CHAPITRE VIII

MANIPULATIONS NUMÉRIQUES DE BASES DE DONNÉES

Table des matières

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 cours est séparée en deux parties. Dans un premier temps (plus théorique), nous allons analyser la structure sous laquelle apparaissent les fichiers volumineux de données (fichier .csv). Puis nous explorerons les possibilités du langage SQL dans un TP.

1. Bases de données (aspects théoriques)

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

Définition : Relation et Base de donnée.

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.

Remarque 1.1.1. 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, plus léger et moins riche graphiquement que le tableur.

Exemple 1.1.2. 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 d'options ESH/HGG et Maths Appli/Maths Appro.

Les étudiants de Dumas				
identifiant	**date de naissance**	**ville de provenance**	**ESH ou HGG**	**Appli ou Appro**
1	04/03/2003	Paris	ESH	Appli
2	22/05/2004	Versailles	HGG	Appli
3	22/10/2003	Versailles	HGG	Appro
4	27/07/2001	Paris	ESH	Appli
5	10/01/2002	Saint-Cloud	ESH	Appli
6	10/01/2002	Rambouillet	ESH	Appro
[...]	[...]	[...]	[...]	[...]

Remarque 1.1.3. 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 : Un peu de vocabulaire

- 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.

1.1.1. *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 prennent les mêmes options. 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 : Clé

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

Exemple 1.1.4. 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 10/01/2002. Le groupe d'attributs {**date de naissance**, **Maths Appli/Maths appro**} est une clé.

Exercice 1.1.5. On considère la relation **voiture** décrite par la table ci-dessous.

immatriculation	pays immatriculation	couleur	nombre places	propriétaire	prop num secu	prop num voiture
2883-AA	France	bleue	5	Luc	1134212	1
1923-BD	France	verte	7	Lucia	2021726	1
PLT-28-190	Portugal	bleue	7	Gael	1012612	1
2383-ZN	France	noire	9	Léila	2125312	2
1209-NQ	France	noire	5	Léila	2125312	1
2634-OS	France	bleue	5	Gael	1162732	1

Dans cette table, **immatriculation** désigne le numéro d'immatriculation de la voiture, **pays immatriculation** le pays dans lequel la voiture est immatriculé, **couleur** la couleur de la voiture, **nombre places** le nombre de places de la voiture, **propriétaire** est le pronom du propriétaire, **prop num secu** le numéro de sécurité sociale du propriétaire (en France, chaque personne possède un unique numéro de sécurité sociale), **prop num voiture** rend compte du fait que si un propriétaire possède plusieurs voitures, il peut les numéroter.

1. Le groupe d'attributs {**immatriculation**} est-il une clé pour notre relation ?
2. Le groupe d'attributs {**immatriculation**, **pays immatriculation**} est-il une clé pour notre relation ?
3. Le groupe d'attributs {**immatriculation**, **pays immatriculation**, **couleur**} est-il une clé pour notre relation ?
4. Le groupe d'attributs {**prop num secu**} est-il une clé pour notre relation ?
5. Le groupe d'attributs {**prop num secu**, **prop num voiture**} 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 : Clés candidate, clé primaire

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**.

Remarque 1.1.6. 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.

Méthode : Comment choisir la clé primaire ?

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 : Clé artificielle

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

Remarque 1.1.8. Un bon programmateur prend en fait l'habitude d'utiliser comme clés primaires des clés artificielles. Il y a (au moins) deux raisons à ça.

1. Dans l'exemple précédent des immatriculations de voitures, nous nous demandions si les deux attributs {**prop num secu**, **prop num voiture**} constituaient une clé candidate. La réponse était oui à condition de ne supposer qu'on ne prenait en compte que des propriétaires de nationalité française. Cependant, si notre application évolue, et que nous souhaitons généraliser à des personnes d'autres nationalités, notre clé n'est plus bonne. Une clé artificielle est donc une bonne solution à *l'évolutivité* de notre base de données.
2. 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.

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.

Étudier dans les villes autour de Saint-Cloud			
ville (PK)	**temps de trajet**	**facilité de travail**	**difficulté de travail**
Saint-Cloud	0	Proche	Pas de bibliothèques
Paris	45	Dynamique/facile de trouver de l'aide	Bruyant
Versailles	30	Joli cadre de vie	Problèmes de train fréquents
Rambouillet	60	Facile de faire du sport	Temps de transport
[...]	[...]	[...]	[...]

