TP 23
Langage SQL partie 2

Merci à Guillaume ROUX (partie III) et à Quentin FORTIER (partie IV) pour le matériel de ce TP.

I Structure des bases de données

Une base de données est en général constituée de **plusieurs tables**, faisant référence les unes aux autres. Pour extraire des informations de la base, il sera alors nécessaire de **croiser** les données à travers plusieurs tables.

I.1 Exemples

Retour sur le TP précédent Lors du TP précédent, notre base de données était constituée d'au moins deux tables, l'une contenant une liste d'aéroports, l'autre une liste de pays. Un extrait simplifié de la table ressemble à ceci :

aéroports			
IATA	nom	ville	ISO
CDG	Roissy-Charles-de-Gaulle	Paris	FR
ORY	Orly	Paris	FR
AMS	Schiphol	Amsterdam	NL
MRS	Marseille-Provence	Marseille	FR
DUS	Düsseldorf	Düsseldorf	DE
MUC	Franz-Josef-Strauß	Munich	DE

pays	
ISO	nom
FR	France
NL	Pays-Bas
DE	Allemagne

On peut constater que pour obtenir à la fois la donnée d'un aéroport et de son pays, il est nécessaire de croiser les deux tables, en les recoupant selon la valeur du code ISO du pays. Par exemple, on lit dans la première table que l'aéroport Roissy-Charles-de-Gaulle se trouve dans le pays FR, puis en lisant la seconde que ceci correspond bien à la France.

Un nouvel exemple Un réseau social a des utilisateurs, qui peuvent poster des messages (avec des photos) et les commenter. Pour organiser tout cela, il faut au moins trois tables :

- Une première table `utilisateurs` où chaque nouvel utilisateur est une entrée. Elle contient donc des informations de l'utilisateur, son nom, son mot de passe pour se connecter.
- Une autre table `publications` servira à conserver les photos publiées et leur légende. Chaque fois qu'un utilisateur met en ligne une photo, elle est stockée et sa légende est ajoutée à la table.
- Enfin, chaque commentaire sous une publication créera une entrée dans une table `commentaires`, qui devra indiquer à quelle publication le commentaire se réfère et par qui il a été posté.

On peut penser que la base ressemble à ceci :

utilisateurs		
nom	motdepasse	ville
lclefevre	arctangente	Versailles
élève1	azerty	Versailles
élève2	jaimelabio	Versailles
cedricgrolet	croissant	Paris

publications			
numéro	date	auteur	contenu
1298445	15/03/2026 18:45	lclefevre	Une photo de chat
1298446	17/03/2026 11:32	élève1	Soirée d'hier
1298447	18/03/2026 08:56	cedricgrolet	Un pain au chocolat

commentaires			
date	sur	par	commentaire
15/03/2026 20:12	1298445	élève1	trop mignon !!!
17/03/2026 12:02	1298446	élève2	c'était trop bien !!
18/03/2026 16:15	1298447	lclefevre	miam miam
18/03/2026 17:01	1298445	élève2	vraiment trop mignon
19/03/2026 15:17	1298447	élève2	*chocolatine
19/03/2026 22:35	1298445	élève2	Il s'appelle comment ?

Croiser les trois tables permet par exemple d'affirmer « le 18/03/2026, élève2 a commenté la photo de chat de lclefevre postée le 15/03/2026 ».

I.2 Clés primaires et secondaires

Pour bien étudier le phénomène de croisement des tables, nous avons besoin de ces notions.

Définition

Dans une table, une **clé primaire** est un attribut (une colonne) qui identifie uniquement chaque entrée.

Quelques exemples :

- Dans la liste des élèves de la classe, le prénom seul ne forme en général pas une clé primaire : il peut y avoir plusieurs élèves avec le même prénom et donc on ne peut pas les distinguer uniquement ainsi. Par contre la donnée entière prénom + nom de famille est bien une clé primaire.
- À l'échelle de tout le pays, la donnée du prénom et du nom de famille n'est pas une clé primaire : il arrive que plusieurs personnes aient à la fois le même prénom et le même nom de famille, et donc cela n'identifie pas uniquement les personnes. Pour remédier à cela, nous avons tous de nombreux numéros d'identification, qui agissent comme des clés primaires pour distinguer uniquement les personnes : numéro de sécurité sociale, numéro de carte d'identité ou de passeport, mais aussi des numéros de clients chez de très nombreux services (client téléphone, client électricité, numéro de déclarant aux impôts, etc).
- Dans la base de données des aéroports, le code IATA à trois lettres (CDG pour Roissy-Charles-de-Gaulle, ORY pour Orly) est une clé primaire pour les aéroports commerciaux, qui identifie uniquement chaque aéroport du monde. Il est utile d'avoir ce code indiqué sur le billet et sur le bagage. Le code ISO des pays est, lui, une clé primaire pour identifier rapidement tous les pays du monde, indépendamment de la langue.

