https://speakerdeck.com/stoker/

What Can An AI Assistant Do For You?

?

What Can An AI Assistant Do For You?

The presentation will take place in **Winchester** on 2026 April 21 13:00 PDT

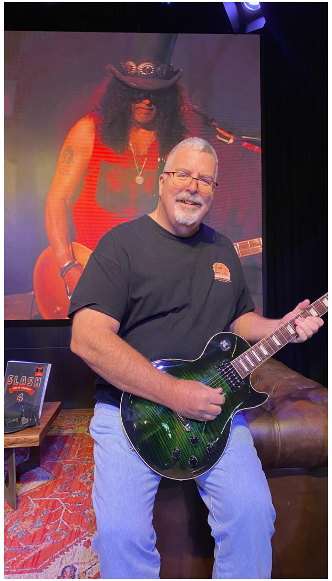
Artificial Intelligence has been over hyped. But it is great at pattern matching and the SQL Syntax is a well known pattern. So if you have an unknown {database | table | query), AI can examine and report on functionality. AI can do the work when your boss wants you to update a working query to add 'quarterly rollups for EMEA'. Syntax errors can easily be spotted AND fixed by your assistant. And you can write queries in English (Or another human language) and let the AI determine the JOINs and GROUP BYs. You will see how using an AI with your database is like the move from VI to IDEs.

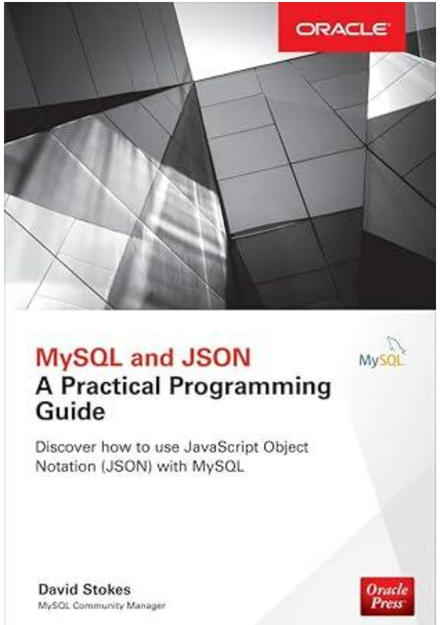
Who Am I?

Dave Stokes

Community Manager at DBeaver

Long time Open Source database proponent




ORACLE

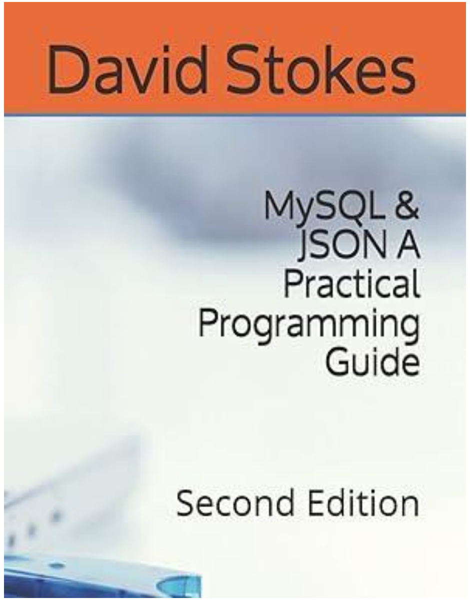
MySQL and JSON
A Practical Programming Guide

Discover how to use JavaScript Object Notation (JSON) with MySQL

MySQL

David Stokes
MySQL Community Manager

Oracle Press


David Stokes

MySQL & JSON A Practical Programming Guide

Second Edition

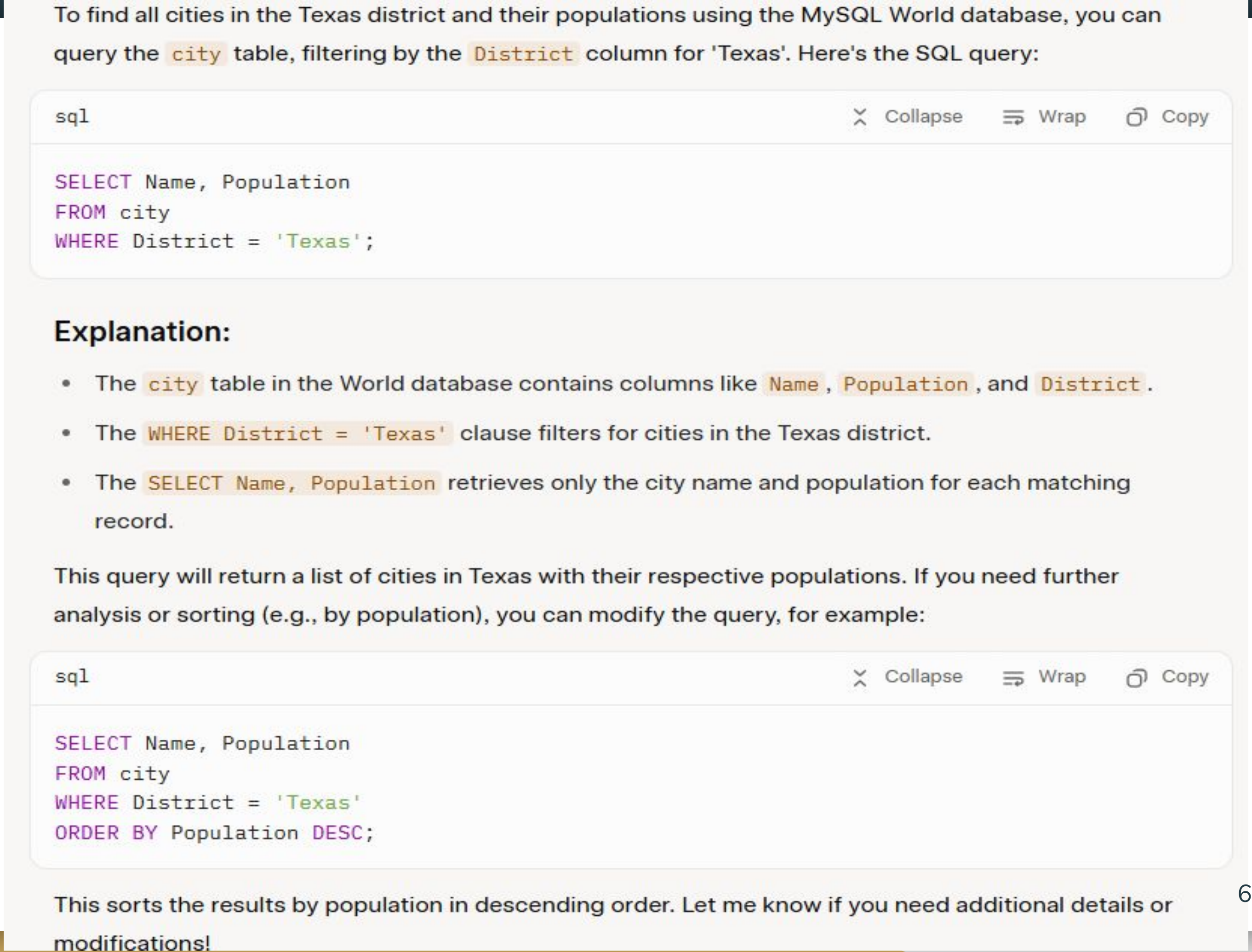
Example 1 -

Ask four LLMs the same question

- Simple Prompt asking an AI directly to ‘do something’. We mention a reference database but are not supplying more details about it.

- ***Using the MySQL World database, how do I find all the cities in the Texas district and their populations?***

Grok



To find all cities in the Texas district and their populations using the MySQL World database, you can query the `city` table, filtering by the `District` column for 'Texas'. Here's the SQL query:

```sql
SELECT Name, Population
FROM city
WHERE District = 'Texas';
```

Explanation:

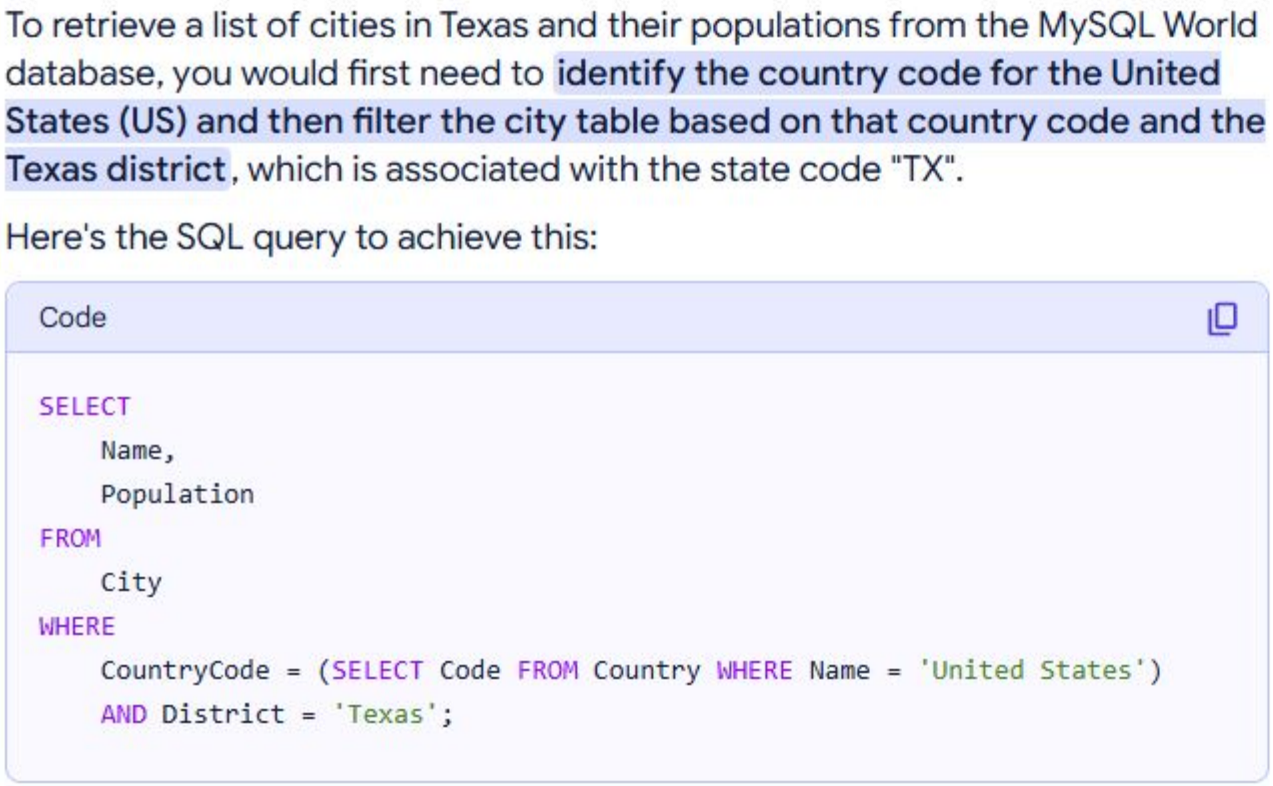
- The `city` table in the World database contains columns like `Name`, `Population`, and `District`.
- The `WHERE District = 'Texas'` clause filters for cities in the Texas district.
- The `SELECT Name, Population` retrieves only the city name and population for each matching record.

