

MCP & Datenbanken: KI als Database Co-Pilot

Willkommen zu diesem praktischen Tutorial! Hier lernen Sie, wie Sie künstliche Intelligenz als Helfer für Datenbankabfragen einsetzen können – ohne jede SQL-Anfrage manuell zu schreiben. Wir nutzen dafür das Model Context Protocol (MCP), das GitHub Copilot direkten Zugriff auf Ihre Datenbanken gibt.

Am Ende dieses Tutorials können Sie mit natürlicher Sprache Datenbanken erkunden, Queries generieren lassen und komplexe Analysen durchführen – alles direkt in VS Code!

Lernziele:

- Verstehen, warum KI + Datenbanken eine sinnvolle Kombination sind
- Das Model Context Protocol (MCP) konzeptionell einordnen können
- Einen MCP-Server in VS Code installieren und konfigurieren
- Mit natürlicher Sprache über GitHub Copilot Datenbanken abfragen
- Grenzen und Trade-offs dieser Technologie reflektieren

KI & Datenbanken: Warum überhaupt?

Stellen Sie sich vor: Sie arbeiten an einem Projekt mit einer unbekannten Datenbank. Hunderte Tabellen, Tausende Spalten – und Sie sollen „schnell mal“ eine Auswertung erstellen. Wo fangen Sie an?

Traditionell würden Sie das Schema erkunden, Dokumentation suchen, Beispiel-Queries studieren und dann mühsam SQL schreiben. Aber was, wenn eine KI Ihnen diese Arbeit abnehmen könnte?

Typische Use Cases für KI + DB:

1. **Schema-Exploration** „Zeige mir alle Tabellen in dieser Datenbank.“
2. **Query-Generierung** „Wie viele Filme wurden nach 2010 veröffentlicht?“
3. **Datenanalyse** „Wer sind die Top 10 Schauspieler nach Anzahl der Filme?“
4. **Debugging** „Warum liefert diese Query keine Ergebnisse?“
5. **Dokumentation** „Erkläre mir die Beziehung zwischen `movies` und `actors`.“

Das Problem: KI-Modelle wie ChatGPT oder GitHub Copilot haben keinen direkten Zugriff auf Ihre lokalen Datenbanken. Sie können zwar SQL schreiben, aber nicht ausführen oder das Schema inspizieren. Hier kommt MCP ins Spiel.

Problem ohne MCP:

- KI kann SQL schreiben, aber nicht ausführen
- Keine Kenntnis über das tatsächliche Schema
- Kein Feedback, ob Queries funktionieren
- Manuelle Copy-Paste-Workflows notwendig

Was ist MCP? (Model Context Protocol)

MCP steht für „Model Context Protocol“ – ein offener Standard, der KI-Modellen strukturierten Zugriff auf externe Ressourcen ermöglicht. Entwickelt von Anthropic, aber offen für alle Tools.

Stellen Sie sich MCP wie eine Werkzeugkiste vor: Der MCP-Server stellt der KI verschiedene Tools zur Verfügung – zum Beispiel „*readquery*, oder “*listtables*“. Die KI entscheidet selbst, welches Tool sie wann aufruft, und generiert dabei bereits das benötigte SQL. Der Server führt es nur noch aus!

KI-Client (GitHub Copilot)

- Versteht natürliche Sprache
- Generiert SQL-Queries selbst
- Entscheidet, welches Tool zu nutzen ist

MCP (Tool-Aufrufe)

MCP-Server (@executeautomation/database-server)

Stellt Tools bereit:

- *read_query(sql)* – Führt SELECT-Queries aus
- *write_query(sql)* – Führt INSERT/UPDATE/DELETE aus
- *list_tables()* – Zeigt alle Tabellen
- *describe_table(name)* – Zeigt Spaltenstruktur
- __ und weitere

SQL

Datenbank (SQLite: IMDB)

- Führt SQL aus
- Liefert Ergebnisse zurück

Wichtig zu verstehen: Die KI ist der „Denker“ – sie übersetzt Ihre Frage in SQL und wählt das passende Tool. Der MCP-Server ist der „Ausführer“ – er bietet die Tools an und führt die SQL-Queries aus. Er übersetzt NICHT Ihre natürliche Sprache!

Beispiel-Ablauf:

1. Sie fragen: „Wie viele Filme gibt es in der Datenbank?“
2. KI denkt: „Ich brauche die Tabelle `title_basics` und ein COUNT... ich nutze das Tool `read_query`“
3. KI generiert SQL: `SELECT COUNT(*) FROM title_basics WHERE titleType = 'movie'`
4. KI ruft Tool auf: `read_query(sql="SELECT COUNT(*) ...")`
5. MCP-Server führt aus: Schickt SQL an die Datenbank
6. Datenbank antwortet: `35505`
7. KI präsentiert: „Es gibt 35.505 Filme in der Datenbank.“

Der MCP-Server für SQLite stellt typischerweise 10+ Tools bereit. Sie werden das später beim Start sehen: „Discovered 10 tools“ bedeutet, dass GitHub Copilot jetzt 10 verschiedene Werkzeuge nutzen kann!

Vorteile von MCP:

- **Direkter Zugriff:** KI sieht das echte Schema via `list_tables()` und `describe_table()`
- **Feedback-Loop:** Queries werden ausgeführt, Fehler sofort sichtbar
- **Kontextbewusst:** KI lernt aus vorherigen Tool-Aufrufen
- **Standardisiert:** Ein Protokoll für viele Tools (Copilot, Claude, etc.)
- **Sicher:** Server läuft lokal, keine Cloud-Upserts

Wichtig: MCP-Server laufen lokal auf Ihrem Rechner. Ihre Daten verlassen niemals Ihren Computer – nur die Query-Ergebnisse werden an die KI gesendet.

Architektur-Details:

- **Client:** GitHub Copilot, Claude Desktop, oder andere MCP-fähige Tools
- **Server:** Node.js-Pakete (z. B. `@executeautomation/database-server`)
- **Transport:** JSON-RPC über stdio/HTTP
- **Protokoll:** [MCP Specification](#)
- **Tools:** Jeder MCP-Server definiert eigene Tools (`readquery`, `writequery`, `list_tables`, etc.)



Hands-on: MCP mit SQLite & IMDB installieren

Jetzt wird es praktisch! Wir installieren einen MCP-Server für SQLite und verbinden ihn mit einer echten IMDB-Datenbank. Sie werden sehen: Die Installation ist einfacher, als Sie denken.

