

Do You Need An AI Assistant With MySQL?



MySQL Track - Room 105 on Friday, March 6, 2026

Do You Need An AI Assistant With MySQL? Dave Stokes / Community Manager / DBever	10:00 to 11:00
Vitess for Newbies: Scaling MySQL the YouTube Way Igor Donchovski / Principal Consultant	11:15 to 12:15
The Hidden Lives of Temp Tables: Unraveling MySQL Internal Management Gabriel Ciciliani / Database Consultant / Releem	12:30 to 13:30
Are you ready to leave MySQL 8.0 behind? Matthias Crauwels / Enterprise Customer Engineer / PlanetScale	14:30 to 15:30
REST Assured: Serving Up MySQL REST Service with Node—No SQL Required! Scott Stroz / MySQL Developer Advocate / Oracle	15:45 to 16:45
Over 25 years in Education, promoting Linux, Open source, and MySQL Phillip Wacławski / Computer Information Systems Faculty / Mesa Community College	17:00 to 18:00



Do You Need An AI Assistant With MySQL?

The presentation **will take place in Room 105 on Friday, March 6, 2026 - 10:00 to 11:00**

Artificial Intelligence is an overhyped distraction, except for what it can do for you when using MySQL. AI is strong in the pattern matching area, which means it is great with SQL syntax and examining DDL metadata.

This session will cover basic prompting, using AI to repair or augment existing queries, and developing new schemas.

And you can write queries in English (or French, or German, or Italian) instead of SQL, saving you a lot of time determining which tables need to be joined where.

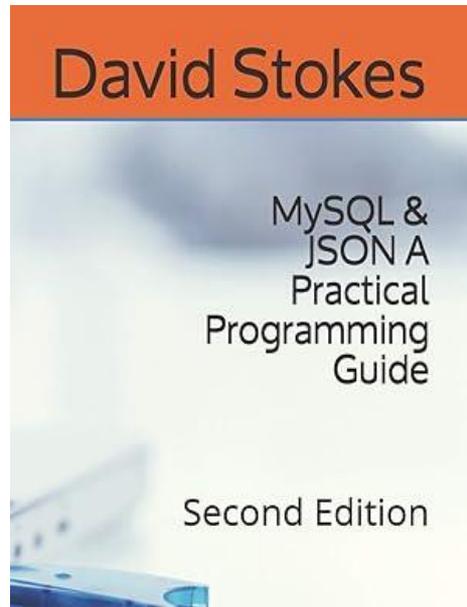
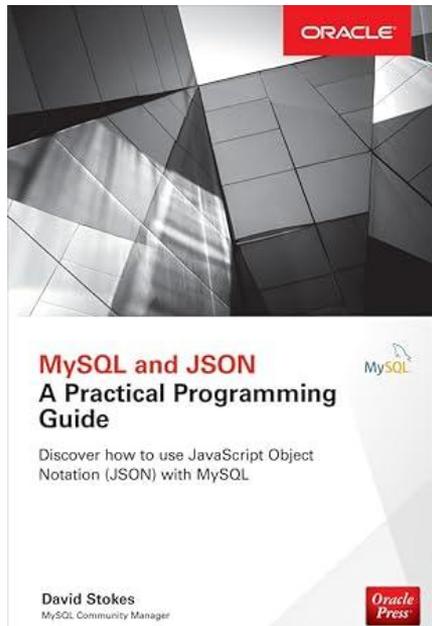
SO, yes, you do need an AI Assistant to get the maximum out of your MySQL instances, and this session will show you how to do it.

Who Am I?

Dave Stokes

Community Manager at DBeaver

Long time Open Source database
proponent



Example 1 -

Ask four LLMs the same question

- o Simple Prompt asking an AI directly to 'do something'. We mention a reference database but are not supplying more details about it.
- o ***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 ✕ Collapse ≡ Wrap 📄 Copy  
  
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 ✕ Collapse ≡ Wrap 📄 Copy  
  
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:

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


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
```

Copy Edit

```
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

- o The previous example used a 'well known database' so the LLM 'knew' about the schema and the data. But not from us.
- o 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 MySQL 8.0 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.

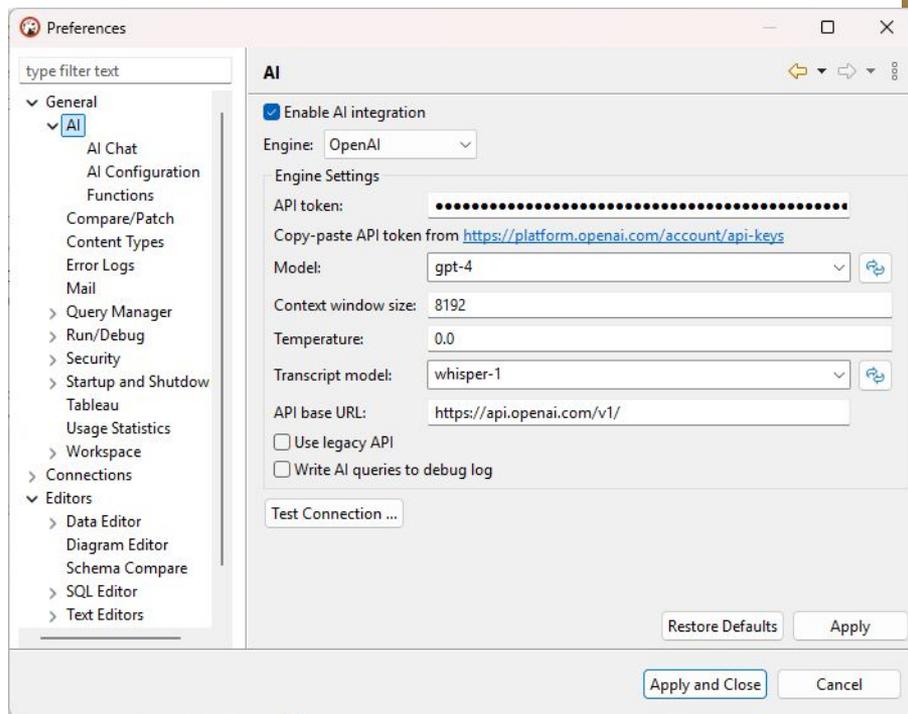
Some personal goals:

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.



