Insertion des données (initial load)

Dans cet exercice, tu vas apprendre à charger des données JSON dans une base MySQL : c'est ce qu'on appelle un **initial load**. Concrètement, nous partirons d'un fichier films.json contenant une liste de films (titre, année, acteurs, genres, images, etc.) et nous allons écrire un script Python qui crée les tables nécessaires (**migrations** simples), puis importe les données de manière **idempotente** : c'est-à-dire qu'on pourra **relancer le script plusieurs fois sans créer de doublons**, tout en mettant à jour les infos si elles changent. Cet exercice te permettra de comprendre les étapes clés d'un **pipeline d'import** : créer un schéma relationnel, lire du JSON, insérer avec des contraintes, gérer les relations entre tables et vérifier le résultat avec des requêtes SQL.

6TTR

Vocabulaire

📘 Migration

En informatique (et en particulier avec les bases de données), une **migration** est une opération qui **modifie la structure de la base** afin de l'adapter aux besoins de l'application : **création** de tables, **ajout** de colonnes, ajout de **clés**, modification de types, etc.

📘 Initial Load

Un **initial load** est la toute **première opération de chargement de données** depuis une source (JSON, CSV, API…) vers une base de données ou un système cible. L'objectif est de peupler la base à partir de zéro avec un jeu complet et propre, qui servira ensuite de référence pour les mises à jour ou les ajouts incrémentaux.

📘 Idempotence

Dans ce contexte, l'**idempotence** veut dire que ton script d'import peut être relancé plusieurs fois sans créer de doublons ni abîmer la base.
Exemple concret :

- Si un film est déjà importé (même `external_id`), il ne doit pas être inséré une deuxième fois → il est simplement **mis à jour** si ses infos ont changé.
- Si un acteur ou un genre existe déjà, on le **réutilise** au lieu de créer une nouvelle ligne identique.
- Si une relation film-acteur existe déjà, on ne la recrée pas (ou bien on met à jour le rôle si nécessaire).

👉 Résultat : que tu lances l'import **1 fois ou 10 fois**, la base contient **le même état cohérent**. C'est ça, l'idempotence.

Objectif : écrire **un petit script Python** qui :

1. **crée** les tables MySQL (migrations simples),
2. **importe** le fichier `films.json`,
3. garantit l'**idempotence** (relancer le script ne duplique pas),
4. reste **compréhensible** (premier exercice).

Créé un schéma entités-associations d'abord, ainsi que la conversion en version "physique" pour MySql (tables, tables de jointure, PK's, FK's...).

Prérequis

- MySQL en local (ou Docker).
- Un utilisateur MySQL avec droits de création : `user/password`.
- Paquet Python :

```
pip install mysql-connector-python
```

- Fichier JSON : `data/films.json` (ton tableau de 25 films).

Schéma minimal (normalisé mais simple)

Nous allons créer 6 tables, mais de manière **guidée** :

- `film(id PK AI, external_id UNIQUE, titre, annee, realisateur, duree_min, resume)`
- `genre(id PK AI, nom UNIQUE)`
- `film_genre(film_id FK, genre_id FK, PK(film_id, genre_id))`

- `acteur(id PK AI, nom UNIQUE)`
- `film_acteur(film_id FK, acteur_id FK, role, PK(film_id, acteur_id))`
- `image(id PK AI, film_id FK, path)`

IDEMPOTENCE :

- `film.external_id` = l'`id` du JSON (UNIQUE) → on peut faire un *upsert*.
- `genre.nom` et `acteur.nom` sont **UNIQUE** → pas de doublons.

Étape 1 — Script "migrations simples"