Voraussetzungen:

- sqlite installiert (Version 3.0+): <https://sqlite.org/index.html>
- VS Code installiert: <https://code.visualstudio.com/>
- GitHub-Account: <https://github.com>
- GitHub Copilot Extension aktiv (mit aktivem Subscription):
<https://docs.github.com/en/education/about-github-education/github-education-for-students/apply-to-github-education-as-a-student>
- Internetverbindung für Downloads

Schritt 1: IMDB-Datenbank herunterladen

Wir nutzen eine vorbereitete IMDB-Datenbank mit Filmen, Schauspielern, Regisseuren und Bewertungen. Die Datei ist 40 MB groß und liegt auf GitHub.

Download via Terminal:

```
cd ~/Downloads  
curl -L -o imdb-40mb.sqlite "https://github.com/andre-dietrich/Datenbankend...
```

```
curl -L -o 40-mb.sqlite "https://github.com/andre-dietrich/Datenbanksysteme-Vorlesung/raw/refs/heads/main/assets/dat/imdb/40-mb.sqlite"
```

Alternativ können Sie die Datei auch direkt im Browser herunterladen. Merken Sie sich den Speicherort – den brauchen wir gleich!

The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the command:

```
curl -L -o 40-mb.sqlite "https://github.com/andre-dietrich/Datenbanksysteme-Vorlesung/raw/refs/heads/main/assets/dat/imdb/40-mb.sqlite"
```

Below the command, the terminal shows the progress of the download:

Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current			
			Dload	Upload	Total	Spent	Left	Speed		
0	0	0	0	0	0	0	--:--:--	--:--:--	--:--:--	0
100	38.5M	100	38.5M	0	0	4063k	0	0:00:09	0:00:09	--:--:-- 6572k

The download took 10 seconds at 15:00:04.

Schritt 1: Datenbank herunterladen

Die Datenbank enthält drei Haupttabellen: *titlebasics* (alle Medientitel), *namebasics* (Personen) und *title_ratings* (Bewertungen). Insgesamt über 178.000 Titel und 260.000 Personen – von 1878 bis 2025!

Datenquelle & Aufbereitung:

Diese Datenbank basiert auf den [offiziellen IMDB Non-Commercial Datasets](#), wurde jedoch für didaktische Zwecke vereinfacht:

- Fokus auf die am häufigsten bewerteten Filme und Serien
- Bereinigung unvollständiger Einträge
- Reduzierung auf 3 Kerntabellen (Original: 7+ Tabellen)
- Verfügbar in drei Größen:
 - **6 MB** (kompakt)
 - **10 MB** (mittel)
 - **40 MB** (umfangreich)

So bleibt die Datenbank realistisch, aber handhabbar für erste Experimente!

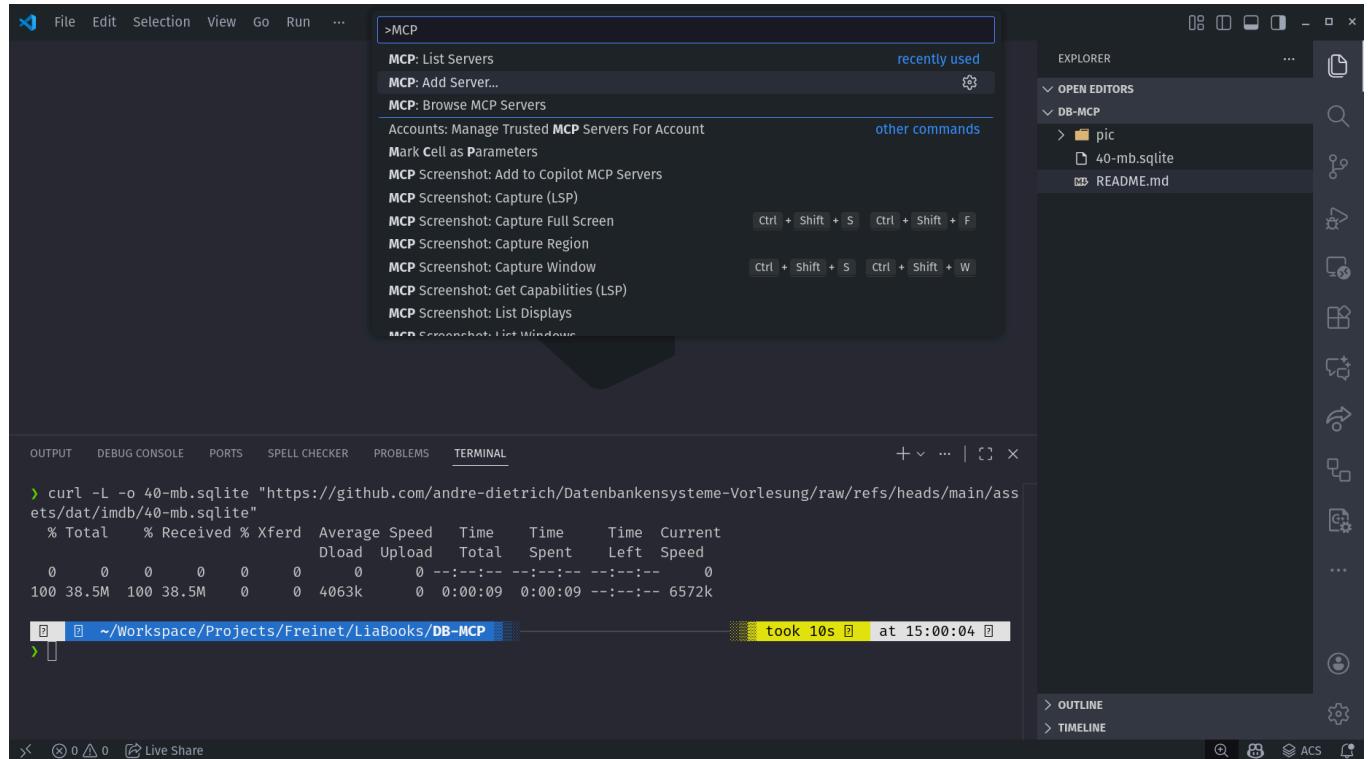
Schritt 2: MCP Database Server installieren

Jetzt installieren wir den MCP-Server. VS Code hat dafür ein praktisches Menü eingebaut – kein Terminal, kein npm nötig!

Öffnen Sie die Command Palette:

- Windows/Linux: **Ctrl + Shift + P**
- macOS: **Cmd + Shift + P**

Tippen Sie „MCP“ und wählen Sie „MCP: Add Server“. VS Code fragt Sie nun nach dem Server-Typ.



Schritt 2: Command Palette öffnen

Wählen Sie „Install from npm package“. Wir nutzen ein fertiges Paket aus dem npm-Registry.

The screenshot shows the Visual Studio Code interface with the MCP (Microservices Configuration) extension open. A context menu is displayed over the MCP icon in the Explorer sidebar, listing options like 'Command (stdio)', 'HTTP (HTTP or Server-Sent Events)', 'NPM Package', 'Pip Package', 'Docker Image', 'Add from another application...', and 'Browse MCP Servers...'. The 'NPM Package' option is selected. The terminal below shows the command `curl -L -o 40-mb.sqlite "https://github.com/andre-dietrich/Datenbankensysteme-Vorlesung/raw/refs/heads/main/assets/dat/imdb/40-mb.sqlite"`, which has completed successfully with a duration of 10 seconds at 15:00:04.

