Document technique

Introduction

Ce document technique s'inscrit dans le cadre de ma formation DATA ANALYST chez OpenClassrooms et vise à démontrer la méthodologie suivie pour la réalisation du projet "Requêtez une base de données avec SQL".

L'objectif principal est de montrer, étape par étape, comment j'ai abordé et résolu les différentes phases du projet. Ce document détaille l'exploration des données, la rédaction du dictionnaire des données, la réalisation du schéma relationnel via Power Archictect, ainsi que la génération et l'importation du code SQL dans une base de données SQLite pour enfin, importer les données issues des fichiers CSV.

1. Exploration des données

Pour cette phase, j'ai utilisé deux fichiers CSV fournis : **Region** et **Contrat**. L'exploration a été réalisée à l'aide de Google Sheets, en m'appuyant sur les filtres pour vérifier rapidement la cohérence des données. Au cours de cette exploration, plusieurs observations ont été relevées :

Valeurs manquantes : Certaines colonnes contiennent des cellules vides, ce qui nécessite une gestion particulière lors de la modélisation.

Contrat_ID	No_voie	B_T_Q	Type_de_voie
100640	6		
100695	91		
100702	5186		
100707	5208		
100717	20		
100738	5245		
100811			
100860	7002		
100861	7042		

- **Colonne Code_dep_code_commune :** Bien que la majorité des valeurs soient numériques, certaines lignes présentent des valeurs alphabétiques. Cette constatation a remis en cause l'utilisation d'un type de données strictement numérique pour cette colonne.

Code_dep_code_commune	Code_postal
2A004	20000
2A004	20000
2A004	20090

- **Attribution erronée de codes départementaux :** Certaines villes, comme Alfortville, Antony ou Arcueil, apparaissent avec des codes départementaux (et noms de département) qui pourraient induire des erreurs dans les requêtes SQL.

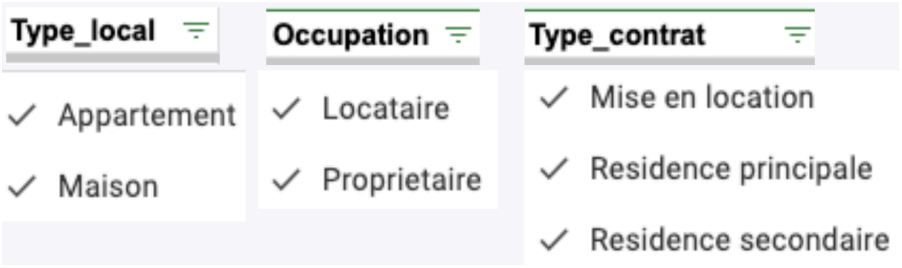
aca_nom	dep_nom	com_nom_maj_court	dep_code	dep_nom_num
Paris	Paris	ALFORTVILLE	75	Paris (75)
Paris	Paris	ANTONY	75	Paris (75)
Paris	Paris	ARCUEIL	75	Paris (75)

- **Données dans la colonne Voie :** Des informations pouvant prêter à confusion (comme des dates ou des adresses incluant des numéros et des noms de rue) ont été relevées, alors qu'il existait déjà une colonne dédiée au numéro de voie et une autre pour le type de voie. Bien que la rue 5 impasse 10 rue du Metz existe, il est important de le prendre en considération lors de l'exécution des requêtes.

Contrat_ID	No_voie	B_T_Q	Type_de_voie	Voie
101890	736		CD	135
102743	4		PL	01/06/1907 00:00
108245	5		IMP	10 RUE DE METZ
114460	12		RUE	11/11/1918 00:00

aca_nom	dep_nom	com_nom_maj_court	dep_code	c
Dijon	Saône-et-Loire	PARIS L HOPITAL	71	S
Toulouse	Tarn	PARISOT	81	T
Toulouse	Tarn-et-Garonne	PARISOT	82	T

- **Contrôle des colonnes à choix limités :** contient uniquement les valeurs attendues, sans fautes ni variations, afin de simplifier le filtrage sans contrainte de casse (par exemple, "Type_local" et "Occupation").



