TNSI	Bases de données	Cours – Interrogation d'une base de données
	SQL	

Objectifs :

- ⇨ Approfondir sa connaissance de SQL
- ⇨ Voir les principales notions pour écrire des requêtes SQL
- ⇨ Voir des exemples d'interrogation de bases de données

"Sorry, bub. You're not in the database."

I - Requêtes simples

Pour cette première partie, nous utiliserons une base assez simple puisqu'elle ne contient qu'une seule table : la base de données « atomes ». A vous de démarrer mysql, passer en UTF-8 (« `CHARSET utf8mb4 ;` ») créer une base vide (« `CREATE DATABASE atomes ;` ») et se placer dessus (« `USE atomes ;` »).
Importer ensuite l'unique table de la base en exécutant le fichier « atomes.sql » (utiliser pour cela l'instruction « `SOURCE nom_fichier ;` » de mysql.
Attention : si le fichier n'est pas dans le répertoire de mysql il faut préciser son chemin d'accès complet.

1) Projection

L'opération de projection consiste à sélectionner seulement certains attributs d'une relation pour ne lister que la projection de la table sur ces attributs.
On utilise pour cela la syntaxe :

```
SELECT arg1, arg2, … FROM nom_table;
```

Où `arg1`, `arg2` sont des arguments de la table `nom_table`. On peut également mettre `*` à la place des arguments ce qui signifie « tous les arguments » donc toute la table.

L'opération de projection va **créer une nouvelle table** (uniquement en mémoire) et c'est cette table qui sera affichée comme résultat de la requête.

Application 1 :

1) Afficher les attributs de la table. Repérer notamment ceux qui désignent le nom et le symbole des éléments.
2) Afficher le nom des éléments chimiques ainsi que leur symbole. Notez ci-dessous la commande SQL employée :

Alias :
Il peut être pratique pour l'affichage de renommer un attribut. Pour cela on utilise juste après le nom de l'attribut le mot-clé « AS » suivi du nom de substitution ou alias (les chaînes de caractère sont encadrées par des simples cotes en SQL – on peut les omettre s'il n'y a pas d'espaces dans le nom).
Exemple : `SELECT c_article AS 'Code article', desc AS Description FROM article ;`

L'utilisation des cotes est indispensable si on veut afficher un nom qui soit un mot-clé du langage SQL (comme DATE pour lequel on écrira par exemple « `decouverte_date AS 'Date'` »).

3) Afficher le nom et la température de fusion des éléments chimiques en utilisant comme titres de colonnes « Nom élément » et « Fusion (°C) ».
Notez ci-dessous la commande SQL employée :

2) Sélection (ou restriction)

La sélection ou restriction consiste à ne sélectionner que les enregistrements de la table qui répondent à un ou plusieurs critères. Cela permet de n'afficher que la partie de la table pertinente pour notre requête.
Pour cela on ajoute à la suite de la requête le mot-clé `WHERE` suivi de la condition à respecter.

Exemple :
Pour lister tous les véhicules de marque Peugeot de la table `vehicule`, on peut faire :

```
SELECT * FROM vehicule WHERE marque = 'Peugeot';
```

Les conditions s'écrivent de façon assez classique avec les opérateurs listés ci-contre.

Opérateur	Description
=	Égal
<> ou !=	Pas égal
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égale à
<=	Inférieur ou égale à
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donné (utile pour les nombres ou dates)
LIKE	Recherche en spécifiant un motif
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle
AND	ET logique entre deux conditions
OR	OU logique entre deux conditions

Précisions :
Pour l'opérateur `LIKE` :

- `'_'` correspond à un caractère quelconque et un seul
- `'%'` correspond à un nombre quelconque de caractères quelconques

Ex : « abrupt » et « écru » correspondent à `LIKE '__ru%'`
Mais pas « rupture » ou « cruel ».

`BETWEEN` s'utilise toujours avec `AND` et il est équivalent à deux conditions reliées par `AND` :
Ex : `nb_eleves BETWEEN 17 AND 35` est équivalent à `nb_eleves >= 17 AND nb_eleves <= 35`.

