Notebook 1 - Base de données - Autour d'une relation unique [1]

L'objet de ce Notebook est de découvrir les principes des bases de données lorsque l'on n'a qu'une seule relation [2]. On en profite pour introduire quasiment toutes les définitions et commandes du langage. Ainsi, le passage à plusieurs tables permettra de faire des révisions en les réutilisant.

Pour cela, on va utiliser Capytale, mais vous pouvez aussi utiliser un logiciel gratuit, comme par exemple `Sqliteman` pour Windows (disponible sur le site de la classe) ou `SQLiteBrowser` pour Mac, les fichiers de base de données nécessaires étant en ligne. Sous Capytale, ils sont directement stockés sur la page, et tout se passe comme sous Python : vous tapez la commande dans une cellule et vous l'exécutez, la base n'a pas à être ouverte car elle l'est directement. Le résultat s'affichera dans une nouvelle cellule, là aussi comme sous Python.

Au concours, les bases de données représentaient jusqu'à présent 15 à 25% des questions d'un sujet d'écrit, mais vous ne pouvez pas en avoir à l'oral. Enfin, aucun logiciel n'est au programme : ce que vous devez savoir faire, c'est « parler SQL », ce que vous saurez faire bientôt !

Les éléments de vocabulaire sont assez transparents, en dehors du mot **relation** sur lequel nous avons déjà insisté deux fois. Ils apparaissent en gras au fil du texte, et seront rassemblés dans la fiche de synthèse du cours. Le TD est assez long, mais plutôt tranquille. Bon courage ! [3]

I - Un peu de vocabulaire

On va utiliser la base de données `initiation.db`, constituée d'une seule **table** appelée **triangles**, dont le **schéma relationnel** (ensemble des données représentées et types (informatiques !) de celles-ci) est

```
triangles(idt:integer, ab:integer, bc: integer, ac:integer)
```

Chaque ligne (ou **enregistrement**) de la **table** `triangles` est constituée d'un ensemble d'**attributs** (ou **champs**) correspondant aux colonnes. Le **domaine** d'un attribut est le type de la variable qui remplit la colonne de l'attribut en question : entier, flottant, chaîne de caractères... Ici, `idt` est le numéro de référence d'un triangle ABC, `ab` est la longueur du segment AB, `ac` celle du segment AC et `bc` celle du segment BC, et leurs domaines sont à chaque fois le type entier. Ainsi, le **schéma relationnel** correspond donc à la donnée des paires attribut : domaine. Chaque enregistrement doit posséder un champs (ou un ensemble de champs) qui l'identifie de façon unique dans la table. Ce champ ou groupe de champs est nommé **clef primaire**, aussi parfois **clef** tout court : ici `idt`. Cette notion permet d'identifier de façon unique et univoque un enregistrement. Par exemple, le numéro de sécu peut jouer le rôle de clef primaire pour identifier de façon unique une personne en France [4]. Il peut y avoir plusieurs choix de clefs primaires possibles. On reviendra dans le cours sur cette notion essentielle. Dernière définition : une **requête** est une « question » que l'on pose grâce au langage SQL pour avoir une information sur une base de données.

♡ Le résultat d'une requête sur une table est lui-même une table. ♡

1. Code Capytale dae0-632147
2. On rappelle, pour l'avoir dit dans la fiche de présentation, que relation est synonyme de table, ou encore tableau.
3. Si le sujet vous passionne par la suite, ou que vous avez besoin d'une référence dans le futur dans votre formation, je vous conseille le très complet lien suivant, même s'il n'est pas forcément super lisible et qu'il faut parfois avoir déjà quelques notions pour bien comprendre : https ://sqlpro.developpez.com/
4. Ce qui est en toute rigueur faux : bébé, résident étranger, ... Mais vous voyez l'idée.

