
SQL-Generierung über Large Language Models

www.bankboard.de - kontakt@bankboard.de

17. Juni 2024

Large Language Models (LLMs) wie beispielsweise ChatGPT oder Gemini sind mächtige Werkzeuge. Daten an diese Modelle weiterzugeben, ist oft problematisch und nicht selten aus Compliance-Gründen verboten. LLMs können in diesem Fall auf einer abstrakteren Ebene eingesetzt werden. Grundlage hierfür ist eine gut dokumentierte Datenstruktur.

In dieser Arbeit wird vereinfacht demonstriert, wie ein LLM (Large Language Model) zur flexiblen Generierung von SQL-Abfragen verwendet werden kann. Als Beispiel dient eine einfache Datenbank mit zwei Tabellen. Der Ablauf ist wie folgt:

1. Es liegt eine gut strukturierte Datenbankbeschreibung vor.
2. Der System-Prompt des LLM enthält ausschließlich die Datenbankbeschreibung (ohne entsprechende Daten) sowie die Anweisung, aus einer verbalen Beschreibung einen für die Datenbank gültigen SQL-Code zu generieren.
3. Der generierte SQL-Code wird zurückgegeben und die Abfrage intern im DV-System verarbeitet.

Das Vorgehen wird in den folgenden Abschnitten näher beschrieben.

1 Datenbankstruktur

Die vereinfachte Datenbank hat zwei Tabellen. Die Tabelle `Kundenstammdaten` beinhaltet die Stammdaten:

Feld	Beschreibung
Kundenname	<p>Typ: VARCHAR(255)</p> <p>Beschreibung: Name des Kunden. Es handelt sich um ein Textfeld, das den vollständigen Namen des Kunden speichert.</p> <p>Schlüssel: Primärschlüssel</p> <p>Indexiert: Ja</p> <p>NULL-Werte erlaubt: Nein</p> <p>Standardwert: Keiner</p>
Branche	<p>Typ: VARCHAR(2)</p> <p>Beschreibung: Schlüssel gemäß Wirtschaftszweige 2008 (WZ 08). Dieses Textfeld speichert in der Form ##.##.##.</p> <p>Schlüssel: Kein</p> <p>Indexiert: Nein</p> <p>NULL-Werte erlaubt: Nein</p> <p>Standardwert: Keiner</p>
PLZ	<p>Typ: VARCHAR(2)</p> <p>Beschreibung: Die ersten beiden Ziffern der Postleitzahl des Kunden. Dieses Feld speichert die Anfangsziffern der PLZ, um den regionalen Standort des Kunden zu identifizieren.</p> <p>Schlüssel: Kein</p> <p>Indexiert: Nein</p> <p>NULL-Werte erlaubt: Nein</p> <p>Standardwert: Keiner</p>
Rating	<p>Typ: INTEGER</p> <p>Beschreibung: Bonitätseinschätzung des Kunden. Es handelt sich um ein numerisches, ganzzahliges Feld mit Werten von 1 bis 22:</p> <ul style="list-style-type: none"> • 1: Sehr gut • 2-20: Abgestufte Bewertung von gut bis sehr schlecht • 21: Im Zahlungsverzug • 22: Im Ausfall <p>Schlüssel: Kein</p> <p>Indexiert: Nein</p> <p>NULL-Werte erlaubt: Nein</p> <p>Standardwert: Keiner</p>

Die zweite Tabelle Transaktionen beinhaltet die Geschäfte der einzelnen Kunden:

Feld	Beschreibung
Kundenname	<p>Typ: VARCHAR(255)</p> <p>Beschreibung: Name des Kunden, der an der Transaktion beteiligt ist. Es handelt sich um ein Textfeld, das den vollständigen Namen des Kunden speichert.</p> <p>Schlüssel: Fremdschlüssel</p> <p>Indexiert: Ja</p> <p>NULL-Werte erlaubt: Nein</p> <p>Standardwert: Keiner</p>
Produkt	<p>Typ: VARCHAR(50)</p> <p>Beschreibung: Art des Finanzprodukts, das in der Transaktion verwendet wird. Mögliche Werte sind:</p> <ul style="list-style-type: none"> • Kredit • Kreditzusage • Schuldschein • Zinsderivate • Zinsswap • Geldhandel • FX-Kassageschäft • FX-Swap • Termingeld • FX-Option • Swaption • Aval <p>Schlüssel: Kein</p> <p>Indexiert: Nein</p> <p>NULL-Werte erlaubt: Nein</p> <p>Standardwert: Keiner</p>
Nominal	<p>Typ: DECIMAL(15,2)</p> <p>Beschreibung: Der Nominalbetrag der Transaktion in der angegebenen Währung. Der Betrag wird als Zahl mit zwei Dezimalstellen gespeichert.</p> <p>Schlüssel: Kein</p> <p>Indexiert: Nein</p> <p>NULL-Werte erlaubt: Nein</p> <p>Standardwert: Keiner</p>

Weiter auf nächster Seite

Fortsetzung von vorheriger Seite

Feld	Beschreibung
Marge	Typ: DECIMAL(6,3) Beschreibung: Die Marge der Transaktion als Prozentsatz auf das Nominal. Die Marge wird als Zahl mit drei Dezimalstellen gespeichert. Schlüssel: Kein Indexiert: Nein NULL-Werte erlaubt: Nein Standardwert: Keiner
Laufzeit	Typ: DATE Beschreibung: Das Datum, an dem die Transaktion abläuft. Es wird als Datum im Format YYYY-MM-DD gespeichert. Schlüssel: Kein Indexiert: Nein NULL-Werte erlaubt: Nein Standardwert: Keiner

Die Tabellen sind über den Kundennamen verknüpft.