2. Dictionnaires des données

Pour constituer le dictionnaire des données, j'ai complété le template pré-rempli fourni dans le cadre du projet. Ce dictionnaire inclut pour chaque colonne :

- Le **nom** de la colonne
- Le **type de données** (ex. CHAR, VARCHAR, INTEGER)
- Les **contraintes** associées (NOT NULL, clé primaire, clé étrangère)
- La **longueur autorisée** pour les champs de type VARCHAR (ex. VARCHAR(5))

Les observations issues de l'exploration ont guidé la définition des types de données et des contraintes. Par exemple :

- Pour la colonne **Code_dep_code_commune**, malgré une majorité de valeurs numériques, la présence de valeurs alphabétiques (notamment pour la Corse) impose l'utilisation d'un type de données textuel plutôt qu'un entier.
- L'absence de cellules vides dans certaines colonnes a permis de définir la contrainte **NOT NULL** pour celles-ci.
- D'autres observations sur la cohérence et le format des données ont aidé à préciser les contraintes (comme la longueur maximale pour les chaînes de caractères).

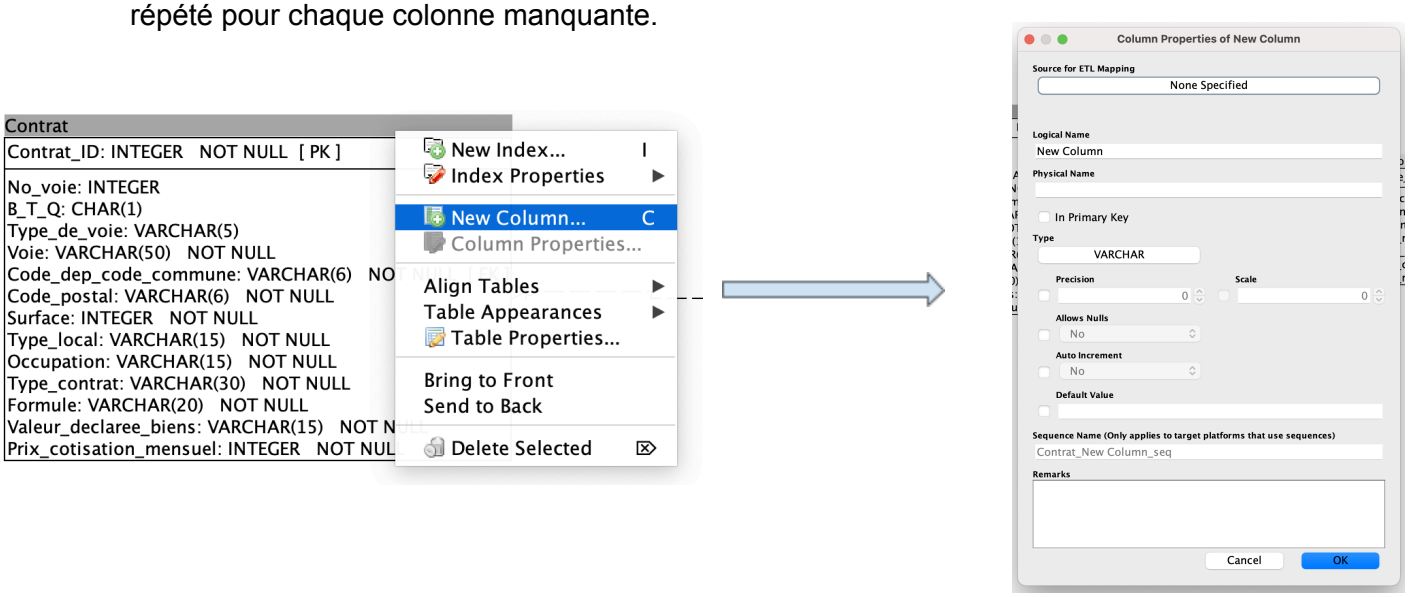
	Nom des colonnes	Type de données	Taille	Clé	Description
CONTRAT.CSV	Contrat_ID	INT		Clé primaire	Id unique pour les contrats
	No_voie	INT			Numéro dans la voie pour l'adresse du logement assuré
	B_T_Q	CHAR	1		Indicateur éventuel de répétition pour l'adresse du logement assuré sur un caractère
