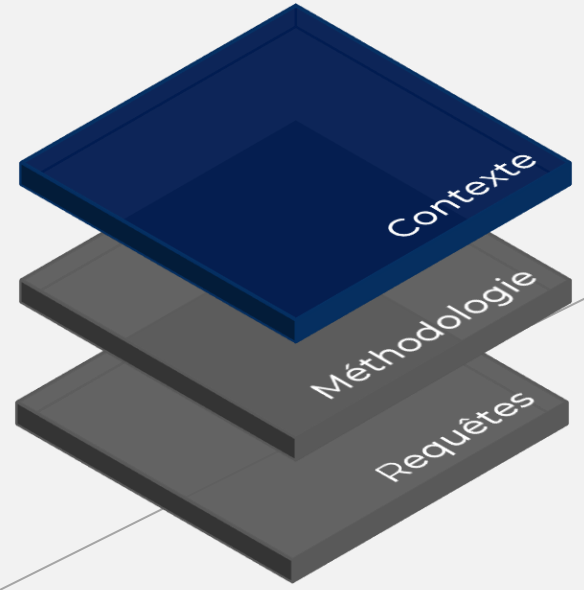
# Création et utilisation de la base de données

Hervé PITTET

Laplace Immo

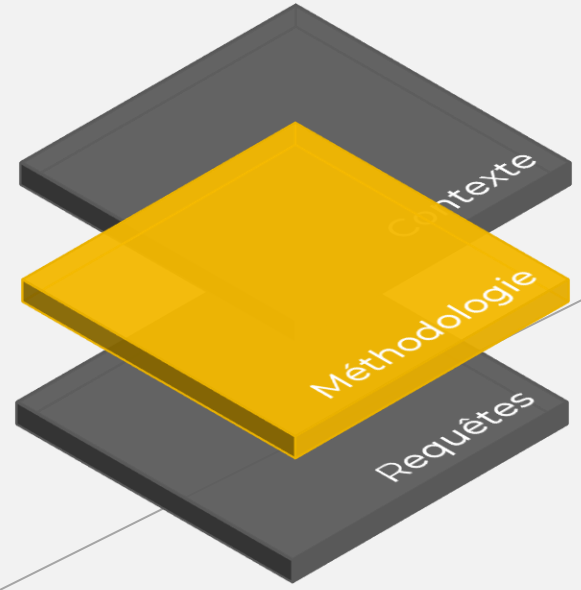

Méthodologie
Contexte
Requêtes




Modification de la base de données permettant de collecter les transactions immobilières et foncières en France

3 fichiers *.xlsx

- Données du site open data des Demandes de valeurs foncières
- Données de l'INSEE avec les recensements de la population
- Données de data.gouv.fr sur les régions (régions, départements, communes)




1
- Exploration des données initiales

2
- Elaboration du dictionnaire des données

3
- Elaboration du schéma relationnel

4

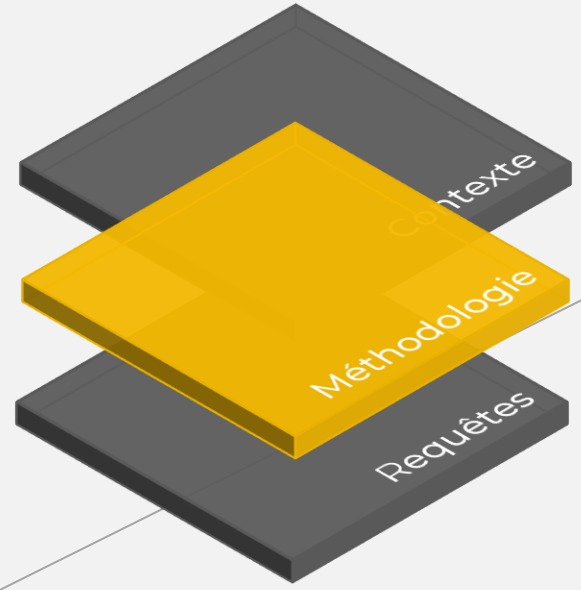
- Code SQL pour création des tables dans la base de données

5

- Modification des données initiales

6

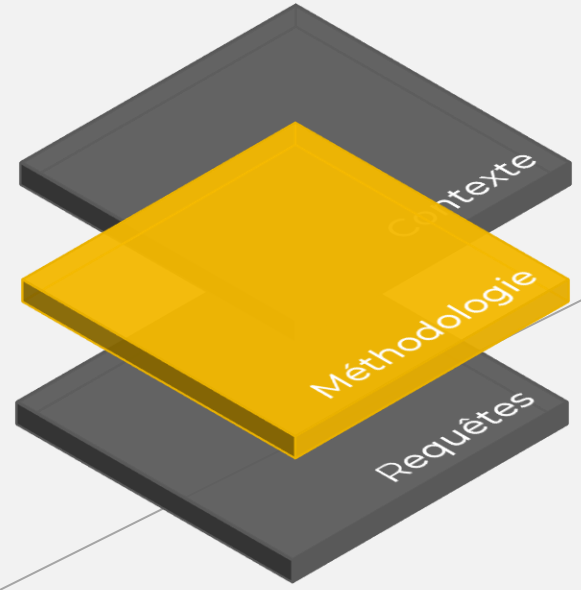
- Génération des fichiers *.csv et importation dans le SGBDR




# Exploration des données initiales

- Ouverture des fichiers *.xlsx
- Filtrer les colonnes
- Trier les colonnes (croissant/décroissant)
- Effectuer un profilage des données à l’aide de Power Query

Nom, Type, Taille, Clé, Description des colonnes




# Elaboration du dictionnaire des données

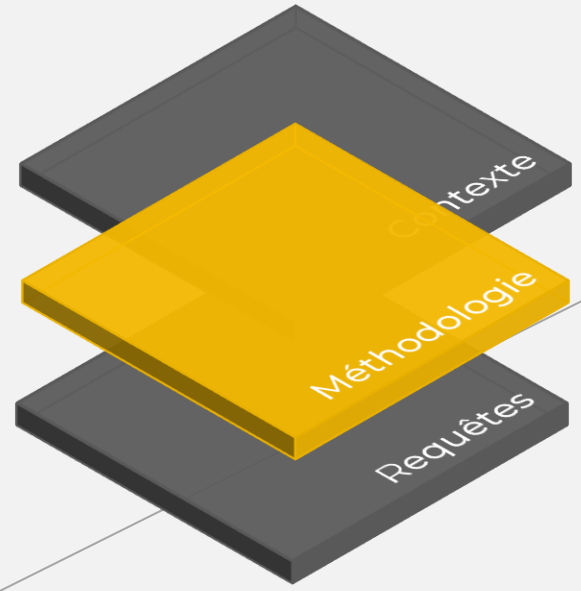
Définition des types de données :

INTEGER = Nombre entiers

FLOAT = Nombre décimal

CHAR = Chaîne de caractères à longueur fixe

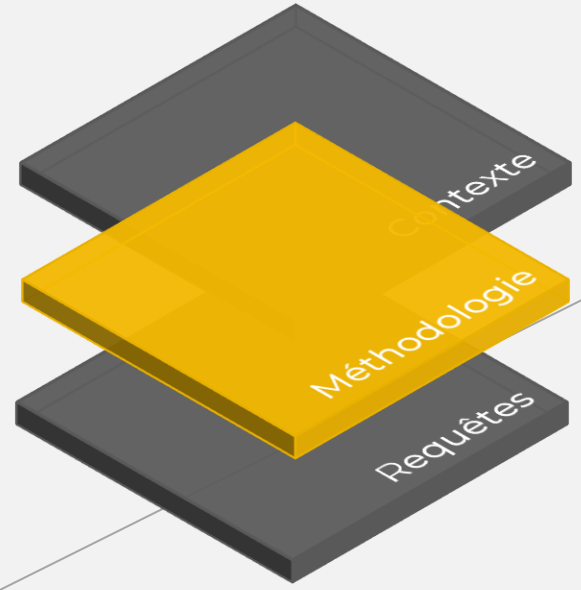
VARCHAR = Chaîne de caractères à longueur variable




# Elaboration du dictionnaire des données

## DICTIONNAIRE DES DONNÉES - Valeurs foncières

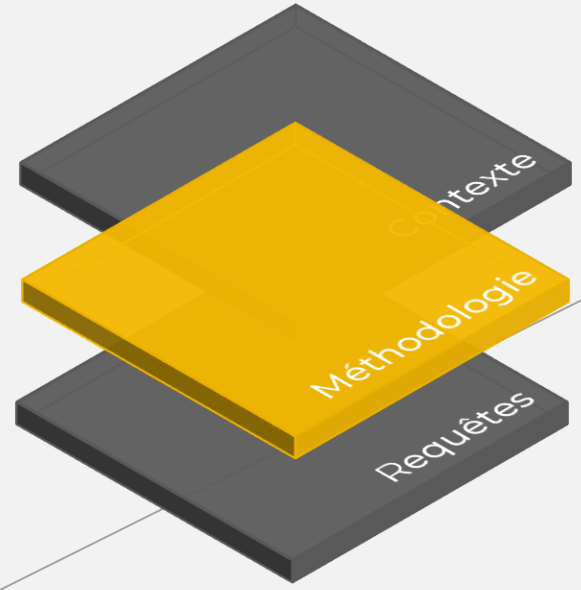
| | CODE | SIGNIFICATION | TYPE | LONGUEUR | Clé | NATURE | REGLE DE GESTION | REGLE DE CALCUL |
|---|---|---|---|---|---|---|---|---|
| TypeLocal.csv | codeTypeLocal | Code du type de local (1= Maison, 2 =Appartement) | INTEGER | 1 | Clé primaire | E | Ne doit pas être nul | |
| | typeLocal | Type du local | VARCHAR | | | E | Ne doit pas être nul | |
| Adresse.csv | idAdresse | ID de l'adresse | INTEGER | | Clé primaire | CO | Ne doit pas être nul | Concaténation de No Voie + B/T/Q + Type de voie + Voie + com_code puis incrémentation à partir de 1 |
| | noVoie | Numéro de la voie | INTEGER | | | E | | |
| | B_T_Q | Bis/Tiers/Quater de l'adresse | CHAR | 1 | | E | | |
| | typeVoie | Plusieurs valeurs (rue, avenue, chemin, etc.) | CHAR | 4 | | E | | |
| | nomVoie | Nom de la voie | VARCHAR | | | E | Ne doit pas être nul | |
| | idcodeDepartement_codeCommune | Clé unique pour les communes | INTEGER | 5 | Clé secondaire | CO | Ne doit pas être nul | Concaténation de dep_code + com_code de Référentiel géographique |
| Bien.csv | idBien | ID du bien | INTEGER | | Clé primaire | CO | Ne doit pas être nul | Concaténation de idAdresse + Préfixe de section + Section + No plan + No volume + 1er lot puis incrémentation à partir de 1 |
| | surfaceCarrez | Surface en m2 | FLOAT | | | E | Ne doit pas être nul | |
| | codeTypeLocal | Code du type de local (1= Maison, 2 =Appartement) | INTEGER | 1 | Clé secondaire | E | Ne doit pas être nul | |
| | surfaceReelle | Surface en m2 bâtie | INTEGER | | | E | Ne doit pas être nul | |
| | nbrePiecePrincipale | Nombre de pièces du bien | INTEGER | | | E | Ne doit pas être nul | |
| | idAdresse | ID de l'adresse | INTEGER | | Clé secondaire | CO | Ne doit pas être nul | Concaténation de No Voie + B/T/Q + Type de voie + Voie + com_code puis incrémentation à partir de 1 |
| Vente.csv | idTransaction | ID de la transaction immobilière | INTEGER | | Clé primaire | C | Ne doit pas être nul | Incrémentation à partir de 1 |
| | idBien | ID du bien | INTEGER | | Clé secondaire | CO | Ne doit pas être nul | Concaténation de idAdresse + Préfixe de section + Section + No plan + No volume + 1er lot puis incrémentation à partir de 1 |
| | Date mutation | Date de la transaction immobilière | DATE | | | E | jj/mm/aaaa + Ne doit pas être nul | |
| | Valeur fonciere | Valeur de la transaction immobilière | FLOAT | | | E | | |