Application 2 :
1) Afficher les noms, symbole et année de découverte de tous les éléments découverts en France
Notez ci-dessous la commande SQL employée :

2) Nombre d'éléments chimique dont le symbole est constitué d'une seule lettre (utiliser la fonction `LENGTH()`).
Notez ci-dessous la commande SQL employée :

3) Compter le nombre de réponses

SQL possède des **fonctions d'agrégation** qui permettent d'effectuer certains calculs basiques sur les enregistrements sélectionnés. Parmi celles-ci on trouve la fonction `COUNT()` qui compte le nombre d'enregistrements.
On peut utiliser `COUNT(*)` pour compter simplement le nombre d'enregistrements sélectionnés par la clause `WHERE` ou `COUNT(nom_attribut)` pour compter le nombre d'enregistrement dont l'attribut spécifié n'est pas `NULL` parmi ceux sélectionnés.

Exemples :
Afficher le nombre de véhicule réparé au moins 2 fois :

```
SELECT COUNT(*) FROM vehicule WHERE nb_reparations >= 2;
```

Afficher le nombre de client ayant déjà commandé (dont la date de dernière commande n'est pas NULL) :
```
SELECT COUNT(date_derniere_commande) FROM client;
```

Application 3 :

1) Afficher le nombre d'éléments qui étaient connus avant l'invention du tableau périodique des éléments par Dimitri Mendeleïev en 1869.
   Notez ci-dessous la commande SQL employée :

2) Afficher le nombre d'éléments qui ont une électronégativité (attribut « `electronegativite` ») connue.
   Notez ci-dessous la commande SQL employée :

3) Sélectionner maintenant la base `atomes_null` dans laquelle les éléments qui n'ont pas de date de découverte définie ont leur date à `NULL`, puis exécuter la même requête qu'au 1). Obtient-on le même résultat ? Comment modifier la requête pour obtenir un résultat correct ?
   Notez ci-dessous la commande SQL employée :

4) Combien y a-t-il eu d'éléments découverts du vivant de Mendeleïev (1834-1907) après la publication de son tableau ?
   Notez ci-dessous la commande SQL employée :

Il existe d'autres fonctions d'agrégation que nous ne détaillerons pas mais qui peuvent être utiles :

Fonction	Description
`SUM(attr)`	Somme sur l'attribut *attr*
`AVG(attr)`	Moyenne de l'attribut *attr*
`MIN(attr)`	Valeur minimale de l'attribut *attr*
`MAX(attr)`	Valeur maximale de l'attribut *attr*

4) Trier les résultats

Pour trier les résultats d'une requête, on peut simplement spécifier :
`ORDER BY attribut1[,attribut2,…]` à la fin de la requête. Il est possible de préciser l'ordre ascendant en rajoutant ASC (valeur par défaut) ou descendant DESC après le nom de l'attribut.

Application 4 :

1) Afficher les noms des éléments chimiques et leur année et pays de découverte par ordre croissant de l'année de découverte.
   Notez ci-dessous la commande SQL employée :

2) Comment les valeurs NULL sont-elles traitées ?

5) Eliminer les doublons

Application 5 :

1) Afficher les pays (ou groupes de pays) ayant découverts des éléments chimiques avec la requête :

```
SELECT decouverte_pays AS Pays FROM atome WHERE decouverte_pays <> '';
```

Que remarque-t-on ?

Pour éviter le problème constaté, on peut utiliser le mot-clé `DISTINCT` juste après le `SELECT` (on peut également l'utiliser avec l'opérateur `COUNT()`). Cela a pour effet de supprimer les doublons dans le résultat de la requête (les lignes exactement identiques qui ont donc les mêmes valeurs pour tous les attributs sélectionnés).

2) Afficher la liste de tous les pays (ou groupes de pays) ayant découverts des éléments chimiques.
Notez ci-dessous la commande SQL employée :

6) Grouper et filtrer (*hors programme*)

On peut demander dans une requête à grouper les lignes par valeurs identiques d'un attribut tout en utilisant des fonctions d'agrégation sur ces groupes de lignes.
Pour cela on utilise le mot-clé `GROUP BY` *`nom_attribut`* à la fin de la requête (après le `WHERE` éventuel).

Application 6 :

1) Afficher la liste de tous les pays (ou groupes de pays) ayant découverts des éléments chimiques ainsi que le nombre d'éléments qu'ils ont découvert.
Notez ci-dessous la commande SQL employée :

2) Faire un tri dans les résultats et répondre à la question : quel est le pays qui a découvert le plus d'éléments chimiques différents au cours des siècles ?

