

**PAI(**LAB**)**

­

**Submitted To:**

**Sir Rasikh**

**Submitted By:**

**Muhammad Mehdi**

**Roll No:**

**SU92-BSAIM-F23-004**

**Task 10**

**Step 1: Creating the Vector Database Schema**

We used Weaviate to create a cloud-hosted vector database. We started by writing a Python script (create\_vector\_db.py) that connects to the Weaviate Cloud Service using API keys, which are loaded from environment variables for security.

Once connected, we created three main collections (or classes) in the database:

* **Movie**: Stores the main information about each movie, like the title, description, director, year, rating, and an ID.
* **Review**: Stores the critic reviews for each movie.
* **Synopsis**: Contains the brief summary or plot of the movie.

Each of these collections uses a HuggingFace transformer model (all-MiniLM-L6-v2) for vectorizing text fields like the title, description, and reviews. The only exception is the director’s name, which we skipped from vectorization because it's just a name and doesn't need semantic understanding.

We also set up relationships between these collections:

* Each **Movie** can have multiple **Reviews** and **a Synopsis**.
* The **Synopsis** also holds a reference back to the movie it belongs to.

By the end of this script, our database structure was fully prepared.

**Step 2: Importing and Linking Data**

In the second part (import\_data\_vector\_db.py), we imported actual movie data from a CSV file (movies\_data.csv). This file included movie titles, descriptions, IDs, years, ratings, synopses, directors, and three separate critic reviews per movie.

Here’s how we handled the data:

1. **Critic Reviews**: For every non-empty review found in the CSV file, we created a unique UUID using its content (so the same text would always get the same ID) and inserted it into the Review collection.
2. **Movies**: We collected all main movie details, then added references linking each movie to its related reviews using their UUIDs.
3. **Synopses**: Each synopsis was inserted and linked back to its respective movie via a UUID. We also made sure that each movie had a hasSynopsis reference to connect it properly.

After inserting all the data and relationships, we closed the connection to the Weaviate client.

**Conclusion**

This assignment helped us understand how to work with vector databases in a real-world scenario. We learned to:

* Set up a semantic database with proper structure.
* Connect Python code to a hosted database using API keys.
* Handle relationships and references between different types of data.
* Insert complex structured data (like movies with multiple reviews and a synopsis) and ensure everything links properly.

Overall, this project gave us hands-on experience in building intelligent search-ready databases that can be used in applications like recommendation systems, semantic search, or AI-based querying.