**SQL pour la Data Science de A à Z ( analyse de data réelles)**

# SECTION 1 - INTRODUCTION

Présentation de la base de données SQLite  
Architecture d'une base de données  
Premières requêtes SQL avec les commandes SELECT, FROM, WHERE et LIMIT

# SECTION 2 - OPERATEURS LOGIQUES ET ORDRE

Utilisation des opérateurs logiques AND et OR  
Trier les résultats avec ORDER BY  
  
SECTION 3 - CHALLENGE 1: REQUETES SQLValidation des acquis

SECTION 4 - REQUETE SUR SQLITE DEPUIS PYTHONAssociation Python / SQL  
Exécuter une requête SQL sur Python

SECTION 5 - PROJET GUIDE: BASE DE DONNEES SQLITE (BONUS PYTHON)2 analyses de données SQLite sous Python (niveau intermédiaire)

SECTION 6 - CALCUL DE STATISTIQUES  
COUNT, Valeur Min/Max, SUM, Moyenne  
Afficher les résultats sur Python

SECTION 7 - STATISTIQUES SUR DES GROUPES DE DONNEESCalcul de stats avec GROUP BY  
Commande AS pour renommer une colonne  
HAVING et les colonnes virtuelles

SECTION 8 - CHALLENGE 2: EXPLORATION DE DONNEES  
Analyse statistique de données

SECTION 9 - MODIFIER LES DONNEES DANS UNE TABLEInsérer/modifier/supprimer des données dans une table  
Filtrer les valeurs manquantes

SECTION 10 - SCHEMA D'UNE TABLE  
Ajouter/supprimer une colonne  
Créer une table + table avec relations  
Commande INNER JOIN

SECTION 11 - NORMALISATION DE LA BASE DE DONNEES ET RELATIONS  
Relations one-to-many et many-to-many  
Join Table

SECTION 12 - PROJET GUIDE: PREPARER LES DONNEES POUR SQLITE (BONUS PYTHON)  
Cleaner un dataset en Python pour l'exporter sur SQLite (niveau intermédiaire)

SECTION 13 - PROJET GUIDE: CREER DES RELATIONS SUR SQLITE (BONUS PYTHON)  
Créer 2 tables liées par une relation many-to-many (niveau intermédiaire)

SECTION 14 - POSTGRESQL  
PostgreSQL vs. SQLite (battle)  
Connexion à une base de données PostgreSQL

SECTION 15 - COMMAND LINE POSTGRESQL  
Exécuter des requêtes SQL sur psql  
Gérer les utilisateurs et leur donner des pouvoirs

SECTION 16 - UTILISATION POSTGRESQL AVEC PYTHON  
Exécuter des requêtes SQL sur Python

SECTION 17 - PROJET FINAL: EXPLORATION DE DONNEES POSTGRESQL  
Etude réelle d'une base de données de location de DVD

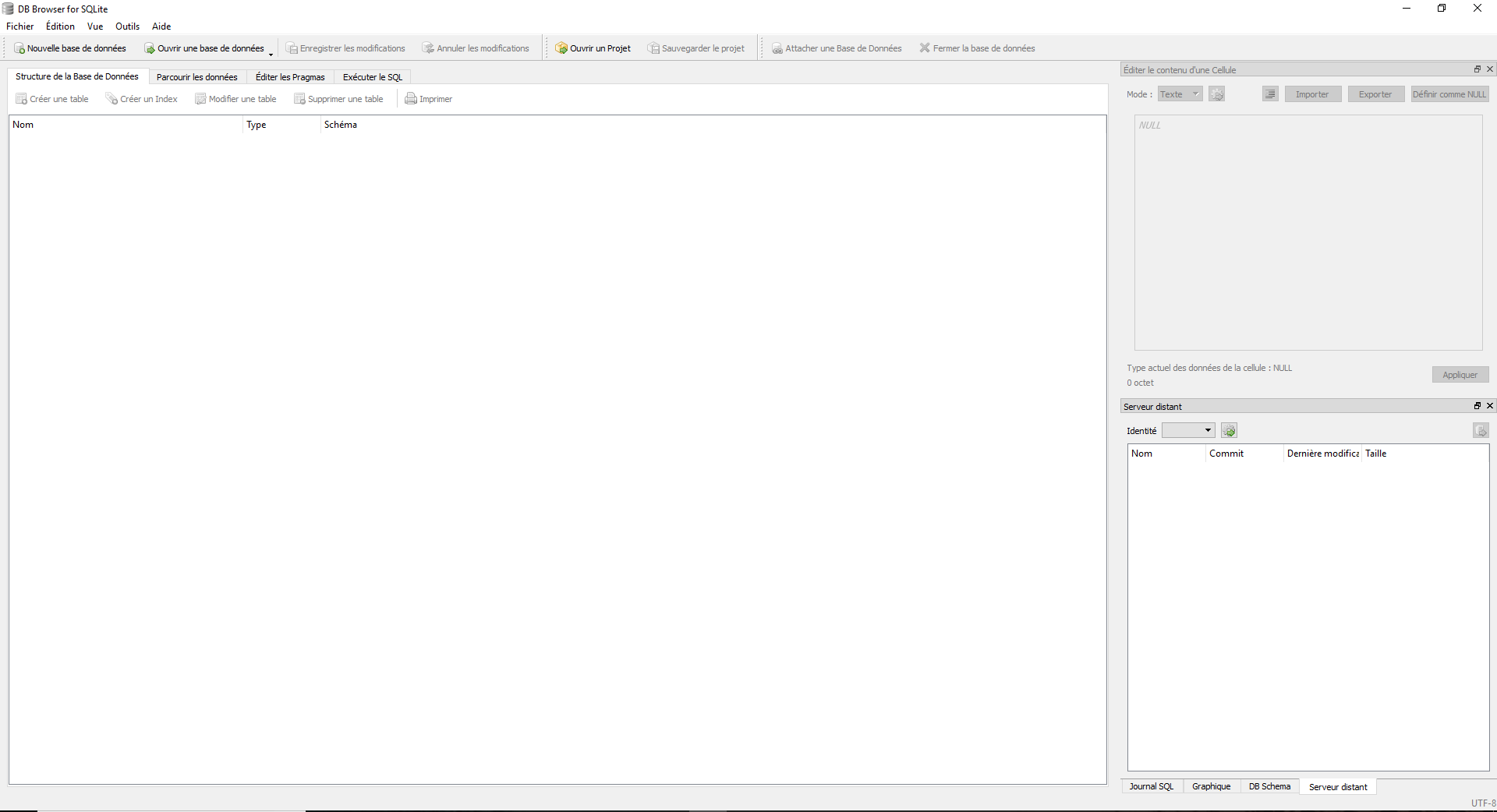
# SECTION 1 - INTRODUCTION

##### **Installation et Utilisation de DB Browser SQLite**

DB Browser pour SQLite

BD Browser : c’est une interface graphique qui permet de manipuler des BDD SQLite

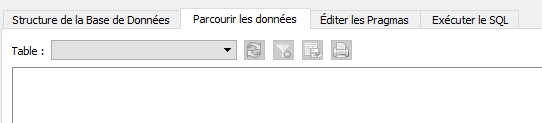
SQLite : c’est une BDD très légère pour l’apprentissage de la syntaxe SQL



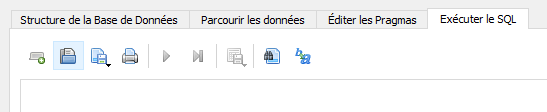
L’interface se présente comme çà :



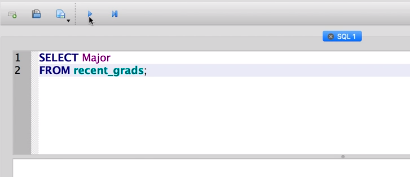
Structure de la base de données : Nous permet de connaitre comment est schématisée la bdd



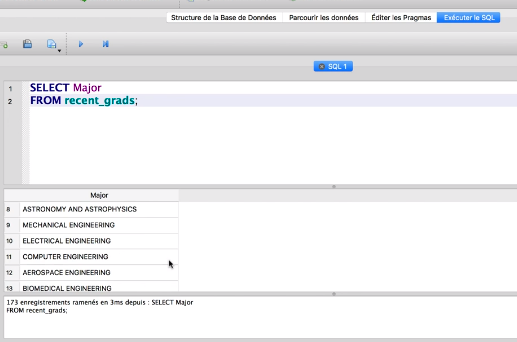
Parcourir les données : nous montre les tables de cette bdd



Exécuter le SQL : Permet de renseigner les requêtes SQL pour interroger la bdd.



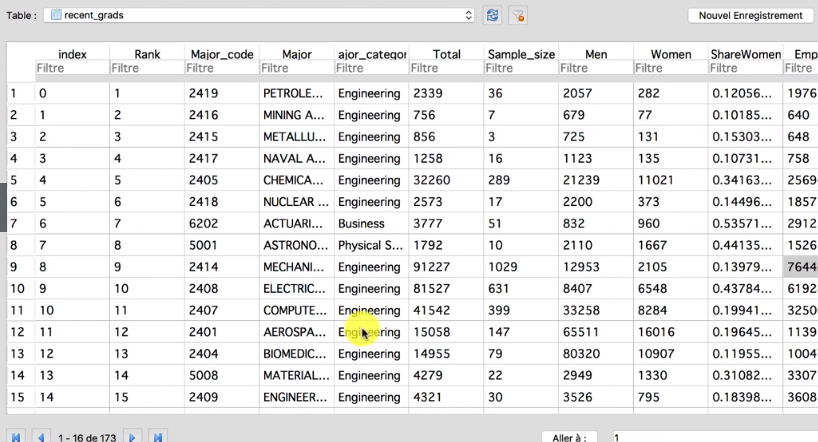
Soit cliquer sur le bouton **play** , ou **F5** pour commiter la requête.



##### **Introduction aux bases de données et SQL**

BDD : Une base de donnée contient des **tables** et ces tables contiennent des informations.

Ces informations se situent dans un ensemble de **colonnes** et un ensemble de **lignes** pour lesquelles chaque lignes a des valeurs pour chaque colonnes.



Ici nous avons 173 informations.

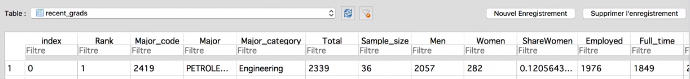
Une BDD : Il s’agit d’un entrepôt prévu pour stocker, questionner et traiter des informations. Les bdd stockent de la data et fournissent une interface pour interagir avec.

SQL : Structure Querie Language

##### **Comprendre les tables, lignes, et colonnes**



BDD contient les résultats d’un sondage sur les diplôme américain et leur revenu à la sortie.



Rank : classement de la major (matière principale) selon le salaire médiant à la sortie des études.

Major\_code : Code nominative de la major (GUID)

Major : Le nom de la major

Major\_category : La catégorie à laquelle appartient la major

Total : le nombre total de personne ayant étudié cette major

Sample\_size :la taille de l’échantillon des diplômés ayant des emplois à temps plein

Men : nombre d’hommes diplômés

Women :nombre de femmes diplômées

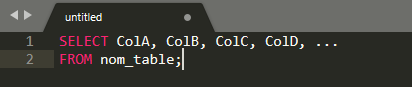
ShareWomen : la proportion de femme diplômées par rapport au total de diplômés par major.

Employed : le nombre de diplômé qui ont un emploi

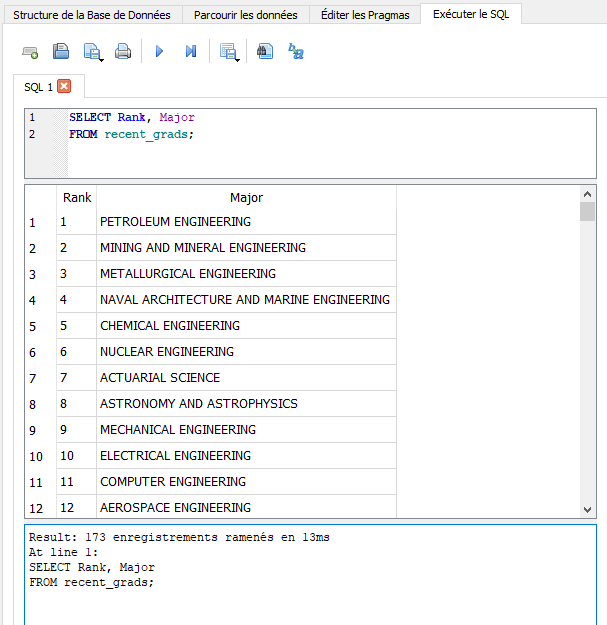
Full\_time : le nombre de diplômé qui ont un emploi à temps plein.

##### **Première requête SQL**

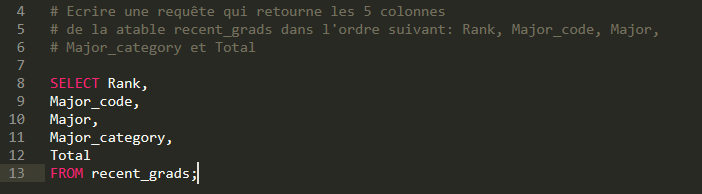
Utilisation de l’éditeur de texte Sublime texte

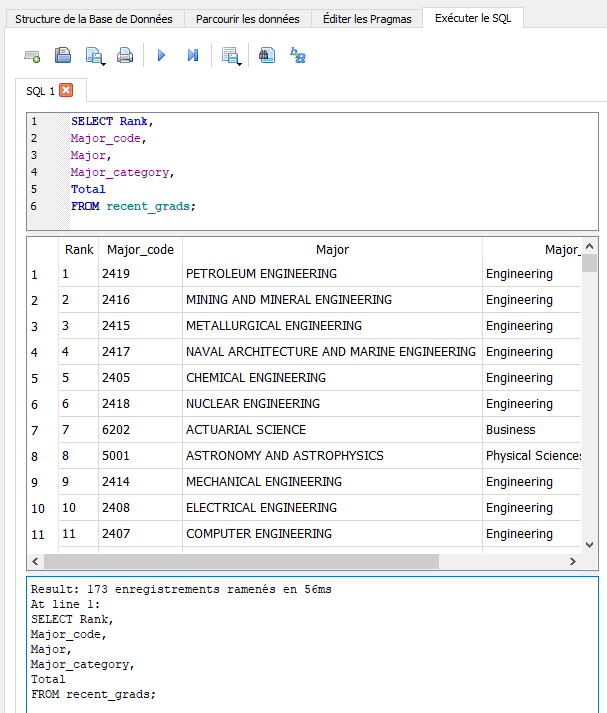


On veut afficher depuis la table recent\_grads, seulement les colonnes Rank et Major



##### **Ordre des colonnes**





RequêtesSQL

##### **Filtrer avec WHERE**

Nous filtrons les données. Récupérer les majors qui ont plus de femmes diplômées que d’hommes.

Pour obtenir ce résultat, on cible la colonne ShareWomen.

ShareWomen : la proportion de femme diplômées par rapport au total de diplômés par major.

ShareWomen > 0.5 : sur l’ensemble des diplômés, il y a plus de 50% de femmes.

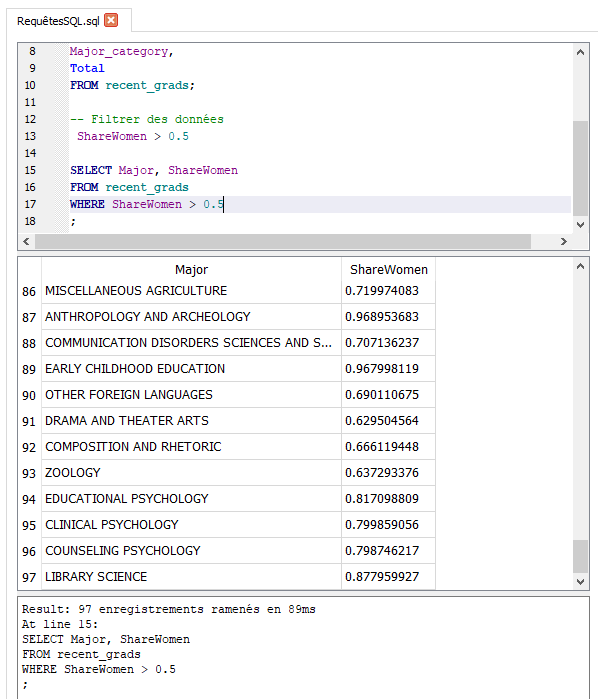


Commande pour ajouter une condition : **WHERE**

La cmd **WHERE** va être suivie de 3 éléments :

* La colonne à filtrer
* Un opérateur de comparaison (= > < …)
* La valeur à comparer.

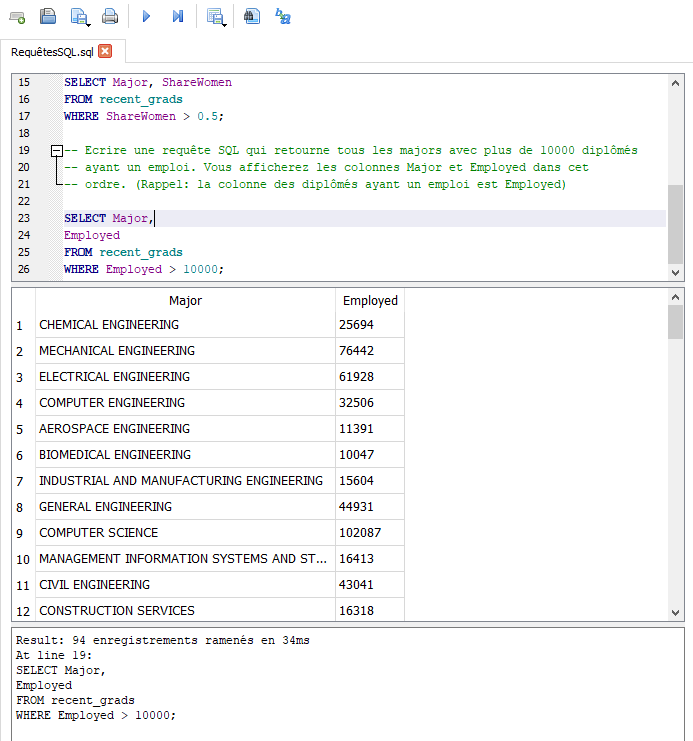
Là çà filtrerait les éléments pour lesquels la colonne A est égale à 3



Les comparateur possible :

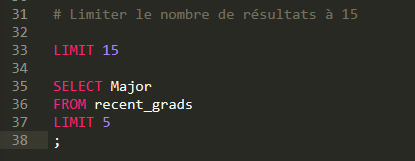
* >Supérieur
* < inférieur
* >= supérieur ou égale
* <= inférieur ou égale
* != différent

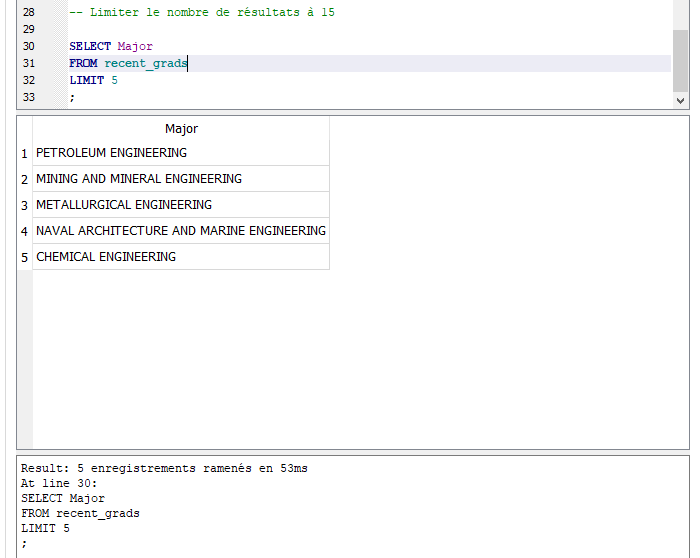
**La condition WHERE se place tjs après le SELECT ET FROM d’une requête.**

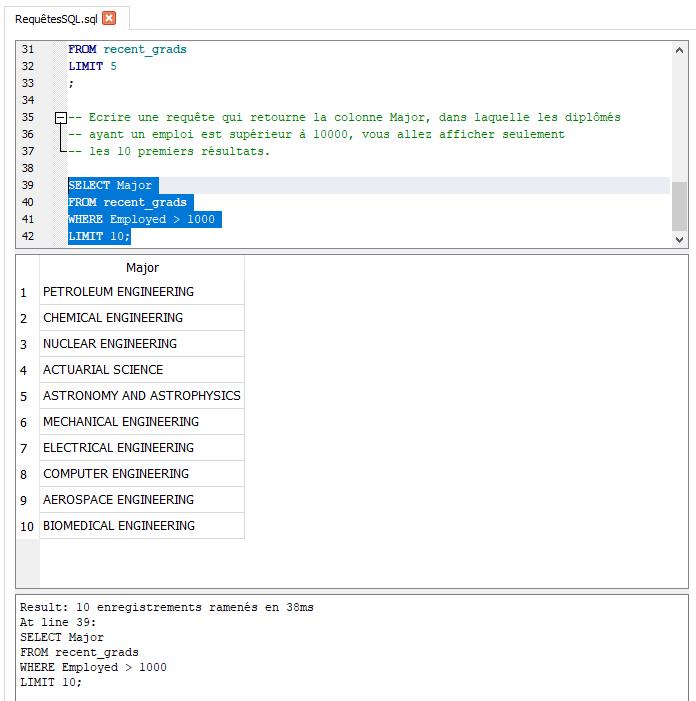


##### **Limiter le nombre de résultats**

La commande **LIMIT**, immédiatement suivie d’un entier qui spécifie le nombre de résultats souhaité.



