Universidade Tecnológica Federal do Paraná - Curitiba, Campus CT
Curso Bacharelado em Sistemas de Informação

Amanda Jury Nakamura
RA2582686
Introdução a Banco de Dados
Profº. Leandro Batista

Relatório de Projeto final

Ferramenta de busca baseada em text-to-sql

Curitiba, Paraná, Brazil
2025

1. Introdução

No panorama atual da análise de dados, a capacidade de extrair informações de maneira eficiente e intuitiva é crucial. Essa extração em bancos de dados relacionais depende do conhecimento aprofundado em linguagens de consulta estruturadas, como o SQL (Structured Query Language).

Text-to-SQL refere-se à capacidade de gerar consultas SQL a partir de uma entrada em linguagem natural. Ou seja, em vez de escrever linhas de código SQL, um usuário pode simplesmente fazer uma pergunta em linguagem natural e o sistema converterá essa pergunta em uma consulta SQL válida. Essa abordagem simplifica drasticamente a interação com bases de dados, tornando a análise de dados acessível.

A ascensão do Text-to-SQL tem sido impulsionada e popularizada, em grande parte, pelos avanços em Grandes Modelos de Linguagem (LLMs). Um LLM é um tipo de inteligência artificial treinada para entender, gerar e processar linguagem humana de forma sofisticada.

No contexto do Text-to-SQL, LLMs são empregados para compreender a intenção por trás da consulta em linguagem natural e mapeá-la para a estrutura de um banco de dados, gerando assim a consulta SQL correspondente. A capacidade dos LLMs de capturar o contexto das perguntas é fundamental para a precisão e a eficácia das ferramentas Text-to-SQL.

2. Objetivos

O objetivo deste documento é relatar o desenvolvimento do projeto final da disciplina de Introdução a Banco de Dados ministrada pelo Professor Leandro Batista de Almeida trazendo aprendizados, dificuldades, e resultados de cada etapa, bem como a organização, softwares e ferramentas utilizadas.

O enunciado exige o desenvolvimento de uma aplicação Python com funcionalidades das bibliotecas Text-to-SQL, integradas com LLMs, para permitir consultas em linguagem natural a bancos de dados convencionais.

O programa será capaz de se conectar ao servidor MySQL, carregar dinamicamente a estrutura das tabelas e seus campos, oferecer uma interface para que o usuário insira perguntas em linguagem natural, converter essas perguntas em consultas SQL executáveis e, por fim, exibir os resultados diretamente na tela.

3. Metodologia

O projeto foi desenvolvido em Python (VERSÃO), o VSCode foi utilizado como o editor e compilador. Ferramentas de inteligência artificial (ChatGPT e Gemini) foram consultadas em diversas etapas como auxílio na ideação do projeto, produção do código, resolução de erros e para buscar alternativas aos processos a serem desenvolvidos em casos de erros.

O sistema de gerenciamento de banco de dados utilizado foi o MySQL, e para os testes, o banco de dados de escolha foi o “University” fornecido anteriormente no semestre para outras atividades.

O LangChain foi o framework de escolha para a construção da aplicação, pois na fase de pesquisa apresentou vantagens sendo uma interface padronizada que interage com diferentes modelos de linguagem. O modelo de linguagem (LLM) utilizado foi o Gemini da Google, sendo a versão 'gemini-1.5-flash-latest' a de escolha após alguns testes a serem descritos posteriormente.

O desenvolvimento do projeto dependeu de bibliotecas específicas como o Langchain e SQLAlchemy para compreender a interação com LLMs e bancos de dados SQL, respectivamente. Para a interface, a biblioteca 'streamlit' foi ideal e de simples aplicação e entendimento.

Todas as informações coletadas foram compiladas em uma página pessoal no Notion para organização e posteriormente utilizadas para a produção deste relatório.

4. Desenvolvimento

Para a organização e melhor entendimento do projeto, a primeira etapa consistiu em uma pesquisa geral sobre o enunciado, principalmente em relação à novos conceitos e softwares ali encontrados, como Text-to-SQL, LLM, script stand-alone e notebook Jupyter. Também foi necessário revisar e ler a documentação da linguagem Python e algumas bibliotecas.