Schritt 3: Installation via npm

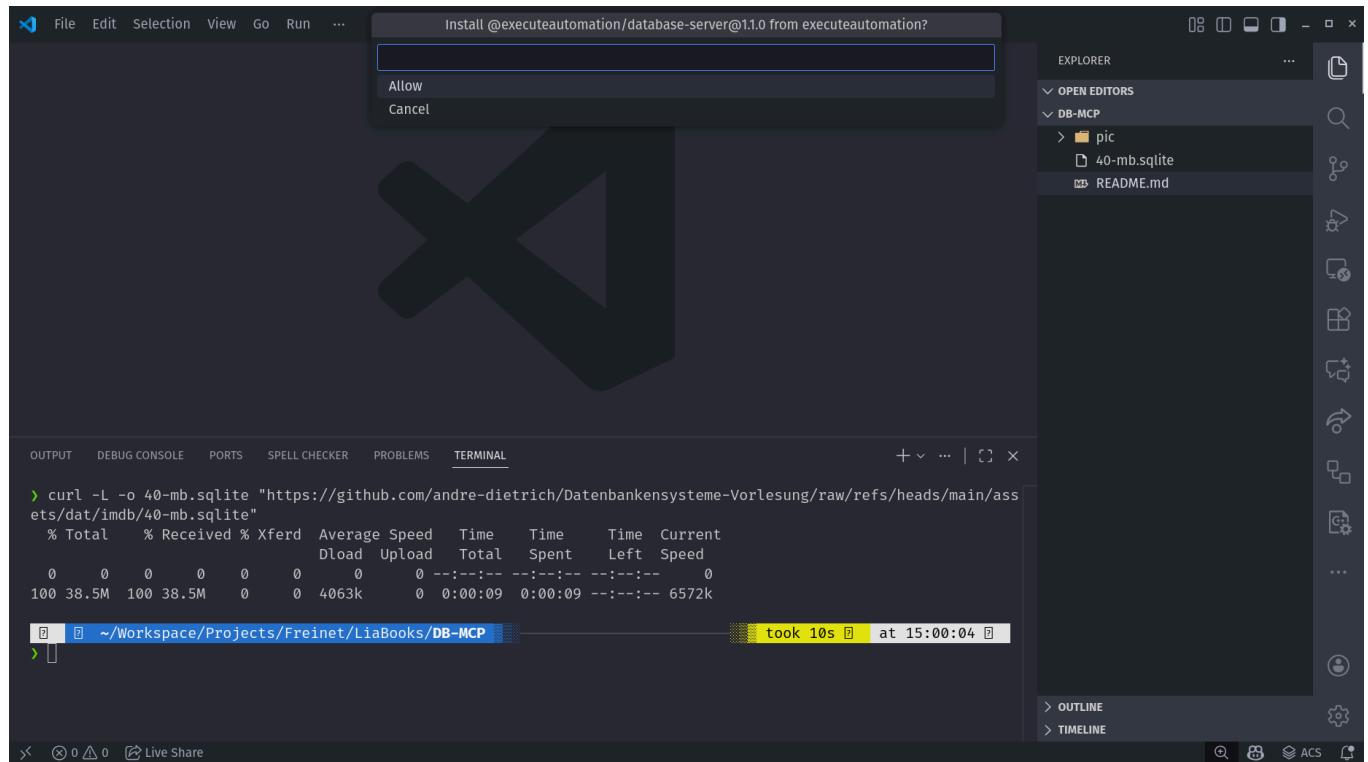
Geben Sie jetzt den Paketnamen ein: `@executeautomation/database-server`. Dies ist ein Community-Server, der SQLite, PostgreSQL und MySQL unterstützt.

<https://www.npmjs.com/package/@executeautomation/database-server>

The screenshot shows the Visual Studio Code interface with the MCP extension open. A search dialog is active in the top center, prompting 'Enter NPM Package Name' with the input field containing '@executeautomation/database-server'. The terminal below shows the same curl command as in the previous step, indicating the package has been successfully installed.

Schritt 4: Package-Name eingeben

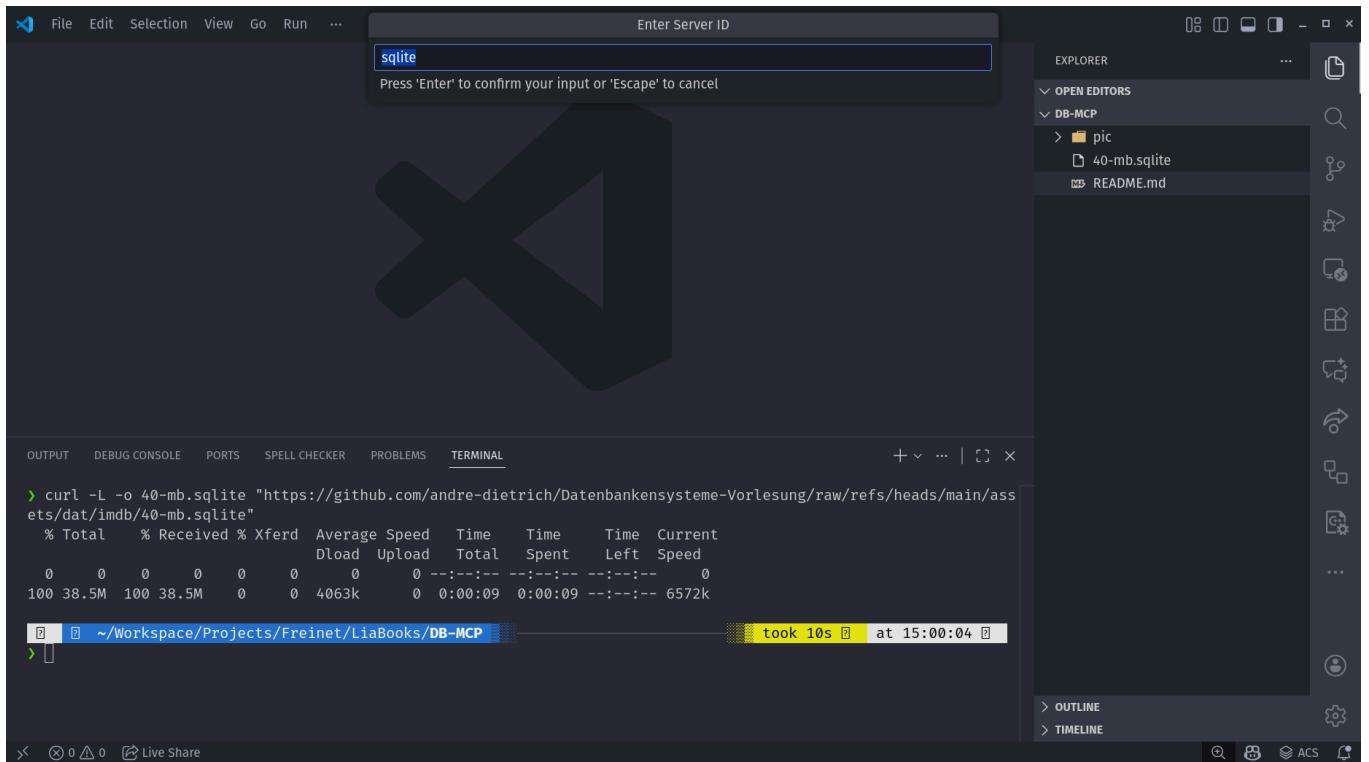
VS Code fragt nach Ihrer Erlaubnis, das Paket zu installieren. Klicken Sie auf „Allow“ – der Server wird nun heruntergeladen und eingerichtet.