This query will return a list of cities in Texas with their respective populations. If you need further analysis or sorting (e.g., by population), you can modify the query, for example:

```sql
SELECT Name, Population
FROM city
WHERE District = 'Texas'
ORDER BY Population DESC;
```

This sorts the results by population in descending order. Let me know if you need additional details or modifications!

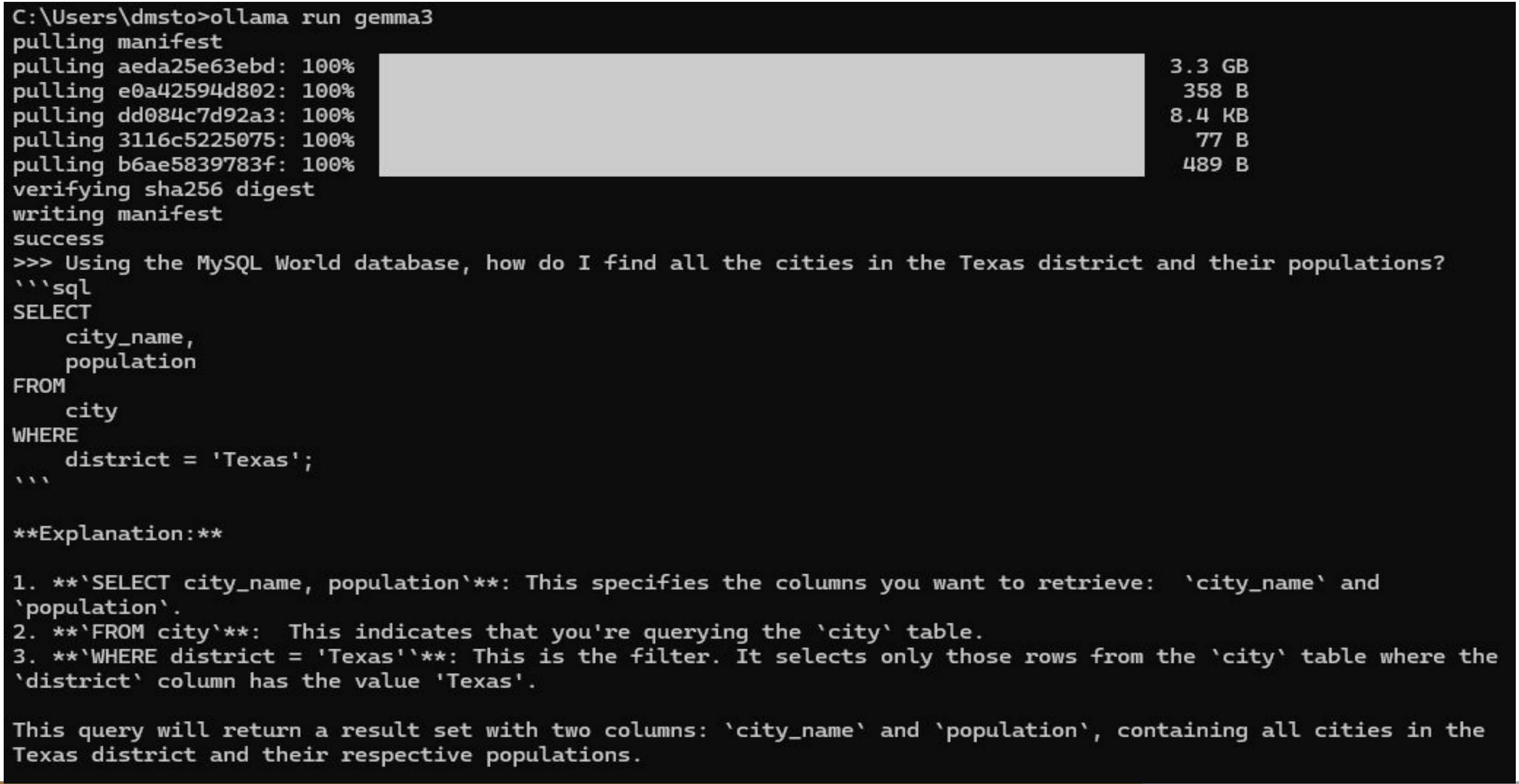
Google



To retrieve a list of cities in Texas and their populations from the MySQL World database, you would first need to **identify the country code for the United States (US) and then filter the city table based on that country code and the Texas district**, which is associated with the state code "TX".

Here's the SQL query to achieve this:

```sql
SELECT
    Name,
    Population
FROM
    City
WHERE
    CountryCode = (SELECT Code FROM Country WHERE Name = 'United States')
    AND District = 'Texas';
```



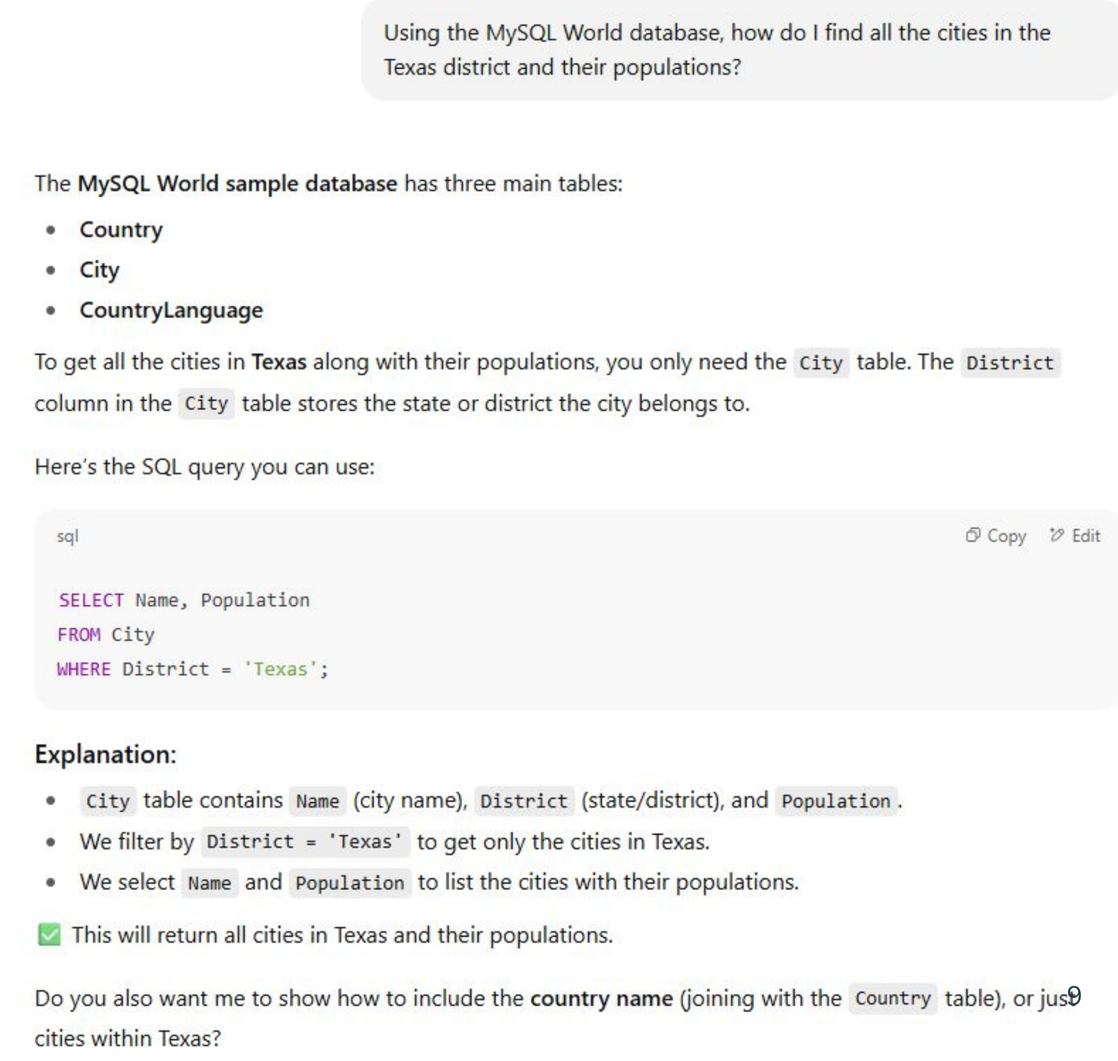
~~~
C:\Users\dmsto>ollama run gemma3
pulling manifest
pulling aeda25e63ebd: 100%                                                        3.3 GB
pulling e0a42594d802: 100%                                                         358 B
pulling dd084c7d92a3: 100%                                                        8.4 KB
pulling 3116c5225075: 100%                                                          77 B
pulling b6ae5839783f: 100%                                                         489 B
verifying sha256 digest
writing manifest
success
>>> Using the MySQL World database, how do I find all the cities in the Texas district and their populations?
```sql
SELECT
    city_name,
    population
FROM
    city
WHERE
    district = 'Texas';
~~~

Explanation:

1. **`SELECT city_name, population`**: This specifies the columns you want to retrieve: `city_name` and
`population`.
2. **`FROM city`**: This indicates that you're querying the `city` table.
3. **`WHERE district = 'Texas'`**: This is the filter. It selects only those rows from the `city` table where the
`district` column has the value 'Texas'.

This query will return a result set with two columns: `city_name` and `population`, containing all cities in the
Texas district and their respective populations.
```
```

ChatGPT



Using the MySQL World database, how do I find all the cities in the Texas district and their populations?

The **MySQL World sample database** has three main tables:

- **Country**
- **City**
- **CountryLanguage**

To get all the cities in **Texas** along with their populations, you only need the `City` table. The `District` column in the `City` table stores the state or district the city belongs to.

Here's the SQL query you can use:

```sql
SELECT Name, Population
FROM City
WHERE District = 'Texas';
```

Explanation:

- `City` table contains `Name` (city name), `District` (state/district), and `Population`.
- We filter by `District = 'Texas'` to get only the cities in Texas.
- We select `Name` and `Population` to list the cities with their populations.

✅ This will return all cities in Texas and their populations.

Do you also want me to show how to include the **country name** (joining with the `Country` table), or just cities within Texas?

Slight variation in the queries produced

```
SELECT
   Name, population
From city
WHERE

  District = ‘Texas’ (Grok, ChatGPT, Ollama)