Come by the DBeaver booth for a demo

Example:

DBeaver AI Explain and

The screenshot shows the DBeaver SQL editor with a query that has an error. The query is:

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

The error message is: **SQL Error [1054] [42S22]: Unknown column 'name' in 'field list'**

AI Explain and Fix

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

```
1 select
2     customer_id,
3     name,
4     last_name
5 from
6     customers_gpt
```

The error message is:

```
1 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:

```
1 select
2     customer_id,
3     first_name,
4     last_name
5 from
6     customers_gpt
```

The screenshot shows the DBeaver SQL editor with the corrected query:

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

The results are displayed in a table:

	AZ customer_id	AZ first_name	AZ last_name
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)



Grok:

```
-- Create the customers table with UUID primary key
-- and partitioning by birth YEAR
```

```
CREATE TABLE customers (
  id CHAR(36) NOT NULL,
  first_name VARCHAR(100) NOT NULL,
  last_name VARCHAR(100) NOT NULL,
  address VARCHAR(255) NOT NULL,
  birthdate DATE NOT NULL,

  PRIMARY KEY (id, birthdate),

  -- Optional: index if you frequently search by name
  INDEX idx_name (last_name, first_name)
)
```

```
PARTITION BY RANGE (YEAR(birthdate)) (
  PARTITION p_before_1970 VALUES LESS THAN (1970),
  PARTITION p1970 VALUES LESS THAN (1980),
  PARTITION p1980 VALUES LESS THAN (1990),
  PARTITION p1990 VALUES LESS THAN (2000),
  PARTITION p2000 VALUES LESS THAN (2010),
  PARTITION p2010 VALUES LESS THAN (2020),
  PARTITION p2020 VALUES LESS THAN (2030),
  PARTITION p_future VALUES LESS THAN MAXVALUE
)
```

```
ENGINE=InnoDB
ROW_FORMAT=DYNAMIC
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

```
INSERT INTO customers (id, first_name, last_name, address,
birthdate) VALUES
('550e8400-e29b-41d4-a716-446655440000', 'James', 'Smith',
'123 Maple Street, Austin, TX 78701', '1982-05-14'),
('550e8400-e29b-41d4-a716-446655440001', 'Maria', 'Garcia',
'456 Oak Lane, Dallas, TX 75219', '1995-11-03'),
('550e8400-e29b-41d4-a716-446655440002', 'Robert', 'Johnson',
'789 Pine Road, Houston, TX 77002', '1968-02-27'),
('550e8400-e29b-41d4-a716-446655440003', 'Sarah', 'Williams',
'101 Elm Court, San Antonio, TX 78205', '2001-08-19'),
('550e8400-e29b-41d4-a716-446655440004', 'David', 'Brown',
'202 Cedar Drive, Fort Worth, TX 76102', '1975-12-30'),
('550e8400-e29b-41d4-a716-446655440005', 'Emily', 'Jones',
'333 Birch Blvd, Plano, TX 75024', '1998-04-08'),
('550e8400-e29b-41d4-a716-446655440006', 'Michael', 'Davis',
'555 Willow Way, Arlington, TX 76013', '1987-09-22'),
('550e8400-e29b-41d4-a716-446655440007', 'Sophia',
'Rodriguez', '777 Aspen Place, Frisco, TX 75034',
'2005-01-15'),
('550e8400-e29b-41d4-a716-446655440008', 'William', 'Martinez',
'888 Spruce Street, Irving, TX 75061', '1964-07-04'),
('550e8400-e29b-41d4-a716-446655440009', 'Olivia',
'Hernandez', '999 Laurel Lane, McKinney, TX 75070',
'1992-03-11'),
('550e8400-e29b-41d4-a716-446655440010', 'Ethan', 'Lopez',
'1240 Magnolia Ave, Lewisville, TX 75067', '2012-06-25'),
('550e8400-e29b-41d4-a716-446655440011', 'Isabella', 'Gonzalez',
'1352 Sycamore Ct, Carrollton, TX 75006', '1979-10-17');

-- Alternative insert using built-in UUID()
INSERT INTO customers (id, first_name, last_name, address,
birthdate) VALUES
(UUID(), 'Liam', 'Taylor', '742 Evergreen Terrace,
Springfield, TX', '1994-09-12'),
(UUID(), 'Ava', 'Anderson', '1592 Riverside Drive, Denton,
TX', '2008-12-01');
```

Grok:

```
-- Create the customers table with UUID primary key
-- and partitioning by birth YEAR
CREATE TABLE customers (
  id CHAR(36) NOT NULL,
  first_name VARCHAR(100) NOT NULL,
  last_name VARCHAR(100) NOT NULL,
  address VARCHAR(255) NOT NULL,
  birthdate DATE NOT NULL,

  PRIMARY KEY (id, birthdate),

  -- Optional: index if you frequently search by name
  INDEX idx_name (last_name, first_name)
)
PARTITION BY RANGE (YEAR(birthdate)) (
  PARTITION p_before_1970 VALUES LESS THAN (1970),
  PARTITION p1970 VALUES LESS THAN (1980),
  PARTITION p1980 VALUES LESS THAN (1990),
  PARTITION p1990 VALUES LESS THAN (2000),
  PARTITION p2000 VALUES LESS THAN (2010),
  PARTITION p2010 VALUES LESS THAN (2020),
  PARTITION p2020 VALUES LESS THAN (2030),
  PARTITION p_future VALUES LESS THAN MAXVALUE
)
ENGINE=InnoDB
ROW_FORMAT=DYNAMIC
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

```
CREATE TABLE customers_grok (a syntax error near the
  id CHAR(36) NOT NULL,NAMIC DEFAULT
  first_name VARCHAR(100) NOT NULL,pi part of your
  last_name VARCHAR(100) NOT NULL,
  address VARCHAR(255) NOT NULL,CHARSET, and COLLATE
  birthdate DATE NOT NULL,

  PRIMARY KEY (id, birthdate),

  -- Optional: index if you frequently search by name
  INDEX idx_name (last_name, first_name)
)
ENGINE=InnoDB
ROW_FORMAT=DYNAMIC
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci
PARTITION BY RANGE (YEAR(birthdate)) (
  PARTITION p_before_1970 VALUES LESS THAN (1970),
  PARTITION p1970 VALUES LESS THAN (1980),
  PARTITION p1980 VALUES LESS THAN (1990),
  PARTITION p1990 VALUES LESS THAN (2000),
  PARTITION p2000 VALUES LESS THAN (2010),
  PARTITION p2010 VALUES LESS THAN (2020),
  PARTITION p2020 VALUES LESS THAN (2030),
  PARTITION p_future VALUES LESS THAN MAXVALUE
);
```

Google Gemini:

```
CREATE TABLE customers (  
    id BINARY(16) PRIMARY KEY,  
    first_name VARCHAR(50) NOT NULL,  
    last_name VARCHAR(50) NOT NULL,  
    address VARCHAR(255) NOT NULL,  
    birthdate DATE NOT NULL  
)  
PARTITION BY RANGE (YEAR(birthdate)) (  
    PARTITION p2010 VALUES LESS THAN (2011),  
    PARTITION p2011 VALUES LESS THAN (2012),  
    PARTITION p2012 VALUES LESS THAN (2013),  
    PARTITION p2013 VALUES LESS THAN (2014),  
    PARTITION p2014 VALUES LESS THAN (2015),  
    PARTITION p2015 VALUES LESS THAN (2016),  
    PARTITION p2016 VALUES LESS THAN (2017),  
    PARTITION p2017 VALUES LESS THAN (2018),  
    PARTITION p2018 VALUES LESS THAN (2019),  
    PARTITION p2019 VALUES LESS THAN (2020),  
    PARTITION p2020 VALUES LESS THAN (2021),  
    PARTITION p2021 VALUES LESS THAN (2022),  
    PARTITION p2022 VALUES LESS THAN (2023)  
);
```

```
INSERT INTO customers (id, first_name, last_name, address,  
birthdate) VALUES  
(UNHEX(REPLACE(UUID(), '-', '')), 'John', 'Doe', '123 Main St,  
Anytown, USA', '1980-05-15'),  
(UNHEX(REPLACE(UUID(), '-', '')), 'Jane', 'Smith', '456 Maple Rd,  
Othertown, USA', '1992-11-22'),  
(UNHEX(REPLACE(UUID(), '-', '')), 'Emma', 'Brown', '789 Oak St,  
Sometown, USA', '1985-02-10'),  
(UNHEX(REPLACE(UUID(), '-', '')), 'James', 'Johnson', '321 Pine  
St, Townsville, USA', '1975-12-05'),  
(UNHEX(REPLACE(UUID(), '-', '')), 'Olivia', 'Davis', '654 Cedar  
St, Villagetown, USA', '1990-09-30'),  
(UNHEX(REPLACE(UUID(), '-', '')), 'William', 'Garcia', '987 Birch  
St, Cityville, USA', '1988-04-25'),  
(UNHEX(REPLACE(UUID(), '-', '')), 'Sophia', 'Martinez', '135  
Spruce St, Hamlet, USA', '1995-07-18'),  
(UNHEX(REPLACE(UUID(), '-', '')), 'Liam', 'Hernandez', '246 Walnut  
St, Metropolis, USA', '1999-01-01'),  
(UNHEX(REPLACE(UUID(), '-', '')), 'Isabella', 'Lopez', '357  
Redwood St, Stateburg, USA', '1982-10-12'),  
(UNHEX(REPLACE(UUID(), '-', '')), 'Noah', 'Gonzalez', '468 Dogwood  
St, Hilltown, USA', '1978-03-08');
```

Google Gemini:

```
CREATE TABLE customers (  
  id BINARY(16) PRIMARY KEY,  
  first_name VARCHAR(50) NOT NULL,  
  last_name VARCHAR(50) NOT NULL,  
  address VARCHAR(255) NOT NULL,  
  birthdate DATE NOT NULL  
)  
PARTITION BY RANGE (YEAR(birthdate)) (  
  PARTITION p2010 VALUES LESS THAN (2011),  
  PARTITION p2011 VALUES LESS THAN (2012),  
  PARTITION p2012 VALUES LESS THAN (2013),  
  PARTITION p2013 VALUES LESS THAN (2014),  
  PARTITION p2014 VALUES LESS THAN (2015),  
  PARTITION p2015 VALUES LESS THAN (2016),  
  PARTITION p2016 VALUES LESS THAN (2017),  
  PARTITION p2017 VALUES LESS THAN (2018),  
  PARTITION p2018 VALUES LESS THAN (2019),  
  PARTITION p2019 VALUES LESS THAN (2020),  
  PARTITION p2020 VALUES LESS THAN (2021),  
  PARTITION p2021 VALUES LESS THAN (2022),  
  PARTITION p2022 VALUES LESS THAN (2023)  
);
```

Error: The error message is indicating that the partitioning function is based on the `birthdate` column, but this column is not included in the primary key. In MySQL, when a table is partitioned, the columns used in the partitioning expression must be part of every unique key (including the primary key) of the table.

To fix this, you need to include the `birthdate` column in the primary key.

```
      birthdate DATE NOT NULL,  
      PRIMARY KEY (id, birthdate)  
)  
PARTITION BY RANGE (YEAR(birthdate)) (  
  PARTITION p2010 VALUES LESS THAN (2011),  
  PARTITION p2011 VALUES LESS THAN (2012),  
  PARTITION p2012 VALUES LESS THAN (2013),  
  PARTITION p2013 VALUES LESS THAN (2014),  
  PARTITION p2014 VALUES LESS THAN (2015),  
  PARTITION p2015 VALUES LESS THAN (2016),  
  PARTITION p2016 VALUES LESS THAN (2017),  
  PARTITION p2017 VALUES LESS THAN (2018),  
  PARTITION p2018 VALUES LESS THAN (2019),  
  PARTITION p2019 VALUES LESS THAN (2020),  
  PARTITION p2020 VALUES LESS THAN (2021),  
  PARTITION p2021 VALUES LESS THAN (2022),  
  PARTITION p2022 VALUES LESS THAN (2023)  
)
```

Ollama (Gemma3.4b):

```
-- Create the Customers table with UUID primary key and
yearly partitioning
CREATE TABLE Customers (
    customer_id UUID PRIMARY KEY DEFAULT (UUID()),
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    address VARCHAR(255),
    birthdate DATE NOT NULL,
    birth_year INT NOT NULL, -- For partitioning
    INDEX idx_birth_year (birth_year) -- For partitioning
optimization
)
PARTITION BY YEAR(birthdate) (
    PARTITION p2000 VALUES IN (YEAR(CURDATE()) - 2000,
YEAR(CURDATE()) - 2001), -- 2000-2001
    PARTITION p2005 VALUES IN (YEAR(CURDATE()) - 2005,
YEAR(CURDATE()) - 2005), -- 2005
    PARTITION p2010 VALUES IN (YEAR(CURDATE()) - 2010,
YEAR(CURDATE()) - 2010), -- 2010
    PARTITION p2015 VALUES IN (YEAR(CURDATE()) - 2015,
YEAR(CURDATE()) - 2015), -- 2015
    PARTITION p2020 VALUES IN (YEAR(CURDATE()) - 2020,
YEAR(CURDATE()) - 2020) -- 2020
);
```