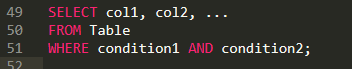




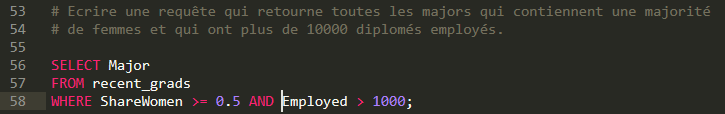
# SECTION 2 - OPERATEURS LOGIQUES ET ORDRE

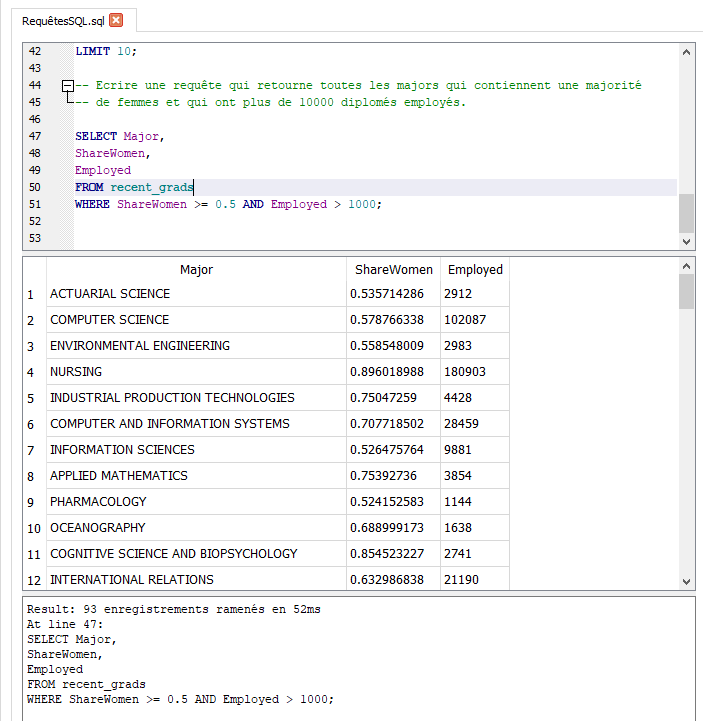
##### **Condition multiples avec AND et OR**

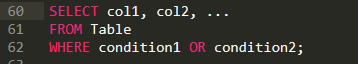
But des conditions multiples : combiner plusieurs critères dans une requête avec **les opérateurs logique AND et OR**



Ici on va afficher à la fois les lignes qui contiennent la condition1 et la condition2

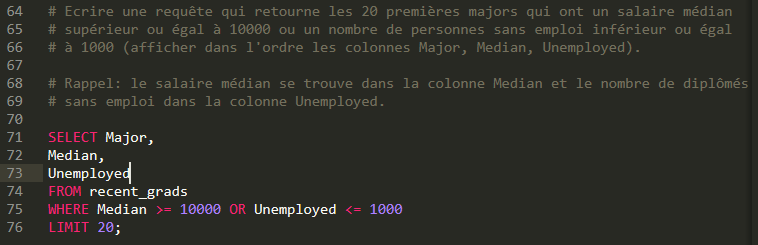


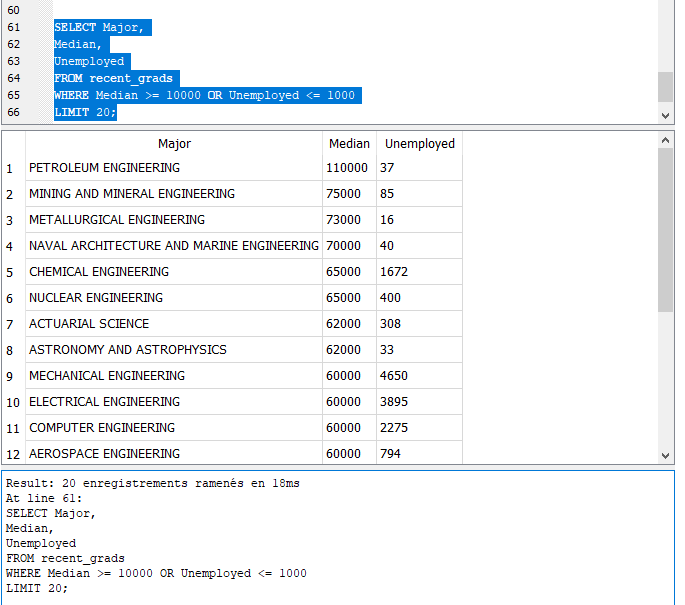




Ici avec la condition OR, on va afficher les lignes qui présentent soit la condition1 soit la condition2

Il est suffisant d’avoir une des 2 conditions pour apparaitre dans les résultats

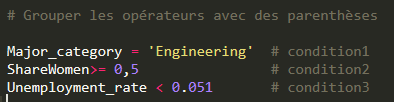


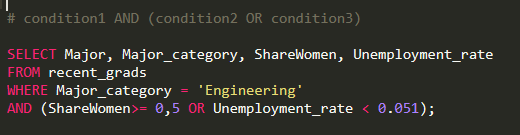


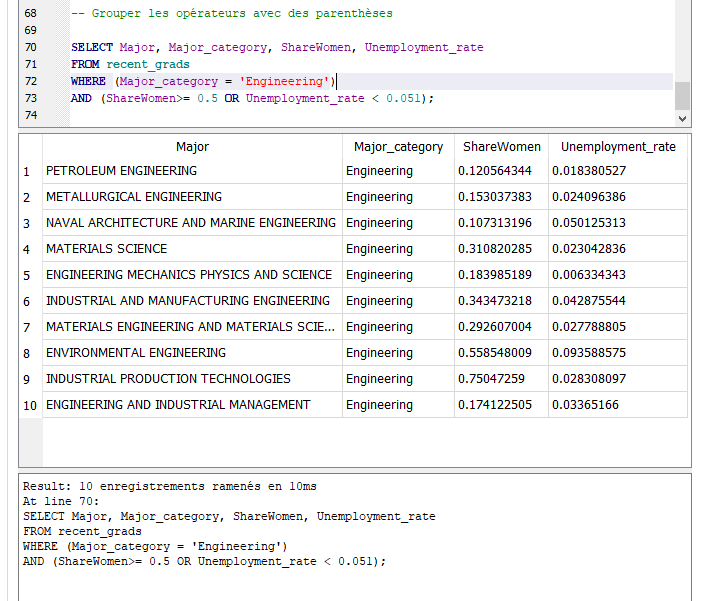
##### **Grouper les opérateurs avec des parenthèses**

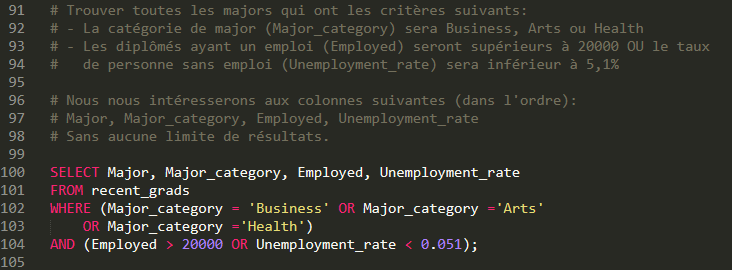
Par exemple si nous souhaitons écrire une requête qui retourne toutes les majors « engineering », donc les matières principales liées à l’ingénierie **et,** qui ont principalement des femmes diplômées **ou** un taux de personnes sans emploi en dessous de 5,1% .

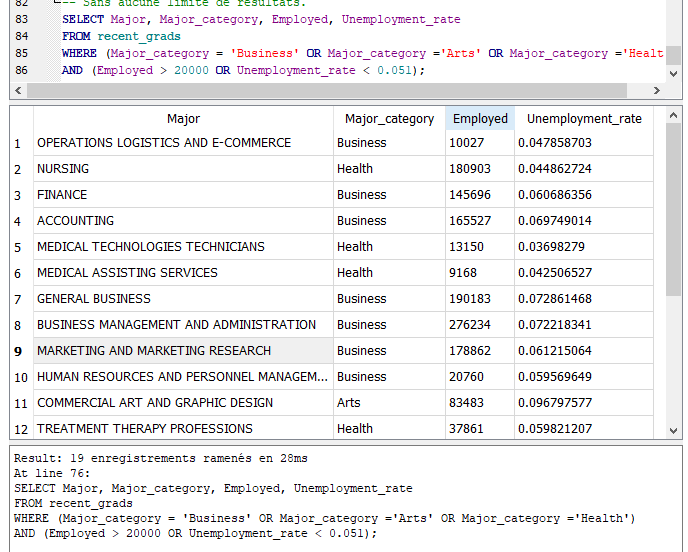
Dans ce cas, nous avons forcément besoin de parenthèses pour exprimer ces conditions. OR et AND ne suffisent pas.





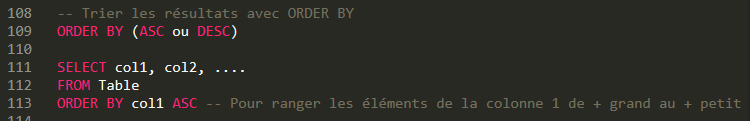


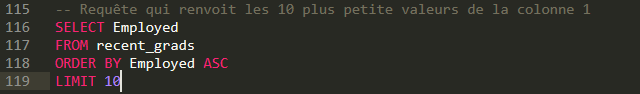


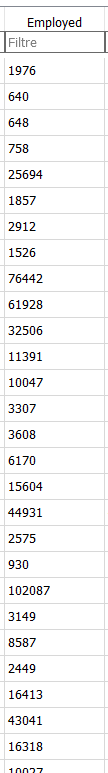
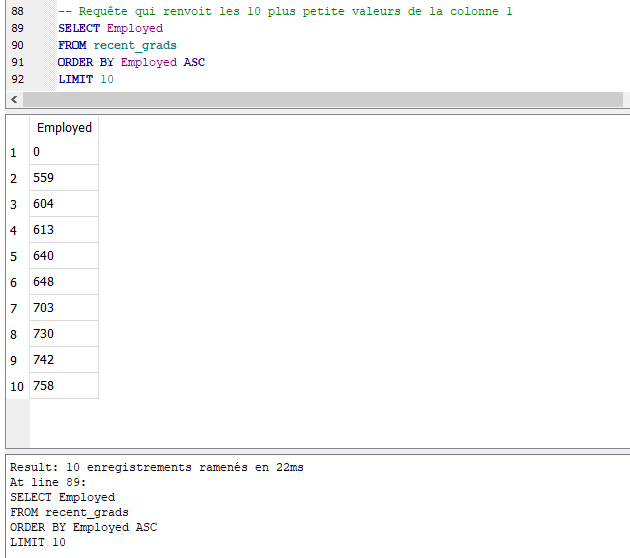


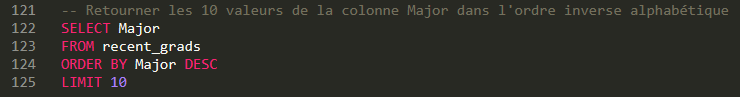
##### **Trier les résultats avec ORDER BY**

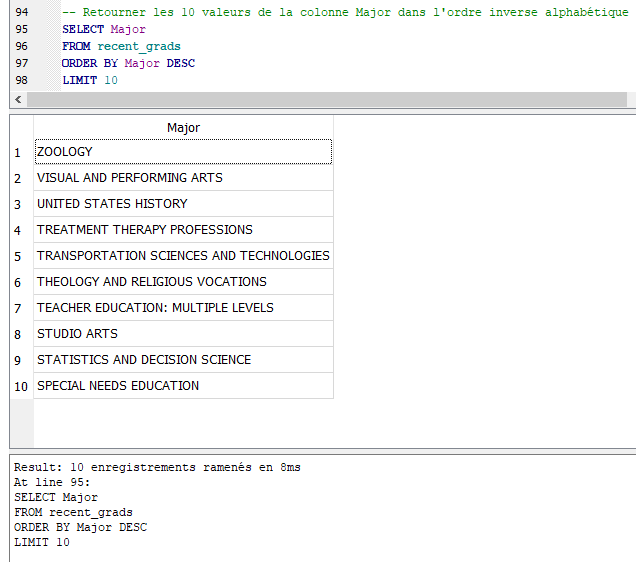
**ORDER BY**, il faut spécifier l’ordre croissant ou décroissant





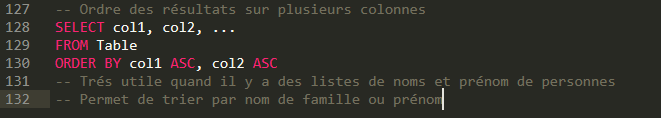


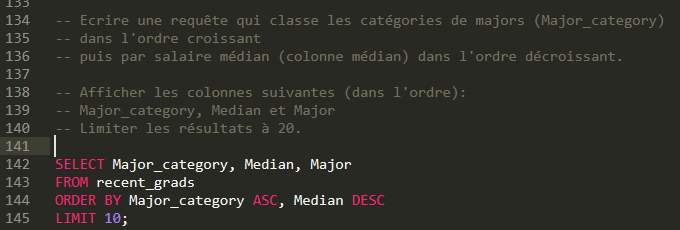


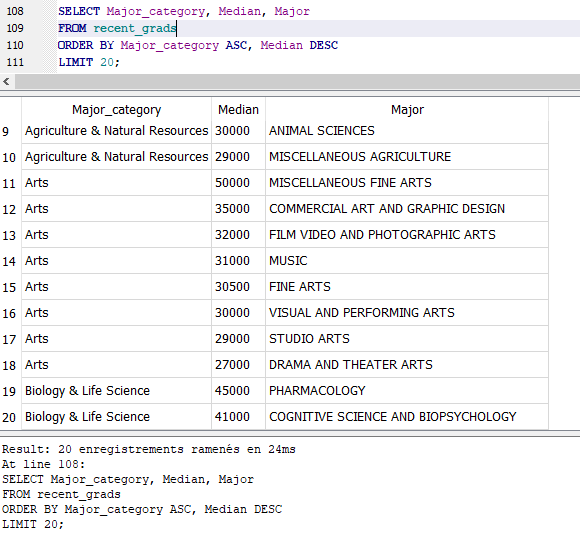


##### **Ordre des résultats sur plusieurs colonnes**

SQL nous permet de spécifier l’ordre à partir de plusieurs colonnes. On peut donc utiliser la colonnes ORDER BY avec 2 colonnes ou plus, et si il y a égalité entre 2 valeurs de la première colonne, alors elle vont être départagées à l’aide de la valeur de la 2d colonne.

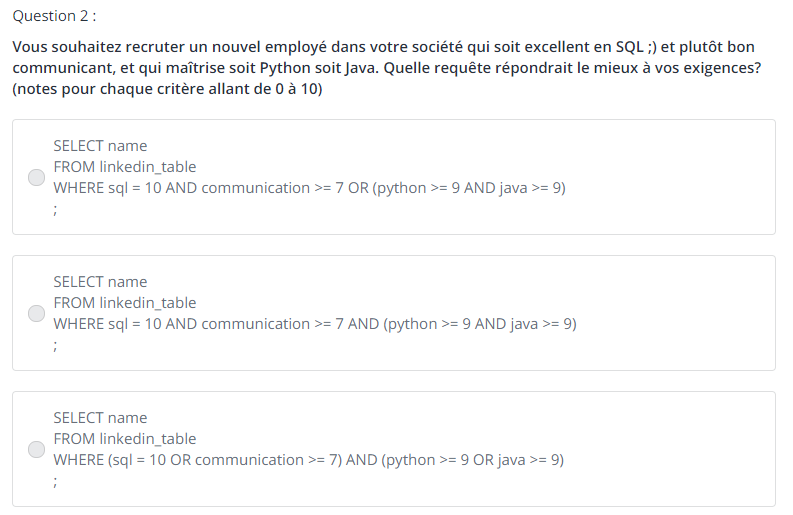




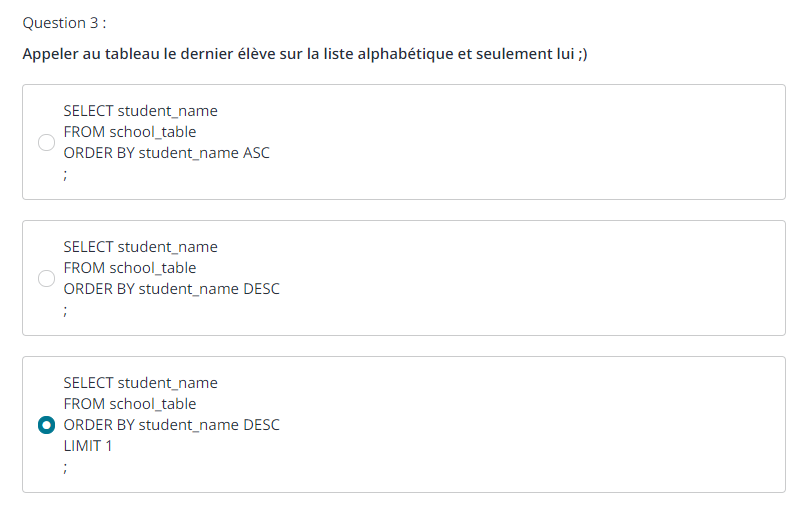


##### **Quiz 1**





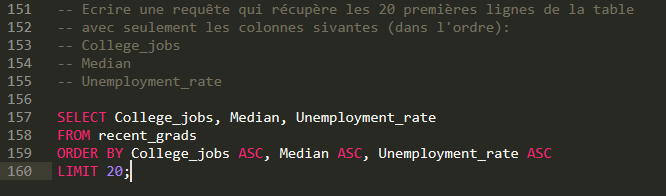


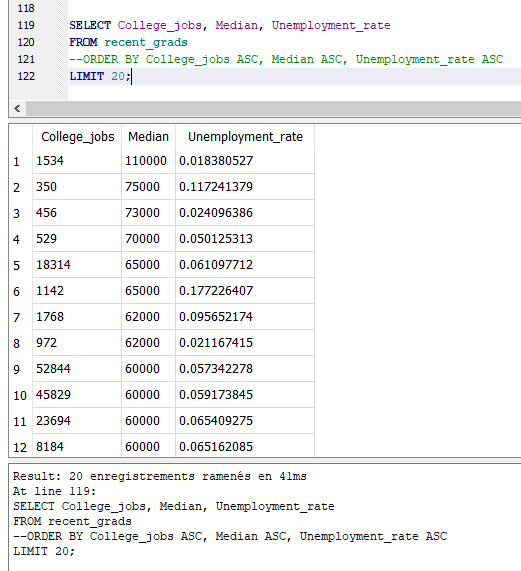




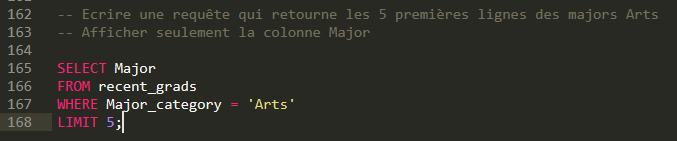
# SECTION 3 – CHALLENGE 1 : REQUÊTES SQL

##### **Utiliser SELECT et LIMIT pour afficher certains éléments de la table**



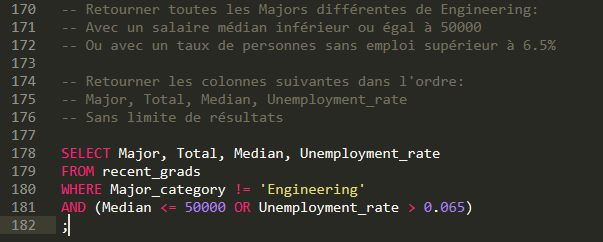


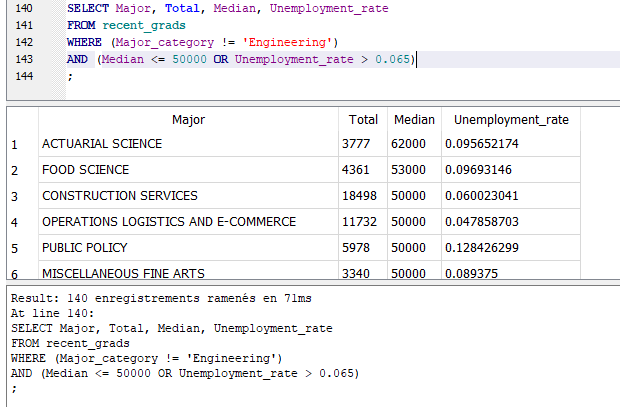
##### **Utiliser WHERE pour filtrer les résultats**



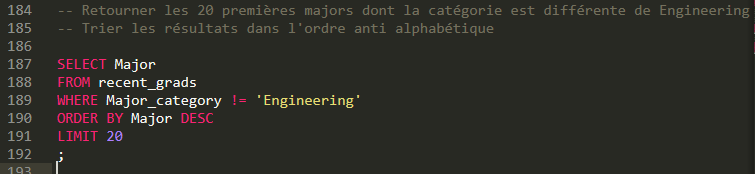


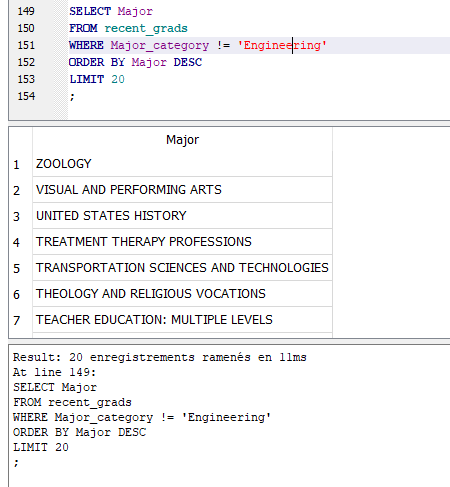
Ajouter des critères avec des opérateurs logiques





##### **Trier les résultats**





# SECTION 4 – REQUÊTES SUR SQLITE AVEC PYTHON

##### **Intérêt de Python**

L’intérêt est de travailler avec les données extraites d’une bdd sur un workflow permettant de manipuler et préparer la data.

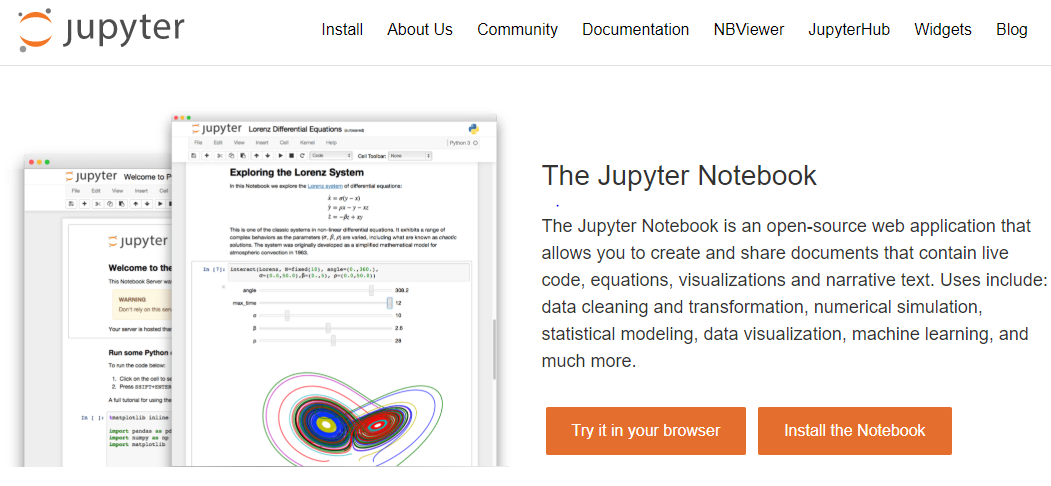
Nous pouvons interagir avec une bdd SQLite de plusieurs façon :

* A travers le module Python SQLite
* A travers la commande SQL
* Ou encore avec des logiciel graphique tel que DB Browser SQLite

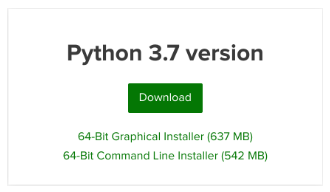
##### **Guide installation Python**

Guide d'installation Python + Jupyter Notebook (Anaconda)

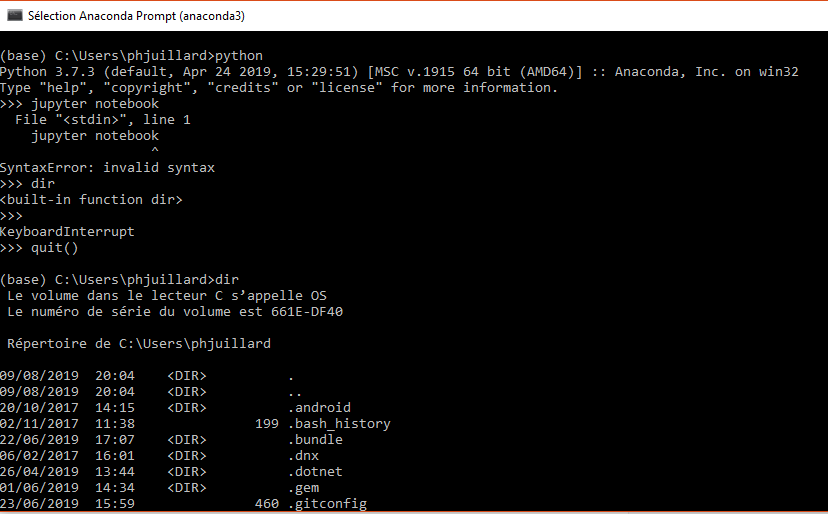
1. Jupyter Website ([jupyter.org](http://jupyter.org/))
2. Descendre sur la page et cliquer sur le bouton *Install the Notebook*



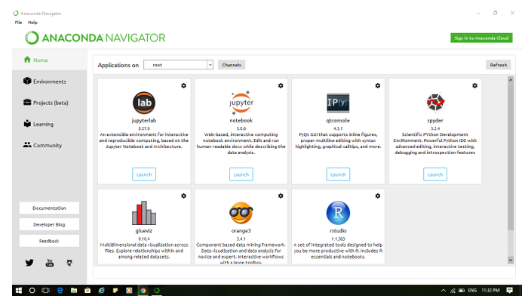
1. Cela vous dirige vers le [lien d'installation Anaconda](https://www.anaconda.com/distribution/) et cliquer sur le bouton d'installation correspondant à votre système (Windows, Mac ou Linux) en choisissant Python > 3.7.