Schritt 5: Installation bestätigen

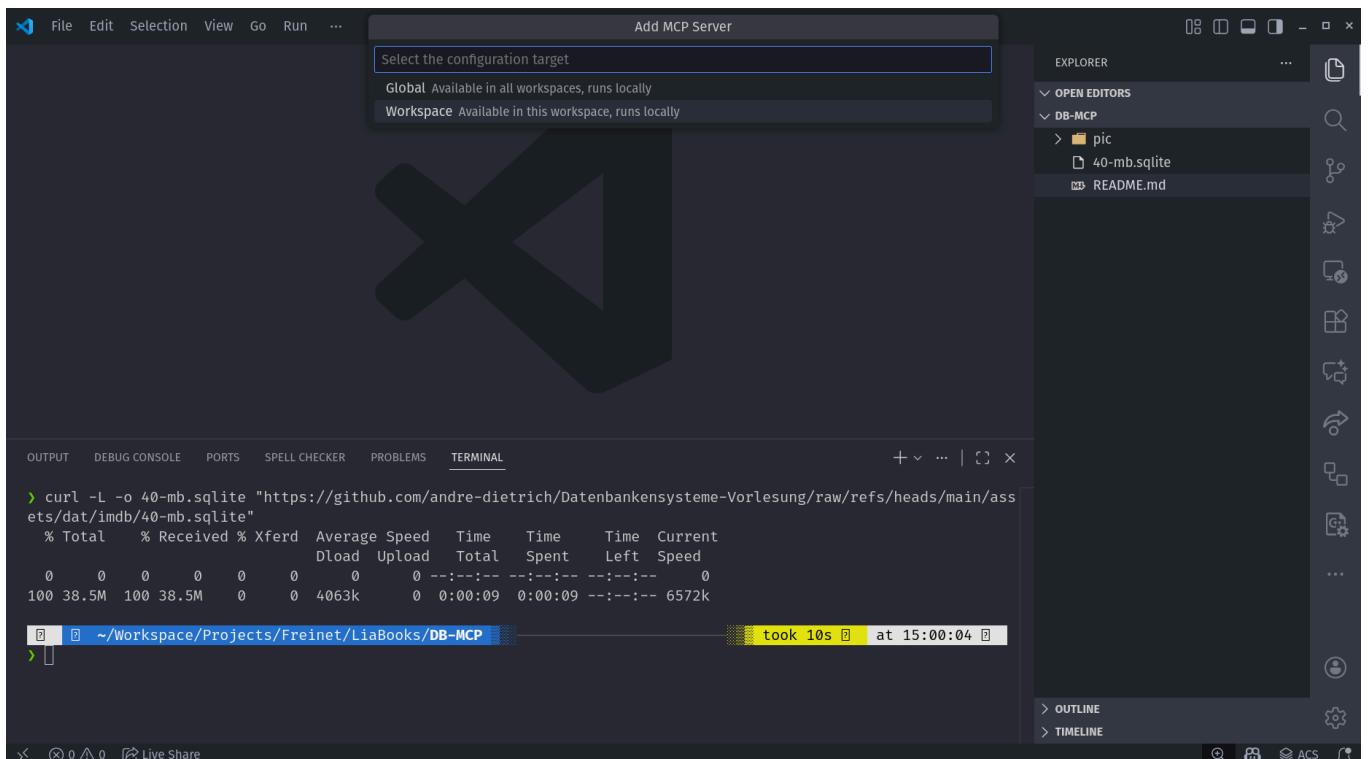
Schritt 3: Server-ID & Konfiguration

Nach der Installation müssen wir dem Server eine eindeutige ID geben. Wählen Sie einen sprechenden Namen, z. B. „sqlite“.



Schritt 6: Server-ID festlegen

VS Code fragt, ob der Server global oder nur für diesen Workspace gelten soll. Wählen Sie „Workspace“ – so bleibt die Konfiguration projektspezifisch.



Schritt 7: Nur für diesen Workspace

Jetzt kommt der wichtigste Schritt: Die Konfiguration. VS Code öffnet automatisch die `mcp.json`-Datei. Hier müssen wir den Pfad zur Datenbank eintragen.

```

{
  "servers": {
    "sqlite": {
      "command": "npx",
      "args": [
        "-y",
        "@executeautomation/database-server",
        "./40-mb.sqlite"
      ],
      "type": "stdio"
    }
  },
  "inputs": []
}

```

Schritt 8: Basis-Konfiguration

Passen Sie den `args`-Abschnitt an: Tragen Sie den vollständigen Pfad zu Ihrer heruntergeladenen Datei ein. `npx` lädt den Server und startet ihn mit der angegebenen Datenbank. `"type": "stdio"` bedeutet, dass die Kommunikation über Standard-Ein- und Ausgabe erfolgt.

Beispiel-Konfiguration:

```

{
  "servers": {
    "sqlite": {
      "command": "npx",
      "args": [
        "-y",
        "@executeautomation/database-server",
        "./40-mb.sqlite"
      ],
      "type": "stdio"
    }
  },
  "inputs": []
}

```

Wichtig: Ersetzen Sie den Pfad durch Ihren tatsächlichen Download-Ordner! Unter Windows nutzen Sie Backslashes oder doppelte Slashes.

```
.vscode > (.) mcp.json > () servers > () sqlite > () args > abc 2
1  {
2    "servers": {
3      "@Error | Restart"
4      "sqlite": {
5        "command": "npx",
6        "args": [
7          "-y",
8          "@executearutomation/database-server",
9          "./40-mb.sqlite"
10         ],
11        "type": "stdio"
12      },
13      "inputs": []
14    }
15  }

OUTPUT DEBUG CONSOLE PORTS SPELL CHECKER PROBLEMS TERMINAL
curl -L -o 40-mb.sqlite "https://github.com/andre-dietrich/Datenbanksysteme-Vorlesung/raw/refs/heads/main/assets/dat/imdb/40-mb.sqlite"
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload Total Spent   Left Speed
0       0     0     0     0       0      0 --:--:-- --:--:-- --:--:-- 0
100 38.5M 100 38.5M 0       0  4063k      0  0:00:09  0:00:09 --:--:-- 6572k
took 10s at 15:00:04
```

Schritt 9: Finalisierte Konfiguration

Schritt 4: Server starten & testen

Geschafft! Die Konfiguration ist abgeschlossen. Jetzt starten wir den Server und prüfen, ob alles funktioniert.

