

YOU

Tengo en mi trabajo varias bases de datos, cada una con más de 500 tablas. Quiero llevar adelante un proyecto de text to sql para que usuarios de todas las areas del negocio puedan interactuar con la base de datos.

Quisiera usar langchain con python, donde basicamente quiero pasarle el prompt optimizado del usuario, junto con toda la informacion de las bases de datos (tablas, columnas, pk, fk, joins, ejemplos sql, etc.). Le paso la consulta al motor de base de datos y luego el llm regresa la respuesta al usuario.

Quisiera empezar con modelos llm que no requieran credenciales, que sea libres.

Quiero que tomes el rol de un experto en IA y me digas el paso a paso con ejemplos claros. Luego el plan de proyecto y todo el codigo para implementarlo. Quiero que razones en cada paso.

BLACKBOX.AI

Perfecto, vamos a abordar tu proyecto de **Text-to-SQL** con LangChain y Python, usando modelos LLM libres (open source) que no requieran credenciales. Te guiaré paso a paso, razonando cada decisión, y al final te daré un plan de proyecto y código de ejemplo.

1. Entendiendo el problema y el contexto

- Tienes **varias bases de datos** con más de 500 tablas cada una.
- Quieres que usuarios no técnicos puedan hacer consultas en lenguaje natural y obtener respuestas.
- Quieres usar **LangChain** para orquestar el flujo.
- Quieres usar modelos LLM **libres, sin credenciales** (ej: modelos locales o de la comunidad).
- Quieres pasar al LLM la información estructural de las bases (tablas, columnas, PK, FK, joins, ejemplos SQL).
- El LLM genera la consulta SQL, la ejecutas en la base, y luego el LLM interpreta el resultado para el usuario.

2. Desafíos principales

- **Escalabilidad y contexto:** 500 tablas es mucha información para un prompt. No puedes pasar toda la metadata en cada prompt.
- **Modelos LLM libres:** Los modelos grandes y buenos suelen requerir hardware potente o servicios cloud. Hay opciones open source pero con limitaciones.
- **Parsing y ejecución:** Debes validar y ejecutar el SQL generado para evitar errores o inyecciones.
- **Interfaz amigable:** El usuario debe interactuar con un chat o interfaz sencilla.

3. Estrategia general

Paso 1: Extraer y organizar metadata de las bases

- Extraer esquema: tablas, columnas, tipos, PK, FK, relaciones.
- Generar un **grafo de relaciones** para facilitar joins.
- Guardar ejemplos de consultas SQL frecuentes (si tienes).

Paso 2: Indexar la metadata para búsqueda

- No puedes pasar toda la metadata al LLM en cada prompt.
- Usa un motor de búsqueda vectorial o simple para recuperar solo la metadata relevante según la consulta del usuario.
- Ejemplo: si el usuario pregunta por "ventas", recupera tablas y columnas relacionadas con ventas.

Paso 3: Usar LangChain para orquestar

- Paso 3.1: Recibir consulta en lenguaje natural.
- Paso 3.2: Buscar metadata relevante.
- Paso 3.3: Construir prompt con metadata relevante + consulta.
- Paso 3.4: LLM genera SQL.
- Paso 3.5: Ejecutar SQL en la base.
- Paso 3.6: LLM interpreta resultado y genera respuesta en NL.

Paso 4: Elegir modelo LLM libre

- Opciones populares:
 - **GPT4All** (local, open source, modelos LLaMA derivados)
 - **Llama.cpp** (ejecuta modelos LLaMA en CPU)
 - **HuggingFace transformers** con modelos como **GPT-J**, **GPT-NeoX**, **Falcon**, etc.
- Para prototipo, GPT4All o Llama.cpp son buenas opciones.

4. Paso a paso con ejemplos

Paso 1: Extraer metadata

Supongamos que usas PostgreSQL. Puedes obtener metadata con consultas:

```
-- Tablas
SELECT table_name FROM information_schema.tables WHERE table_schema='public';

-- Columnas
SELECT table_name, column_name, data_type FROM information_schema.columns WHERE table_schema='public';

-- PK
SELECT tc.table_name, kcu.column_name
FROM information_schema.table_constraints tc
JOIN information_schema.key_column_usage kcu
  ON tc.constraint_name = kcu.constraint_name
WHERE tc.constraint_type = 'PRIMARY KEY' AND tc.table_schema='public';

-- FK
SELECT tc.table_name, kcu.column_name, ccu.table_name AS foreign_table, ccu.column_name AS foreign_column
FROM information_schema.table_constraints AS tc
JOIN information_schema.key_column_usage AS kcu
  ON tc.constraint_name = kcu.constraint_name
JOIN information_schema.constraint_column_usage AS ccu
  ON ccu.constraint_name = tc.constraint_name
WHERE tc.constraint_type = 'FOREIGN KEY' AND tc.table_schema='public';
```

Guarda esta info en estructuras Python (diccionarios, JSON).