En fait, il est toujours possible de créer artificiellement une clé primaire, simplement en définissant un **numéro d'identification** (communément appelé ID), qui augmente de 1 automatiquement à chaque entrée, et le logiciel qui gère la base de données sait très bien le faire.

Définition

Dans une table d'une base de données, une **clé étrangère** (ou : **secondaire**) est un attribut dont les valeurs font référence à une clé primaire dans une autre table.

Exemples :

- Dans la base de données des aéroports, la colonne `ISO` de la table `aéroports` est une clé étrangère : ses valeurs sont à trouver dans la table `pays` où ce sont des clés primaires.
- Dans l'exemple de réseau social ci-dessus, la table `publications` contient la clé étrangère `auteur`, qui fait référence à la liste des utilisateurs. Pour savoir de qui provient la publication, il est nécessaire de croiser le contenu de cette colonne avec la table `utilisateurs`. Dans ce même exemple, la table `commentaires` contient *deux* clés étrangères : `par` qui fait référence à l'auteur du commentaire, qu'on va trouver dans la table `utilisateurs`, et `sur` qui fait référence au numéro de la publication qui est commentée, et qu'on va trouver dans `publications`.

L'intérêt de savoir qu'une colonne est une clé étrangère, et de savoir à quelle autre table elle fait référence, c'est de garantir que toutes les données de la base sont bien cohérentes. Supposons qu'on rentre des aéroports à la main et qu'on commette une erreur, par exemple on indique un aéroport avec le pays FS au lieu de FR, qui n'existe pas. Alors on ne peut plus répondre à la question « dans quel pays se situe cet aéroport », cela provoque une erreur. Et rapidement les requêtes SQL plus compliquées n'ont plus de sens et bloquent (par exemple on ne peut pas du tout répondre à « combien d'aéroports par pays » car cet aéroport n'est attribué à aucun pays). On parle de **corruption** des données. Au contraire le logiciel de gestion de base de données, si on lui déclare où sont les clés primaires et étrangères, sait vérifier au fur et à mesure des ajouts et suppressions dans la base que les données restent bien cohérentes et donc peut **garantir l'intégrité des données**.

Ajoutons aussi que l'intérêt d'utiliser pour les clés primaires des nombres entiers ou des codes à deux ou trois lettres, c'est de pouvoir bien plus rapidement comparer et trier selon ces valeurs plutôt qu'en comparant des chaines de caractères en entier (ce qui est toujours une opération lourde), c.f. l'annexe du TP sur les dictionnaires.

I.3 Jointure

Connaissant les clés primaires et étrangères, on peut enfin croiser intelligemment les tables.

Définition

L'opération de **jointure** entre deux tables consiste à combiner toutes les colonnes de l'une avec toutes les colonnes de l'autre, le long d'un attribut en commun.

Dans l'exemple des aéroports, la jointure des tables `aéroports` et `pays` selon l'attribut `ISO` donne la table suivante :

IATA	nom	ville	ISO	pays
CDG	Roissy-Charles-de-Gaulle	Paris	FR	France
ORY	Orly	Paris	FR	France
AMS	Schiphol	Amsterdam	NL	Pays-Bas
MRS	Marseille-Provence	Marseille	FR	France
DUS	Düsseldorf	Düsseldorf	DE	Allemagne
MUC	Franz-Josef-Strauß	Munich	DE	Allemagne

Dans le deuxième exemple, la jointure des tables `publications` et `commentaires` le long des numéros de publications (qui s'appelle `numéro` dans la table des publications et `sur` dans la table des commentaires) correspond à mettre côte à côte les publications et leurs commentaires ; si ensuite on ne garde que la publication numéro 1298445 on obtient bien

numéro	date	auteur	contenu	date	par	commentaire
1298445	15/03/2025 18:45	lclefevre	Une photo de chat	15/03/2025 20:12	élève1	trop mignon !!!
1298445	15/03/2025 18:45	lclefevre	Une photo de chat	18/03/2025 17:01	élève2	vraiment trop mignon
1298445	15/03/2025 18:45	lclefevre	Une photo de chat	19/03/2025 22:35	élève2	Il s'appelle comment ?

Remarquez qu'ici le mot `date` apparait deux fois avec deux significations différentes ; en fait il faudra utiliser la syntaxe `publications.date` pour le premier et `commentaires.date` pour le second. On pourrait aussi joindre trois tables, pour obtenir à la fois des informations sur les publications, leur auteur, et leurs commentaires.