Após as pesquisas e análise completa do enunciado, foi feito um rascunho com os passos a serem desenvolvidos. Isso foi feito com base em tutoriais online e através de canais no youtube, como Alejandro AO. Os passos consistiram em: 1 - escolha do LLM, e framework para a construção da aplicação; 2 - identificação e download das bibliotecas; 3 - conexão com banco de dados; 4 - conversão Text-to-SQL; 5 - Interface para entrada e saída dos dados.

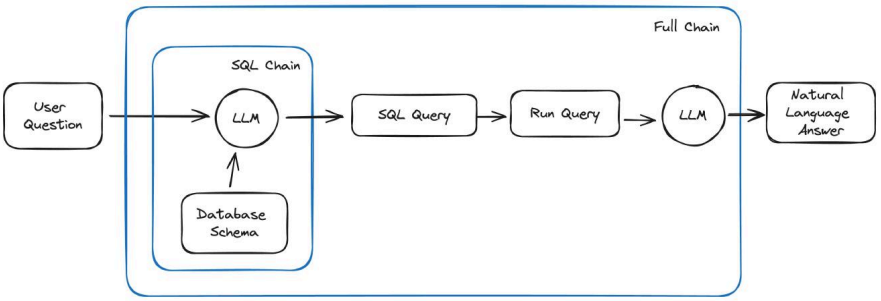


Figura 01. Fonte principal de inspiração para o fluxo de procedimentos definido para este projeto.
Fonte: <https://alejandro-ao.com/chat-with-mysql-using-python-and-langchain/>

Tendo escolhido o LangChain e o Gemini como ferramentas, foi necessário fazer o download das bibliotecas Langchain, SQLAlchemy e 'os':

- `os`: Interação com o sistema operacional, como variáveis de ambiente.
- `sqlalchemy`: ORM para Python, conecta com bancos de dados SQL.
- `langchain_community.utilities.SQLDatabase`: Utilitário da Langchain para interagir com bancos de dados SQL.

- `langchain_experimental.sql`: Módulos experimentais de SQL no Langchain.
- `langchain_google_genai.chat_models`: Interage com modelos de chat do Google Generative AI (Gemini) no Langchain.
- `langchain_community.agent_toolkits.create_sql_agent`: Cria agentes para interagir com bancos de dados SQL no Langchain.

A conexão com o banco de dados utilizou a biblioteca SQLAlchemy, já mencionada, para criar uma engine de conexão de forma flexível, e a SQLAlchemy que usa o driver PyMySQL por baixo para se comunicar com o MySQL, conforme abaixo:

```
engine = create_engine(f"mysql+pymysql://{usuario}:{senha}@{host}/{banco}")
```

O usuário, senha, host e banco, foram definidos anteriormente a criação da variável ‘engine’.

Encontrei algumas dificuldades para realizar essa conexão através de outras bibliotecas como o mysql.connector e db_config que foram as primeiras sugestões, tentei diversas formas, e apesar de funcionar no terminal, não rodava corretamente no VSCode. Para o teste, inseri uma linha de print para saber até onde o código rodava corretamente.

Outro erro que tive que corrigir foi um problema encontrado devido a presença do “@” na senha do banco de dados local. Foi necessário alterar a senha para corrigir o problema.

Para definir o LLM criei uma variável a qual foi atribuída uma função ‘ChatGoogleGenerativeAI’ da biblioteca ‘langchain_google_genai.chat_models’, que recebe como parâmetros o modelo do Gemini e a temperatura, indicando o grau de aleatoriedade nas respostas geradas, sendo graus baixos ideais para respostas mais determinísticas e seguras necessárias na análise de bancos de dados.

