

Comprendre et charger une base de données dans un SGBDR

Ce document technique explicite et précise les étapes permettant de construire une base de données relationnelle dans un Système de Gestion de Base de Données Relationnelle (SGBDR) à partir de données brutes contenues dans des fichiers au format CSV.

Pour l'exemple, le SGBDR qui est utilisé dans cette démarche est *SQLiteStudio*.

I. Comprendre les données brutes

Nous disposons de 2 fichiers source en guise de données brutes : *Contrat.csv* et *Region.csv*. Une première chose à faire avant d'explorer les fichiers de données est de comprendre le format dans lequel ces données ont été stockées dans les fichiers. En utilisant un éditeur de fichiers comme Visual Studio Code (voir Figure 1), on constate d'une part que nos 2 fichiers respectent bien le format .csv, avec une première ligne donnant le nom des colonnes, et d'autre part qu'ils utilisent le point-virgule ';' comme séparateur. Enfin, le fichier *Contrat.csv* ne contient pas de caractères accentués tandis que c'est l'inverse pour le fichier *Region.csv*. Enfin, on note bien le nombre total de lignes non-vides pour chaque fichier :

- 30336 pour *Contrat.csv* dont une ligne d'en-tête ;
- 38917 pour *Region.csv* dont une ligne d'en-tête.

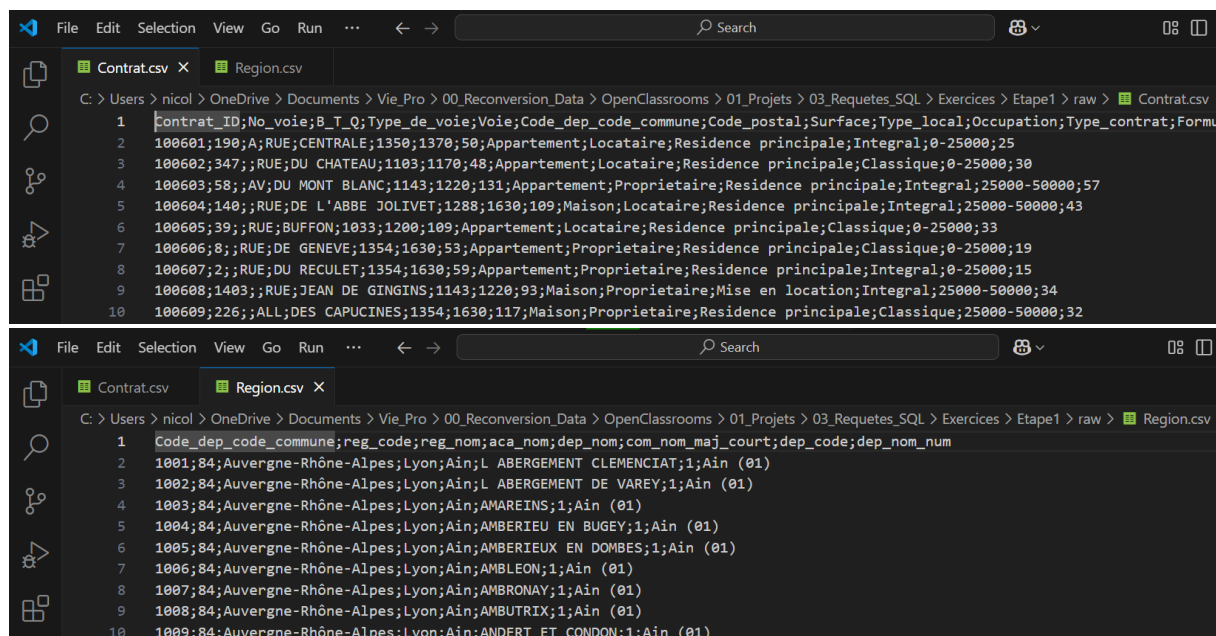


Figure 1 : captures d'écrans données brutes avec Visual Studio

Visual Studio Code n'est pas ergonomique pour explorer les données, mais il nous permet de nous assurer que ces fichiers sont bien lisibles par Excel. On ouvre les fichiers source avec Excel pour en explorer le contenu. Cette fois-ci, les données sont bien regroupées par colonne, même si certaines cellules sont vides. On utilise le raccourci clavier Ctrl+Shift+L pour insérer des filtres pour chaque colonne : après cette manipulation, des petits boutons fléchés apparaissent pour chaque cellule non-vide de la première ligne, ce sont nos filtres. Cela est bien plus pratique pour explorer les données (voir Figure 2). On vérifie que l'on a bien les mêmes nombres de lignes pour chacun des fichiers.