  CountryCode = (SELECT Code FROM Country WHERE
  Name = 'United States')
      AND District = 'Texas'; (Google)
```

Example 1 - Conclusion

- The previous example used a ‘well known database’ so the LLM ‘knew’ about the schema and the data. But not from us.
- The responses were pretty consistent. All produced good SQL that did the job

🔜 But what about when the data is not so well known?

Example 2 - Customer Table

Write the SQL to create a table on a PostgreSQL 18 server to store customer data. The data will include a first name, a last name, an address, a birthdate, and a unique UUID primary key. Partition data by the year of birth. And produce test data of at least ten records.

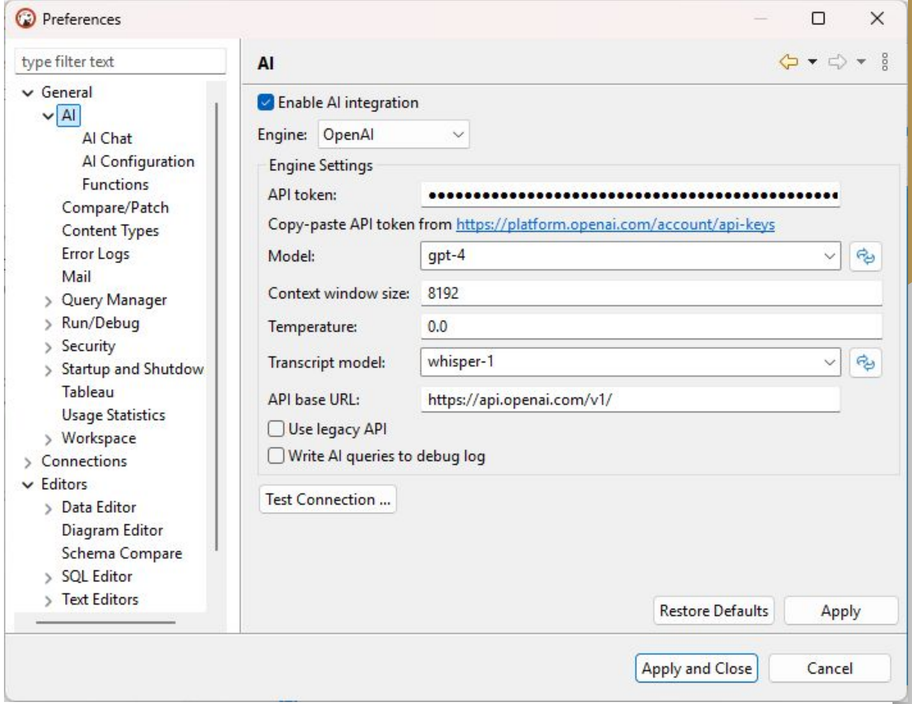
Some personal goals:

I wanted to write NO SQL (not NoSQL) for the following examples (minimal coding on my part).

I used the DBeaver AI Explain and Fix to correct issues.

DBeaver
AI Explain and Fix

opens an AI chat with a detailed explanation of the error and a corrected query, which can be immediately executed.



AI

☑ Enable AI integration

Engine: OpenAI

Engine Settings

API token: ••

Copy-paste API token from https://platform.openai.com/account/api-keys

Model: gpt-4

Context window size: 8192

Temperature: 0.0

Transcript model: whisper-1

API base URL: https://api.openai.com/v1/

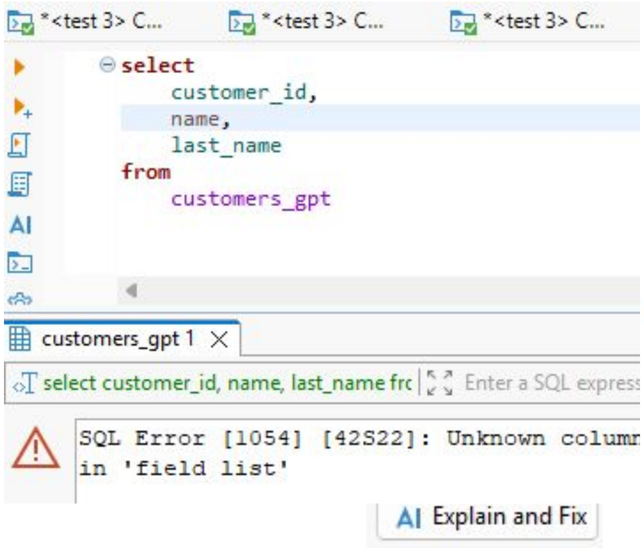
☐ Use legacy API

☐ Write AI queries to debug log

Ask me for a demo

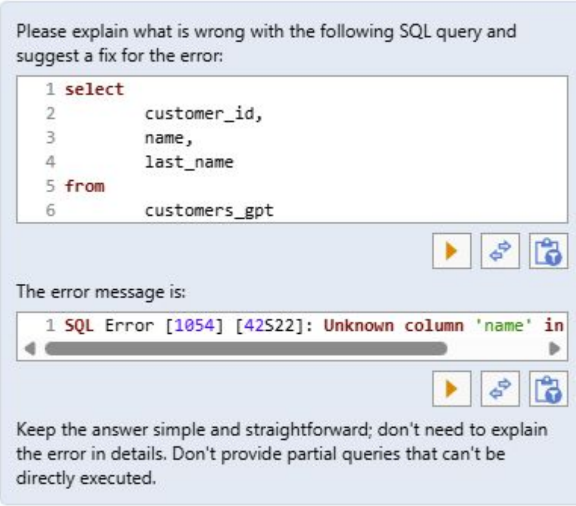
Example:

DBeaver
AI Explain anc



```
select
    customer_id,
    name,
    last_name
from
    customers_gpt
```

```
SQL Error [1054] [42S22]: Unknown column
in 'field list'
```



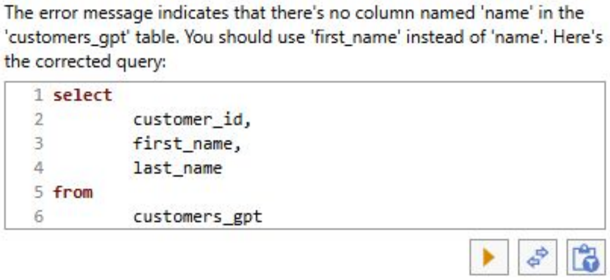
Please explain what is wrong with the following SQL query and suggest a fix for the error:

```
select
        customer_id,
        name,
        last_name
from
        customers_gpt
```

The error message is:

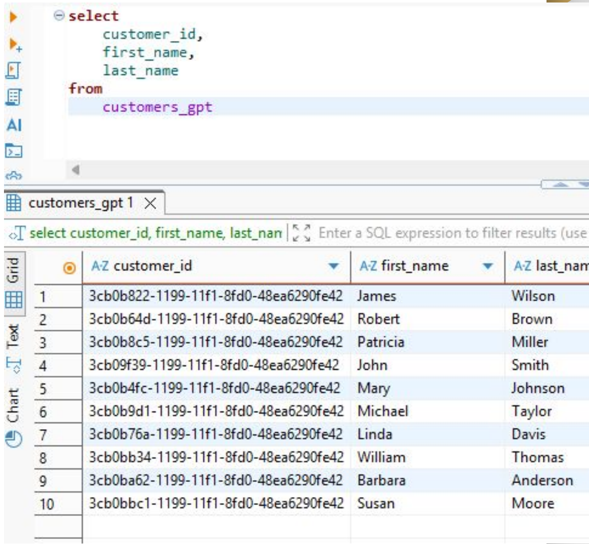
```
SQL Error [1054] [42S22]: Unknown column 'name' in
```

Keep the answer simple and straightforward; don't need to explain the error in details. Don't provide partial queries that can't be directly executed.



The error message indicates that there's no column named 'name' in the 'customers_gpt' table. You should use 'first_name' instead of 'name'. Here's the corrected query:

```
select
        customer_id,
        first_name,
        last_name
from
        customers_gpt
```



```
select
    customer_id,
    first_name,
    last_name
from
    customers_gpt
```

	customer_id	first_name	last_nam
1	3cb0b822-1199-11f1-8fd0-48ea6290fe42	James	Wilson
2	3cb0b64d-1199-11f1-8fd0-48ea6290fe42	Robert	Brown
3	3cb0b8c5-1199-11f1-8fd0-48ea6290fe42	Patricia	Miller
4	3cb09f39-1199-11f1-8fd0-48ea6290fe42	John	Smith
5	3cb0b4fc-1199-11f1-8fd0-48ea6290fe42	Mary	Johnson
6	3cb0b9d1-1199-11f1-8fd0-48ea6290fe42	Michael	Taylor
7	3cb0b76a-1199-11f1-8fd0-48ea6290fe42	Linda	Davis
8	3cb0bb34-1199-11f1-8fd0-48ea6290fe42	William	Thomas
9	3cb0ba62-1199-11f1-8fd0-48ea6290fe42	Barbara	Anderson
10	3cb0bbc1-1199-11f1-8fd0-48ea6290fe42	Susan	Moore

So *how well* did they do?

(surprisingly poorly, they work better with other databases)

```sql
-- Enable the pgcrypto extension for gen_random_uuid() (safe to run if already installed)
CREATE EXTENSION IF NOT EXISTS pgcrypto;
-- Parent table with declarative partitioning by birthdate (yearly ranges = "by year of birth")
-- Primary key must include the partition key (birthdate) per PostgreSQL rules
CREATE TABLE grok_customers (
   id         UUID         NOT NULL DEFAULT gen_random_uuid(),
   first_name VARCHAR(100) NOT NULL,
   last_name  VARCHAR(100) NOT NULL,
   address    TEXT         NOT NULL,
   birthdate  DATE         NOT NULL,
   PRIMARY KEY (id, birthdate)
)
PARTITION BY RANGE (birthdate);
```

Grok

```sql
-- Create yearly partitions (one per relevant year + a default catch-all)
CREATE TABLE customers_y1980 PARTITION OF customers
   FOR VALUES FROM ('1980-01-01') TO ('1981-01-01');
CREATE TABLE customers_y1985 PARTITION OF customers
   FOR VALUES FROM ('1985-01-01') TO ('1986-01-01');
CREATE TABLE customers_y1990 PARTITION OF customers
   FOR VALUES FROM ('1990-01-01') TO ('1991-01-01');
CREATE TABLE customers_y1995 PARTITION OF customers
   FOR VALUES FROM ('1995-01-01') TO ('1996-01-01');
CREATE TABLE customers_y2000 PARTITION OF customers
   FOR VALUES FROM ('2000-01-01') TO ('2001-01-01');
CREATE TABLE customers_y2005 PARTITION OF customers
   FOR VALUES FROM ('2005-01-01') TO ('2006-01-01');
CREATE TABLE customers_y2010 PARTITION OF customers
   FOR VALUES FROM ('2010-01-01') TO ('2011-01-01');
-- Default partition for any birthdates outside the defined years
CREATE TABLE customers_default PARTITION OF customers DEFAULT;
```

Grok