# Elaboration du dictionnaire des données

## DICTIONNAIRE DES DONNÉES - Référentiel géographique

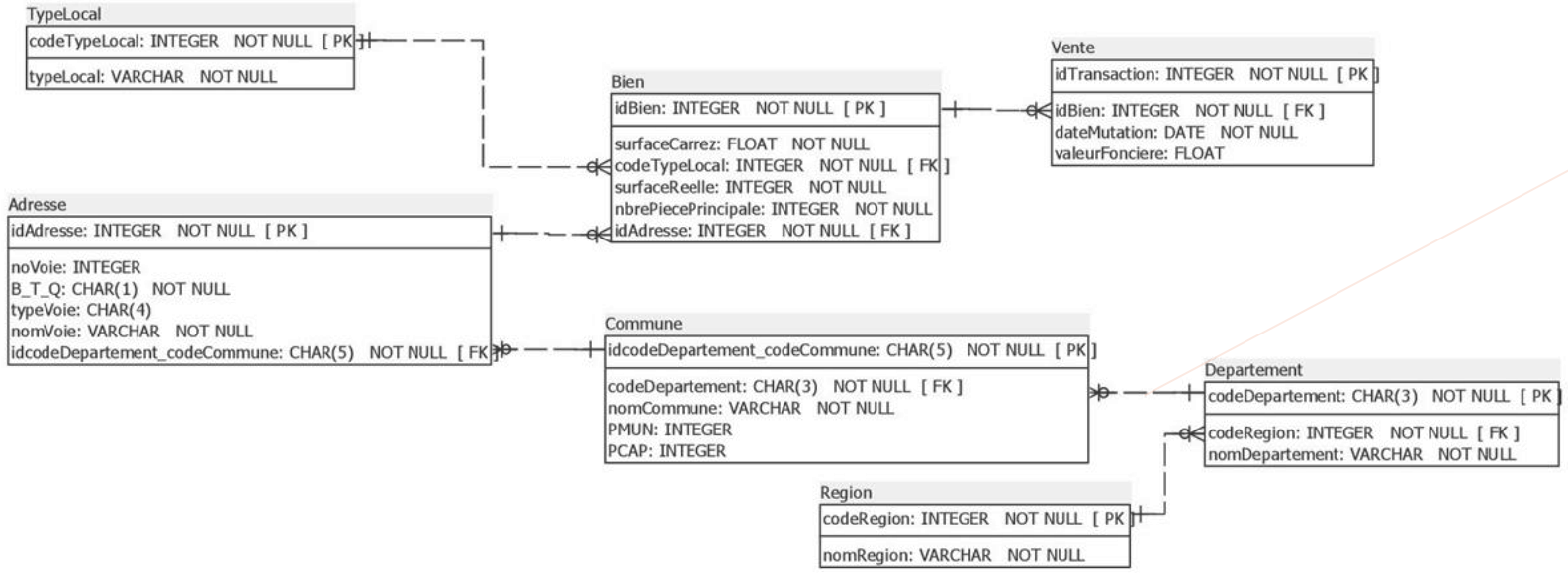
| | CODE | SIGNIFICATION | TYPE | LONGUEUR | Clé | NATURE | REGLE DE GESTION | REGLE DE CALCUL |
|---|---|---|---|---|---|---|---|---|
| Region.csv | codeRegion | Code de la région | INTEGER | 3 | Clé primaire | E | Ne doit pas être nul | |
| | nomRegion | Nom de la région | VARCHAR | | | E | Ne doit pas être nul | |
| Departement.csv | codeDepartement | Code du département | CHAR | 3 | Clé primaire | E | Ne doit pas être nul | |
| | codeRegion | Code de la région | INTEGER | | Clé secondaire | E | Ne doit pas être nul | |
| | nomDepartement | Nom du département | VARCHAR | | | E | Ne doit pas être nul | |
| Commune.csv | idcodeDepartement_codeCommune | Clé unique pour les communes | INTEGER | 5 | Clé primaire | CO | Ne doit pas être nul | Concaténation de dep_code + com_code de Référentiel géographique |
| | codeDepartement | Code du département | CHAR | 3 | Clé secondaire | E | Ne doit pas être nul | |
| | nomCommune | Nom de la commune en majuscule | VARCHAR | | | E | Ne doit pas être nul | |
| | PMUN | Valeur de la population municipale | INTEGER | | | E | | |
| | PCPAP | Valeur de la population comptée à part | INTEGER | | | E | | |

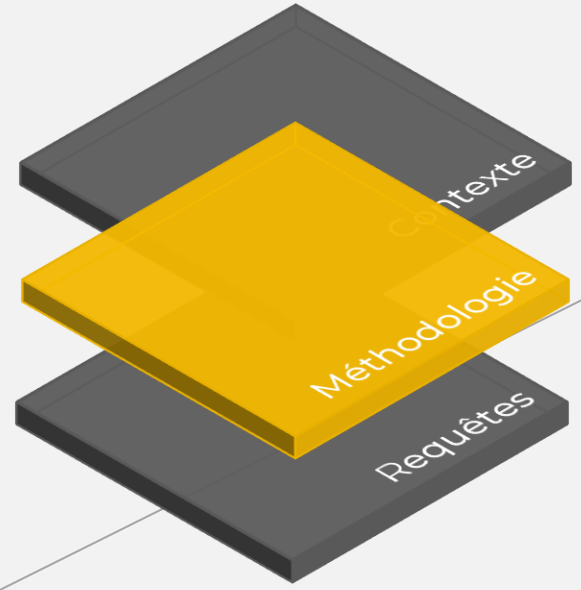



# Elaboration du schéma relationnel

## Logiciel : SQL Power Architect

## Création des tables et des relations





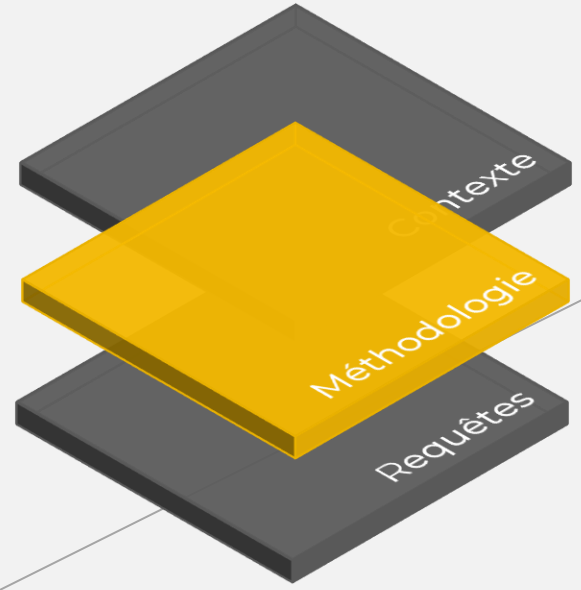

# Code SQL pour génération des tables

```
CREATE TABLE TypeLocal (
        codeTypeLocal INTEGER NOT NULL,
        typeLocal VARCHAR NOT NULL,
        CONSTRAINT codeTypeLocal_pk PRIMARY KEY
(codeTypeLocal)
);

CREATE TABLE Region (
        codeRegion INTEGER NOT NULL,
        nomRegion VARCHAR NOT NULL,
        CONSTRAINT codeRegion_pk PRIMARY KEY (codeRegion)
);

CREATE TABLE Departement (
        codeDepartement CHAR(3) NOT NULL,
        codeRegion INTEGER NOT NULL,
        nomDepartement VARCHAR NOT NULL,
        CONSTRAINT codeDepartement_pk PRIMARY KEY
(codeDepartement)
);
```

Suite du code sur le slide suivant

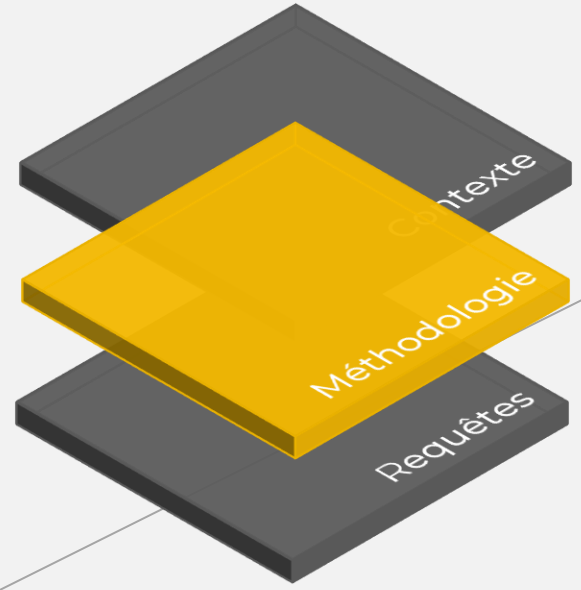



# Code SQL pour génération des tables

```
CREATE TABLE Commune (
        idcodeDepartement_codeCommune CHAR(5) NOT NULL,
        codeDepartement CHAR(3) NOT NULL,
        nomCommune VARCHAR NOT NULL,
        PMUN INTEGER,
        PCAP INTEGER,
        CONSTRAINT idcodeDepartement_codeCommune_pk
PRIMARY KEY (idcodeDepartement_codeCommune)
);

CREATE TABLE Adresse (
        idAdresse INTEGER NOT NULL,
        noVoie INTEGER,
        B_T_Q CHAR(1) NOT NULL,
        typeVoie CHAR(4),
        nomVoie VARCHAR NOT NULL,
        idcodeDepartement_codeCommune CHAR(5) NOT NULL,
        CONSTRAINT idAdresse_pk PRIMARY KEY (idAdresse)
);
```

