S2J1-2 : SQL (PostgreSQL ou MySQL).

Objectif :

- Comprendre les concepts fondamentaux des bases de données relationnelles.
- Apprendre à écrire des requêtes SQL pour interagir avec les données.
- Manipuler, explorer et extraire des informations utiles à partir de tables.

---

1. Introduction aux bases de données relationnelles.

Un Data Engineer travaille souvent avec des bases de données relationnelles comme PostgreSQL, MySQL ou SQLite. Voici les concepts clés :

- **Concepts fondamentaux** :
  - **Table** : Une structure tabulaire composée de colonnes et de lignes.
  - **Clé primaire** : Une colonne (ou combinaison de colonnes) qui identifie de manière unique une ligne.
  - **Clé étrangère** : Une colonne qui établit une relation avec une autre table.
  - **Relations** : Les bases relationnelles organisent les données dans des tables liées entre elles (relations).
- **Schéma relationnel** :

  Comprendre comment plusieurs tables s'articulent dans une base grâce aux clés primaires et étrangères.
- **Installation et outils pratiques** :
  - Installer une base locale comme SQLite ou PostgreSQL.
  - Utiliser des outils comme **DBeaver**, **pgAdmin**, ou des notebooks avec SQLAlchemy pour exécuter des requêtes.

---

2. Langage SQL : Concepts de base.

a. Création et exploration d’une table.

- Créer une table :

```
CREATE TABLE ventes ( id INT PRIMARY KEY, produit VARCHAR(100), quanti
té INT, prix DECIMAL(10, 2), date_vente DATE );
```

- Insérer des données :

```
INSERT INTO ventes (id, produit, quantité, prix, date_vente) VALUES
(1, 'Ordinateur', 3, 499.99, '2023-12-01');
```

- Explorer les données :

```
SELECT * FROM ventes;
```

---

b. Lecture et sélection des données.

- Sélections simples :

```
SELECT produit, prix FROM ventes;
```

- **Filtres** avec WHERE :

```
SELECT * FROM ventes WHERE quantité > 2;
```

- Opérateurs logiques :

```
SELECT * FROM ventes WHERE quantité > 2 AND prix < 500;
```

- Tri des résultats :

```
SELECT * FROM ventes ORDER BY prix DESC;
```

---

c. Agrégations et calculs.

- **Fonctions d'agrégation** :

```
SELECT COUNT(*) AS nombre_de_ventes, SUM(prix * quantité) AS revenu_to
tal FROM ventes;
```

- **Regroupement des données** :

```
SELECT produit, SUM(quantité) AS total_quantité FROM ventes GROUP BY p
roduit;
```

---

3. Relations entre tables.

a. Créer plusieurs tables reliées.

- Table des clients :

```
CREATE TABLE clients ( client_id INT PRIMARY KEY, nom VARCHAR(100), em
ail VARCHAR(100) );
```

- Table des commandes :

```
CREATE TABLE commandes ( commande_id INT PRIMARY KEY, client_id INT, p
roduit VARCHAR(100), quantité INT, FOREIGN KEY (client_id) REFERENCES
clients(client_id) );
```

---

b. Requêtes avec jointures.

Les jointures sont essentielles pour combiner des données provenant de plusieurs tables.

- **Jointure interne (INNER JOIN)** :

```
SELECT c.nom, cmd.produit, cmd.quantité FROM clients c INNER JOIN comm
andes cmd ON c.client_id = cmd.client_id;
```

- **Jointure gauche (LEFT JOIN)** :

  Pour inclure toutes les lignes de la table principale, même si elles n'ont pas de correspondance.

```
SELECT c.nom, cmd.produit FROM clients c LEFT JOIN commandes cmd ON c.
client_id = cmd.client_id;
```

---

4. Optimisation des requêtes

- **Indexation des colonnes** :

  Les index permettent de rendre les requêtes plus rapides, surtout pour les tables volumineuses.

```
CREATE INDEX idx_produit ON ventes(produit);
```

- **Utilisation des vues** :

  Une vue est une requête sauvegardée sous forme d'objet dans la base de données.

```
CREATE VIEW ventes_annuelles AS SELECT produit, SUM(quantité) AS total
_vendu FROM ventes GROUP BY produit;
```

Exercice pratique (Mini-projet)

1. **Créer une base de données relationnelle simple :**
    - Une table pour des clients.
    - Une table pour des produits.
    - Une table pour des commandes, avec des clés étrangères vers les deux premières.
2. **Insérer des données** dans ces tables.
3. **Réaliser les requêtes suivantes :**
    - Trouver les produits achetés par un client spécifique.
    - Calculer le revenu total généré par produit.
    - Lister tous les clients qui n'ont pas passé de commandes (LEFT JOIN).
4. **Exporter vos résultats en CSV** si votre outil le permet.

Pourquoi c'est important pour un Data Engineer ?

- **Gestion des données relationnelles** : SQL est souvent utilisé pour interagir avec les entrepôts de données relationnels (Redshift, BigQuery, etc.).
- **Construction de pipelines de données** : Vous écrirez des requêtes pour extraire et transformer des données avant de les charger dans un autre système.
- **Performance et scalabilité** : Comprendre comment optimiser les requêtes est