Server-Liste anzeigen:

- Öffnen Sie erneut die Command Palette (**Ctrl** / **Cmd** + **Shift** + **P**)
- Tippen Sie „MCP: List Servers“

The screenshot shows the VS Code interface with the MCP extension installed. The MCP command palette is open, listing commands such as 'Add Server...', 'List Servers', and 'Browse MCP Servers'. The terminal window shows the output of a curl command to download a SQLite database. The Explorer sidebar displays the project structure, including the mcp.json configuration file.

Schritt 10: Verfügbare Server

Sie sollten jetzt Ihren „sqlite“ sehen und auswählen. Klicken Sie darauf, um den Server zu starten, zu stoppen, zu konfigurieren, etc.

The screenshot shows the VS Code interface with the MCP extension installed. The MCP command palette is open, displaying a list of available servers: 'Add Server', 'MCP Screenshot Running', 'Copilot Container Tools Stopped', 'pylance mcp server Stopped', 'sqlite Error Process exited with code 1', and 'microsoft/markdown Stopped'. The terminal window shows the output of a curl command to download a SQLite database. The Explorer sidebar displays the project structure, including the mcp.json configuration file.

Schritt 11: Server auswählen

Klicken Sie auf „Start Server“. Der MCP-Server wird nun gestartet und verbindet sich mit der Datenbank.

The screenshot shows the VS Code interface with the MCP configuration file open. In the center, a modal window titled '\$Select action for 'sqlite'' lists options: Start Server, Show Configuration, Show Output, Configure Model Access, and Browse Resources. Below the modal, the code editor shows the configuration for the SQLite server, which includes a command to run 'npx @executeautomation/database-server' with arguments '-y' and '@executeautomation/database-server' on the file './40-mb.sqlite'. The terminal tab shows the output of a curl command to download the SQLite database file from GitHub.

Schritt 12: Server starten

Der Server wird gestartet – Sie sehen im Output-Tab ob alles geklappt hat. Die Status-Anzeige **Discovered 10 tools** bedeutet das GitHub Copilot Zugriff hat und dabei 10 unterschiedliche Funktionen zur Verfügung stehen!

The screenshot shows the VS Code interface with the MCP configuration file open. The terminal tab displays the output of the curl command, followed by the server logs. The logs show the server starting and discovering 10 tools. The status bar at the bottom right indicates 'Discovered 10 tools'.

Schritt 13: Server läuft erfolgreich

Perfekt! Jetzt können wir mit der eigentlichen Arbeit beginnen: Datenbankabfragen über natürliche Sprache.

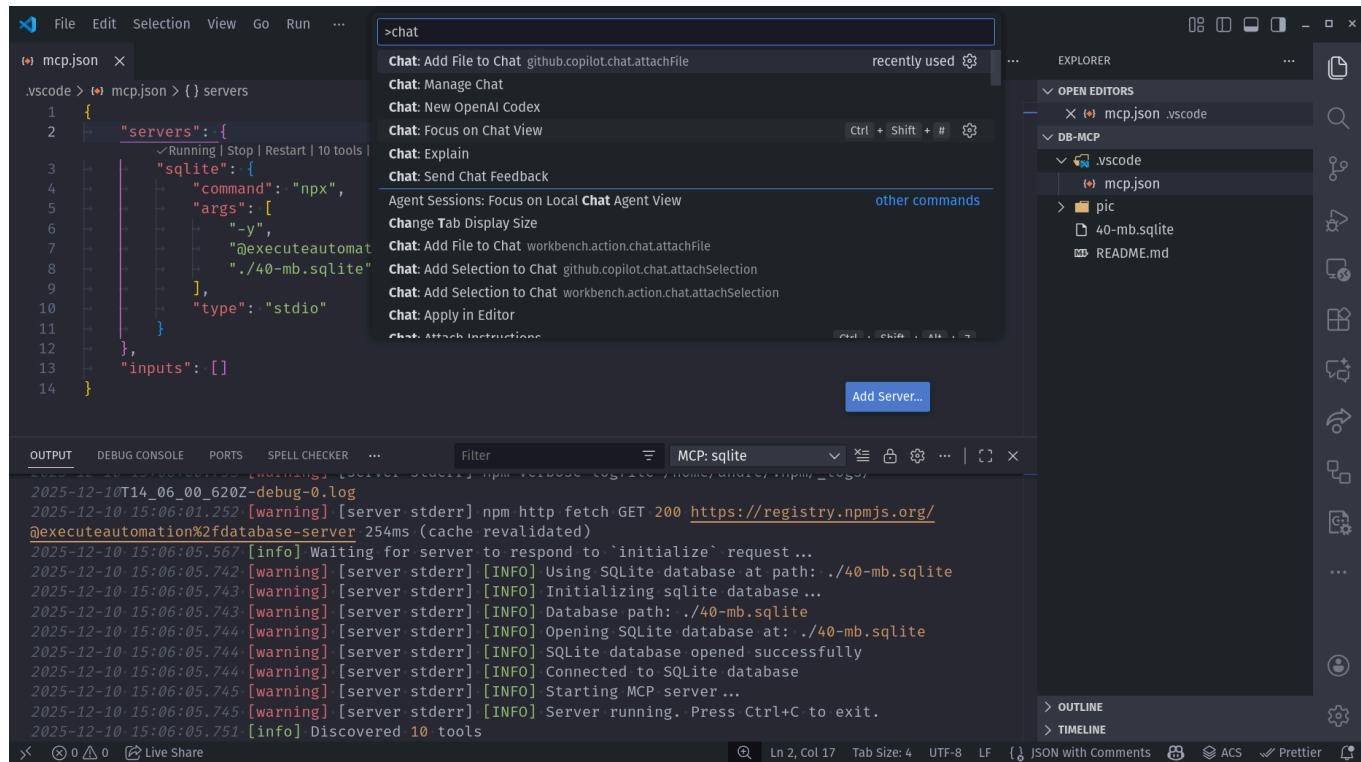
Praktische Anwendung: Mit der DB über KI sprechen

Jetzt kommt der spannende Teil: Sie öffnen GitHub Copilot Chat und stellen Fragen – die KI generiert SQL, führt es aus und zeigt Ihnen die Ergebnisse. Alles automatisch!