Um problema encontrado nesta etapa surgiu na escolha da versão do Gemini. O primeiro modelo testado foi “models/gemini-2.0-flash-live-001” por sugestão do ChatGPT. Ao rodar o código, tinha-se um erro indicando que o modelo escolhido não estava disponível e então, no terminal, já aparecia uma sugestão de comando para printar os modelos disponíveis. A partir daí foi escolhido o gemini-1.5-flash-latest dentro das opções da lista.

A consulta do banco de dados com o LangChain foi feita a partir da criação de uma variável “agente_executor” que recebe a função ‘create_sql_agent(llm, bando de dados, tipo do agente)’ da biblioteca langchain_community.agent_toolkits. Ela é capaz de interpretar linguagem natural, gerar a query SQL, executar no banco e retornar uma resposta. Com a funcionalidade de teste, criei uma variável ‘resposta’ que invoca o agente executor definido previamente e também recebe o input (pergunta). Nessa etapa o input é inserido no código e a resposta é obtida no terminal.

Streamlit é uma biblioteca Python que permite criar aplicativos web simples para projetos de dados e IA com poucas linhas de código, e foi escolhida para fazer a interface deste projeto. Para usá-lo foi necessário ler a documentação e entender as

principais funções que definem principalmente os botões e enunciados. A partir daí, usei Python para programação da interface.

5. Resultados

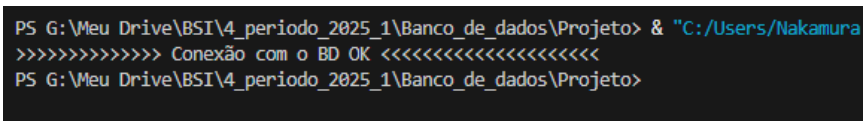
O código final está demonstrado na imagem abaixo e segue como anexo à entrega do projeto:

```python
import os
os.environ["GOOGLE_API_KEY"] = "[redacted]"

from sqlalchemy import create_engine
from langchain_community.utilities import SQLDatabase
from langchain_google_genai.chat_models import ChatGoogleGenerativeAI
from langchain_community.agent_toolkits import create_sql_agent
import streamlit as st #para a interface

# Conecta ao banco de dados
usuario = "root"
senha = "[redacted]"
host = "localhost"
banco = "university"
engine = create_engine(f"mysql+pymysql://{usuario}:{senha}@{host}/{banco}")
db = SQLDatabase(engine)

#teste conexao com o bd
print('>>>>>>>>>>>>>>> Conexão com o BD OK <<<<<<<<<<<<<<<<<<<<')

#llm = ChatGoogleGenerativeAI(model="models/gemini-2.0-flash-live-001", temperature=0) -> problema com o modelo
llm = ChatGoogleGenerativeAI(model="gemini-1.5-flash-latest", temperature=0)
agent_executor = create_sql_agent(llm=llm, db=db, agent_type="openai-tools", verbose=True)
# verbose+True -> permite ver o processo de "Pensamento"

#Testar o código sem a interface
#resposta = agent_executor.invoke({"input": "Qual a soma dos salários dos professores??"})
#print(resposta['output'])

#------------------- INTERFACE----------------------------#
st.title("Pergunte algo sobre o banco de dados University")

# Campo de entrada de texto
query_usuario = st.text_input("Digite sua pergunta:", "")

# Quando o usuário clica no botão
if st.button("Consultar"):
    if query_usuario.strip() == "":
        st.warning("Por favor, digite uma pergunta.")
    else:
        with st.spinner("Consultando o banco de dados..."):
            try:
                resposta = agent_executor.invoke({"input": query_usuario})
                st.success("Resposta:")
                st.write(resposta['output'])
            except Exception as e:
                st.error(f"Erro: {str(e)}")
```

Figura 02. Código completo.

Para alterar o banco de dados a ser consultado, basta alterar o usuário, senha, host e nome do banco de dados nas linhas 10 a 15 do código fonte. Nos testes, foi usado o banco de dados University disponibilizado na disciplina como exemplo e para resolução de exercícios.