Ce tableau n'a pas nécessairement la même taille que le tableau de départ (c'est même extrêmement rare), jusqu'à n'avoir peut-être même qu'une case, comme par exemple lorsque l'on demande une valeur calculée sur le tableau (le nombre d'éléments de celui-ci par exemple).

> Il existe bien entendu de nombreux autres types accessibles aux données d'une table : booléen, dates, heures... Mais aucune n'est au programme. Ainsi, si on veut par exemple représenter des dates, on peut contourner le problème en les écrivant comme des chaînes de caractères, l'ordre lexicographique (celui du dictionnaire) permet alors d'obtenir le sens d'écoulement du temps.

II - Syntaxe de base des requêtes SQL : SELECT...FROM...WHERE...

La requête de base s'écrit de la façon suivante :

SELECT exp **FROM** tab **WHERE** cond

Cette requête permet d'obtenir la ou les expression(s) « exp » construite(s) à partir des entrées de la table « tab » vérifiant la condition « cond ». Attention ! L'ordre des mots-clefs est important, il ne peut pas être différent de celui-ci (et si vous changez cet ordre, autant vous dire que le correcteur met zéro et passe à la suite, sans lire l'idée générale de votre réponse). On va travailler de façon déductive et intuitive : devinettes, bon sens, etc... Chaque sous-partie permettra d'introduire progressivement de nouveaux mots-clefs, indiqués dans le titre. À vous !

1 Que vont réaliser les requêtes SQL suivantes ? Essayez de le deviner avant de le vérifier :

SELECT idt, ab **FROM** triangles

SELECT idt **FROM** triangles **WHERE** ab+bc+ac=100

SELECT *, ab+bc+ac **FROM** triangles **WHERE** ab+bc+ac>=100

SELECT ab+bc+ac **FROM** triangles **WHERE** ab*ab+bc*bc <> ac*ac [5]

En déduire le fonctionnement général d'un requête SQL et le sens des trois mots clefs. Seuls les deux premiers sont INDISPENSABLES à toute requête SQL, au sens strict du terme : on ne peut pas s'en passer. Dans la suite, on doit obtenir le résultat UNIQUEMENT avec des requêtes SQL (on ne compte pas à la main !). Notez aussi le rôle de l'astérisque pour obtenir tous les champs de la table sur lesquels on effectue la requête.

2 Déterminer la requête permettant d'obtenir les identifiants et les longueurs des côtés des triangles dont le périmètre est inférieur à 50.

On peut utiliser les opérateurs booléens classiques **AND**, **OR** et **NOT** dans le **WHERE**. Voyons cela.

3 Déterminer les enregistrements qui correspondent à des triangles tels que ab<20 et ac<10 et bc>15.

4 Déterminer la requête permettant d'afficher l'identifiant et le périmètre des triangles dont le côté ab est supérieur à 248 mais dont le côté bc est inférieur à 50.

5 Il existe un triangle diabolique dans la base : ce n'est en réalité pas un triangle, car il ne respecte pas l'inégalité triangulaire pour tous ses côtés. Quelle(s) relation(s) doivent être vérifiées sur ab, bc et ac pour s'assurer que le triplet (ab, bc, ac) correspond bien à un triangle géométrique ? Déterminer l'identifiant correspondant à l'intrus.

> On dit que `WHERE` permet d'effectuer une **sélection** des données, on parle aussi de **filtrage**, tandis que `SELECT` permet d'effectuer une **projection** des grandeurs `exp` demandées.

5. Notez ici le symbole particulier `<>` qui signifie « différent de », et donc qui n'est pas le symbole Python `!=`.

III - Opérateurs ensemblistes : UNION, INTERSECT et EXCEPT

Les opérateurs **UNION**, **INTERSECT** et **EXCEPT** permettent d'obtenir l'union, l'intersection ou la différence au sens ensembliste entre des tables obtenues par des requêtes SQL. Il est nécessaire que les mêmes champs soient retournés dans les différentes requêtes SQL. On va voir que la plupart du temps, on peut faire la même chose avec des requêtes utilisant `WHERE`.