	A	B	C	D	E	F	G	H	I
1	Code_dep	reg_code	reg_nom	aca_nom	dep_nom	com_nom	dep_code	dep_nom	m
2	1001	84	Auvergne-Rhône-Alpes	Lyon	Ain	LABERGEMIEUX	1	Ain (01)	
3	1002	84	Auvergne-Rhône-Alpes	Lyon	Ain	LABERGEMIEUX	1	Ain (01)	
4	1003	84	Auvergne-Rhône-Alpes	Lyon	Ain	AMAREINS	1	Ain (01)	
5	1004	84	Auvergne-Rhône-Alpes	Lyon	Ain	AMBERIEU EN	1	Ain (01)	
6	1005	84	Auvergne-Rhône-Alpes	Lyon	Ain	AMBERIEUX E	1	Ain (01)	
7	1006	84	Auvergne-Rhône-Alpes	Lyon	Ain	AMBLEON	1	Ain (01)	
8	1007	84	Auvergne-Rhône-Alpes	Lyon	Ain	AMBRONAY	1	Ain (01)	
9	1008	84	Auvergne-Rhône-Alpes	Lyon	Ain	AMBUTRIX	1	Ain (01)	
10	1009	84	Auvergne-Rhône-Alpes	Lyon	Ain	ANDERT ET C	1	Ain (01)	
11	1010	84	Auvergne-Rhône-Alpes	Lyon	Ain	ANGLEFORT	1	Ain (01)	
12	1011	84	Auvergne-Rhône-Alpes	Lyon	Ain	APREMONT	1	Ain (01)	
13	1012	84	Auvergne-Rhône-Alpes	Lyon	Ain	ARANC	1	Ain (01)	
14	1013	84	Auvergne-Rhône-Alpes	Lyon	Ain	ARANDAS	1	Ain (01)	
15	1014	84	Auvergne-Rhône-Alpes	Lyon	Ain	ARBENT	1	Ain (01)	
16	1015	84	Auvergne-Rhône-Alpes	Lyon	Ain	ARBIGNIEU	1	Ain (01)	
17	1016	84	Auvergne-Rhône-Alpes	Lyon	Ain	ARBIGNY	1	Ain (01)	
18	1017	84	Auvergne-Rhône-Alpes	Lyon	Ain	ARGIS	1	Ain (01)	
19	1018	84	Auvergne-Rhône-Alpes	Lyon	Ain	ARLOD	1	Ain (01)	
20	1019	84	Auvergne-Rhône-Alpes	Lyon	Ain	ARMIX	1	Ain (01)	

Figure 2 : exploration des fichiers source avec Excel

On peut maintenant explorer le contenu de chaque colonne et renseigner le dictionnaire des données (voir Figure 3).

	Nom des colonnes	Type de données	NULL ?	Taille	Clé	Description
CONTRAT.CSV	Contrat_ID	INTEGER	NOT		Clé primaire	Id unique pour les contrats
	No_voie	INTEGER				Numéro dans la voie pour l'adresse du logement assuré
	B_T_Q	CHAR				Indicateur éventuel de répétition pour l'adresse du logement assuré sur un caractère
	Type_de_voie	VARCHAR		4		Type de voie pour l'adresse du logement assuré: rue, av (Avenue), rte (Route), ...
	Voie	VARCHAR	NOT			Libellé de la voie pour l'adresse du logement assuré
	Code_dep_code_commune	VARCHAR	NOT	6	Clé étrangère	Concaténation du code département et code commune pour avoir une clé unique
	Code_postal	INTEGER	NOT			Code postal pour l'adresse du logement assuré
	Surface	INTEGER	NOT			Surface (en m²) du logement assuré
	Type_local	VARCHAR	NOT			Type de bien : appartement ou maison
	Occupation	VARCHAR	NOT			Statut immobilier de l'occupant du logement (contractant) : locataire ou propriétaire
	Type_contrat	VARCHAR	NOT			Type de contrat pour le logement assuré : résidence principale/secondaire, mise en location
REGION.CSV	Formule	VARCHAR	NOT			Formule du contrat pour le logement assuré : classique ou intégral
	Valeur_declaree_bien	VARCHAR	NOT			Fourchette de la valeur déclarée du bien (en €)
	Prix_cotisation_mensuel	INTEGER	NOT			Montant (en €) de la cotisation mensuelle stipulée dans le contrat
	Code_dep_code_commune	VARCHAR	NOT	6	Clé primaire	Concaténation du code département et code commune pour avoir une clé unique
	reg_code	INTEGER	NOT			Code de la région administrative française (0 = Collectivités d'Outre-Mer, 1 chiffre = Outre-Mer, 2 chiffres = France métropolitaine ou Corse)
	reg_nom	VARCHAR	NOT			Libellé de la région administrative
	aca_nom	VARCHAR	NOT			Libellé de l'académie de rattachement
	dep_nom	VARCHAR	NOT			Libellé du département
	com_nom_maj_court	VARCHAR	NOT			Libellé du nom abrégé de la commune, uniquement avec des lettres capitales et des espaces
	dep_code	VARCHAR	NOT	3		Code du département : 2 chiffres pour les départements de France métropolitaine, 2A ou 2B pour la Corse, 3 chiffres pour les Outre-mers
	dep_nom_num	VARCHAR	NOT			Concaténation nom du département - espace - code département entre parenthèses

Figure 3 : dictionnaire des données

Remarques :

- toujours garder en tête quelles sont les clés primaires de chaque table : en guise de bonne pratique, on voit ici que les clés primaires sont les premiers attributs de chaque table ;
- dans le fichier *Contrat.csv*, les champs *No_voie*, *B_T_Q*, *Type_de_voie* peuvent être sans valeur (NULL) ;
- attention à quelques subtilités :
 - dans *Contrat.csv*, le champ *Code_postal* est un entier d'au plus 5 chiffres en écriture décimale (inférieur à 99999) ;