Suite du code sur le slide suivant

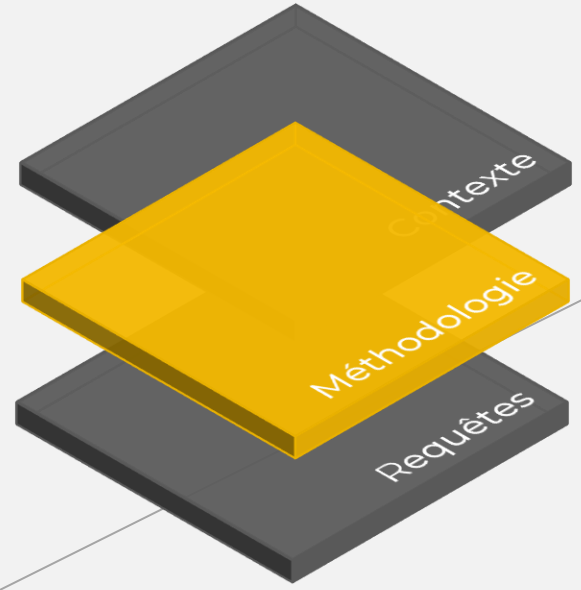



# Code SQL pour génération des tables

```
CREATE TABLE Bien (
        idBien INTEGER NOT NULL,
        surfaceCarrez FLOAT NOT NULL,
        codeTypeLocal INTEGER NOT NULL,
        surfaceReelle INTEGER NOT NULL,
        nbrePiecePrincipale INTEGER NOT NULL,
        idAdresse INTEGER NOT NULL,
        CONSTRAINT idBien_pk PRIMARY KEY (idBien)
);

CREATE TABLE Vente (
        idTransaction INTEGER NOT NULL,
        idBien INTEGER NOT NULL,
        dateMutation DATE NOT NULL,
        valeurFonciere FLOAT,
        CONSTRAINT idTransaction_pk PRIMARY KEY
(idTransaction)
);
```

Suite du code sur le slide suivant

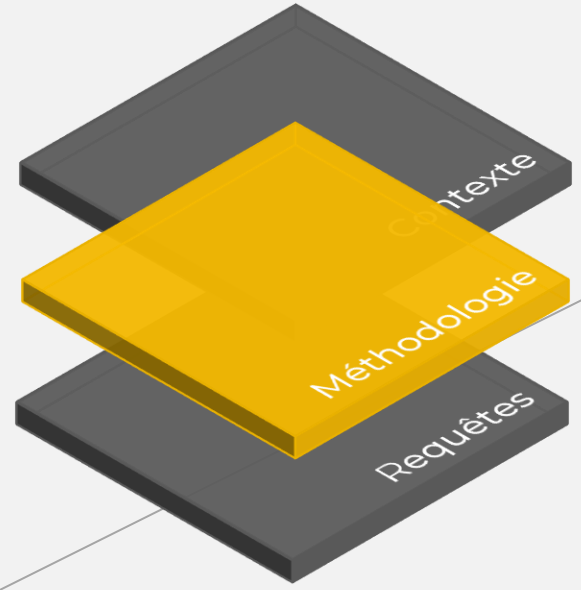



# Code SQL pour génération des tables

```
ALTER TABLE Bien ADD CONSTRAINT TypeLocal_Bien_fk
FOREIGN KEY (codeTypeLocal)
REFERENCES TypeLocal (codeTypeLocal)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE Departement ADD CONSTRAINT
Region_Departement_fk
FOREIGN KEY (codeRegion)
REFERENCES Region (codeRegion)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;
```

Suite du code sur le slide suivant

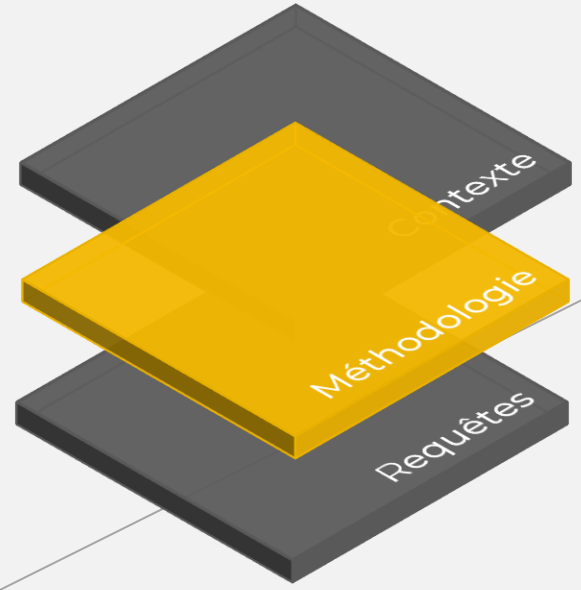



# Code SQL pour génération des tables

```
ALTER TABLE Commune ADD CONSTRAINT
Departement_Commune_fk
FOREIGN KEY (codeDepartement)
REFERENCES Departement (codeDepartement)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE Adresse ADD CONSTRAINT Commune_Adresse_fk
FOREIGN KEY (idcodeDepartement_codeCommune)
REFERENCES Commune (idcodeDepartement_codeCommune)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;
```

Suite du code sur le slide suivant

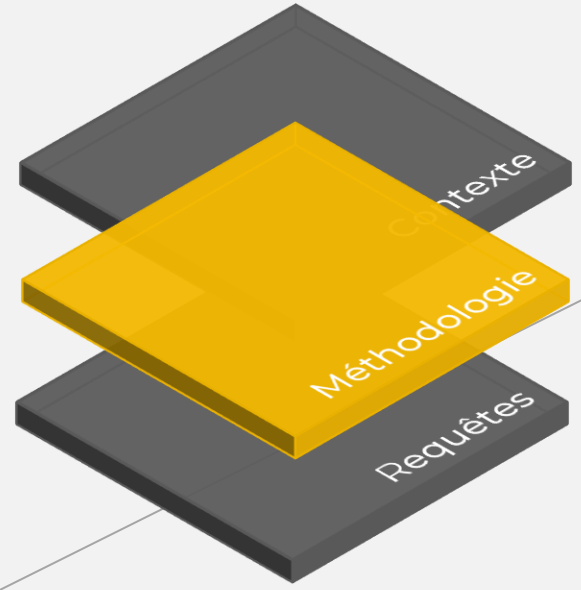



# Code SQL pour génération des tables

```
ALTER TABLE Bien ADD CONSTRAINT Adresse_Bien_fk
FOREIGN KEY (idAdresse)
REFERENCES Adresse (idAdresse)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE Vente ADD CONSTRAINT Bien_Vente_fk
FOREIGN KEY (idBien)
REFERENCES Bien (idBien)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;
```

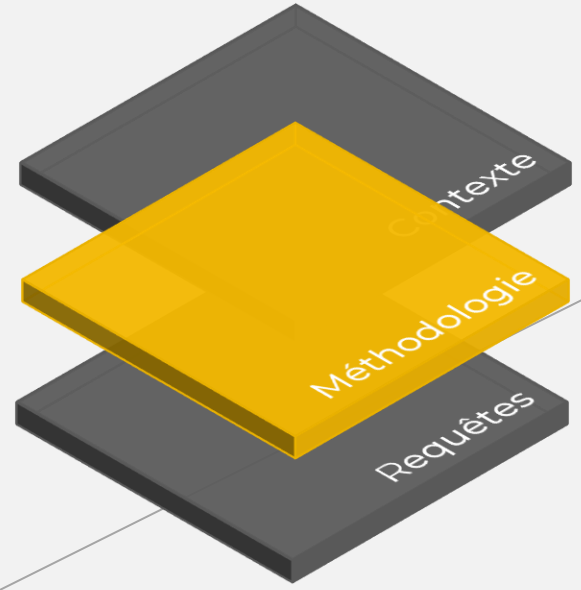
Suite du code sur le slide suivant




# Code SQL pour génération des tables

- laplace_immo_BDD (SQLite 3)
  - Tableaux (7)
    - Adresse
      - Colonnes (6)
        - idAdresse
        - noVoie
        - B_T_Q
        - typeVoie
        - nomVoie
        - idcodeDepartement_codeCommune
      - Index
      - Déclencheurs
    - Bien
      - Colonnes (6)
        - idBien
        - surfaceCarrez
        - codeTypeLocal
        - surfaceReelle
        - nbrePiecePrincipale
        - idAdresse
      - Index
      - Déclencheurs
    - Commune
      - Colonnes (5)
        - idcodeDepartement_codeCommune
        - codeDepartement
        - nomCommune
        - PMUN
        - PCAP
      - Index
      - Déclencheurs

- laplace_immo_BDD (SQLite 3)
  - Tableaux (7)
    - Fermer
    - Adresse
    - Bien
    - Commune
    - Departement
      - Colonnes (3)
        - codeDepartement
        - codeRegion
        - nomDepartement
      - Index
      - Déclencheurs
    - Region
      - Colonnes (2)
        - codeRegion
        - nomRegion
      - Index
      - Déclencheurs
    - TypeLocal
      - Colonnes (2)
      - Index
      - Déclencheurs
    - Vente
      - Colonnes (4)
      - Index
      - Déclencheurs

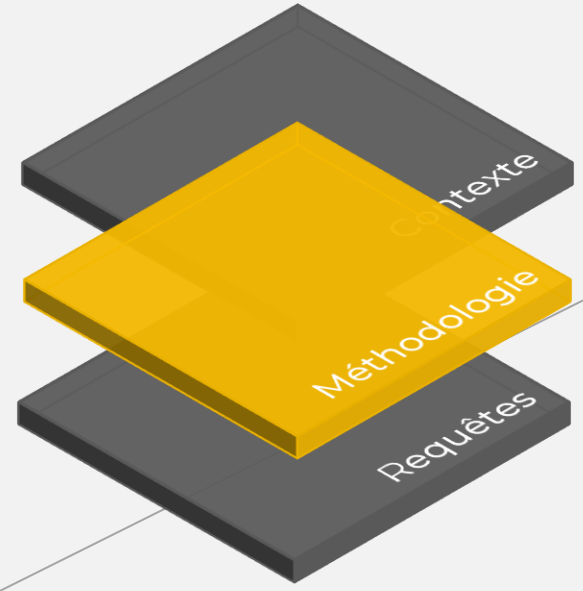



# Modification des données initiales

Logiciel : Power Query

- Suppression de colonnes
- Concaténation
- Fusion de requêtes
- Suppression des doublons
- Indexation

Respect du RGPD => pas de données personnelles




# Génération des fichiers csv / importation

```
select *
from Adresse
```

Table | Formulaire — Nombre de lignes chargées : 30793

```
select *
from departement
```

Table | Formulaire — Nombre de lignes chargées : 109

```
select *
from Bien
```

Table | Formulaire — Nombre de lignes chargées : 34169

```
select *
from Region
```