6 Que renvoient les deux requêtes suivantes ?

```
SELECT * FROM triangles WHERE ab > 100 AND 170 > ab
```

et

```
SELECT * FROM triangles WHERE ab*ab + ac*ac = bc*bc
```

7 Devinez et vérifiez en tapant ce que fait la requête suivante (bon, c'est assez évident vous allez voir...). Notez que les requêtes sont écrites sur trois lignes pour être lisibles, mais vous pouvez tout à fait tout taper à la suite.

```
SELECT * FROM triangles WHERE ab > 100 AND 170 > ab
UNION
SELECT * FROM triangles WHERE ab*ab + ac*ac = bc*bc
```

Comment aurait-on pu obtenir la même chose avec une requête « classique » ?

8 Même question pour la requête

```
SELECT * FROM triangles WHERE ab > 100 AND 170 > ab
INTERSECT
SELECT * FROM triangles WHERE ab*ab + ac*ac = bc*bc
```

Était-il possible d'obtenir ce résultat avec une requête plus classique ? Il faut remarquer queque chose d'important : si un enregistrement fait partie des résultats des deux requêtes, il ne sera pas indiqué deux fois : l'opérateur UNION supprime les doublons !

9 Même question, qui est aussi l'occasion de deviner le but du nouveau mot-clef **BETWEEN**[6] :

```
SELECT * FROM triangles WHERE ab BETWEEN 100 AND 170
EXCEPT
SELECT * FROM triangles WHERE ab*ab + ac*ac = bc*bc
```

Pouvez-vous retrouver le résultat de la requête utilisant le mot-clef **UNION** avec une requête utilisant le mot-clef **EXCEPT** ?

On remarque qu'il n'y a pas unicité de la façon d'écrire une requête en SLQ. Il est intéressant de savoir culturellement que le logiciel qui interprète la requête que vous tapez va lui-même l'optimiser avant d'effectuer la recherche, ce qui fait que toutes les requêtes sont équivalentes en termes d'efficacité ! Cela est très différent d'un langage de programmation pour lequel les performances dépendent directement de ce que vous taperez comme programme.

6. Attention, ce mot-clef n'est pas au programme mais il peut tomber si on vous le donne, et il est tellement facile de comprendre ce qu'il fait...

IV - Fonctions dites d'agrégation : MIN, MAX, SUM, AVG, COUNT

Les fonctions d'agrégation permettent d'obtenir un résultat construit à partir d'un ensemble d'enregistrements résultants d'une sélection effectuée avec `WHERE`. Ces mots s'utilisent toujours après **SELECT** et avant **FROM** : ainsi, la fonction F s'utilise selon

SELECT F(*arguments***) FROM** *table*

Par exemple : **SELECT COUNT**(idt) **FROM** triangles, **SELECT MAX**(ab) **FROM** triangles, ... On vous laisse deviner à quoi ils servent, rien qu'à leurs noms. **Attention !** Il y a un piège très classique avec les agrégats, sur lequel on reviendra dans la partie sur les requêtes imbriquées.

10 Combien d'enregistrements sont renvoyés par la requête **SELECT** * **FROM** triangles **WHERE** ab+ac+bc>950 ? Et d'ailleurs, que renvoie « en français » cette requête ?

11 À l'aide de de la fonction **COUNT**, retrouver directement ce résultat

12 Déterminer la plus petite valeur des produits ab*ac*bc pour les périmètres supérieurs ou égaux à 100.

13 Déterminer le nombre de triangles équilatéraux, puis le périmètre maximal, minimal et moyen de ces triangles (avec quatre requêtes, pas une seule !).

14 Donner la somme des identifiants des triangles équilatéraux (cela n'a bien entendu aucun intérêt en pratique).

> Les fonctions d'agrégations permettent, avec `WHERE`, d'effectuer une opération sur tous les enregistrements résultants d'une requête.