Copilot Chat öffnen

Klicken Sie auf das Chat-Icon in der VS Code-Seitenleiste oder nutzen Sie das Keyboard-Shortcut.

- Windows/Linux: **Ctrl + Shift + I**
- macOS: **Cmd + Shift + I**
- Oder nutzen Sie die Command Palette und tippen „Chat“



Schritt 14: Chat-Interface starten

Sie sehen jetzt das Chat-Fenster. Wählen Sie eine KI Ihrer Wahl und wechseln sie in den Agent-Modus.

The screenshot shows the VS Code interface with the MCP extension installed. In the center, the `mcp.json` file is open, displaying configuration for a SQLite database server. The `servers` section includes a command to run `npx @executeautomation/database-server` with arguments `--y ./40-mb.sqlite` and a type of `stdio`. Below the code editor is a terminal window showing logs from the database server starting up and connecting to the SQLite database at `./40-mb.sqlite`. The terminal also indicates that 10 tools are discovered. The right sidebar shows the Explorer view with files like `pic`, `40-mb.sqlite`, and `README.md`.

```
.vscode > mcp.json > () servers
1 {
2   "servers": [
3     {
4       "sqlite": {
5         "command": "npx",
6         "args": [
7           "-y",
8           "@executeautomation/database-server",
9           "./40-mb.sqlite"
10          ],
11         "type": "stdio"
12       },
13     },
14   ],
15   "inputs": []
16 }
17
18 }
```

```
path: ./40-mb.sqlite
2025-12-10 15:06:05.744 [warning] [server stderr] [INFO] Opening
SQLite database at: ./40-mb.sqlite
2025-12-10 15:06:05.744 [warning] [server stderr] [INFO] SQLite
database opened successfully
2025-12-10 15:06:05.744 [warning] [server stderr] [INFO] Connected
to SQLite database
2025-12-10 15:06:05.745 [warning] [server stderr] [INFO] Starting
MCP server ...
2025-12-10 15:06:05.745 [warning] [server stderr] [INFO] Server
running. Press Ctrl+C to exit.
2025-12-10 15:06:05.751 [info] Discovered 10 tools
```

Schritt 15: MCP verbunden

Beispiel: Schema erkunden

Beginnen wir einfach: Lassen Sie sich alle Tabellen anzeigen. Tippen Sie einfach in natürlicher Sprache!

Prompt:

Welche Tabellen gibt es in der IMDB-Datenbank?

GitHub Copilot analysiert Ihre Frage, generiert die passende SQL-Query, führt sie aus und zeigt Ihnen die Ergebnisse – alles in Sekunden. Da hier ein Zugriff auf die echte Resource besteht, müssen Sie jede Anfrage bestätigen, Sie können aber auch den automatischen Modus aktivieren.

The screenshot shows the MCP configuration file (mcp.json) in VS Code. The configuration includes a SQLite server setup with command and arguments to execute automation scripts. The MCP interface window displays logs for connecting to a SQLite database and starting the MCP server.

```

{
  "servers": {
    "sqlite": {
      "command": "npx",
      "args": [
        "-y",
        "@executeautomation/database-server",
        "./40-mb.sqlite"
      ],
      "type": "stdio"
    }
  },
  "inputs": []
}

```

Logs:

```

path: ./40-mb.sqlite
2025-12-10 15:06:05.744 [warning] [server stderr] [INFO] Opening SQLite database at: ./40-mb.sqlite
2025-12-10 15:06:05.744 [warning] [server stderr] [INFO] SQLite database opened successfully
2025-12-10 15:06:05.744 [warning] [server stderr] [INFO] Connected to SQLite database
2025-12-10 15:06:05.745 [warning] [server stderr] [INFO] Starting MCP server ...
2025-12-10 15:06:05.745 [warning] [server stderr] [INFO] Server running. Press Ctrl+C to exit.
2025-12-10 15:06:05.751 [info] Discovered 10 tools

```

Beispiel 1: Tabellen erkunden

Sie sehen jetzt drei Tabellen: `title_basics` (178.124 Titel), `name_basics` (260.050 Personen) und `title_ratings` (Bewertungen). Die KI hat automatisch die passende Query generiert.

The two screenshots show the MCP interface after discovering three tables. The left screenshot shows the initial state with the message 'Discovering 10 tools'. The right screenshot shows the results of the discovery, listing the three tables found: `title_basics`, `name_basics`, and `title_ratings`.

The screenshot shows the VS Code interface with the following details:

- mcp.json** file open in the editor, showing a configuration for a SQLite server named "sqlite". The configuration includes a command to run "npx ./executeautomation/database-serve" with arguments "-y" and "./40-mb.sqlite", and a type of "stdio".
- OUTPUT** panel titled "MCP:sqlite" showing log output from the SQLite database initialization and connection process.
- Log output content:


```
Initializing sqlite database ...
2025-12-10 15:06:05.743 [warning] [server stderr] [INFO]
Database path: ./40-mb.sqlite
2025-12-10 15:06:05.744 [warning] [server stderr] [INFO]
Opening SQLITE database at ./40-mb.sqlite
2025-12-10 15:06:05.745 [warning] [server stderr] [INFO]
Database successfully opened
2025-12-10 15:06:05.746 [warning] [server stderr] [INFO]
Connected to SQLite database
2025-12-10 15:06:05.746 [warning] [server stderr] [INFO]
Starting MCP Server ...
2025-12-10 15:06:05.746 [warning] [server stderr] [INFO]
```

Aufgabe 1: Tabellenstruktur analysieren

Bevor Sie Daten abfragen, sollten Sie verstehen, wie die Datenbank aufgebaut ist. In dieser Aufgabe erkunden Sie die Struktur der Tabellen – welche Spalten gibt es, welche Datentypen, welche Beziehungen?

Die KI kann Ihnen dabei helfen, das Schema zu visualisieren und zu verstehen. Probieren Sie verschiedene Fragen aus!