II Rappels de SQL

On rappelle brièvement la forme générale des requêtes SQL qu'on utilise, voir le TP précédent :

```
SELECT attributs
FROM table
WHERE condition
ORDER BY critère ASC | DESC
```

où

- `attributs` est un attribut de la table, ou plusieurs séparés par des virgules, ou le caractère `*` ; on peut renommer les attributs avec `AS`, éventuellement `SELECT DISTINCT` pour avoir une seule fois chaque entrée,
- `table` est le nom de la table dans laquelle on sélectionne,
- `condition` est une condition exprimée sur les attributs, utilisant les opérations mathématiques `+`, `-`, `*`, `/`, les comparaisons `>`, `<`, `>=`, `<=`, `=`, `<>`, les mots-clés `AND`, `OR`, `NOT`,
- `critère` est l'attribut selon lequel ordonner, éventuellement par ordre croissant ou décroissant.

On rappelle aussi les fonctions d'agrégation `MIN()`, `MAX()`, `SUM()`, `COUNT()`, `AVG()` qui s'utilisent *dans* `SELECT` et parfois utilisées en regroupant d'abord des valeurs entres elles selon un autre attribut, typiquement :

```
SELECT fonction(attribut1) AS résultat, attribut2
FROM table
WHERE condition /* condition sur la recherche, avant de faire un calcul */
GROUP BY attribut2
HAVING condition /* condition sur le résultat du calcul */
ORDER BY critère ASC | DESC
```

Nous y ajoutons maintenant la syntaxe générale d'une jointure (voir sur les exemples), qui prend la forme

```
SELECT attributs
FROM table1
JOIN table2
ON critère
```

où `critère` désigne en général l'égalité entre un des attributs de `table1` et un attribut de `table2`.

Il arrive que les deux tables possèdent un même nom d'attribut et alors il faut utiliser la syntaxe `table.attribut`, par exemple dans notre mini réseau social les tables `publications` et `commentaires` contiennent toutes les deux un attribut `date`, qu'on appelle donc `publications.date` et `commentaires.date`, le critère sera donc de les joindre selon `publications.date=commentaires.date`. Éventuellement on utilise `AS` pour les renommer.

À combiner avec toutes les autres commandes SQL vues jusque là, appliquées sur les attributs d'une table ou de l'autre !

III Base de données : observations de mammifères

La base de données `mammiferes.db` contient des informations sur tous les mammifères observés en région parisienne. Il y a deux tables, dans l'une on range toutes les espèces de mammifères, avec diverses informations sur l'espèce, et dans l'autre on rajoute une nouvelle entrée à chaque fois qu'une espèce est observée avec la date et le lieu précis d'observation.

Ces données sont publiques, chacun peut y contribuer, le projet dépend du Museum National d'Histoire Naturelle : https://openobs.mnhn.fr/

Exercice 1
Observer la base de données. Quelles sont les clés primaires et étrangères ici ?

Le premier exercice contient seulement des révisions et pas de jointures.

Exercice 2
Écrire des requêtes SQL pour :

1. Donner toutes les observations par ordre chronologique.
2. Donner tous les ordres d'animaux.
3. Donner la plus grande latitude à laquelle un animal a été observé.
4. Combien d'observations ont été effectuées avant 1980 ?
5. i. Donner le nombre d'observations par années.
   ii. … ordonné par nombre d'observations.
6. Donner, pour chaque identifiant d'animal, la latitude moyenne et la longitude moyenne auxquelles il a été observé.

On s'intéresse maintenant à la question de jointure. La syntaxe pour recoller les deux tables est la suivante, testez :

```
SELECT * FROM observations JOIN animaux ON animal=cdRef;
```

