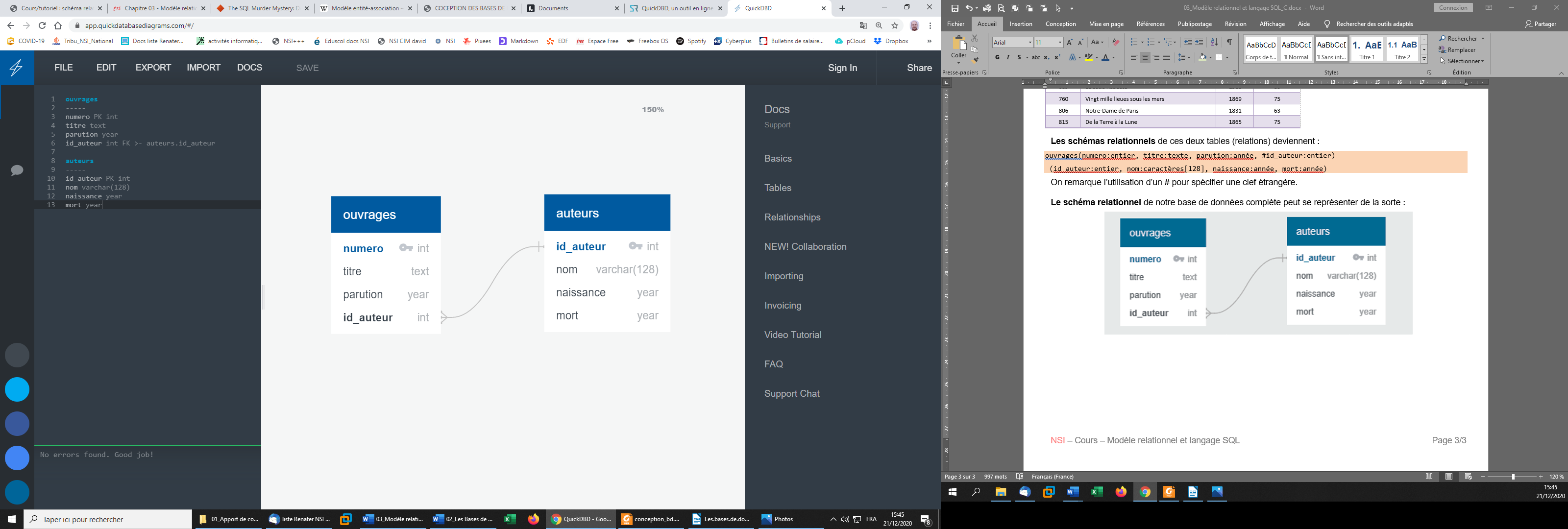
Pour présenter le langage SQL nous utiliserons le logiciel [DB Browser for SQLite](https://sqlitebrowser.org/) et nous reprendrons l'exemple des relations *ouvrages* et *auteurs :*

auteurs(id\_auteur:entier, auteur:caractères[128], naissance:année, mort:année)

ouvrages(numero:entier, titre:texte, date:année, #id\_auteur:entier)



