Tutorial: Integrating Free Generative AI APIs into a DBMS Project

1. Motivation

Modern database systems often need to go beyond simple data storage and retrieval. Generative AI can be used to:

- Automatically generate insights or summaries from stored data.
- Provide natural language interfaces for querying databases.
- Create intelligent recommendation systems.

In this tutorial, we will learn how to integrate a **free generative AI API** (such as OpenAI's free-tier API or Hugging Face Inference API) into a DBMS-based project.

—

2. Project Scenario

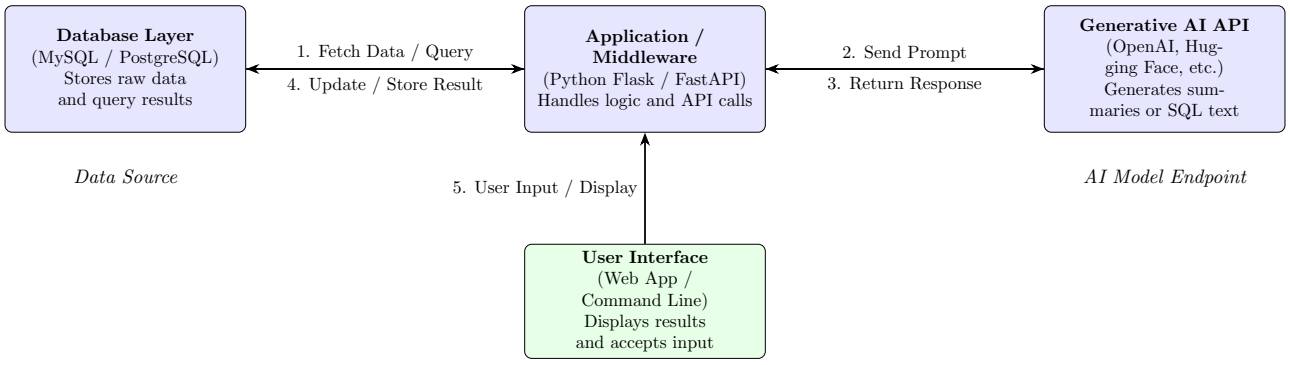
Imagine you are developing a **Movie Review Management System**. The database stores user reviews, ratings, and movie details.

We want to:

- Summarize long user reviews automatically using a generative AI model.
- Generate a sentiment label (*Positive*, *Negative*, *Neutral*) for each review.

—

3. System Architecture



—

4. Database Schema

A simple SQL table design:

Listing 1: movie_reviews table

```
CREATE TABLE movie_reviews (
    review_id INT PRIMARY KEY AUTO_INCREMENT,
    movie_title VARCHAR(255),
    review_text TEXT,
    user_name VARCHAR(100),
    sentiment VARCHAR(20),
    summary TEXT
);
```

—

5. Connecting to the Database

You can use `mysql.connector` or `psycopg2` in Python to connect to the database.

Listing 2: Connect to the database

```
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password",
    database="movies"
)
cursor = conn.cursor()
```

—

6. Making a Free AI API Call

For demonstration, we use the **Hugging Face Inference API** (no key required for some models).

Listing 3: Calling a Hugging Face API

```
import requests, json

API_URL = "https://api-inference.huggingface.co/models/facebook/bart-large-
    ↪ cnn"
headers = {"Authorization": "Bearer␣YOUR_TOKEN"} # Optional if using a free
    ↪ model

def summarize(text):
    payload = {"inputs": text}
    response = requests.post(API_URL, headers=headers, json=payload)
    result = response.json()
    return result[0]['summary_text']
```

—

7. Integrating with the Database

Listing 4: Fetch review, generate summary, and update DB

```
cursor.execute("SELECT␣review_id,␣review_text␣FROM␣movie_reviews␣WHERE␣
    ↪ summary␣IS␣NULL")
rows = cursor.fetchall()
```

```
for review_id, text in rows:
    summary = summarize(text)
    cursor.execute(
        "UPDATE movie_reviews SET summary=%s WHERE review_id=%s",
        (summary, review_id)
    )
conn.commit()
```

—

8. Hints for Extensions

Encourage students to:

- Implement sentiment classification using another free model.
- Add a column to store the AI confidence score.
- Develop a small web dashboard (Flask/Streamlit) to visualize reviews and summaries.

—

9. Ethical Discussion

- Always mention AI-generated content clearly.
- Avoid sending private or sensitive data to external APIs.
- Implement rate-limiting and caching to respect API usage.

—

10. Summary

This exercise demonstrates:

- How to extend DBMS projects with intelligent AI features.
- How to handle external API calls safely and integrate them into workflows.
- The real-world value of connecting structured databases with generative models.

—

11. Challenge for Students

1. Add a second AI feature: automatic keyword extraction for each review.
2. Allow users to search reviews using natural language prompts (AI-assisted SQL generation).

—