```
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

```
CREATE TABLE chatgpt_customers (
   id UUID PRIMARY KEY DEFAULT
gen_random_uuid(),
   first_name TEXT NOT NULL,
   last_name TEXT NOT NULL,
   address TEXT,
   birthdate DATE NOT NULL
) PARTITION BY RANGE (birthdate);
```

```
CREATE TABLE chatgpt_customers (
   id UUID,
   first_name TEXT NOT NULL,
   last_name TEXT NOT NULL,
   address TEXT,
   birthdate DATE NOT NULL,
   PRIMARY KEY (id, birthdate)
) PARTITION BY RANGE (birthdate);
```

```
SQL Error [0A000]: ERROR: unique
constraint on partitioned table
must include all partitioning
columns
 Detail: PRIMARY KEY constraint on
table "chatgptx_customers" lacks
column "birthdate" which is part of
the partition key.
```

```
CREATE TABLE customers_1980s PARTITION OF chatgpt_customers
FOR VALUES FROM ('1980-01-01') TO ('1990-01-01');
CREATE TABLE customers_1990s PARTITION OF chatgpt_customers
FOR VALUES FROM ('1990-01-01') TO ('2000-01-01');
CREATE TABLE customers_2000s PARTITION OF chatgpt_customers
FOR VALUES FROM ('2000-01-01') TO ('2010-01-01');
CREATE TABLE customers_2010s PARTITION OF chatgpt_customers
FOR VALUES FROM ('2010-01-01') TO ('2020-01-01');
CREATE TABLE customers_other PARTITION OF chatgpt_customers
DEFAULT;
```

ChatGPT

Google Gemini

```sql
-- Ensure UUID generation is available
CREATE EXTENSION IF NOT EXISTS "pgcrypto";
-- Create the parent table
CREATE TABLE google_customers (
   customer_id UUID DEFAULT gen_random_uuid(),
   first_name TEXT NOT NULL,
   last_name TEXT NOT NULL,
   address TEXT,
   birthdate DATE NOT NULL,
   PRIMARY KEY (customer_id, birthdate)
) PARTITION BY RANGE (birthdate);
```

```sql
-- Partition for the 80s
CREATE TABLE gcustomers_1980s
PARTITION OF google_customers
   FOR VALUES FROM ('1980-01-01')
TO ('1990-01-01');
-- Partition for the 90s
CREATE TABLE gcustomers_1990s
PARTITION OF google_customers
   FOR VALUES FROM ('1990-01-01')
TO ('2000-01-01');
-- Partition for the 2000s
CREATE TABLE gcustomers_2000s
PARTITION OF google_customers
   FOR VALUES FROM ('2000-01-01')
TO ('2010-01-01');
```

```sql
INSERT INTO google_customers (first_name, last_name, address, birthdate) VALUES
('Arthur', 'Dent', '42 Miller’s Lane, West Country', '1982-05-11'),
('Tricia', 'McMillan', 'Highgate, London', '1985-09-21'),
('Ford', 'Prefect', 'Betelgeuse Seven', '1980-03-03'),
('Ellen', 'Ripley', 'Nostromo Deck A', '1992-01-07'),
('Kara', 'Thrace', 'Caprica City', '1995-12-14'),
('William', 'Adama', 'Qualacta Sector', '1990-04-18'),
('Marty', 'McFly', '9303 Lyon Drive, Hill Valley', '2005-06-12'),
('Buffy', 'Summers', '1630 Revello Drive, Sunnydale', '2001-01-19'),
('Dana', 'Scully', 'Annapolis, Maryland', '1984-02-23'),
('Fox', 'Mulder', 'Chilmark, Massachusetts', '1981-10-13');
```

Google Gemini

```sql
-- Create the Customer table with UUID primary key and yearly partitioning
CREATE TABLE ollama_customer (
   customer_id UUID PRIMARY KEY DEFAULT gen_random_uuid
   first_name VARCHAR(255) NOT NULL,
   last_name VARCHAR(255) NOT NULL,
   address TEXT,
   birthdate DATE NOT NULL,
   birth_year INT NOT NULL, -- For partitioning
   created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
   updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
) PARTITION BY RANGE (birth_year);
-- Create the yearly partitions
CREATE TABLE ocustomer_2000_2009 PARTITION OF ollama_customer FOR VALUES FROM (1999) TO (2010);
CREATE TABLE ocustomer_2010_2019 PARTITION OF ollama_customer FOR VALUES FROM (2010) TO (2020);
CREATE TABLE ocustomer_2020_2029 PARTITION OF ollama_customer FOR VALUES FROM (2020) TO (2030);
CREATE TABLE ocustomer_2030_ PARTITION OF ollama_customer FOR VALUES FROM (2030) TO (MAXVALUE);
```

```
SQL Error [0A000]: ERROR: unique
constraint on partitioned table must
include all partitioning columns
 Detail: PRIMARY KEY constraint on table
"ollama_customer" lacks column
"birth_year" which is part of the
partition key.
```

Ollama Gemma3.5

```sql
-- Insert Test Data
INSERT INTO ollama_customer (first_name, last_name, address, birthdate, birth_year) VALUES
('Alice', 'Smith', '123 Main St, Anytown', '1995-05-10', 1995),
('Bob', 'Johnson', '456 Oak Ave, Anytown', '2002-11-22', 2002),
('Charlie', 'Brown', '789 Pine Ln, Anytown', '1988-03-15', 1988),
('David', 'Williams', '101 Elm Rd, Anytown', '2011-07-08', 2011),
('Eve', 'Davis', '222 Maple Dr, Anytown', '2005-09-28', 2005),
('Frank', 'Miller', '333 Birch Ct, Anytown', '1992-12-03', 1992),
('Grace', 'Wilson', '444 Cedar St, Anytown', '2013-04-17', 2013),
('Henry', 'Moore', '555 Walnut Ave, Anytown', '2000-06-25', 2000),
('Ivy', 'Taylor', '666 Cherry Ln, Anytown', '1985-01-12', 1985),
('Jack', 'Anderson', '777 Willow Rd, Anytown', '2016-08-30', 2016),
('Kevin', 'Thomas', '888 Pine St, Anytown', '1998-04-05', 1998);
```

```
SQL Error [23514]: ERROR: no
partition of relation
"ollama_customer" found for row
 Detail: Partition key of the
failing row contains (birth_year) =
(1995).
"ollama_customer" lacks column
"birth_year" which is part of the
partition key.
```

The error message indicates that the table "ollama_customer" is a partitioned table and there is no partition available for the value 1995 in the column "birth_year".
To fix this error, you need to create a partition in the "ollama_customer" table that can accommodate the "birth_year" value of 1995 and other years that are being inserted.
However, without knowing the partitioning strategy of your table (range, list, etc.), it's impossible to provide a specific SQL command to fix the issue.
You should consult with your database administrator or the person who designed the table to understand the partitioning strategy and create the necessary partitions.

Ollama Gemma3.5

Approaches to table design

	Grok	Google	Ollama	ChatGPT
id	UUID	UUID	UUID	UUID
names	VARCHAR(100)	TEXT	TEXT	VARCHAR(255)
birthdate	DATE	DATE	DATE	DATE
PK	id, birthdate	id, birthdate	id, birth_year	birth_year, customer_id

Approaches to partitioning

Grok

```
-- Create yearly partitions
(one per relevant year + a
default catch-all)
CREATE TABLE customers_y1980
PARTITION OF customers
    FOR VALUES FROM
('1980-01-01') TO
('1981-01-01');
CREATE TABLE customers_y1985
PARTITION OF customers
    FOR VALUES FROM
('1985-01-01') TO
('1986-01-01');
CREATE TABLE customers_y1990
PARTITION OF customers
    FOR VALUES FROM
('1990-01-01') TO
('1991-01-01');
CREATE TABLE customers_y1995
PARTITION OF customers
    FOR VALUES FROM
('1995-01-01') TO
('1996-01-01');
CREATE TABLE customers_y2000
```

Google

```
-- Partition for the 80s
CREATE TABLE gcustomers_1980s
PARTITION OF google_customers
    FOR VALUES FROM
('1980-01-01') TO
('1990-01-01');
-- Partition for the 90s
CREATE TABLE gcustomers_1990s
PARTITION OF google_customers
    FOR VALUES FROM
('1990-01-01') TO
('2000-01-01');
-- Partition for the 2000s
CREATE TABLE gcustomers_2000s
PARTITION OF google_customers
    FOR VALUES FROM
('2000-01-01') TO
('2010-01-01');
```

Ollama

```
CREATE TABLE
ocustomer_2000_2009
PARTITION OF
ollama_customer FOR VALUES
FROM (1999) TO (2010);
CREATE TABLE
ocustomer_2010_2019
PARTITION OF
ollama_customer FOR VALUES
FROM (2010) TO (2020);
CREATE TABLE
ocustomer_2020_2029
PARTITION OF
ollama_customer FOR VALUES
FROM (2020) TO (2030);
CREATE TABLE
ocustomer_2030_ PARTITION
OF ollama_customer FOR
VALUES FROM (2030) TO
(MAXVALUE);
```

ChatGPT

```
CREATE TABLE
customers_1980s PARTITION
OF chatgpt_customers
FOR VALUES FROM
('1980-01-01') TO
('1990-01-01');
CREATE TABLE
customers_1990s PARTITION
OF chatgpt_customers
FOR VALUES FROM
('1990-01-01') TO
('2000-01-01');
CREATE TABLE
customers_2000s PARTITION
OF chatgpt_customers
FOR VALUES FROM
('2000-01-01') TO
('2010-01-01');
CREATE TABLE
customers_2010s PARTITION
OF chatgpt_customers
FOR VALUES FROM
('2010-01-01') TO
('2020-01-01');
```

Example 2 - Conclusion

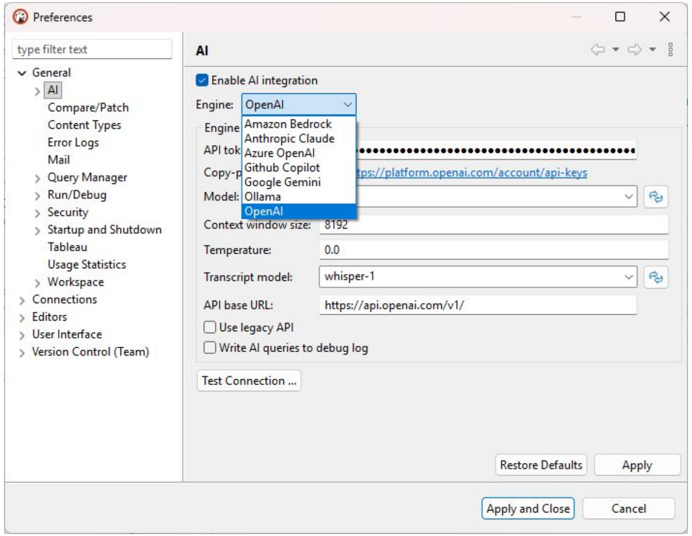
- Creating new table by LLM is a mixed bag
- All four LLMs do better with SQLite and MySQL than PostgreSQL
- Wild partitioning schemes
- Probably not best choices on character fields

DBeaver’s AI Assistant

- Security
    - By default, gets only the metadata (DDL) of the schema
    - Optionally
        - Foreign Keys
        - Data
- You can select tables to be used, exclude others

DBeaver Enterprise

- Default is to only send metadata
- Optional
  - PK, Unique, and FKs.
  - Data



AI

☑ Enable AI integration

Engine: OpenAI

API tok ••

Copy-p tps://platform.openai.com/account/api-keys

Model:

Context window size: 8192

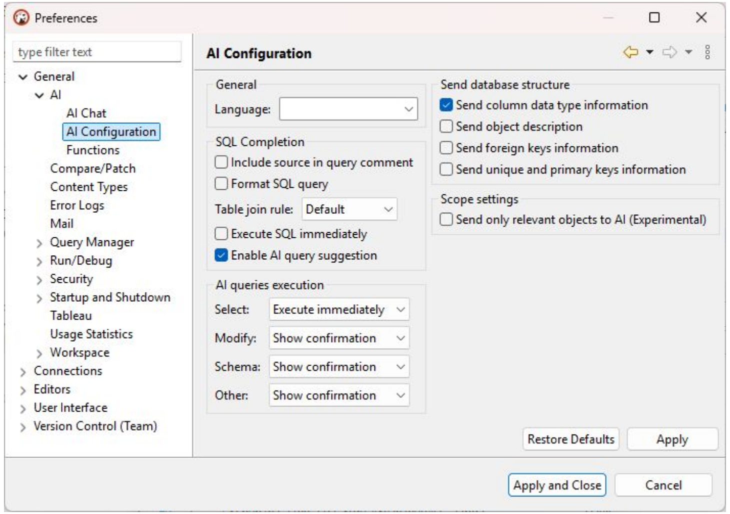
Temperature: 0.0

Transcript model: whisper-1

API base URL: https://api.openai.com/v1/

☐ Use legacy API

☐ Write AI queries to debug log



AI Configuration

General

Language:

SQL Completion

☐ Include source in query comment

☐ Format SQL query

Table join rule: Default

☐ Execute SQL immediately

☑ Enable AI query suggestion

AI queries execution

Select: Execute immediately

Modify: Show confirmation

Schema: Show confirmation

Other: Show confirmation

Send database structure

☑ Send column data type information

☐ Send object description

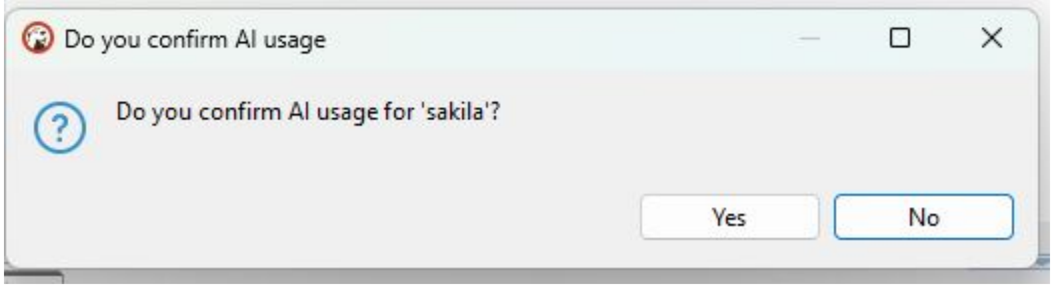
☐ Send foreign keys information

☐ Send unique and primary keys information

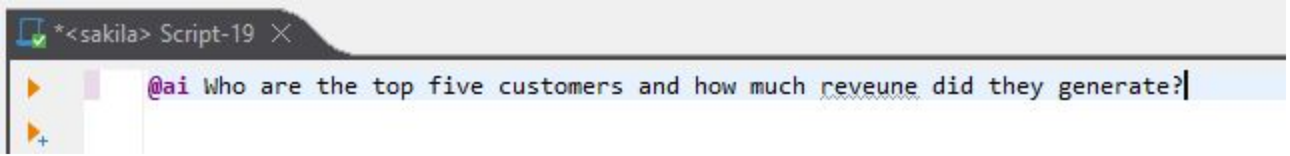
Scope settings

☐ Send only relevant objects to AI (Experimental)

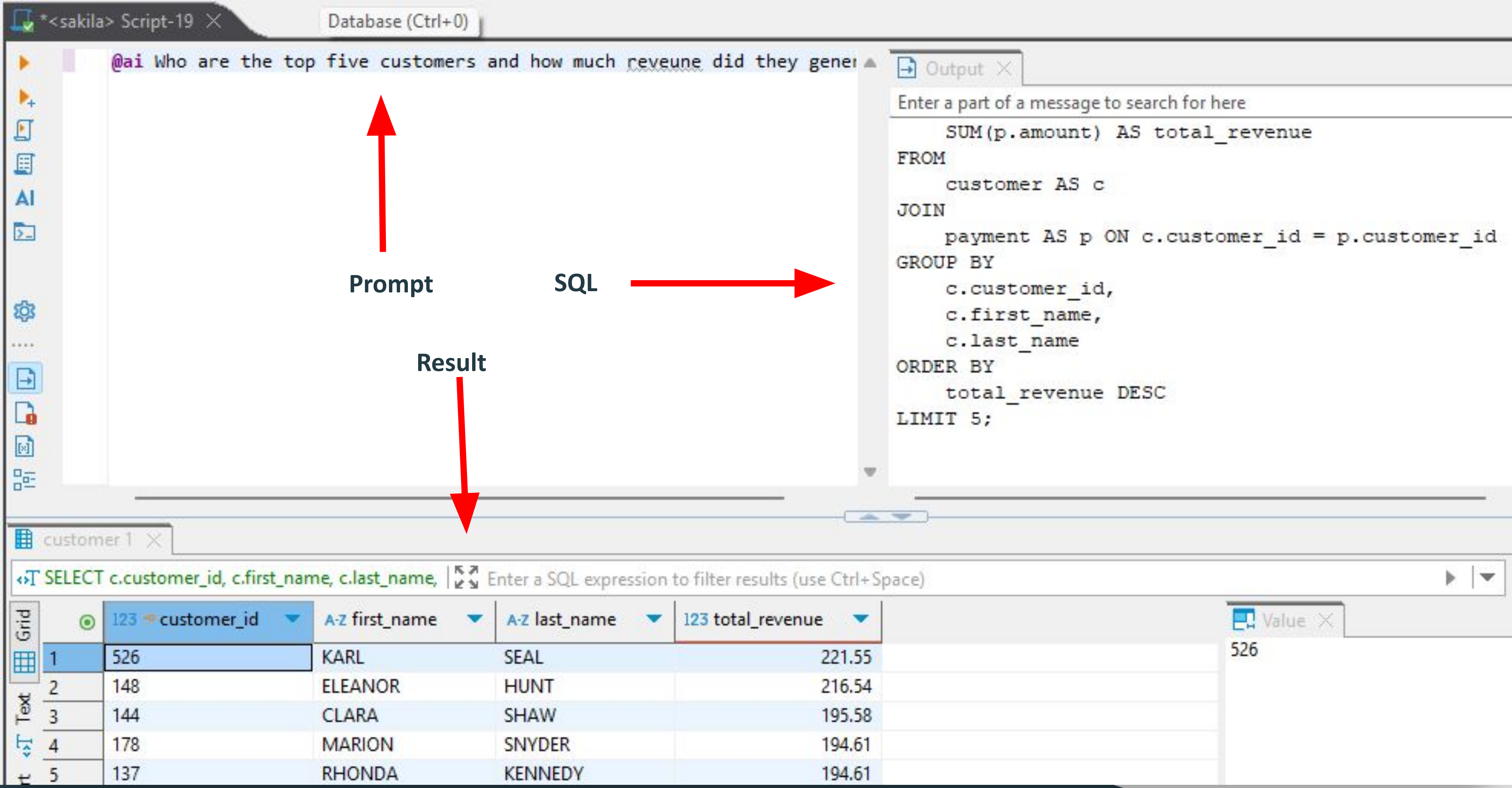
You control
Your data



Do you confirm AI usage for 'sakila'?



```
@ai Who are the top five customers and how much reveune did they generate?
```



```
@ai Who are the top five customers and how much reveune did they gener
```

Prompt

SQL

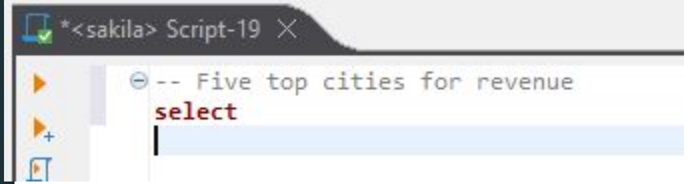
```
    SUM(p.amount) AS total_revenue
FROM
    customer AS c
JOIN
    payment AS p ON c.customer_id = p.customer_id
GROUP BY
    c.customer_id,
    c.first_name,
    c.last_name
ORDER BY
    total_revenue DESC
LIMIT 5;
```

Result

```
SELECT c.customer_id, c.first_name, c.last_name,
```

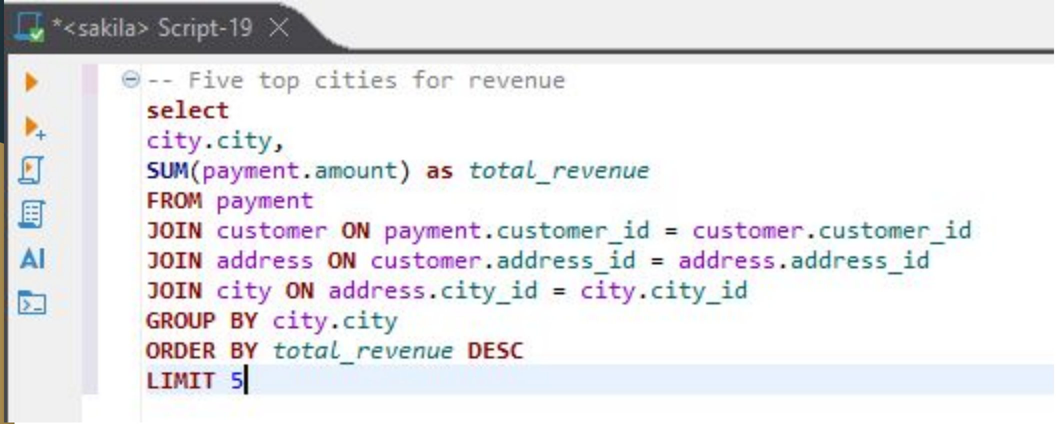
	customer_id	first_name	last_name	total_revenue
1	526	KARL	SEAL	221.55
2	148	ELEANOR	HUNT	216.54
3	144	CLARA	SHAW	195.58
4	178	MARION	SNYDER	194.61
5	137	RHONDA	KENNEDY	194.61