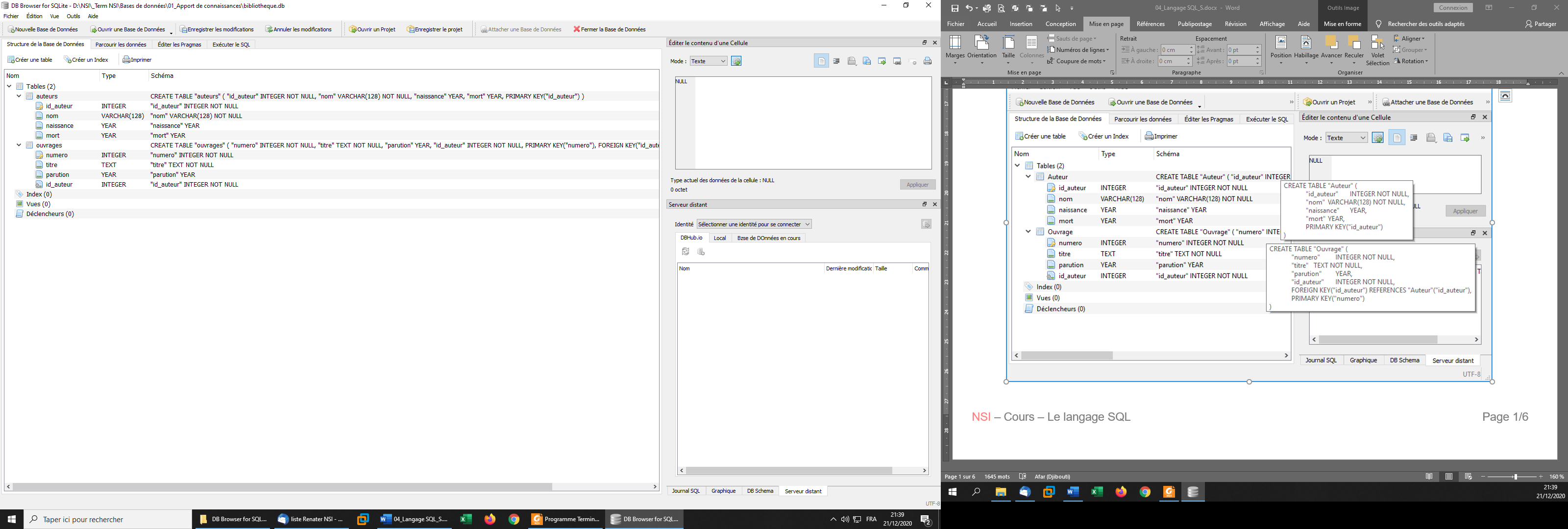
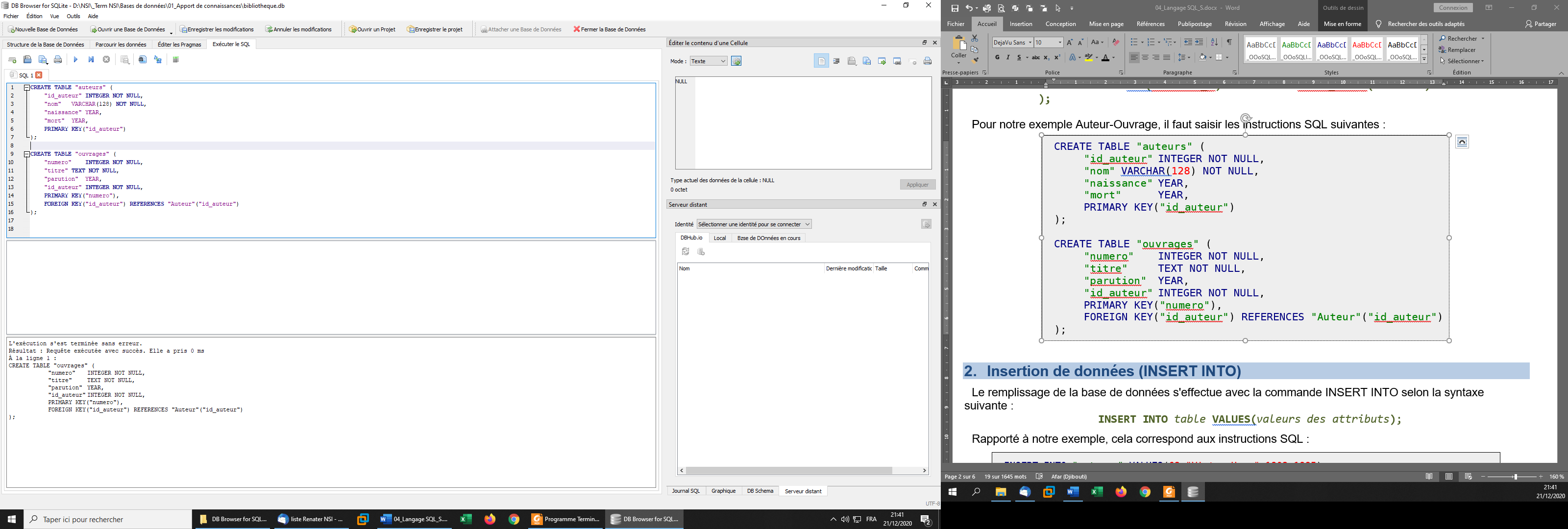