- certaines entrées de `Code_dep_code_commune` ont 6 caractères (communes situées en Outre-Mer), et les lettres A et B peuvent apparaître (départements de la Corse : '2A' et '2B') ;
- les entrées du champ `reg_code` sont des entiers d'au plus 2 chiffres en écriture décimale (inférieurs à 99), et le nombre de chiffres est porteur d'information : le nombre 0 correspond à des territoires d'Outre-Mer qui ne sont pas rattachés à des entités administratives de type Région ; les autres nombres à 1 chiffre (inférieurs à 10) correspondent à des DROM, les nombres à 2 chiffres correspondent à des régions métropolitaines ou à la Corse ;
- les entrées du champ `dep_code` sont des chaînes de caractères avec 2 chiffres pour les départements métropolitains, les chaînes '2A' et '2B' correspondent respectivement à la Corse du Sud et la Haute-Corse, les chaînes à 3 chiffres correspondent à des territoires d'Outre-Mer.

À quoi peuvent servir ces tables ?

La première table *Contrat.csv* regroupe des données relatives à des contrats d'assurance (clé primaire artificielle : `Contrat_ID`) liés à des logements : y figurent notamment l'adresse du bien, sa surface, son statut d'occupation, sa valeur, et des informations sur l'exploitation commerciale de ce bien (type de formule, cotisation).

La seconde table *Region.csv* donne pour chaque commune (clé primaire artificielle : `Code_dep_code_commune`) des informations liées à son département, son académie et sa région de rattachement. La clé primaire « naturelle » de cette table est en fait la combinaison des colonnes `dep_code` et `Com_nom_maj_court`.

En termes de **multiplicité**, chaque contrat n'a qu'un seul `Code_dep_code_commune`, mais un même `Code_dep_code_commune` peut être associé à plusieurs contrats (**multiplicité 1 à n**).

En utilisant ces deux tables conjointement, on est en mesure de réaliser des regroupements géographiques des données liées aux contrats d'assurance.

II. Diagramme UML de la base de données

Le diagramme UML de la base de données peut être réalisé avec des logiciels comme *SQL Power Architect*. Il suffit de renseigner pour chacun des champs le type de données attendu, ainsi que les contraintes associées : taille maximale pour les chaînes de caractère, si le champ peut être vide, si un attribut sert de clé étrangère (FK) à une autre table, ce qui est le cas pour l'attribut `Code_dep_code_commune` (voir Figure 4 ci-dessous).

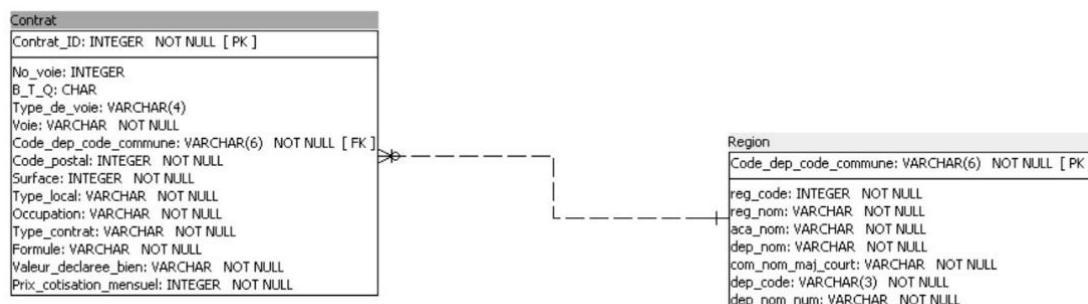


Figure 4 : diagramme UML réalisé avec SQL Power Architect

III. Code SQL de la structure de la base de données

L'avantage pratique à l'utilisation de *SQL Power Architect*, c'est qu'il génère le code SQL qui permet de charger la structure de la base de données dans le SGBDR (voir Figure 5).

