Informatique - TD 8 | Bases de données

Dans la première partie (p. 1 à 3) de ce TP, on rappelle ce qu'est une base de données et le vocabulaire qui lui est associé. Ensuite (p. 3), vous pourrez télécharger si besoin un logiciel adapté à ce travail, afin de vous servir pour tester les différentes "requêtes" (à partir de la page 4).

Principe et vocabulaire

But

Dans les TP précédents, on a rappelé comment ranger et manipuler des données de tout type dans des listes et des dictionnaires, ainsi que des nombres dans des tableaux. On souhaite maintenant pouvoir stocker, gérer, retrouver des informations rapidement et les traiter (éventuellement suivant plusieurs critères à la fois). Il se trouve que pour ce type d'actions un peu plus complexes, ces outils se révèlent souvent insuffisants ou pénibles à utiliser. Admettons par exemple que l'on souhaite stocker les informations (nom et adresse de l'acheteur, détails de la commande, prix total, etc...) relatives à un ensemble de bons de commandes du type ci-dessous :

FACTURE N°100

BCPST & Co
1 rue des Deuxième Année,
77300 FONTAINEBLEAU
72 68 25 52 87

DATE : 01/01/2050

ADRESSE DE LIVRAISON
BONHOMME EN ROUGE
10 rue des Lutins
99 999, PAULNOR
06 24 12 20 22

mail : santa.klaus@hottemail.com

Détails	Réf	Prix	Quantité	MONTANT HT
Traineau avec hotte	01-100	10 000,00 €	1	10 000,00 €
costume rouge	01-203	102,30 €	1	102,30 €
rênes marron étoilées	04-502	203,90 €	1	203,90 €
			SOUS-TOTAL	10 306,20 €
			TAUX TVA	20,00%
			Réduction spécial Noël	1 000,00 €
			TOTAL	13 367,44 €

Montant réglé par renne volant le : 01/01/2050.

NOUS VOUS REMERCIONS DE VOTRE CONFIANCE.

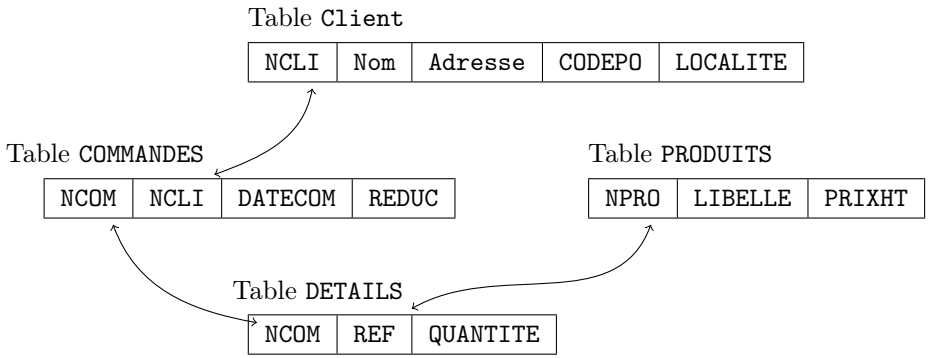
La première idée que l'on peut avoir est de stocker par tout ceci dans un grand tableau du type

NumCom	Date	NomClient	adresse	mail	tél	Produit	Quantité	Prix	...

mais ce type de stockage dit "plat" a un grand incovénient. Il y a beaucoup de doublons. Par exemple, pour cette commande, il faudrait écrire 3 lignes différentes. (Une pour chaque produit acheté qui contient 3 fois les coordonnées complètes du client.)
Si on imagine limiter le nombre de produits par commande, on peut ne faire qu'une seule ligne par commande :

NumCom	Date	Nom	adr.	mail	tél	Produit1	Prix	Produit2	etc...

mais si le client passe plusieurs commandes, ses données personnelles seront néanmoins stockées inutilement à de multiples reprises. On peut également voir que les prix des produits potentiellements achetés par plusieurs clients apparaissent également inutilement sur plusieurs lignes différentes.
On préférera donc utiliser un *"schéma relationnel"*, ici on verra qu'on peut utiliser un ensemble de plusieurs tableaux reliés entre eux par des points communs (qui ne portent pas forcément le même nom, mais qui désignent la même chose) :



Pour ce faire, on peut imaginer
- créer un ou plusieurs tableau(x) python dont chaque ligne contiendrait toutes les informations souhaitées, mais **c'est impossible** car on veut stocker d'autres données que des nombres
- mettre les informations dans des "listes de listes", mais ceci apparait **très compliqué et fastidieux**.
- utiliser des dictionnaires. Possible, mais difficile à manipuler pour ce qui est de la recherche rapide d'informations dans les valeurs du dictionnaire.