On va voir un peu plus loin comment effectuer ce type d'opérations mais sur des **regroupements** d'enregistrements ayant une propriété commune, grâce aux commandes `GROUP BY` et `HAVING`.

V - Mots-clés ORDER BY, LIMIT, OFFSET

Ces mots-clefs s'utilisent en toute fin de requête. Notons que les deux derniers ne sont pas universels et dépendents du « dialecte » SQL utilisé. Il existe même parfois des sous-syntaxes spécifiques... Bon, on fera comme si le programme était bien écrit sur ce sujet et on retiendra ce que nous vous mettons ci-dessous comme syntaxes.

15 En utilisant le mot-clé **ORDER BY**, renvoyer les enregistrements correspondant aux triangles équilatéraux classés par ordre de périmètres croissants (avec **ASC**) puis par ordre de périmètres décroissants (avec **DESC**). La syntaxe est du type

```
ORDER BY prop ASC/DESC
```

16 En utilisant les mots-clés **ORDER BY**, **LIMIT** et **OFFSET** déterminer les enregistrements correspondant aux triangles équilatéraux classés du $3^{\text{ème}}$au $7^{\text{ème}}$par ordre de périmètres croissants. Valider le résultat (à l'œil) à l'aide de la question précédente. Exemple de syntaxe :

SELECT * FROM table **LIMIT** 10

donnera les dix premiers enregistrements. Si on ajoute **OFFSET 5** à la fin, on aura 10 résultats mais en partant du $6^{\text{ème}}$, la numérotation interne partant de 0.

VI - GROUP BY et HAVING

Remarque : désolé par avance pour le ridicule inintérêt des questions à venir, on se rattrapera dans le prochain TD.

La commande **GROUP BY**, qui se place après un éventuel **WHERE**, permet de regrouper des d'enregistrements ayant une propriété commune par groupes, et d'effectuer des opérations sur ces groupes. On utilise donc une fonction d'agrégation qui va s'appliquer à chacun des regroupements formés.

17 Que renvoie la requête suivante ?

```
SELECT count(*), ac+bc+ab FROM triangles GROUP BY ab+ac+bc ORDER BY ac+bc+ab
```

18 Que renvoie la requête suivante ?

```
SELECT max(ab), ab+ac+bc FROM triangles GROUP BY ab+ac+bc
```

19 À l'aide de la commande `GROUP BY`, écrire une requête qui permet d'obtenir le périmètre moyens des triangles de même valeur de longueur AB et pour lesquels cette longueur AB est inférieure où égale à 10.

La commande **HAVING** permet éventuellement de préciser une condition s'appliquant aux regroupements obtenus avec **GROUP BY**. Cette commande se place donc logiquement à la suite de l'attribut du **GROUP BY** et portera sur le mesure calculée par la fonction d'agrégation. Autrement dit, c'est une opération de filtrage sur les valeurs des fonctions d'agrégats, là où `WHERE` l'était pour les enregistrements individuels.

20 Que renvoie la requête suivante ?

```
SELECT count(*), ac+bc+ab FROM triangles GROUP BY ab+ac+bc HAVING count(*) >280
```

21 Proposer une requête affichant, pour chaque valeur possible du produit des longueurs des côtés, la valeur minimale du côté AB, dans le cas où cette dernière vaut au moins 50.

VII - Les requêtes imbriquées

Cette notion consiste à utiliser le résultat d'une requête, souvent contenant une fonction d'agrégation, comme valeur utilisée dans une autre requête. On peut souvent faire sans, mais c'est une idée qui simplifie énormément la vie parfois. L'idée plus précise est la suivante : le résultat de toute requête est elle-même une table, éventuellement d'une ligne et une colonne. Ainsi, on pourrait imaginer dans une seconde requête utiliser à côté du **FROM** ou du **WHERE** non pas une table existante, mais plutôt la table obtenue comme résultat de la requête prédécente ! Pour changer, un petit exemple.