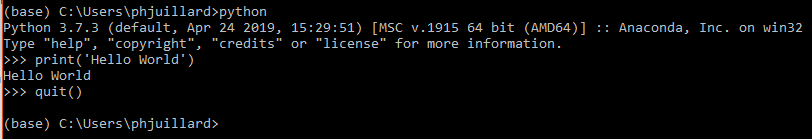
1. Suivre les instructions pour l'installer sur votre système, assez similaire à l'installation de n'importe quel logiciel.
2. *Une fois Anaconda installé, si vous êtes sur****Mac****ou****Linux****lancer votre terminal (Application/utilitaires/Terminal) puis taper python  pour vérifier que l'installation a fonctionné (quitter Python en tapant quit() ) et jupyter notebook  pour lancer Jupyter Notebook (quitter en tapant sur les touches ctrl+C). Utiliser la commande ls  pour afficher les fichiers/dossiers présents dans le dossier actuel et cd nom\_dossier  pour vous déplacer dans votre arborescence.*
3. Si vous êtes sur **Windows**, faites une recherche Windows '**Anaconda Prompt**', puis l'ouvrir et ensuite utiliser les commandes identiques à la ligne du dessus: jupyter notebook  pour lancer un notebook (ctrl+C pour quitter) et python  pour exécuter du code python (quit()  pour quitter). Utiliser la commande dir   pour afficher les fichiers/dossiers présents dans le dossier actuel et cd nom\_dossier  pour vous déplacer dans votre arborescence.



1. **Autre méthode** pour démarrer Jupyter Notebook, ouvrir le logiciel **Anaconda-Navigator** en faisant une recherche sur Windows ou Mac et en sélectionnant l'application **Jupyter Notebook** une fois le logiciel ouvert (screenshot ci-dessous).

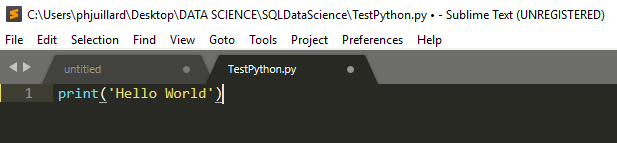


##### **Installation Python et Jupyter Notebook**



On souhaite exécuter un fichier python (.py), plusieurs mthd possible, soit avec la cmd directement ou avec Jupyter Notebook.

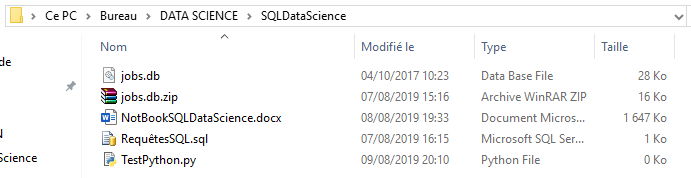
On va sur sublime text, new file en python : **TestPython.py**



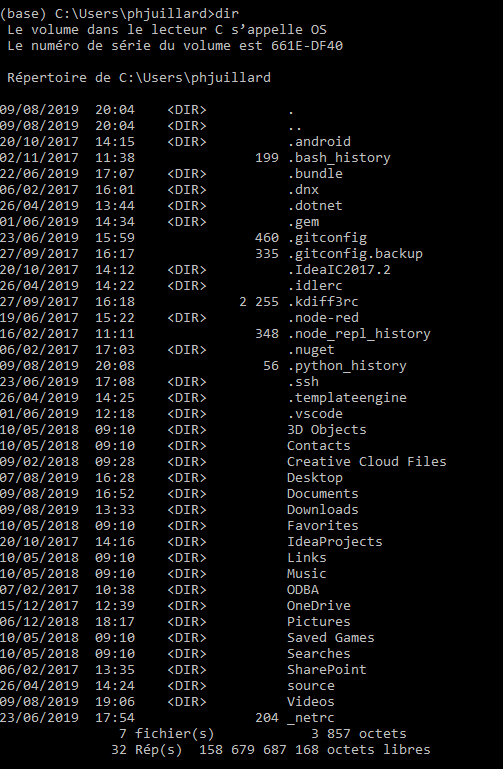
* La méthode dans la cmd

On souhaite ouvrir le fichier python TestPython enregistré dans le dossier ci-dessous dans la cmd.

C:\Users\phjuillard\Desktop\DATA SCIENCE\SQLDataScience



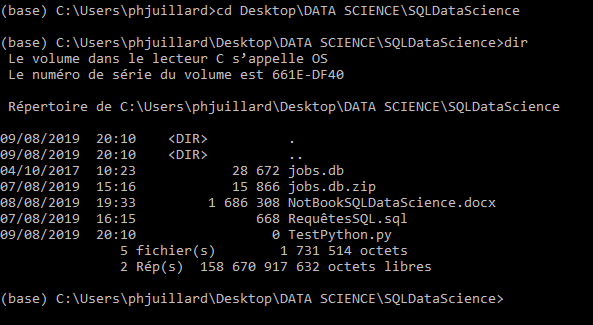
On tape dir dans cmd pour afficher tous les fichiers/dossiers présents dans le dossier actuel où on se trouve avec la cmd.



Nous sommes à la racine du Users phjuillard, on va rentrer dans le dossier SQLDataScience

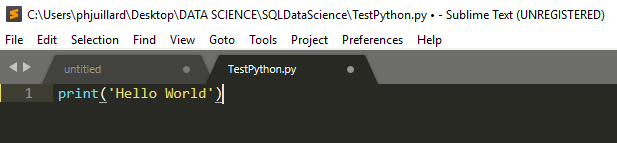
C:\Users\phjuillard\Desktop\DATA SCIENCE\SQLDataScience

cd Desktop\DATA SCIENCE\SQLDataScience  pour se déplacer dans mon arborescence



Maintenant on exécute le fichier python depuis la cmd

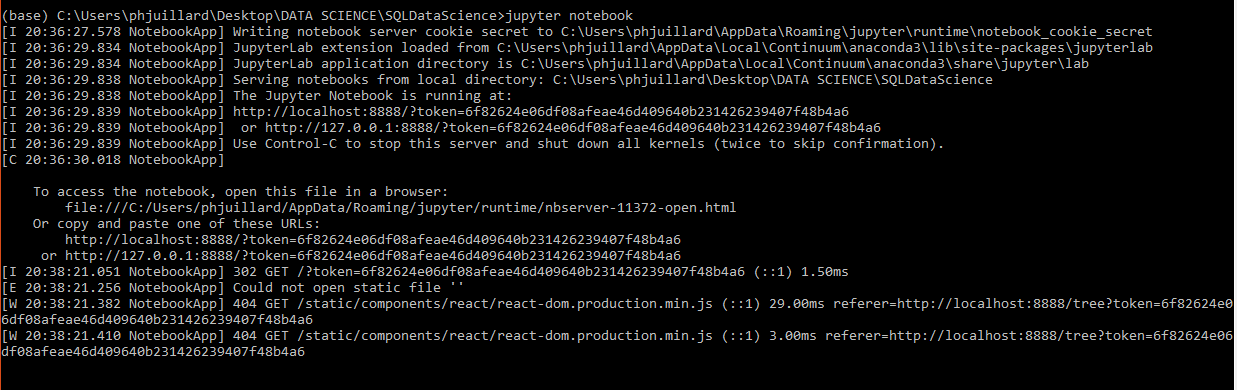




* La méthode dans Jupyter Notebook

Plus agréable pour rédiger des $cript python

Pour le lancer il suffit de marquer jupyter notebook

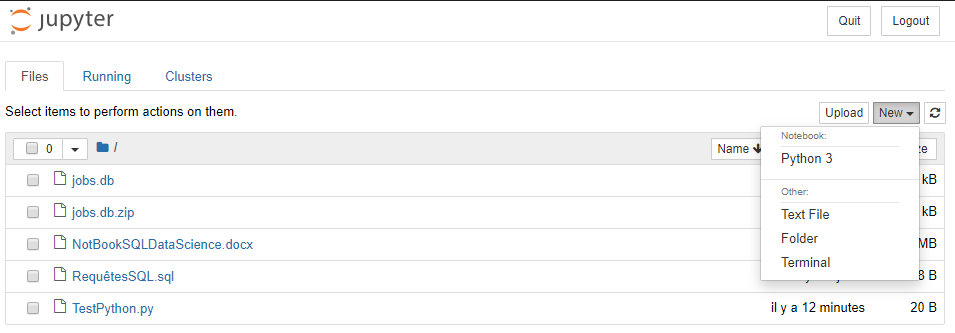


<http://localhost:8888/?token=6f82624e06df08afeae46d409640b231426239407f48b4a6>

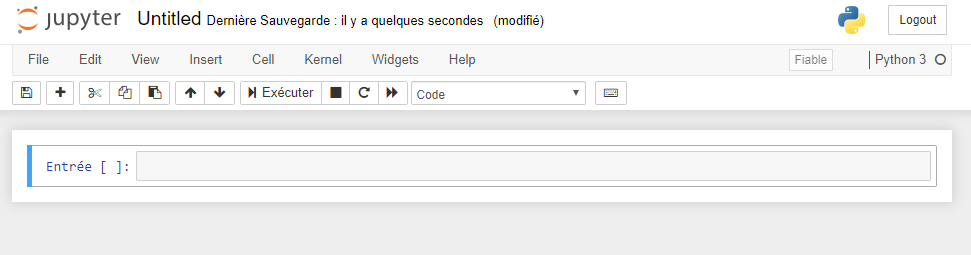


Je suis déjà dans le dossier SQLDataScience, et je vois directement les fichiers qui sont présents dans ce dossier.

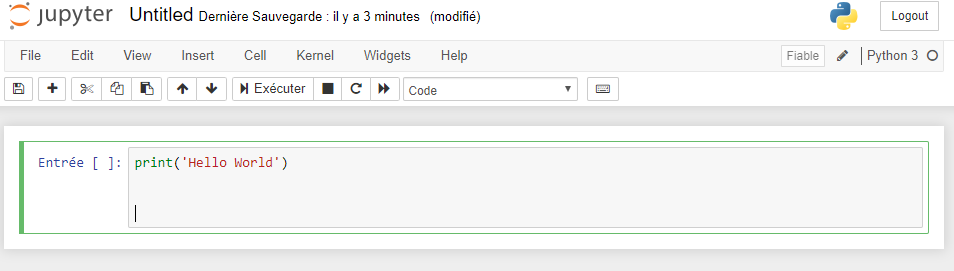
Là on va créer un nouveau fichier Jupyter Notebook



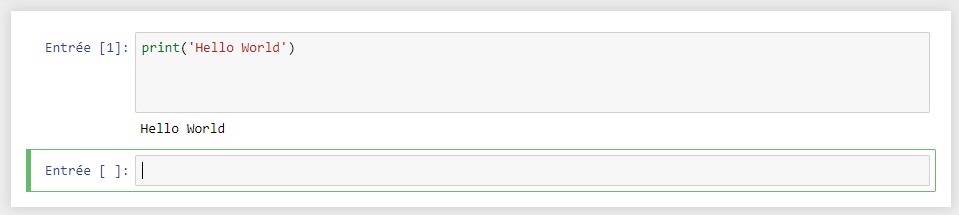
On choisi notre version de python, Python 3



Voilà un jupyter Notebook sur lequel on va pouvoir travailler nos $cript Python pour interroger la bdd.



**NB : si on tape enter, on va à la ligne. Pour exécuter une ligne, on tape sur shift+enter**

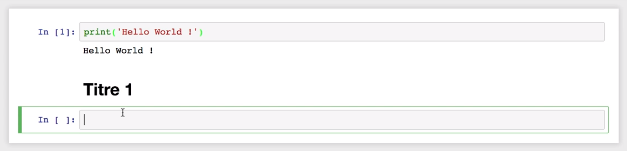
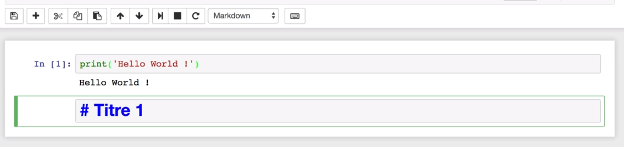


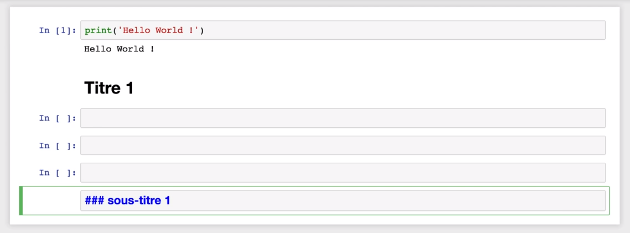
Add Value de Jupyter Notebook:

* C’est ultra simple à lancer, à mettre en place
* C’est ultra simple d’écrire des $cript Python dessus.
* On peut aussi se mettre en Markdone au lieu de code

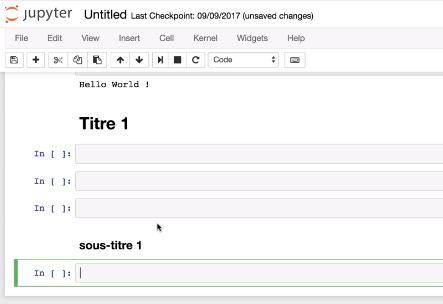


Et en markdone, en utilisant les #### je vais pouvoir séquencer mon travail





**CODE PYTHON**

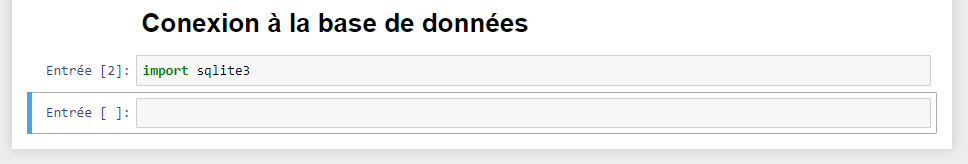


##### **Connexion à la base de données**

Comment se connecter à une BDD SQLite à l’aide Python ?

Pour cela on a un module qui s’appelle SQLite 3, qui est déjà installé sur Python, donc pas besoin d’installer une librairie supplémentaire.

Pour importer cet environnement, on utilise la commande import sqlite3



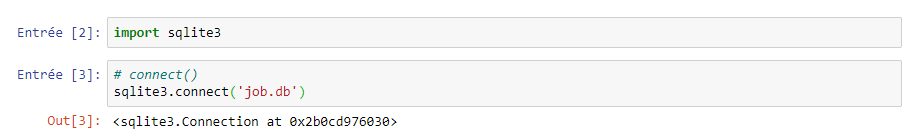
Une fois la librairie importée, on peut utiliser la fonction connect de cette librairie.

Cette fonction connect requière un seul paramètre, ce paramètre est le fichier de la bdd dont nous souhaitons nous connecter. Donc un fichier SQLite.

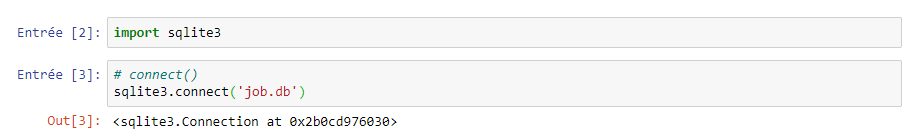
La fonction connect va retourner une instance de connexion, qui maintient la connexion à la bdd.

Ca va nous permettre d’écrire des requêtes, d’afficher, d’ajouter des données dans la bdd SQLite.

La requête complète de connexion sera sqlite3 (car c’est la librairie) **.** connect (on utilise la fonction de la librairie sqlite3) **(‘**le nom du fichier de la bdd’**)**



On voit bien que nous sommes connecté avec cette mention :



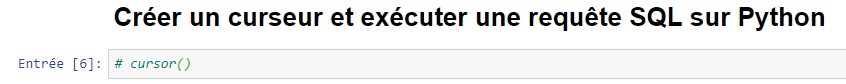
Maintenant on va assigner [3] à une variable connexion

Donc dès que nous aurons besoin d’exécuter une requête ou autre, on utilisera cette variable connexion avec Python 😊.



##### **Créer un Curseur et exécuter une requête SQL sur Python**

Avant d’exécuter une requête, on a besoin d’utiliser un curseur de la classe cursor.



Ce curseur va nous permettre :

* d’exécuter une requête auprès de la bdd.
* D’analyser les résultats de la bdd
* De convertir les résultats en objet Python
* De stocker les résultats dans une variable locale

Donc après avoir exécuter la requête et converti les résultats en liste de tuples, l’instance du curseur stocke la liste en tant que variable locale

**Tuple : c’est une structure que Python utilise pour représenter une séquence de valeur. Un peu comme une liste.**



*Exemple d’une liste en Python (syntaxe)*

Dans notre cas, on va travailler avec des tuples.

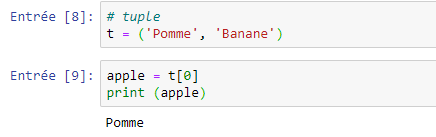
A la différence d’une liste, dans un tuple, chaque éléments de ce tuple sera non modifiable.

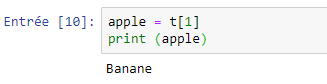
Alors qu’une liste je peux modifier chaque éléments ou en modifier.



*Exemple d’un Tuple (syntaxe)*

On remplit notre tuple (Pomme, Banane). Si je créé une variable apple, qui récupère la première des valeurs de mon tuple . Ensuite j’affiche ma variable apple avec print()





**Add value des Tuples : Les Tuples sont plus rapides que les listes. Donc très utiles pour les grandes bdd en analyse de masse en Data Science.**

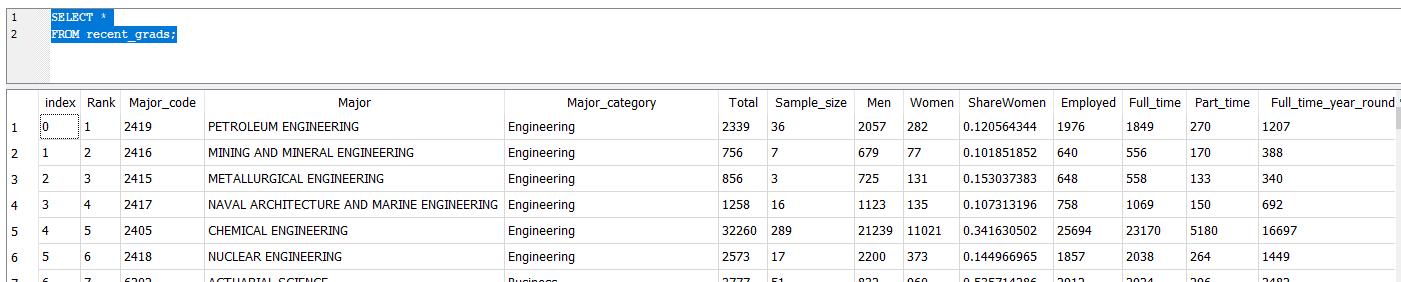
Comment utiliser le curseur pour exécuter des requêtes SQL ?

On va utiliser la mthd cursor sur l’instance de connexion connexion crée en Entrée-[5] ci-dessus.



Maintenant on va écrire notre requête SQL sur la table recent\_grads de la bdd jobs.db

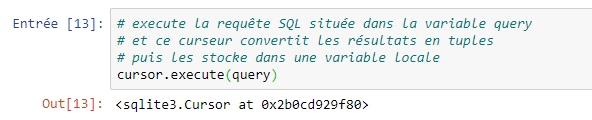
On affiche l’ensemble des colonnes de la table recent\_grads sur DB Browser :



Ensuite on va exécuter cette recette avec Python.

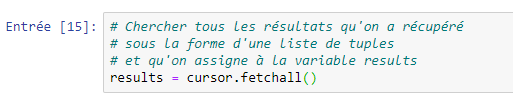


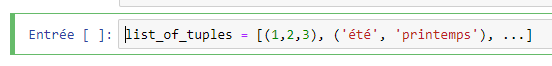
Puis on va exécuteravec la mthd execute cette requête SQL, enregistrée dans la variable query , donc on va pouvoir exécuter directement le mot clé query qui est notre variable avec le curseur.



RAF : Récupérer les résultats de cette variable locale

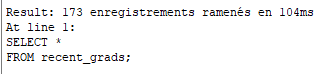
Donc on crée une variable results avec le curseur en utilisant la méthode fetchall (qui récupère toutes les données) en associant ma variable locale **( )**

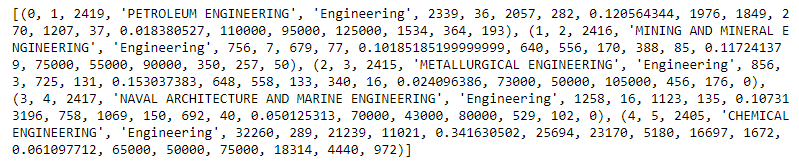




*Exemple d’une liste de tuples (syntaxe)*

Maintenant que nous avons récupéré toutes les données de notre requête SQL dans la variable results , on va afficher ces résultats (partiellement, 5 sur 173 présents)

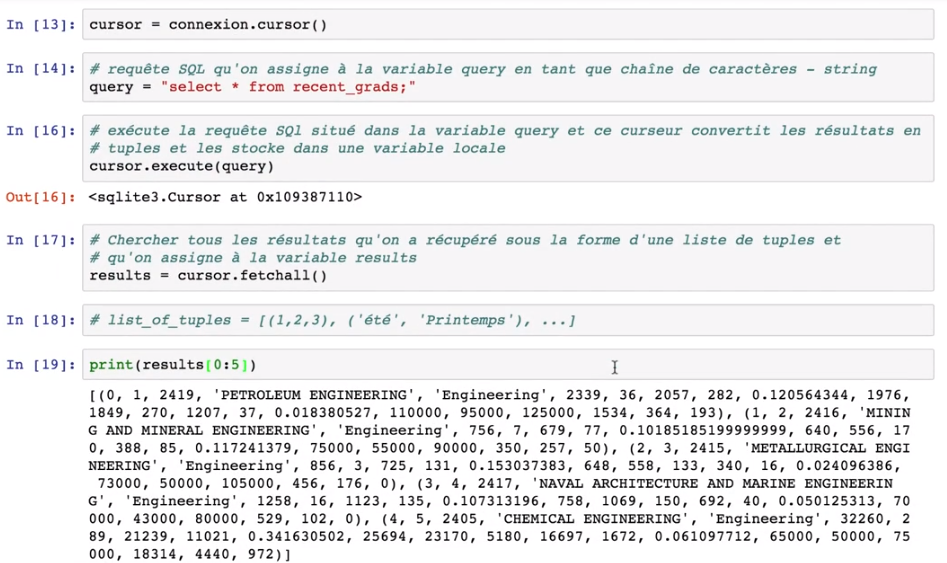
 réparti en 23 colonnes 😉



**SYNTHESE TECHNIQUE :**

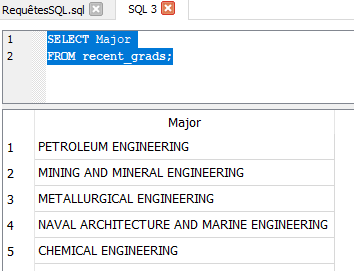
* [2] - On utilise la librairie sqlite3 avec import sqlite3
* [5] - On se connecte à la bdd jobs.db avec la mthd connect en créant une variable connexion.
* [13] - Ensuite on applique la mthd cursor qu’on applique à la variable connexion
* On rédige notre requête SQL (testée sur DB Browser/ SQL server …)
* [14] - On l’assigne à une variable query. Entre **‘’** car c’est une string !
* [16] - On applique la mthd execute sur le curseur avec en paramètre la requête SQL via la variable query.
* [17] - Enfin on applique la mthd fetchall en créant une variable results sur notre curseur. Çà va l’appliquer sur la variable locale qui comprend tous nos tuples = Ce qui permet d’aller chercher tous les résultats, mais pas les afficher.
* [19] - On les affiche avec print en paramétrant et limitant les données à afficher.