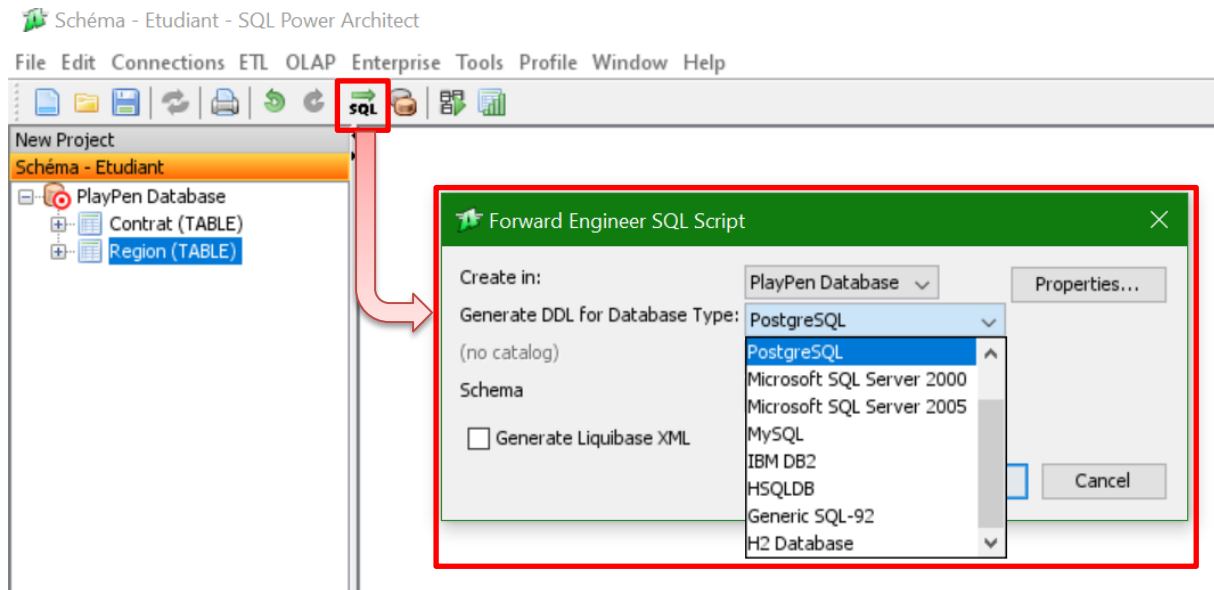


Figure 5 : générer le code SQL pour créer la structure de la base de données

On remarque que le SGBDR SQLite ne figure pas parmi les formats de sortie de *SQL Power Architect*. On choisit le format PostgreSQL car pour notre compréhension, c'est le format le plus fidèle à la logique du langage SQL : voir tableau ci-dessous, issu d'OpenClassrooms (<https://openclassrooms.com/fr/courses/6971126-implementez-vos-bases-de-donnees-relationnelles-avec-sql/7139618-decouvrez-le-systeme-de-gestion-de-base-de-donnees-sgbd>).

	MySQL	Oracle Database	PostgreSQL	SQLite
Popularité	😊😊	😊 (Big Data)	😞	😊 (prototypage)
Prix	gratuit	très cher	gratuit	gratuit
Licence	fermée	fermée	open-source	open-source
Fidélité syntaxe SQL	😞	😞	😊😊	😊
Utilisé par...	Facebook	Samsung	Spotify	apps Android

Le code SQL généré par *SQL Power Architect* est le suivant :

```
CREATE TABLE Region (  
    Code_dep_code_commune VARCHAR(6) NOT NULL,  
    reg_code INTEGER NOT NULL,  
    reg_nom VARCHAR NOT NULL,  
    aca_nom VARCHAR NOT NULL,  
    dep_nom VARCHAR NOT NULL,  
    com_nom_maj_court VARCHAR NOT NULL,  
    dep_code VARCHAR(3) NOT NULL,  
    dep_nom_num VARCHAR NOT NULL,  
    CONSTRAINT region_pk PRIMARY KEY (Code_dep_code_commune)  
);
```

```
CREATE TABLE Contrat (  
    Contrat_ID INTEGER NOT NULL,  
    No_voie INTEGER,  
    B_T_Q CHAR,  
    Type_de_voie VARCHAR(4),  
    Voie VARCHAR NOT NULL,  
    Code_dep_code_commune VARCHAR(6) NOT NULL,  
    Code_postal INTEGER NOT NULL,  
    Surface INTEGER NOT NULL,  
    Type_local VARCHAR NOT NULL,  
    Occupation VARCHAR NOT NULL,  
    Type_contrat VARCHAR NOT NULL,  
    Formule VARCHAR NOT NULL,  
    Valeur_declaree_bien VARCHAR NOT NULL,  
    Prix_cotisation_mensuel INTEGER NOT NULL,  
    CONSTRAINT contrat_pk PRIMARY KEY (Contrat_ID)  
);
```

```
ALTER TABLE Contrat ADD CONSTRAINT region_contrat_fk  
FOREIGN KEY (Code_dep_code_commune)  
REFERENCES Region (Code_dep_code_commune)  
ON DELETE NO ACTION  
ON UPDATE NO ACTION  
NOT DEFERRABLE;
```

Les 2 premiers blocs d'instructions, commençant par le mot-clé `CREATE TABLE` permettent de charger la structure des tables dans *SQLiteStudio*, le dernier bloc est censé permettre de créer la contrainte de clé étrangère, mais ce dernier bloc d'instructions (notamment `ALTER TABLE`) ne fonctionne pas dans *SQLiteStudio*. On ne conserve donc que les 2 premiers blocs d'instructions, qui fonctionnent parfaitement – créant ainsi une base de données vide – puis on crée la contrainte de clé étrangère dans l'interface de *SQLiteStudio*.

Pour cela, après chargement de la structure des tables, il faut double-cliquer, dans l'arborescence à gauche de la fenêtre, sur la table dont on veut ajouter la contrainte de clé étrangère, de façon à faire apparaître le gestionnaire des attributs (voir Figure 6).

Attention : l'assignation des contraintes de clés étrangères devrait être réalisée dans un dernier temps, **après** avoir vérifié qu'il y a bien concordance des valeurs de la table fille avec celle de la table mère (voir section V), c'est-à-dire que toutes les valeurs prises par la clé étrangère dans la table fille (ici *Contrat*) existent bel et bien dans la table mère (ici *Region*), dont c'est la clé primaire. Ceci afin d'éviter de réaliser l'assignation des clés étrangères à plusieurs reprises.