— utiliser un ou plusieurs tableaux de type excel ou openoffice, mais là aussi, la recherche d'informations pour un grand nombre d'entrées s'avère fastidieuse et souligner les liens entre les différents éléments de la (ou les) feuille(s) de calcul n'est pas évident.

On va donc utiliser un type de stockage de données tout autre : les bases de données.

L'objectif des bases de données va être d'organiser efficacement des données dans des tableaux ("tables") que l'on peut relier ponctuellement par des liaisons ("jointures") et de faciliter l'accès aux informations en faisant appel à "des requêtes".

Définition :

Une *base de données* est une collection de données (stockées dans un ou plusieurs fichiers). Elle est constituée d'un ensemble de *tables* contenant des données identifiées chacune par un unique identifiant. L'utilisateur accède aux données à travers des *requêtes* (*queries* en anglais).

Elle est faite pour enregistrer des faits, des opérations au sein d'un organisme (entreprise, administration, banque, université, hôpital, ...) ou autre. L'utilisateur n'a pas besoin de connaître la manière dont les données sont stockées pour interroger la base de données.

Plus précisément

Revenons au bon de commande factice donné en début de TP. Barrons les informations inutiles à stocker (redondantes ou calculables) :

FACTURE N°100

BCPST & Co
1 rue des Deuxième Année,
77300 FONTAINEBLEAU
72 68 25 52 87

DATE : 01/01/2050

ADRESSE DE LIVRAISON
BONHOMME EN ROUGE
10 rue des Lutins
99 999, PAULNOR
06 24 12 20 22

mail : santa.klaus@hottemail.com

Détails	Réf	Prix	Quantité	MONTANT HT
Traineau avec hotte	01-100	10 000,00 €	1	~~10 000,00 €~~
costume rouge	01-203	102,30 €	1	~~102,30 €~~
rênes marron étoilées	04-502	203,90 €	1	~~203,90 €~~
			SOUS-TOTAL	~~10 306,20 €~~
			TAUX TVA	20,00%
			Réduction spécial Noël	1 000,00 €
			TOTAL	~~10 867,44 €~~

On a barré les données MONTANT, TOTAL et SOUS-TOTAL.
On va maintenant pouvoir créer 4 tables, dans lesquelles ont va pouvoir trouver :
— L'ensemble des références de commande avec le client concerné
— L'ensemble des clients de la société
— Les détails des commandes
— L'enemble des produits de la société

Table COMMANDES

NCOM	NCLI	DATECOM	REDUC
100	512	01/01/2023	1000

Table Client

NCLI	Nom	Adresse	CODEPO	LOCALITE
512	BONHOMME EN ROUGE	10 Rue des Lutins	99999	PAULNOR

Table DETAILS

NCOM	REF	QUANTITE
100	01-100	1
100	01-203	1
100	04-502	1

Table PRODUITS

NPRO	LIBELLE	PRIXHT
01-100	Traineau avec hotte	10000
01-203	costume rouge	102,3
04-502	Rênes marron étoilées	203,9

Avec plusieurs commandes, on peut imaginger une partie de la base de données :

Table COMMANDES

NCOM	NCLI	DATECOM	REDUC
100	512	01/01/2023	1000
101	324	01/01/2023	825
102	409	02/01/2023	55

Table Client

NCLI	Nom	Adresse	CODEPO	LOCALITE
512	BONHOMME EN ROUGE	10 Rue des Lutins	99999	PAULNOR
324	PERSONNE FACTICE1	32 rue de l'invention	10198	VILLEINVENTEE
409	IDENTITE SECRETE	43 Bvd des concours	01238	AGROVETO

Table DETAILS

NCOM	REF	QUANTITE
100	01-100	1
100	01-203	1
100	04-502	1
101	02-098	2
101	03-422	3
102	04-325	10

Table PRODUITS

NPRO	LIBELLE	PRIXHT
01-100	Traineau avec hotte	10000
01-203	costume rouge	102,3
04-502	Rênes marron étoilées	203,9
02-098	Bottes de neige magiques TU	55
03-422	Grelots enchantés	15,25
04-325	Bonnet de Noël	8

Du vocabulaire

Relations

Définition :
Une base de données est un ensemble de tables que l'on appelle aussi *relations* ou *schéma de relations* (qui sont donc concrètement des tableaux à deux dimensions).