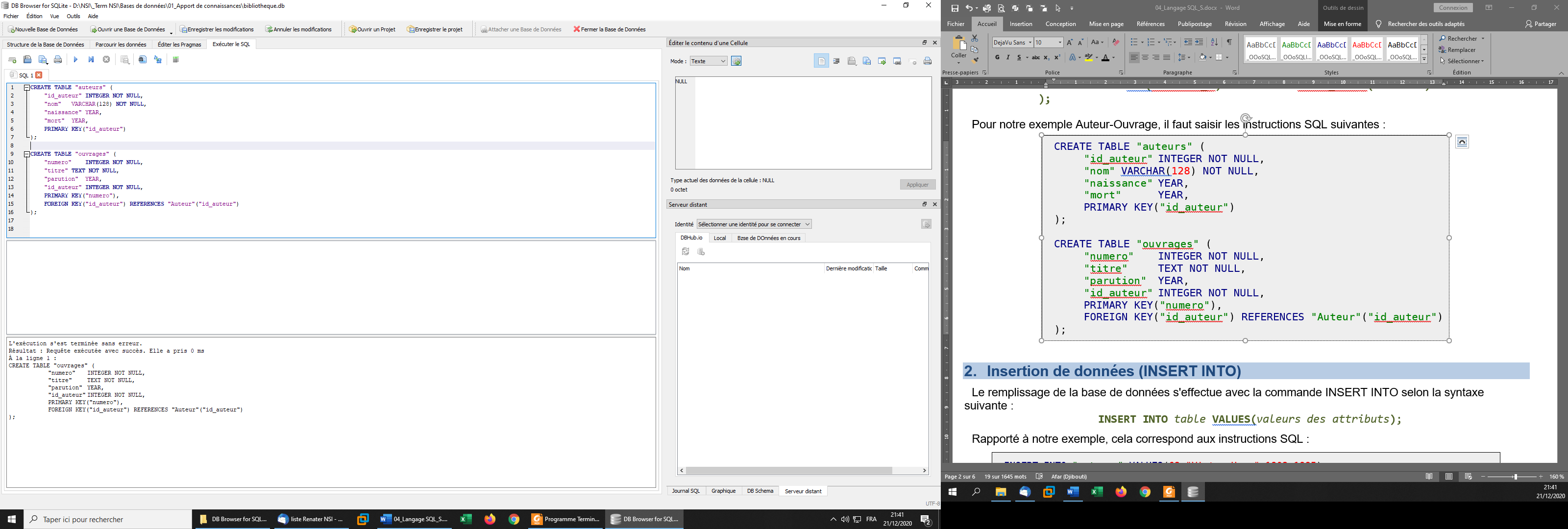
Remarque :