On peut de même filtrer les résultats en utilisant une fonction d'agrégation à l'aide du mot-clé `HAVING` qui se comporte globalement comme un `WHERE` mais peut s'appliquer aux fonctions d'agrégation.

3) Afficher la liste des pays ayant découverts au moins 10 éléments.
Notez ci-dessous la commande SQL employée :

II - Requêtes portant sur plusieurs tables

1) Principe

Nous allons maintenant effectuer des requêtes qui requièrent le croisement d'informations sur plusieurs tables.

Avant de travailler sur une base réelle, voyons comment le SGBD combine plusieurs relations en prenant un exemple simple de trois tables élémentaires :

```
CREATE DATABASE couleurs;
USE couleurs;
CREATE TABLE fr (id INT UNSIGNED PRIMARY KEY, couleur VARCHAR(20));
INSERT INTO fr VALUES (1,'rouge'), (2,'bleu'), (3, 'jaune');
CREATE TABLE en (id INT UNSIGNED PRIMARY KEY, color VARCHAR(20));
INSERT INTO en VALUES (1,'red'), (2,'blue'), (3, 'yellow');
CREATE TABLE de (id INT UNSIGNED PRIMARY KEY, farbe VARCHAR(20));
INSERT INTO de VALUES (1,'rot'), (2,'blau'), (3, 'gelb');
```

Application 7 :

1) Qu'obtient-on si on demande l'affichage des deux premières tables avec « `SELECT * FROM fr, en;` » ? Est-ce le résultat attendu ?

2) Qu'obtiendra-t-on si on sélectionne les 3 tables ? Vérifier en écrivant la requête correspondante.

Quand on sélectionne plusieurs relations, le SGBD effectue le *produit cartésien* des deux tables. C'est comme s'il créait une nouvelle table contenant les attributs de toutes les tables et avec toutes les combinaisons de n-uplets possibles.
Ce n'est généralement pas le comportement souhaité et on veut généralement lister les enregistrements d'une table qui correspondent à ceux de l'autre table. Par exemple on veut lister tous les véhicules de la table `vehicule` avec leurs propriétaires de la table `client`. Si on fait le produit cartésien simple, on aura l'affichage de tous les véhicules pour chaque client. Il faudrait que chaque véhicule ne soit associé qu'à son propriétaire.

Nous avons créé des liens entre les relations avec les clés étrangères et c'est en exploitant ces clés que nous allons pouvoir réaliser une **jointure**. La jointure est le produit cartésien de deux tables en réduisant les résultats aux n-uplets dont les clés primaire et étrangères correspondent.

La **jointure** de deux tables est l'opération qui consiste à combiner tous les attributs d'une table avec tous les attributs d'une autre table en prenant tous les enregistrements pour lesquels il y a égalité entre un certain attribut d'une table et un autre attribut identique dans l'autre table (généralement la clé primaire d'une table et la clé étrangère de l'autre).

En SQL il suffit d'insérer une clause `WHERE` explicitant cette condition.
Ex : `SELECT * FROM fr,en,de WHERE fr.id = en.id AND en.id = de.id;` affichera la jointure des 3 tables.

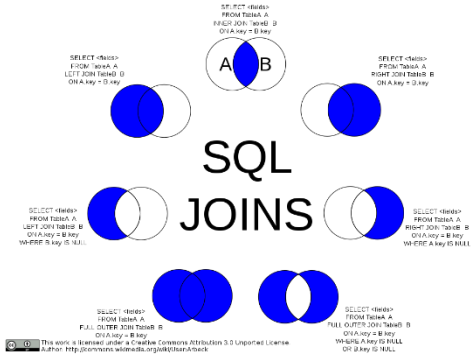
On remarque que pour spécifier un attribut d'une table dont le nom existe aussi dans une autre relation, il faut utiliser la syntaxe *`nom_de_la_table.nom_attribut`*. On peut utiliser des alias.