	Type_de_voie	VARCHAR	5		Type de voie pour l'adresse du logement assuré: rue, av (Avenue), rte (Route), …
	Voie	VARCHAR	50		Libellé de la voie pour l'adresse du logement assuré
	Code_dep_code_commune	VARCHAR	6	Clé secondaire	Concaténation du code département et code commune pour avoir une clé unique
	Code_postal	VARCHAR	6		Code postal pour l'adresse du logement assuré
	Surface	INT			Superficie du bien immobilier
	Type_local	VARCHAR	15		Précise le type de bien (maison ou appartement)
	Occupation	VARCHAR	15		Précise le type d'occupant du bien (locataire ou propriétaire)
	Type_contrat	VARCHAR	30		Exprime le type de contrat (Résidence principale ou secondaire, ou mise en location)
	Formule	VARCHAR	20		Est le type de formule choisi pour assurer le bien
	Valeur_declaree_biens	VARCHAR	15		Fourchette de la valeur des biens déclarés à assurer
	Prix_cotisation_mensuel	INT			Montant du prix de la cotisation mensuelle
REGION.CSV	Code_dep_code_commune	VARCHAR	6	Clé primaire	Concaténation du code département et code commune pour avoir une clé unique
	reg_code	INT			code de la région
	reg_nom	VARCHAR	40		nom de la région
	aca_nom	VARCHAR	20		Nom de l'académie lié à la région
	dep_nom	VARCHAR	20		nom du département concerné
	com_nom_maj_court	VARCHAR	50		Nom de la commune
	dep_code	VARCHAR	5		code du departement
	dep_nom_num	VARCHAR	50		concaténation du nom puis du numéro de département