Table | Formulaire — Nombre de lignes chargées : 19

```
select *
from Commune
```

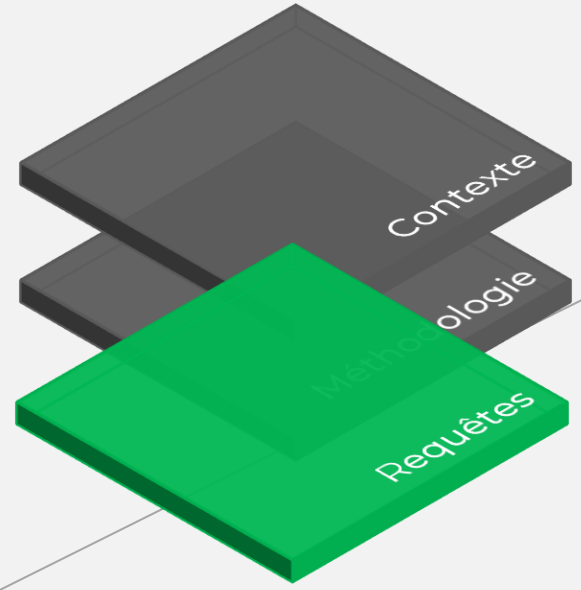
Table | Formulaire — Nombre de lignes chargées : 38916

```
select *
from TypeLocal
```

Table | Formulaire — Nombre de lignes chargées : 2

```
select *
from Vente
```

Table | Formulaire — Nombre de lignes chargées : 34169

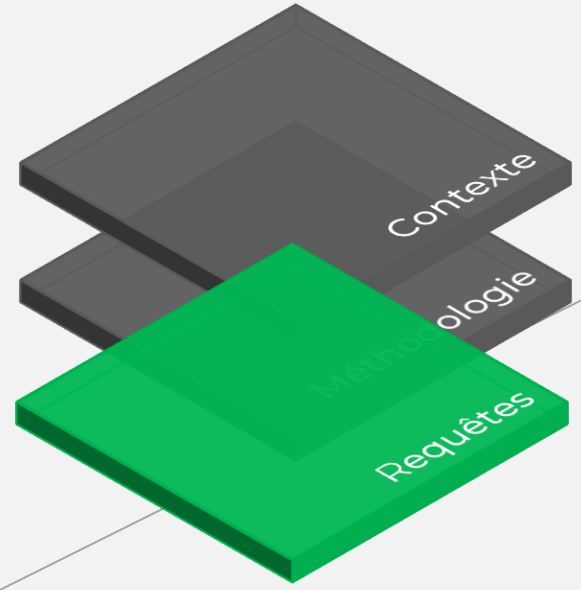



# Nombre total d'appartements vendus au 1er semestre 2020

```
SELECT COUNT(DISTINCT V.idBien) AS nombreAppartementsVendus-- permet de s'assurer que chaque bien est compté une seule fois
  FROM Vente V
      JOIN
      Bien B ON V.idBien = B.idBien-- la table Vente est jointe à la table Bien via idBien
      JOIN
      TypeLocal TL ON B.codeTypeLocal = TL.codeTypeLocal-- la table Bien est jointe à la table TypeLocal via codeTypeLocal pour vérifier le type de bien
  WHERE TL.typeLocal = 'Appartement'/* on filtre uniquement les transactions où le type de local (typeLocal) est Appartement */ AND
      strftime('%Y-%m-%d', substr(dateMutation, 7, 4) || '-' || substr(dateMutation, 4, 2) || '-' || substr(dateMutation, 1, 2)) BETWEEN '2020-01-01' AND '2020-06-30';
```

RESULTAT :

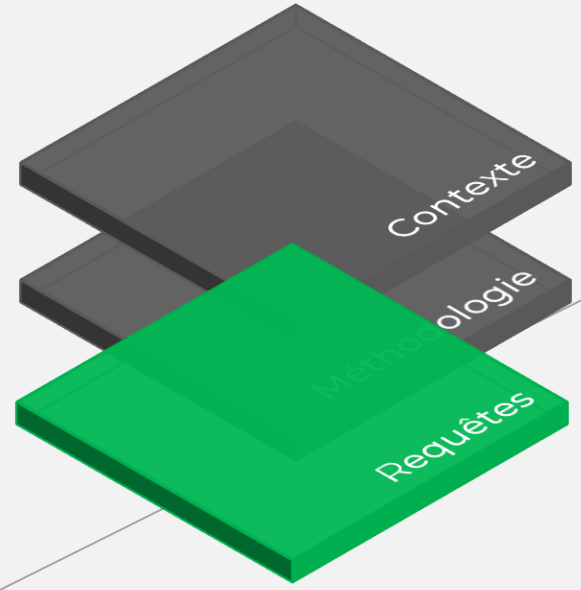
| | nombreAppartementsVendus |
|---|---|
| 1 | 31378 |




# Nombre de ventes d'appartement par région pour le 1er semestre 2020

```
SELECT R.nomRegion,-- sélection du nom des régions pour l'affichage du résultat demandé
    COUNT(DISTINCT V.idbIEN) AS nombreAppartementsVendus-- on compte le nombre de transactions
(idTransaction) par région pour déterminer le nombre de ventes
  FROM Vente V
    JOIN
    Bien B ON V.idBien = B.idBien-- la table Vente est joint à la table Bien via idBien
    JOIN
    TypeLocal TL ON B.codeTypeLocal = TL.codeTypeLocal-- la table Bien est jointe à TypeLocal pour vérifier qu'il
s'agit d'un appartement
    JOIN
    Adresse A ON B.idAdresse = A.idAdresse-- On joint la table Adresse via idAdresse pour obtenir les informations
d'adresse du bien
    JOIN
    Commune C ON A.idcodeDepartement_codeCommune = C.idcodeDepartement_codeCommune-- On joint la
table Commmune pour déterminer le code Commune
    JOIN
    Departement D ON C.codeDepartement = D.codeDepartement-- On joint la table Departement pour
déterminer le code Departement
    JOIN
    Region R ON D.codeRegion = R.codeRegion-- On joint la table Region pour déterminer la région associée à
chaque vente
 WHERE TL.typeLocal = 'Appartement'/* On filtre uniquement les biens de type Appartement */ AND
    strftime('%Y-%m-%d', substr(dateMutation, 7, 4) || '-' || substr(dateMutation, 4, 2) || '-' || substr(dateMutation, 1, 2)
) BETWEEN '2020-01-01' AND [2020-06-30]-- on filtre uniquement les transactions ayant eu lieu durant le 1er
semestre 2020
 GROUP BY R.nomRegion-- on groupe les résultats par région (nomRegion)
 ORDER BY nombreAppartementsVendus DESC;-- les résultats sont ordonnés par le nombre de ventes en ordre
décroissant
```

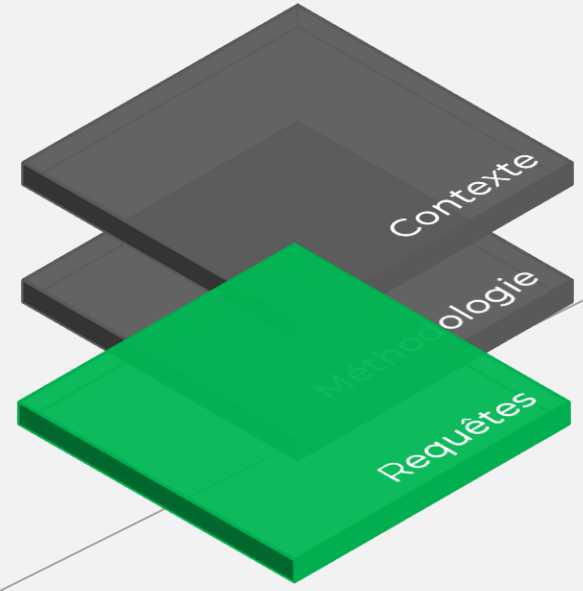
Résultat de la requête sur le slide suivant




# Nombre de ventes d'appartement par région pour le 1er semestre 2020

RESULTAT :

| | nomRegion | nombreAppartementsVendus |
|---|---|---|
| 1 | Ile-de-France | 13995 |
| 2 | Provence-Alpes-Côte d'Azur | 3649 |
| 3 | Auvergne-Rhône-Alpes | 3253 |
| 4 | Nouvelle-Aquitaine | 1932 |
| 5 | Occitanie | 1640 |
| 6 | Pays de la Loire | 1357 |
| 7 | Hauts-de-France | 1254 |
| 8 | Grand Est | 984 |
| 9 | Bretagne | 983 |
| 10 | Normandie | 862 |
| 11 | Centre-Val de Loire | 696 |
| 12 | Bourgogne-Franche-Comté | 376 |
| 13 | Corse | 223 |
| 14 | Martinique | 94 |
| 15 | La Réunion | 44 |
| 16 | Guyane | 34 |
| 17 | Guadeloupe | 2 |

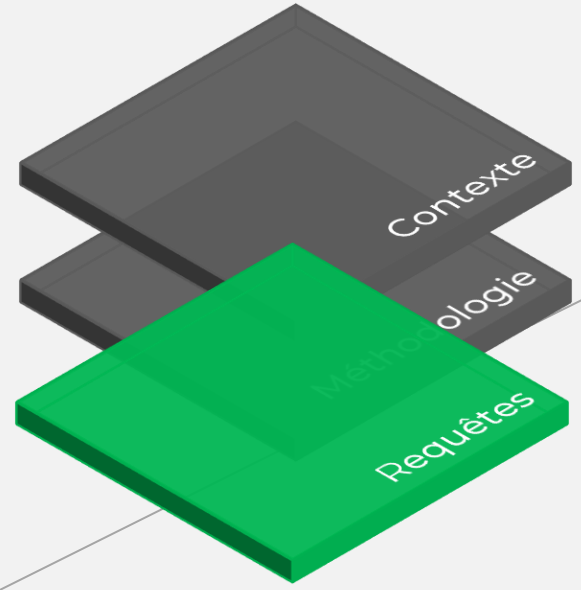



# Proportion des ventes d'appartements par le nombre de pièces

```
SELECT B.nbrePiecePrincipale AS nombrePieces, -- Sélection de la colonne de la table Bien
    ROUND(COUNT(V.idTransaction) /*Compte le nbre de transactions pour chaque groupe*/ * 100.0 /
SUM(COUNT(V.idTransaction) ) OVER () /*division par le total des transactions */, 2) AS proportionVentes
 FROM Vente V
    JOIN
    Bien B ON V.idBien = B.idBien
    JOIN
    TypeLocal TL ON B.codeTypeLocal = TL.codeTypeLocal
WHERE TL.typeLocal = 'Appartement'
GROUP BY B.nbrePiecePrincipale -- regroupe lessésultats par nbre de pièces
ORDER BY nombrePieces; -- ordonne les résultats en fonction du nbre de pièces
```

RESULTAT :

| | nombrePieces | proportionVentes |
|---|---|---|
| 1 | 0 | 0.1 |
| 2 | 1 | 21.48 |
| 3 | 2 | 31.18 |
| 4 | 3 | 28.57 |
| 5 | 4 | 14.21 |
| 6 | 5 | 3.55 |
| 7 | 6 | 0.65 |
| 8 | 7 | 0.17 |
| 9 | 8 | 0.05 |
| 10 | 9 | 0.03 |
| 11 | 10 | 0.01 |
| 12 | 11 | 0 |

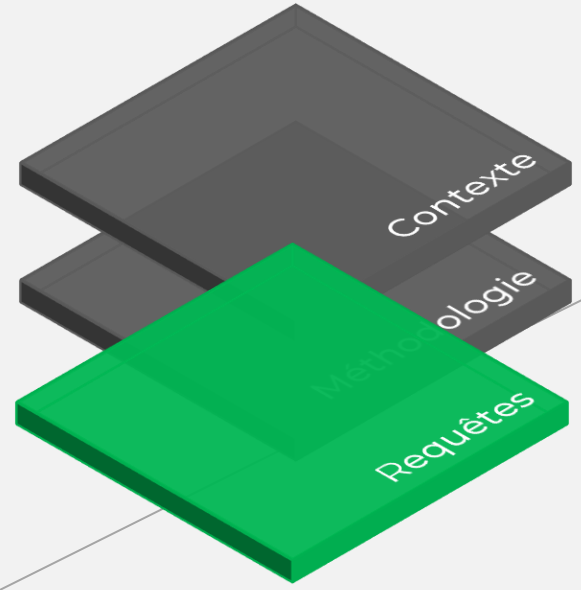



# Liste des 10 départements où le prix du mètre carré est le plus élevé

```
SELECT D.nomDepartement,-- Sélection des départements
    round(AVG(V.valeurFonciere / B.surfaceCarrez), 0) AS prixM2Moyen-- Calcul du prix au m2 pour chaque vente
 FROM Vente V
    JOIN
    Bien B ON V.idBien = B.idBien
    JOIN
    Adresse A ON B.idAdresse = A.idAdresse
    JOIN
    Commune C ON A.idcodeDepartement_codeCommune = C.idcodeDepartement_codeCommune
    JOIN
    Departement D ON C.codeDepartement = D.codeDepartement
 WHERE B.surfaceCarrez > 0-- On exclut les biens sans surface définie pour éviter la division par zéro
 GROUP BY D.nomDepartement-- On agrège les résultats par département (nomDepartement)
 ORDER BY prixM2Moyen DESC-- Les résultats sont triés par prixM2Moyen en ordre décroissant
 LIMIT 10;-- la requête limite les résultats aux 10 premiers départements avec les prix au mètre carré les plus élevés
```

RESULTAT :

| | nomDepartement | prixM2Moyen |
|---|---|---|
| 1 | Paris | 12084 |
| 2 | Hauts-de-Seine | 7301 |
| 3 | Val-de-Marne | 5428 |
| 4 | Haute-Savoie | 4781 |
| 5 | Alpes-Maritimes | 4756 |
| 6 | Seine-Saint-Denis | 4386 |
| 7 | Yvelines | 4276 |
| 8 | Rhône | 4100 |
| 9 | Corse-du-Sud | 4063 |
| 10 | Gironde | 3807 |

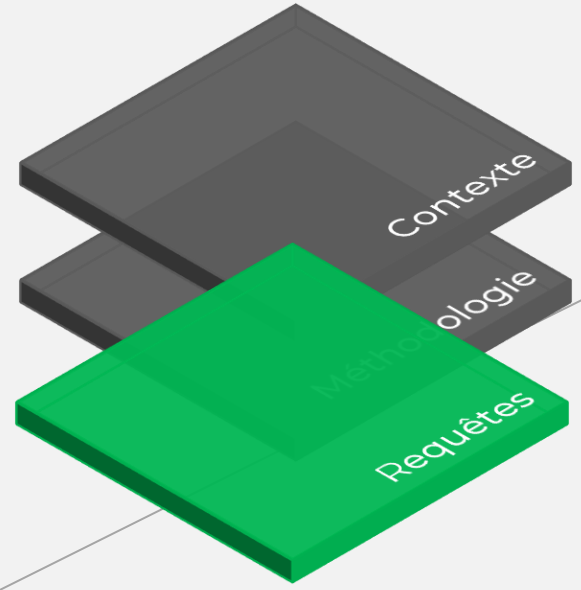



# Prix moyen du m2 d'une maison en Île-de-France

```
SELECT ROUND(AVG(V.valeurFonciere / B.surfaceCarrez), 0) AS prixMoyenM2-- Calcule la moyenne du prix au mètre carré, arrondie à 0 décimale
  FROM Vente V
      JOIN
      Bien B ON V.idBien = B.idBien
      JOIN
      Adresse A ON B.idAdresse = A.idAdresse
      JOIN
      Commune C ON A.idcodeDepartement_codeCommune = C.idcodeDepartement_codeCommune
      JOIN
      Departement D ON C.codeDepartement = D.codeDepartement
      JOIN
      Region R ON D.codeRegion = R.codeRegion
      JOIN
      TypeLocal TL ON B.codeTypeLocal = TL.codeTypeLocal
 WHERE TL.typeLocal = 'Maison'/* Filtre pour ne considérer que les maisons */ AND
      LOWER(R.nomRegion) LIKE '%ile%de%France%';-- Filtre pour ne considérer que les biens situés en Île-de-France
```

RESULTAT :

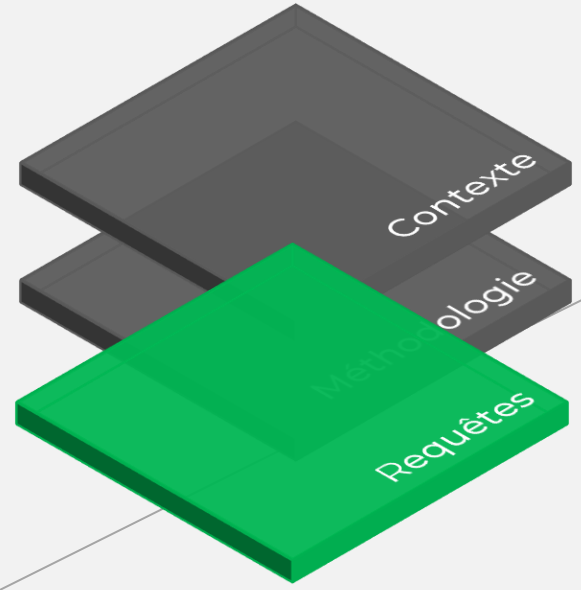
| | prixMoyenM2 |
|---|---|
| 1 | 3765 |




# Liste des 10 appartements les plus chers avec la région et le nombre de m2

```
SELECT B.idBien,
      R.nomRegion AS region,
      B.surfaceCarrez AS surfaceM2,
      MAX(V.valeurFonciere) AS prix
  FROM Vente V
      JOIN
      Bien B ON V.idBien = B.idBien
      JOIN
      Adresse A ON B.idAdresse = A.idAdresse
      JOIN
      Commune C ON A.idcodeDepartement_codeCommune = C.idcodeDepartement_codeCommune
      JOIN
      Departement D ON C.codeDepartement = D.codeDepartement
      JOIN
      Region R ON D.codeRegion = R.codeRegion
      JOIN
      TypeLocal TL ON B.codeTypeLocal = TL.codeTypeLocal
 WHERE TL.typeLocal = 'Appartement' AND
      CAST (TRIM(V.valeurFonciere) AS REAL) > 0-- S'assurer que la valeur foncière est numérique et supérieure à
zéro
 GROUP BY B.idBien,
         R.nomRegion,
         B.surfaceCarrez
 ORDER BY prix DESC
 LIMIT 10;
```

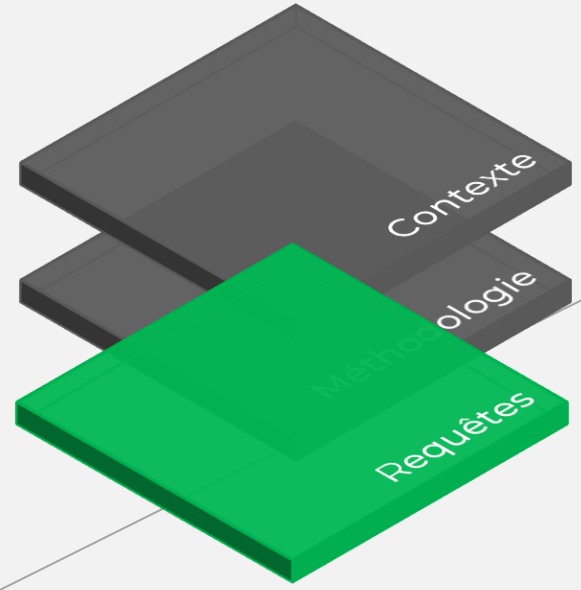
Résultat de la requête sur le slide suivant




# Liste des 10 appartements les plus chers avec la région et le nombre de m2

RESULTAT :

| | idBien | region | surfaceM2 | prix |
|---|---|---|---|---|
| 1 | 30636 | Ile-de-France | 9,1 | 9000000.0 |
| 2 | 6166 | Ile-de-France | 64 | 8600000 |
| 3 | 4288 | Ile-de-France | 20,55 | 8577713 |
| 4 | 8720 | Ile-de-France | 42,77 | 7620000 |
| 5 | 11236 | Ile-de-France | 253,3 | 7600000 |
| 6 | 19093 | Ile-de-France | 139,9 | 7535000 |
| 7 | 499 | Ile-de-France | 360,95 | 7420000 |
| 8 | 17663 | Ile-de-France | 595 | 7200000 |
| 9 | 2305 | Ile-de-France | 122,56 | 7050000 |
| 10 | 20354 | Ile-de-France | 79,38 | 6600000 |

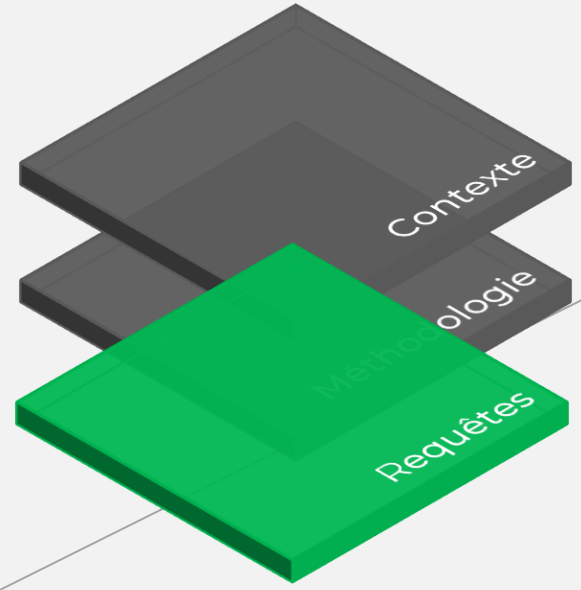



# Taux d'évolution du nombre de ventes entre le premier et le second trimestre de 2020

```
WITH VentesT1/* utilisation de la fonction WITH pour créer des sous-requêtes temporaires */ AS (
   SELECT COUNT( * ) AS nombreVentes
     FROM Vente V
    WHERE strftime/* Cette expression convertit la date dateMutation du format jj/mm/aaaa au format aaaa-mm-jj,
qui est compatible avec les opérations de date dans SQLite */('%Y-%m-%d', substr(dateMutation, 7, 4)/* extrait
l'année en 4 caractères dès le 7ème caractères */ || '-'/* concatène avec '-' */ || substr(dateMutation, 4, 2)/* extrait le
mois en 2 caractères dès le 4ème caractères */ || '-' || substr(dateMutation, 1, 2) ) BETWEEN '2020-01-01' AND  '2020-
03-31'
),
VentesT2 AS (
   SELECT COUNT( * ) AS nombreVentes
     FROM Vente V
    WHERE strftime('%Y-%m-%d', substr(dateMutation, 7, 4) || '-' || substr(dateMutation, 4, 2) || '-' ||
substr(dateMutation, 1, 2) ) BETWEEN '2020-04-01' AND '2020-06-30'
)
SELECT ROUND( ( (V2.nombreVentes - V1.nombreVentes) * 100.0 / V1.nombreVentes), 3) AS tauxEvolution
  FROM VentesT1 V1,
       VentesT2 V2;
```

RESULTAT :

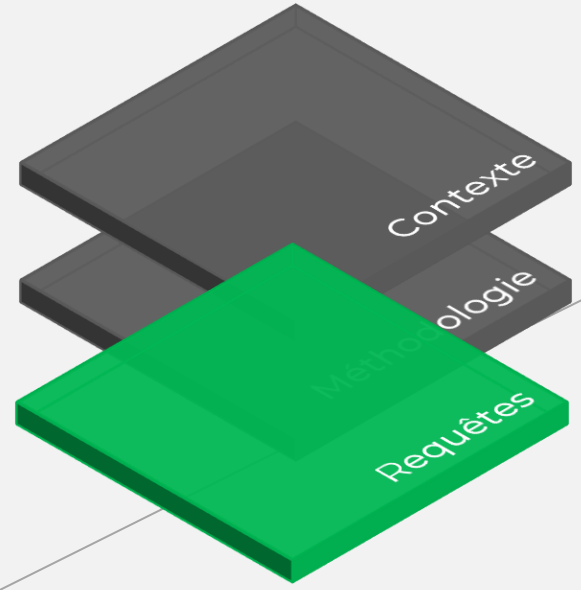
| | tauxEvolution |
|---|---|
| 1 | 3.678 |




# Le classement des régions par rapport au prix au mètre carré des appartements de plus de 4 pièces

```
WITH Appartements /* Cette sous-requête sélectionne uniquement les appartements ayant plus de 4 pièces, en
récupérant la surface et la valeur foncière pour calculer le prix au mètre carré, ainsi que la région du bien */ AS (
    SELECT B.idBien,
        B.surfaceCarrez,
        V.valeurFonciere,
        C.codeRegion
     FROM Bien B
        JOIN
        Vente V ON B.idBien = V.idBien
        JOIN
        Adresse A ON B.idAdresse = A.idAdresse
        JOIN
        Commune CO ON A.idcodeDepartement_codeCommune = CO.idcodeDepartement_codeCommune
        JOIN
        Departement D ON CO.codeDepartement = D.codeDepartement
        JOIN
        Region C ON D.codeRegion = C.codeRegion
        JOIN
        TypeLocal TL ON B.codeTypeLocal = TL.codeTypeLocal
    WHERE B.nbrePiecePrincipale > 4 AND
        TL.typeLocal = 'Appartement'
),
```

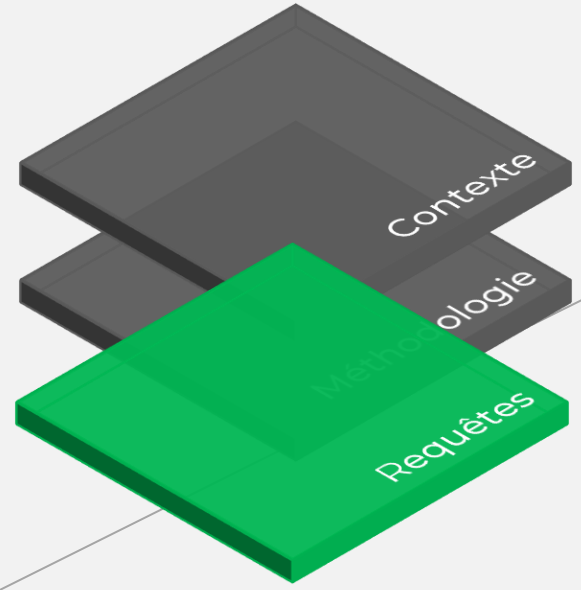
Suite du code sur le slide suivant




# Le classement des régions par rapport au prix au mètre carré des appartements de plus de 4 pièces

```
PrixM2ParRegion /* Cette sous-requête calcule le prix moyen au mètre carré pour chaque région en ne retenant
que les biens avec une surface non nulle */ AS (
    SELECT C.codeRegion,
        C.nomRegion,
        round(AVG(V.valeurFonciere / B.surfaceCarrez), 0) AS prixMoyenM2
     FROM Bien B
        JOIN
        Vente V ON B.idBien = V.idBien
        JOIN
        Adresse A ON B.idAdresse = A.idAdresse
        JOIN
        Commune CO ON A.idcodeDepartement_codeCommune = CO.idcodeDepartement_codeCommune
        JOIN
        Departement D ON CO.codeDepartement = D.codeDepartement
        JOIN
        Region C ON D.codeRegion = C.codeRegion
        JOIN
        TypeLocal TL ON B.codeTypeLocal = TL.codeTypeLocal
    WHERE B.nbrePiecePrincipale > 4 AND
        B.surfaceCarrez > 0 AND
        TL.typeLocal = 'Appartement'
    GROUP BY C.codeRegion,
          C.nomRegion
)
SELECT /* Cette requête affiche le classement des régions en fonction du prix moyen au mètre carré, ordonné du
plus cher au moins cher */ nomRegion,
      prixMoyenM2
  FROM PrixM2ParRegion
 ORDER BY prixMoyenM2 DESC;
```

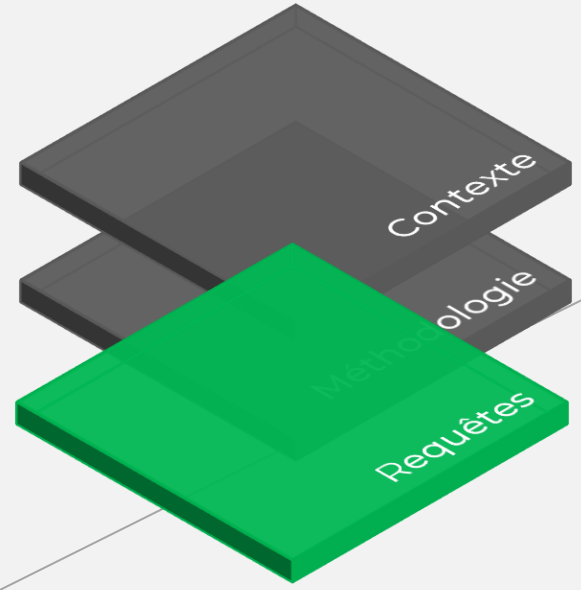
Résultat de la requête sur le slide suivant




# Le classement des régions par rapport au prix au mètre carré des appartements de plus de 4 pièces

RESULTAT :

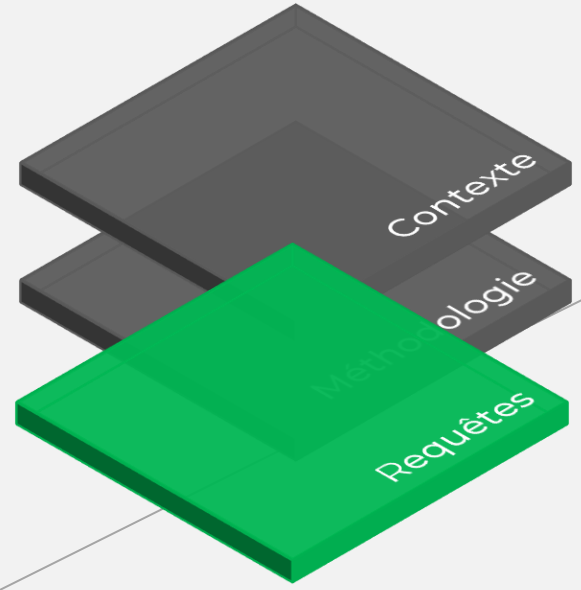
| | nomRegion | prixMoyenM2 |
|---|---|---|
| 1 | Ile-de-France | 8807 |
| 2 | La Réunion | 3660 |
| 3 | Provence-Alpes-Côte d'Azur | 3617 |
| 4 | Corse | 3118 |
| 5 | Auvergne-Rhône-Alpes | 2904 |
| 6 | Nouvelle-Aquitaine | 2477 |
| 7 | Bretagne | 2427 |
| 8 | Pays de la Loire | 2329 |
| 9 | Hauts-de-France | 2200 |
| 10 | Occitanie | 2107 |
| 11 | Normandie | 2026 |
| 12 | Grand Est | 1561 |
| 13 | Centre-Val de Loire | 1460 |
| 14 | Bourgogne-Franche-Comté | 1261 |
| 15 | Martinique | 575 |




# Liste des communes ayant eu au moins 50 ventes au 1er trimestre

```
-- 1. Filtrer les ventes pour ne sélectionner que celles qui ont eu lieu pendant le 1er trimestre (janvier à mars)
-- 2. Joindre les tables nécessaires pour obtenir les informations sur les communes associées à chaque vente
-- 3. Grouper les résultats par commune pour compter le nombre total de ventes par commune
-- 4. Filtrer les résultats pour ne garder que les communes ayant au moins 50 ventes
WITH VentesT1 /* Sous-requête temporaire pour réaliser les étapes 1 à 4 */ AS (
    SELECT C.nomCommune,
           COUNT( * ) AS nombreVentes
      FROM Vente V
           JOIN
           Bien B ON V.idBien = B.idBien
           JOIN
           Adresse A ON B.idAdresse = A.idAdresse
           JOIN
           Commune C ON A.idcodeDepartement_codeCommune = C.idcodeDepartement_codeCommune
     WHERE strftime('%Y-%m-%d', substr(V.dateMutation, 7, 4) || '-' || substr(V.dateMutation, 4, 2) || '-' ||
substr(V.dateMutation, 1, 2) ) BETWEEN '2020-01-01' AND [2020-03-31]
     GROUP BY C.nomCommune
    HAVING COUNT( * ) >= 50
)
SELECT /* Affiche le nom de la commune et le nombre total de ventes */ nomCommune,
       nombreVentes
  FROM VentesT1
 ORDER BY /* Classe les communes en ordre décroissant selon le nombre de ventes, les communes ayant le plus
de ventes apparaissant en premier */ nombreVentes DESC;
```