But : **créer si n'existe pas** (safe à relancer).

```
# migrate.py
import mysql.connector as mysql

DB = dict(host="127.0.0.1", user="root", password="password", database="cinema", auth_plugin='

DDL = [
    # Charset/Collation pour bien gérer les accents
    "CREATE DATABASE IF NOT EXISTS cinema CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci;",
    "USE cinema;",
"""
    CREATE TABLE IF NOT EXISTS film (
      id INT PRIMARY KEY AUTO_INCREMENT,
      external_id INT NOT NULL,
      titre VARCHAR(255) NOT NULL,
      annee INT NOT NULL,
      realisateur VARCHAR(255),
      duree_min INT,
      resume TEXT,
      UNIQUE KEY uq_film_external_id (external_id),
      INDEX idx_film_titre (titre),
      INDEX idx_film_annee (annee)
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
""",
"""
    CREATE TABLE IF NOT EXISTS genre (
      id INT PRIMARY KEY AUTO_INCREMENT,
      nom VARCHAR(100) NOT NULL,
      UNIQUE KEY uq_genre_nom (nom)
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
""",
"""
    CREATE TABLE IF NOT EXISTS film_genre (
      film_id INT NOT NULL,
```

```
        genre_id INT NOT NULL,
        PRIMARY KEY (film_id, genre_id),
        FOREIGN KEY (film_id) REFERENCES film(id) ON DELETE CASCADE,
        FOREIGN KEY (genre_id) REFERENCES genre(id) ON DELETE CASCADE
      ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
    """,
    """
      CREATE TABLE IF NOT EXISTS acteur (
        id INT PRIMARY KEY AUTO_INCREMENT,
        nom VARCHAR(255) NOT NULL,
        UNIQUE KEY uq_acteur_nom (nom)
      ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
    """,
    """
      CREATE TABLE IF NOT EXISTS film_acteur (
        film_id INT NOT NULL,
        acteur_id INT NOT NULL,
        role VARCHAR(255),
        PRIMARY KEY (film_id, acteur_id),
        FOREIGN KEY (film_id) REFERENCES film(id) ON DELETE CASCADE,
        FOREIGN KEY (acteur_id) REFERENCES acteur(id) ON DELETE CASCADE
      ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
    """,
    """
      CREATE TABLE IF NOT EXISTS image (
        id INT PRIMARY KEY AUTO_INCREMENT,
        film_id INT NOT NULL,
        path VARCHAR(400) NOT NULL,
        FOREIGN KEY (film_id) REFERENCES film(id) ON DELETE CASCADE,
        INDEX idx_image_film (film_id)
      ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
    """
  ]

  def main():
      conn = mysql.connect(**DB)
      cur = conn.cursor()
      for stmt in DDL:
          cur.execute(stmt)
      conn.commit()
      cur.close()
      conn.close()
      print("✅ Migrations OK (tables prêtes).")

  if __name__ == "__main__":
      main()
```

À comprendre

- `CREATE TABLE IF NOT EXISTS` → relancer sans casser.
- Clés **UNIQUE** + **FK** → cohérence + idempotence.

Étape 2 — Import (lecture JSON → insert idempotent)

Stratégie **simple** :

- Upsert du film via `external_id UNIQUE` (**sans dupliquer**).
- `get_or_create` pour `genre` et `acteur` (sur leur `nom UNIQUE`).
- Récréer les liaisons (film-genre, film-acteur) **sans dupliquer**.

```python
# import_films.py
import json
import mysql.connector as mysql

DB = dict(host="127.0.0.1", user="root", password="password", database="cinema", auth_plugin='

def get_conn():
    return mysql.connect(**DB)

def get_or_create_genre(cur, nom):
    # essaie d'insérer, sinon récupère l'id existant
    cur.execute("INSERT IGNORE INTO genre(nom) VALUES (%s)", (nom,))
    cur.execute("SELECT id FROM genre WHERE nom=%s", (nom,))
    return cur.fetchone()[0]

def get_or_create_acteur(cur, nom):
    cur.execute("INSERT IGNORE INTO acteur(nom) VALUES (%s)", (nom,))
    cur.execute("SELECT id FROM acteur WHERE nom=%s", (nom,))
    return cur.fetchone()[0]

def upsert_film(cur, f):
"""
    Retourne l'id interne du film après upsert.
    ON DUPLICATE KEY UPDATE garantit l'idempotence.
"""
cur.execute("""
          INSERT INTO film (external_id, titre, annee, realisateur, duree_min, resume)
          VALUES (%s, %s, %s, %s, %s, %s)
          ON DUPLICATE KEY UPDATE
            titre=VALUES(titre),
            annee=VALUES(annee),
            realisateur=VALUES(realisateur),
            duree_min=VALUES(duree_min),
            resume=VALUES(resume)
""", (f["id"], f["titre"], f["annee"], f.get("realisateur"), f.get("duree_min"), f.get("resume")
    # Récupérer l'id interne
    cur.execute("SELECT id FROM film WHERE external_id=%s", (f["id"],))
    return cur.fetchone()[0]

def link_film_genre(cur, film_id, genre_id):
    # évite les doublons grâce à la PK(film_id, genre_id)
cur.execute("""
```

```
        INSERT IGNORE INTO film_genre (film_id, genre_id) VALUES (%s, %s)
""", (film_id, genre_id))

def link_film_acteur(cur, film_id, acteur_id, role):
cur.execute("""
        INSERT INTO film_acteur (film_id, acteur_id, role)
        VALUES (%s, %s, %s)
        ON DUPLICATE KEY UPDATE role=VALUES(role)
""", (film_id, acteur_id, role))

def insert_images(cur, film_id, images):
    # simple: on ajoute; si tu veux l'idempotence stricte, vérifie l'existence du path avant
    for p in images:
        cur.execute("INSERT INTO image (film_id, path) VALUES (%s, %s)", (film_id, p))

def import_json(path_json):
    with open(path_json, encoding="utf-8") as f:
        films = json.load(f)

    conn = get_conn()
    cur = conn.cursor()
    try:
        # petite transaction globale (ici dataset modeste)
        conn.start_transaction()
        for f in films:
            # 1) upsert film
            film_id = upsert_film(cur, f)

            # 2) genres
            for g in f.get("genres", []):
                gid = get_or_create_genre(cur, g.strip())
                link_film_genre(cur, film_id, gid)

            # 3) acteurs
            for a in f.get("acteurs", []):
                aid = get_or_create_acteur(cur, a["nom"].strip())
                role = a.get("role")
                link_film_acteur(cur, film_id, aid, role)

            # 4) images (option simple : on ne déduplique pas ici)
            insert_images(cur, film_id, f.get("images", []))

        conn.commit()
        print(f"✅ Import OK ({len(films)} films).")
    except Exception as e:
        conn.rollback()
        print("❌ Erreur, rollback :", e)
    finally:
        cur.close()
        conn.close()

if __name__ == "__main__":
    import_json("data/films.json")
```