	Nom	Type de données	Clé primaire	Clé étrangère	Unique	Contrôle	Non NULL	Collecter	Généré
1	Contrat_ID	INTEGER							
2	No_voie	INTEGER							
3	B_T_Q	CHAR							
4	Type_de_voie	VARCHAR (4)							
5	Voie	VARCHAR							
6	Code_dep_code_commune	VARCHAR (6)							
7	Code_postal	INTEGER							
8	Surface	INTEGER							
9	Type_local	VARCHAR							

Figure 6 : gestionnaire d'attributs table fille (avant manipulation)

Double-cliquer sur la ligne correspondant à l'attribut à modifier, la fenêtre des contraintes apparaît (voir Figure 7). Cocher 'Clef étrangère' et configurer la contrainte.

Contraintes

☐  Clef primaire
 ☒  Clef étrangère

Configurer

Configurer

Figure 7 : fenêtre des contraintes d'un attribut

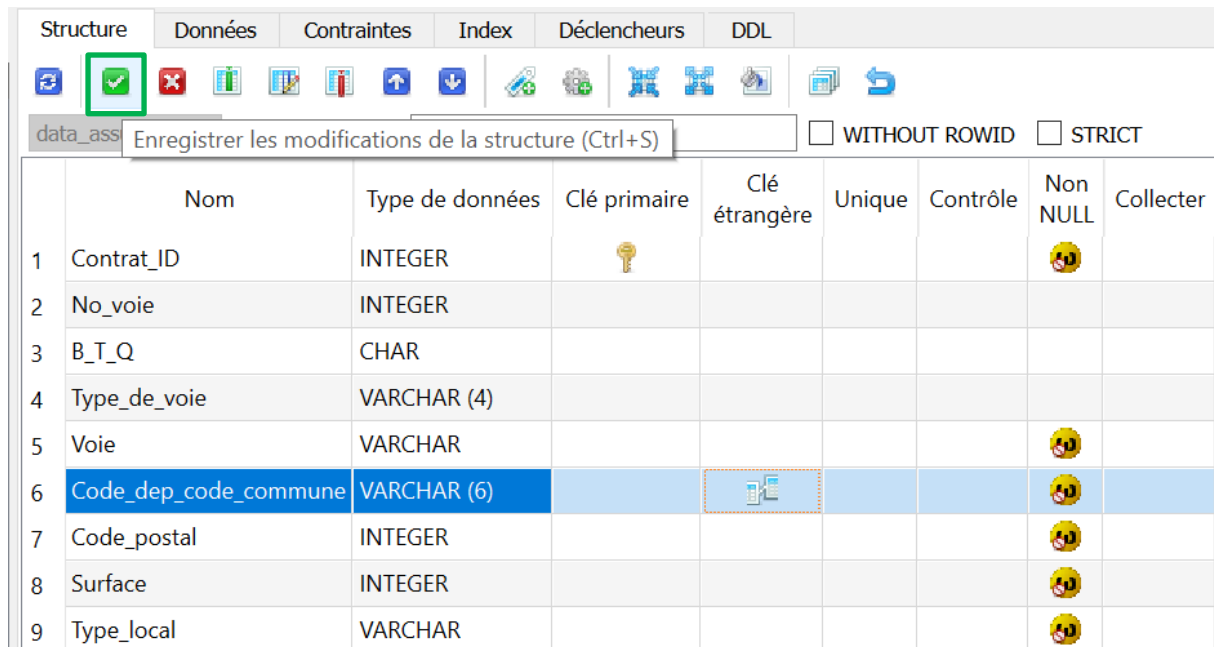


Figure 8 : ne pas oublier d'enregistrer les modifications de structure !

Une fois cette manipulation faite, *SQLiteStudio* présente le code SQL correspondant à exécuter pour mettre en place la contrainte de clé étrangère. Ce code est reproduit en annexe. Il suffit d'accepter la proposition de *SQLiteStudio*, et la contrainte est alors bien assignée.

IV. Charger les données dans la base de données

À ce stade, la structure de la base de données a bien été chargée, mais celle-ci est encore vide. Il faut y importer les données issues des fichiers source de données brutes. Cela peut se faire grâce à l'interface de *SQLiteStudio*, en cliquant sur le bouton Importer (voir Figure 9).