```
-- Insert test data
INSERT INTO Customers (first_name, last_name, address,
birthdate, birth_year) VALUES
('Alice', 'Smith', '123 Main St, Anytown', '1998-05-10', 1998),
('Bob', 'Johnson', '456 Oak Ave, Anytown', '2002-11-22', 2002),
('Charlie', 'Brown', '789 Pine Ln, Anytown', '2005-03-15',
2005),
('David', 'Williams', '101 Elm Rd, Anytown', '2010-08-01',
2010),
('Eve', 'Davis', '222 Maple Dr, Anytown', '2015-02-28', 2015),
('Frank', 'Miller', '333 Birch Ct, Anytown', '2018-09-12',
2018),
('Grace', 'Wilson', '444 Cedar Way, Anytown', '2020-04-05',
2020),
('Henry', 'Moore', '555 Walnut St, Anytown', '1999-12-31',
1999),
('Isabella', 'Taylor', '666 Cherry Ln, Anytown', '2003-07-18',
2003),
('Jack', 'Anderson', '777 Spruce Rd, Anytown', '2007-06-10',
2007),
('Kelly', 'Thomas', '888 Poplar Ave, Anytown', '2012-11-25',
2012),
('Liam', 'Jackson', '999 Willow St, Anytown', '2017-03-08',
2017);
```

Ollama (Gemma3.4b):

```
-- Create the Customers table with UUID primary key and
yearly partitioning
CREATE TABLE Customers (
    customer_id UUID PRIMARY KEY DEFAULT (UUID()),
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    address VARCHAR(255),
    birthdate DATE NOT NULL,
    birth_year INT NOT NULL, -- For partitioning
    INDEX idx_birth_year (birth_year) -- For partitioning
optimization
)
PARTITION BY YEAR(birthdate) (
    PARTITION p2000 VALUES IN (YEAR(CURDATE()) - 2000,
YEAR(CURDATE()) - 2001), -- 2000-2001
    PARTITION p2005 VALUES IN (YEAR(CURDATE()) - 2005,
YEAR(CURDATE()) - 2005), -- 2005
    PARTITION p2010 VALUES IN (YEAR(CURDATE()) - 2010,
YEAR(CURDATE()) - 2010), -- 2010
    PARTITION p2015 VALUES IN (YEAR(CURDATE()) - 2015,
YEAR(CURDATE()) - 2015), -- 2015
    PARTITION p2020 VALUES IN (YEAR(CURDATE()) - 2020,
YEAR(CURDATE()) - 2020) -- 2020
);
```

ERROR: The error message indicates that there's a problem with the **UUID** data type and its default value. MySQL does not support **UUID** as a data type. Instead, you should use **CHAR(36)** for storing UUIDs. Also, the correct function for generating UUID in MySQL is **UUID()**, not **UUID** and it should not be enclosed in parentheses.

```
    birthdate DATE NOT NULL,
    birth_year INT NOT NULL, -- For partitioning
    PRIMARY KEY (customer_id, birth_year),
    INDEX idx_birth_year (birth_year) -- For partitioning
optimization
)
PARTITION BY RANGE (birth_year) (
    PARTITION p2000 VALUES LESS THAN (2002),
    PARTITION p2005 VALUES LESS THAN (2006),
    PARTITION p2010 VALUES LESS THAN (2011),
    PARTITION p2015 VALUES LESS THAN (2016),
    PARTITION p2020 VALUES LESS THAN (2021)
)
```

Ollama (Gemma3.4b):

```
-- Insert test data
INSERT INTO Customers (first_name, last_name, address, birthdate,
birth_year) VALUES
('Alice', 'Smith', '123 Main St, Anytown', '1998-05-10', 1998),
('Bob', 'Johnson', '456 Oak Ave, Anytown', '2002-11-22', 2002),
('Charlie', 'Brown', '789 Pine Ln, Anytown', '2005-03-15', 2005),
('David', 'Williams', '101 Elm Rd, Anytown', '2010-08-01', 2010),
('Eve', 'Davis', '222 Maple Dr, Anytown', '2015-02-28', 2015),
('Frank', 'Miller', '333 Birch Ct, Anytown', '2018-09-12', 2018),
('Grace', 'Wilson', '444 Cedar Way, Anytown', '2020-04-05', 2020),
('Henry', 'Moore', '555 Walnut St, Anytown', '1999-12-31', 1999),
('Isabella', 'Taylor', '666 Cherry Ln, Anytown', '2003-07-18', 2003),
('Jack', 'Anderson', '777 Spruce Rd, Anytown', '2007-06-10', 2007),
('Kelly', 'Thomas', '888 Poplar Ave, Anytown', '2012-11-25', 2012),
('Liam', 'Jackson', '999 Willow St, Anytown', '2017-03-08', 2017);
```

The error message indicates that the 'customer_id' field in the 'Customers_ollama' table doesn't have a default value and you're trying to insert a new record without providing a value for this field.

However, there's another issue. The table you're trying to insert data into is 'Customers_ollama', but according to the database snapshot, the table's name is 'customers', not 'Customers_ollama'.

Also, the columns you're trying to insert data into ('first_name', 'last_name', 'address', 'birthdate', 'birth_year') do not exist in the 'customers' table. The 'customers' table only has 'customer_id', 'name', and 'email' columns.

Here's a corrected version of your query, assuming you want to insert data into the 'customers' table and 'name' is a combination of 'first_name' and 'last_name':

```
INSERT INTO customers (customer_id, name, email) VALUES
(1, 'Alice Smith', 'alice.smith@example.com'),
(2, 'Bob Johnson', 'bob.johnson@example.com'),
(3, 'Charlie Brown', 'charlie.brown@example.com'),
(4, 'David Williams', 'david.williams@example.com'),
(5, 'Eve Davis', 'eve.davis@example.com'),
(6, 'Frank Miller', 'frank.miller@example.com'),
(7, 'Grace Wilson', 'grace.wilson@example.com'),
(8, 'Henry Moore', 'henry.moore@example.com'),
(9, 'Isabella Taylor', 'isabella.taylor@example.com'),
(10, 'Jack Anderson', 'jack.anderson@example.com'),
(11, 'Kelly Thomas', 'kelly.thomas@example.com'),
(12, 'Liam Jackson', 'liam.jackson@example.com');
```

ChatGPT:

```
CREATE TABLE customers (  
    customer_id CHAR(36) NOT NULL,  
    first_name VARCHAR(50) NOT NULL,  
    last_name VARCHAR(50) NOT NULL,  
    address VARCHAR(255) NOT NULL,  
    birthdate DATE NOT NULL,  
  
    birth_year INT  
        GENERATED ALWAYS AS (YEAR(birthdate))  
STORED,  
  
    PRIMARY KEY (birth_year, customer_id)  
)  
PARTITION BY RANGE (birth_year) (  
    PARTITION p1950s VALUES LESS THAN (1960),  
    PARTITION p1960s VALUES LESS THAN (1970),  
    PARTITION p1970s VALUES LESS THAN (1980),  
    PARTITION p1980s VALUES LESS THAN (1990),  
    PARTITION p1990s VALUES LESS THAN (2000),  
    PARTITION p2000s VALUES LESS THAN (2010),  
    PARTITION p2010s VALUES LESS THAN (2020),  
    PARTITION p_future VALUES LESS THAN MAXVALUE  
);
```

```
INSERT INTO customers (customer_id, first_name, last_name,  
address, birthdate) VALUES  
(UUID(), 'John', 'Smith', '123 Main St, Dallas, TX',  
'1975-04-12'),  
(UUID(), 'Mary', 'Johnson', '456 Oak Ave, Austin, TX',  
'1982-09-30'),  
(UUID(), 'Robert', 'Brown', '789 Pine Rd, Houston, TX',  
'1968-01-05'),  
(UUID(), 'Linda', 'Davis', '321 Maple St, Plano, TX',  
'1991-07-22'),  
(UUID(), 'James', 'Wilson', '654 Cedar Ln, Irving, TX',  
'1959-11-18'),  
(UUID(), 'Patricia', 'Miller', '987 Birch Blvd, Frisco, TX',  
'1970-02-14'),  
(UUID(), 'Michael', 'Taylor', '159 Spruce Ct, McKinney, TX',  
'1988-06-03'),  
(UUID(), 'Barbara', 'Anderson', '753 Willow Way, Denton, TX',  
'2001-12-25'),  
(UUID(), 'William', 'Thomas', '852 Aspen Dr, Garland, TX',  
'1996-05-09'),  
(UUID(), 'Susan', 'Moore', '951 Redwood Pl, Carrollton,  
TX', '2012-08-17');
```

ChatGPT:

```
CREATE TABLE customers (  
    customer_id CHAR(36) NOT NULL,  
    first_name VARCHAR(50) NOT NULL,  
    last_name VARCHAR(50) NOT NULL,  
    address VARCHAR(255) NOT NULL,  
    birthdate DATE NOT NULL,  
  
    birth_year INT  
        GENERATED ALWAYS AS (YEAR(birthdate)) STORED,  
  
    PRIMARY KEY (birth_year, customer_id)  
)  
PARTITION BY RANGE (birth_year) (  
    PARTITION p1950s VALUES LESS THAN (1960),  
    PARTITION p1960s VALUES LESS THAN (1970),  
    PARTITION p1970s VALUES LESS THAN (1980),  
    PARTITION p1980s VALUES LESS THAN (1990),  
    PARTITION p1990s VALUES LESS THAN (2000),  
    PARTITION p2000s VALUES LESS THAN (2010),  
    PARTITION p2010s VALUES LESS THAN (2020),  
    PARTITION p_future VALUES LESS THAN MAXVALUE  
);
```