Reprenons cette table là :

NCLI	Nom	Adresse	CODEPO	LOCALITE
512	BONHOMME EN ROUGE	10 Rue des Lutins	99999	PAULNOR
324	PERSONNE FACTICE1	32 rue de l'invention	10198	VILLEINVENTEE
409	IDENTITE SECRETE	43 Bvd des concours	01238	AGROVETO

On note que chaque relation a un nom unique et contient les données relatives à des entités de même nature.

Définition :
On appelle *Attributs* les caractéristiques dont on souhaite enregistrer les valeurs, ce sont les colonnes des tables.

Dans notre exemple, les attributs sont :

NCLI	Nom	Adresse	CODEPO	LOCALITE

Définition :
On appelle *Domaine* le type d'un attribut. Chaque attribut a un seul type mais les attributs d'une même table peuvent être de différents types.

Dans notre exemple, les attributs sont tous des chaînes de caractères. Dans la table `PRODUIT`, on voit apparaître une colonne d'entiers. On peut rencontrer aussi des "flottants" (i.e. des réels), des dates, `NULL` (absence de donnée). Notons que ces deux derniers ne sont pas au programme.

Définition :
Les *enregistrements (ou **valeurs**)* d'une relation sont les lignes de la table. Chaque enregistrement est un n-uplet (n étant le nombre de colonne) dont les éléments appartiennent à chaque colonne de la table, il décrit les données relatives à une entité.

Voici un enregistrement de la relation `Client` :

512	BONHOMME EN ROUGE	10 Rue des Lutins	99999	PAULNOR

Définition :
Le nombre d'enregistrement est le *cardinal de la relation*. Les enregistrements sont deux à deux distincts (pas de redondance des données). De plus, l'ordre des enregistrements n'est pas fixé, ni garanti par le système de gestion de base de données (mais ceci n'a aucune importance). Les enregistrements constituent le *contenu* de la base.

Les clés

Définition :
Pour garantir la non-répétition des enregistrements, les bases de données utilisent une *clé* ou *identifiant* qui permet d'identifier chaque enregistrement. Elle garantit que deux enregistrements distincts ont deux clés distinctes.
La clé que l'on choisit pour caractériser les enregistrements est appelée *clé primaire*. Sa valeur permet de désigner une ligne de notre relation.

Dans le cas de la table `Client`, la clé primaire est l'attribut `NCLI`.
Dans le cas de `DETAILS`, aucun des attributs n'identifie une ligne de manière unique. On peut donc par exemple rajouter un attribut (souvent noté `Id`) qui sera la clé primaire

Id	NCOM	REF	QUANTITE
1	100	01-100	1
2	100	01-203	1

La clé primaire est quelquefois dénuée de signification (comme pour la table `DETAILS` ci-dessus), son seul rôle est d'identifier les enregistrements. Souvent, c'est un entier automatiquement incrémenté à chaque nouvelle entrée.

Définition :
On dit que la clé est ***étrangère*** dans une table si c'est un attribut de cette table prenant les mêmes valeurs que la clé primaire d'une autre table.

La clé étrangère va donc permettre de faire le lien entre plusieurs relations. Dans le cas de la table `DETAILS`, `REF` est une clé étrangère puisque c'est la clé primaire de la relation `PRODUIT`, sous le nom `NPRO`. La valeur de la clé étrangère dans une relation sert donc à référencer une ligne d'une autre relation (et à établir des relations entre les tables).