SQLite implémente un nombre restreint de types de données. En particulier il n'existe pas de chaîne limitée à un nombre prédéfini de caractères ni de type année. Il reste cependant possible d'utiliser ces types dans la définition du schéma relationnel, mais il faut garder à l'esprit que ceux-ci se verront convertis automatiquement en texte ou en entier.

# Création d'une base de données et des tables (CREATE)

Pour créer une base de données on commence tout simplement par définir son schéma relationnel dans le SGBDR. Ceci peut se faire à l'aide d'outils graphiques ou bien directement en langage SQL.

Cette première étape consiste à créer une nouvelle base de données, puis à créer les tables qui la constituent en définissant leur schéma relationnel et en précisant les clés primaires et étrangères (contrainte d'intégrité de référence).



La création d'une base de données peut s'effectuer aussi dans l'interface de ligne de commande **sqlite3** ou bien sous l'onglet « Exécuter le SQL » de DB Browser for SQLite avec la commande **CREATE TABLE** en suivant la syntaxe :

**CREATE TABLE** *table* **(**

*attribut\_1* **type,**

*attribut\_2* **type,**

...

*attribut\_n* **type,**

**PRIMARY KEY(***attribut\_1***),**

**FOREIGN KEY(***attribut\_k***) REFERENCES** *autre\_table***(***attribut***)**

**);**

Pour notre exemple auteurs-ouvrages, il faut saisir les instructions SQL suivantes :

CREATETABLE"auteurs"(

"id\_auteur"INTEGER NOT NULL,

"nom"VARCHAR(128),

"naissance"YEAR,

"mort"YEAR,

PRIMARYKEY("id\_auteur")

);

CREATETABLE"ouvrages"(

"numero"INTEGER NOT NULL,

"titre"TEXT,

"parution"YEAR,

"id\_auteur"INTEGER,

PRIMARYKEY("numero"),

FOREIGNKEY("id\_auteur")REFERENCES"auteurs"("id\_auteur")

);

# Insertion de données (INSERT INTO)

Le remplissage de la base de données s'effectue avec la commande INSERT INTO selon la syntaxe suivante :

**INSERT INTO** *table* **VALUES(***valeurs des attributs***)**;

Rapporté à notre exemple, cela correspond aux instructions SQL :

INSERT INTO "auteurs" VALUES(63,"Victor Hugo",1802,1885);

INSERT INTO "auteurs" VALUES(75,"Jules Verne",1828,1905);

INSERT INTO "auteurs" VALUES(83,"Ken Follett",1949,0);

INSERT INTO "auteurs" VALUES(84,"John Steinbeck",1902,1968);

INSERT INTO "auteurs" VALUES(96,"Comtesse de Ségur",1799,1874);

INSERT INTO "ouvrages" VALUES(142,"Le Dernier Jour d'un condamné",1829,63);

INSERT INTO "ouvrages" VALUES(241,"Voyage au centre de la Terre",1864,75);

INSERT INTO "ouvrages" VALUES(323,"Les raisins de la colère",1939,84);

INSERT INTO "ouvrages" VALUES(327,"Les Petites Filles modèles",1858,96);

INSERT INTO "ouvrages" VALUES(358,"Les Misérables",1862,63);

INSERT INTO "ouvrages" VALUES(419,"Les Malheurs de Sophie",1858,96);

INSERT INTO "ouvrages" VALUES(492,"Le Tour du monde en quatre-vingts jours",1872,75);

INSERT INTO "ouvrages" VALUES(677,"Le réseau corneille",2001,83);

INSERT INTO "ouvrages" VALUES(689,"Le code Rebecca",1980,83);

INSERT INTO "ouvrages" VALUES(760,"Vingt mille lieues sous les mers",1869,75);

INSERT INTO "ouvrages" VALUES(806,"Notre-Dame de Paris",1831,63);

INSERT INTO "ouvrages" VALUES(815,"De la Terre à la Lune",1865,75);

On peut réaliser la même opération dans DB Browser for SQLite, en se rendant dans l'onglet « Parcourir les données ». Il faut alors créer de nouveaux enregistrements et saisir les informations des différentes entrées pour chacune des tables.