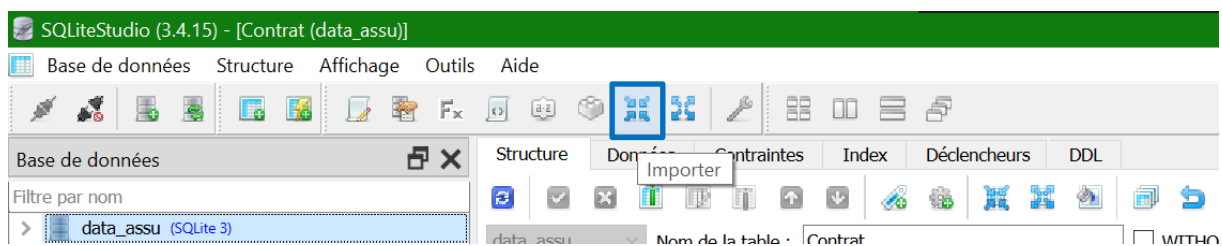
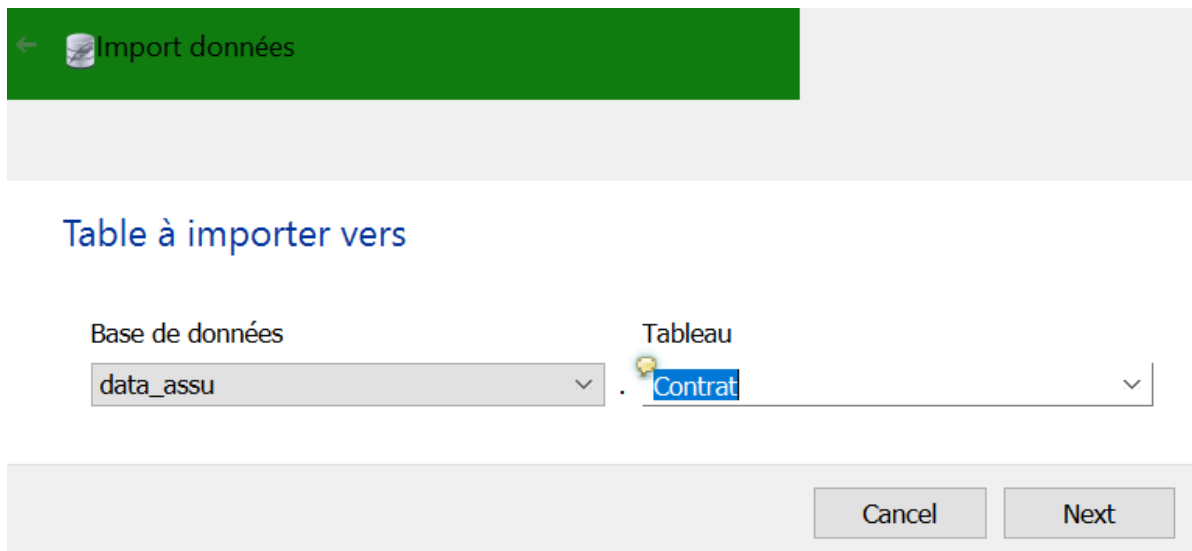


Figure 9 : bouton Importer de SQLiteStudio

Après avoir précisé dans quelle table les données à venir vont être chargées (voir Figure 10), apparaît la fenêtre de l'utilitaire d'import (voir Figure 11) où il faut préciser l'emplacement et le format du fichier source : encodage, caractère de séparation, quelles valeurs doivent être considérées comme des champs vides, si la première ligne du fichier contient le nom des colonnes...



← Import données

Table à importer vers

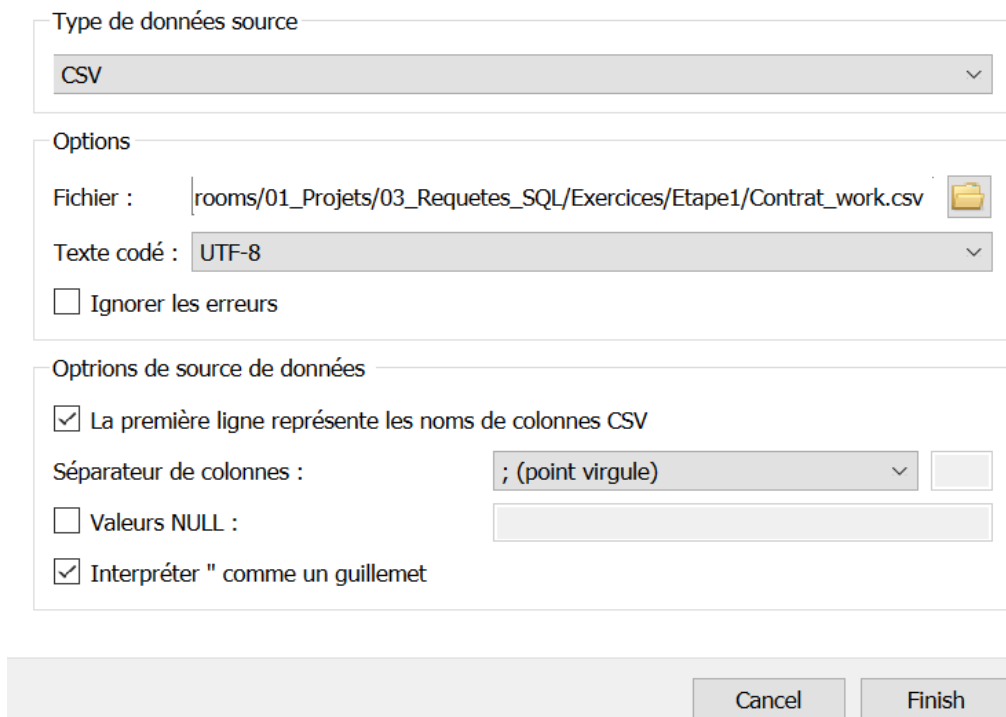
Base de données : data_assu

Tableau : Contrat

Cancel Next

Figure 10 : préciser la table de destination des données à charger

Source de données à importer de



Type de données source : CSV

Options

Fichier : rooms/01_Projets/03_Requetes_SQL/Exercices/Etape1/Contrat_work.csv

Texte codé : UTF-8

☐ Ignorer les erreurs

Options de source de données

☒ La première ligne représente les noms de colonnes CSV

Séparateur de colonnes : ; (point virgule)

☐ Valeurs NULL :

☒ Interpréter " comme un guillemet

Cancel Finish

Figure 11 : utilitaire d'import des données

Parcours Data Analyst - Comprendre et charger une base de données dans un SGBDR

Après import des données, il n'y a plus qu'à vérifier que toutes les colonnes (attributs) et que toutes les lignes (objets ou instances) ont bien été importées. Dans notre cas, on constate que les nombres de lignes et de colonnes concordent avec ce qu'on a vu en I (voir Figure 12 ci-dessous).