Les requêtes

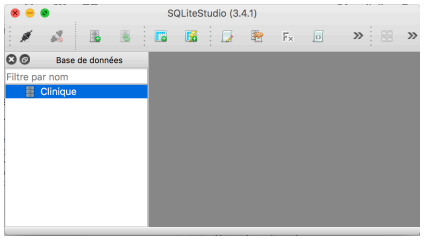
Utilisation d'un gestionnaire de bases de données

Définition :
Un *système de gestion de bases de données (SGBD)* est un logiciel qui organise et gère les données de façon transparente pour l'utilisateur, on va interagir avec elles par l'intermédiaire de *requêtes* exprimées dans le langage SQL (Structured Query Language).

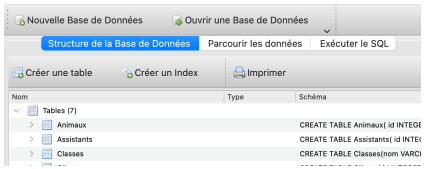
Si vous ne disposez pas sur votre ordinateur d'un tel logiciel, vous pouvez télécharger `DB Browser` sur le site https://sqlitebrowser.org/dl/ ou `SQLite Studio` sur le site https://sqlitestudio.pl. Nous utilisons ici à titre d'exemple `SQLite Studio` *(Le fonctionnement des deux logiciels est très similaire.)*

Afin d'illustrer nos exemples, téléchargez, sur le le site habituel bcpst2.f1.free.fr (partie "documents" associée au TP), la base de donnée `Clinique.db` *(accessible par clic direct)* et ouvrez là à l'aide de ce votre logiciel. Vous obtenez :

`Avec SQLite Studio :`

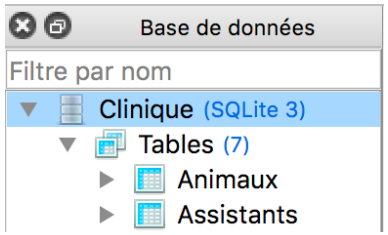


`Avec DB Browser :`



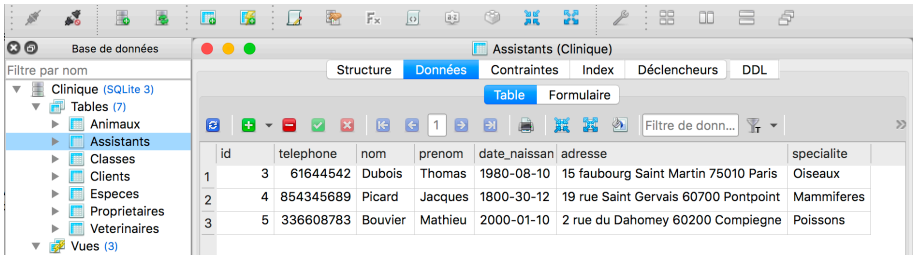
la structure (i.e. les différentes tables et autre qui la composent) de la table est visible directement.

Avec `SQLite`, si la structure ne s'affiche pas, dans le fenêtre de gauche, double cliquez ensuite sur le nom de la base de donnée (BDD) `"Clinique"`. Vous devez voir s'afficher la structure $\longrightarrow$ *(image tronquée)*

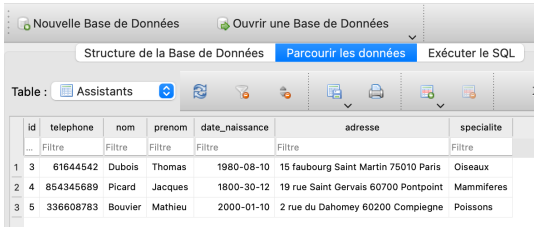


On pourra trouver les différents atributs de chaque table en cliquant sur la table concernée (par exemple `Assistants`). Pour observer les différens enregistrement, cliquer sur `''Données''` :

`Pour SQLite :`



`Pour DB Browser` : Il faut clliquer sur `''Données''` puis sélectionner la table souhaitée dans le menu déroulant :



On peut travailler sur cette base de données, ou récupérer des informations à l'aide de requêtes.
Celles-ci comportent trois instructions fondamentales :

- `CREATE` : créer une table
- `INSERT` : ajouter des données dans la table
- `SELECT` : interroger la table.