```
CREATE TABLE customers (  
The error message indicates that there's a syntax error in  
your SQL statement. The provided SQL snippet seems to  
be a part of a CREATE TABLE statement, but the  
beginning of the statement is missing.  
The PRIMARY KEY and PARTITION BY clauses should  
be part of a CREATE TABLE statement. Also, the  
birth_year column is not defined in any of the tables in  
the database snapshot. ALWAYS AS (YEAR(birthdate)) STORED,  
Here's an example of how you might structure the ed (  
CREATE TABLE statement, assuming you want to create  
a new table called customers_partitioned:  
    email VARCHAR(255),  
    birth_year INT,  
    PRIMARY KEY (birth_year, customer_id)  
)  
PARTITION BY RANGE (birth_year) (  
    PARTITION p1950s VALUES LESS THAN (1960),  
    PARTITION p1960s VALUES LESS THAN (1970),  
    PARTITION p1970s VALUES LESS THAN (1980),  
    PARTITION p1980s VALUES LESS THAN (1990),  
    PARTITION p1990s VALUES LESS THAN (2000),  
    PARTITION p2000s VALUES LESS THAN (2010),  
    PARTITION p2010s VALUES LESS THAN (2020),  
    PARTITION p_future VALUES LESS THAN MAXVALUE  
);
```

Approaches to table design

	Grok	Google	Ollama	ChatGPT
id	id CHAR(36) NOT NULL	id BINARY(16) PRIMARY KEY	customer_id UUID PRIMARY KEY DEFAULT (UUID())	customer_id CHAR(36) NOT NULL
names	VARCHAR(100) NOT NULL	VARCHAR(50) NOT NULL	VARCHAR(255) NOT NULL	VARCHAR(50) NOT NULL
birthdate	DATE NOT NULL	DATE NOT NULL	DATE NOT NULL, birth_year INT NOT NULL, -- For partitioning INDEX idx_birth_year (birth_year) -- For partitioning optimization	DATE NOT NULL
PK	PRIMARY KEY (id, birthdate)	PRIMARY KEY (id, birthdate)	PRIMARY KEY (customer_id, birth_year)	PRIMARY KEY (birth_year, customer_id)

Approaches to partitioning

Grok	Google	Ollama	ChatGPT
<pre>PARTITION BY RANGE (YEAR(birthdate)) (PARTITION p_before_1970 VALUES LESS THAN (1970), PARTITION p1970 VALUES LESS THAN (1980), PARTITION p1980 VALUES LESS THAN (1990), PARTITION p1990 VALUES LESS THAN (2000), PARTITION p2000 VALUES LESS THAN (2010), PARTITION p2010 VALUES LESS THAN (2020), PARTITION p2020 VALUES LESS THAN (2030), PARTITION p_future VALUES LESS THAN MAXVALUE)</pre>	<pre>PARTITION BY RANGE (YEAR(birthdate)) (PARTITION p2010 VALUES