3. Schéma relationnel

Pour réaliser le schéma relationnel, j'ai utilisé Power Architect à partir d'un template incomplet. Ce template contenait déjà les deux tables principales, mais il manquait certaines colonnes et leurs paramètres. J'ai complété le schéma en me basant sur les informations extraites du dictionnaire de données, en ajoutant les colonnes manquantes et en configurant précisément leurs propriétés (type de données : VARCHAR, INTEGER, etc., contraintes NOT NULL, taille autorisée, etc.).

3.1. Processus et méthodologie

Ajout des Colonnes :
En sélectionnant la table avec le curseur, j'ai effectué un clic droit et choisi l'option *New Column*. Cela ouvre l'onglet *Column Properties of new column* où j'ai pu paramétrer les caractéristiques de la nouvelle colonne (nom, type, contrainte, taille). Ce procédé a été répété pour chaque colonne manquante.

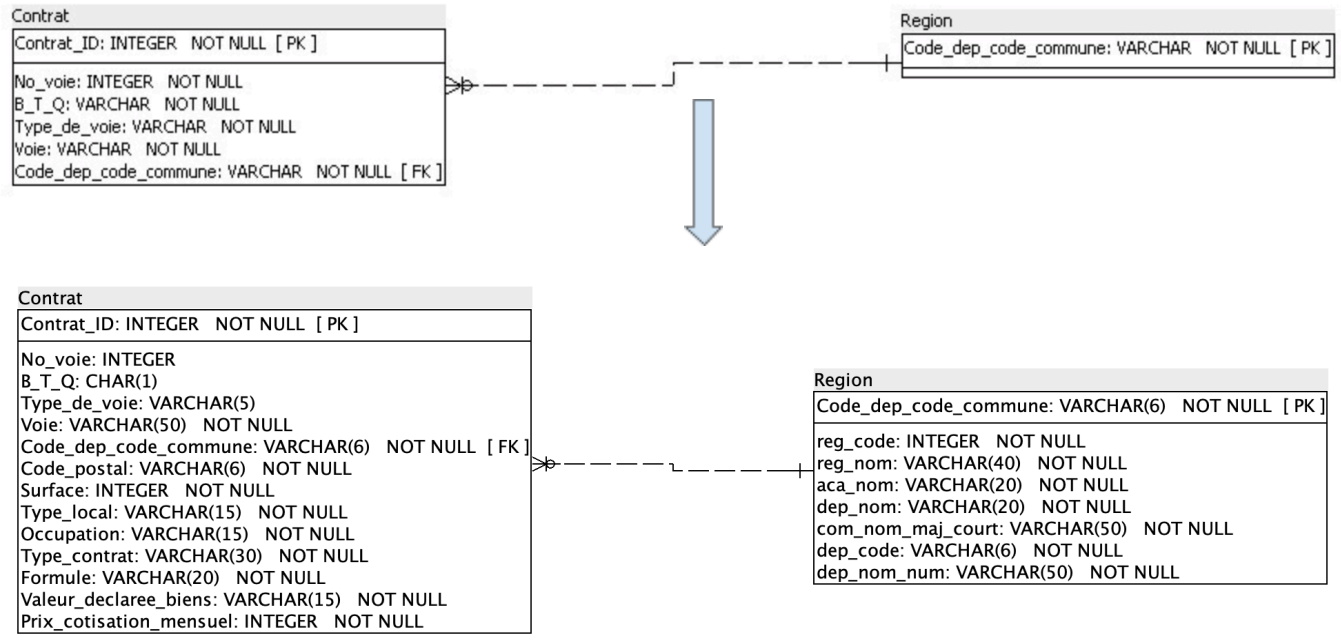


Relations 1 à Plusieurs :
Le template initial comportait déjà une relation de type **1 à plusieurs** entre les tables. Pour rappel, une relation 1 à plusieurs signifie qu'un enregistrement dans la table "1" (par exemple, une région ou un département) peut être associé à plusieurs enregistrements dans la table "plusieurs" (par exemple, des contrats). Ce type de relation est particulièrement adapté dans ce contexte, car il permet de modéliser la réalité où un département peut être rattaché à plusieurs contrats.
Étant donné que cette relation était déjà définie, aucune modification n'a été nécessaire sur ce point.

Difficultés Rencontrées :

La principale difficulté a été de m'adapter rapidement aux fonctionnalités de Power Architect, un outil que je ne maîtrisais pas entièrement. Cela m'a obligé à explorer et à expérimenter pour paramétrer correctement chaque colonne et pour comprendre l'interface de création des colonnes.

Captures d'Écran Avant/Après :



4. Génération du code SQL

Pour obtenir le code SQL à partir du schéma relationnel, j'ai utilisé la fonctionnalité **"Forward engineer SQL script"** de Power Architect. En cliquant sur ce bouton, un onglet de paramétrage s'est ouvert ; après avoir validé les options (en cliquant sur "OK"), une nouvelle fenêtre est apparue avec le code SQL généré. J'ai alors copié ce code et l'ai mis en forme sur Notion en utilisant un bloc de code avec la syntaxe SQL.