Au programme des BCPST ne se trouve que la dernière partie : interroger la table.

requête de base : la projection

L'ensemble des exemples de requêtes se font ici sur la BDD "Clinique"

> La forme générale d'une requête en `SQL` est :
>
> `SELECT ... FROM table;`

Toutes les requêtes commencent par `SELECT`, se terminent par un point-virgule. On peut écrire en majuscule ou minuscule, ceci n'a pas d'impact sur l'interprétation.
Quelques exemples sur la BDD *(que vous testerez un peu plus bas, patience...)* :

```
SELECT Nom FROM Clients;

SELECT * FROM Clients;

SELECT Nom, Adresse FROM Clients;
```

Les mots-clés `SELECT ... FROM` réalisent l'interrogation de la table. On a demandé ci-dessus respectivement :

- un attribut (`Nom`) de la table (`Clients`) désignée en le nommant,
- tous les attributs en les désignant par une étoile,
- plusieurs en les séparant par une virgule.

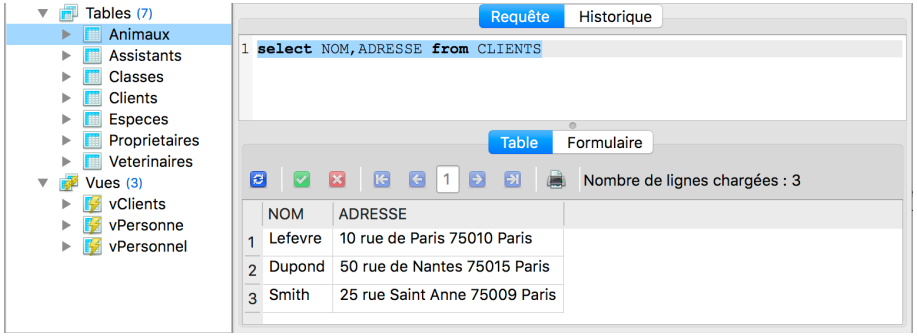
On qualifie cette première et dernière requête de **projection**, on dit qu'on projette la table `Clients` sur les colonnes `Nom` et `Adresse`.

Définition :
La *projection* dans le concept de bases de données consiste à extraire des colonnes d'une table.

Pour interroger la BDD,

`Avec SQLite :`

La première fois, il faut ouvrir la fenêtre d'édition de requêtes. Pour ce faire, repérerez l'icone dans le menu et cliquez dessus pour faire apparaître ladite fenêtre. Saisissez ensuite votre requête dans l'éditeur, (par exemple `SELECT Nom, Adresse FROM Clients;`) puis cliquez sur la flèche d'exécution . Regardez le résultat. Vous devez obtenir



Repérez que vous avez accès à cette fenêtre par un onglet en bas à droite *(Si vous réappuyez sur l'icone "nouvel éditeur SQL" ci-dessus, vous recréez une nouvelle fenêtre supplémentaire.)*

`Avec DBBrowser :`

Dans la barre des menus, cliquez sur `Exécutez le SQL`. Saisissez ensuite votre requête dans l'éditeur, (par exemple `SELECT Nom, Adresse FROM Clients;`) puis cliquez sur l'une des deux flèches d'exécution . Le résultat s'affiche dans la fenêtre du dessous.

Exercice 1: [Correction] Faire afficher la liste des enregistrements de la table `Animaux` avec uniquement les colonnes du nom de l'animal et de son espèce.

requête de base : la selection

Voici maintenant comment obtenir des données avec contrainte. Ici, avoir les données de l'ensemble des chiens soignés dans le cabinet

```
SELECT *  FROM Animaux WHERE Espece = 'Chien';
```

Le mot-clé `WHERE` filtre les données qui répondent à un critère de sélection.

Définition :
La *sélection* dans le concept de bases de données consiste à extraire des lignes d'une table.

Voici résumé les autres requêtes de base les plus classiques :

- `SELECT DISTINCT` : projection sans doublon
- `ORDER BY ... ASC / DESC` : trier les résultats par ordre croissant / décroissant
- `LIKE` : compare la condition à quelque chose qui est une chaîne de caractères dans laquelle on utilise les jokers `%` (pour remplacer un nombre quelconque de caractères) et `_` (pour remplacer un seul caractère).
- `LIMIT` : limiter à un nombre d'enregistrements que l'on va donner.
- `LIMIT a OFFSET b` : rend `a` enregistrements en commençant à `b+1` où `a` et `b` sont des nombres entiers.
- `UNION, INTERSECT, EXCEPT` : opérations ensemblistes sur les tables.