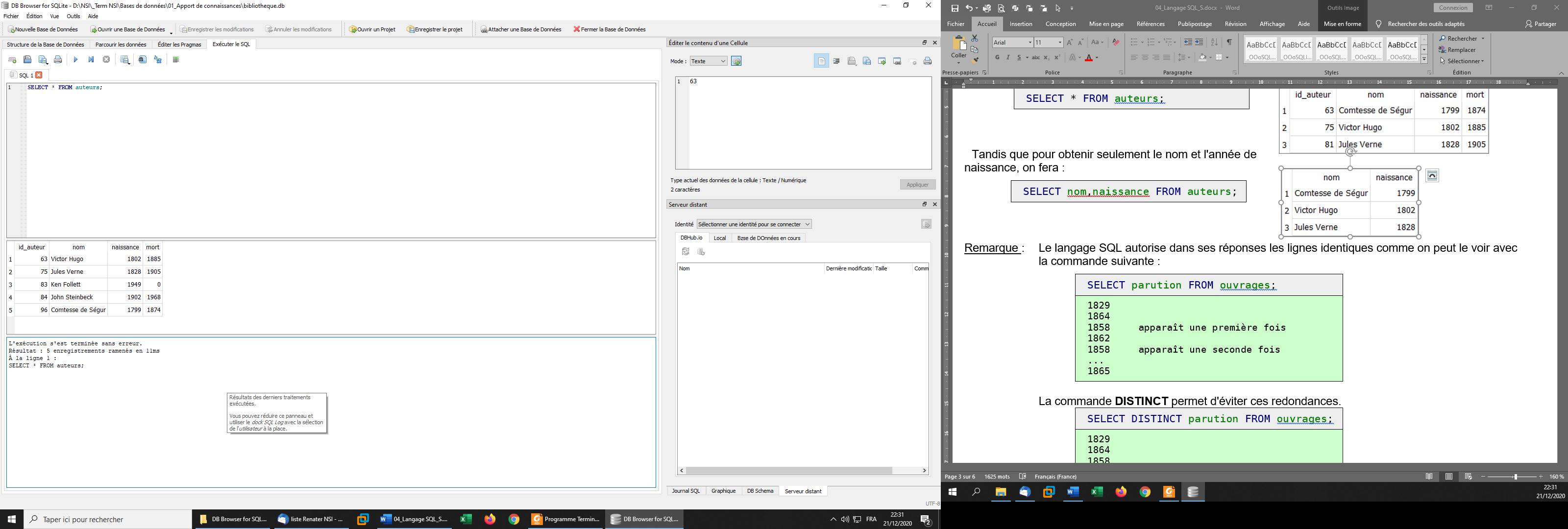
Nous venons de voir comment créer les tables de la base de données avec la commande CREATE, et comment remplir celle-ci en insérant les entrées dans les différentes tables au moyen de la commande INSERT INTO. Passons maintenant aux commandes usuelles.

# La commande (SELECT)

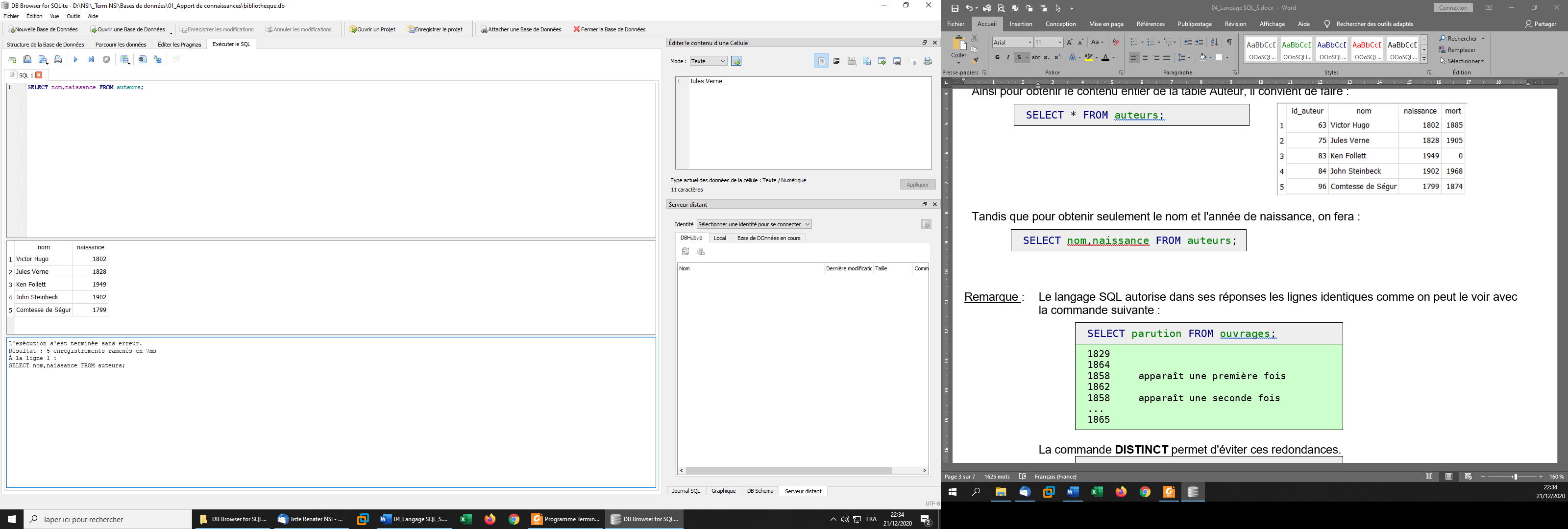
La projection consiste à créer une nouvelle relation en conservant certains attributs (en partie ou en totalité). Autrement dit, la projection opère une sélection sur les colonnes d'une table. Elle se réalise avec la commande SELECT, qui est de loin la commande la plus utilisée en SQL, en respectant la syntaxe :