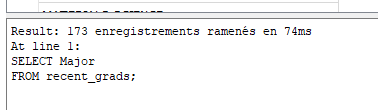




CHALLENGE :

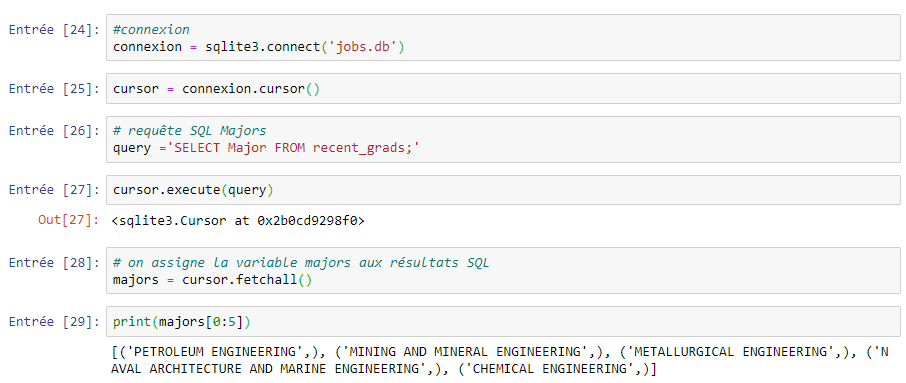
* Ecrire une requête qui retourne toutes les valeurs de la colonne Major depuis la table recent\_grads.
* Stocker les résultats (list de tuples) et assigner les à la variable majors.
* Enfin, afficher les 5 premiers tuples résultats de majors.



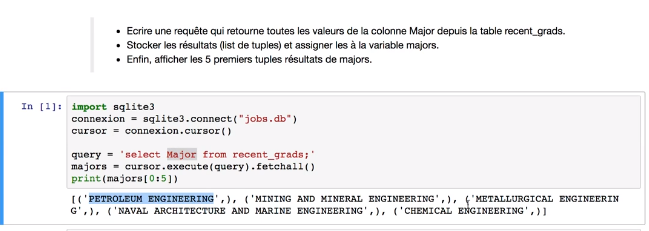




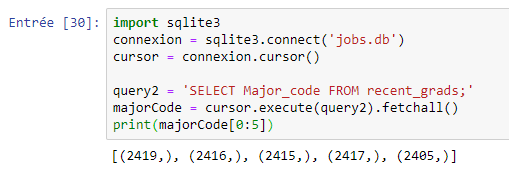
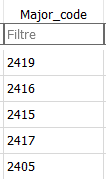
Rajouter import sqlite3 quand c’est un nouveau jupyter



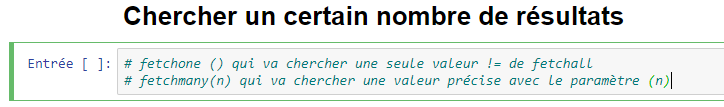
CORRECTION :



* Ecrire une requête qui retourne toutes les valeurs de la colonne Major\_code depuis la table recent\_grads.
* Stocker les résultats (list de tuples) et assigner les à la variable majorCode.
* Enfin, afficher les 5 premiers tuples résultats de majorCode.

##### **Chercher un certain nombre de résultats**



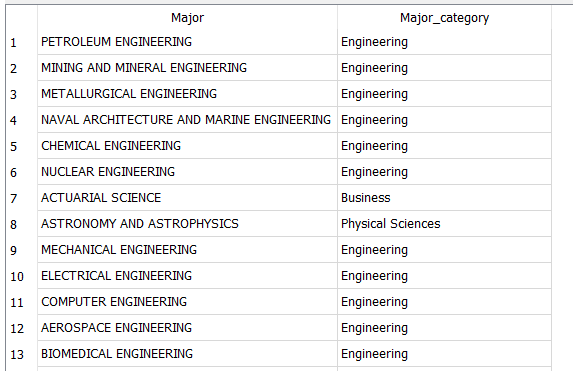
**Chaque instance de curseur contient un décompte interne qui s’incrémente à chaque fois que l’on récupère des résultats.**

* Ce qui signifique que lorsque nous appelons la mthd fetchone, cela retourne un seul résultat et incrémente le décompte de un = çà ajoute la valeur un
  + Ca veut dire que si on utilise de nouveau cette mthd fetchone, cela retournera le deuxième tuple. Et ainsi de suite…
* Si juste après on utilise la mthd fetchmany(5) , ca va afficher du 3eme tuple au 8eme tuple.
* **A chaque fois çà démarre à la position actuelle du cursor.**

Exemple :

On exécute une requête qui retourne les colonnes Major, Major\_category de la table recent\_grads.

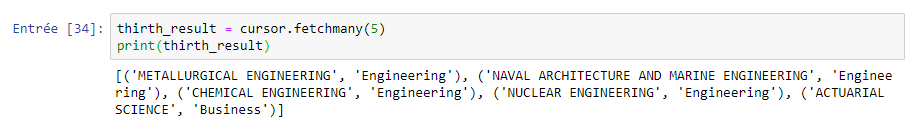






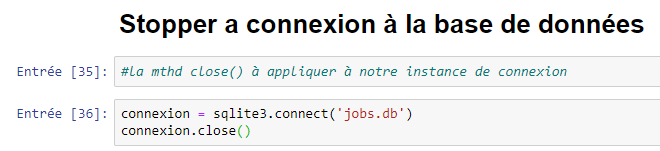
* Après l’execution de la première mthd fetchone(), le curseur a incrémenté d’un pour executer la 2d mthd fetchone, d’où l’affichage de la 2d ligne des colonnes Major et Major\_category.

On teste la mthd fetchmany(n) avec n =5. Ce qui va afficher les résultats des lignes 3 à 8 suivant la position actuelle du curseur, via la variable thirth\_result.



##### **Stopper la connexion à la base de données**

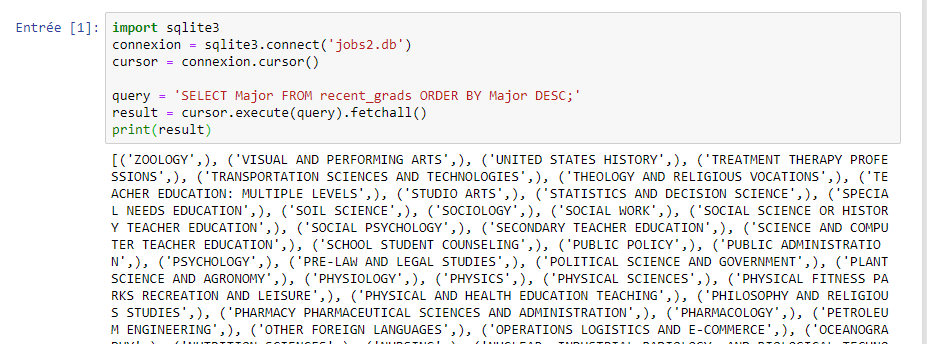
SQLite restreint l’accès à la bdd. Ce qui oblige à stopper la connexion quand on a fini de travailler. Ce qui permet à une autre personne de se connecter à la bdd.

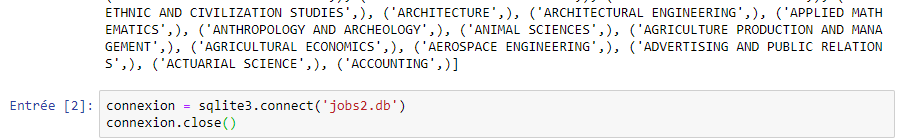


CHALLENGE 2 :



* Connectez-vous à la bdd jobs2.db
* Ecrire et exécuter une requête qui retourne toutes les majors dans l’ordre alphabétique inverse.
* Assigner le résultat complet à la variable result
* Enfin, fermer la connexion à la base de données.





Possible de fermer la connexion à la suite dans la même entrée 😉

# SECTION 5 – PROJET GUIDE : BASE DE DONNEES SQLITE (Bonus PYTHON)

##### **Présentation projet**

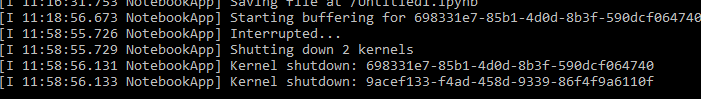
AIMS : Montrer ce qui est possible de faire avec une bdd en SQL, Python et la cmd SQLite schell.

On va travailler sur la bdd World Trade Book, qui contient des data sur la démographie de 267 pays dans le monde.

Les colonnes que nous allons le plus utiliser :

* population : nombre d’habitant en 2015
* population\_growth : le taux annuel de croissance de la population en %
* area : surface totale de chaque pays

##### **Présentation de la commande SQLite et exécution de requêtes SQL**

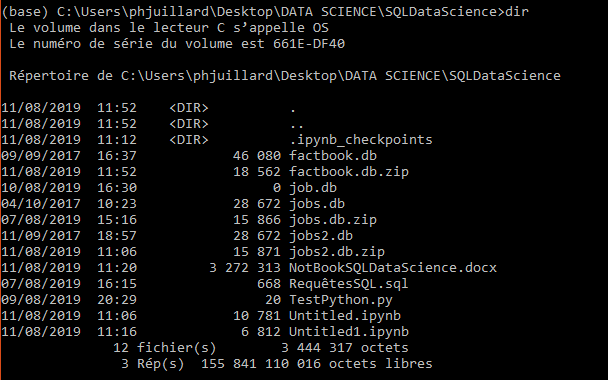


On s’aperçoit que NoteBookApp fonctionne toujours en arrière-plan. Donc Ctrl+C

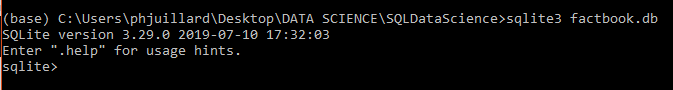


Puis ENTER

On vérifie dans le dossier SQLDataScience, que notre BDD factbook.db est là avec la cmd dir

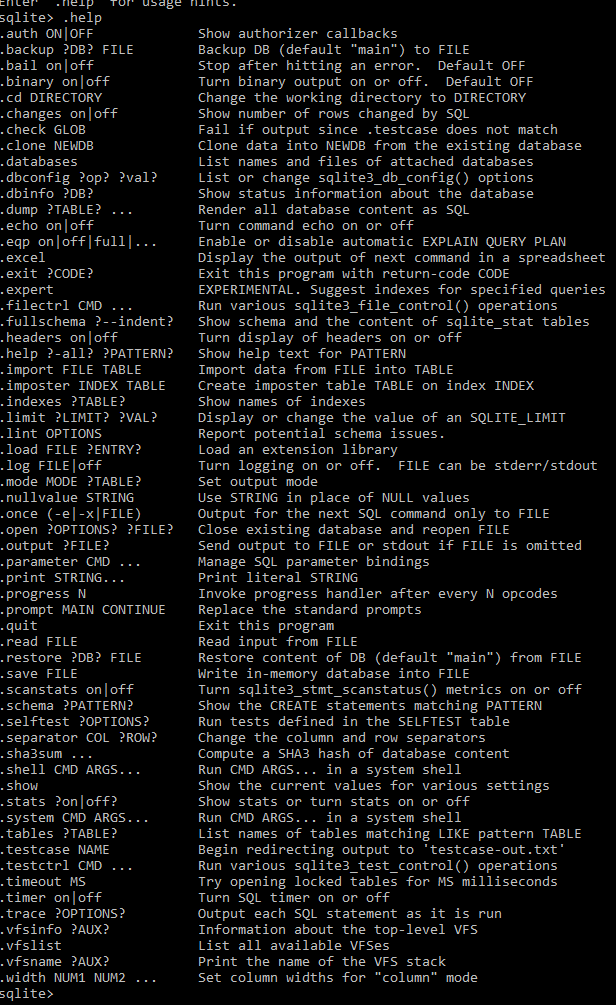


A partir de là, on peut taper la cmd sqlite3 +factbook.db , le nom de notre fichier BDD :



Donc là je suis directement connecté à la bdd factbook.db depuis la cmd sqlite.

On peut taper la cmd .help pour avoir accès à une liste de cmd 👓



Il y a aussi la cmd .tables qui est sensée afficher la liste des tables présentes dans la BDD.



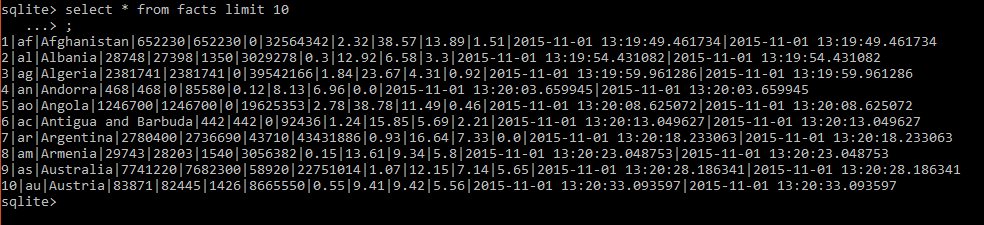
On peut voir qu’il y a 2 tables dans la BDD, facts et landmarks.

On va taper une requête SQL sur la cmd sqlite>

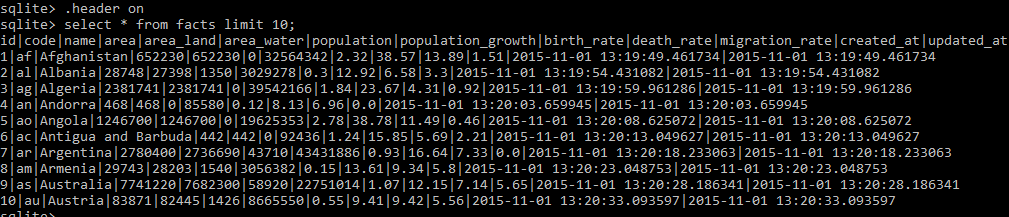
Sélectionner toutes les colonnes de la tables facts, limitant à 10 résultats



🧐 **Tant que je n’ai pas taper le ; de fin de requête, je peux aller à la ligne pour écrire ma requête**

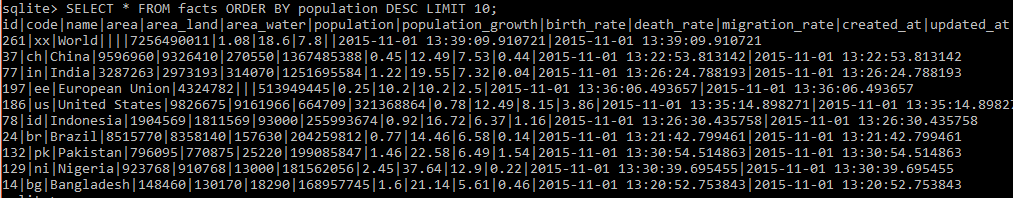


Pour formaliser l’affichage de la data obtenue, on peut utiliser la cmd .header on pour afficher les titre des colonnes.

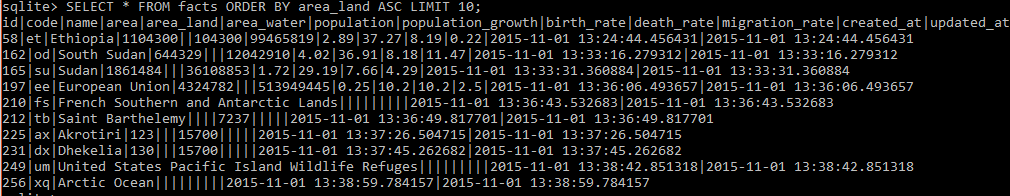


Exemples de requêtes :





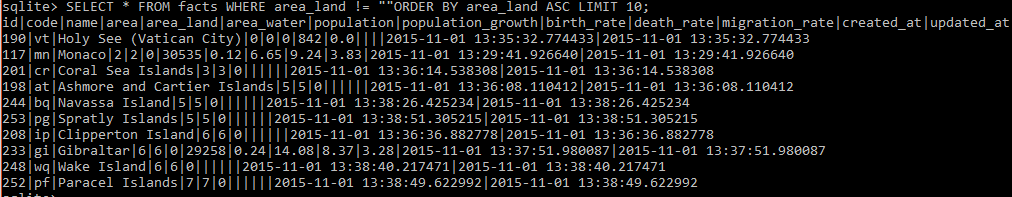




* Dans cet exemple, les datas sont erronées. Des Pays qui ne sont pas des pays dans la colonne Name. Des données manquantes. Les données sont en désordre.

Donc pour *cleaner* nos data, on rajoute un filtre WHERE à notre requête par exemple, en filtrant tous les pays qui n’ont aucune valeur sur la surface terrestre, colonne area\_land :





Pour quitter la bbd via SQLite, cmd .quit



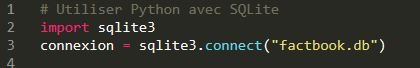
Retour dans la cmd classique.

##### **Utiliser Python avec SQLite**

Création du fichier query.py



On se connecte à la BDD factbook.db

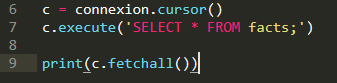


On créé un curseur c avec la mthd cursor() , on a une instance du curseur c



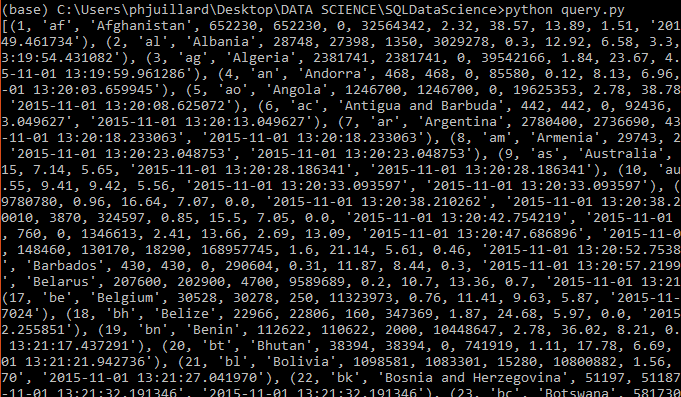
On exécute une commande avec la mthd execute() , soit comme d’habitude avec la création d’une variable query et au sein de cette query, on a écrit une requête SQL en string.

Mais on peut très bien écrire de suite la requête en string entre ‘’ avec la mthd execute()



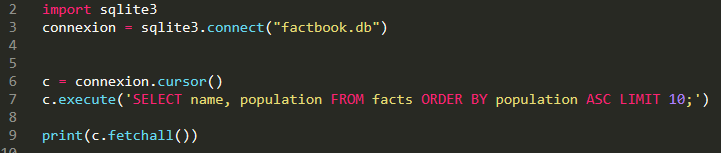
Dans la cd, on appel le fichier python query.py avec la cmd python+nom du fichier que l’on vient de sauvegarder.

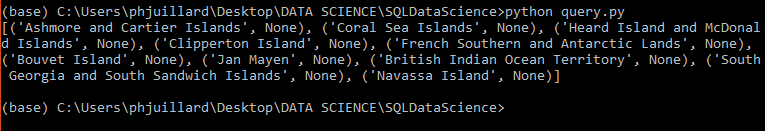


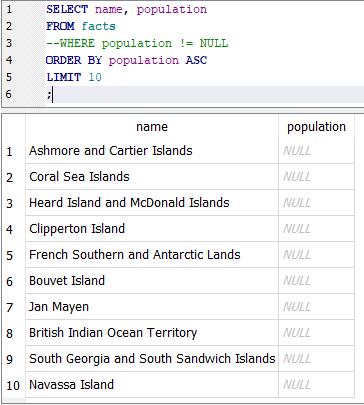
🤮🤢

EXERCICE :

* Modifier le code dans query.py afin que celui-ci sélectionne les 10 pays les moins peuplés de la table facts, et afficher le résultat.
* Exécuter query.py à partir de la commande python query.py



 👌👌🏿👌🏻



##### **Calculer des prévisions démographiques**

Comment calculer une estimation de la population en 2050 ?

2 choses pour çà :

* On va utiliser la librairie Pandas
* On va écrire une fonction python

AIMS du projet : Montrer ce qu’il est possible de faire à partir d’une BDD. En manipulant les données avec Python.

On crée un fichier python growth.py 

Ce qui nous permet d’utiliser pd lorsque l’on utilise une fonction de la librairie pandas.

On importe la librairie Pandas et on lui affecte une variable pd



L’importation de sqlite car on va en avoir besoin



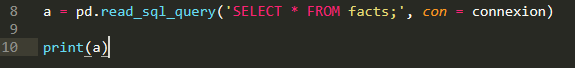
Instance de connexion

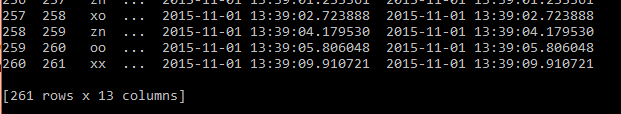
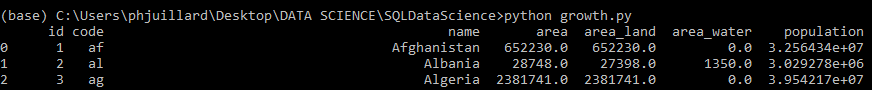


Appel à une nouvelle fonction read-sql-query , qui une fonction de la librairie Pandas. Qui nous permet de lier nos données dans un *data friends* (une sorte de tableau), en créant la variable a.

Cette fonction va prendre 2 paramètres :

* Le 1er - la requête SQL
* Le 2d - l’objet de connexion à la BDD, soit l’instance connexion via con





*Mthd* ***dropna()*** *qui supprime la ligne si il y a une valeur manquante dans une des colonnes associées*.

OBJECTIF : Avoir une estimation de la population pour chaque pays en 2050

On a le nombre d’habitants dans chaque pays actuellement ainsi qu’un taux de croissance.

D’après la fonction mathématique 

N =population finale en 2050

N0 = population initiale en 2015

e = constante, obtenue dans la librairie math d’où l’import dans growth.py 

r = taux de croissance annuelle en décimal

t = nombre d’année entre l’estimation et l’initiale

Exemple : Pays de 5000 Habitants, taux de croissance t de 4% (0,04), estimation de 2015 🡪 2050, donc t = 35

N = 5000 × e\*\*(0,04 × 35)

Donc on définit (traduit) cette fonction mathématique en python, suivre la syntaxe :



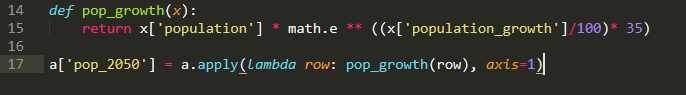
On nome cette fonction math pop\_growth()

x = paramètre d’entrée de la fonction

Donc on va appliquer à la place de x, tous les pays du fichier, càd chaque ligne on va lui appliquer cette fonction mathématique.

Et pour appliquer la même fonction à chaque lignes d’un tableau d’un dataframe Pandas, on utilise la mthd apply().

On sort de la fct pop\_growth()

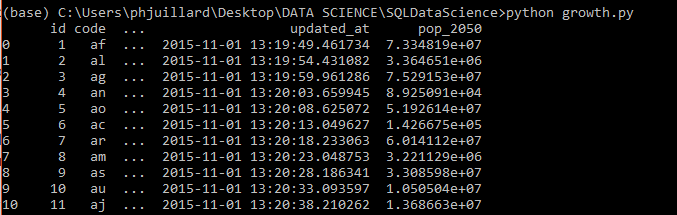


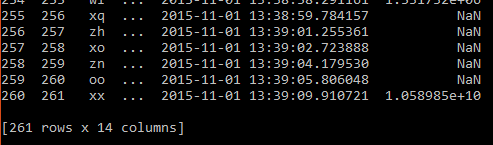
On crée une nouvelle colonne nommée pop\_2050 dans notre valeur a, qui est le tableau que nous avons créé tt à l’heure ligne 8 (p44).

= a auquel on applique la mthd apply() et pour cette colonne on applique la fonction python pop\_growth() que nous avons créé ligne 14, à chaque ligne de notre BDD

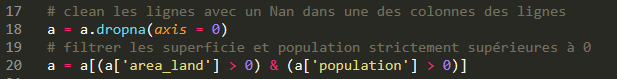


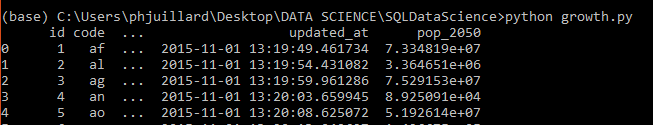
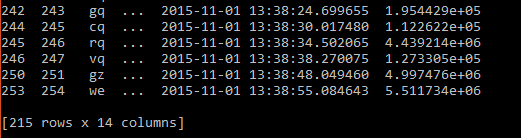
Affichage de a





*Mthd dropna() qui supprime la ligne si il y a une valeur manquante (NaN) dans une des colonnes associées*.