Paso 2: Indexar metadata

Ejemplo simple: crear un índice invertido o usar FAISS para búsqueda vectorial.

```
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS

# Supongamos que tienes una lista de strings con descripciones de tablas y columnas
metadata_texts = [
    "Tabla ventas: id_venta, fecha, monto, id_cliente",
    "Tabla clientes: id_cliente, nombre, email",
    # ...
]

embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
vectorstore = FAISS.from_texts(metadata_texts, embeddings)
```

Luego, para una consulta:

```
query = "Información sobre ventas y clientes"
docs = vectorstore.similarity_search(query, k=3)
# docs tendrá la metadata más relevante para el prompt
```

Paso 3: Construir prompt y usar LLM

Ejemplo de prompt:

Eres un asistente que convierte lenguaje natural a SQL. Aquí tienes la metadata relevante:

```
{metadata_relevante}
```

Consulta: {consulta_usuario}

Genera la consulta SQL para PostgreSQL.

Paso 4: Ejecutar SQL y mostrar resultado

Usa `psycopg2` o `sqlalchemy` para ejecutar la consulta y obtener resultados.

Paso 5: Interpretar resultado con LLM

Puedes pasar el resultado (o resumen) al LLM para que genere una respuesta en lenguaje natural.

5. Código ejemplo básico con LangChain y GPT4All

```
from langchain.llms import GPT4All
from langchain.chains import LLMChain
from langchain.prompts import PromptTemplate
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS
import psycopg2

# 1. Cargar metadata (ejemplo simplificado)
metadata_texts = [
    "Tabla ventas: id_venta, fecha, monto, id_cliente",
    "Tabla clientes: id_cliente, nombre, email",
]

# 2. Crear índice vectorial
embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
vectorstore = FAISS.from_texts(metadata_texts, embeddings)

# 3. Configurar LLM local GPT4All
llm = GPT4All(model="ggml-gpt4all-j-v1.3-groovy.bin") # modelo local descargado

# 4. Prompt template
template = """
Eres un asistente que convierte lenguaje natural a SQL para PostgreSQL.
Metadata relevante:
{metadata}

Consulta:
{query}

Genera la consulta SQL.
"""

prompt = PromptTemplate(input_variables=["metadata", "query"], template=template)
chain = LLMChain(llm=llm, prompt=prompt)

# 5. Función para ejecutar SQL
def ejecutar_sql(sql_query):
    conn = psycopg2.connect(dbname="tu_db", user="usuario", password="pass", host="localhost")
    cur = conn.cursor()
    cur.execute(sql_query)
    result = cur.fetchall()
    cur.close()
```

```
conn.close()
return result

# 6. Función principal
def text_to_sql(query_usuario):
    # Buscar metadata relevante
    docs = vectorstore.similarity_search(query_usuario, k=2)
    metadata_relevante = "\n".join([doc.page_content for doc in docs])

    # Generar SQL
    sql = chain.run(metadata=metadata_relevante, query=query_usuario)
    print("SQL generado:", sql)

    # Ejecutar SQL
    resultado = ejecutar_sql(sql)
    print("Resultado:", resultado)

    # Aquí podrías pasar resultado a otro LLM para interpretar y responder al usuario
    return resultado

# Ejemplo de uso
if __name__ == "__main__":
    consulta = "Muéstrame las ventas del último mes con el nombre del cliente"
    respuesta = text_to_sql(consulta)
    print(respuesta)
```

6. Plan de proyecto

Fase	Actividades	Duración estimada
1. Análisis y extracción de metadata	Extraer esquema, relaciones, ejemplos SQL	1-2 semanas
2. Indexación y búsqueda de metadata	Implementar vectorstore para metadata	1 semana
3. Integración con LLM local	Configurar GPT4All o similar, crear prompts	1-2 semanas
4. Ejecución segura de SQL	Validar y ejecutar consultas, manejo de errores	1 semana
5. Interpretación de resultados	Generar respuestas en NL con LLM	1 semana
6. Interfaz de usuario	Chatbot web o CLI	2 semanas
7. Pruebas y ajustes	Test con usuarios, optimización	2 semanas

7. Recomendaciones finales

- Para bases muy grandes, considera dividir metadata por áreas o módulos.
- Usa validación estricta para evitar SQL malicioso.
- Considera cachear resultados frecuentes.
- Para producción, evalúa modelos más potentes o servicios cloud si es posible.
- Documenta bien el esquema para facilitar mantenimiento.

Si quieres, puedo ayudarte a extender el código para interpretar resultados o crear la interfaz. ¿Quieres que siga con eso?