Prompt-Ideen für Schema-Exploration:

1. Spalten einer Tabelle erkunden:

- „Zeige mir die Struktur der Tabelle `title_basics`.“
- „Welche Spalten hat die Tabelle `name_basics`?“
- „Was für Datentypen haben die Spalten in `title_ratings`?“

2. Beziehungen zwischen Tabellen finden:

- „Wie sind die Tabellen `title_basics` und `title_ratings` verbunden?“
- „Gibt es gemeinsame Spalten zwischen den Tabellen?“
- „Welche Spalte könnte als Primärschlüssel dienen?“

3. Beispieldaten anschauen:

- „Zeige mir 3 Beispielzeilen aus `title_basics`.“
- „Wie sehen die Daten in der `genres`-Spalte aus?“
- „Gibt es NULL-Werte in der Datenbank?“

4. Diagramme generieren:

- „Erstelle ein dbdiagram für die gesamte Datenbank.“
- „Zeige mir ein ASCII-Art-Diagramm der Tabellenbeziehungen.“
- „Visualisiere die Spalten aller Tabellen.“

Probieren Sie mindestens 3–4 dieser Fragen aus! Die KI wird automatisch DESCRIBE-, PRAGMA- oder SELECT-Statements generieren, um das Schema zu analysieren.

Interaktive Visualisierung:

Wenn die KI Ihnen ein **dbdiagram** generiert hat, können Sie es hier einfügen und live rendern lassen:

1
2
3
4
5
6
7
8
9





dbdiagram.io

Beispiel-Struktur für dbdiagram:

```
Table title_basics {  
    tconst TEXT [pk]  
    titleType TEXT  
    primaryTitle TEXT  
    startYear TEXT  
    genres TEXT  
}
```



```
Table title_ratings {  
    tconst TEXT [pk]  
    averageRating TEXT  
    numVotes TEXT
```

```
}
```

Ref: title_basics.tconst – title_ratings.tconst

Wenn Sie die Struktur verstanden haben, können Sie gezielter Fragen stellen. Welche Spalten sind interessant? Welche Tabellen müssen Sie joinen? Das ist die Grundlage für alle weiteren Analysen!

Reflexion:

- Welche Tabelle enthält die meisten Informationen?
- Wie sind die Tabellen miteinander verknüpft? (Hinweis: `tconst`, `nconst`)
- Welche Spalten haben ungewöhnliche Datentypen? (Warum ist `startYear` TEXT statt INTEGER?)
- Gibt es Spalten, die mehrere Werte enthalten? (z. B. `genres`, `primaryProfession`)

Aufgabe 2: Die Filmdatenbank erkunden

Jetzt sind Sie dran! In dieser Aufgabe lernen Sie die IMDB-Datenbank Schritt für Schritt kennen. Stellen Sie der KI die folgenden Fragen – in Ihrer eigenen Sprache, ohne perfektes SQL!

Arbeiten Sie die Fragen der Reihe nach durch. Beobachten Sie, wie die KI Ihre natürliche Sprache in SQL übersetzt und welche Ergebnisse zurückkommen.

Frage 1: Genres erkunden

Stellen Sie dem KI-Agenten diese Frage:

„Wie viele Genres sind in der Datenbank?“

Beobachten Sie:

- Welche Tabelle wählt die KI aus? (Tipp: `title_basics` enthält eine `genres`-Spalte)
- Wie lautet die generierte SQL-Query?
- Wie viele verschiedene Genre-Kombinationen gibt es? (Die Top 3: Comedy ~14k, Drama ~13k, Documentary ~11k)

Die KI wird vermutlich die Tabelle `genres` abfragen und ein `COUNT(*)` oder `COUNT(DISTINCT ...)` verwenden. Notieren Sie sich die Anzahl!

Frage 2: Filme zählen

Nächste Frage:

„Wie viele Filme gibt es in der Datenbank?“

Überlegen Sie:

- Ist die Anzahl höher oder niedriger als erwartet? (Antwort: ~35.500 Filme)
- Aus welchem Zeitraum stammen die Filme? (1878–2025!)
- Was ist mit den anderen ~142.000 Titeln?

Über 178.000 Titel! Aber Vorsicht: Das sind nicht nur Filme, sondern auch TV-Episoden, Serien, Videospiele und mehr. Die KI zeigt Ihnen auch die generierte SQL-Query – so können Sie lernen, wie die Abfrage aufgebaut ist.

Frage 3: Genre-Zuordnungen prüfen

Jetzt wird es interessanter:

„Wie viele Filme sind Genres zugeordnet?“

Vergleichen Sie:

- Ist diese Zahl identisch mit der Gesamtzahl der Filme?
- Wenn nicht: Gibt es Filme ohne Genre?

Bonus-Frage (optional):

„Zeige mir 5 Filme, die keinem Genre zugeordnet sind.“

Hier lernen Sie etwas über Datenqualität: Nicht alle Filme haben vollständige Metadaten. Solche Lücken sind typisch für reale Datenbanken.

Frage 4: Serien in der Datenbank?

Eine spannende Hypothese:

„Gibt es Serien in der Datenbank?“

Tipp: Die KI sucht in der Spalte `titleType` nach `tvSeries`.

Antwort vorweg:

- Ja! ~11.600 TV-Serien
- Plus ~90.500 TV-Episoden!
- Lassen Sie sich ein paar bekannte Titel anzeigen

Die IMDB-Datenbank enthält oft nicht nur Filme, sondern auch TV-Serien, Miniserien oder TV-Specials. Die KI findet das passende Feld automatisch!

Frage 5: Weitere Medientypen entdecken

Letzte Erkundung:

„Was gibt es noch für Medientypen in der Datenbank?“

Erwarten Sie:

- Eine Liste aller `titleType`-Werte
- Die Top 5: `tvEpisode` (90k), `movie` (~35k), `short` (21k), `tvSeries` (~11k), `tvMovie` (~6k)
- Außerdem: `video`, `tvMiniSeries`, `videoGame`, `tvSpecial`, `tvShort`

Bonus-Frage (optional):

„Zeige mir die Verteilung der Medientypen (Anzahl pro Typ).“

Glückwunsch! Sie haben jetzt einen systematischen Überblick über die Datenbank. Sie wissen, welche Tabellen es gibt, wie die Daten strukturiert sind und wo mögliche Lücken existieren.

Reflexion:

- Welche Frage hat Sie überrascht?
- Wie gut hat die KI Ihre Absicht verstanden?
- Hätten Sie die SQL-Queries selbst so geschrieben?
- Welche weiteren Fragen möchten Sie stellen?

Experimentieren Sie weiter! Jede Frage bringt Sie näher daran, die Datenbank intuitiv zu verstehen – und gleichzeitig lernen Sie, wie SQL funktioniert.

Aufgabe 3: Komplexe Analysen durchführen

Jetzt wird es anspruchsvoll! In dieser Aufgabe entwickeln Sie eigene Fragestellungen und lassen die KI komplexere Analysen durchführen. Sie entscheiden selbst, was Sie herausfinden möchten.

Überlegen Sie sich interessante Fragen zur Datenbank und formulieren Sie diese in natürlicher Sprache. Die KI wird automatisch komplexe SQL-Queries mit Joins, Gruppierungen und Aggregationen generieren.