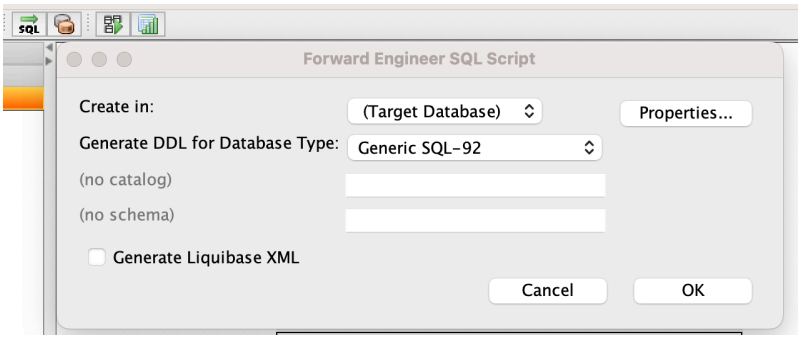
Extraits de Code et Mise en Évidence :

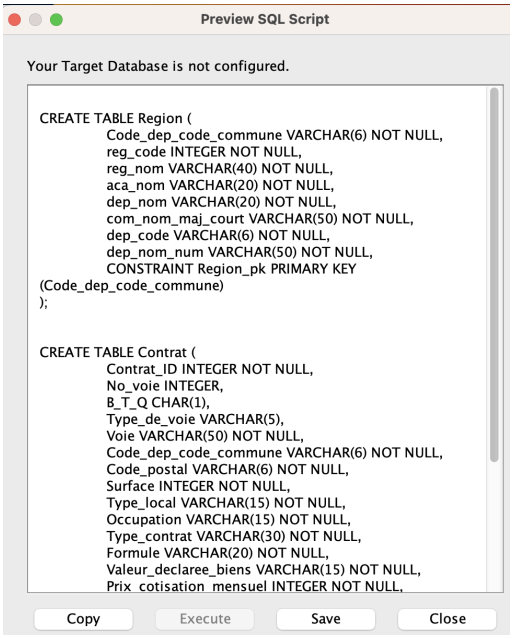
Pour illustrer ce processus, j'inclus des captures d'écran du code SQL généré. Parmi les aspects mis en avant, on peut noter :

- **Définition des Tables et des Colonnes :**
  Les instructions `CREATE TABLE` précisent les colonnes, leur type (ex. `VARCHAR`, `INTEGER`), ainsi que les contraintes comme `NOT NULL`.

- **Définition des Clés Primaires et Étrangères :**
  Le code inclut la création de clés primaires (via `PRIMARY KEY`) et de clés étrangères permettant de garantir la relation 1 à plusieurs entre les tables, ce qui correspond exactement à la structure modélisée dans Power Architect.

- **Contraintes et Paramétrages :**
  La mise en forme du code met en exergue les contraintes définies, telles que la taille autorisée pour les champs de type `VARCHAR` (par exemple, `VARCHAR(5)` pour certaines colonnes) et l'absence de valeurs nulles grâces à `NOT NULL`.