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 Saint-Cloud.

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

Définition : Clé étrangère

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 1.1.9. 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 Saint-Cloud. La ville de provenance est donc une clé étrangère dans la relation des étudiants.

Méthode : Comment choisir les attributs à regrouper dans une relation ?

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 Saint-Cloud à une ville voisine change (bientôt la ligne 15 !), 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 Saint-Cloud, mais qu'aucun étudiant n'habite dans cette ville, on ne peut pas.

Pour éviter la redondance, on peut suivre la règle suivante :

Si un attribut A dépend uniquement d'un groupe d'attributs G et que G est minimal (c'est-à-dire que si on enlève un attribut à G, alors A ne dépend plus uniquement que des attributs restants), alors il est possible de créer une nouvelle table qui ne contient que les attributs A et G. G sera d'ailleurs une clé candidate pour la nouvelle table et tout autre attribut B qui ne dépend que de G peut être déplacé aussi dans la nouvelle relation.

Exemple 1.1.10. Dans notre exemple récurrent, nous avons appliqué ce principe dans la table des villes voisines : le temps de trajet, les facilités et difficultés à travailler ne dépendent que de la ville et nous les avions placé dans la nouvelle table.

Définition : Normalisation

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.

1.1.2. *Les tables d'association.* 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 clé étrangère 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 suivante :

Définition : Table d'association

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 1.1.11. Construisons donc la relation des écoles de commerce, puis la table d'association des étudiants et des écoles dans lesquelles ils sont admissibles.

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
[...]	[...]	[...]

On aurait presque pu considérer que le rang de l'école de commerce est une clé candidate mais cela nous aurait empêché de considérer que des écoles peuvent être à égalité.

Nous pouvons maintenant 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
[...]	[...]

Remarque 1.1.12. 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.

1.2. **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.

1.2.1. *Projection et Restriction.*

Définition : Projection

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

Exemple 1.2.1. 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	04/03/2003
2	22/05/2004
3	22/10/2003
4	27/07/2001
5	10/01/2002
6	10/01/2002

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 : Restriction

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 1.2.2. **1.** Le résultat de la restriction de la table des étudiants sous la condition date de naissance $>$ 11/01/2002 produit la table suivante.

Les étudiants de Dumas				
identifiant	**date de naissance**	**Ville de provenance**	**ESH ou HGG**	**Appli ou Appro**
1	04/03/2003	Paris	ESH	Appli
2	22/05/2004	Versailles	HGG	Appli
3	22/10/2003	Versailles	HGG	Appro

2. La projection de la table des étudiants avec la condition ESH ou HGG = 'ESH' produit la table suivante.

Les étudiants de Dumas				
identifiant	**date de naissance**	**Ville de provenance**	**ESH ou HGG**	**Appli ou Appro**
1	04/03/2003	Paris	ESH	Appli
4	27/07/2001	Paris	ESH	Appli
5	10/01/2002	Saint-Cloud	ESH	Appli
6	10/01/2002	Rambouillet	ESH	Appro

1.2.2. *Union, Différence, Intersection.*

Définition : Union

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

Remarque 1.2.3. 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.

Exemple 1.2.4. Si on a dans deux relations différentes des renseignements sur les étudiants de la prépa, par exemple

identifiant	**date de naissance**	**Ville de provenance**	**ESH ou HGG**	**Appli ou Appro**
1	04/03/2003	Paris	ESH	Appli
2	22/05/2004	Versailles	HGG	Appli
3	22/10/2003	Versailles	HGG	Appro
4	27/07/2001	Paris	ESH	Appli

et

identifiant	**date de naissance**	**Ville de provenance**	**ESH ou HGG**	**Appli ou Appro**
16	07/12/2000	Saint-Cloud	HGG	Appro
25	01/02/2003	Paris	ESH	Appli
32	11/09/2001	Saint-Cloud	ESH	Appro

alors la réunion des deux relations produit la table

identifiant	date de naissance	Ville de provenance	ESH ou HGG	Appli ou Appro
1	04/03/2003	Paris	ESH	Appli
2	22/05/2004	Versailles	HGG	Appli
3	22/10/2003	Versailles	HGG	Appro
4	27/07/2001	Paris	ESH	Appli
16	07/12/2000	Saint-Cloud	HGG	Appro
25	01/02/2003	Rambouillet	ESH	Appli
32	11/09/2001	Saint-Cloud	ESH	Appro

La différence est l'opération inverse de la réunion.