Themenvorschläge (wählen Sie selbst!):

Zeitreihen-Analysen

- Wie hat sich die Filmproduktion über die Jahrzehnte entwickelt?
- Gibt es Trends in bestimmten Genres über die Zeit?
- Welches Jahrzehnt hatte die höchste Produktion?

Tipp: Die `startYear`-Spalte enthält das Erscheinungsjahr. Überlegen Sie, wie man Jahrzehnte gruppieren könnte.

Zeitreihen-Analysen zeigen historische Trends. Die KI kann automatisch Dekaden berechnen und gruppieren – probieren Sie verschiedene Zeiträume aus!

Ratings & Qualität

- Welche Filme haben die höchsten Bewertungen?
- Gibt es einen Zusammenhang zwischen Anzahl der Votes und Rating?
- Welche Genres schneiden im Durchschnitt am besten ab?

Tipp: Die Tabelle `title_ratings` enthält `averageRating` und `numVotes`. Überlegen Sie: Sollten Filme mit nur wenigen Votes berücksichtigt werden?

Ratings-Analysen sind spannend, aber Vorsicht: Ein Film mit 5 Stimmen und Rating 10.0 ist nicht unbedingt besser als einer mit 50.000 Stimmen und 8.5. Wie könnte man das filtern?

Personen & Verbindungen

- Welche Personen haben in den meisten Titeln mitgewirkt?
- Wie viele Schauspieler vs. Regisseure gibt es in der Datenbank?
- Welche Personen sind sowohl Schauspieler als auch Regisseur?

Tipp: Die Tabelle `name_basics` enthält `primaryProfession`. Diese Spalte kann mehrere Berufe enthalten (z. B. „actor,director“).

Personen-Analysen zeigen die Vielseitigkeit von Filmschaffenden. Die KI kann mit String-Operationen wie LIKE oder INSTR in Mehrfach-Kategorien suchen.

Genre-Verteilung

- Welche Genres dominieren die Datenbank?
- Wie viele Filme haben mehrere Genres?
- Gibt es Genre-Kombinationen, die besonders häufig vorkommen?

Tipp: Die `genres`-Spalte kann mehrere Genres enthalten, getrennt durch Kommas (z. B. „Action, Adventure, Sci-Fi“).

Genre-Analysen zeigen Markt-Trends. Beachten Sie: Ein Film kann mehrere Genres haben – das macht die Analyse komplexer, aber auch interessanter!

Datenqualität & Vollständigkeit

- Wie viele Titel haben keine Jahresangabe?
- Welcher Prozentsatz der Filme hat Ratings?
- Gibt es Personen ohne Geburtsjahr?

Tipp: Fehlende Werte werden als `\N` gespeichert (IMDB-Standard). Sie können mit `WHERE spalte = '\N'` oder `WHERE spalte != '\N'` filtern.

Datenqualität ist ein kritisches Thema. Reale Datenbanken haben immer Lücken – diese zu erkennen und zu quantifizieren ist eine wichtige Fähigkeit!

Ihre Aufgabe:

1. Wählen Sie 2-3 Themenbereiche aus der Liste oben
2. Formulieren Sie **eigene Fragen** dazu – in natürlicher Sprache!
3. Lassen Sie die KI die Queries generieren und ausführen
4. **Analysieren Sie die Ergebnisse:** - Sind die Zahlen plausibel? - Welche SQL-Techniken hat die KI verwendet? (JOIN, GROUP BY, HAVING, Subqueries?) - Könnten Sie die Query selbst nachvollziehen oder sogar verbessern?

Experimentieren Sie frei! Es gibt keine „richtige“ Frage – wichtig ist, dass Sie die Ergebnisse verstehen und kritisch hinterfragen. Die KI ist Ihr Werkzeug, aber Sie bleiben der Analyst!

Reflexion: Wann macht MCP Sinn?

MCP ist mächtig – aber nicht für jeden Use Case geeignet. Lassen Sie uns die Stärken und Grenzen analysieren.

Stärken:

-  **Explorative Phasen:** Schnelles Erkunden unbekannter Datenbanken
-  **Prototyping:** Ad-hoc Analysen ohne aufwändige Tool-Setups
-  **Lernhilfe:** Anfänger sehen, wie SQL-Queries aufgebaut sind
-  **Dokumentation:** Automatische Beschreibungen von Schema-Beziehungen
-  **Fehlersuche:** KI kann Queries debuggen und optimieren

Aber es gibt auch klare Grenzen. MCP ist kein Ersatz für fundiertes SQL-Wissen!

Grenzen & Trade-offs:

- **⚠️ Lernkurve:** Verständnis für gute Prompts notwendig
- **⚠️ Abhängigkeit:** Ohne MCP-Server funktioniert nichts
- **⚠️ Performance:** Komplexe Queries können ineffizient sein
- **⚠️ Sicherheit:** Read-only für Produktivdaten empfohlen
- **⚠️ Kosten:** GitHub Copilot ist kostenpflichtig (~10\$/Monat)

Wichtig: MCP ist ein Werkzeug, kein Autopilot. Sie müssen die Ergebnisse verstehen und validieren können.

Best Practice:

1. Nutzen Sie MCP für Exploration & Prototyping
2. Überprüfen Sie generierte Queries kritisch
3. Lernen Sie aus den Beispielen – schreiben Sie dann selbst SQL
4. Für Produktiv-Code: Review & Tests sind Pflicht!

Zusammenfassung & Nächste Schritte

Glückwunsch! Sie haben erfolgreich einen MCP-Server installiert und mit natürlicher Sprache Datenbanken abgefragt. Das öffnet völlig neue Workflows!

Was Sie gelernt haben:

- **✓** Warum KI + Datenbanken sinnvoll kombiniert werden können
- **✓** Wie MCP als Brücke zwischen KI und Daten funktioniert
- **✓** Installation & Konfiguration eines MCP-Servers in VS Code
- **✓** Praktische Datenbankabfragen über GitHub Copilot
- **✓** Grenzen und Trade-offs dieser Technologie

Jetzt sind Sie dran: Probieren Sie weitere Prompts aus, experimentieren Sie mit komplexen Queries und hinterfragen Sie die Ergebnisse kritisch. MCP ist ein Werkzeug – nutzen Sie es weise!

Nächste Schritte:

1.  Erstellen Sie 5 eigene Prompts für die IMDB-Datenbank
2.  Analysieren Sie die generierten SQL-Queries – was ist gut, was ineffizient?
3.  Versuchen Sie, einen zweiten MCP-Server hinzuzufügen (z. B. Ihre eigene SQLite-DB)
4.  Überlegen Sie: Für welche Ihrer Projekte wäre MCP nützlich?

Viel Erfolg beim Ausprobieren – und vergessen Sie nicht: Die beste Lernmethode ist, selbst zu experimentieren. Machen Sie Fehler, verstehen Sie sie, und wachsen Sie daran!



Happy querying! 