Chapitre 4 — Initialisation & synchronisation (JSON → SQLite)

---

1. C'est quoi ?

Ce chapitre traite de la **création automatique** de la base et de la **synchronisation** entre fichiers JSON (locaux) et base SQLite.

---

2. Pourquoi on l'utilise ?

- Pour **initialiser** automatiquement la base (tables, fichiers)
- Pour **importer / migrer** des données existantes (format JSON)
- Pour assurer une **cohérence entre sources** (pas de doublon, maj correcte)

---

3. Fichiers concernés

Fichier	Rôle
`init_db.py`	vérifie si la base existe, sinon la crée avec les bonnes tables
`sqlLib.py`	contient les fonctions métiers pour la DB : lecture, écriture, update
`.json` locaux	fichiers à synchroniser avec la base (par script ou interface)

---

4. Fonctions clés dans sqlLib.py

Fonction	Rôle
`isTableExists(tableName)`	Vérifie si une table existe déjà dans la base
`getRelatedSQLInfo(tableName)`	Retourne les infos PRAGMA de la table (colonnes, types…)
`getPrimaryColumn(tableName)`	Donne la colonne PK (clé primaire)
`getAllRows(tableName)`	Récupère toutes les lignes d'une table
`isEntryExists(tableName, key)`	Vérifie si une entrée existe déjà
`createLine(tableName, dict)`	Insère une ligne en dict dans la table
`updateLine(tableName, dict)`	Met à jour une ligne si elle existe
`deleteLine(tableName, id)`	Supprime une entrée
`updateDb(tableName, data)`	Met à jour plusieurs lignes (appelé par syncDB)
`syncDB(jsonPath)`	Lis un .json local, met à jour la base correspondante

---

5. Exemple d’usage (dans un script)

```
from sqlLib import syncDB

# Met à jour les tables à partir des fichiers JSON locaux
syncDB("./data/machines.json")
syncDB("./data/logs.json")
```

✅Résultat :

Tu as un mécanisme de **mise à jour automatique** entre fichiers JSON et base SQLite, réutilisable dans ton app ou ton API.