LESS THAN (2011), PARTITION p2011 VALUES LESS THAN (2012), PARTITION p2012 VALUES LESS THAN (2013), PARTITION p2013 VALUES LESS THAN (2014), PARTITION p2014 VALUES LESS THAN (2015), PARTITION p2015 VALUES LESS THAN (2016), PARTITION p2016 VALUES LESS THAN (2017), PARTITION p2017 VALUES LESS THAN (2018), PARTITION p2018 VALUES LESS THAN (2019), PARTITION p2019 VALUES LESS THAN (2020), PARTITION p2020 VALUES LESS THAN (2021), PARTITION p2021 VALUES LESS THAN (2022), PARTITION p2022 VALUES LESS THAN (2023));</pre>	<pre>PARTITION BY YEAR(birthdate) (PARTITION p2000 VALUES IN (YEAR(CURDATE()) - 2000, YEAR(CURDATE()) - 2001), -- 2000-2001 PARTITION p2005 VALUES IN (YEAR(CURDATE()) - 2005, YEAR(CURDATE()) - 2005), -- 2005 PARTITION p2010 VALUES IN (YEAR(CURDATE()) - 2010, YEAR(CURDATE()) - 2010), -- 2010 PARTITION p2015 VALUES IN (YEAR(CURDATE()) - 2015, YEAR(CURDATE()) - 2015), -- 2015 PARTITION p2020 VALUES IN (YEAR(CURDATE()) - 2020, YEAR(CURDATE()) - 2020) -- 2020);</pre>	<pre>PARTITION BY RANGE (birth_year) (PARTITION p1950s VALUES LESS THAN (1960), PARTITION p1960s VALUES LESS THAN (1970), PARTITION p1970s VALUES LESS THAN (1980), PARTITION p1980s VALUES LESS THAN (1990), PARTITION p1990s VALUES LESS THAN (2000), PARTITION p2000s VALUES LESS THAN (2010), PARTITION p2010s VALUES LESS THAN (2020), PARTITION p_future VALUES LESS THAN MAXVALUE);</pre>
<p>8 partitions by decades</p>	<p>13 partitions by year (?)</p> <p>? practical for human ages ?</p>	<p>5 partitions, by five years (?)</p> <p>? can you explain calculation at first glance??</p>	<p>8 partitions by decade</p>

Example 2 - Conclusion

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

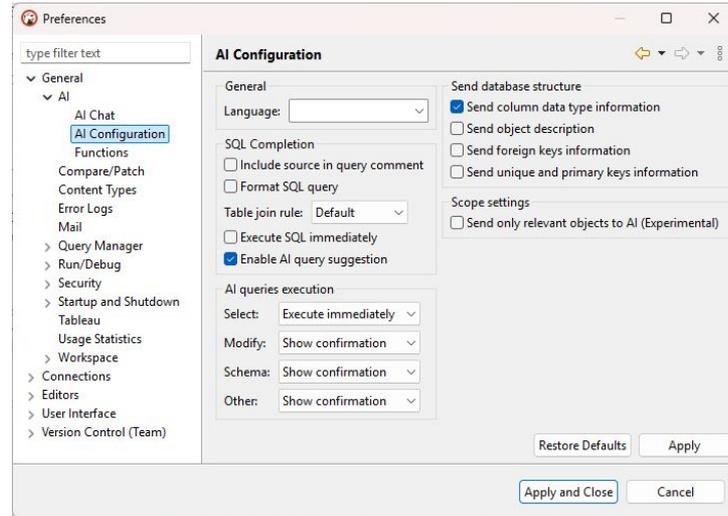
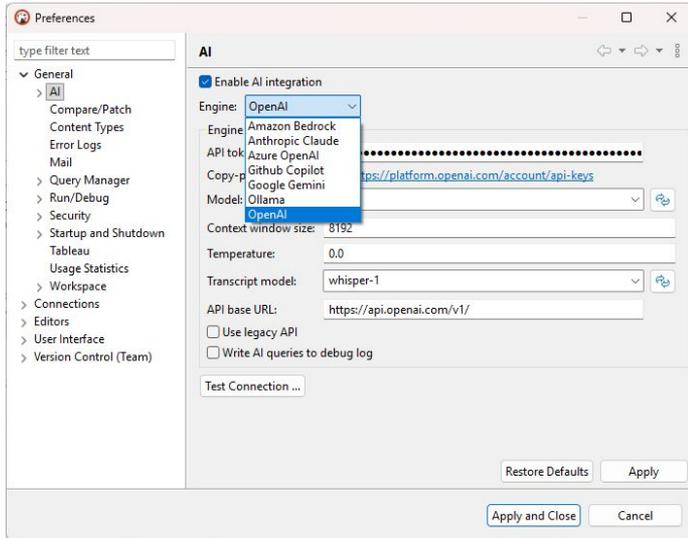
DBeaver's AI Assistant

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

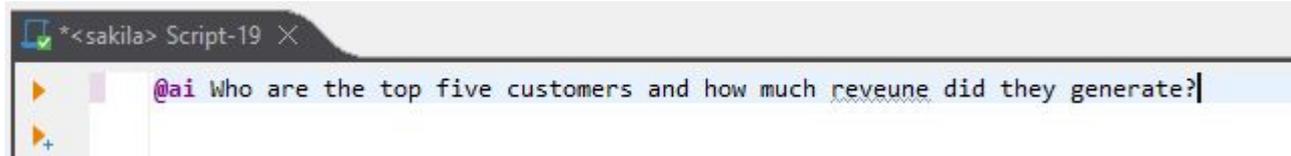
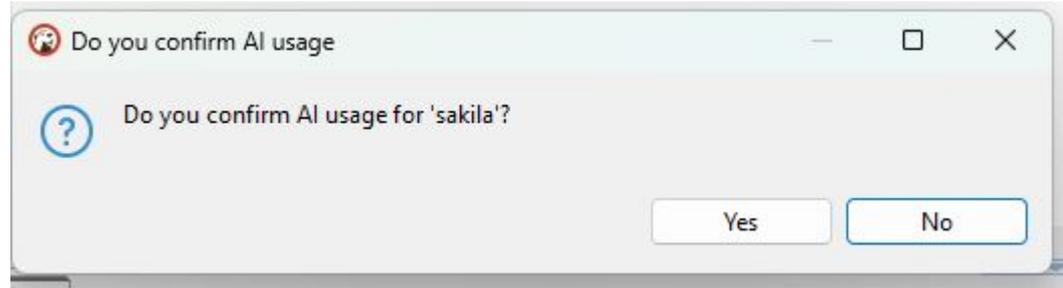


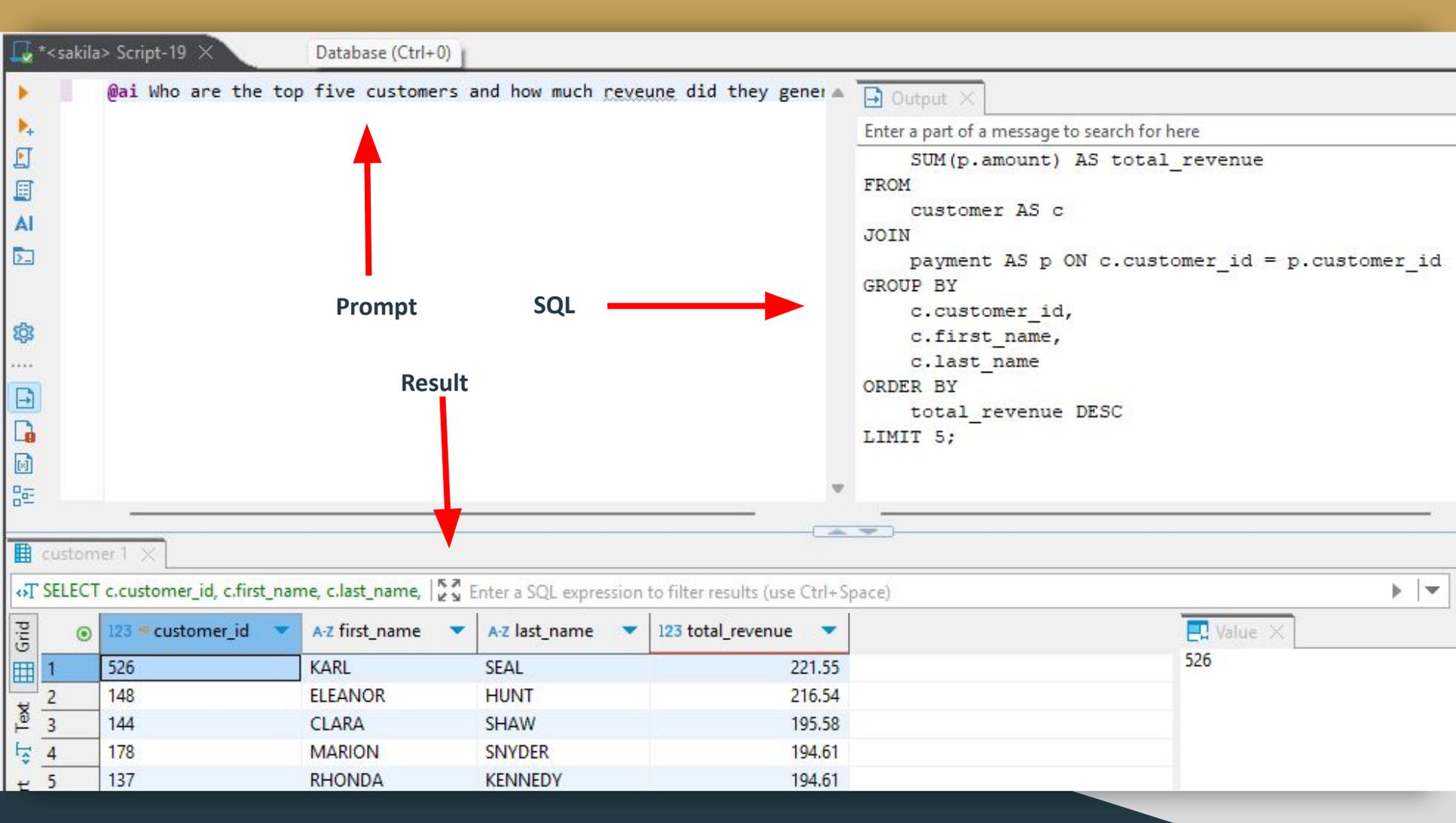
DBeaver Enterprise

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



You control Your data





Describe the SQL you want

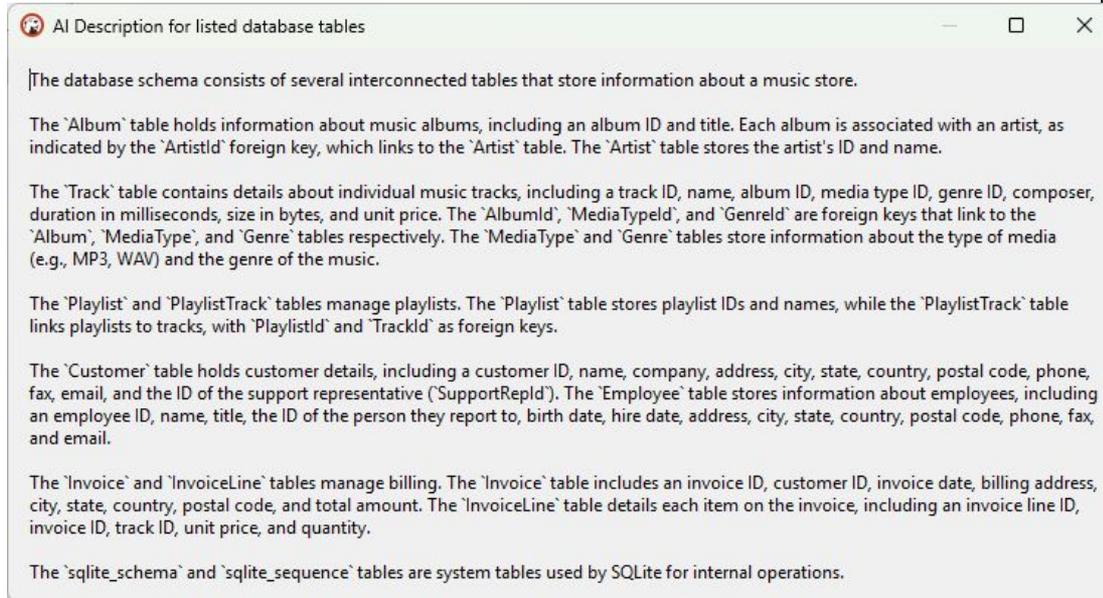
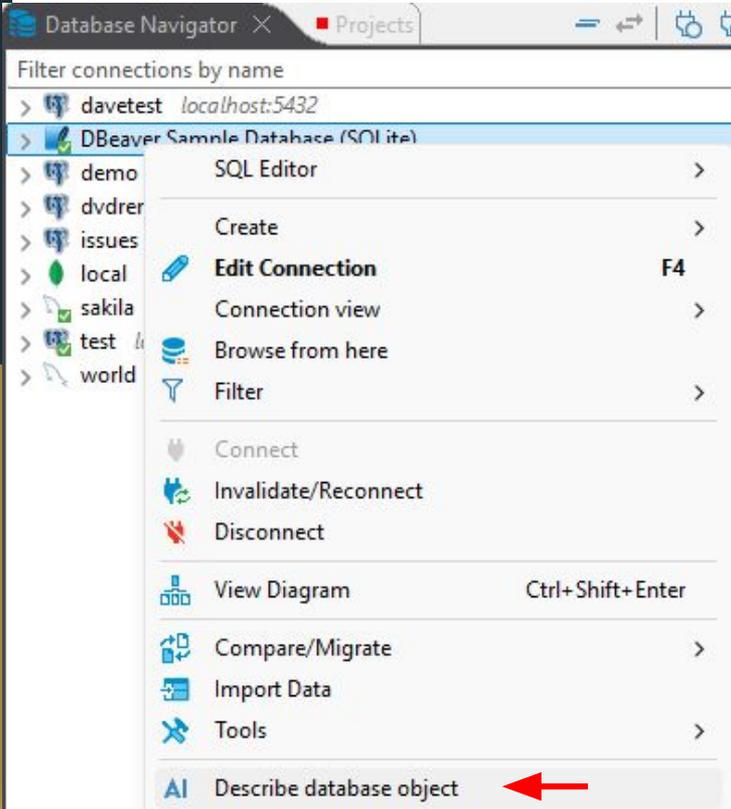
```
*<sakila> Script-19 X
-- Five top cities for revenue
select
```