Il existe cependant une syntaxe plus lisible spécifique à la jointure avec le mot-clé `JOIN` :

```
SELECT fr.couleur, en.color, de.farbe
       FROM fr JOIN en ON fr.id=en.id JOIN de ON fr.id = de.id;
```

Si les attributs sur lesquels on fait la jointure ont le même nom dans les différentes tables, on peut également utiliser le mot-clé `NATURAL JOIN` :

```
SELECT * FROM fr NATURAL JOIN en NATURAL JOIN de;
```



Il existe de nombreuses variantes de jointure (ici on a présenté uniquement le INNER JOIN qui est le seul au programme de terminale). Voir le schéma ci-contre ou les explications sur les liens :
https://sql.sh/cours/jointures
https://www.ionos.fr/digitalguide/hebergement/aspects-techniques/sql-outer-join/

2) Applications

Voyons maintenant l'intérêt des jointures avec une base de données nettement plus complète : `datafilms` qui contient un extrait de l'IMDB. La structure de cette BD est la suivante :

Commencer par importer la base de données en utilisant le fichier « datafilms.sql ». Vérifiez qu'elle contient bien 106000 enregistrements dans `lesacteurs`, 4775 dans `lesfilms` et dans 4759 `lesrealisateurs`.

Application 8 :

1) Donner la liste de tous les acteurs ayant participé au film « Star wars ».
   Notez ci-dessous la commande SQL employée :

2) Donner la liste de tous les films d'Harrison Ford.
   Notez ci-dessous la commande SQL employée :

3) Afficher la liste alphabétique de tous les réalisateurs de la base de données. Combien y a-t-il de réalisateurs différents ?
   Notez ci-dessous la commande SQL employée :

4) Quels sont tous les acteurs ayant tourné avec le réalisateur Peter Jackson. Combien y en a-t-il ?
Notez ci-dessous la commande SQL employée :

5) Lister les films réalisés par Steven Spielberg. Combien y en a-t-il ? Même question avec tous les films réalisés par Steven Spielberg avec Harrison Ford.
Notez ci-dessous la commande SQL employée :

6) Donner tous les couples acteur, réalisateur ayant tourné sur plus de 4 films ensembles. Combien y en a-t-il ? A combien retombe ce nombre si on enlève les couples où l'acteur et le réalisateur sont le même (comme pour Woody Allen) ?
Notez ci-dessous les commandes SQL employées :

3) Requêtes imbriquées

On a vu précédemment que l'instruction `SELECT` renvoyait une table. Il est donc possible (et utile) d'utiliser le résultat d'une requête comme une table dans une autre requête. Dans ce cas, il suffit de mettre la requête entre parenthèses pour qu'elle soit utilisée comme une table par une autre requête.

Exemple :

```
SELECT attribut1, attribut2 FROM table ;   -- Requête simple
SELECT attribut1, attribut2 FROM (
       SELECT attribut1, attribut2, attribut3 FROM table1
             NATURAL JOIN table2 WHERE attribut1 > 10);  -- Requête imbriquée
```

Cette technique est souvent utilisée avec le mot-clé `IN` pour vérifier qu'un des attributs vérifie une condition complexe.

7) Quels sont les films où les acteurs Harrison Ford et Liam Neeson ont tournés tous les deux ?
Notez ci-dessous la commande SQL employée :