O teste da conexão do banco de dados foi satisfatório utilizando 'create_engine' da biblioteca SQLAlchemy, como demonstrado no teste, comando na linha 19 para printar a mensagem:



```
PS G:\Meu Drive\BSI\4_periodo_2025_1\Banco_de_dados\Projeto> & "C:/Users/Nakamura
>>>>>>>>>>>>>>> Conexão com o BD OK <<<<<<<<<<<<<<<<<<<<
PS G:\Meu Drive\BSI\4_periodo_2025_1\Banco_de_dados\Projeto>
```

Figura 03. Resposta obtida no terminal ao teste de conexão ao banco de dados.

O teste da consulta do banco de dados com o LangChain foi através de comandos simples que invocavam o agente executor e passam como parâmetro uma pergunta, como no exemplo:

```
27    #testar o código sem a interface
28    resposta = agent_executor.invoke({"input": "Qual a soma dos salários dos professores de música??"})
29    print(resposta['output'])
30
```

Figura 04. Linha de código referente ao comando de exemplo.

Como no create_sql_agent o parâmetro ‘verbose’ foi definido como True, no terminal é exibido o "raciocínio" e as ferramentas que utiliza para chegar à resposta. Este processo está demonstrado nas figuras 5 e 6.

Ali, é mostrado o processo de chamada da ferramenta ‘sql_db_list_tables’ para obter as tabelas do banco de dados, depois o agente da LangChain define quais as tabelas necessárias para responder à questão, e então invoca o sql_db_schema para descrever as tabelas ao LLM.

O LLM usa as informações obtidas para entender como as tabelas estão estruturadas e conseguir formular a consulta SQL correta para responder à sua pergunta.

```
> Entering new SQL Agent Executor chain...

Invoking: `sql_db_list_tables` with `{}`


advisor, classroom, course, department, instructor, prereq, section, student, takes, teaches, time_slot
Invoking: `sql_db_schema` with `{'table_names': 'instructor, teaches, course'}`
responded: The list of tables returned is: advisor, classroom, course, department, instructor, prereq, section, student, takes, teaches, tim
e\_slot. I will query the schemas of the tables that seem most relevant: instructor, teaches, and course.



CREATE TABLE course (
        course_id VARCHAR(8) NOT NULL,
        title VARCHAR(50),
        dept_name VARCHAR(20),
        credits DECIMAL(2, 0),
        PRIMARY KEY (course_id),
        CONSTRAINT course_ibfk_1 FOREIGN KEY(dept_name) REFERENCES department (dept_name) ON DELETE SET NULL,
        CONSTRAINT course_chk_1 CHECK ((`credits` > 0))
)COLLATE utf8mb4_0900_ai_ci DEFAULT CHARSET=utf8mb4 ENGINE=InnoDB

/*
3 rows from course table:
course_id       title   dept_name       credits
BIO-101 Intro. to Biology       Biology 4
BIO-301 Genetics        Biology 4
BIO-399 Computational Biology   Biology 3
*/


CREATE TABLE instructor (
        `ID` VARCHAR(5) NOT NULL,
        name VARCHAR(20) NOT NULL,
        dept_name VARCHAR(20),
        salary DECIMAL(8, 2),
        PRIMARY KEY (`ID`),
        CONSTRAINT instructor_ibfk_1 FOREIGN KEY(dept_name) REFERENCES department (dept_name) ON DELETE SET NULL,
        CONSTRAINT instructor_chk_1 CHECK ((`salary` > 29000))
)COLLATE utf8mb4_0900_ai_ci DEFAULT CHARSET=utf8mb4 ENGINE=InnoDB

/*
3 rows from instructor table:
ID      name    dept_name       salary
10101   Srinivasan      Comp. Sci.      71500.00
12121   Wu      Finance 90000.00
15151   Mozart  Music   40000.00
*/
```