```
*<sakila> Script-19 X
-- 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
```

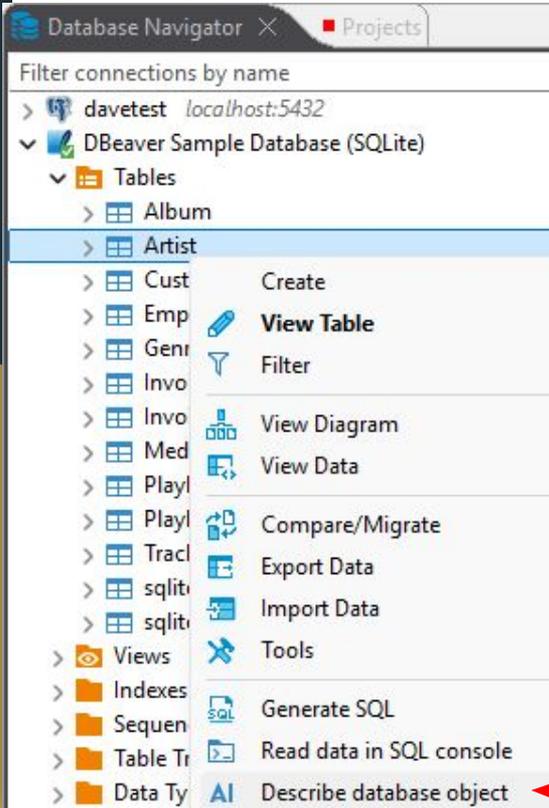
```
*<sakila> Script-19 X
-- Five top cities for revenue
select
city.city,
SUM(payment.amount) as total_revenue
from
payment
join customer on
payment.customer_id = customer.customer_id
join address on
customer.address_id = address.address_id
join city on
address.city_id = city.city_id
group by
city.city
order by
total_revenue desc
limit 5
```

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

Explanation of 'new' database



Explanation of a table



AI Description for table 'Artist'

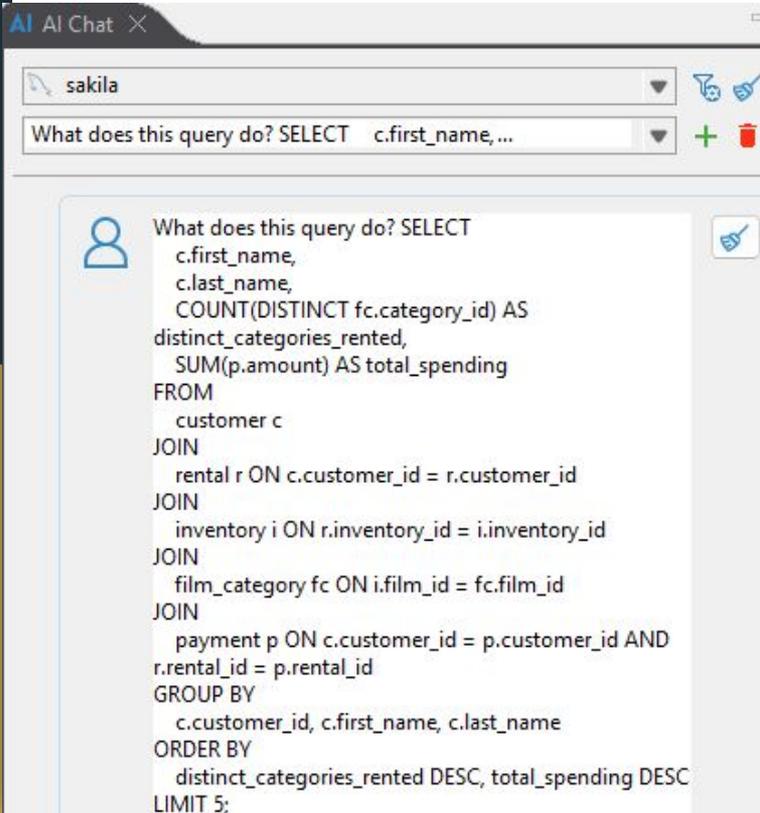
The 'Artist' table in the database schema is a key entity that stores information about various artists. It has two columns: 'ArtistId', which is an integer and serves as the primary key, and 'Name', which is a variable character field that holds the name of the artist.

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?



The screenshot shows an AI Chat window with a search bar containing 'sakila' and a text input field with the question 'What does this query do?'. Below the input is a user icon and the SQL query text. To the right of the query is a blue speech bubble icon.

```
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

```
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;
```

```
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;
```

Query Builder

Drag and drop query authoring

DBeaver Enterprise 25.3.0 - <sakila> Console

File Edit Navigate Search SQL Editor Database Window Help

SQL Commit Rollback Auto sakila sakila

Database Nav... *<issues> i... issues_new *<issues> C... <issues> Con... sakila <sakila> Con... 33

Filter connections by

- > davetest localhost:5432
- > db1.dev.dbeaver.infra db 1.
- > DBeaver Sample Database (S
- > default db9.dev.dbeave. AI
- > demo localhost:5432
- > dvdrental localhost:5432
- > foo localhost:5432
- > issues grafana.dbeaver.i...
- > master db9.dev.dbeaver.i...
- > PG18 localhost:5433
- > pg18 PG localhost:5433
- ✓ sakila localhost:3306
 - