Gestion de données

Philippe Helluy

18 septembre 2025

Table des matières

Objectifs

- Les fondamentaux : bases de données relationnelles, SGDB, SQL, algèbre relationnelle, Python Pandas.
- Avancées récentes : noSQL
- outils cloud (API, JSON), Spark.
- Gestion des données semi-structurées : LLM, RAG.

SQL

- Introduction à SQL.
- Création et gestion des tables.
- Requêtes SQL pour l'interrogation des données.
- Un peu de maths : algèbre relationnelle.

Introduction à SQL et SQLite

- SQL (Structured Query Language) est un langage de programmation déclaratif (par opposition à impératif) conçu pour gérer et manipuler des bases de données relationnelles.
- SQLite est un système de gestion de base de données relationnelle (SGBD) léger et autonome, largement utilisé pour des applications embarquées et mobiles.
- SQLite est sans serveur (*serverless*). Systèmes plus riches de type client-serveur : PostgreSQL, MySQL, etc.

Exemple

Nom	Prénom	Nom Animal	Espèce	Âge
Dupont	Jean	Rex	Chien	5
Martin	Claire	Miaou	Chat	3
Martin	Claire	Panpan	Lapin	2

Table – Liste des animaux

- **Redondance** : Les informations du client sont répétées pour chaque animal qu'il possède (gaspillage de mémoire).
- **Possibilité d'erreur** : Mise à jour difficile des informations du client, risque d'incohérence des données.
- **Intégrité des données** : Pas de vérification visible pour s'assurer que chaque animal est associé à un client valide.

Solution : Trois tables séparées

Tables `client` et `animal`

Table `client`

id	nom	prenom
1	Dupont	Jean
2	Martin	Claire

Table `animal`

id	nom	espece	age
1	Rex	Chien	5
2	Miaou	Chat	3
3	Panpan	Lapin	2

Solution : Table `maitre` et Concepts clés

Table `maitre`

Table `maitre`

animal_id	client_id
1	1
2	2
3	2

Concepts clés

- **Clé primaire (PRIMARY KEY)** : Chaque table a une clé primaire qui identifie de manière unique chaque enregistrement. Par exemple, `id` dans les tables `client` et `animal`, et la combinaison `(animal_id, client_id)` dans la table `maitre`.
- **Clé étrangère (FOREIGN KEY)** : Les colonnes `animal_id` et `client_id` dans la table `maitre` sont des clés étrangères qui référencent les clés primaires des tables `animal` et `client`, respectivement.

Introduction à la Normalisation

- La normalisation est un processus de structuration d'une base de données pour minimiser la redondance et éviter les anomalies.
- Objectifs principaux :
  - Réduction de la redondance des données.
  - Amélioration de l'intégrité des données.
  - Simplification de la maintenance et des mises à jour des données.
- Formes normales couramment utilisées :
  - Première Forme Normale (1FN)
  - Deuxième Forme Normale (2FN)
  - Troisième Forme Normale (3FN)
  - Forme Normale de Boyce-Codd (FNBC)

Formes Normales et Avantages

- **1FN** : Chaque table a une clé primaire et les attributs contiennent des valeurs atomiques.
- **2FN** : Tous les attributs non-clés dépendent de la totalité de la clé primaire.
- **3FN** : Aucun attribut non-clé ne dépend d'un autre attribut non-clé.
- **FNBC** : Chaque déterminant est une clé candidate.
- Avantages de la normalisation :
  - Amélioration de l'intégrité des données.
  - Réduction de la redondance des données.
  - Simplification de la maintenance et des mises à jour des données.
- Exemple : Décomposition d'une table non normalisée en tables normalisées en 3FN.

Création de la base de données

- Pour créer une base de données SQLite, utilisez la commande suivante dans le terminal :

```
sqlite3 veterinaire.db
```

- Cela crée un fichier ‘veterinaire.db‘ qui contiendra votre base de données.
- Pour quitter SQLite, utilisez la commande :

```
.exit
```

Création des tables

- Table `animal` :

```
CREATE TABLE animal (
    id INTEGER PRIMARY KEY,
    nom TEXT NOT NULL,
    espece TEXT NOT NULL,
    age INTEGER
);
```

- Table `client` :

```
CREATE TABLE client (
    id INTEGER PRIMARY KEY,
    nom TEXT NOT NULL,
    prenom TEXT NOT NULL
);
```

Création des tables (suite)

- Table `maitre` :

```
        CREATE TABLE maitre (
            animal_id INTEGER,
            client_id INTEGER,
            FOREIGN KEY (animal_id)
    REFERENCES animal(id),
            FOREIGN KEY (client_id)
    REFERENCES client(id),
            PRIMARY KEY (animal_id, client_id
    )
        );
```

Insertion de données

- Insertion dans la table `animal` :

```
INSERT INTO animal (nom, espece, age)
VALUES ('Rex', 'Chien', 5);
INSERT INTO animal (nom, espece, age)
VALUES ('Miaou', 'Chat', 3);
INSERT INTO animal (nom, espece, age)
VALUES ('Panpan', 'Lapin', 2);
```

- Insertion dans la table `client` :

```
INSERT INTO client (nom, prenom)
VALUES ('Dupont', 'Jean');
INSERT INTO client (nom, prenom)
VALUES ('Martin', 'Claire');
```

Insertion de données dans la table `maitre`

- Insertion dans la table `maitre` :

```
INSERT INTO maitre (animal_id,
client_id)
VALUES (1, 1);
INSERT INTO maitre (animal_id,
client_id)
VALUES (2, 2);
INSERT INTO maitre (animal_id,
client_id)
VALUES (3, 2);
```

Commentaires sur l'insertion dans la table `maitre`

- **Génération automatique des clés** :
  - Les clés primaires des tables animal et client sont générées automatiquement, mais les clés étrangères dans maitre doivent correspondre à des valeurs existantes.
- **Vérification automatique** :
  - Le système de gestion de base de données (SGBD) vérifie automatiquement que les valeurs insérées dans les colonnes de clé étrangère correspondent à des clés primaires existantes dans les tables référencées.
- **Intégrité des données** :
  - Ces mécanismes aident à maintenir l'intégrité des données en empêchant les insertions qui violeraient les relations définies entre les tables.

Requêtes de base

- Sélectionner tous les animaux :

```
SELECT * FROM animal;
```

- Sélectionner tous les clients :

```
SELECT * FROM client;
```

Jointures

- Jointure (interne) entre les tables `animal`, `maitre`, et `client` :

```
SELECT animal.nom, client.nom, client.prenom
FROM animal
JOIN maitre ON animal.id = maitre.animal_id
JOIN client ON maitre.client_id = client.id;
```

Mise à jour des données

- Mettre à jour l'âge d'un animal :

```
        UPDATE animal SET age = 6 WHERE nom =
    'Rex';
```

Suppression de données - Base

- Supprimer un animal :

```
DELETE FROM animal WHERE nom = 'Rex';
```

- **Contraintes d'intégrité référentielle** :
  - La suppression peut échouer si l'animal est référencé dans une autre table (par exemple, ‘maitre‘).
  - Cela évite la création de références orphelines.

Suppression de données - Avancé

- **Option ON DELETE CASCADE** :
  - Permet de supprimer automatiquement les enregistrements référençant l'animal dans d'autres tables.
  - Exemple de définition :

```
CREATE TABLE maitre (
    animal_id INTEGER,
    client_id INTEGER,
    FOREIGN KEY (animal_id) REFERENCES
   animal(id) ON DELETE CASCADE,
    FOREIGN KEY (client_id) REFERENCES
   client(id),
    PRIMARY KEY (animal_id, client_id)
);
```

Requêtes avancées

- Sélectionner les animaux avec leur propriétaire :

```
SELECT animal.nom, animal.espece,
client.nom, client.prenom
FROM animal
JOIN maitre ON animal.id = maitre.
animal_id
JOIN client ON maitre.client_id =
client.id;
```

Requêtes avancées - Opérateurs logiques

- Sélectionner les chiens :

```
SELECT * FROM animal WHERE espece = 'Chien';
```

- Sélectionner les clients qui ont des chiens :

```
SELECT c.nom, c.prenom
FROM client c
JOIN maitre m ON c.id = m.client_id
JOIN animal a ON m.animal_id = a.id
WHERE a.espece = 'Chien';
```

- Sélectionner les animaux âgés de plus de 3 ans :

```
SELECT * FROM animal WHERE age > 3;
```

Requêtes avancées - Agrégats et Groupement

- Compter le nombre d'animaux par espèce :

```
SELECT espece, COUNT(*) as
nombre_animaux
FROM animal
GROUP BY espece;
```

- Trouver le client qui possède le plus d'animaux :

```
SELECT c.nom, c.prenom, COUNT(*) as
nombre_animaux
FROM client c
JOIN maitre m ON c.id = m.client_id
GROUP BY c.id
ORDER BY nombre_animaux DESC
LIMIT 1;
```

Requêtes avancées - Clients avec le plus d'animaux

- Lister tous les clients qui possèdent le plus grand nombre d'animaux :

```
WITH AnimalCounts AS (
    SELECT c.id, c.nom, c.prenom, COUNT(*) as
    nombre_animaux
    FROM client c
    JOIN maitre m ON c.id = m.client_id
    GROUP BY c.id, c.nom, c.prenom
)
SELECT nom, prenom, nombre_animaux
FROM AnimalCounts
WHERE nombre_animaux = (SELECT MAX(
   nombre_animaux) FROM AnimalCounts);
```

- Cette requête utilise une CTE (Common Table Expression) pour calculer le nombre d'animaux par client.
- Elle sélectionne ensuite les clients dont le nombre d'animaux est égal au maximum trouvé.

Algèbre relationnelle : Relations et schémas

(Edgar Frank Codd, "A Relational Model of Data for Large Shared Data Banks", 1970)

Définition (schéma relationnel)

Un **schéma de relation** est de la forme :

$$R(A_1, A_2, \ldots, A_n),$$

où R est le nom de la relation et A_i sont des **attributs**, chacun associé à un domaine (ensemble avec des contraintes sur les éléments) $Dom(A_i)$.

Définition (relation)

Une **relation** sur $R(A_1, \ldots, A_n)$ est un sous-ensemble du produit cartésien :

$$R \subseteq Dom(A_1) \times \cdots \times Dom(A_n).$$

Exemple :

$$client(id, nom, prenom), \quad animal(id, nom, espece, age).$$

Algèbre relationnelle : Tuples et opérations

- Un **tuple** est de la forme $(a_1, \ldots, a_n)$ avec $a_i \in Dom(A_i)$.
- En SQL, les relations sont représentées par des tables et les tuples sont des entrées (des lignes) de la table.
- Les opérateurs de l'algèbre relationnelle transforment des ensembles de tuples en d'autres ensembles de tuples.
- C'est une **algèbre close** : toute opération produit une nouvelle relation. Cela garantit que les résultats des opérations peuvent être réutilisés dans d'autres opérations.

Sélection (σ)

$$\sigma_C(R) = \{ t \in R \mid C(t) \}$$

- Filtre les tuples qui satisfont une condition C.
- Exemple : Sélectionner les chats :

```
SELECT * FROM animal WHERE espece = 'Chat';
```

Projection (π)

$$\pi_{A_1,\ldots,A_k}(R) = \{\, t[A_1, \ldots, A_k] \mid t \in R \,\}$$

- Conserve seulement certains attributs.
- Exemple : Noms et espèces des animaux.

```
SELECT nom, espece FROM animal;
```

Renommage (ρ)

$$\rho_{B \leftarrow A}(R)$$

- Permet de renommer un attribut ou une relation.
- Exemple : renommer `nom` en `nom_animal`.

```
SELECT nom AS nom_animal, espece FROM animal;
```

Union et Intersection

$$R \cup S = \{ t \mid t \in R \text{ ou } t \in S \}$$

$$R \cap S = \{ t \mid t \in R \text{ et } t \in S \}$$

Exemple SQL :

```
(SELECT * FROM animal WHERE age > 2)
UNION
(SELECT * FROM animal WHERE espece = 'Chien');
```

Différence et Produit cartésien

$$R - S = \{ t \mid t \in R \text{ et } t \notin S \}$$

$$R \times S = \{ (t, u) \mid t \in R, u \in S \}$$

Exemple : Animaux qui ne sont pas des chiens :

```
(SELECT * FROM animal)
EXCEPT
(SELECT * FROM animal WHERE espece = 'Chien');
```

Jointures

Définition

Soient $R(X, Y)$ et $S(Y, Z)$ deux relations avec attribut commun Y. La jointure naturelle est :

$$R \bowtie S = \{ t \cup u \mid t \in R,\ u \in S,\ t[Y] = u[Y] \}.$$

Exemple SQL avec NATURAL JOIN :

```
SELECT * FROM animal NATURAL JOIN maitre;
```

Jointure avec ON : la syntaxe utilisant ON permet de spécifier explicitement la condition de jointure (moins de risques d'erreurs)

```
SELECT * FROM animal JOIN maitre ON animal.id =
    maitre.animal_id;
```

Jointures comme Produit Cartésien et Sélection

Interprétation

La jointure naturelle peut être interprétée comme un produit cartésien suivi d'une sélection. Plus précisément, nous calculons d'abord le produit cartésien des relations R et S, puis nous sélectionnons les tuples qui satisfont la condition de jointure.

Produit Cartésien et Sélection

Soient $R(X, Y)$ et $S(Y, Z)$ deux relations. La jointure naturelle peut être exprimée comme :

$$R \bowtie S = \sigma_{R.Y=S.Y}(R \times S)$$

où :

- $R \times S$ est le produit cartésien de R et S.
- $\sigma_{R.Y=S.Y}$ est une sélection qui ne conserve que les tuples où les valeurs de l'attribut commun Y sont égales.

Exemple SQL

La jointure naturelle peut être exprimée en SQL en utilisant un produit cartésien suivi d'une sélection avec la clause WHERE :

```
SELECT * FROM animal, maitre
WHERE animal.id = maitre.animal_id;
```

Groupement

- Le groupement est une opération qui consiste à regrouper des éléments d'un ensemble selon un critère commun.
- Des fonctions d'agrégation peuvent être appliquées aux groupes. L'ordre des éléments ne doit pas affecter le résultat.
- Exemples de fonctions d'agrégation :
  - Somme (SUM)
  - Produit (PRODUCT)
  - Moyenne (AVG)
  - Maximum (MAX)
  - Minimum (MIN)
  - Comptage (COUNT)
- Une fonction d'aggrégation f doit être symétrique : pour toute permutation σ des arguments, on a :

$$f(x_1, x_2, \ldots, x_n) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$$

Groupement en SQL

- En SQL, le groupement se fait avec la clause `GROUP BY`.
- Exemple : Compter le nombre d'animaux par espèce (fonction COUNT).

```
SELECT espece, COUNT(*) as nombre_animaux
FROM animal
GROUP BY espece;
```

- Exemple : Calculer l'âge moyen des animaux par espèce (fonction AVG).

```
SELECT espece, AVG(age) as age_moyen
FROM animal
GROUP BY espece;
```

Groupement en SQL (suite)

- Pour filtrer les groupes, on utilise la clause `HAVING`.
- Exemple : Sélectionner les espèces ayant plus d'un animal.

```
SELECT espece, COUNT(*) as nombre_animaux
FROM animal
GROUP BY espece
HAVING COUNT(*) > 1;
```

- La clause `HAVING` est similaire à `WHERE`, mais elle s'applique aux groupes plutôt qu'aux lignes individuelles.

Lois de l'algèbre relationnelle

- **Commutativité :**

$$R \cup S = S \cup R, \quad R \cap S = S \cap R, \quad R \bowtie S = S \bowtie R$$

- **Associativité :**

$$(R \cup S) \cup T = R \cup (S \cup T), \quad (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

- **Distributivité :**

$$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$$

Optimisation des requêtes

- Les lois permettent de transformer les requêtes pour réduire le coût.
- Exemple : **pousser les sélections au plus bas**.

$$\sigma_C(R \bowtie S) = (\sigma_C(R)) \bowtie S \quad \text{si } C \text{ ne dépend que de } R.$$

- Cela évite de calculer un gros produit cartésien inutile.