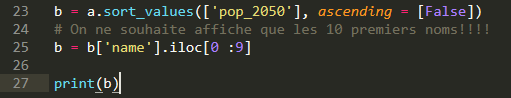
 

**Notre conclusion va être d’afficher les noms des 10 pays qui seront les plus peuplé en 2050.**

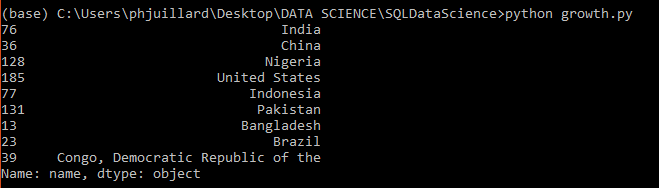
* A titre indicatif, il va falloir trier du plus grand au plus petit, il va falloir afficher les 10 plus grands.

On appel une nouvelle variable b, et l’on va utiliser la mthd sort\_values() qui permet de trier la colonne pop\_2050 de a , en triant du plus grand au plus petit via ascending = [**F**alse], soit l’ordre decroissant, pas ascending 😉

On affiche pour b, uniquement les noms des pays avec la mthd iloc[0 : n] , spécifique à Pandas, qui permet de sortir (afficher) tous les éléments de 0 à n voulu. Ici 10, donc de 0 à 9.

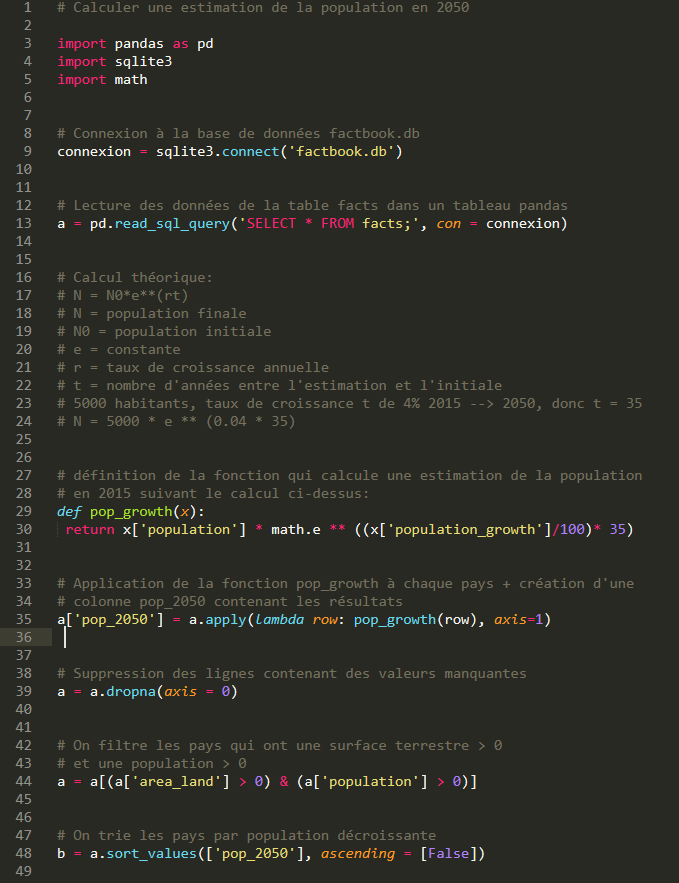
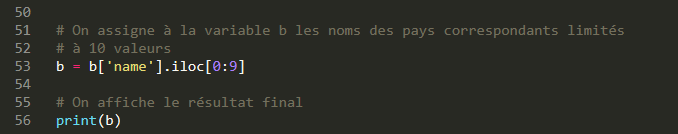


Les 10 pays estimés les plus peuplés en 2050 seront : L’inde dépasse la chine si le taux de croissance qui reste constant durant 35 ans.



🙆‍♂️🙆‍♀️🙆🏿‍♀️🙆🏾‍♂️

##### **Code source : Prévisions démographiques de la section précédente**

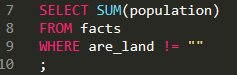
 

##### **Sommer des lignes d’une colonne pour calculer la surface totale**

La fonction SUM de SQL permet d’additionner toutes les valeurs présentes dans une colonne.



Là, on rajoute une condition WHERE pour retirer les informations manquantes dans les colonnes.

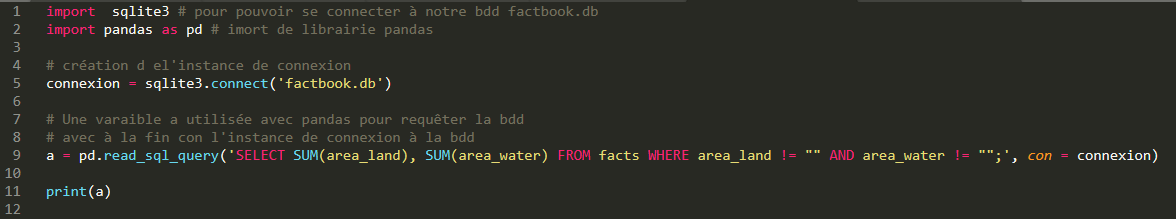
Exercice :

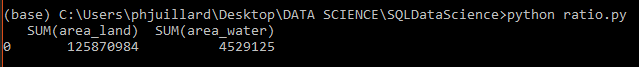
* Calculer le ratio de la surface terrestre totale sur la surface des océans totale appartenant aux pays de la table facts.
* Il est conseillé de créer un fichier python nommé ratio.py 

On va d’abord récupérer le totale de la surface terrestre de la colonne area\_lands

Ensuite on récupère le totale de la surface eau de la colonne area\_water

Nous diviserons les 2 sommes et afficherons le résultat







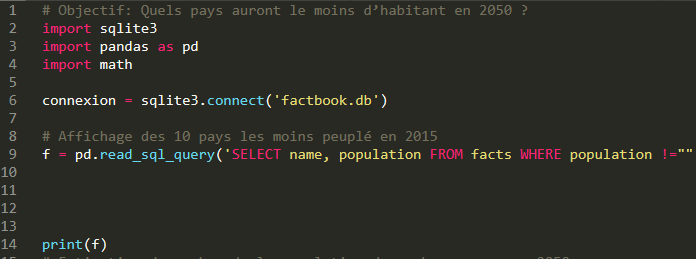


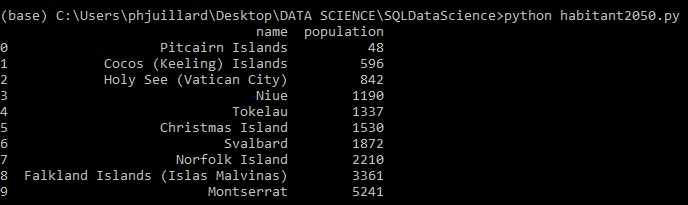
Ca signifie que sur l’ensemble des pays qui possèdent de la terre et de la mer, on a environ 28 plus de terre que de mer.

A investiguer pour s’exercer :

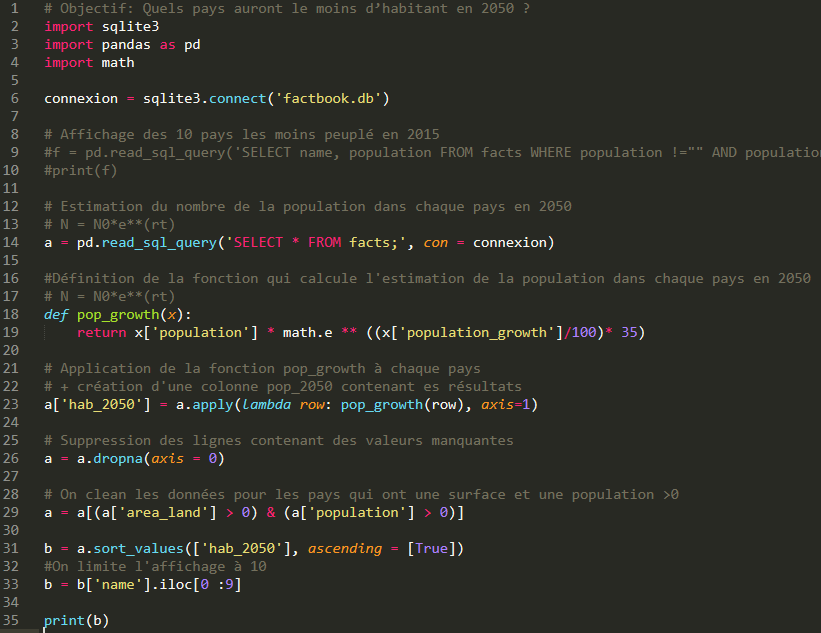
* Quels pays auront le moins d’habitant en 2050 ?





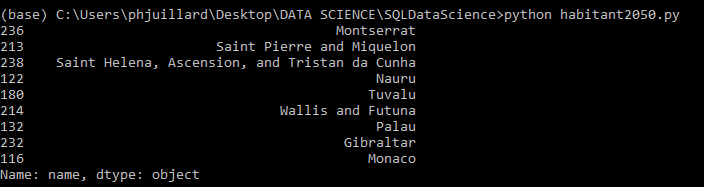


Puis, estimation de la population en 2050 avec la colonne hab\_2050

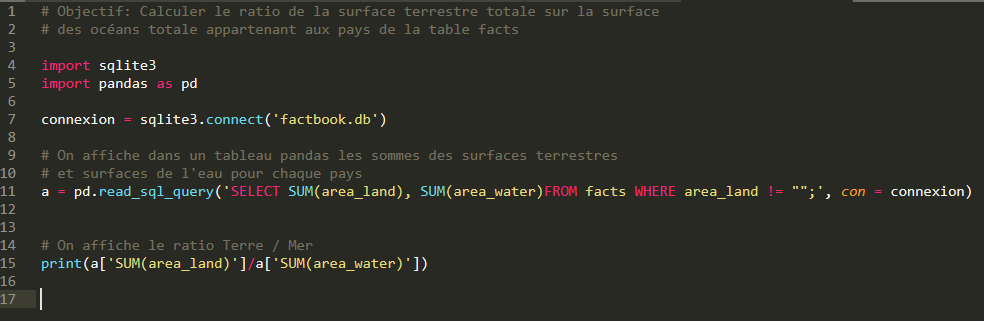


Différence avec l’exercice

Les 10 pays estimés les moins peuplés en 2050 seront : Montserrat passe de la 10eme à la 1ere place si le taux de croissance reste constant durant 35 ans.



##### **Code source : Ratio surface**



# SECTION 6 – CALCUL DE STATISTIQUES EN SQL

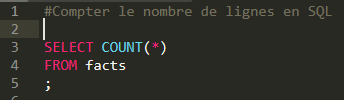
##### **Introduction**

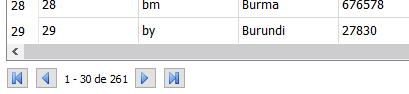
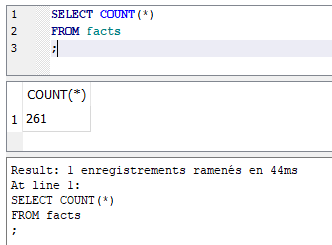
Type de calculs :

* Moyenne
* Somme
* Min / Max
* Retirer les doublons et afficher les valeurs uniques d’une colonne

##### **Compter le nombre de lignes en SQL**

Utilisation de la fonction SQL COUNT() et entre () on indique la/les colonne(s) où l’on souhaite compter le nombre d’éléments présent (lignes).

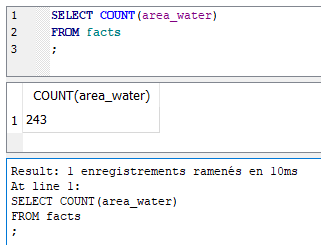




On dénombre 261 lignes dans la tables facts.

Maintenant on souhaite compter le nombre d’éléments non null dans une colonne de la table facts, comme area\_water :



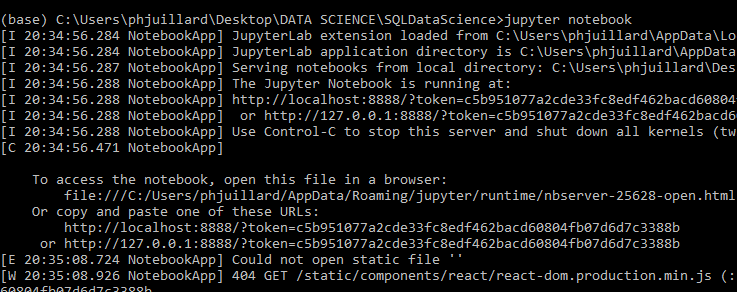


Maintenant la même chose avec Python 🐍🌌📙 :

Avec Python, on va calculer les valeurs non nulls d’une colonne.

Seulement attention, cela va nous retourner une liste avec un seul tuple. Donc pour extraire la valeur qui nous intéresse, à savoir le nombre de valeur non nulls, il va falloir extraire le 1er élément du 1er tuple des résultats.

Retour Jupyter Notebook

 **+ open lien!**

Mission : Compter le nombre de valeur non nulls de la colonne birth\_rates de la table facts :

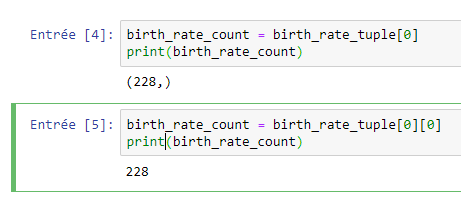


[(228,)] >> Donc c’est bien une liste syntaxe, et dans cette liste il y a un tuple, et nous la valeur qui nous intéresse est la première valeur de ce tuple.

Pour améliorer l’affichage, plus clean, on va créer une nouvelle variable birth\_rate\_count :

(Le taux de naissance qui sont non null comptés dans la bdd)

Donc on récupère le résultat 228 via la variable birth\_rate\_tuple (qui est une liste contenant 1 tuple), et pour récupérer le 1er tuple de cette liste on ajoute **[**0**]**

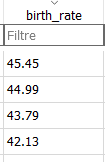
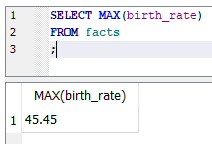


On obtient bien 228 clean, à savoir le nombre de valeurs non null dans la table birth\_rate.

##### **Trouver les valeurs minimales et maximales d’une colonne en SQL**

**La fonction COUNT() peut être utilisée dans n’importe quelle colonne quelque soit le format des données présentes (string, int, decimal…).**

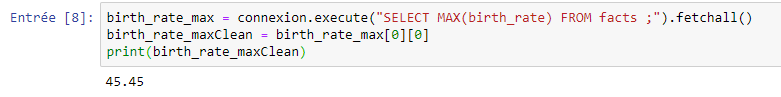
**Les fonctions MIN() et MAX() ne peuvent être utilisées que dans les colonnes de données numérique (int, decimal, …). Elles déterminent la valeur max ou min.**





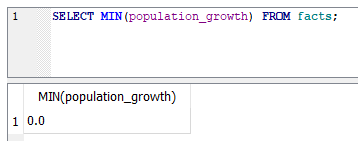
**Penser à l’import sqlite3 et à la connexion à la bdd dans ces codes !**

Maintenant la même chose avec Python 🐍🌌📙 :

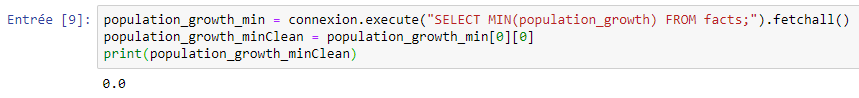


Exercices : Travail à réaliser sur DB puis sur Jupyter Notebook

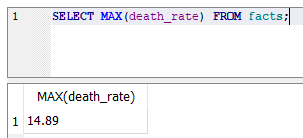
* Utiliser la fonction MIN sur la colonne population\_growth.



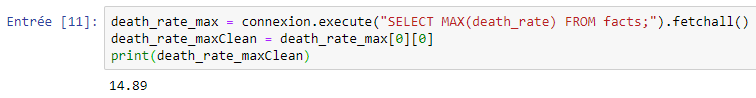
Maintenant la même chose avec Python 🐍🌌📙 :



* Utiliser la fonction MAX sur la colonne death\_rate.

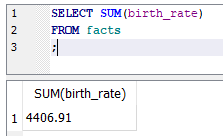


Maintenant la même chose avec Python 🐍🌌📙 :

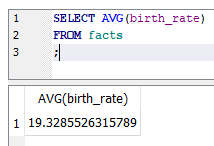


##### **Calculer des sommes et des moyennes en SQL**

La fonction SQL SUM() permet de calculer le totale de toutes les valeurs non null d’une/des colonne(s) numérique(s).

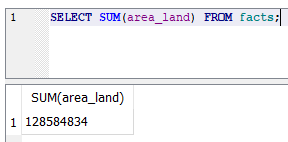


La fonction SQL AVG() permet de calculer la moyenne (average) de toutes les valeurs non null d’une/des colonne(s) numérique(s).



Exercices : Travail à réaliser sur DB puis sur Jupyter Notebook

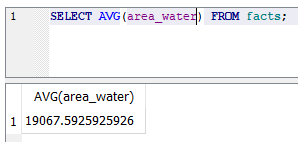
* Utiliser la fonction SUM pour trouver le total de la colonne area\_land



Maintenant la même chose avec Python 🐍🌌📙 :



* Utiliser la fonction AVG pour trouver le total de la colonne area\_water





**Penser à fermer la connexion à la bdd après utilisation!**

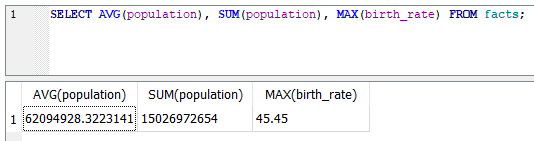


##### **Combiner plusieurs fonctions**

Ecrire une requête qui calcule :

* La moyenne de la colonne population
* La somme de la colonne population
* Le maximum de la colonne birth\_rate





Maintenant la même chose avec Python 🐍🌌📙 :

NB – Afficher les 3 résultats que vous assignerez à 3 variables différentes

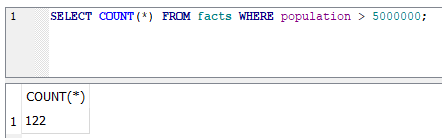


##### **Associer les fonctions de calcul avec la condition WHERE**

L’objectif de cette section est de montrer que l’on peut utiliser les fonctions de calculs (SUM, COUNT, AVG, MAX, MIN) avec une condition WHERE.

Cà va nous permettre de calculer des statistiques sur des ensembles plus réduit.

Afficher ttes les colonnes où la population est sup à 5 millions :

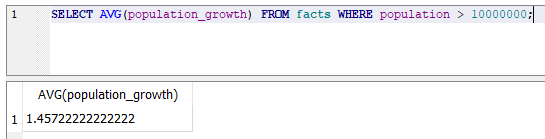


Il y a 122 pays en bdd, dont la population est supérieure à 5 millions d’habitants.

Exercice :

Ecrire une requête SQL qui calcule la moyenne de la colonne population\_growth pour les pays ayant une population supérieure à 10 millions d’habitants.

Analyse en BDD et avec Python



Maintenant la même chose avec Python 🐍🌌📙 :



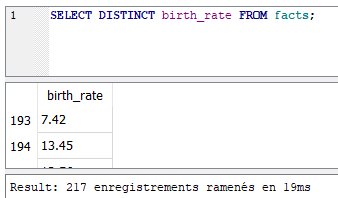
##### **Sélectionner des éléments ou données uniques**

Comment afficher une valeur unique dans une colonne aux valeurs identiques ? Eviter les doublons.

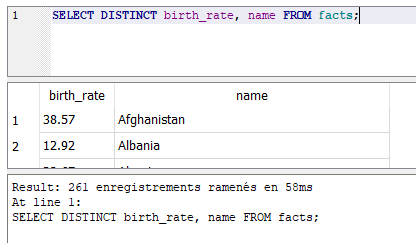
On utilise la commande DISTINCT()

* Requête qui supprime les valeurs identiques de la colonne birth\_rate



 pour 261 lignes totales 😉

A noter que si on associe 2 colonnes en recherche, il faut que les valeurs soient identiques dans les 2 colonnes dans les même lignes !



Car pas 2 pays avec même nom 😉

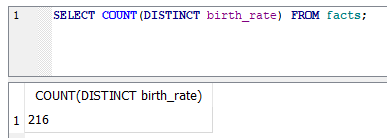
Maintenant la même chose avec Python 🐍🌌📙 :



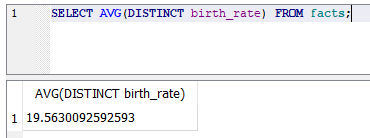
##### **Associer les fonctions de calcul avec DISTINCT**

Si nous souhaitons compter le nombre d’objets uniques dans la colonne population, nous pourrions associer la fonction COUNT() avec la commande DISTINCT () .

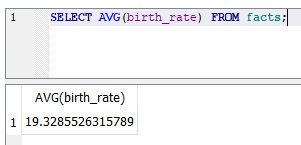






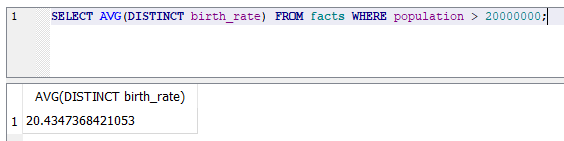


Et sans DISTINCT :



Exercice :

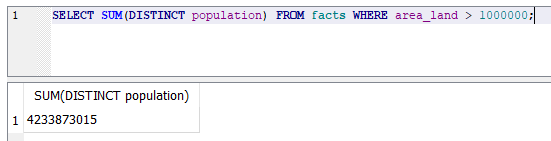
* Trouver la moyenne des valeurs distinctes de la colonne birth\_rate pour lesquelles population est plus grand que 20 millions.



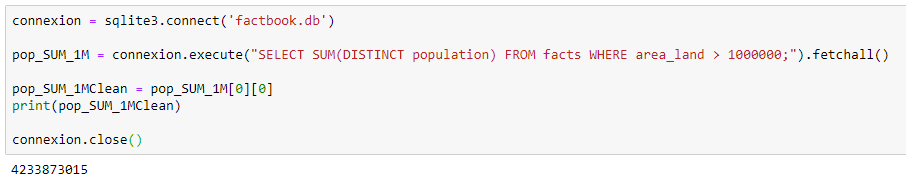
Maintenant la même chose avec Python 🐍🌌📙 :



* Trouver la somme de toutes les valeurs distinctes de la colonne population pour lesquelles area\_land est plus grand que 1 million



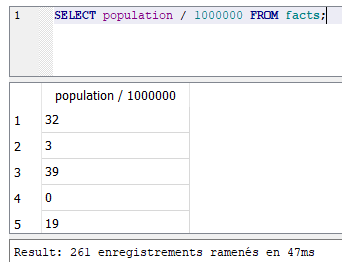
Maintenant la même chose avec Python 🐍🌌📙 :



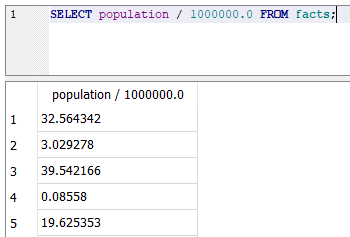
##### **Opérations mathématiques en SQL**

Parfois il peut être intéressant de faire des opérations mathématiques sur les colonnes d’une tables, le tout en SQL, par exemple, si nous souhaitons exprimer la colonne population en million, nous pouvons diviser la colonne par le nombre 1 000 000, ce qui simplifiera la lecture de la data.



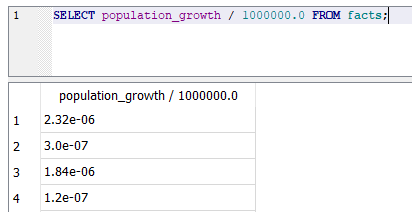


Nous avons des valeurs à 0, ce qui implique que la population est < 1M. Hors il aurait pu être intéressant d’avoir les chiffres après la virgule. Le plus simple est de rajouter 1000000.0 . La colonne population passe d’un format de valeur d’entier à decimal.