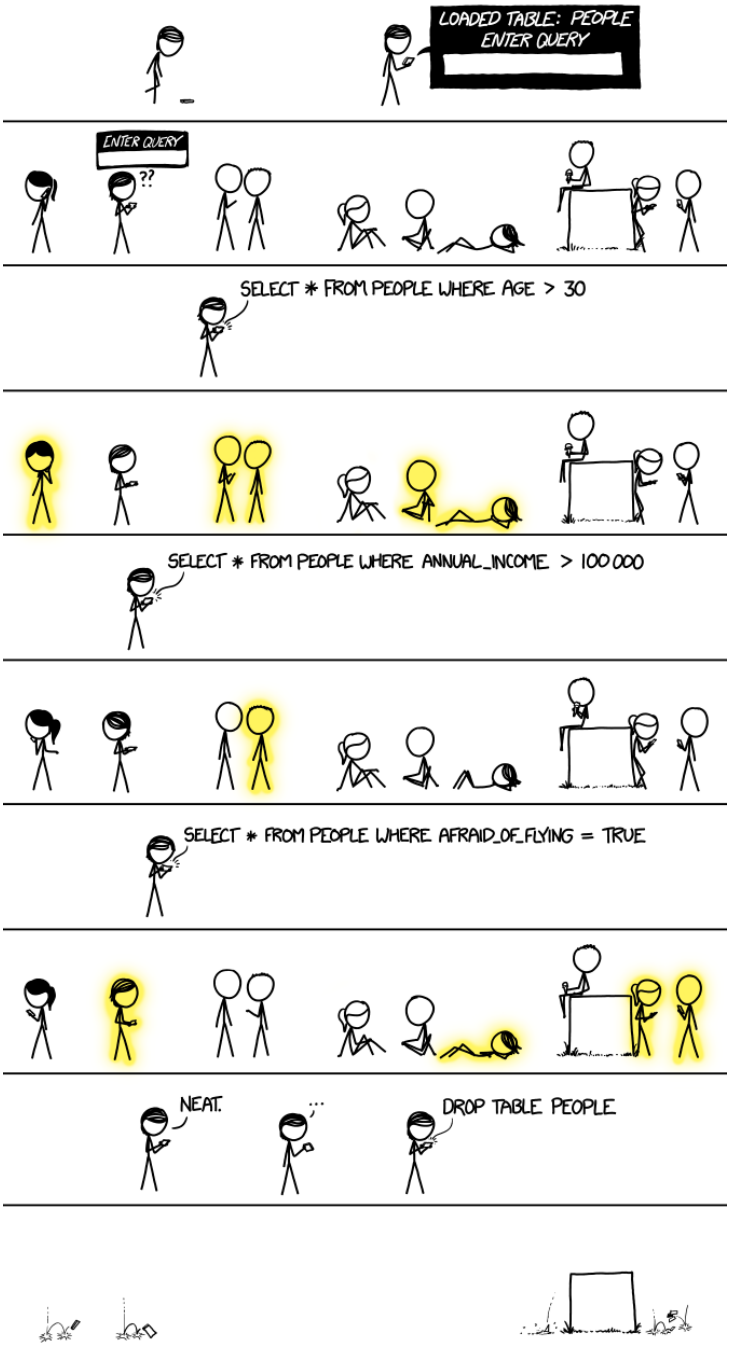
8) Lister les films réalisés par Martin Scorsese. Combien y en a-t-il ? Même question avec tous les films réalisés par Martin Scorsese sans l'acteur Robert De Niro.
Notez ci-dessous les commandes SQL employées :

Références :

Cours : https://sqlpro.developpez.com/cours/sqlaz/select/

Référence du langage SQL : https://sql.sh/ et https://www.w3bai.com/fr/sql/default.html

Exercices d'entraînement : https://www.w3schools.com/SQl/exercise.asp, https://colibri.unistra.fr/fr/course/list/notions-de-base-en-sql ou https://fxjollois.github.io/cours-sql/



Exercices – Interrogation d'une base de données

Exercice 1 : Festival de musique

Soit la base de données d'un festival de musique : Dans une représentation peut participer un ou plusieurs musiciens. Un musicien ne peut participer qu'à une seule représentation.

- Representation (**Num_Rep**, titre_Rep , lieu)
- Musicien (**Num_mus**, nom , #Num_Rep)
- Programmer (Date , #Num_Rep, tarif)

Ecrire les requêtes permettant d'afficher :
1) La liste des titres des représentations.
2) La liste des titres des représentations ayant lieu au « théâtre allissa ».
3) La liste des noms des musiciens et des titres et les titres des représentations auxquelles ils participent.
4) La liste des titres des représentations, les lieux et les tarifs du 25/07/2008.
5) Le nombre des musiciens qui participent à la représentations n°20.
6) Les représentations et leurs dates dont le tarif ne dépasse pas 30 €.