Lien avec SQL

- Chaque opération de l'algèbre relationnelle correspond à une clause SQL :
  - $\sigma \rightarrow$ `WHERE`
  - $\pi \rightarrow$ `SELECT`
  - $\cup, \cap, - \rightarrow$ `UNION, INTERSECT, EXCEPT`
  - $\bowtie \rightarrow$ `JOIN`
  - γ (groupement) $\rightarrow$ `GROUP BY`

Révisions

https://www.youtube.com/watch?v=ZIkDkxeFpo4

Introduction à Python Pandas

- Pandas est une bibliothèque Python pour la manipulation et l'analyse de données tabulaires.
- Un **DataFrame** est une structure bidimensionnelle (lignes et colonnes), similaire à une table SQL.
- Cas d'usage : exploration de données, prétraitement pour l'analyse ou l'IA, complément à SQL.
- Installation et importation :

```
import pandas as pd
```

Charger une base de données dans Pandas

- Charger les tables de `veterinaire.db` dans des DataFrames :

```
import sqlite3
import pandas as pd
# Connexion a la base SQLite
conn = sqlite3.connect('veterinaire.db')
# Charger les tables
animals = pd.read_sql_query("SELECT * FROM
    animal;", conn)
clients = pd.read_sql_query("SELECT * FROM
    client;", conn)
masters = pd.read_sql_query("SELECT * FROM
    maitre;", conn)
```

- Exemple de résultat pour `animals` :

id	nom	espece	age
0	Rex	Chien	5
1	Miaou	Chat	3
2	Panpan	Lapin	2

Sélections et filtrage avec Pandas

- Filtrer les chiens (équivalant `WHERE` en SQL) :

```
chiens = animals[animals['espece'] == 'Chien'
    ]
```

- Sélectionner des colonnes (équivalant `SELECT` en SQL) :

```
noms_especes = animals[['nom', 'espece']]
```

- Résultat pour `chiens` :

id	nom	espece	age
0	Rex	Chien	5

Jointures avec Pandas

- Équivalent SQL :

```sql
SELECT animal.nom, client.nom, client.prenom
FROM animal
JOIN maitre ON animal.id = maitre.animal_id
JOIN client ON maitre.client_id = client.id;
```

- En Pandas :

```python
result = animals.merge(masters, left_on='id',
    right_on='animal_id')\
                .merge(clients, left_on='
   client_id', right_on='id')\
                [['nom_x', 'nom_y', 'prenom'
   ]]
result.columns = ['nom_animal', 'nom_client',
    'prenom']
```

- Résultat :

nom_animal	nom_client	prenom
Rex	Dupont	Jean

Agrégations avec Pandas

- Compter les animaux par espèce (équivalant `GROUP BY`) :

```
compte_par_espece = animals.groupby('espece')
    .size().reset_index(name='nombre_animaux')
```

- Résultat :

espece	nombre_animaux
Chien	1
Chat	1
Lapin	1

- Nombre d'animaux par client :

```
animaux_par_client = masters.merge(clients,
    left_on='client_id', right_on='id')\
                                .groupby(['nom', '
    prenom'])\
                                .size().
    reset_index(name='nombre_animaux')
```

TP 1 : SQL de base

Objectifs

- Installation et configuration d'un environnement SQL (par exemple, SQLite, MySQL, PostgreSQL, etc.).
- Exercices pratiques sur la création de tables, l'insertion de données et l'exécution de requêtes SQL.

Module 2 : Bases de données avancées

- Introduction aux bases de données NoSQL.
- Présentation des bases de données de graphes.
- Concepts avancés de gestion des transactions.

Module 3 : Manipulation de données dans le cloud

- Introduction au cloud computing et aux bases de données dans le cloud.
- Présentation des outils de manipulation de données dans le cloud (par exemple, Google BigQuery, Amazon Redshift).
- Introduction à PySpark pour le traitement de données distribuées.

TP 2 : Manipulation de données dans le cloud

Objectifs

- Configuration d'un environnement cloud (par exemple, Google Cloud Platform, AWS).
- Exercices pratiques sur la manipulation de données avec PySpark et autres outils cloud.

Module 4 : Retrieval-Augmented Generation (RAG)

- Introduction aux modèles de langage de grande taille (LLM).
- Présentation de la technique RAG et ses applications.
- Intégration de RAG dans des applications de traitement du langage naturel.

TP 3 : Utilisation de RAG

Objectifs

- Installation et configuration des bibliothèques nécessaires pour RAG.
- Exercices pratiques sur l'utilisation de RAG pour la génération de texte et la récupération d'informations.