Exercices :

* Afficher les valeurs de la colonne population\_growth en millions (décimales)



Maintenant la même chose avec Python 🐍🌌📙 :



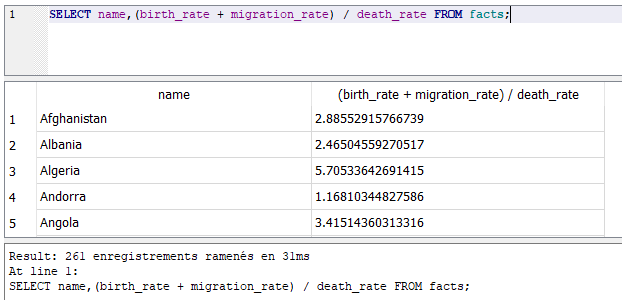
##### **Exécuter des opérations mathématiques entre colonnes**

Comment faire des opérations entre colonnes ? Par exemple, exprimer le taux de naissance divisé par le taux de décès.



Mais rajoutons de la pertinence à nos résultats en ajoutant le taux de migration dans chaque pays pour exprimer la croissance ou la décroissance des pays :





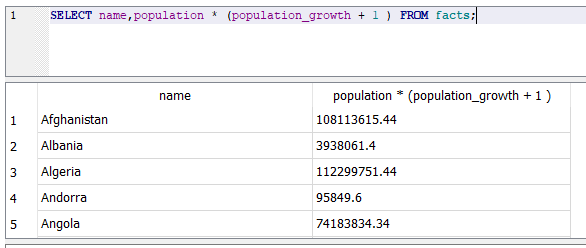
Un résultat supérieure à 1 signifie que les naissance et les flux migratoire couvrent les décès dans le pays. Du coup, la population à tendance à croitre.

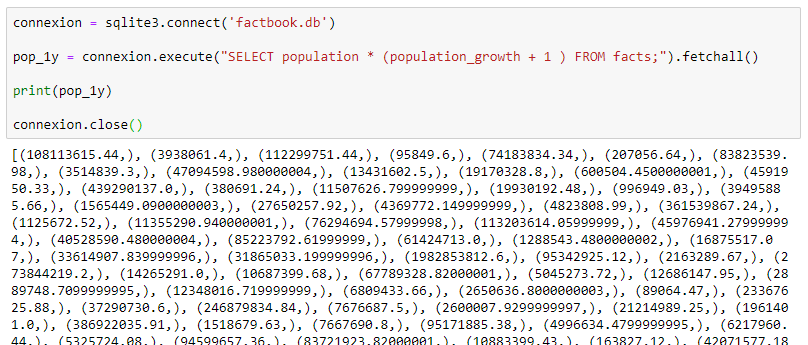
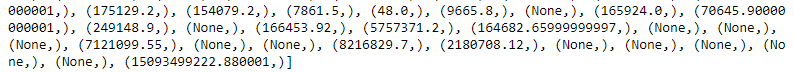
Si le résultat est inférieur à 1, c’est l’effet inverse, les décès sont plus nombreux par rapport aux naissance et aux flux migratoires, la population décroit.



Exercice :

* Ecrire une requête SQL qui calcule pour chaque pays le nombre d’habitants qu’il y aura l’année suivante.
* Indice : Multiplier les colonnes population et population\_growth pour avoir l’augmentation sur 1an et ajouter la colonne population au résultat.



# SECTION 7 – STATISTIQUES SUR DES GROUPES DE DONNEES

##### **Introduction**

Comment grouper des éléments et effectuer des stats sur ces groupes ?

Dans cette section, on se concentre sur les requête SQL sans passer par Python/Jupyter Notebook.

On utilisera la BDD jobs.db

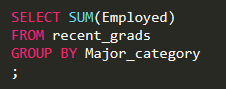
Dans cette section, à l’aide des groupes que l’on va former, vont nous permettre de répondre à ces 3 questions :

* Quelle est la proportion de femmes dans chaque catégorie de matière principales ou majors ?
* Quelle catégorie de majors a le plus grand nombre d’employés diplômés ?
* Quel est le pourcentage de personnes dans chaque catégorie de majors finissant avec des emplois à faible revenu ?

##### **Calculer des statistiques en regroupant des éléments par niveau**

La commande SQL GROUP BY() permet de faire des statistiques sur des groupes.

Quand on l’utilise, SQL créé un groupe pour chaque valeur unique de la colonne sélectionnée auprès de GROUP BY()



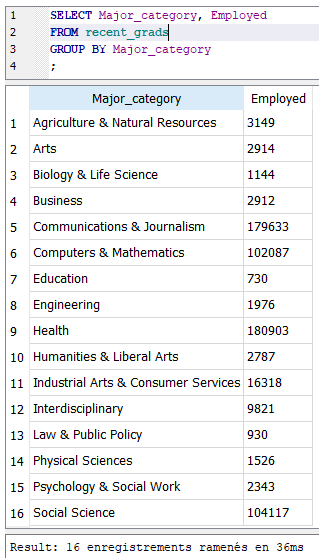


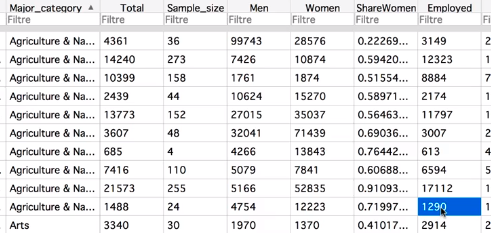
Ca nous retourne 16 résultats. Et dans chacun de ces résultats, il y a la somme des diplômés qui ont un emploi.

Cette requête regroupe par nom distinct et va nous permettre de faire des calculs sur les autres colonnes, ici sur la colonne Employed.

On va grouper toutes les Major\_category, et nous allons pouvoir exécuter une fonction SQL, SUM() par exemple, sur une autre colonne. Finalement cette fonction va calculer que sur les éléments du groupe.

Par exemple, si j’oublie la fonction SUM() dans la requête précédente, il se passe quoi ?



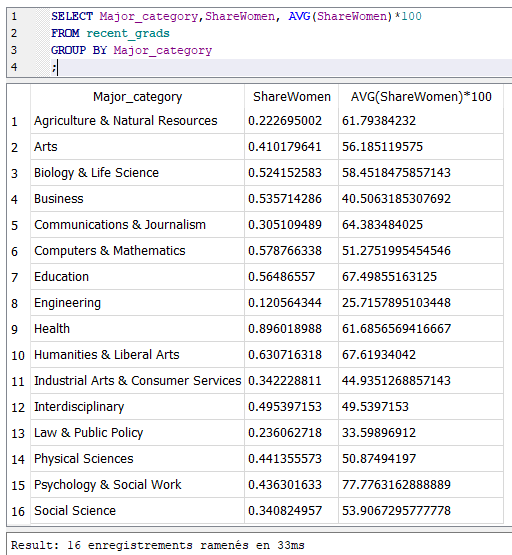
Si je ne fais pas de fonction SUM(), la valeur retenue dans le groupe est la première trouvée.

La commande GROUP BY() touche chaque colonne, ce qui nous permet de faire des stats après la commande SELECT.

Exercice :

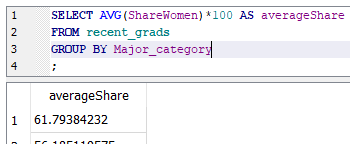
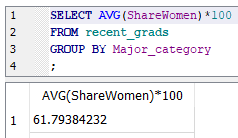
* Trouver le pourcentage moyen de femmes pour chaque catégorie de majors.
* Vous afficherez les colonnes Major\_category et le pourcentage moyen de femmes.

\*100 pour avoir un % qui s’affiche pour AVG(Sharewomen)



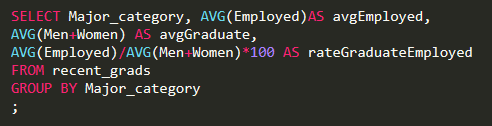
##### **Renommer les colonnes avec la commande AS**

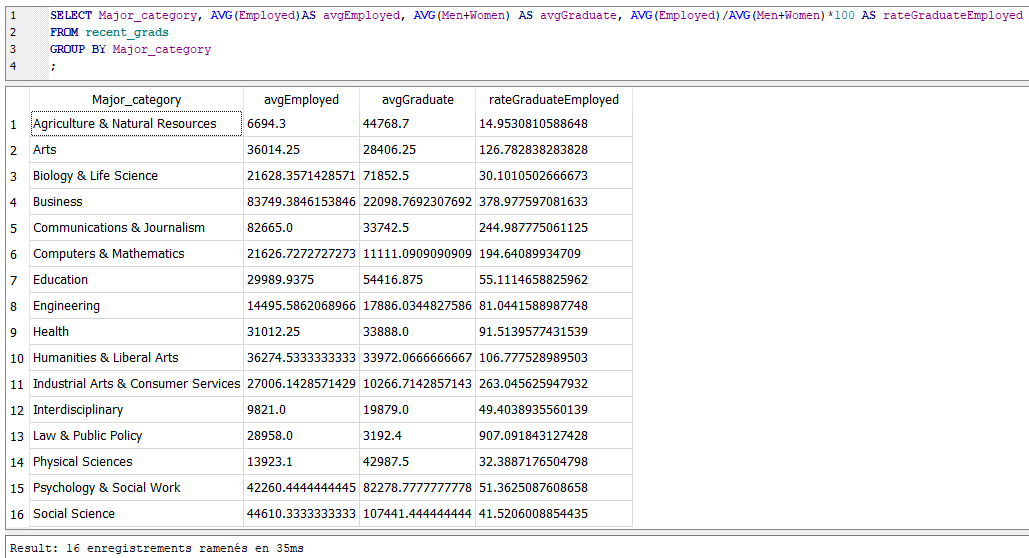
Utiliser la commande AS

##### **Pratique**

* Trouver pour chaque catégorie de major, le pourcentage de diplômés ayant un emploi.
* Modifier le nom de la colonne des résultats
* Indice : Le pourcentage de diplômés ayant un emploi se calcule en divisant la moyenne des diplômés ayant un emploi sur la moyenne totale des diplômés.

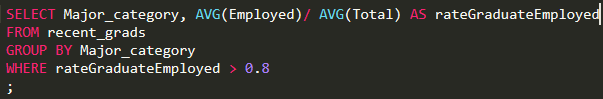




##### **Colonnes virtuelles et HAVING**

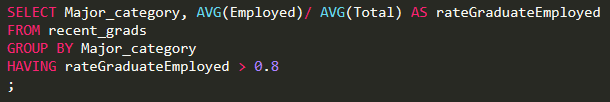
On souhaite filtrer sur la colonne rateGraduateEmployed, qui définis le % de diplomés employés pour chacune des 16 catégorie de Major.

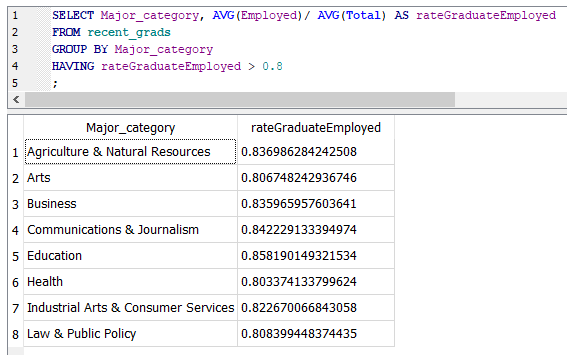
* On veut sélectionner/trier seulement les Major\_category dont le rateGraduateEmployed est supérieur à 80%, soit là où les dipomés ont le plus de chance de trouve run job avec le diplôme.



La commande WHERE ne fonctionne pas sur les colonnes virtuelles pour filtrer

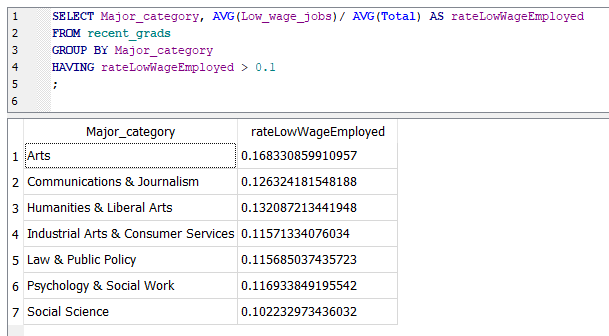
Nous devons utiliser la commande HAVING





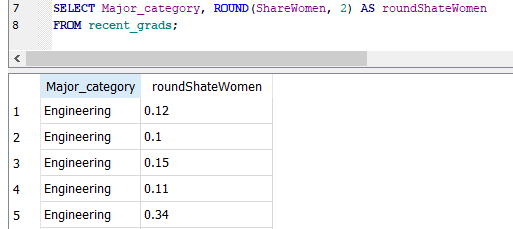
Exercice : 💀⚰️☠️

* Trouver toutes les catégories de major pour lesquelles la proportion de diplômés avec de faibles revenues est supérieur à 0.1.

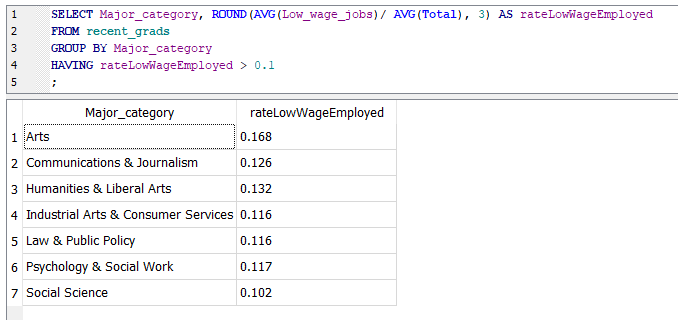


##### **Arrondir les résultats avec la fonction ROUND**

On va arrondir nos résultats de la requête précédente avec la fonction ROUND qui prend 2 paramètres, la colonne que l’on souhaite arrondir et le nombre d’élément après la virgule :

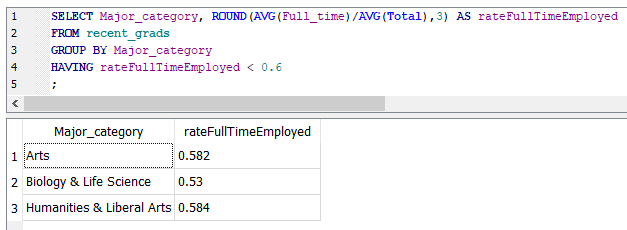


Pour les colonnes virtuelles comme dans le précédent chapitre :

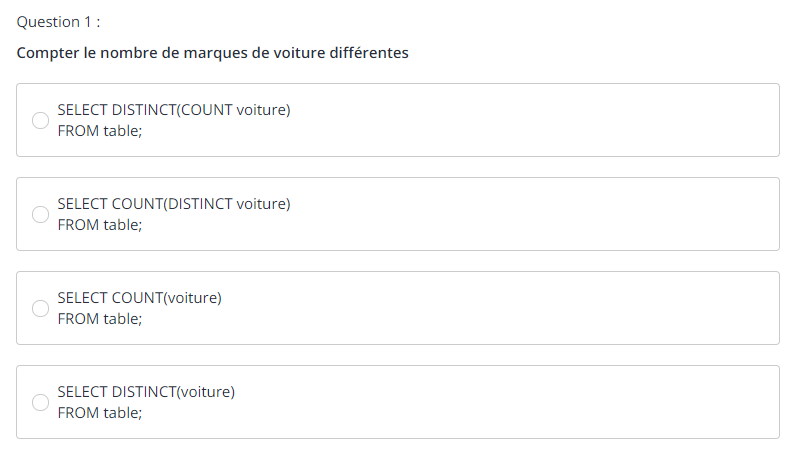


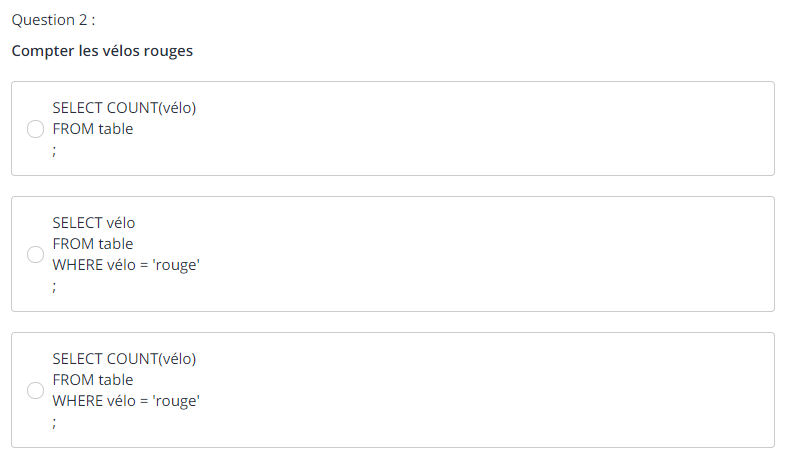
Exercice : 💩

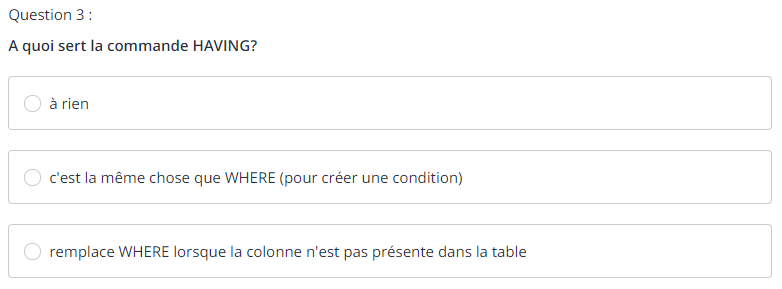
* Afficher le pourcentage de diplômés travaillant à temps plein (colonne Full\_time) pour chaque Major\_category
* Pour chaque catégorie de major, vous afficherez seulement les résultats inférieurs à 60%.



##### **Quiz**







* on l'utilise sur une colonne virtuelle créée par une autre commande (ex: GROUP BY)

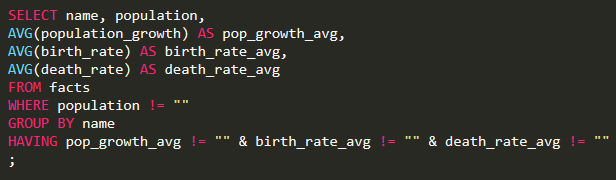
# SECTION 8 – CHALLENGE 2 : EXPLORATION DE DONNEES

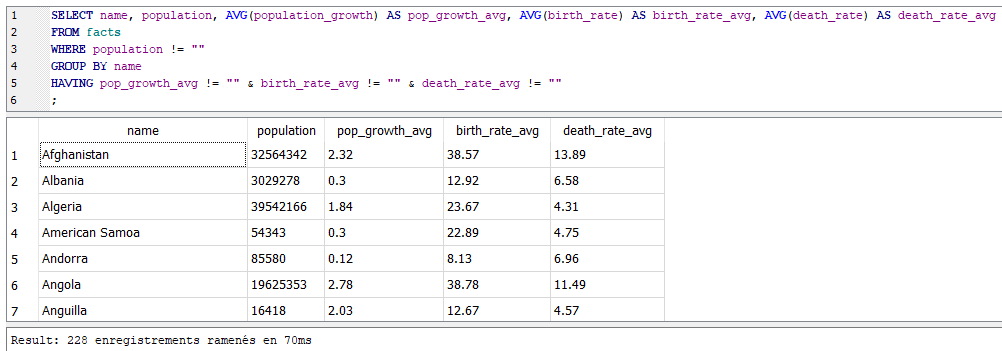
##### **Introduction**

* Pratique de calculs statistiques SQL sur la table facts de factbook.db
* Réalisation de projection démographique

##### **Calcul de moyennes**

* En SQL, calculer la somme des population et les moyennes des colonnes population\_growth, birth\_rate et death\_rate sur la table facts de la bdd factbook.db.
* Puis, en Python exécuter cette requête SQL.
* Enfin récupérer les résultats dans les variables pop\_sum, pop\_growth\_avg, birth\_rate\_avg et death\_rate\_avg







🤔 🤯

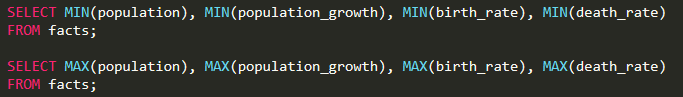
Correction :





##### **Trouver les valeurs extrêmes**

* Calculer les valeurs minimums et maximums des colonnes population, population\_growth, birth\_rate et death\_rate (2 requêtes SQL bien choisies)
* Assigner les bonnes valeurs aux variables pop\_min, pop\_max, pop\_growth\_min, pop\_growth\_max, birth\_rate\_min, birth\_rate\_max, death\_rate\_min, death\_rate\_max.





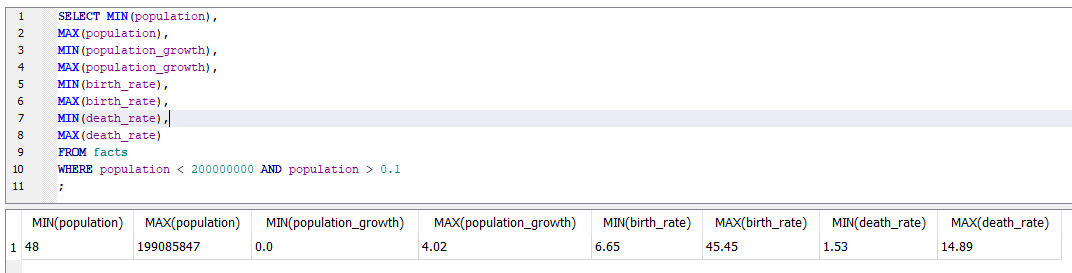
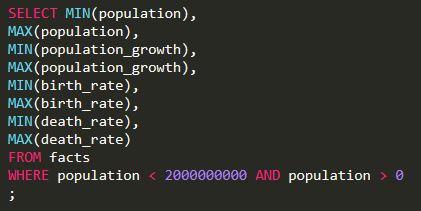


##### **Filtrer les valeurs**

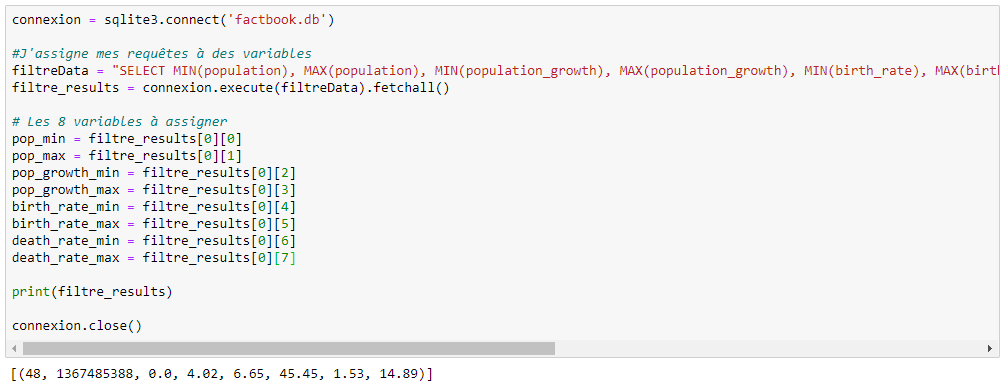


Il y a des données peu crédibles dans nos résultats, dû aux datas des entitées qui ne sont pas des pays comme dans la ligne world.

* Calculer les valeurs minimums et maximums des colonnes population (pour lesquelles la colonne population est inférieure à 2 milliards et supérieure à 0) : population\_growth, birth\_rate et death\_rate
* Assigner les bonnes valeurs aux variables pop\_min, pop\_max, pop\_growth\_min, pop\_growth\_max, birth\_rate\_min, birth\_rate\_max, death\_rate\_min, death\_rate\_max.