Résultat de la requête sur le slide suivant

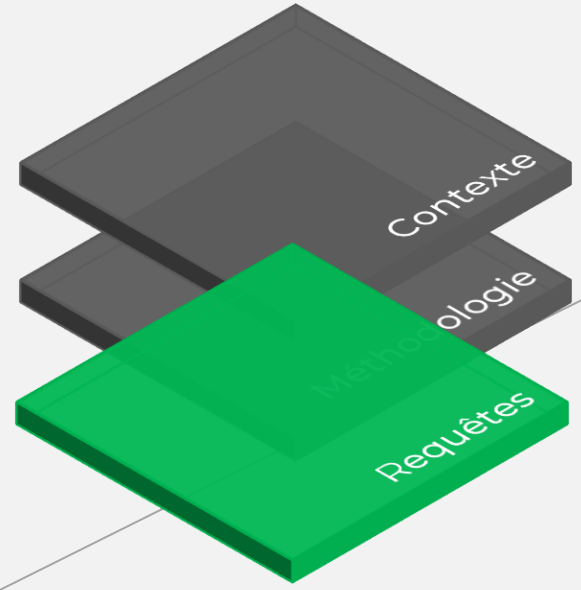



# Liste des communes ayant eu au moins 50 ventes au 1er trimestre

RESULTAT :

| | nomCommune | nombreVentes |
|---|---|---|
| 1 | PARIS 17 | 228 |
| 2 | PARIS 15 | 215 |
| 3 | PARIS 18 | 209 |
| 4 | NICE | 173 |
| 5 | PARIS 11 | 169 |
| 6 | PARIS 16 | 165 |
| 7 | BORDEAUX | 157 |
| 8 | PARIS 14 | 146 |
| 9 | PARIS 20 | 127 |
| 10 | NANTES | 119 |
| 11 | PARIS 19 | 116 |
| 12 | PARIS 12 | 110 |
| 13 | PARIS 10 | 109 |
| 14 | GRENOBLE | 106 |
| 15 | PARIS 9 | 106 |
| 16 | BOULOGNE BILLANCOURT | 99 |
| 17 | PARIS 13 | 94 |
| 18 | PARIS 7 | 87 |
| 19 | PARIS 6 | 86 |
| 20 | ASNIERES SUR SEINE | 81 |
| 21 | MARSEILLE 8 | 81 |
| 22 | COURBEVOIE | 80 |
| 23 | PARIS 3 | 79 |
| 24 | PARIS 5 | 79 |

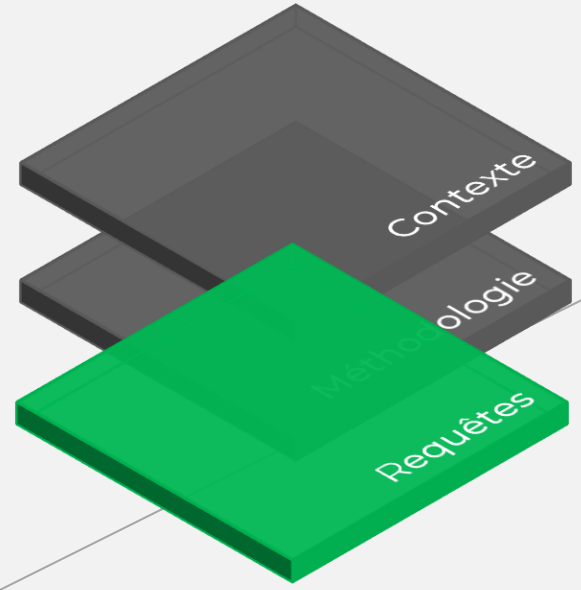
| | nomCommune | nombreVentes |
|---|---|---|
| 25 | TOULOUSE | 78 |
| 26 | ANTIBES | 77 |
| 27 | MARSEILLE 4 | 72 |
| 28 | MARSEILLE 1 | 71 |
| 29 | RUEIL MALMAISON | 68 |
| 30 | VINCENNES | 68 |
| 31 | LILLE | 67 |
| 32 | MARSEILLE 9 | 66 |
| 33 | MONTREUIL | 65 |
| 34 | ANGERS | 64 |
| 35 | NIMES | 63 |
| 36 | LA CIOTAT | 62 |
| 37 | PARIS 8 | 62 |
| 38 | SETE | 62 |
| 39 | PARIS 2 | 61 |
| 40 | RENNES | 61 |
| 41 | PARIS 4 | 60 |
| 42 | LEVALLOIS PERRET | 59 |
| 43 | TOULON | 59 |
| 44 | ST MAUR DES FOSSES | 56 |
| 45 | AJACCIO | 54 |
| 46 | VERSAILLES | 54 |
| 47 | PUTEAUX | 53 |
| 48 | ISSY LES MOULINEAUX | 50 |




# Différence en pourcentage du prix au mètre carré entre un appart. de 2 pièces et un appart. de 3 pièces

```
-- 1. Calculer le prix moyen au mètre carré pour les appartements de 2 pièces
-- 2. Calculer le prix moyen au mètre carré pour les appartements de 3 pièces
-- 3. Calculer la différence en pourcentage entre les deux valeurs
WITH PrixMoyen2Pieces /* Sous-requête PrixMoyen2Pieces : Calcul du prix moyen au mètre carré pour les
appartements de 2 pièces */ AS (
   SELECT AVG(V.valeurFonciere / B.surfaceCarrez) AS prixMoyenM2_2p -- Calcule le prix moyen au mètre carré en
divisant la valeur foncière par la surface Carrez pour chaque appartement de 2 pièces
     FROM Vente V
         JOIN
         Bien B ON V.idBien = B.idBien
         JOIN
         TypeLocal TL ON B.codeTypeLocal = TL.codeTypeLocal
    WHERE B.nbrePiecePrincipale = 2 AND
         B.surfaceCarrez > 0 AND
         TL.typeLocal = 'Appartement'
),
PrixMoyen3Pieces AS (
   SELECT AVG(V.valeurFonciere / B.surfaceCarrez) AS prixMoyenM2_3p
    FROM Vente V
         JOIN
         Bien B ON V.idBien = B.idBien
         JOIN
         TypeLocal TL ON B.codeTypeLocal = TL.codeTypeLocal
    WHERE B.nbrePiecePrincipale = 3 AND
         B.surfaceCarrez > 0 AND
         TL.typeLocal = 'Appartement'
)
```

Suite du code sur le slide suivant



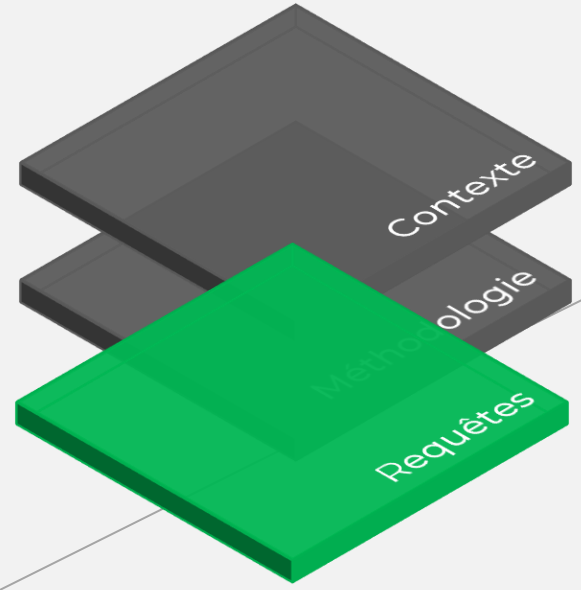

# Différence en pourcentage du prix au mètre carré entre un appart. de 2 pièces et un appart. de 3 pièces

```
SELECT ROUND( ( (P3.prixMoyenM2_3p - P2.prixMoyenM2_2p) / P2.prixMoyenM2_2p) * 100, 2) AS
differencePourcentage -- Calcule la différence en pourcentage arrondie à 2 décimales entre le prix moyen au
mètre carré des appartements de 2 pièces et de 3 pièces
  FROM PrixMoyen2Pieces P2,
      PrixMoyen3Pieces P3;

-- Si le prix au mètre carré des appartements de 3 pièces est supérieur à celui des 2 pièces, le pourcentage sera
positif. Inversement, si les appartements de 2 pièces ont un prix au mètre carré plus élevé, le pourcentage sera
négatif
```

RESULTAT :

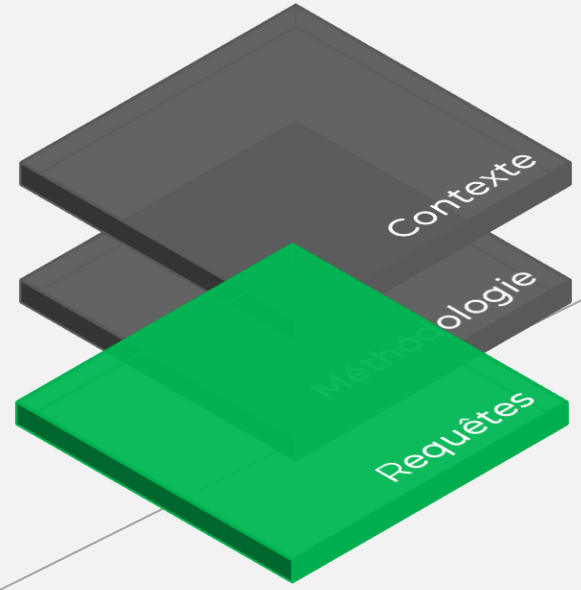
| | differencePourcentage |
|---|---|
| 1 | -12.68 |