Describe the SQL you want

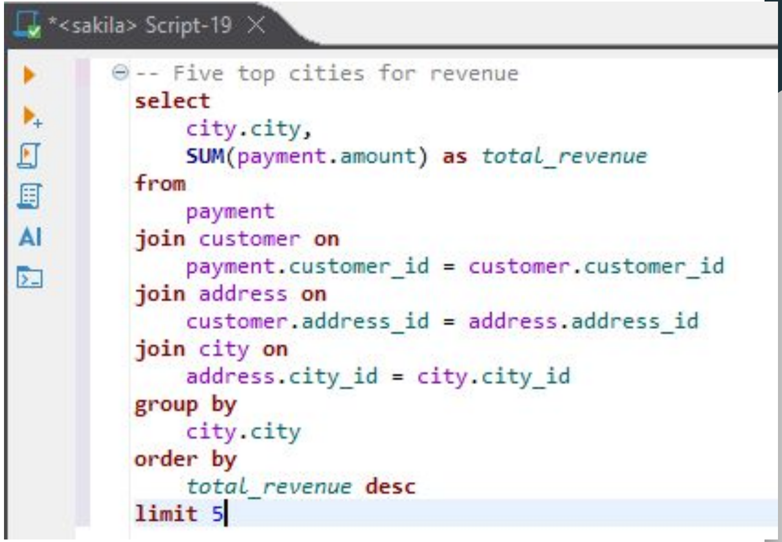


```
-- Five top cities for revenue
select

```



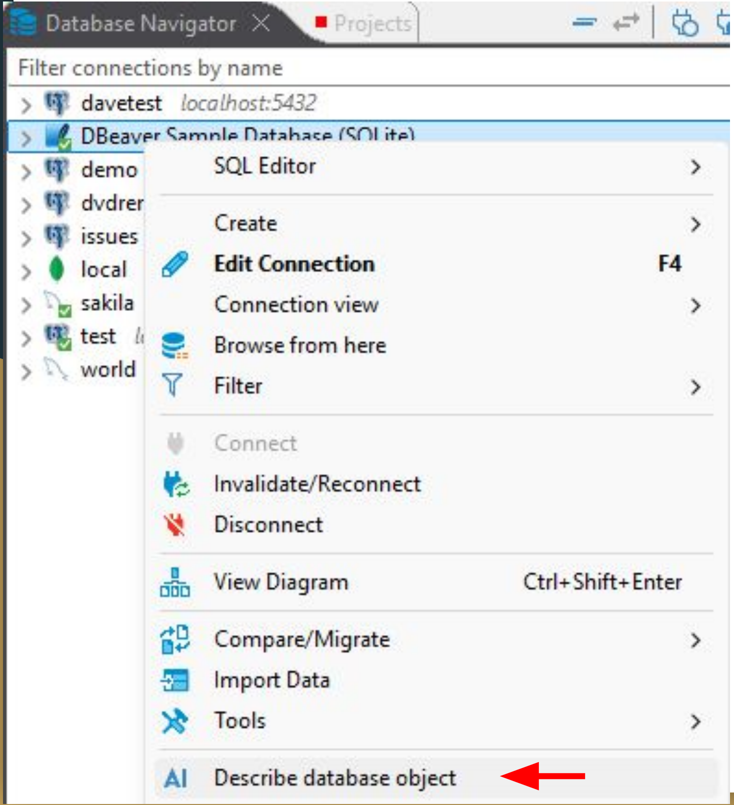
```
-- Five top cities for revenue
select
city.city,
SUM(payment.amount) as total_revenue
FROM payment
JOIN customer ON payment.customer_id = customer.customer_id
JOIN address ON customer.address_id = address.address_id
JOIN city ON address.city_id = city.city_id
GROUP BY city.city
ORDER BY total_revenue DESC
LIMIT 5
```

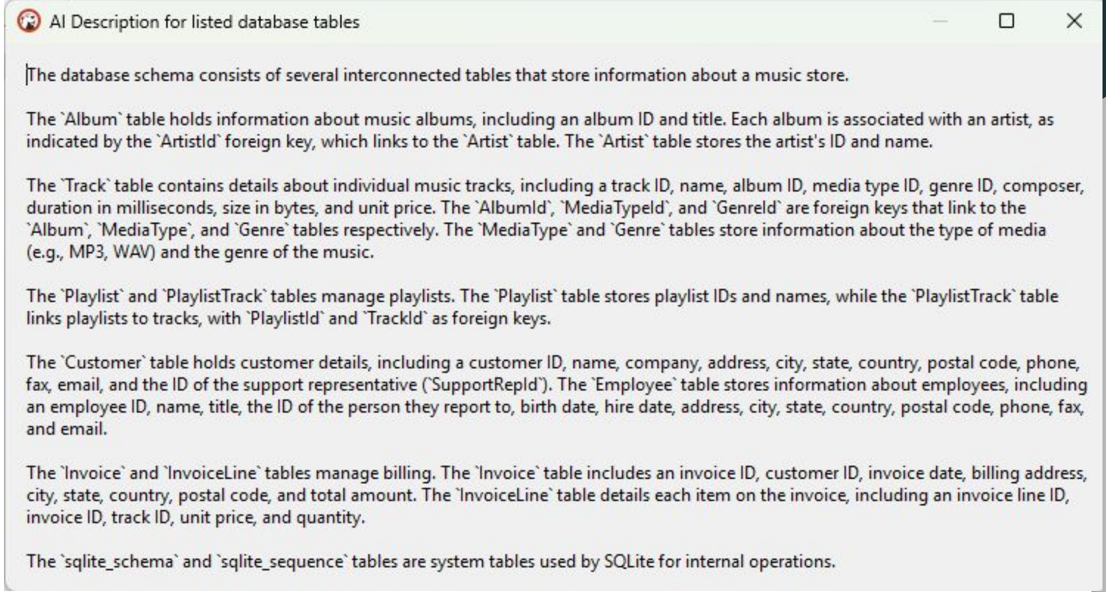


```
-- Five top cities for revenue
select
    city.city,
    SUM(payment.amount) as total_revenue
from
    payment
join customer on
    payment.customer_id = customer.customer_id
join address on
    customer.address_id = address.address_id
join city on
    address.city_id = city.city_id
group by
    city.city
order by
    total_revenue desc
limit 5
```

You can describe what you want, hit CTRL+ALT+→, and generate SQL

Explanation of ‘new’ database





The database schema consists of several interconnected tables that store information about a music store.

The `Album` table holds information about music albums, including an album ID and title. Each album is associated with an artist, as indicated by the `ArtistId` foreign key, which links to the `Artist` table. The `Artist` table stores the artist's ID and name.

The `Track` table contains details about individual music tracks, including a track ID, name, album ID, media type ID, genre ID, composer, duration in milliseconds, size in bytes, and unit price. The `AlbumId`, `MediaTypeId`, and `GenreId` are foreign keys that link to the `Album`, `MediaType`, and `Genre` tables respectively. The `MediaType` and `Genre` tables store information about the type of media (e.g., MP3, WAV) and the genre of the music.

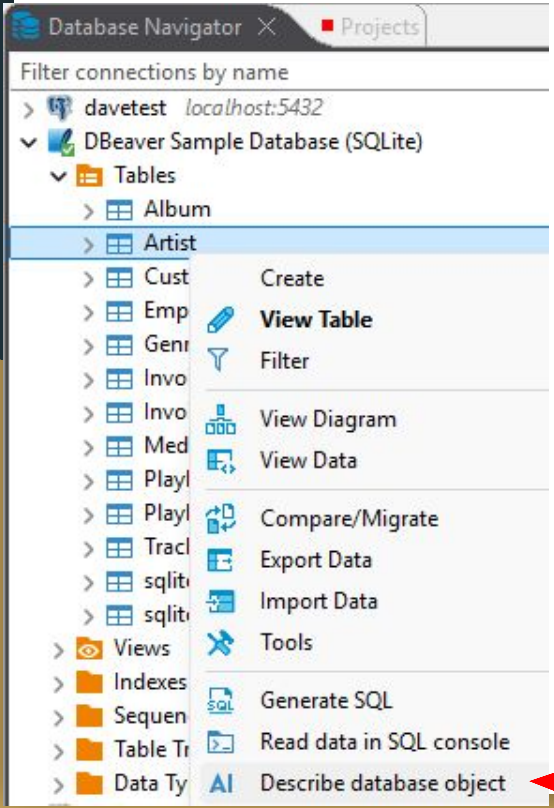
The `Playlist` and `PlaylistTrack` tables manage playlists. The `Playlist` table stores playlist IDs and names, while the `PlaylistTrack` table links playlists to tracks, with `PlaylistId` and `TrackId` as foreign keys.

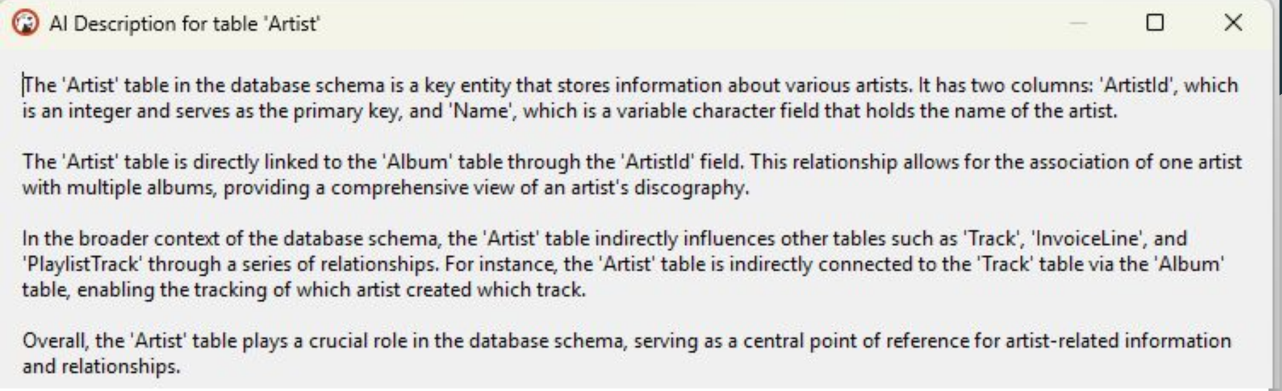
The `Customer` table holds customer details, including a customer ID, name, company, address, city, state, country, postal code, phone, fax, email, and the ID of the support representative (`SupportRepId`). The `Employee` table stores information about employees, including an employee ID, name, title, the ID of the person they report to, birth date, hire date, address, city, state, country, postal code, phone, fax, and email.

The `Invoice` and `InvoiceLine` tables manage billing. The `Invoice` table includes an invoice ID, customer ID, invoice date, billing address, city, state, country, postal code, and total amount. The `InvoiceLine` table details each item on the invoice, including an invoice line ID, invoice ID, track ID, unit price, and quantity.

The `sqlite_schema` and `sqlite_sequence` tables are system tables used by SQLite for internal operations.