Maintenant avec Python par Jupyter Notebook 🐍🌌📙 :



##### **Prédire la future croissance démographique**

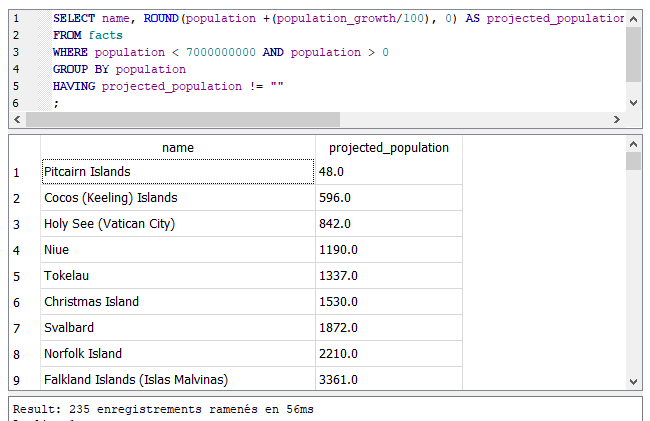
* Retourner les valeurs de la population estimée l’année suivante en prenant en compte les remarques suivantes :
* Arrondir les valeurs à l’entier près (valeurs non décimales)
* Filtrer les lignes qui contiennent des valeurs NULL comme valeur pour les colonnes population ou population\_growth(population is not null)
* Restreindre la requête aux pays ayant une population inférieure à 7 milliards et supérieur à 0.

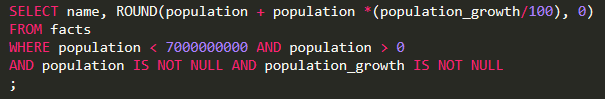
🐍🌌📙

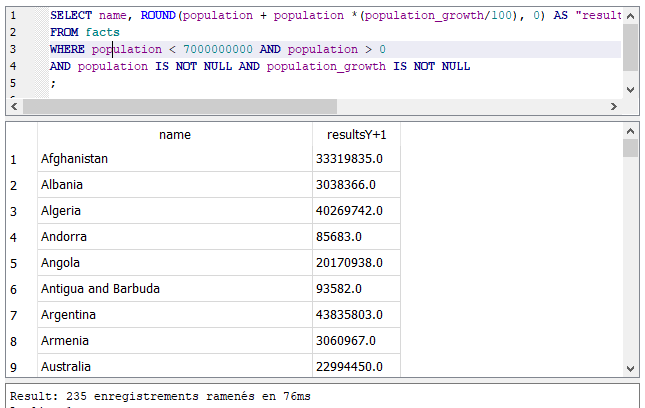
* Assigner les résultats à la variable projected\_population.
* Afficher les 10 premières valeurs du résultat.



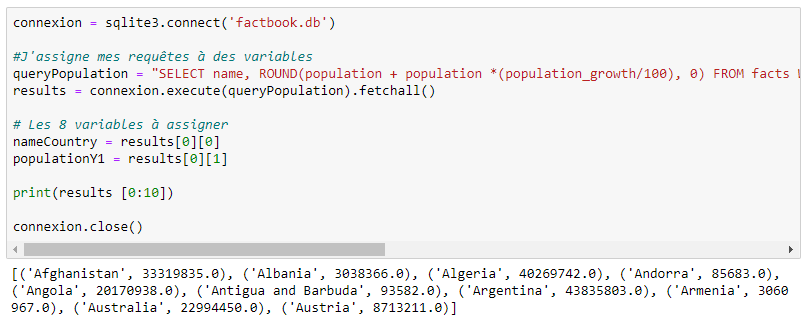








Maintenant avec Python par Jupyter Notebook 🐍🌌📙 :

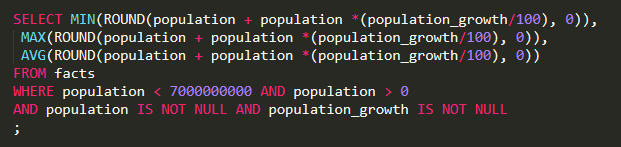


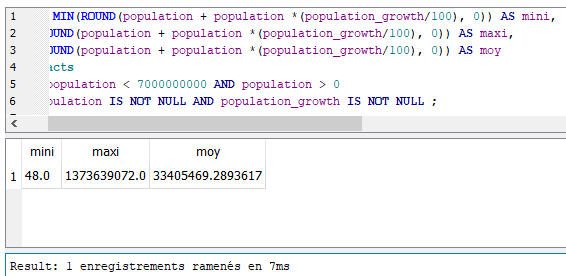
##### **Explorer plus sur les estimations démographiques**

* Ecrire une requête SQL qui retourne :
* La valeur min de l’estimation de la population et l’assigner à la variable pop\_proj\_min.
* La valeur max de l’estimation de la population et l’assigner à la variable pop\_proj\_max.
* La valeur moyenne de l’estimation de la population et l’assigner à la variable pop\_proj\_avg.

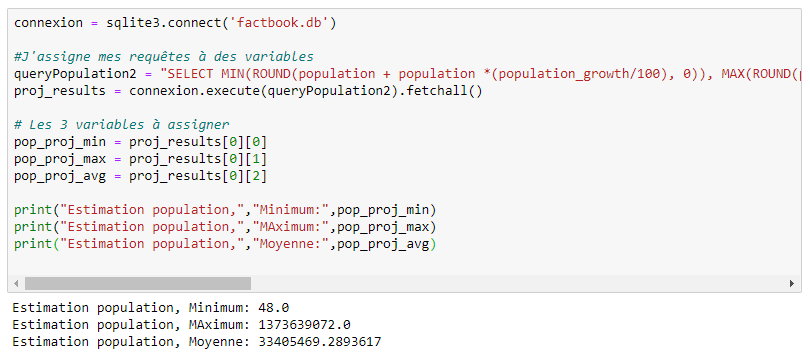
🐍🌌📙

* Afficher les résultats





Maintenant avec Python par Jupyter Notebook 🐍🌌📙 : **Mise en forme des résultats**



# SECTION 9 – MODIFIER LES DONNEES DANS UNE TABLE

##### **Introduction**

INSERT : ajouter des données dans une table

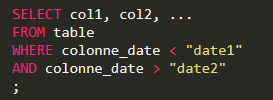
UPDATE : modifier des valeurs

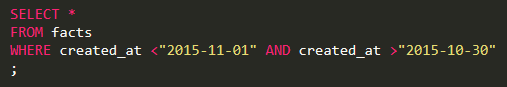
DELETE : supprime des données existantes

On utilisera la bdd factbook2.db



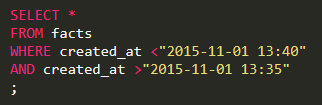
##### **Les dates en SQL**



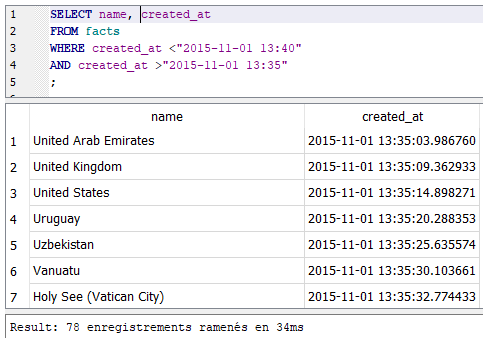


Ici SQL compare la colonne created\_at de la table facts à la date 2015-11-01, et il va sélectionner ttes les lignes pour lesquelles les valeurs de la colonne created\_at sont inférieur à cette date.

Ensuite on utilise l’opérateur AND pour créer une autre comparaison pour sélectionner les dates qui seront après le 2015-10-30.



Donc cette requête va sélectionner toutes les données qui ont été créées entre 2015-11-01 13:35 et le 2015-11-01 13:40.



##### **Les différents types de données sur SQLite**

INTEGER : Tous les nombres entiers

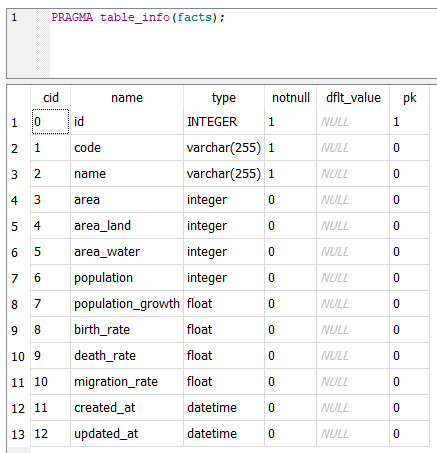
REAL et FLOAT : les nombres à virgule

TEXT et VARCHAR(255) : lignes de caractères

DATETIME : les dates

* Si je souhaite voir les types de données pour chaque colonne d’une table, utilise rla commande PRAGMA()





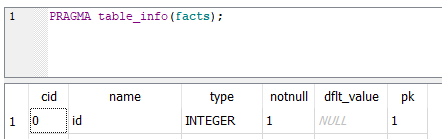
##### **Les clés primaires (Primary Key)**

Chaque table SQL possède une clé primaire.

Clé primaire : c’est une colonne où chaque valeur est unique.

La clé primaire, c’est le moyen qu’a SQL pour identifier chaque ligne.

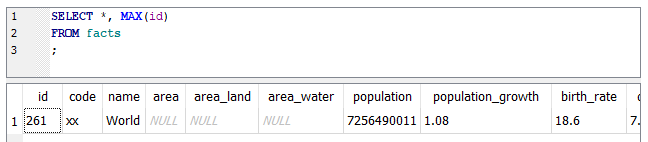
La plupart des tables on une colonne d’entier que l’on nomme ID par défaut, et cette colonne est une clé primaire.

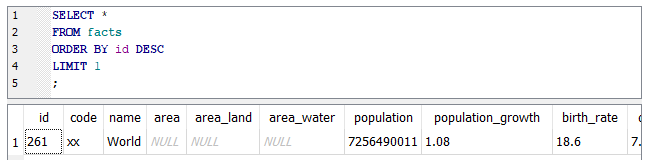


Cette colonne ID de type entier, est la clé primaire (pk=1(true)) de la table facts.

Exercice :

* Ecrire une requête SQL qi affiche toutes la ligne possédant l’id le plus grand.



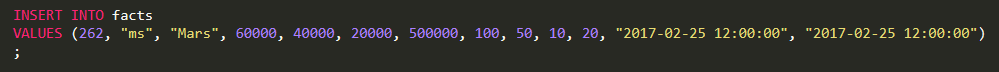


##### **Insérer des données dans une table**

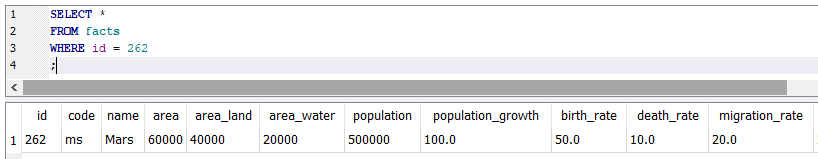
Si on souhaite ajouter des lignes dans une table QL, on utilise la commande INSERT INTO

Ainsi que la commande VALUES(toutes les valeurs que compte le nombre de colonne de la table)



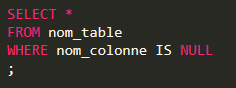


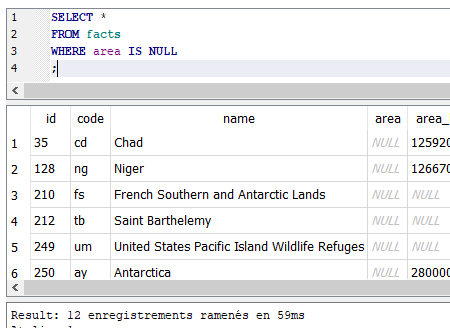




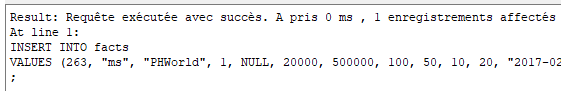
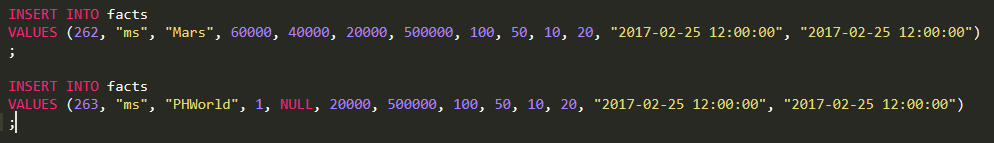
##### **Valeurs manquantes**

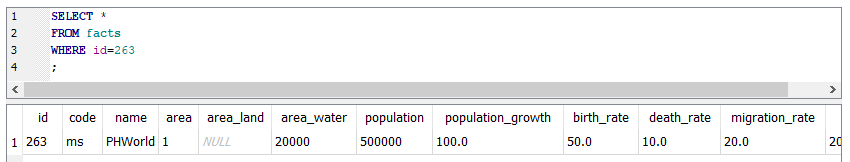
L’idée est que lorsque nous avons une valeur manquante dans une table, elle sera indiquée avec le mot clé NULL.





Lorsque l’on rentre des données en BDD, il peut arriver qu’il manque certaine données, du coup on remplace la donnée manquante par NULL, ici dans la colonne area\_land pour id 263 :

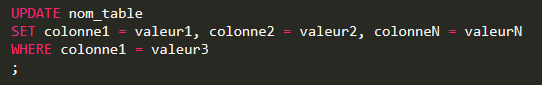




Certaines colonnes peuvent avoir une contrainte NOT NULL, qui spécifie quelles ne peuvent contenir aucune valeurs manquantes comme la colonne name..

##### **Modifier des valeurs**

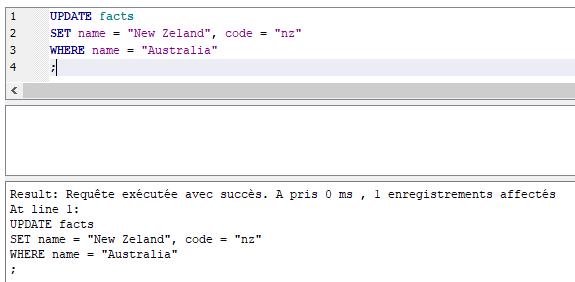
Utilisation de la commande UPDATE

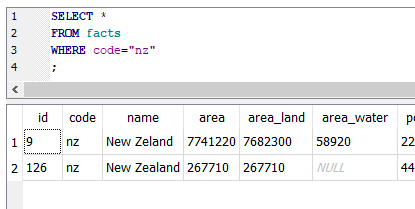


Le SET est pour ajouter/modifier une/es nouvelle(s) valeur(s) à la table.

La cmd WHERE où on spécifie quelle ligne on souhaite modifier.

Maintenant on va renommer le pays « Austalia » par « New Zeland » par exemple :

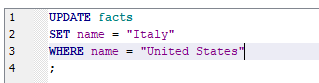


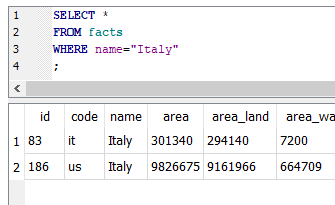


Possibilité d’affiner les modifications avec en addition du WHERE, des connecteur logique OR et AND par exemple.

Exercice :

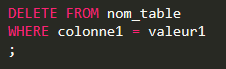
* Ecrire une requête qui modifie le nom United States par Italy :

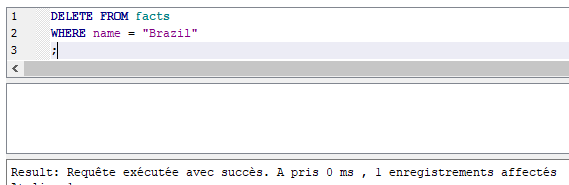




##### **Supprimer des valeurs**

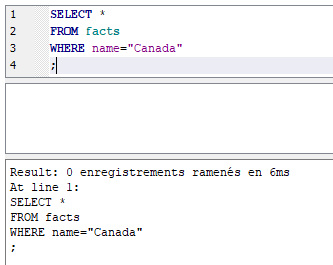
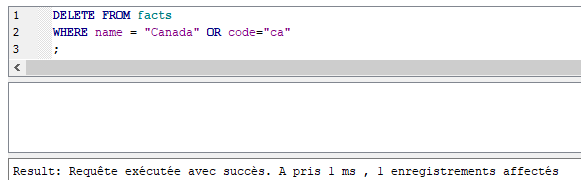
On utilise la commande DELETE FROM



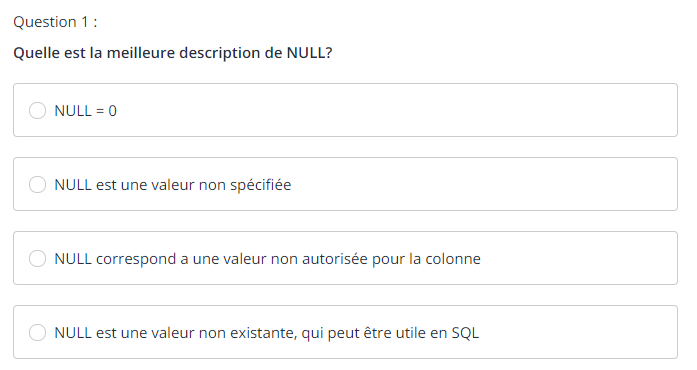


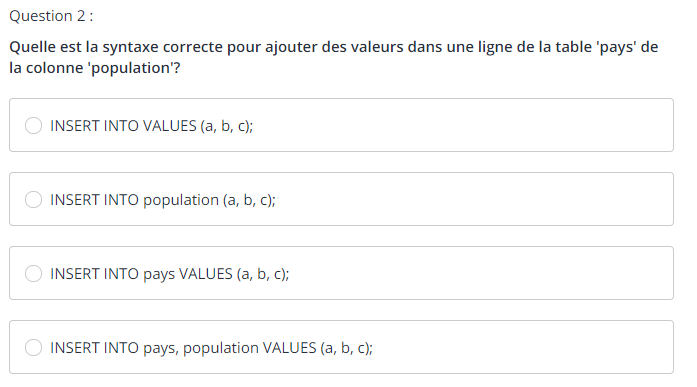
Exercice :

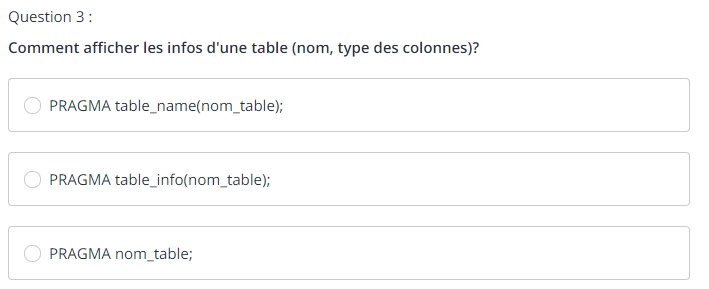
* Ecrire une requête SQL qui supprime toutes les lignes caractérisant le Canada



##### **Quiz3**







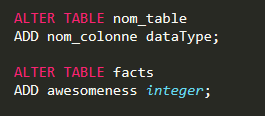
# SECTION 10 – SCHEMA D’UNE TABLE

##### **Introduction**

* Plan ou schéma d’une table
* Relation entre les tables
* Commande INNER JOIN

##### **Ajouter des colonnes**

Pour ajouter une colonne on utilise la commande ALTER TABLE+ADD



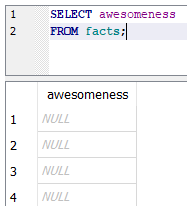
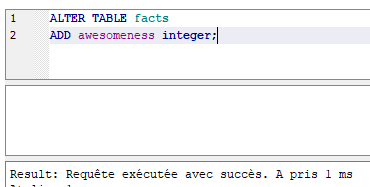
ALTER TABLE indique que nous allons effectuer un changement dans le schéma de table.

**facts** c’est la tables que nous allons modifier

ADD indique que nous ajoutons une colonne

**awesoeness** est le nom que l’on donne à cette colonne dont le type de données dans cette colonne est *integer* (int).

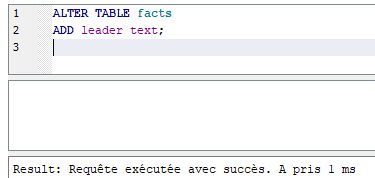
**Quand nous ajoutons une colonne, les valeurs associées ont la valeur NULL**



Exercice :

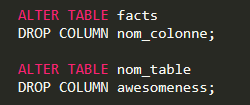
* Ecrire une requête SQL qui ajoute une colonne qu’on nommera leader à la table facts, avec comme type de données text.



##### **Supprimer des colonnes**

Pour supprimer une colonne on utilise la commande ALTER TABLE+DROP COLUMN



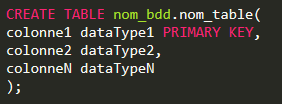
DROP COLUMN Nous indique que nous allons supprimer une colonne.

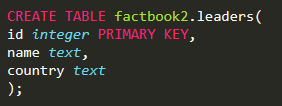
La commande ne fonctionne pas sur l’ensemble des bdd SQL. Sur SQLite, on ne peut pas l’utiliser. Dans PostgreSQL elle fonctionne !

**Créer et Supprimer une table**

Comment créer une nouvelle table avec un nouveau schéma de table ?

En utilisant la commande CREATE TABLE



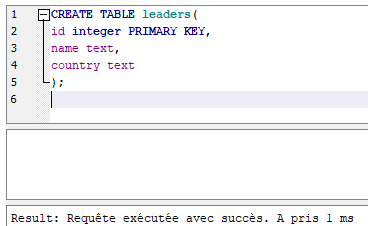


La commande CREATE TABLE qui indique que nous voulons créer une table

On précise le nom de la bdd factbook2 , ainsi que le nom de la table que nous souhaitons créer leaders.

On ouvre la parenthèse pour spécifier à l’intérieur les colonnes et les types de données présentes.

**Ne pas oublier la clé primaire comme porte d’entrée de la table !!!!**



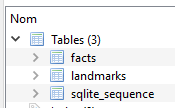


Comment supprimer une table de la bdd ?

En utilisant la commande DROP TABLE







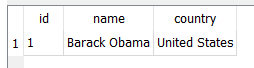
Plus de table leaders dans la bdd factbook2

##### **Relation entre tables**

* Nous venons juste de créer une table leaders contenant les noms des dirigeants des pays qui eux sont répertorié dans la table facts.
* Au lieu d’ajouter une colonne leaders dans la table facts, nous avons créé une colonne country dans la table leaders contenant les noms des pays.
* Ce n’est pas idéal et surtout les 2 tables facts et leaders ne sont pas liées.
* Vous regardez la table facts pour avoir les infos d’un pays et si vous souhaitez le chef d’état de ce pays, vous devez faire une recherche depuis la table leaders alors que nous sommes dans la table facts.

Nous ajouterons une ligne à notre table leaders



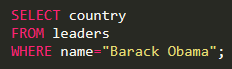


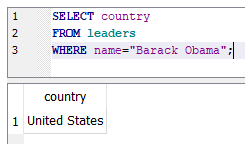
Supposons que l’on connaisse le leader d’un pays et l’on souhaite obtenir les informations en lien avec le pays de ce leader.

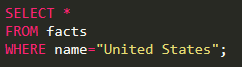
On doit forcément faire une recherche depuis la table facts en mentionnant le nom du pays.

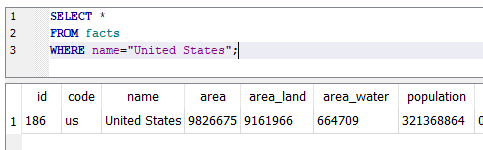
* On extrait la colonne country de la table leaders, à partir de la colonne name du leader,
* Puis on interroge la table facts avec le nom du pays obtenu.

Ce qui nous fait 2 requêtes.









**Conclusion :**

**Pour l’instant nous avons besoin de 2 requêtes SQL pour obtenir les informations demandées. Il n’y a aucun moyen de combiner les données avec ce schéma de tables.**

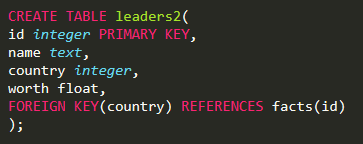
Nous devons créer des relations directement dans le schéma de la table.

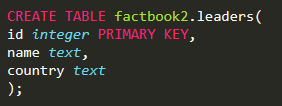
Le type de relations communes est appelée est la clé étrangère FOREIGN KEY.

Et cette clé étrangère/externe va pointer sur un élément(colonne) d’une autre table.