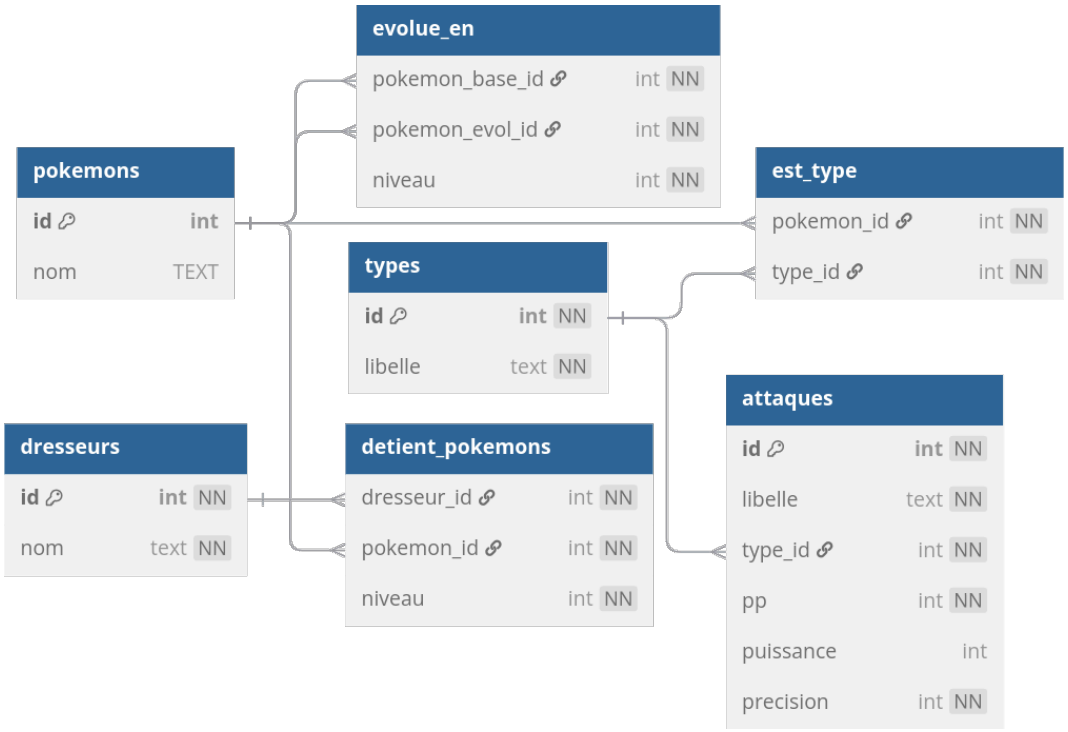
Exercice 3
Écrire une requête SQL pour :

1. Donner côte à côte les dates d'observations, noms vernaculaires, et noms scientifiques, des observations.
2. Donner le nom vernaculaire des animaux, classés par date d'observation la plus ancienne.
3. Donner pour chaque famille d'animaux, le nombre de fois où la famille a été observée.
4. Donner le nom de tous les animaux ayant été observés avant 1910.
5. Donner l'espèce du dernier animal ayant été observé.

IV Base de données : attrapez-les tous

La base de données `pokemon.db` contient la liste de tous les Pokémon, avec pour chacun leur type, leurs attaques, et leurs évolutions, ainsi qu'une liste de dresseurs pouvant posséder des Pokémon. La structure de la base est cette fois nettement plus compliquée : il faut bien entendu une table pour chaque Pokémon, mais aussi une table pour indiquer en quoi ils peuvent évoluer ; une table pour contenir les types et savoir quel Pokémon est de quel type ; de même, une table pour contenir les attaques et les relier à un type ; une table des dresseurs, et une autre table pour savoir quel dresseur possède quels Pokémon.

La structure de la base de données est représentée par le diagramme ci-dessous, où les clés primaires sont indiquées par une icône de clé. Les flèches d'une table vers l'autre correspondent à des clés étrangères, indiquées par une icône de cadenas : elles indiquent précisément que la clé est étrangère et vers quelle clé primaire d'une autre table elle pointe.



Exercice 4
Observer la base de données.

Le contenu est aussi disponible en ligne en mode interactif : https://sql-exercices.github.io/

Pour répondre aux questions, il faudra parfois joindre *plusieurs* tables. La syntaxe ressemble à :

```
SELECT attributs
FROM table1
JOIN table2
JOIN table3
ON critères
```

où `critères` devra être formulé avec des opérations `AND` (plusieurs colonnes à recoller).

Il faudra même utiliser des *auto-jointures* (joindre une table avec elle-même), mais il est alors nécessaire de renommer la table :

```
SELECT attributs FROM table AS table1
JOIN table AS table2
ON critères
```

Celles-ci permettent de mettre en relation une entrée de la table avec une *autre* entrée de la *même* table, éventuellement via d'autres tables intermédiaires.

Exercice 5
Écrire des requêtes SQL pour :

1. Donner le nombre total de Pokémon.
2. Afficher le nom de chaque Pokémon avec, à côté, le niveau auquel il évolue.
3. Afficher tous les Pokémons détenus par le dresseur Pierre.
4. Afficher le nom de chaque attaque, avec son type.
5. Afficher chaque dresseur avec chacun de ses Pokémon.
6. Afficher chaque Pokémon avec son niveau d'évolution et le nom de son évolution.
7. i. Afficher chaque dresseur avec son nombre de Pokémon.
   ii. … et le niveau moyen de ses Pokémon.
8. i. Afficher les dresseurs qui possèdent au moins 4 Pokémon.
   ii. … tous de niveau au moins 50.
   iii. … ordonné par le niveau du Pokémon le plus élevé qu'ils possèdent.