22 Que renvoie la requête suivante ?

```
SELECT MIN(ab+bc+ac) FROM triangles
```

En déduire le résultat de la **requête imbriquée** suivante :

```
SELECT * FROM triangles WHERE ab+ac+bc = (SELECT MIN(ab+bc+ac) FROM triangles)
```

Attention toutefois, la syntaxe qui consisterait à écrire directement

```
SELECT idt FROM triangles WHERE ab+ac+bc = MIN(ab+bc+ac)
```

est fausse et ne marcherait pas.

> On ne peut pas mettre une fonction d'agrégat juste à côté d'un `WHERE`, il faut énoncer une requête.

Nous allons désormais voir le fameux piège évoqué auparavant lors de la définition des agrégats.

23 Que renvoie selon vous la requête suivante :

```
SELECT *, min(ab+bc+ac) FROM triangles WHERE ab+ac+bc = (SELECT MIN(ab+bc+ac) FROM triangles)
```

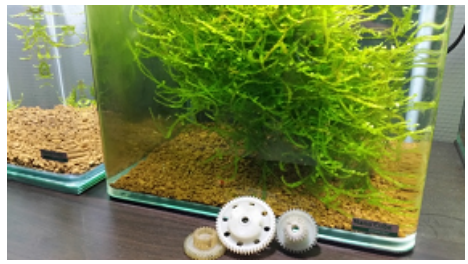
Est-ce correct, d'après la question précédente ? Si non, essayez d'expliquer pourquoi.

VIII - Un ancien DS machine - Les petits poissons dans l'eau

Le code Capytale pour cet exercice est différent, puisqu'il faut utiliser une autre base de données : 1266-635198.

Remarque préliminaire : les questions ne sont pas par ordre de difficulté, donc si vous bloquez à l'une, passez à la suivante, les questions sont relativement indépendantes.

Votre professeur préféré (ainsi vous pouvez choisir entre celui de SI ou bien celui d'informatique, sachant que l'un ne peut être l'autre) souhaite peupler un aquarium, comme sur la photo ci-dessous qui comporte un indice sur son propriétaire (vous constaterez que les poissons sont morts). Pour faciliter son choix, il a à sa disposition une base de données contenant la liste des espèces maintenues habituellement en captivité (les tailles sont en cm). La base de données qui vous est fournie se nomme `liste_poissons.db`.



Cette base constituée d'une seule table dont le schéma relationnel est

```
poissons(poisson_Id : integer, NomScientifique : text,
Famille : text, Descripteur : text, EsperanceVie : text,
ZoneDeVie : text, Origine : text, PHMin : real, PHMax
: real, GHMin : real, GHMax : real, TailleMale : real,
TailleFemelle : real, TempMin : real, TempMax : real)
```

1 Combien y a-t-il de poissons dans cette base ?

2 Comment se nomme le poisson ayant l'identifiant 458 (nom scientifique du poisson rouge).

3 Combien y a-t-il de poissons dans la famille du poisson trouvé à la question précédente ?

4 Quel poisson des fonds vit au Canada ?

5 Combien y a-t-il de poissons pour lesquelles la femelle est plus grande (strictement) que le mâle et pouvant survivre à une température de plus de 31°C (strictement) ?

6 Parmi les résultats de la question précédente, donner le nom du poisson ayant la plus faible espérance de vie.

Revenons à l'aquarium de votre professeur. L'eau de conduite en région parisienne possède un pH compris entre 6 à 8 et une dureté GH comprise entre 9 et 17. L'aquarium est chauffé entre 24°C et 26°C. On précise que les valeurs limites de ces critères peuvent être atteintes.

7 Combien de poissons peuvent être choisis selon les critères précédents ?

8 L'aquarium choisi étant de petite taille, on se limite à des poissons de taille inférieure à 10 cm parmi ceux trouvés à la question précédente. Donner le nom des personnes ayant découvert en 1970 des espèces pouvant convenir.