Quelques requêtes que vous êtes invités à essayer :

```
SELECT * FROM Animaux WHERE Espece = 'Chien' OR Espece = 'Moineau';

SELECT * FROM Animaux WHERE Nom LIKE '%i';

SELECT DISTINCT Espece FROM Animaux WHERE Dernierpoids>10 ORDER BY Espece ASC
LIMIT 2;

SELECT DISTINCT Espece FROM Animaux WHERE Dernierpoids>10   UNION   SELECT Espece
FROM Animaux WHERE Dernierpoids<5;
```

Remarque : L'utilisation de `UNION` pour deux tables identiques (comme dans l'exemple ci-dessus) n'est pas pertinente. Il serait plus lisble de remplacer par :

```
SELECT DISTINCT Espece FROM Animaux WHERE Dernierpoids>10 OR Dernierpoids<5;
```

Les commandes `UNION, INTERSECT, EXCEPT` n'ont véritablement d'intérêt que si on veut réunir deux tables **différentes**. Attention, nécessite toutefois d'avoir les mêmes noms d'attributs à sélectionner ! Par exemple, si on veut sélectionner l'ensemble des noms des vétérinaires et des animaux :

```
SELECT DISTINCT Nom FROM Animaux UNION SELECT DISTINCT Nom FROM Veterinaires;
```

Exercice 2: [Correction] Dans la table `Clients`, afficher la liste des noms, prénoms des clients nés dans les années 90. *(On remarquera que les dates de naissances sont écrites comme des textes commençant par l'année.)*

Jointures

On peut, dans une base de données, croiser des informations présentes dans plusieurs tables par l'intermédiaire d'une jointure (c'est d'ailleurs le grand intérêt d'une base de données). On peut joindre deux tables quand elles ont un attribut commun.

Par exemple, si on veut afficher la classe relative à tous les animaux de la table `Animaux`. Celle-ci ne figure pas directement la ladite table, mais figure dans la table `Especes`. Or, Les tables `Animaux` et `Especes` ont en commun le nom des espèces (Chien, moineau, etc...)
Cela s'écrit ainsi :

```
SELECT * FROM Animaux JOIN Especes ON Especes.Nom=Animaux.Espece;
```

(Vérifier qu'on obtient bien les 2 tables collées l'une à l'autre) On peut maintenant mixer avec des sélections et n'afficher que ceux qui sont des mammifères :

```
SELECT * FROM Animaux JOIN Especes ON Especes.Nom=Animaux.Espece
WHERE Especes.CLASSE='Mammiferes';
```

On constate que pour lever toute ambiguïté sur les noms d'attributs dans le cas où on considère deux tables, on utilise alors un préfixe par exemple pour `Especes.CLASSE`. On peut s'en passe si les colonnes concernées des deux tables ont des noms différents

```
SELECT * FROM Animaux JOIN Especes ON Especes.Nom=Animaux.Espece
WHERE CLASSE='Mammiferes';
```

On peut également joindre plus de tables en réitérant la demande autant de fois que nécessaire :

```
SELECT ... FROM ... JOIN ... ON ... JOIN ... ON ... ...;
```

Exercice 3: [Correction]

1. Pour chaque animal, faire afficher sur une même ligne : son identifiant dans la BDD, ainsi que le nom et l'adresse de son propriétaire. *On se servira de la table* `Proprietaires` *; et de la table* `Clients` *;)*
2. Faire maintenant afficher le nom de l'animal au lieu de son identifiant en joignant une troisième table `Animaux`.

Agrégation

Il est possible de regrouper certains enregistrements d'une table à l'aide du mot-clé `GROUP BY` (on les regroupe alors selon les valeurs d'un attribut) pour ensuite leur appliquer une fonction comme :

- `COUNT( )` : nombre d'enregistrements
- `MAX( )` : valeur maximale d'un attribut
- `MIN( )` : valeur minimale d'un attribut
- `SUM( )` : somme d'un attribut
- `AVG( )` : moyenne d'un attribut

Par exemple, pour connaître le nombre de chiens soignés dans la clinique :

```
SELECT COUNT(*) FROM Animaux WHERE Espece = 'Chien';
```

Pour connaître le nombre d'animaux par espèces soignés :

```
SELECT COUNT(*) FROM Animaux GROUP BY Espece;
```