2 Umsetzung

Als LLM soll beispielsweise ChatGPT benutzt werden. Folgender Systemprompt wird vorgegeben:

Als SQL-Experte hast Du die Datentabellenbeschreibungen {Kundenstammdaten} und {Transaktionen}. Bitte erstelle auf Basis einer verbalen Anfrage eine SQL-Abfrage. Du kannst notwendige zusätzliche Informationen selbstständig ergänzen. Bitte keine Warnungen oder Verbesserungsvorschläge. Bei unklaren Anweisungen darfst Du nachfragen, ansonsten bitte ausschließlich den SQL-Code ausgeben und SQL-gemäß kommentieren. Bitte ausschließlich die Daten wie in der Datenbeschreibung verwenden und die Anfrage so präzise wie möglich umsetzen. Insbesondere bei Produktanfragen, Branchen und Orten genau überprüfen, inwiefern die Anfrage richtig und vollständig abgedeckt wird.

Im nachfolgenden Python-Code ist der Systemprompt in der Datei systemP.txt abgelegt, die Datenanfrage ist in der Datei Anfrage.txt.

```
1 from openai import OpenAI
2 input_folder = "C:/Daten"
3 client = OpenAI()
```

```

4 with open(f"{input_folder}/systemP.txt", 'r', encoding='utf-8
    ') as file:
5     systemprompt = file.read()
6 with open(f"{input_folder}/Anfrage.txt", 'r', encoding='utf-8
    ') as file:
7     frage = file.read()
8 antwort = client.chat.completions.create(
9     model="gpt-4o",
10    messages=[
11        {"role": "system", "content": systemprompt},
12        {"role": "user", "content": frage}
13    ]
14 )
15 sql_query = antwort.choices[0].message.content
16 print(sql_query)

```

Selbst dieser einfache Aufbau ermöglicht bereits umfassende Abfragen, die verbal vorgegeben werden können:

Anfrage:

Für den Kunden XYZ bräuchte ich eine Ertragsübersicht nach Derivaten, Kredit und Sonstigen Produkten.

Antwort:

```

SELECT
CASE
WHEN Produkt IN ('Zinsswap', 'FX-Swap', 'FX-Option', '
    Swaption') THEN 'Derivate'
WHEN Produkt = 'Kredit' THEN 'Kredit'
ELSE 'Sonstige Produkte'
END AS Produktkategorie,
SUM(Nominal * Marge / 100) AS Ertrag
FROM
Transaktionen
WHERE
Kundenname = 'XYZ'
GROUP BY
Produktkategorie;

```

oder als weiteres Beispiel:

Anfrage:

Liste alle Zinsderivate mit allen Transaktionsdaten auf, die mit Banken getätigt wurden und in den nächsten 6 Monaten fällig werden.

Antwort:

```
SELECT
t.Kundenname,
t.Produkt,
t.Nominal,
t.Marge,
t.Laufzeit
FROM
Transaktionen t
JOIN
Kundenstammdaten k ON t.Kundenname = k.Kundenname
WHERE
k.Branche LIKE '64%'
AND t.Produkt IN ('Zinsswap', 'FX-Swap', 'FX-Option', '
Swaption')
AND t.Laufzeit BETWEEN CURDATE() AND DATE_ADD(CURDATE(),
INTERVAL 6 MONTH);
```

Das SQL-Ergebnis kann nun intern in den entsprechenden Datenbanken verarbeitet und die entsprechenden Datensätze ausgegeben werden. Zu keiner Zeit ist ein Zugriff des LLMs auf die Daten nötig.

3 Fazit

Die Hauptvorteile von LLMs liegen in ihrer Fähigkeit, Aufgaben zu automatisieren, Muster zu erkennen und komplexe Anfragen zu verarbeiten. Diese Vorteile kommen jedoch nur zum Tragen, wenn die zugrunde liegenden Daten klar und strukturiert sind. Ein gut organisierter Datenbestand ermöglicht es dem Modell, effizient zu arbeiten und maximalen Nutzen zu bieten.

Komplexe oder schlecht dokumentierte Daten können zu Verwirrung und ineffizienter Verarbeitung führen. Das Modell könnte Schwierigkeiten haben, relevante Informationen zu extrahieren und korrekt zu interpretieren, was die Komplexität eher erhöht und den Nutzen mindert.

Es müssen nicht immer Daten weitergegeben werden, um LLMs sinnvoll zu nutzen. Die Struktur der Daten hat genügend Informationen, um sinnvolle Erleichterungen im Tagesablauf zu erreichen. Vorausgesetzt, sie sind klar und präzise definiert. Auf Letzterem sollte der primäre Fokus innerhalb eines Unternehmens sein.