The screenshot shows the SQLiteStudio 3.4.15 interface. The left pane displays the database structure for 'data_assu', showing a hierarchy of tables and columns. The right pane shows a query result for the 'Contrat' table, displaying columns like 'Code dep', 'req code', 'req nom', 'aca nom', 'dep nom', 'com nom', 'maj', 'court', 'dep code', and 'dep nom'. The status bar indicates that the query was executed successfully and returned 38916 rows.

	Code dep	req code	req nom	aca nom	dep nom	com nom	maj	court	dep code	dep nom
1	1001	84	Auvergne-Rhône-Alpes	Lyon	Ain	L ABERGEMENT CLEMENCIAT	1		Ain (01)	
2	1002	84	Auvergne-Rhône-Alpes	Lyon	Ain	L ABERGEMENT DE VAREY	1		Ain (01)	
3	1003	84	Auvergne-Rhône-Alpes	Lyon	Ain	AMAREINS	1		Ain (01)	
4	1004	84	Auvergne-Rhône-Alpes	Lyon	Ain	AMBERIEUX EN BUGEY	1		Ain (01)	
5	1005	84	Auvergne-Rhône-Alpes	Lyon	Ain	AMBERIEUX EN DOMBES	1		Ain (01)	
6	1006	84	Auvergne-Rhône-Alpes	Lyon	Ain	AMBLEON	1		Ain (01)	
7	1007	84	Auvergne-Rhône-Alpes	Lyon	Ain	AMBRONAY	1		Ain (01)	

Barre d'état: [15:10:55] Requête terminée en 0.000 seconde(s). [15:11:30] Requête terminée en 0.000 seconde(s).

Figure 12 : base de données dans le SGBDR après import des données
vérifier les nombres de lignes et de colonnes

V. Contrôle des valeurs de la clé étrangère

Des erreurs ou inexactitudes ont pu se glisser dans les fichiers source, et celles-ci peuvent être difficiles à détecter avec des outils basiques de visualisation de données comme Excel. En particulier, toutes les valeurs de clé étrangère qui apparaissent dans une table fille doivent exister parmi les valeurs de la clé primaire dans la table mère. Après avoir chargé les données (mais avant d'avoir assigné les contraintes de clé étrangère), il suffit d'exécuter une requête similaire à :

```
SELECT DISTINCT FKey, Attribut_controle
FROM Table_fille
WHERE FKey NOT IN (SELECT PKey FROM Table_mere);
```

Dans notre cas, cela donne :

The screenshot shows a database management interface with two tabs: 'Requête' (Query) and 'Historique' (History). The 'Requête' tab is active, displaying a SQL query. Below the query, there is a toolbar with various icons for executing and managing the query. To the right of the toolbar, it says 'Nombre de lignes chargées : 3' (Number of lines loaded: 3). Below the toolbar, there is a table with two columns: 'Code dep code commune' and 'Code postal'. The table contains three rows of data.

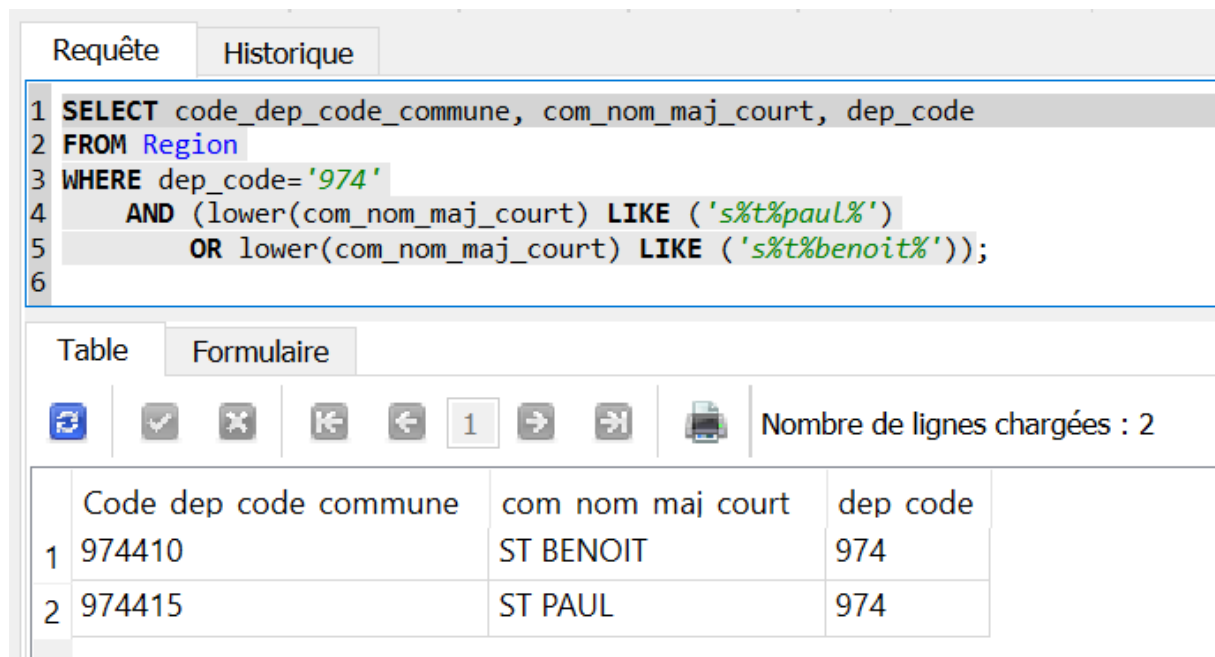
	Code dep code commune	Code postal
1	97460	97460
2	97434	97434
3	97470	97470