`WHERE` permet alors d'imposer les conditions à l'intérieur d'un groupe. Ainsi, pour connaître, pour chaque espèce, le nombre d'animaux soignés de plus de 10kg :

```
SELECT COUNT(*) FROM Animaux WHERE DERNIERPOIDS>10 GROUP BY
Espece;
```

`HAVING` permet lui d'imposer d'imposer un "au moins". Ainsi, pour connaître le nombre total d'animaux par espèce contenant **au moins un animal** de plus de 10kg :

```
SELECT COUNT(*) FROM Animaux GROUP BY Espece HAVING
DERNIERPOIDS>10;
```

Sous-requêtes

On peut imbriquer une requête dans un filtre WHERE ou HAVING une sous-requête (simplement encadrée par une paire de parenthèses), le plus simple est de faire un exemple. Supposons que l'on souhaite déterminer les identifiants des clients possédant un chien soigné dans la clinique :

```
SELECT Client FROM Proprietaires WHERE Animal IN (SELECT ID FROM
Animaux WHERE Espece='Chien');
```

Exercice 4: [Correction] Afficher les informations complètes des propriétaires de chien (id, téléphone, etc...) obtenues dans la table `Clients`

Exercice 5: [Correction] On ne lâche rien jusqu'au bout ... !
On considère l'animal "Titi". Afficher, l'ensemble des vétérinaires auxquels son propriétaire peut avoir affaire (sachant que celui-ci peut avoir plusieurs animaux.)

Les exercices d'application

Exercice 6: [Correction]

Ouvrez la Base de données nommée `GeoFrance`. Constatez qu'elle est contituée de 3 tables et observez le contenu de ces tables.
Les communes répertorient des données sur les communes des département 1 à 27 (sauf 20) et on va appeler ceci ici "La France" même si on est conscient qu'il en manque un certain nombre !

Pour toutes les demandes ci-dessous, on attend une requête "en une seule ligne" c'est -à-dire du type `SELECT ... ;` *(un seul point-virgule pour toute la demande) qui affiche très exactement ce qui est attendu, ni plus, ni moins.*

1. Afficher le nombre d'habitants en France en 2010. *(Vous devez trouver* 12 127 403*.)*

2. Afficher toutes les informations des communes du département n° 1.

3. **a)** Déterminer les 100 villes situées les plus au Sud (i.e. les latitudes les plus petites).
(Vous devez obtenir : L'Hospitalet-près-l'Andorre, Mérens-les-Vals, Orlu etc...) Vérifier bien que vous avez 100 communes en faisant défiler votre résultat jusqu'en bas...

b) Afficher maintenant seulement le nom de la deuxième plus grande ville (en superficie) du département 12.
(*On trouve une ville dont le nom contient presque celui d'un continent...)*

4. **a)** Afficher toutes les informations des communes du département de l'Aube.

b) Donner la moyenne des superficies des villes dans le département de `'Cote-d'Or'`. *(Vous devez obtenir environ* 15*)*

c) Afficher le nombre d'habitant en 2012 de la région `Bretagne`

5. **a)** Afficher la liste des couples `(numéro du Département, Population du département en 2012)`, ordonnée par ordre croissant de numéros des départements.

b) Afficher maintenant la même chose, mais en triant dans l'ordre de la population totale décroissante.

6. Véritifier qu'il y a 6344 villes au sud de Dijon (i.e. latitude inférieure)

7. On appelle ici "densité des villes du département" la quantité $\frac{\text{nombre d'habitant en 2012}}{\text{surface du département}}$. Faire afficher pour chaque département : le numéro du departement, la densité des villes du département, tout ceci en triant les données obtenues par cette même quantité décroissante de densité.

