TP 22
Langage SQL partie 2

Merci à Guillaume ROUX pour le matériel de ce TP.

I Plus sur les 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.

Exemple 1. Lors du TP précédent, notre base de donnée était constituée d'au moins deux tables, l'une contenant une liste d'aéroports, l'autre une liste de pays. Un extrait un peu 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.

Exemple 2. Un réseau social a des utilisateurs, qui peuvent poster des photos et les commenter. Pour garder tout cela en mémoire, il faut au moins trois tables :

1. 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.
2. Une autre table `publications` servira à conserver les photos publiées. Chaque fois qu'un utilisateur met en ligne une photo, elle est ajoutée à la table.
3. Enfin, chaque commentaire sous une photo créera une entrée dans une table `commentaires` des commentaire, qui devra indiquer à quelle photo 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	15/03/2024 18:45	lclefevre	Une photo de chat
1298446	17/03/2024 11:32	élève1	Soirée d'hier
1298447	18/03/2024 08:56	cedricgrolet	Un pain au chocolat

commentaires			
date	**sur**	**par**	**commentaire**
15/03/2024 20:12	1298445	élève1	trop mignon !!!
17/03/2024 12:02	1298446	élève2	c'était trop bien !!
18/03/2024 16:15	1298447	lclefevre	miam miam
18/03/2024 17:01	1298445	élève2	vraiment trop mignon

Croiser les trois tables permet par exemple d'affirmer « le 18/03/2024, élève2 a commenté la photo de chat de lclefevre postée le 15/03/2024 ».

Définition 1. Dans une table, une **clé primaire** est un attribut (une colonne) qui est unique pour chaque entrée.

Exemple 3.
1. 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.
2. 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.
3. 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).
4. Dans la base de donnée 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.

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 2. Dans une table d'une base de donnée, une **clé étrangère** (ou : **secondaire**) est un attribut (une colonne) dont les valeurs sont des clés primaires dans une autre table.

Exemple 4.
1. Dans la base de donnée 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.
2. 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 données « 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 est cet aéroport » et rapidement les requêtes SQL plus compliquées n'ont plus de sens et bloquent ! Le logiciel de gestion de base de données, si on lui déclare où sont les clés étrangères, sait justement vérifier au fur et à mesure des ajouts et suppressions dans la base que les données restent bien cohérentes.
Et 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, cf. l'annexe du TP sur les dictionnaires, le mot-clé est recherche par dichotomie !

Définition 3. L'opération de **jointure** entre deux tables consiste à combiner toutes les colonnes de l'une avec toutes les colonnes de l'autre, lorsqu'elles ont un certain 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	nom
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 on ne garde que la publication numéro 1298445 on obtient bien

numéro	date	auteur	contenu	date	par	commentaire
1298445	15/03/2024 18:45	lclefevre	Une photo de chat	15/03/2024 20:12	élève1	trop mignon !!!
1298445	15/03/2024 18:45	lclefevre	Une photo de chat	18/03/2024 17:01	élève2	vraiment trop mignon

Remarquer 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 :

```
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 `DISTINCT 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 groupant des valeurs entres elles, typiquement :

```
SELECT FONCTION(attribut1), attribut2
FROM table
WHERE condition
GROUP BY attribut2
HAVING condition
ORDER BY critère ASC / DESC
```

où la condition dans `HAVING` est testée *après* le calcul de la fonction (on garde les données dont le résultat du calcul réalise la condition), alors que `WHERE` est appliqué dès le départ (on sélectionne d'abord les données sur lesquelles on va effectuer le calcul).
La syntaxe générale d'une jointure prend la forme (voir sur les exemples)

```
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`. Éventuellement on utilise `AS` pour les renommer. S'ensuivent toutes les autres commandes SQL vues jusque là, appliqués sur les attributs d'une table ou de l'autre !

III Cas 1 : observation des mammifères

La base de donnée `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.

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. (a) Donner le nombre d'observation par années.
   (b) ... 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 animaux JOIN observations ON cdRef=animal;
```

Exercice 3. Écrire une requête SQL pour :

1. Donner côte à côte les noms vernaculaires, noms scientifiques, et dates d'observations.
2. Donner le nom vernaculaire des animaux, classés par date d'observation la plus récente.
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 Cas 2 : suivi des cas de paludisme

La base de données `paludisme.db` contient des informations sur les cas de paludisme observés dans un certain nombre de pays concernés. Une table contient seulement les pays et leur population par année d'observation, et une autre table contient les nouveaux cas recensés et les décès, par année et par pays.

Exercice 4. Observer la base de données. Y a-t-il des clés primaires ici ?

Pour « recoller » les tables et voir apparaitre côte à côte les noms des pays, les cas de paludisme et leurs décès, et la population totale du pays, la bonne commande est :

```
SELECT * FROM palu JOIN demographie ON iso=pays AND annee=periode;
```

En effet, chaque entrée de la table `demographie` est identifiée uniquement par les deux attributs `pays` et `période`, qu'il faut donc recoller selon les attributs appelés `iso` et `annee` de la table `palu`.
À vous maintenant d'écrire les requêtes suivantes utilisant des jointures.

Exercice 5. Écrire des requêtes SQL pour :

1. Donner toutes les données de la table palu de l'année 2010 des pays où le nombre de décès dus au paludisme est supérieur ou égal à 1000.
2. (a) Donner le nombre de décès dus au paludisme dans le monde sur toute la période étudiée.
   (b) … groupé par année.
3. (a) Classer les pays et années, par nombre de décès.
   (b) … puis classer les pays par nombre de décès sur toute la période étudiée.
4. (a) (sans jointure) Le *taux de mortalité* d'une maladie est le rapport entre le nombre de cas et le nombre de décès de cette maladie. Donner pour chaque pays et chaque année le taux de mortalité, en pourcentage.
   (b) … en classant par taux de mortalité.
   (c) … en groupant toutes les années entre elles (cumuler les cas et les décès).
5. (a) (avec jointure) Le *taux de d'incidence* d'une maladie est le rapport entre le nombre de nouveaux cas chaque année et la population totale, qu'on exprimera cette fois en nombre de cas pour 100 000 personnes. Donner le taux de prévalence, par pays et par année, puis ordonné selon le taux de prévalence.
   (b) Cette fois-ci on ne peut pas grouper facilement les années entre elles (car la population varie !) À la place, donner la moyenne du taux de prévalence, sur les années d'observation, par pays.

V Pour aller plus loin

Quelques techniques avancées que nous n'avons pas évoquées :

1. On peut joindre plusieurs tables d'un coup :

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

   Mais pour l'appliquer il est nécessaire d'avoir une base avec plusieurs tables…
2. Il existe des *auto-jointures*, mais il est alors nécessaire de renommer la table

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

   Cela permet de croiser les entrées d'une table entre elles.
3. Il existe aussi des *sous-requêtes*. Le résultat d'une commande `SELECT` étant par définition de type table — le SQL que nous utilisons ne manipule que des tables et renvoie des tables — on peut l'utiliser à l'intérieur d'une autre commande, par exemple d'un `WHERE`. `SELECT`.

   ```
   SELECT ... FROM ... WHERE ... = (SELECT ...) ...
   ```

D'autres exercices, sur une base de données complète avec de nombreuses tables à croiser :
https://deptfod.cnam.fr/bd/tp/