Figure 13 : exemple de contrôle de clé étrangère

Dans notre cas, on a 3 valeurs de clé étrangère qui n'existent pas dans la table mère. Après recherche sur Internet, on comprend que :

Code_dep_code_commune Valeurs erronées	Code_postal Attribut de contrôle	Nom commune	Département
97460	97460	Saint-Paul	La Réunion (974)
97434	97434	Saint-Paul	La Réunion (974)
97470	97470	Saint-Benoît	La Réunion (974)

On effectue une requête dans la table mère pour trouver les bonnes valeurs de clé :



The screenshot shows a database query interface with two tabs: 'Requête' (Query) and 'Historique' (History). The 'Requête' tab is active, displaying a SQL query:

```
1 SELECT code_dep_code_commune, com_nom_maj_court, dep_code
2 FROM Region
3 WHERE dep_code='974'
4 AND (lower(com_nom_maj_court) LIKE ('s%%paul%'))
5 OR lower(com_nom_maj_court) LIKE ('s%%benoit%'));
6
```

Below the query, there is a toolbar with icons for saving, checking, deleting, and navigating. To the right of the toolbar, it says 'Nombre de lignes chargées : 2' (Number of lines loaded : 2). Below the toolbar is a table with the following data:

	Code dep code commune	com nom maj court	dep code
1	974410	ST BENOIT	974
2	974415	ST PAUL	974

Figure 14 : retrouver les valeurs correctes de clé étrangère grâce à la vraie clé primaire de la table mère

On a donc trouvé des erreurs dans nos fichiers source, mais on sait comment les corriger.

La meilleure pratique est de garder une copie des fichiers source avec les erreurs (dans un sous-dossier raw_data, par exemple), et de corriger ces erreurs (en éditant les fichiers sous Excel par exemple) dans un autre fichier dit de travail, et de suivre les modifications apportées aux fichiers initiaux de données brutes, de façon à pouvoir revenir en arrière si les corrections étaient inappropriées.

Concernant la base de données dans le SGBDR, il faut supprimer la table de données erronée avec la commande **DROP TABLE** Table_fille, recréer la structure de la table comme on l'a vu en III (avec la commande **CREATE TABLE** Table_fille [...]) **avec** la contrainte de clé étrangère, puis charger les données corrigées en suivant la procédure vue en IV.

Annexe : code SQL (*SQLite*) pour assignation contrainte de clé étrangère

Ce code a été généré automatiquement par le SGBDR *SQLiteStudio*.

```
PRAGMA foreign_keys = 0;
```

```
CREATE TABLE sqlitestudio_temp_table AS SELECT * FROM Contrat;  
DROP TABLE Contrat;
```

```
CREATE TABLE Contrat (  
    Contrat_ID            INTEGER      NOT NULL,  
    No_voie               INTEGER,  
    B_T_Q                CHAR,  
    Type_de_voie          VARCHAR (4),  
    Voie                  VARCHAR      NOT NULL,  
    Code_dep_code_commune VARCHAR (6) NOT NULL  
                        REFERENCES Region (Code_dep_code_commune),  
    Code_postal            INTEGER      NOT NULL,  
    Surface               INTEGER      NOT NULL,  
    Type_local            VARCHAR      NOT NULL,  
    Occupation            VARCHAR      NOT NULL,  
    Type_contrat          VARCHAR      NOT NULL,  
    Formule               VARCHAR      NOT NULL,  
    Valeur_declaree_bien  VARCHAR      NOT NULL,  
    Prix_cotisation_mensuel INTEGER      NOT NULL,  
    CONSTRAINT contrat_pk PRIMARY KEY (Contrat_ID)  
);  
  
INSERT INTO Contrat (Contrat_ID, No_voie, B_T_Q, Type_de_voie, Voie,  
    Code_dep_code_commune, Code_postal, Surface,  
    Type_local, Occupation, Type_contrat, Formule,  
    Valeur_declaree_bien, Prix_cotisation_mensuel)  
  
    SELECT Contrat_ID, No_voie, B_T_Q, Type_de_voie, Voie,  
        Code_dep_code_commune, Code_postal, Surface,  
        Type_local, Occupation, Type_contrat, Formule,  
        Valeur_declaree_bien, Prix_cotisation_mensuel  
    FROM sqlitestudio_temp_table;
```

```
DROP TABLE sqlitestudio_temp_table;
```

```
PRAGMA foreign_keys = 1;
```