FE 8 - Bases de données Solutions

Exercice 1

```
SELECT Nom,Espece FROM Animaux;
```

Exercice 2

```
SELECT Nom, Prenom From Clients WHERE DATE_NAISSANCE LIKE '__9%';
```

Exercice 3

1.

```
SELECT Proprietaires.Animal, Clients.Nom,Clients.adresse FROM
Clients JOIN Proprietaires ON Proprietaires.Client=Clients.Id;
```

2. Il faut joindre 2 tables : `Clients` et `Proprietaires` qui ont comme point commun

$$\texttt{Proprietaires.Client} \longleftrightarrow \texttt{Clients.Id}$$

```
SELECT Animaux.Nom, Clients.Nom,Clients.adresse FROM Clients JOIN
Proprietaires ON Proprietaires.Client=Clients.Id join Animaux on
Animaux.ID=Proprietaires.Animal;
```

Exercice 4

On peut faire plusieurs sous-requêtes :

```
SELECT* FROM Clients WHERE id in (SELECT Client FROM
Proprietaires WHERE Animal IN (SELECT ID FROM Animaux WHERE
Espece='Chien'));
```

Exercice 5

Le numéro de l'animal Titi est l'identifiant de la table `Animaux`, d'où la première requête :

```
SELECT Id FROM Animaux WHERE Nom='Titi';
```

On en déduit le (ou les, s'il y a plusieurs Titi) numéro(s) du propriétaire dans la table `Proprietaires` ;

```
SELECT Client FROM Proprietaires WHERE Animal IN (SELECT Id FROM
Animaux WHERE Nom='Titi');
```

On retrouve les autres animaux dans la même table des propriétaires :

```
SELECT Animal FROM Proprietaires WHERE Client IN (... Commande
précédente ...);
```

Pour chacun de ces animaux, on cherche son espece dans la table `Animaux` ;

```
SELECT Espece FROM Animaux WHERE Id IN (... Commande précédente
...);
```

Et finalement on affiche les vétérinaires spécialisés grâce à la table `VETERINAIRES` puis sa classe avec la table `Especes` ;

```
SELECT Classe FROM Especes WHERE Nom IN (... Commande précédente
...);
```

et pour finir, le vétérinaire associé :

```
SELECT DISTINCT Nom,Prenom FROM VETERINAIRES WHERE Specialite IN
(...Commande précédente...);
```

Exercice 6

1. C'est une simple requête de somme :

```
SELECT SUM(POP2010) FROM Communes;
```

2. C'est une simple requête avec restriction :

```
SELECT * FROM Communes WHERE Departement=1;
```

3. **a)** On combine un ordre avec une limite de nombre :

```
SELECT Nom FROM Communes ORDER BY Latitude ASC LIMIT 100;
```

b) Il faut ordonner en ordre décroissant et sélectionner seulement la deuxième occurence :

```
SELECT Nom FROM Communes WHERE Departement=12 ORDER BY Surface
DESC LIMIT 1 OFFSET 1;
```

4. **a)** ça se complique. Les informations sont dans deux tables différentes. Il faut donc les relier. Leur point commun est le numéro du département :

`Communes.Departement` $\leftrightarrow$ `Departements.Id`

```
SELECT * FROM Communes JOIN Departements ON
Communes.Departement=Departements.Id WHERE
Departements.Nom='Aube';
```

b) Le nom du département n'est pas disponible dans la table `Communes`. Il faut aller le chercher dans la table `Departements`, d'où

```
SELECT avg(Surface) FROM Communes JOIN Departements ON
Communes.departement=Departements.Id;
```

c) Il nous faut rejoindre cette fois-ci les trois tables `Communes, Departements` et `Regions` avec

`Communes.Departement` $\leftrightarrow$ `Departements.Id`

`Departements.Region` $\leftrightarrow$ `Regions.Id`

```
SELECT Sum(Pop2012) FROM Communes JOIN Departements ON
Communes.Departement=Departements.Id JOIN Regions ON
Departements.Region=Regions.Id WHERE Regions.nom='Bretagne';
```

5. **a)** Nous avons besoin ici de regrouper les sommes par département, d'où :

```
SELECT Departement,SUM(Pop2012) FROM Communes GROUP BY
Departement ORDER BY Departement;
```

b)

```
SELECT Departement,SUM(Pop2012) FROM Communes GROUP BY
Departement ORDER BY SUM(Pop2012) DESC;
```

6. Pour information, on trouve sur la table que la latitude de Dijon est 47.3167

```
 SELECT COUNT(Nom) FROM Communes WHERE Latitude<(SELECT Latitude
FROM Communes WHERE Nom='Dijon');
```

7.

```
 SELECT Departement,Sum(Pop2012)/Sum(Surface) From Communes GROUP
BY Departement ORDER BY Sum(Pop2012)/Sum(Surface) DESC;
```