Explanation of a table





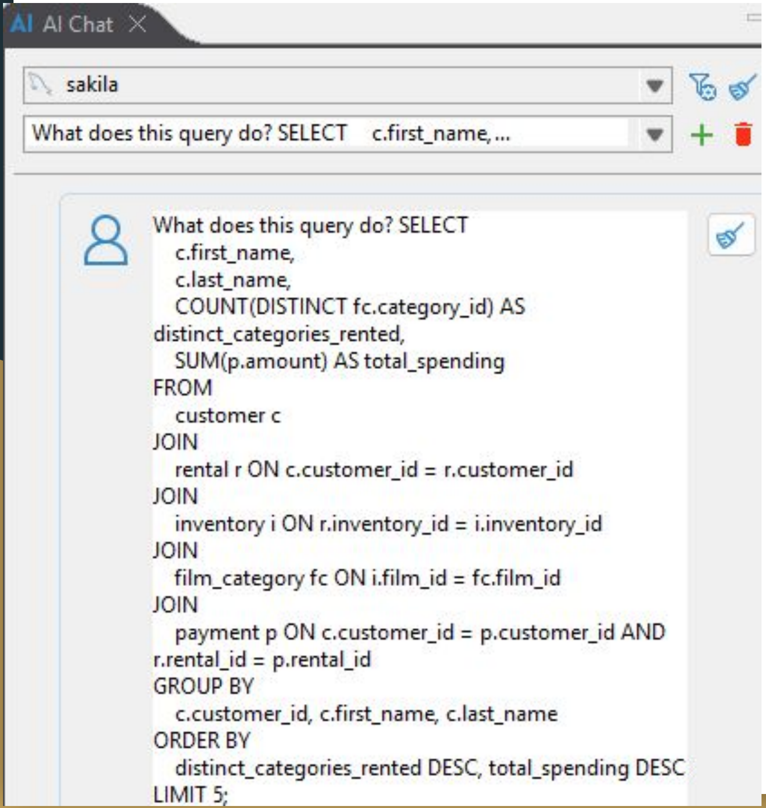
The 'Artist' table in the database schema is a key entity that stores information about various artists. It has two columns: 'ArtistId', which is an integer and serves as the primary key, and 'Name', which is a variable character field that holds the name of the artist.

The 'Artist' table is directly linked to the 'Album' table through the 'ArtistId' field. This relationship allows for the association of one artist with multiple albums, providing a comprehensive view of an artist's discography.

In the broader context of the database schema, the 'Artist' table indirectly influences other tables such as 'Track', 'InvoiceLine', and 'PlaylistTrack' through a series of relationships. For instance, the 'Artist' table is indirectly connected to the 'Track' table via the 'Album' table, enabling the tracking of which artist created which track.

Overall, the 'Artist' table plays a crucial role in the database schema, serving as a central point of reference for artist-related information and relationships.

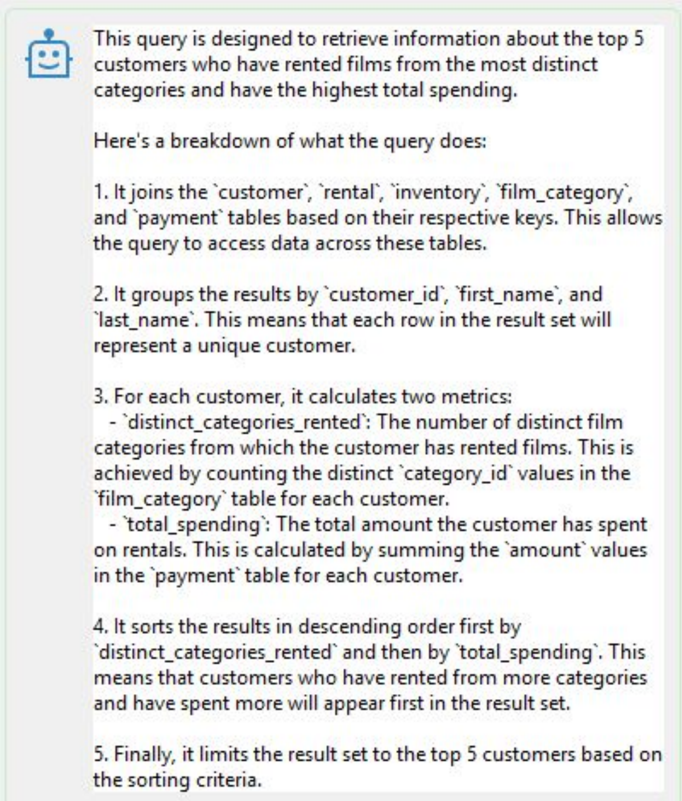
What if you have a question about a query?



sakila

What does this query do? SELECT   c.first_name, ...

```
What does this query do? SELECT
    c.first_name,
    c.last_name,
    COUNT(DISTINCT fc.category_id) AS
distinct_categories_rented,
    SUM(p.amount) AS total_spending
FROM
    customer c
JOIN
    rental r ON c.customer_id = r.customer_id
JOIN
    inventory i ON r.inventory_id = i.inventory_id
JOIN
    film_category fc ON i.film_id = fc.film_id
JOIN
    payment p ON c.customer_id = p.customer_id AND
r.rental_id = p.rental_id
GROUP BY
    c.customer_id, c.first_name, c.last_name
ORDER BY
    distinct_categories_rented DESC, total_spending DESC
LIMIT 5;
```



This query is designed to retrieve information about the top 5 customers who have rented films from the most distinct categories and have the highest total spending.

Here's a breakdown of what the query does:

1. It joins the `customer`, `rental`, `inventory`, `film_category`, and `payment` tables based on their respective keys. This allows the query to access data across these tables.

2. It groups the results by `customer_id`, `first_name`, and `last_name`. This means that each row in the result set will represent a unique customer.

3. For each customer, it calculates two metrics:
  - `distinct_categories_rented`: The number of distinct film categories from which the customer has rented films. This is achieved by counting the distinct `category_id` values in the `film_category` table for each customer.
  - `total_spending`: The total amount the customer has spent on rentals. This is calculated by summing the `amount` values in the `payment` table for each customer.

4. It sorts the results in descending order first by `distinct_categories_rented` and then by `total_spending`. This means that customers who have rented from more categories and have spent more will appear first in the result set.

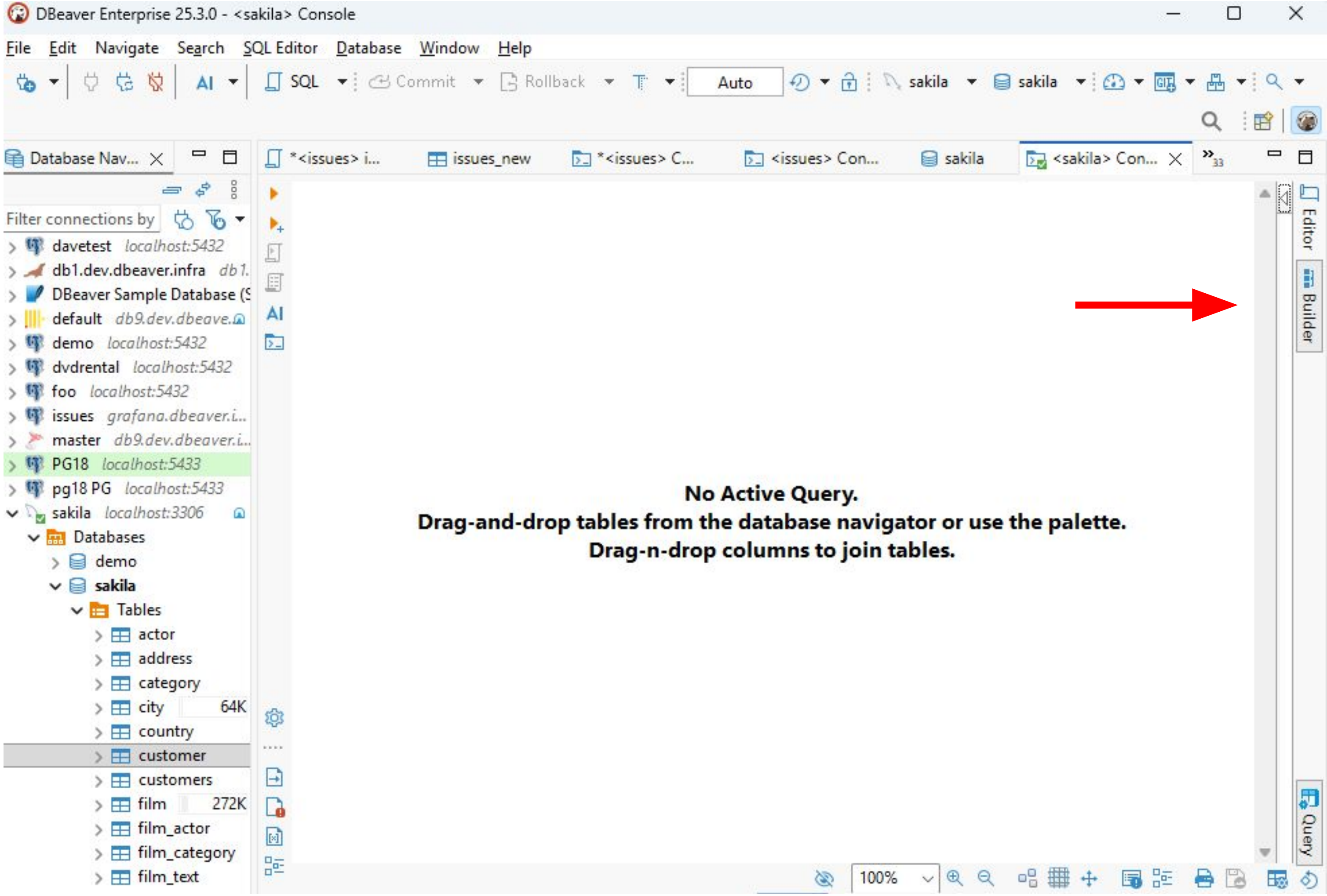
5. Finally, it limits the result set to the top 5 customers based on the sorting criteria.