Points pédagogiques clés

- `INSERT IGNORE` + `UNIQUE` → **get_or_create** sans si/else compliqué.
- `ON DUPLICATE KEY UPDATE` → **upsert** propre.
- **Transaction** : si ça casse, **rollback** (rien d'à moitié importé).
- **Idempotence** : relance le script → pas de doublons, les films sont **mis à jour**.

Étape 3 — Vérifications (exemples concrets)

A) Compter les films importés

```sql
SELECT COUNT(*) FROM film;
```

B) Vérifier un film précis

```sql
SELECT id, external_id, titre, annee FROM film WHERE external_id=101;
```

C) Genres liés à “Inception”

```sql
SELECT g.nom
FROM film f
JOIN film_genre fg ON fg.film_id=f.id
JOIN genre g ON g.id=fg.genre_id
WHERE f.external_id=101;
```

D) Acteurs + rôle

```sql
SELECT a.nom, fa.role
FROM film f
JOIN film_acteur fa ON fa.film_id=f.id
JOIN acteur a ON a.id=fa.acteur_id
WHERE f.external_id=101;
```

E) Images liées

```sql
SELECT path FROM image i
JOIN film f ON f.id=i.film_id
WHERE f.external_id=101;
```

Bonnes pratiques (version “élève”)

- **Toujours** mettre `utf8mb4` pour gérer accents/emoji.
- Normaliser strings : `strip()`, éviter espaces doubles.
- **Ne jamais** mettre d’URL en dur côté Flask : stocke juste le **path** d’image.
- **Commence petit** : teste d’abord avec 3 films, puis 25.
- Si tu veux que les images soient aussi **idempotentes**, remplace `insert_images` par :

```
cur.execute("SELECT 1 FROM image WHERE film_id=%s AND path=%s", (film_id, p))
if not cur.fetchone():
    cur.execute("INSERT INTO image (film_id, path) VALUES (%s, %s)", (film_id, p))
```

Variante “ultra-simplifiée” (si nécessaire)

> Pour une première heure de cours : **2 TABLES** seulement.

- `film(id PK AI, external_id UNIQUE, titre, annee, resume)`
- `image(id PK AI, film_id FK, path)`

Idées : ignorer dans un premier temps `genre`, `acteur`, puis **ajouter** ces tables à l’exercice 2 (migration qui étend le schéma, et script d’import qui les remplit).

Ce que tu dois réussir à la fin

- Lancer `python migrate.py` → “✅ Migrations OK”.
- Lancer `python import_films.py` → “✅ Import OK (25 films)”.
- Relancer `python import_films.py` → **aucun doublon**, mais les valeurs mises à jour si tu modifies le JSON.
- Vérifier 2–3 films avec les requêtes SQL de contrôle.

Prochaines étapes (facultatif)

- Ajouter un **mode** `--dry-run` qui affiche les actions sans écrire.
- Écrire des **tests** (pytest) pour `get_or_create_*`.
- Ajouter des **indexes** selon les recherches que tu feras dans Flask (`WHERE annee=…`, `LIKE titre`).

Si tu veux, je te fournis une **version monofichier** qui fait *migrations + import* en un seul `python app.py` (avec un petit menu texte).