# Les moyennes de valeurs foncières pour le top 3 des communes des départements 6, 13, 33, 59 et 69

```
-- 1. Filtrer les ventes pour les départements spécifiés.
-- 2. Calculer la moyenne des valeurs foncières pour chaque commune.
-- 3. Classer les communes par moyenne des valeurs foncières.
-- 4. Sélectionner les trois communes ayant les moyennes les plus élevées pour chaque département.
WITH MoyennesCommune /* Sous-requête MoyennesCommune */ AS (
   SELECT C.nomCommune, -- Calcule la moyenne des valeurs foncières pour chaque commune, regroupée par le
nom de la commune et le département
          D.codeDepartement,
          AVG(V.valeurFonciere) AS moyenneValeurFonciere
     FROM Vente V
          JOIN -- Permet de relier les tables Vente, Bien, Adresse, Commune, et Departement en fonction de leur
relation
          Bien B ON V.idBien = B.idBien
          JOIN
          Adresse A ON B.idAdresse = A.idAdresse
          JOIN
          Commune C ON A.idcodeDepartement_codeCommune = C.idcodeDepartement_codeCommune
          JOIN
          Departement D ON C.codeDepartement = D.codeDepartement
    WHERE D.codeDepartement IN (6, 13, 33, 59, 69) -- Filtre les ventes pour ne retenir que celles appartenant aux
départements spécifiés
    GROUP BY C.nomCommune, -- Regroupe les résultats par commune et département
             D.codeDepartement
),
ClassementCommune /* Sous-requête ClassementCommune */ AS (
   SELECT nomCommune,
          codeDepartement,
          moyenneValeurFonciere,
          ROW_NUMBER() OVER /* Classe les communes par moyenne de valeur foncière au sein de chaque
département */ (PARTITION BY codeDepartement /* Classe les communes séparément pour chaque
département */ ORDER BY moyenneValeurFonciere DESC /* Classe les moyennes de valeur foncière de manière
décroissante */) AS rang
     FROM MoyennesCommune
)
```

Suite du code sur le slide suivant

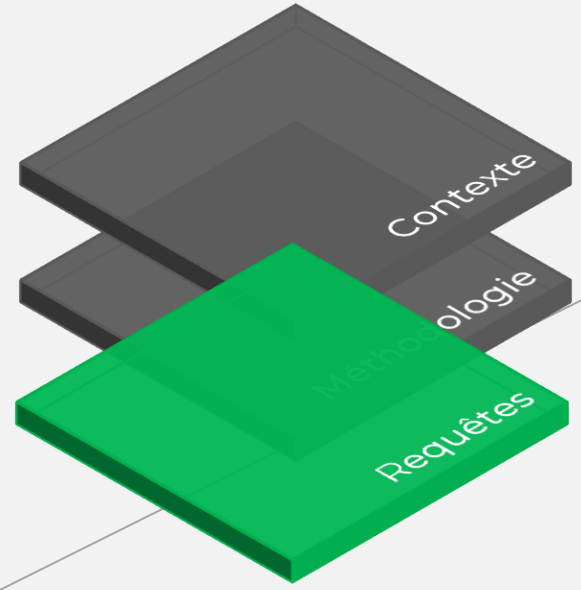



# Les moyennes de valeurs foncières pour le top 3 des communes des départements 6, 13, 33, 59 et 69

```
SELECT nomCommune, -- Sélection finale
      codeDepartement,
      ROUND(moyenneValeurFonciere, 0) AS moyenneValeurFonciere
  FROM ClassementCommune
 WHERE rang <= 3 -- Sélectionne uniquement les trois communes ayant les moyennes les plus élevées pour chaque département
 ORDER BY codeDepartement, -- Trie les résultats par département et moyenne de valeur foncière en ordre décroissant
         moyenneValeurFonciere DESC;
```

RESULTAT :

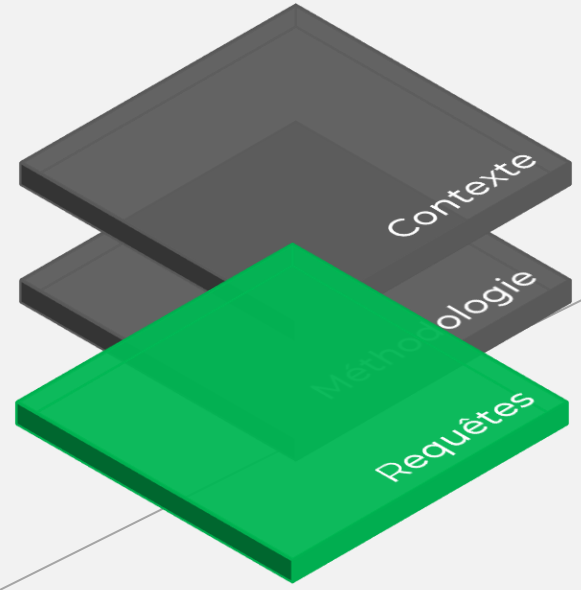
| | nomCommune | codeDepartement | moyenneValeurFonciere |
|---|---|---|---|
| 1 | GIGNAC LA NERTHE | 13 | 330000 |
| 2 | ST SAVOURNIN | 13 | 314425 |
| 3 | CASSIS | 13 | 313417 |
| 4 | LEGE CAP FERRET | 33 | 549501 |
| 5 | VAYRES | 33 | 335000 |
| 6 | ARCACHON | 33 | 307436 |
| 7 | BERSEE | 59 | 433202 |
| 8 | CYSOING | 59 | 408550 |
| 9 | HALLUIN | 59 | 322250 |
| 10 | ST JEAN CAP FERRAT | 6 | 968750 |
| 11 | EZE | 6 | 655000 |
| 12 | MOUANS SARTOUX | 6 | 476898 |
| 13 | VILLE SUR JARNIOUX | 69 | 485300 |
| 14 | LYON 2 | 69 | 455217 |
| 15 | LYON 6 | 69 | 426968 |




# Les 20 comm.avec le plus de transac. pour 1000 hab. pour les comm. qui dépassent les 10 000 hab.

```
-- 1. Calculer le nombre total d'habitants pour chaque commune (PMUN + PCAP).
-- 2. Filtrer les communes ayant plus de 10 000 habitants.
-- 3. Compter le nombre de transactions pour chaque commune.
-- 4. Calculer le nombre de transactions pour 1 000 habitants.
-- 5. Classer les communes en fonction de ce ratio et sélectionner les 20 premières
WITH CommunePopulation /* Sous-requête CommunePopulation */ AS (
   SELECT C.nomCommune,
        C.idcodeDepartement_codeCommune,
        (C.PMUN + C.PCAP) AS populationTotale -- Calcule la population totale pour chaque commune en
additionnant les colonnes PMUN (population municipale) et PCAP (population comptée à part)
     FROM Commune C
    WHERE populationTotale > 10000 -- Filtre les communes pour ne conserver que celles ayant plus de 10 000
habitants
),
TransactionsParCommune /* Sous-requête TransactactionsParCommune */ AS (
   SELECT C.nomCommune,
        COUNT(V.idTransaction) AS nombreTransactions -- Compte le nombre total de transactions pour chaque
commune
     FROM Vente V
        JOIN
        Bien B ON V.idBien = B.idBien
        JOIN
        Adresse A ON B.idAdresse = A.idAdresse
        JOIN
        Commune C ON A.idcodeDepartement_codeCommune = C.idcodeDepartement_codeCommune
    GROUP BY C.nomCommune -- Regroupe les transactions par commune
),
```

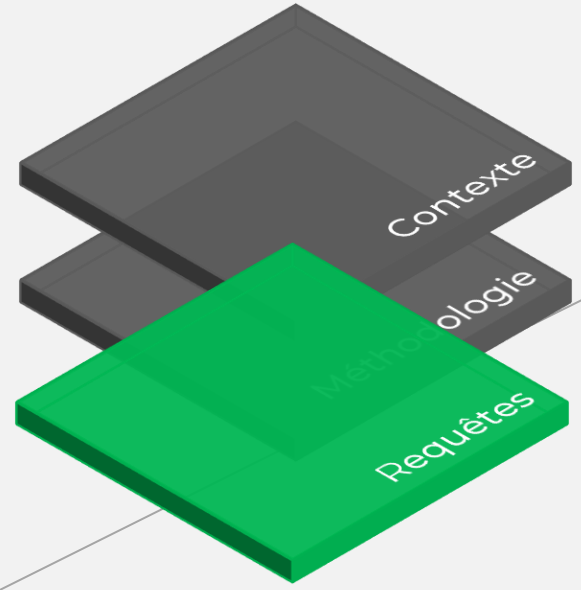
Suite du code sur le slide suivant




# Les 20 comm.avec le plus de transac. pour 1000 hab. pour les comm. qui dépassent les 10 000 hab.

```
TransactionsPar1000Habitants /* Sous-requête TransactionsPar1000Habitants */ AS (
   SELECT CP.nomCommune,
          TP.nombreTransactions,
          CP.populationTotale,
          (TP.nombreTransactions * 1000.0) / CP.populationTotale AS transactionsPour1000 -- Calcule le nombre de
transactions pour 1 000 habitants
     FROM TransactionsParCommune TP
          JOIN
          CommunePopulation CP ON TP.nomCommune = CP.nomCommune -- Associe chaque commune avec sa
population calculée précédemment
)
SELECT /* Sélection finale */ nomCommune,
       ROUND(transactionsPour1000, 2) AS transactionsPour1000,
       populationTotale,
       nombreTransactions
  FROM TransactionsPar1000Habitants
 ORDER BY transactionsPour1000 DESC -- Trie les communes par nombre de transactions pour 1 000 habitants,
de manière décroissante
 LIMIT 20; -- Limite les résultats aux 20 premières communes
```

Résultat de la requête sur le slide suivant




# Les 20 comm.avec le plus de transac. pour 1000 hab. pour les comm. qui dépassent les 10 000 hab.

RESULTAT :

| | nomCommune | transactionsPour1000 | populationTotale | nombreTransactions |
|---|---|---|---|---|
| 1 | PARIS 2 | 5.84 | 21735 | 127 |
| 2 | PARIS 1 | 4.92 | 16055 | 79 |
| 3 | PARIS 3 | 4.69 | 34306 | 161 |
| 4 | ARCACHON | 4.62 | 11898 | 55 |
| 5 | LA BAULE | 4.58 | 16797 | 77 |
| 6 | PARIS 4 | 4.08 | 29390 | 120 |
| 7 | ROQUEBRUNE CAP MARTIN | 3.99 | 13041 | 52 |
| 8 | PARIS 8 | 3.83 | 36250 | 139 |
| 9 | SANARY SUR MER | 3.5 | 17160 | 60 |
| 10 | PARIS 9 | 3.43 | 60563 | 208 |
| 11 | LA LONDE LES MAURES | 3.43 | 10776 | 37 |
| 12 | PARIS 6 | 3.38 | 41171 | 139 |
| 13 | ST CYR SUR MER | 3.24 | 11725 | 38 |
| 14 | CHANTILLY | 3.13 | 11178 | 35 |
| 15 | PORNICHET | 3.06 | 11440 | 35 |
| 16 | ST MANDE | 3.06 | 22576 | 69 |
| 17 | PARIS 10 | 3.04 | 86863 | 264 |
| 18 | MENTON | 2.94 | 30981 | 91 |
| 19 | ST HILAIRE DE RIEZ | 2.87 | 11501 | 33 |
| 20 | VINCENNES | 2.81 | 50230 | 141 |

# Merci !