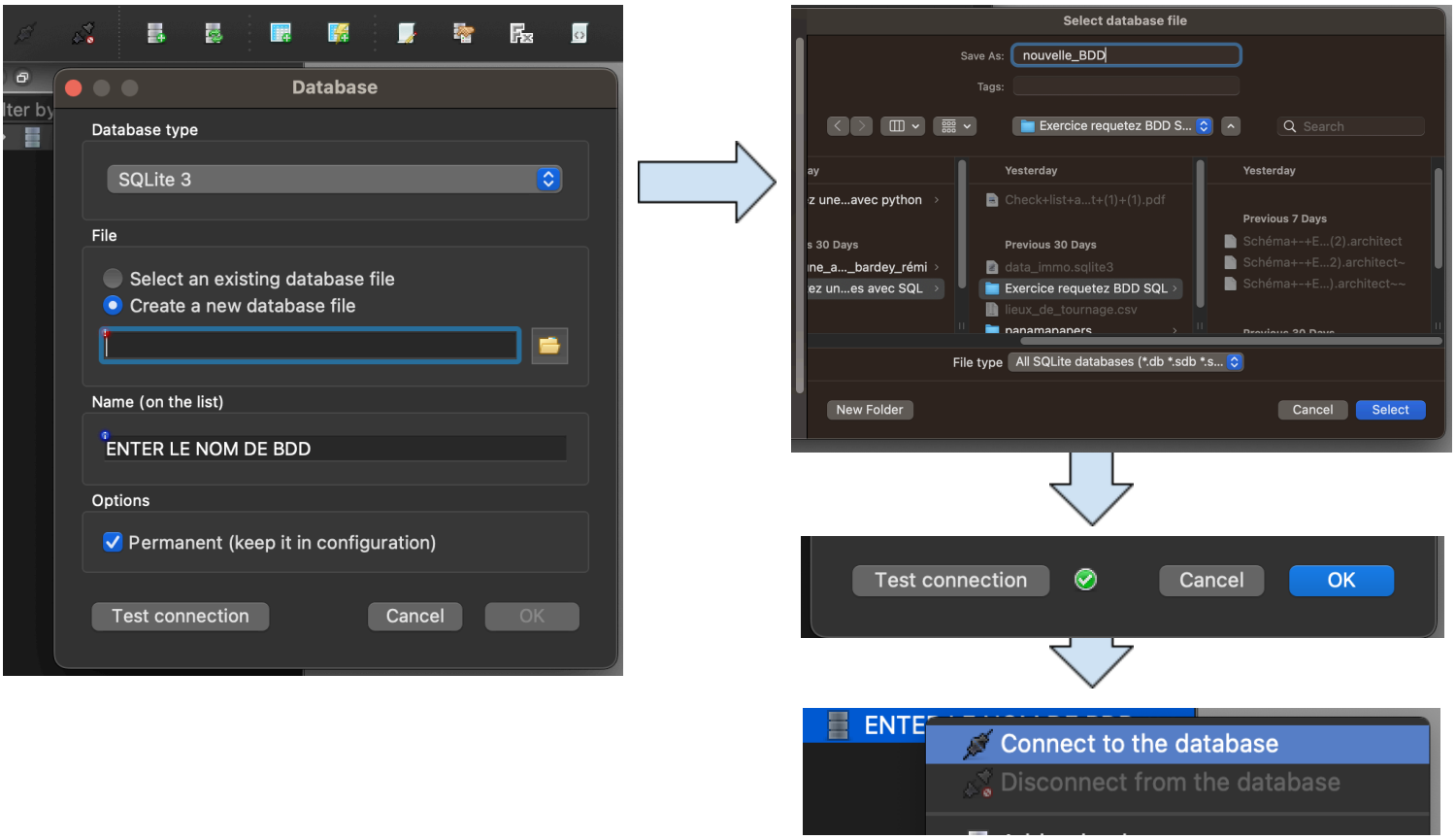




5. Création de la Base de données SQLite

5.1 Création de la BDD

Pour créer la base de données SQLite, j'ai utilisé l'interface graphique de SQLite. J'ai cliqué sur **"Add a new database"**, ce qui a ouvert un onglet dédié où j'ai choisi le chemin de stockage et nommé la BDD. Après avoir testé la connexion et validé, j'ai ensuite connecté la base de données en effectuant un clic droit sur celle-ci puis en sélectionnant **"Connect"**.