**SELECT** *[attribut\_1,attribut\_2,...,attribut\_n|\*]* **FROM** *table*;

où \* est un raccourci qui signifie « *tous les attributs* »

Ainsi pour obtenir le contenu entier de la table Auteur, il convient de faire :

|  |
| --- |
| SELECT \* FROM auteurs; |

Tandis que pour obtenir seulement le nom et l'année de naissance, on fera :

|  |
| --- |
| SELECT nom,naissance FROM auteurs; |

Remarque : Le langage SQL autorise dans ses réponses les lignes identiques comme on peut le voir avec la commande suivante :

|  |
| --- |
| SELECT parution FROM ouvrages; |
| 1829  1864  1858 apparaît une première fois  1862  1858 apparaît une seconde fois  ...  1865 |

La commande **DISTINCT** permet d'éviter ces redondances.

|  |
| --- |
| SELECT DISTINCT parution FROM ouvrages; |
| 1829  1864  1858  1862  1872  1860  1869  1831  1865 |

# La sélection (WHERE)

La sélection, aussi appelée restriction, crée une nouvelle relation en sélectionnant les tuples sur la base de conditions à définir et qui portent sur les attributs. Autrement dit, la sélection opère une sélection sur les lignes selon un prédicat qui doit être précisé. Elle s'effectue à l'aide de la commande **WHERE** selon la syntaxe :

**SELECT** *[attributs|\*]* **FROM** *table* **WHERE** *prédicat*;

Le prédicat s'écrit au moyen des opérateurs suivants :

Comparaisons : **= < > <= >= <>** (différent)

Opérateurs logiques : **AND OR NOT**

Encadrements : **BETWEEN**

Motifs de chaînes : **LIKE** (**\_** pour un caractère quelconque, **%** pour plusieurs)

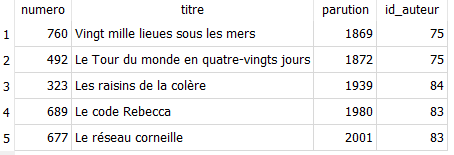
Ordre d’affichage : **ORDER BY** attribut(s) (par défaut ordre croissant ASC, pour un ordre

décroissant ajouter **DESC** en fin de commande)

Voici quelques exemples de sélections :

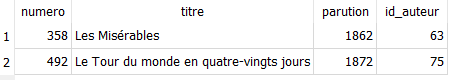
* Les ouvrages parus après 1865 dans l’ordre croissant de l’année de parution.

|  |
| --- |
| SELECT \* FROM ouvrages WHERE parution > 1865 ORDER BY parution; |



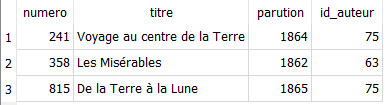
* Les ouvrages parus en 1862 ou 1872.