Figura 05. Parte I do “Raciocínio” para resposta da pergunta.

```
CREATE TABLE teaches (
        `ID` VARCHAR(5) NOT NULL,
        course_id VARCHAR(8) NOT NULL,
        sec_id VARCHAR(8) NOT NULL,
        semester VARCHAR(6) NOT NULL,
        year DECIMAL(4, 0) NOT NULL,
        PRIMARY KEY (`ID`, course_id, sec_id, semester, year),
ELETE CASCADE,
        CONSTRAINT teaches_ibfk_2 FOREIGN KEY(`ID`) REFERENCES instructor (`ID`) ON DELETE CASCADE
)COLLATE utf8mb4_0900_ai_ci DEFAULT CHARSET=utf8mb4 ENGINE=InnoDB

/*
3 rows from teaches table:
ID      course_id       sec_id  semester        year
76766   BIO-101 1       Summer  2017
76766   BIO-301 1       Summer  2018
10101   CS-101  1       Fall    2017
*/
Invoking: `sql_db_query` with `{'query': "SELECT sum(salary) FROM instructor WHERE dept_name = 'Music'"}`


[(Decimal('40000.00'),)]A soma dos salários dos professores de música é 40000.00.


> Finished chain.
A soma dos salários dos professores de música é 40000.00.
```

Figura 06. Parte II do “Raciocínio” para resposta da pergunta.

Para a integração da interface com o streamlit o código feito foi o demonstrado na imagem 07. Ali estão definidos qual enunciado deve estar no título, no campo de entrada de texto, no botão, no loading, e o que deve aparecer no final junto com as respostas.

```python
#------------------- INTERFACE----------------------------#
st.title("Pergunte algo sobre o banco de dados University")

# Campo de entrada de texto
query_usuario = st.text_input("Digite sua pergunta:", "")

# Quando o usuário clica no botão
if st.button("Consultar"):
    if query_usuario.strip() == "":
        st.warning("Por favor, digite uma pergunta.")
    else:
        with st.spinner("Consultando o banco de dados..."):
            try:
                resposta = agent_executor.invoke({"input": query_usuario})
                st.success("Resposta:")
                st.write(resposta['output'])
            except Exception as e:
                st.error(f"Erro: {str(e)}")

```

Figura 07. Código da interface com o streamlit.

O comando para abrir a aplicação é <streamlit run "g:/Meu Drive/BSI/4_periodo_2025_1/Banco_de_dados/ProjetoFinal/finalProjectDB.py"> e a partir daí uma aba no navegador é iniciada com a interface definida no código.

Figura 08. Interface da aplicação streamlit.

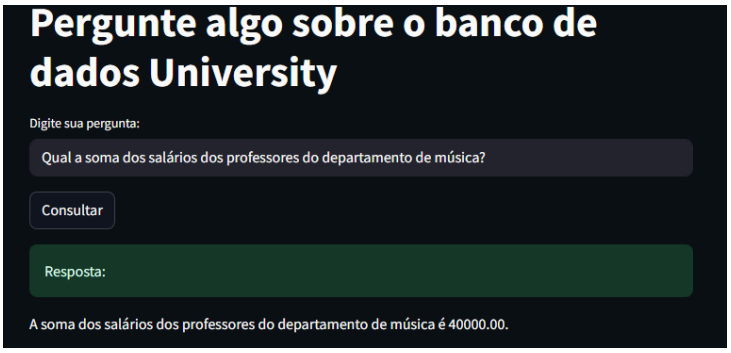


Figura 09. Interface da aplicação streamlit com a resposta à pergunta em linguagem natural..

6. Conclusão

O resultado do projeto foi satisfatório e condizente com o que foi solicitado no enunciado, além de ter sido muito relevante no aprendizado e andamento da disciplina de Introdução a Banco de Dados. O código faz a conexão com um banco de dados corretamente, apresenta uma interface de consulta e é capaz de interpretar linguagem natural para geração da query correta e geração de uma resposta também em linguagem natural.

O processo de desenvolvimento foi, de certa forma, complicado, principalmente no início devido a falta de conhecimentos prévios sobre o tema. Entender o fluxo de dados e os processos pelos quais o projeto devia passar foi um passo crucial, e tutoriais e ferramentas de inteligência artificial foram de grande ajuda.