Définition : Différence

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

Remarque 1.2.5. Attention à l'ordre, l'opération de différence n'est pas commutative.

Enfin l'intersection correspond à la même opération qu'en théorie des ensembles.

Définition : Intersection

L'*intersection* de deux relations de même schéma R_1 et R_2 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.

1.2.3. *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	Martinez
prof2	Danino
prof3	Lièvre
prof4	Perrot
prof5	Roumaneix
prof6	Merlin

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 : Produit cartésien

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.

Dans notre exemple, le produit cartésien des relations professeurs et étudiants renvoie la table suivante

identifiant prof	Nom du professeur	identifiant étudiant
prof1	Martinez	1
prof1	Martinez	2
prof1	Martinez	3
prof2	Danino	1
prof2	Danino	2
prof2	Danino	3
prof3	Lièvre	1
prof3	Lièvre	2
prof3	Lièvre	3
prof4	Perrot	1
prof4	Perrot	2
prof4	Perrot	3
prof5	Roumaneix	1
prof5	Roumaneix	2
prof5	Roumaneix	3
prof6	Merlin	1
prof6	Merlin	2
prof6	Merlin	3

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

1.2.4. *Jointure.* La jointure est le concept fondamental de l'algèbre relationnelle. Jusqu'à présent, nous avons vu que les clés étrangères 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 Saint-Cloud 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 : Jointure (interne)

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 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.*

Remarque 1.2.6. Il s'agit d'une jointure interne, la seule au programme de ECG. Sans rentrer dans les détails, mentionnons seulement qu'il existe d'autres types de jointure, qui peuvent éventuellement différer de celle-ci lorsqu'un attribut d'une ligne n'est pas renseigné (ce qui n'est pas interdit, ni dans la théorie, ni dans le langage SQL).

Exemple 1.2.7. La jointure de la table des étudiants

identifiant (PK)	date de naissance	ville de provenance (FK)	ESH ou HGG	Appli ou Appro
1	04/03/2003	Paris	ESH	Appli
2	22/05/2004	Versailles	HGG	Appli
3	22/10/2003	Versailles	HGG	Appro
4	27/07/2001	Paris	ESH	Appli
5	10/01/2002	Saint-Cloud	ESH	Appli
6	10/01/2002	Rambouillet	ESH	Appro

avec la table des villes (projetée pour ne garder que le temps de trajet)

ville (PK)	temps de trajet
Saint-Cloud	0
Paris	45
Versailles	30
Rambouillet	60

selon la condition ville de provenance = ville donne la table suivante

etudiant. identifiant	etudiant. date de naissance	etudiant. ville de provenance	etudiant. ESH ou HGG	etudiant. Appli ou Appro	ville. nom de ville	ville. temps de trajet
1	04/03/2003	Paris	ESH	Appli	Paris	45
2	22/05/2004	Versailles	HGG	Appli	Versailles	30
3	22/10/2003	Versailles	HGG	Appro	Versailles	30
4	27/07/2001	Paris	ESH	Appli	Paris	45
5	10/01/2002	Saint-Cloud	ESH	Appli	Saint-Cloud	0
6	10/01/2002	Rambouillet	ESH	Appro	Rambouillet	60

Exercice 1.2.8. Montrer que la jointure est l'enchaînement d'un produit cartésien et d'une restriction.

1.2.5. *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

Quelle est le temps de trajet moyen des étudiants pour chaque option ?

Il s'agit donc de calculer une valeur pour chaque choix d'options ESH-Appli, ESH-Appro, HGG-Appli, HGG-Appro.

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 : Agrégat

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 : Fonction d'agrégation

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.

Exercice type concours.

Un fabricant d'ordinateur souhaite publier des données statistiques sur la durée de vie des appareils fabriqués à partir de l'an 2000. Dans une base de données, on dispose d'une table ordinateur contenant des informations sur tous les ordinateurs produits par le fabricant. Cette table possède les attributs (ou colonnes) suivants.

- id (de type INTEGER) : le numéro d'identification de l'ordinateur.
- année_ fabrication (de type INTEGER) : l'année de fabrication de l'ordinateur.
- adresse_ip (de type INTEGER) : l'adresse IP de l'ordinateur.
- année_ panne (de type INTEGER) : l'année où l'ordinateur a cessé de fonctionner, valant -1 si l'ordinateur est encore en état de marche.