5.2 Importation du code SQL

Ensuite, dans l'éditeur SQL de SQLite, j'ai collé le code SQL généré à partir du schéma relationnel. Après avoir vérifié que la structure des tables était correcte (création des tables, définitions des colonnes et des contraintes telles que NOT NULL, les clés primaires et étrangères), j'ai exécuté le script pour créer la structure de la base de données.

```
CREATE TABLE Region (
Code_dep_code_commune VARCHAR(6) NOT NULL,
reg_code INTEGER NOT NULL,
reg_nom VARCHAR(40) NOT NULL,
aca_nom VARCHAR(20) NOT NULL,
dep_nom VARCHAR(20) NOT NULL,
com_nom_maj_court VARCHAR(50) NOT NULL,
dep_code VARCHAR(6) NOT NULL,
dep_nom_num VARCHAR(50) NOT NULL,
PRIMARY KEY (Code_dep_code_commune)
)
```

```
CREATE TABLE Contrat (
Contrat_ID INTEGER PRIMARY KEY,
No_voie INTEGER,
B_T_Q VARCHAR(1),
Type_de_voie VARCHAR(5),
Voie VARCHAR(50) NOT NULL,
Code_dep_code_commune VARCHAR(6) NOT NULL,
Code_postal VARCHAR(6) NOT NULL,
Surface INTEGER NOT NULL,
Type_local VARCHAR(15) NOT NULL,
Occupation VARCHAR(15) NOT NULL,
Type_contrat VARCHAR(30) NOT NULL,
Formule VARCHAR(20) NOT NULL,
Valeur_declaree_biens VARCHAR(15) NOT NULL,
Prix_cotisation_mensuel INTEGER NOT NULL,
FOREIGN KEY (code_dep_code_commune) REFERENCES region (code_dep_code_commune)
ON DELETE NO ACTION
ON UPDATE NO ACTION
);
```

5.3 Import des fichiers .CSV

Pour importer les données :

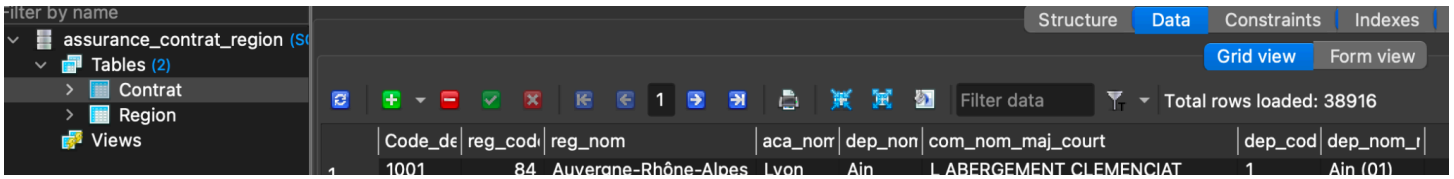
- J'ai effectué un clic droit sur la base de données dans SQLite et sélectionné **"Import"**.
- J'ai choisi le fichier CSV correspondant à la table souhaitée (d'abord **Region**, puis **Contrat**).
- Lors de l'import, SQLite offre plusieurs options :
    - Utiliser les premières lignes comme en-tête.
    - Modifier le séparateur pour bien identifier les colonnes (dans ce cas, les CSV utilisaient le point-virgule « ; » au lieu de la virgule).
- J'ai également vérifié le nombre de lignes importées afin de m'assurer que toutes les données étaient correctement intégrées.

Table Contrat :

assurance Table name: Contrat WITHOUT ROWID STRICT

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	Contrat_ID	INTEGER								NULL
2	No_voie	INTEGER								NULL
3	B_T_Q	VARCHAR (1)								NULL
4	Type_de_voie	VARCHAR (5)								NULL
5	Voie	VARCHAR (50)								NULL
6	Code_dep_code_commune	VARCHAR (6)								NULL
7	Code_postal	VARCHAR (6)								NULL
8	Surface	INTEGER								NULL
9	Type_local	VARCHAR (15)								NULL
10	Occupation	VARCHAR (15)								NULL
11	Type_contrat	VARCHAR (30)								NULL
12	Formule	VARCHAR (20)								NULL
13	Valeur _declaree_biens	VARCHAR (15)								NULL
14	Prix_cotisation_mensuel	INTEGER								NULL

Table Région :



assurance | Table name: Region | WITHOUT ROWID | STRICT

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	Code_dep_code_commune	VARCHAR (6)								NULL
2	reg_code	INTEGER								NULL
3	reg_nom	VARCHAR (40)								NULL
4	aca_nom	VARCHAR (20)								NULL
5	dep_nom	VARCHAR (20)								NULL
6	com_nom_maj_court	VARCHAR (50)								NULL
7	dep_code	VARCHAR (6)								NULL
8	dep_nom_num	VARCHAR (50)								NULL

5.4 Difficultés et ajustements

Un point important a été de respecter l'ordre des opérations :

- **Création des tables :** Il était nécessaire de créer la table **Region** avant la table **Contrat** en raison des dépendances (clé étrangère).
- **Importation des données :** De même, lors de l'import, la table **Region** devait être importée en premier, suivie de celle de **Contrat**, afin d'éviter des problèmes d'intégrité référentielle.