Augment queries on the fly

Prompt: Please list by state and list their revenue

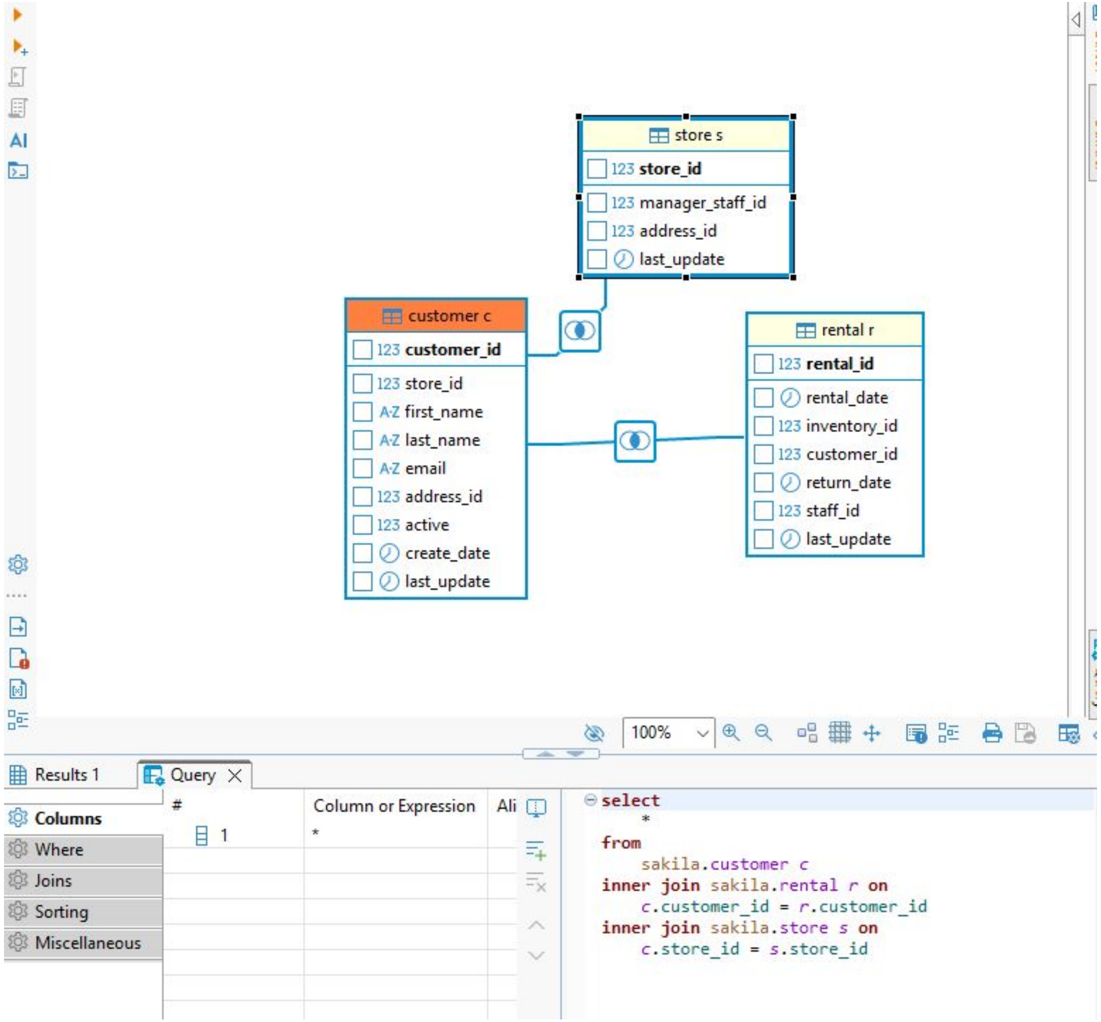
```sql
SELECT
   c.customer_id,
   c.first_name,
   c.last_name,
   COUNT(r.rental_id) AS rental_count
FROM
   customer AS c
JOIN
   address AS a ON c.address_id = a.address_id
JOIN
   city AS ci ON a.city_id = ci.city_id
JOIN
   country AS co ON ci.country_id = co.country_id
JOIN
   rental AS r ON c.customer_id = r.customer_id
WHERE
   co.country = 'United States'
GROUP BY
   c.customer_id
ORDER BY
   rental_count DESC
LIMIT 10;
```

```sql
SELECT
   co.country,
   ci.city AS state,
   c.first_name,
   c.last_name,
   SUM(p.amount) AS revenue
FROM
   customer AS c
JOIN
   address AS a ON c.address_id = a.address_id
JOIN
   city AS ci ON a.city_id = ci.city_id
JOIN
   country AS co ON ci.country_id = co.country_id
JOIN
   payment AS p ON c.customer_id = p.customer_id
WHERE
   co.country = 'United States'
GROUP BY
   co.country, ci.city, c.customer_id
ORDER BY
   revenue DESC;
```



No Active Query.
Drag-and-drop tables from the database navigator or use the palette.
Drag-n-drop columns to join tables.

Drag tables on to the palette



```
select
    *
from
    sakila.customer c
inner join sakila.rental r on
    c.customer_id = r.customer_id
inner join sakila.store s on
    c.store_id = s.store_id
```

ase detail the steps needed to move the
World database from MySQL and also
show any needed sql

To move a database from MySQL to PostgreSQL, you can follow these general steps:

1. Export the MySQL database to a SQL dump file.
2. Convert the MySQL SQL dump file to a PostgreSQL-compatible format.
3. Import the converted SQL dump file into PostgreSQL.

Here's how you can do it:

1. Export the MySQL database to a SQL dump file:

```
mysqldump -u mysql_username -p mysql_database > mysql_dump.sql
```

Replace `mysql_username` and `mysql_database` with your actual MySQL username and database name.

2. Convert the MySQL SQL dump file to a PostgreSQL-compatible format. There's no built-in tool in MySQL or PostgreSQL that can do this, because the SQL dialects and features of MySQL and PostgreSQL are different. You'll need to use a third-party tool like pgloader or manually edit the SQL dump file.

If you're using pgloader, you can use a command like this:

```
pgloader mysql://mysql_username@localhost/mysql_database postgresql://postgres_username@localhost/postgres_database
```

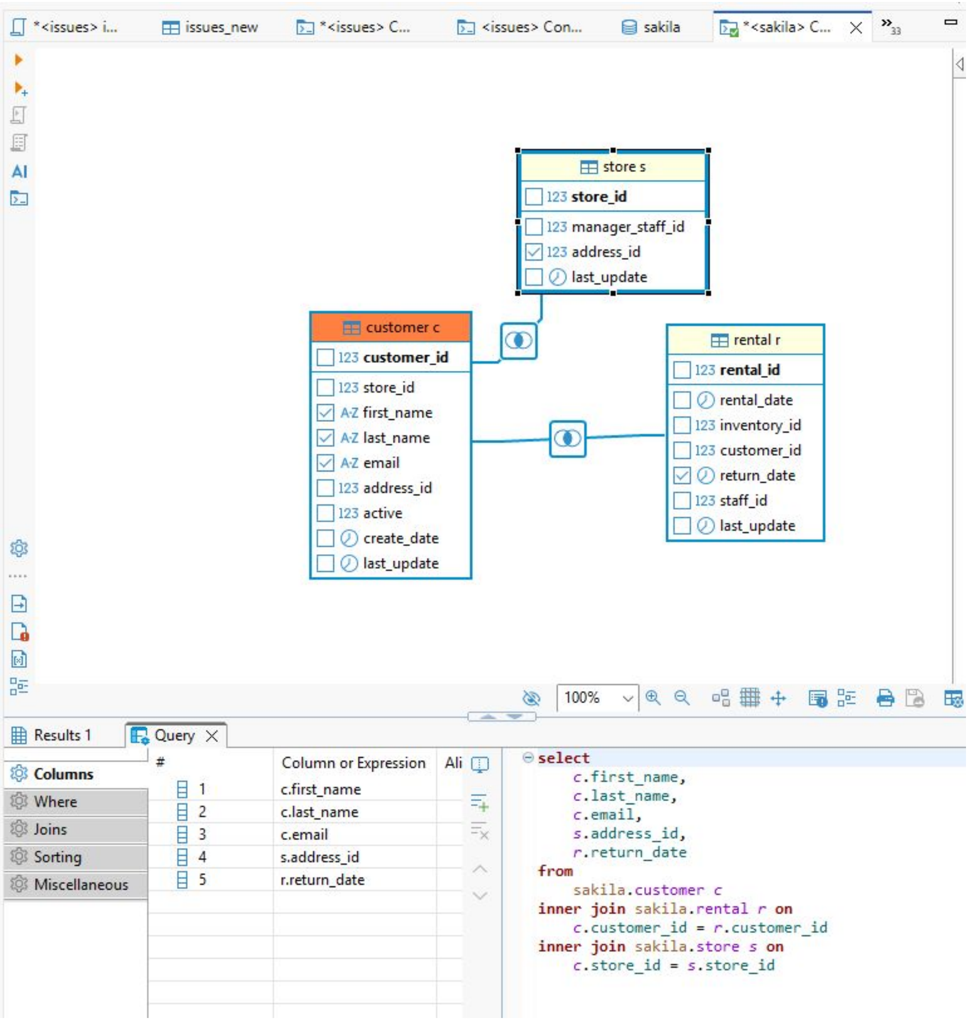
Replace `mysql_username`, `mysql_database`, `postgres_username`, and `postgres_database` with your actual MySQL and PostgreSQL usernames and database names.

3. If you manually edited the SQL dump file, you can import it into PostgreSQL with a command like this:

```
psql -U postgres_username -d postgres_database < postgres_dump.sql
```

Select desired columns

Change join type



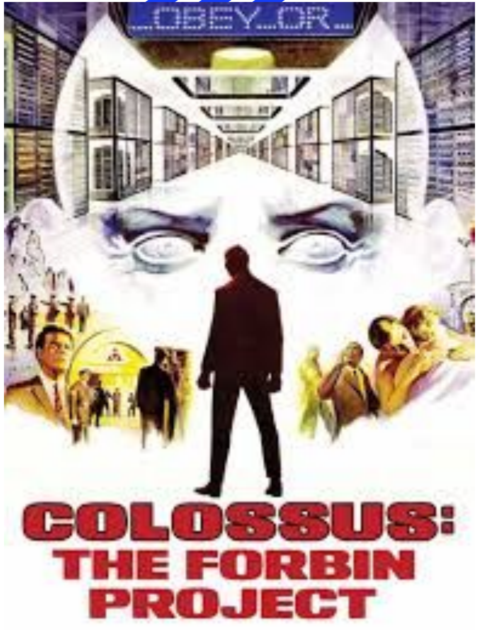
store s
- ☐ 123 **store_id**
- ☐ 123 manager_staff_id
- ☑ 123 address_id
- ☐ last_update

customer c
- ☐ 123 **customer_id**
- ☐ 123 store_id
- ☑ A-Z first_name
- ☑ A-Z last_name
- ☑ A-Z email
- ☐ 123 address_id
- ☐ 123 active
- ☐ create_date
- ☐ last_update

rental r
- ☐ 123 **rental_id**
- ☐ rental_date
- ☐ 123 inventory_id
- ☐ 123 customer_id
- ☑ return_date
- ☐ 123 staff_id
- ☐ last_update

#	Column or Expression	Ali
1	c.first_name	
2	c.last_name	
3	c.email	
4	s.address_id	
5	r.return_date	

```
select
    c.first_name,
    c.last_name,
    c.email,
    s.address_id,
    r.return_date
from
    sakila.customer c
inner join sakila.rental r on
    c.customer_id = r.customer_id
inner join sakila.store s on
    c.store_id = s.store_id
```



When an AI Goes Wrong

How do I use json_table() with Postgresql18?

-ChatGPT

As of **PostgreSQL 18**, there is **no native `JSON_TABLE()` function** like you might know from Oracle Database or MySQL.

PostgreSQL still relies on its own powerful JSON/JSONB functions to achieve the same result.

- what is the latest version of PostgreSQL that you have been trained on?
- My training includes PostgreSQL through version 16 (covering 16 and its 16.x maintenance releases up to my knowledge cutoff of 06/2024).

Do You Need An AI Assistant?

Yes!

What you get

- Write quick and dirty queries, or complex queries
- In English, French, German, Spanish, Basque, or …
- It can explain queries or data structures for you
- Can augment queries faster than you.
- Do as much or as little coding as you desire!
- Model Context Protocol (MCP) will make AI ubiqudos

?

Thank you for your attention!

DBeaver.com

David.Stokes@DBeaver.com
https://www.linkedin.com/in/davidmstokes/

Slide deck @
https://speakerdeck.com/stoker/