|  |
| --- |
| SELECT \* FROM ouvrages WHERE parution = 1862 OR parution = 1872; |



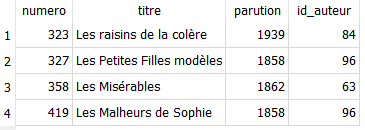
* Les ouvrages parus entre 1860 et 1865.

|  |
| --- |
| SELECT \* FROM ouvrages WHERE parution BETWEEN 1860 AND 1865; |



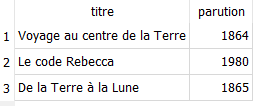
* Les ouvrages dont le titre commence par « Les ».

|  |
| --- |
| SELECT \* FROM ouvrages WHERE titre LIKE "Les%"; |



* Le titre et l'année de parution des ouvrages dont le titre ne contient pas « i ».

|  |
| --- |
| SELECT titre,parution FROM ouvrages WHERE NOT titre LIKE "%i%"; |



# La jointure interne (INNER JOIN)

La jointure consiste à créer une nouvelle relation en composant plusieurs relations. Autrement dit, la jointure crée une nouvelle table à partir de plusieurs tables.

Il existe plusieurs sortes de jointures qui apparient différemment les entrées des relations. Parmi celles-ci, la jointure interne crée une nouvelle table en liant uniquement les lignes de deux tables qui respectent une certaine condition portant sur les colonnes (un prédicat). Elle s'opère avec la commande **JOIN** … **ON**.

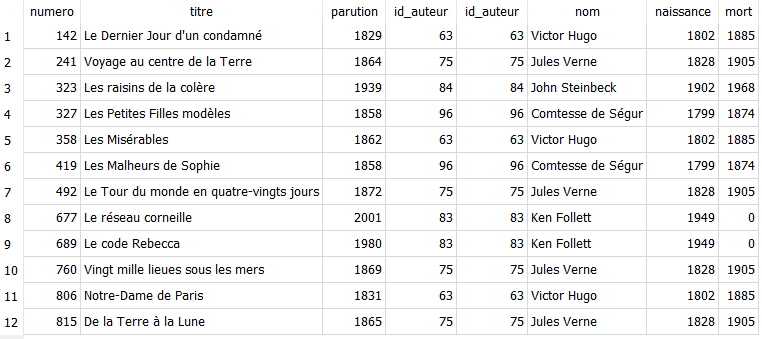
**SELECT** *[attributs|\*]* **FROM** *table1* **JOIN** *table2* **ON** *prédicat***;**

Lorsque le prédicat utilise des noms de colonnes identiques dans les deux tables on doit préfixer ces noms par leur table avec un point de séparation (table.colonne).

Voici quelques exemples de jointures internes :

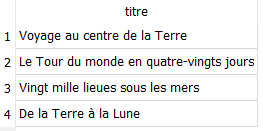
* Les ouvrages appariés à leurs auteurs.

|  |
| --- |
| SELECT \* FROM ouvrages JOIN auteurs ON ouvrages.id\_auteur = auteurs.id\_auteur; |



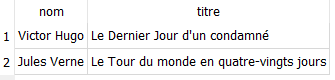
* Les titres des ouvrages de Jules Verne.

|  |
| --- |
| SELECT titre FROM ouvrages JOIN auteurs  ON ouvrages.id\_auteur = auteurs.id\_auteur  WHERE nom = "Jules Verne"; |



* Les noms des auteurs et les titres de leurs ouvrages uniquement lorsqu'ils contiennent le mot "jour".

|  |
| --- |
| SELECT DISTINCT nom,titre FROM auteurs JOIN ouvrages  ON auteurs.id\_auteur = ouvrages.id\_auteur  AND titre LIKE "%jour%"; |



Notez comment LIKE est insensible à la casse des caractères.