6. Analyse de la requête 2

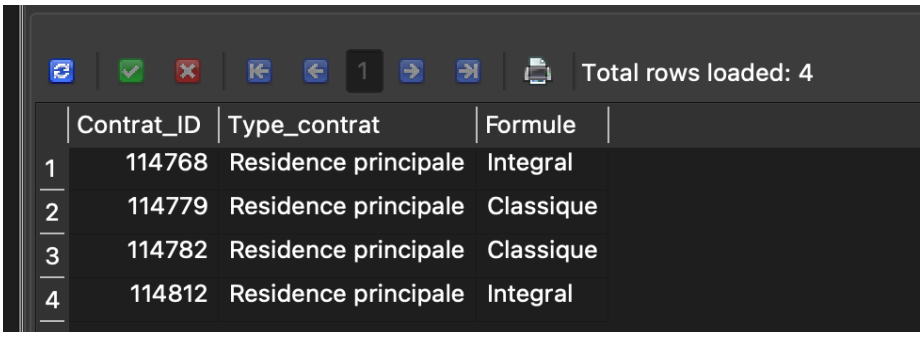
Pour extraire les contrats du département 71, j'ai d'abord essayé de filtrer sur la colonne `code_postal` de la table **Contrat** en recherchant les codes commençant par '71'. Cependant, cette méthode renvoyait des résultats erronés (par exemple, le code '7120' renvoyait une ville comme Sablière dans l'Ardèche).

Pour résoudre ce problème, j'ai préféré effectuer une jointure entre la table **Contrat** et la table **Region** en utilisant la colonne `Code_dep_code_commune`, et j'ai filtré sur la colonne `dep_code` de **Region** (valeur '71'). Cette approche garantit que seuls les contrats réellement associés au département 71 sont récupérés.

Requête 2 : Lister les numéros de contrats avec le type de contrat et leur formule pour les maisons du département 71.

```sql
SELECT c.contrat_ID, c.Type_contrat, c.Formule
FROM Contrat AS c
LEFT JOIN Region AS r
ON c.Code_dep_code_commune = r.Code_dep_code_commune
WHERE lower(c.type_local) LIKE 'maison%'
AND r.dep_code = '71';
```

Résultat :



Total rows loaded: 4

	Contrat_ID	Type_contrat	Formule
1	114768	Residence principale	Integral
2	114779	Residence principale	Classique
3	114782	Residence principale	Classique
4	114812	Residence principale	Integral

7. Conclusion

Ce projet m'a permis de comprendre l'importance cruciale de bien qualifier les types de données et de paramétrer correctement les colonnes dès la phase d'exploration. J'ai ainsi pu réaliser un code SQL fiable et maintenable en m'appuyant sur une analyse approfondie des données avant de me lancer dans l'implémentation.

Les points forts résident dans ma capacité à plonger dans l'analyse des données et à identifier les besoins réels, tandis que les difficultés ont principalement été d'ordre informatique, notamment face à des messages d'erreur imprévus lors de l'exécution des requêtes. Cette expérience m'a aussi mis en évidence la nécessité d'approfondir ma connaissance des outils et de m'entraîner davantage sur des aspects tels que l'ordre et l'indentation dans les requêtes SQL.

Globalement, ce projet m'a offert une vision plus large du métier de Data Analyst, en soulignant qu'aller à l'essentiel et maîtriser les fondations techniques sont des clés pour assurer la fiabilité et l'efficacité de la solution développée.