1. **a.** Écrire une requête SQL permettant de déterminer le nombre total d'ordinateurs produits par le fabricant.

b. Écrire une requête SQL permettant de déterminer le nombre total d'ordinateur ayant cessé de fonctionner exactement un an après leur production.

c. Dans cette question uniquement, on suppose que la durée de vie en années d'un ordinateur est une variable aléatoire de loi géométrique de paramètre p inconnu.

Expliquer de quelle manière le résultat des requêtes écrites dans les questions précédentes permettent d'estimer le paramètre p.

Voir le chapitre consacré à l'estimation.

2. Un attribut durée_ vie, de type INTEGER a été ajouté à la table ordinateur. Aux champs de l'attribut durée_ vie a été affectée la valeur -1.

Écrire une requête SQL permettant de modifier la table ordinateur en affectant, pour chaque ordinateur sa durée de vie à l'attribut durée_ vie. Dans le cas des ordinateurs encore en état de marche, on ne modifiera pas la valeur -1 déjà affectée.

3. Dans cette question, on cherche à déterminer s'il est raisonnable de représenter la durée de vie d'un ordinateur par une variable aléatoire de loi géométrique d'un certain paramètre p que l'on cherchera à approcher.

a. Expliquer comment le résultat de la requête suivante permet d'obtenir une valeur approchée de p :

```
SELECT AVG(duree_vie) FROM ordinateurs
```

b. La base de données compte au total 10 000 ordinateurs. On exécute les commandes suivantes :

```
SELECT COUNT(*)/10000 FROM ordinateurs WHERE duree_vie = 1;
SELECT COUNT(*)/10000 FROM ordinateurs WHERE duree_vie = 2;
.
.
.
SELECT COUNT(*)/10000 FROM ordinateurs WHERE duree_vie = 24;
```

Expliquer de quelle manière, les données de la table ordinateurs peuvent être exploitées pour déterminer s'il est raisonnable de représenter la durée de vie d'un ordinateur par une variable aléatoire qui suit la loi géométrique.

On pourra s'aider de la question ***8*** *de l'exercice 1 de* ***ECRICOME*** *2024 dont cet exercice est extrait.*

2. Le langage SQL (enquête sur les Panama Papers)

Cette deuxième partie du cours est rédigée sous forme d'un TP. 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 et la base de données que nous allons utiliser.

1. Le programme à télécharger s'appelle SQLite, il est léger, libre et gratuit, et se trouve ici.
2. La base de données est disponible sur ma page Internet à cette adresse. Elle est écrite dans un format .sqlite3 (qui est en fait un groupe de fichiers .csv), qui est bien sûr compatible avec le logiciel SQLite. Attention, le fichier est gros (environ 80MB).

2.1. **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 œil. 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 Etat. 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'Etat, 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.

2.2. **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 TP, notamment la société *Big Data Crunchers Limited*. Elle a été créée de toutes pièces pour servir de fil rouge.

Remarque 2.2.1. 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.

2.3. **Objectif du TP.** Je vous propose de vous mettre dans la peau d'un enquêteur qui enquête sur le financement d'un réseau criminel.

Vous avez au cours de votre 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. Vous ne savez pas qui se cache derrière cette société, mais vous pensez 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.

Vous allez donc enquêter sur cette mystérieuse société, mais aussi sur les intermédiaires qui ont aidé à la créer, car vous pensez qu'il sera peut-être possible de les accuser de complicité.

2.4. **Un peu de vocabulaire.**

Définition : Société

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

Définition : Société offshore

C'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 : Société écran

Une société écran est une société fictive, créée pour dissimuler les transactions financières d'une ou de plusieurs autres sociétés.

Définition : Intermédiaire

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.

Définition : 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.

Définition : Bénéficiaire

en anglais : beneficial owner ou beneficiary) C'est la personne réellement propriétaire de la société. Dans le monde offshore, l'identité du bénéficiaire est souvent gardée secret.

Remarque 2.4.1. 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.

2.5. **Déroulement du TP.**

1. Commençons tout d'abord par créer la table **entity** qui accueillera les sociétés offshore contenues dans les Panama Papers.

```
CREATE TABLE entity (
  id INTEGER,
  name TEXT NOT NULL,
  jurisdiction TEXT,
  jurisdiction_description TEXT,
  company_type TEXT,
  id_address INTEGER
  incorporation_date DATE,
  inactivation_date DATE,
  status TEXT,
  service_provider TEXT,
  country_codes TEXT,
  countries TEXT,
  source TEXT,
  PRIMARY KEY(id),
  FOREIGN KEY(id_address) REFERENCES address(id)
)
```

La commande CREATE TABLE permet de créer une table et de renseigner son nom (sur la première ligne) et les différents attributs. Nous devons spécifier le type de chaque attribut, ici du texte (chaîne de caractères), un nombre entier ou une date. D'autres options sont possibles, comme un nombre réel ou un booléen (un objet qui prend 2 valeurs, VRAI ou FAUX).

Le mot-clé NOT NULL nous empêche de créer une ligne sans renseigner le nom de la société.

Enfin, nous avons indiqué la clé primaires de la table et une clé étrangère en l'attribut id-address qui fait référence à la colonne id de la table **address**.

2. Maintenant que nous avons créé la structure de la table, insérons une ligne

```
INSERT INTO entity(id, name, jurisdiction,
    jurisdiction_description, incorporation_date) VALUES
     (0,'Une societe', 'IMG', 'Le pays imaginaire', '
    2023-02-03');
```

Nous spécifions la table à remplir grâce à INSERT INTO, puis nous indiquons entre parenthèse les colonnes que nous voulons compléter, puis nous donnons les valeurs à insérer après le mot clé VALUES. Ces valeurs doivent être dans le même ordre que le nom des attributs.

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

3. Chercher à quoi sert la commande DELETE FROM et supprimer de la table la ligne que nous venons de créer.

Nous supposons dorénavant que la table ***entity*** *a été remplie : c'est celle que vous avez téléchargé dans la base de données.*

4. La commande SELECT permet de communiquer avec la base de données. À chaque requête avec SELECT, le SGBDD nous renvoie une table. Exécuter la commande

```
SELECT * FROM entity ;
```

Le caractère * derrière SELECT signifie que nous voulons obtenir toutes les lignes et toutes les colonnes disponibles.

Comparer la commande précédente avec

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

À quoi sert le mot-clé DISTINCT ?

5. Exécuter la commande

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

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

6. 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. Exécuter

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

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

Remarque 2.5.1. 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.

7. 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.

8. Le produit cartésien s'obtient facilement grâce à SELECT :

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

Attention le temps de calcul est parfois long.

9. 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. On suppose que deux sociétés qui ont même nom et même adresse sont les mêmes. Faire une projection des tables **intermediary** et dans **entity** pour ne conserver que les attributs name et id_address.

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

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

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

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

e. (∗) 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.

10. Nous voulons maintenant trouver l'adresse de la mystérieuse société *Big Data Crunchers Limited*.

a. Nous allons déjà faire la jointure de la table **entity** avec la table **address** Pour cela, il faut exécuter la commande

```
SELECT *
FROM entity
  JOIN address ON entity.id_address = address.
    id_address;
```

Les tables à joindre sont en effet **entity** et **address** (dans cette ordre) et la condition de jointure apparaît après le mot-clé ON. Si la condition de jointure porte sur un groupe d'attributs plus grand, on utilise la syntaxe

```
SELECT * FROM t1 JOIN t2 ON (t1.fk1 = t2.pk1 AND t1.
    fk2 = t2.pk2);
```

b. À l'aide de l'exercice 1.2.8, 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.

11. 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*.

12. 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. Comparer les requêtes

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

et

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

Quel est l'intérêt de la première option ?

d. 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;
```

13. 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 comme dans la commande

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

a. Que constatez-vous lorsqu'on exécute

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

b. Enfin, exécuter la commande

```
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 ;
```

Qu'en déduisez-vous ?

2.6. **Conclusion : les commandes 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.

3. Sujets d'annales en lien avec ce chapitre.

Ce chapitre est une nouveauté du programme 2022. À une seule exception près, aucun sujet n'a pour le moment incorporé le matériel que nous venons de voir ici. Difficile de savoir à quoi s'attendre donc.

1. **ECRICOME**
   - 2024 Exercice 1 Partie 4.
2. **EDHEC**
   -

3. EML

-

4. HEC/ESSEC

-