Exemple :

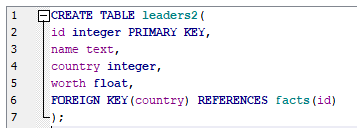
* On créé une nouvelle table leaders2, et on intègre une FOREIGN KEY pour la colonne country avec comme référence l’id de la table facts.

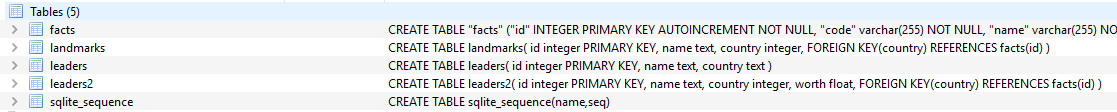


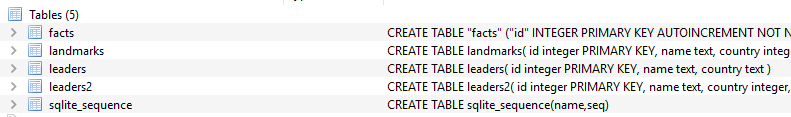
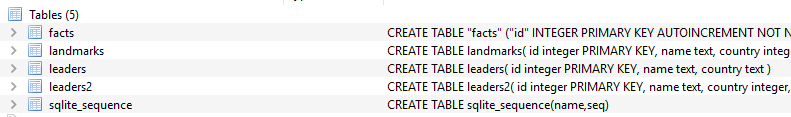


[Attention] ‘text’ à remplacer ici par ‘integer’ car il s’agit de la FOREIGN KEY en référence à la colonne id (de type integer) de la table facts.

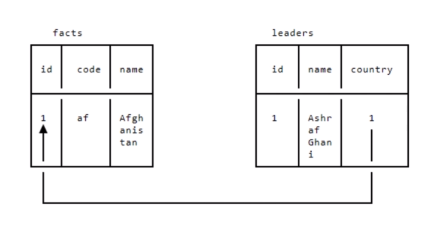
Maintenant, country est une colonne en correspondance avec comme référence, la colonne id de la table facts







[Attention], après avoir créé la table leaders2, la colonne country ne peut seulement avoir des valeurs entier (integer) présent dans la colonne id de la table facts.

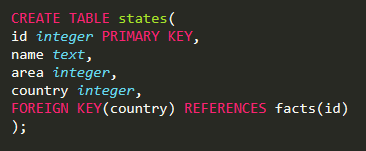


Exercice :

Ecrire une requête SQL qui crée une table states dans la bdd factbook2.

La table states contiendra les colonnes suivantes :

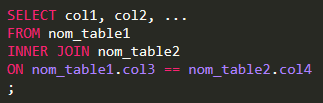
* id : integer qui sera la clé primaire
* name: text
* area: integer
* country: integer qui sera la clé externe de la colonne id de la table facts.



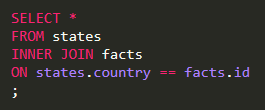


##### **Commande INNER JOIN**

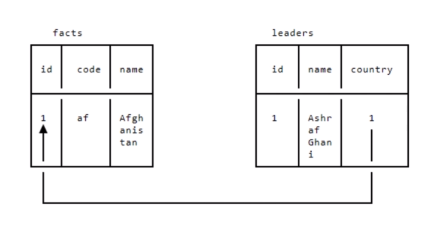
Pour utiliser les relations entre les tables, on utilise la commande INNER JOIN



Exemple :



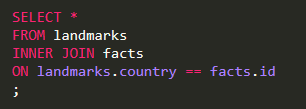
* INNER JOIN indique que nous combinons les données de 2 tables en une seule requête.
* facts qui spécifie la table que nous allons combiner avec la table states.
* La commande ON indique comment SQL raccorde une donnée de la table states à une donnée de la table facts.
* states.country spécifie que la colonne country sera utilisée pour faire la liaison avec la colonne id de la table facts, via facts.id .

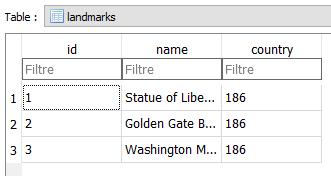
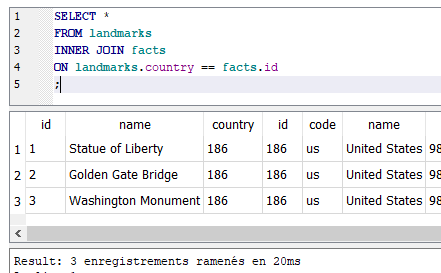


states

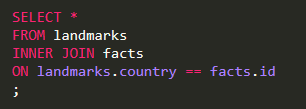
Exercice :

* Ecrire une requête SQL qui affiche toutes les colonnes des tables landmarks et facts (présentes dans la base de données factbook2.db





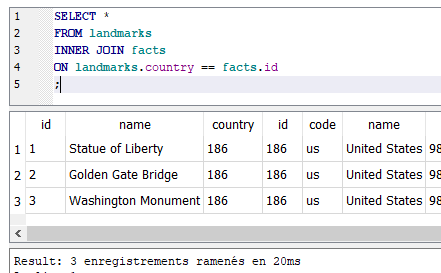
Chaque table dans une commande JOIN a un côté !

 Ici on peut intervertir le sens d’affichage !

Dans cette requête, la table landmarks est le côté gauche de la table.

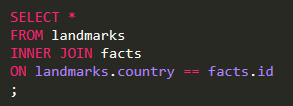
Et la table facts est le côté droit.

Quand on affiche cette requête, les colonnes de gauche seront celles de la table landmarks, et celles de droite seront celles de facts.

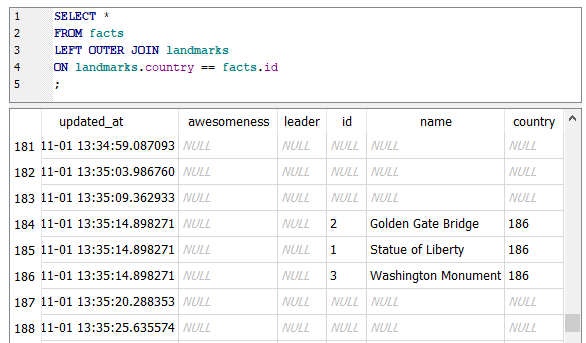


##### **Types de liaison / combinaison entre 2 tables**

INNER JOIN = affiche seulement les lignes pour lesquelles il y a un match entre les valeurs, sur la partie ON. Ici affichage uniquement lors d’une correspondance entre la colonne landmarks.country et la colonne facts.id



LEFT OUTER JOIN = ce type de liaison affiche tous les types de la table de gauche. Et si il y a une ligne de la table gauche qui ne présente aucun match avec l’autre côté, alors on affichera la valeur NULL.



RIGHT OUTER JOIN = ce type de liaison affiche tous les types de la table de droite. Et si il y a une ligne de la table droite qui ne présente aucun match avec l’autre côté, alors on affichera la valeur NULL.

[Attention] Interdite sur SQLite

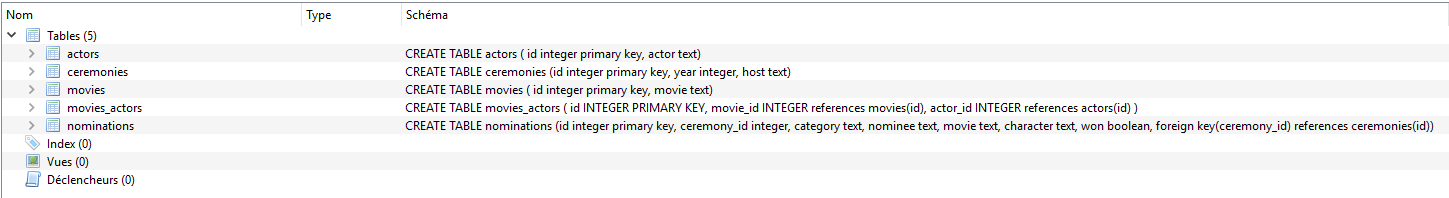
# SECTION 11 – NORMALISATION DE LA BASE DE DONNEES ET RELATIONS

Relation one-to-many et many-to-many

Join Table

Découverte de la base de données

Pour cette section on va utiliser la bdd academy\_awards.db

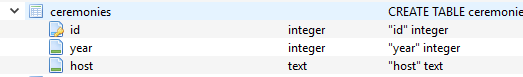


On ava utiliser les tables nominations et :



🔑Clé primaire

🔑Clé étrangère



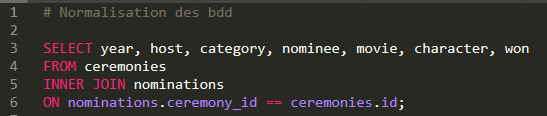
🔑Clé primaire

Normalisation de la base de données

Quel intérêt de normaliser notre bdd

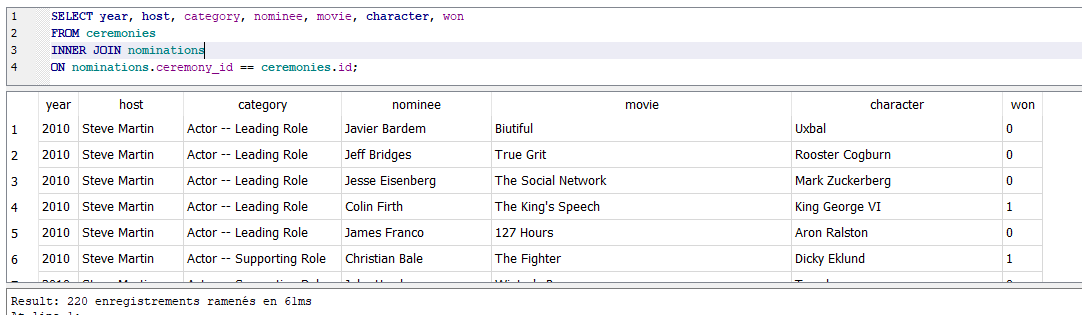
La table cérémonies contient les informations sur chaque cérémonies. Alors que la table nomination contient les informations sur chaque nominations individuelles.

Pourquoi nous n’avons pas une seule table nominations qui contient toutes les informations ?



ON car c’est la colonne id de la table ceremonies qui est liée à ceremony\_id de la table nominations.

Nous faisons apparaitre cette nouvelle table commune, puis nous expliquerons pourquoi c’est une erreur d’architecture de données de joindre ces 2 tables en une seule.



Inconvénient de cet architecture de données :

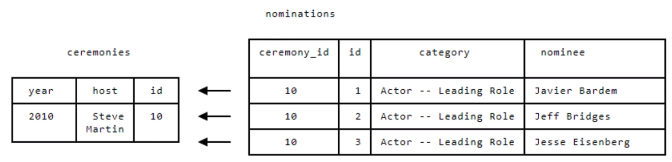
* On remarque qu’il y a bcp de données redondantes. Toutes ces redondances ont tendances à prendre bcp de place de stockage et créer des difficultés.
* Si nous souhaitons supprimer ou modifier une valeur dans les colonnes year ou host, nous devrions le faire sur chaque ligne présentant les même valeurs. Donc risques de se tromper.
* Modifier des lignes peut être fastidieux pour de grosse bdd. Donc plus ma bdd augment, plus la vitesse de MàJ de ma bdd se dégrade.

Par csq, l’architecte de bdd a choisi de créer 2 tables distinctes ceremonies et nomination afin d’optimiser la maintenance de la bdd. D’où l’intérêt des clés primaire étrangère et des relations entre tables..

Types de relation entre tables

One-to-many

Une relation one-to-many existe lorsque plusieurs lignes d’une table sont liées à une seule lignes d’une autre table.



🔑

🔑

Plusieurs lignes de la table nominations peuvent être liées à une seule ligne de la table ceremonies.

A noter qu’une ligne de la table ceremonies ne contient pas de lien avec une ligne de la table nominations.

Avec la relation one-to-many il n’y a qu’une référence que dans un sens !

Exemple :

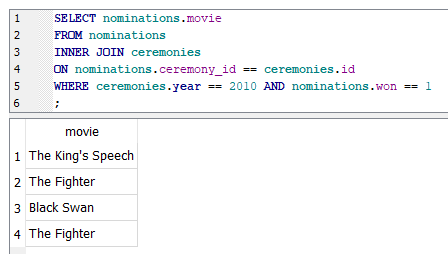
* Une assurance voiture peut avoir plein de clients assurés. Par contre un assuré a une seule assurance voiture.
* Une maman biologique peut avoir plusieurs enfants mais un enfant n’as qu’une seule mère biologique.

Requête sur une base de données normalisée

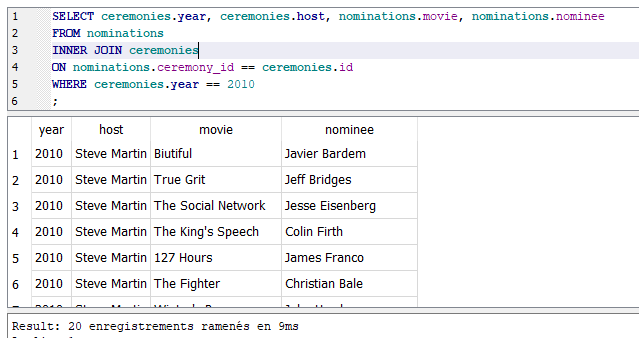
Une requête sur une base normalisée peut être plus complexe et longue à écrire.

Créer une requête qui retourne ts les films qui ont gagné une récompense en 2010 :

Il va être nécessaire de lier les tables Ceremonies et Nominations pour pouvoir filtrer sur les films de l’année 2010 et filtrer sur les films récompensés.



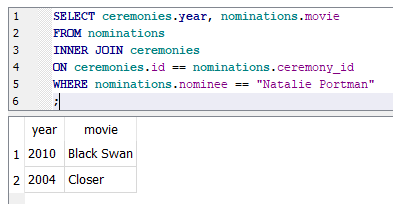
Maintenant on souhaite afficher l’année, le nom du maitre de ceremonie, le film et les nominés pour ce film :



On a bien filtré sur l’année 2010 et le maître de ceremonie et on affiche bien les 2 autres colonnes de la table nominations.

Exercice :

* Ecrire une requête qui retourne tous les films pour lesquels l’acrice Natalie Portman fut nominée pour un oscar (indiquer aussi l’année de la nomination).
* Tester la requête sur SQL sur DB Browser puis afficher les résultats sur Jupyter Notebook.



ON : se demander sur quel critères sont liées les tables

On importe la librairie sqlite3 et on crée une instance de connexion à la bdd :

import sqlite3

connexion = sqlite3.connect('academy\_awards.db')

On crée la variable portman\_query qui va contenir notre requête SQL :

portman\_query ="SELECT ceremonies.year, nominations.movie FROM nominations INNER JOIN ceremonies ON ceremonies.id == nominations.ceremony\_id WHERE nominations.nominee == **'**Natalie Portman**'**;"

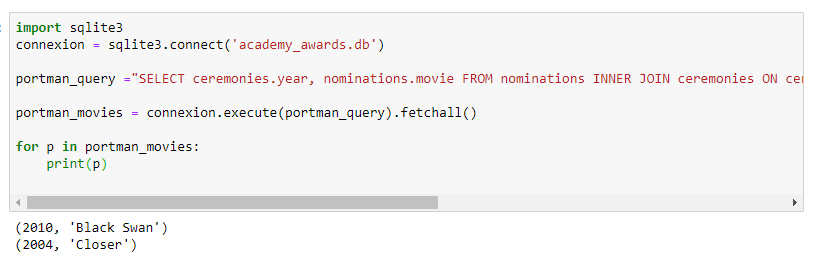
Mainteant on récupère ts les résultats, donc ts les films pour lesquels Nathalie Portman est nominée en créant la variable portman\_movies :

portman\_movies = connexion.execute(portman\_query).fetchall()

Pour afficher les résultats, on crée une petite boucle pour afficher à la ligne les résultats :

for p in portman\_movies:

print(p)



Créer une Join Tabe ou table intermédiaire

Many-to-Many:

Ici nous avons beosin de représenter la relation entre acteur (table actors) et film (table movies).





Avec ce genre d’architecture, nous ne pouvons pas utiliser de relation one-to-many. Car un acteur peut avoir jouer dans plusieurs films, et un film peut avoir plusieurs acteurs.

La solution est d’utiliser une relation many-to-many. Qui contiendra les tables movies et actors et une 3eme table intermédiaire, que l’on appel Join Table (movies\_actors), qui aura pour but de lier les 2 tables.



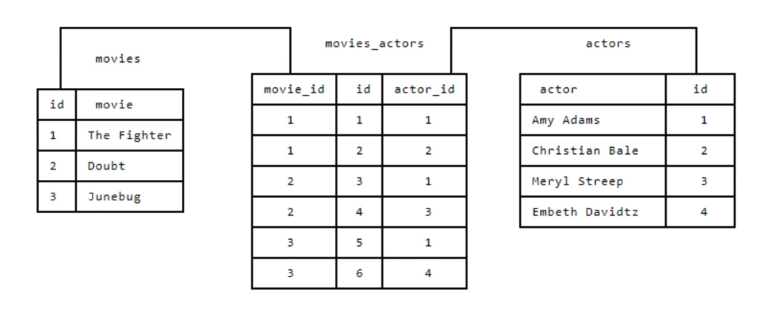


L’idée de la join tabe movies\_actors est de contenir des clés externes pour les 2 tables actors et movies.

Qui présenteront chacune une relation one-to-many avec la table intermédiaire.

Donc un film pourra avoir plusieurs ligne de la join table.

Et un acteur pourra avoir plusieurs ligne de la join table.



Traduction, les tables actors et movies ont une relation many-to-many.

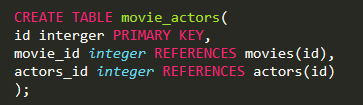
On remarque que chacune des tables à sa colonne id.

movies.id = movies\_actors.movie\_id

actors.id = movies\_actors.actor\_id

movies\_actors.id est présente pour identifier chaque ligne de la table.

Requête pour créer la join table :



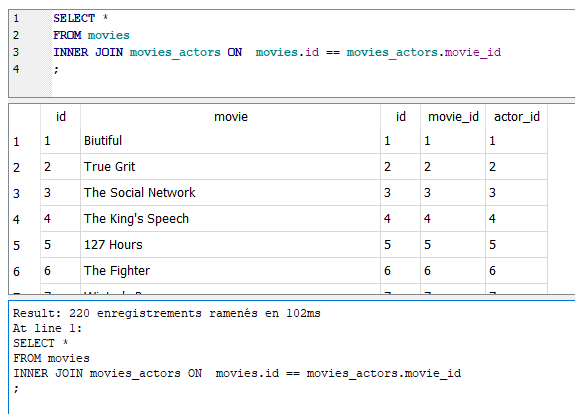
Requête SQL pour une relation many-to-many

Nous remarquons que la table movies\_actors ne contient que des id qi font références aux id des tables movies et actors.

Supposons que nous souhaitions connaitre les acteurs qui jouent dans le film INVICTUS, nominé pour un oscar entre 2001 et 2010.

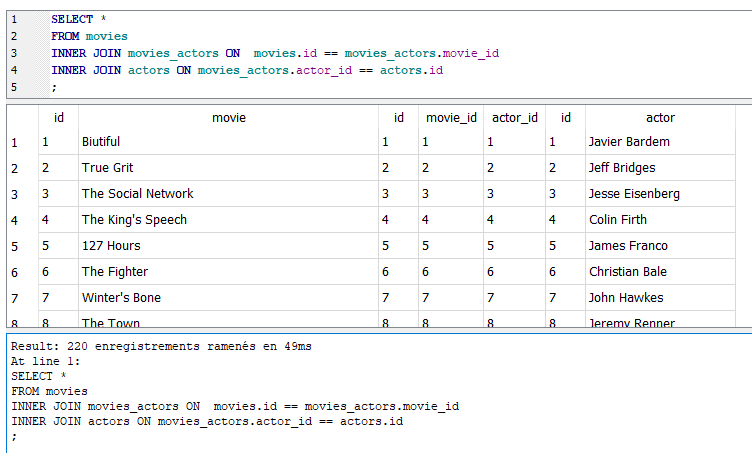
Pour cela nous devons utiliser plusieurs raccord sur trois tables !

On va commencer par relier les tables movies avec movies\_actors :



L’id que nous récupérons à ce stade correspond à la table actors.

On va effectuer nootre seconde jointure, en ajoutant la liaison entre les tables movies\_actors et actors.



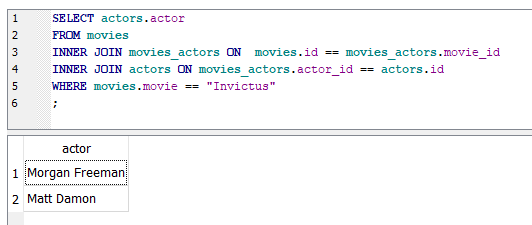
Cette fois, s’ajoute une colonne actor (avec sa colonne id associée) qui correspond à la table actors.

A ce stade on a un acteur qui correspond à un film.

L’objectif est de connaitre LES acteur du film INVICTUS.

On a besoin d’afficher les acteurs.

Pour avoir seulement les aceurs du film INVICTUS, on va rajouter une condition WHERE



Pour résumer, nous démarrons avec notre table movies,

Nous la relions à la table movies\_actors celon le critère movies.id égale movies\_actors.movie\_id

Puis on lie cette table movies\_actors à la table actors avec ce critère movies\_actors.actor.id égale actors.id

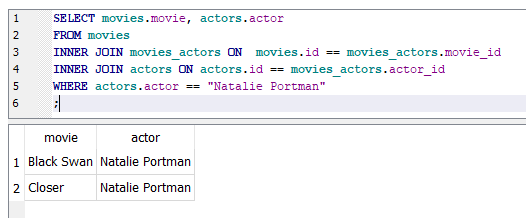
L’inverse est possible en commençant par la table actors puis movies.

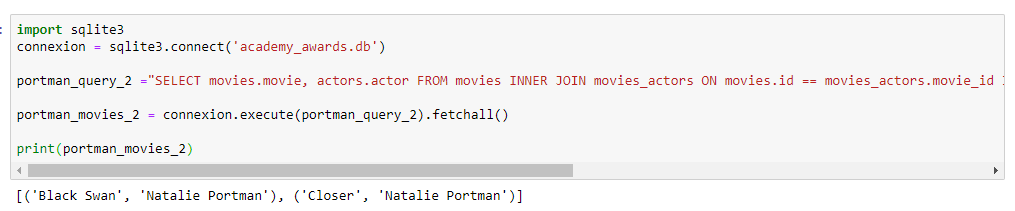
On y applique la condition WHERE pour cibler Invictus.

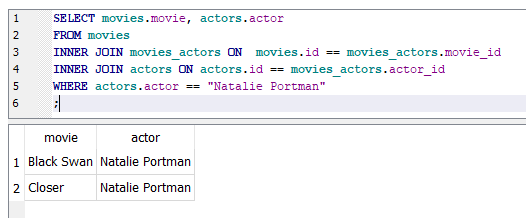
Pratique sur la relation many-to-many

Ecrire une requête qui retourne tous les films dans lesquels « Natalie Portman » a joué.

Tester la requête SQL sur DB Browser puis afficher les résultats sur Jupyter Notebook.





Double cote dans la condition pour SQL

Simple cote dans la condition pour Jupyter Notebook

La normalisation de bdd, càd séparer certaine données en tables différente aide à réduire la redondance des données. Et nous aide à découper notre bdd en différentes tables.

A utiliser avec parcimonie pour éviter les requêtes complexes.

NB, adapter le schema de nos tables en fonction des besoin de la bdd.

De commencer par une bdd non normalisée.

Commencer par une énorme table ou quelques grosses tables puis restructurer après avec une normalisation.

SECTION 12 - PROJET GUIDE: PREPARER LES DONNEES POUR SQLITE (BONUS PYTHON)  
Cleaner un dataset en Python pour l'exporter sur SQLite (niveau intermédiaire)

Introduction

Le Dataset

Filtrer les données

Nettoyer les colonnes Unnamed et Won ?

Nettoyage de la colonne Additionnal Info

Exporter vers SQLite