarque

☞ L'ICIJ tient à préciser à toute personne souhaitant utiliser la base de données les points suivants :

(1) L'utilisation de sociétés offshores et de trusts n'est pas toujours illégale. Les personnes, sociétés ou autres entités citées dans la base de données n'ont donc pas forcément enfreint la loi ou agi de manière illégitime.
(2) Beaucoup de personnes ou entités ont des noms similaires. Avant de conclure que deux noms correspondent à la même personne ou entité, il est conseillé de vérifier leurs adresses respectives ou toute autre information pertinente.
(3) En cas d'erreur dans la base de données, prendre contact avec l'ICIJ.

3.4 La partie pratique : l'action commence

☞ On commence par lancer SQLite Studio, charger la base de données panamapapers.sqlite3 et s'y connecter. (On renvoie à l'annexe pour les détails techniques de mise en place.)

(1) Les tables sur lesquelles on va travailler sont déjà créées dans la BDD. Néanmoins, il faut savoir créer une table et y ajouter des entrées.

Commandes SQL

La commande `CREATE TABLE` permet de créer une table et de renseigner son nom (sur la première ligne) et les différents attributs.

On spécifier le type de chaque attribut, une chaîne de caractères (TEXT), un nombre entier ou réel (INTEGER ou FLOAT), une date (DATE, au format AAAA-MM-JJ), ou aussi un booléen (un objet qui prend 2 valeurs, TRUE ou FALSE - BOOLEAN).

Le mot-clé NOT NULL nous empêche de créer une ligne sans renseigner l'attribut correspondant, indispensable notamment pour la PK.

☞ Les mots-clés SQL sont insensibles à la casse mais ils sont souvent écrits en majuscules.

(a) On va commencer par créer une table (que l'on effacera ensuite car elle ne nous servira pas pour autre chose que manipuler la commande introduite ci-dessus) intitulée TP_SQL.

```
CREATE TABLE TP_SQL (
        id INTEGER,
        nom_etudiant TEXT NOT NULL,
        exp_sql BOOLEAN,
        khube BOOLEAN,
        date_TP DATE,
        note FLOAT,
        PRIMARY KEY(id),
)
```

Après avoir exécuté la requête, on voit la table TP_SQL dans le schéma de la BDD.

(b) La table est alors vide. On va la remplir.

Commandes SQL

Pour insérer une ligne dans une table, on utilise la commande `INSERT INTO` suivie du nom de la table, puis, entre des premières parenthèses la liste des attributs que l'on souhaite compléter, ensuite, le mot clé VALUES suivi d'une autre paire de parenthèses qui contiennent les valeurs que l'on souhaite insérer.

☞ Ces valeurs doivent être dans le même ordre que le nom des attributs.

☞ Si certaines valeurs sont laissées vides, elles ne contiennent aucune valeur.

(i) Insérer alors une ligne avec l'identifiant 0, votre nom, si vous avez déjà fait du sql, si vous êtes khube ou non et la date.

(ii) Insérer une autre ligne avec les informations relatives à votre voisin.e.

(c) On veut supprimer des lignes (puis la table).

Commandes SQL

La commande `DELETE FROM nom_de_la_table WHERE condition` permet de supprimer toutes les lignes de la table où la condition est vérifiée.

Supprimer d'abord la ligne de votre voisin. Vérifier qu'elle a disparu.
Supprimer la ligne vous concernant.

(d) La table ne contient plus aucune ligne mais est encore présente dans le schéma de la BDD.

Commandes SQL

La commande `DROP TABLE nom_de_la_table` permet de supprimer une table du schéma de la BDD.

Supprimer la table `TP_SQL`.

(2) La commande `SELECT`.
Ce commande sert à dialoguer avec la BDD. À chaque exécution d'une requête avec `SELECT`, le logiciel renvoie une table.

(a) Exécuter la requête

```
SELECT * FROM entity ;
```

À quoi sert le caractère $*$ derrière `SELECT` ?

(b) Comparer la commande précédente avec

```
SELECT DISTINCT * FROM entity ;
```

À quoi sert le mot-clé `DISTINCT` ?

(c) Exécuter la commande

```
SELECT id, name, status FROM entity ;
```

Que voyez-vous ? Quelle est la méthode SQL pour obtenir une projection ?

(3) Commençons maintenant notre enquête ! Nous allons chercher cette mystérieuse société dont le nom est *Big Data Crunchers Limited*.
Il s'agit donc de fabriquer une restriction de la relation **entity**.
C'est l'affaire du mot clé `WHERE`.

(a) Exécuter

```
SELECT * FROM entity WHERE name ='Big Data Crunchers Ltd.';
```

Qu'apprend-on sur la société qu'on cherche ?

Commandes SQL

Pour trouver la société *Big Data Crunchers Limited*, nous avons utilisé l'opérateur de comparaison "=".
D'autres opérateurs de comparaison existent :

- $A = B$: A est égal à B.
- $A <> B$: A est différent de B.
- $A > B$ et $A < B$: A est supérieur/inférieur à B.
- $A >= B$ et $A =< B$: A est supérieur/inférieur ou égal à B.
- A `BETWEEN` A `AND` C : A est compris entre B et C.
- A `LIKE 'chaine de caractère'` : pour comparer A à une chaine de caractère donnée.
- A `IN` $(B_1, B_2, \cdots)$: A est présent dans la liste $(B_1, B_2, \cdots)$.
- A `IS NULL` : A n'a pas de valeur.

(b) Les opérateurs logiques `OR`, `AND` et `NOT` signifient respectivement OU, ET et NON. Grâce à ces opérateurs, on peut complexifier un peu nos conditions.
Exécuter

```
SELECT * FROM entity
WHERE (id < 10000004 AND (NOT id < 10000000)) OR (name = 'Big Data
    Crunchers Ltd.');
```

et interpréter.

(4) Le produit cartésien s'obtient facilement grâce à `SELECT`.

```
SELECT * FROM entity, address ;
```

Attention le temps de calcul est parfois long.

(5) Nous voulons maintenant savoir si *Big Data Crunchers Limited* a servi d'intermédiaire. Les intermédiaires peuvent être soit des personnes physiques, soit des sociétés. Il y a donc peut-être des sociétés qui sont à la fois dans la table **intermediary** et dans **entity**.
Pour utiliser un opérateur binaire (intersection, union, différence) il faut que les tables aient le même schéma, ce qui n'est pas le cas ici.

(a) Pour avoir la liste des sociétés de **entity** et des intermédiaires, on utilise le mot clé UNION. Exécuter (et commenter)

```
SELECT name, id_address FROM entity
UNION
SELECT name, id_address FROM intermediary ;
```

(b) Utiliser le mot clé EXCEPT pour trouver les sociétés qui ne sont pas des intermédiaires.

(c) Utiliser enfin le mot clé INTERSECT pour trouver les sociétés qui sont aussi des intermédiaires. Chercher si *Big Data Crunchers Limited* en fait partie.

(d) (∗) Imaginons que deux tables présentent un grand nombre d'attributs et on veut savoir si une ligne d'une table se trouve aussi dans la deuxième table, sans avoir à comparer tous les attributs un par un. Trouver une commande SQL qui réponde à ce problème.

(6) Nous voulons maintenant trouver l'adresse de la mystérieuse société *Big Data Crunchers Limited*. Pour cela il va falloir faire une *jointure*.

Commandes SQL

Considérons deux tables table1 (de clé étrangère fk) et table2 de clé primaire pk). Pour obtenir la jointure de la table table1 avec la table table2 sous la condition fk = pk, on utilise la commande

```
SELECT * FROM table1 JOIN table2 ON (table1.fk = table2.pk);
```

(a) Quelle commande permet alors de faire la jointure de la table entity avec la table address sous la condition id_adress=id_adress?

(b) À l'aide de l'Exercice 3, expliquer pourquoi la commande

```
SELECT * FROM entity, address WHERE entity.id_address = address.id_address
    ;
```

a le même effet que la commande de jointure.

(c) Retrouver maintenant l'adresse de la société mystère.

(7) Retrouvons maintenant les intermédiaires qui ont participé à la création de la société *Big Data Crunchers Limited*.

(a) Quel est le rôle de la table **assoc_inter_entity** ?

(b) Exécuter et interpréter la commande

```
SELECT
    i.id as intermediary_id,
    i.name as intermediary_name,
    e.id as entity_id,
    e.name as entity_name,
    e.status as entity_status
FROM
    intermediary i,
    assoc_inter_entity a,
    entity e
WHERE
    a.entity = e.id
    AND a.inter = i.id
    AND e.name = 'Big Data Crunchers Ltd.' ;
```

Le mot-clé as permet de renommer les attributs en quelque chose de plus lisible. De même, les lettres a, e et i dans le FROM sont aussi des alias.

(c) Conclure sur les intermédiaires qui ont servi à la création de *Big Data Crunchers Limited*.

(8) Nous voulons maintenant incriminer les sociétés qui ont bénéficié des services des deux intermédiaires que nous avons trouvés, dans chacune des juridictions. Nous devons donc faire des **agrégations**.

(a) Exécuter la commande

```
SELECT status, count(*) FROM entity GROUP BY status ;
```

Ici nous avons placé l'attribut de partitionnement status derrière le mot-clé GROUP BY et la fonction d'agrégation count() dans le SELECT. Que renvoie cette commande ?

(b) Exécuter et interpréter

```
SELECT max(incorporation_date) AS maxi FROM entity;
```

La fonction max est une autre fonction d'agrégation.

(c) Exécuter et interpréter la commande

```
SELECT
    i.id as intermediary_id,
    i.name as intermediary_name,
    e.jurisdiction,
    count(*)
FROM
    intermediary i,
    assoc_inter_entity a,
    entity e
WHERE
    a.entity = e.id
    AND a.inter = i.id
    AND (i.id = 5000 OR i.id = 5001)
GROUP BY
    i.id, i.name, e.jurisdiction;
```

(9) Enfin, il est parfois utile de savoir réorganiser les lignes d'une table selon le critère que l'on choisit. On utilise pour ça le mot clé ORDER BY.

(a) Que constatez-vous lorsqu'on exécute

```
SELECT * FROM entity ORDER BY lifetime DESC;
```

(b) Enfin, exécuter la commande ci-dessous. Interpréter.

```
SELECT
    i.id AS intermediary_id,
    i.name AS intermediary_name,
    e.jurisdiction,
    e.jurisdiction_description,
    count(*) as cnt
FROM
    intermediary i,
    assoc_inter_entity a,
    entity e
WHERE
    a.entity = e.id AND
    a.inter = i.id AND
    (i.id = 5000 OR i.id = 5001)
GROUP BY
    i.id, i.name, e.jurisdiction, e.
        jurisdiction_description
ORDER BY
    cnt DESC ;
```

À retenir!

- La commande CREATE TABLE.
- Les mots clés PRIMARY KEY et FOREIGN KEY avec la méthode pour pointer vers une clé candidate d'une autre table.
- Les stratégies de projection restriction et produit cartésien avec SELECT.
- Pour la restriction, les opérateurs de comparaison.
- Les opérateurs logiques OR, AND et NOT.
- Le mot clé DISTINCT.
- Les opérations binaires sur les tables données par les mots clé UNION, INTERSECT, EXCEPT.
- La méthode de jointure.
- Le mot-clé GROUP BY pour faire un agrégat.
- Les fonctions d'agrégation count(*), min, max, avg, sum (compter, minimum, maximum, moyenne et somme).
- Le mot-clé ORDER BY.

4 Un exemple d'exercice de concours - Écricome sujet zéro (2)

Sur le marché des véhicules d'occasion, on observe en général une baisse du prix de revente (ou décote) d'un véhicule lorsque le nombre de kilomètres parcourus augmente. Une bonne estimation de cette baisse de prix permet au vendeur de fixer avec précision le prix de revente d'un véhicule.

On dispose d'une base de données comportant deux tables `vehicule` et `annonce` décrites ci-dessous.

- La table `vehicule` recense des informations sur les modèles de véhicules en vente sur le marché. Elle est composée des attributs suivants :
  - `id_vehicule` (de type `INTEGER`) : un code permettant d'identifier de façon unique chaque référence de véhicule (marque et modèle).
  - `marque` (de type `TEXT`) : le nom du constructeur du véhicule.
  - `modele` (de type `TEXT`) : le modèle du véhicule, un constructeur proposant en général plusieurs modèles de véhicules à la vente
  - `prix_neuf` (de type `INTEGER`) : prix de vente du véhicule neuf.

- La table annonce regroupe des informations sur un grand nombre d'annonces de véhicules d'occasion. Chaque enregistrement correspond à une annonce et possède les attributs suivants.
  - `id_annonce` (de type `INTEGER`) : un code permettant d'identifier chaque annonce de façon unique.
  - `id_vehicule` (de type `INTEGER`) : l'identifiant du modèle de véhicule vendu, qui correspond à l'identifiant utilisé dans la table `vehicules`.
  - `annee` (de type `INTEGER`) : année de première mise en circulation du véhicule.
  - `km` (de type `INTEGER`) : nombre de kilomètres parcourus par le véhicule au moment de la revente.
  - `prix_occasion` (de type `INTEGER`) : prix de vente du véhicule d'occasion.

(1) En justifiant brièvement, identifier une clef primaire dans chacune des tables `vehicule` et `annonce`, ainsi qu'une clef étrangère dans la table `annonce`.

(2) Écrire une requête SQL permettant d'extraire les noms de tous les modèles de véhicules mis en vente par le constructeur `Dubreuil Motors`.

(3) Expliquer le fonctionnement de la requête SQL suivante et préciser l'effet éventuel de cette requête sur chacune des tables `vehicule` et `annonce`.

```
UPDATE annonce
SET prix_occasion = prix_neuf
FROM vehicule
WHERE vehicule . id_vehicule = annonce . id_vehicule
AND vehicule . prix_neuf < annonce . prix_occasion
```

(4) À l'aide d'une jointure, écrire une requête SQL permettant d'obtenir, sur une même table, la liste de toutes les annonces de la table `annonce` avec les attributs suivants :

- l'identifiant de l"annonce `id_annonce`.
- le kilométrage `km`,
- le prix de vente du véhicule neuf `prix_neuf`,
- le prix de l'annonce d'occasion `prix_occasion`.

5 Annexe : Manipuler SQL sur sa machine

☞ **[BDD]** La base de données sur laquelle on va travailler est disponible en ligne, sur ma page en cliquant sur ce lien. Elle est écrite dans un format .sqlite3 (qui est en fait un groupe de fichiers .csv), qui est bien sûr compatible avec SQLite (que nous allons présenter ci-après). Attention, le fichier est gros (environ 80MB).

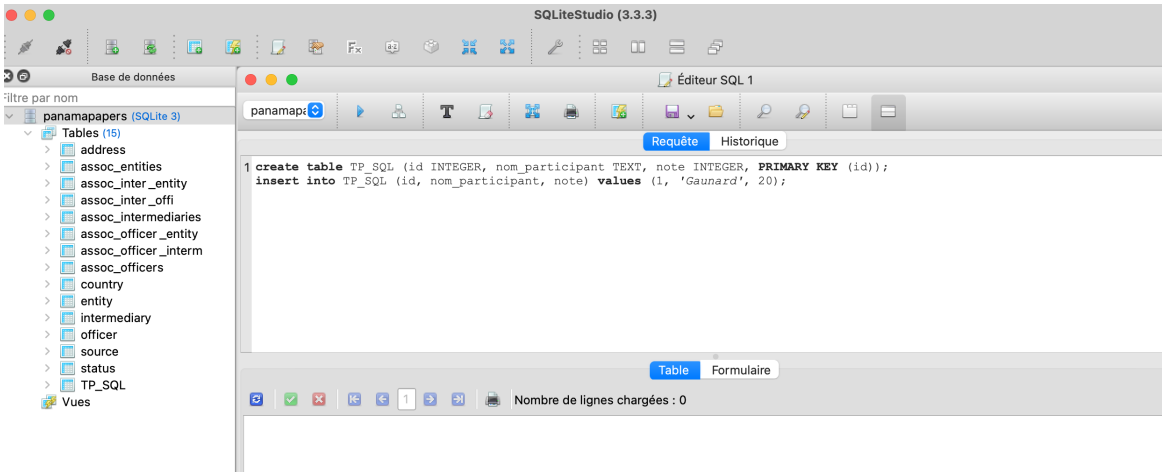
☞ **[SQLite]** Pour communiquer (et modifier) avec cette base de données, on utilise le langage SQL (*Structured Query Language*). Pour les requêtes écrites dans ce langage soient *interprétées* et exécutées, nous avons besoin d'un interpréteur de langage.
On a choisi SQLite, qui est gratuit (le code est en *OpenSource*) et comme son nom l'indique extrêmement léger (tant dans la taille du fichier source que dans son utilisation). Si SQLite est déjà installé sur certaines versions de macOS, on le trouvera ici.

☞ **[Interface]** On pourrait se contenter d'une manipulation des requêtes SQL via l'environnement `>sqlite3` d'un terminal, mais on trouve ça un peu austère. On a donc décidé d'utiliser un gestionnaire de base de données avec une interface intuitive; il sera facile et plus agréable d'y taper du script SQL et de voir immédiatement l'effet sur les tables de la BDD. On a choisi, utilisateurs de Mac que nous sommes, le logiciel SQLite Studio, disponible gratuitement ici.

Pour nos amis sous Windows, Cubrid semble être une alternative satisfaisante. On confesse ne l'avoir pour l'instant pas testée.

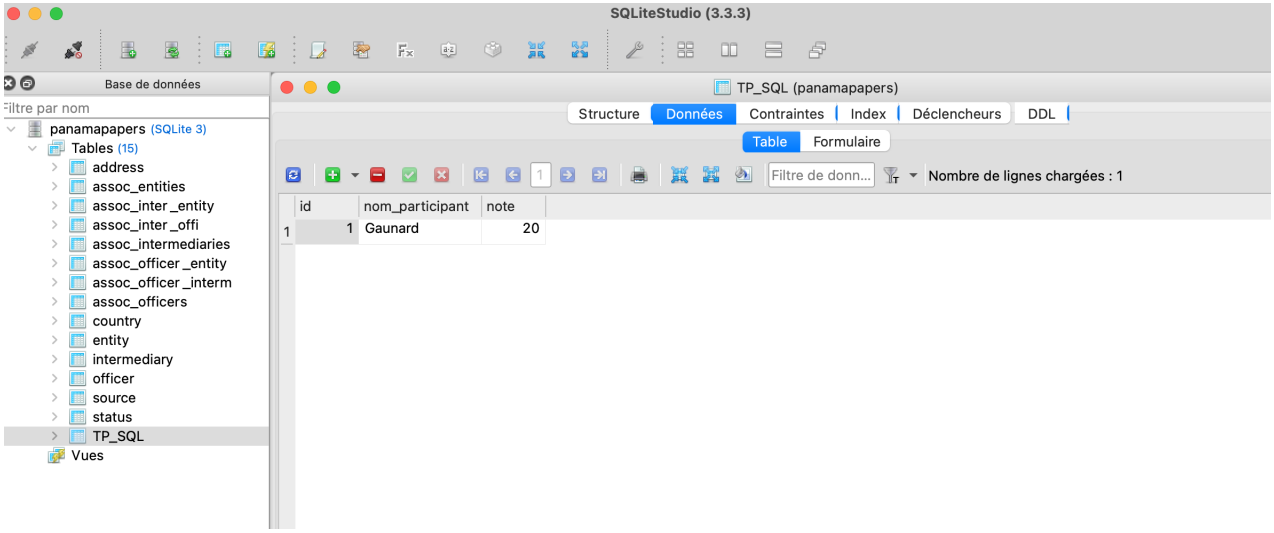
Capture d'écran

☞ On peut écrire les requêtes dans une fenêtre script SQL



Capture d'écran

☞ On visualise facilement le contenu d'une table et le schéma de la BDD



Capture d'écran

☞ On visualise facilement le résultat d'une requête