Exercice 2 : Salariés et salaires

Soit la base de données suivante :

- departement :(**no_dep**, nom, directeur, ville)
- employes : (**no_emp**, nom, profession, date_embauche, salaire, montant_commission, #no_dep)

Exprimez en SQL les requêtes permettant d'obtenir :
1) La liste des employés ayant une commission
2) Les noms, emplois et salaires des employés par emploi croissant, et pour chaque emploi, par salaire décroissant
3) Le salaire moyen des employés
4) Le salaire moyen du département Production
5) Les numéros de département et leur salaire maximum
6) Les différentes professions et leur salaire moyen
7) Le salaire moyen par profession le plus bas
8) Le ou les emplois ayant le salaire moyen le plus bas, ainsi que ce salaire moyen

Exercice 3 : Etudiants

Soit le modèle relationnel suivant relatif à la gestion des notes annuelles d'une promotion d'étudiants :

- etudiant(**no_etudiant**, Nom, Prénom)
- matiere(**code_mat**, libelle, coef)
- evaluer(#no_etudiant, #code_mat, date_eval, note)

Exprimez en SQL les requêtes suivantes :
1) Quel est le nombre total d'étudiants ?
2) Quelles sont, parmi l'ensemble des notes, la note la plus haute et la note la plus basse ?
3) Quelles sont les moyennes de chaque étudiant dans chacune des matières ?

Exercice 4 : Gestion de projets

Soit la base de données intitulée "gestion_projet" permettant de gérer les projets relatifs au développement de logiciels. Elle est décrite par la représentation textuelle simplifiée suivante :

- developpeur (**NumDev**, NomDev, AdrDev, EmailDev, TelDev)
- projet (**NumProj**, TitreProj, DateDeb, DateFin)
- logiciel (**CodLog**, NomLog, PrixLog, #NumProj)
- realisation (#NumProj, #NumDev)

Ecrire les requêtes SQL permettant de déterminer :

1) Les noms et les prix des logiciels appartenant au projet ayant comme titre « gestion de stock », triés dans l'ordre décroissant des prix.
2) Le total des prix des logiciels du projet numéro 10. Lors de l'affichage, le titre de la colonne sera « cours total du projet ».
3) Le nombre de développeurs qui ont participé au projet intitulé « gestion de stock »
4) Les projets qui ont plus que 5 logiciels
5) Les numéros et noms des développeurs qui ont participés à tous les projets.
6) Les numéros de projets auxquels tous les développeurs de la base participent.

Exercice 5 : Compagnie aérienne

On considère la base de données suivante :

- pilote (**numpil**, nompil, ville, salaire)
- avion (**numav**, nomav, capacite, ville)
- vol (**numvol**, #numpil, #numav, ville_dep, ville_arr, h_dep, h_arr)

Donner les requêtes SQL permettant de répondre aux questions suivantes :

1) Donner toutes les informations sur les pilotes de la compagnie.
2) Donnez la liste des avions dont la capacité est supérieure à 350 passagers.
3) Quels sont les numéros et noms des avions localisés à Lyon ?
4) Quels sont les numéros des pilotes en service et les villes de départ de leurs vols ?
5) Quel est le nom des pilotes domiciliés à Roubaix dont le salaire est supérieur à 6000 € ?
6) Quels sont les avions (numéro et nom) localisés à Paris ou dont la capacité est inférieure à 350 passagers ?
7) Quels sont les numéros des pilotes qui ne sont pas en service ?
8) Donnez le numéro des vols effectués au départ de Paris par des pilotes de Roubaix ?
9) Quels sont les vols effectués par un avion qui n'est pas localisé à Paris ?
10) Quelles sont les villes desservies à partir de la ville d'arrivée d'un vol qui part de Lille ?

Exercice 6 : Café

On considère la base de données suivante :

- servir (cafe, boisson)
- frequenter (client, cafe)
- apprecier (client, boisson)

Exprimer les requêtes suivantes en SQL.

1) Les cafés qui servent une boisson appréciée par 'Ahmed'.
2) Les clients qui vont dans les mêmes cafés que Ahmed.
3) Les clients qui fréquentent au moins un café où l'on sert une boisson qu'ils aiment.
4) Les clients qui ne fréquentent aucun café où l'on sert une boisson qu'ils aiment.
5) Les clients qui fréquentent tous les cafés.
6) Les clients qui fréquentent tous les cafés qui servent au moins une boisson qu'ils aiment.
7) Les clients qui ne fréquentent que les cafés qui servent une boisson qu'ils aiment.
8) Donner pour chaque client, le nombre de cafés servant une boisson qu'ils aiment.
9) Les clients qui fréquentent au moins 2 cafés où l'on sert une boisson qu'ils aiment.