# Les requêtes d’agrégation

Les fonctions d’agrégation dans le langage SQL permettent d’effectuer des opérations statistiques sur un ensemble d’enregistrement. Elles permettent des opérations qui servent à compter le nombre de réponse à la requête, récupérer l’enregistrement le plus petit, le plus grand ou bien encore de déterminer la valeur moyenne sur plusieurs enregistrements.

Les principales fonctions sont les suivantes :

* **AVG()** pour calculer la moyenne sur un ensemble d’enregistrement
* **COUNT()** pour compter le nombre d’enregistrement sur une table ou une colonne distincte
* **MAX()** pour récupérer la valeur maximum d’une colonne sur un ensemble de ligne. Cela s’applique à la fois pour des données numériques ou alphanumérique
* **MIN()** pour récupérer la valeur minimum de la même manière que MAX()
* **SUM()** pour calculer la somme sur un ensemble d’enregistrement

Principe d’utilisation :

**SELECT fonction**(colonne) **FROM** table

Voici un exemple d’utilisation :

* Compter le nombre d’ouvrages.

|  |
| --- |
| SELECT COUNT(\*) FROM ouvrages; |

* Compter le nombre d’ouvrages de Ken Folett.

|  |
| --- |
| SELECT COUNT(\*) FROM ouvrages WHERE id\_auteur = 83 ; |

* Trouver la date de parution la plus ancienne.

|  |
| --- |
| SELECT MIN(parution) FROM ouvrages; |

# La mise à jour (UPDATE)

La mise à jour consiste à modifier les valeurs dans des lignes. Elle s'effectue à l'aide de la commande **UPDATE** ... **SET** en précisant la table concernée, ainsi que les colonnes à modifier avec leurs nouvelles valeurs. Elle est le plus souvent complétée d'une sélection (**WHERE**) afin d'identifier les lignes à mettre à jour. En son absence ce sont toutes les lignes qui sont alors modifiées. La syntaxe est la suivante :

**UPDATE** *table* **SET** *attribut\_1=valeur\_1, …, attribut\_n=valeur\_n* **[WHERE** *prédicat***];**

Voici un exemple de mise à jour :

* Changement du titre de l'ouvrage portant le numéro 492.

|  |
| --- |
| UPDATE ouvrages SET titre = "Le tour du monde en 80 jours" WHERE numero = 492;  SELECT \* FROM ouvrages WHERE numero = 492; |



# La suppression (DELETE)

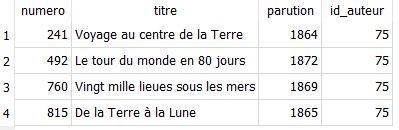
La suppression consiste à supprimer une ou plusieurs lignes dans une table. Elle se fait généralement sur la base d'un prédicat, qui sert à identifier les lignes concernées. En l'absence de ce prédicat, ce sont alors toutes les lignes de la table qui sont supprimées. La commande de suppression **DELETE** suit la syntaxe :

**DELETE FROM** *table* **[WHERE** *prédicat***];**

Voici deux exemples de suppression :

* Suppression des ouvrages dont le id\_auteur n'est pas 75 (*Victor Hugo*).

|  |
| --- |
| DELETE FROM ouvrages WHERE id\_auteur <> 75;  **-- affichage des ouvrages restants après la suppression**  SELECT \* FROM ouvrages; |



* Suppression des auteurs dont le id\_auteur n'est pas 75 (*Victor Hugo*).

|  |
| --- |
| DELETE FROM auteurs WHERE id\_auteur <> 75;  **-- affichage des auteurs restants après la suppression**  SELECT \* FROM auteurs; |



Remarque : Si on commence par le second exemple, un message d'erreur est affiché indiquant que la commande ne respecte pas la contrainte de référence imposée par l'utilisation d'une clé étrangère. Il est en effet interdit de supprimer un auteur tant que des ouvrages de cet auteur sont présents dans la table *ouvrages*.

|  |
| --- |
| **-- PRAGMA foreign\_keys=ON;**  DELETE FROM auteurs WHERE id\_auteur <> 75; |
| Error: FOREIGN KEY constraint failed |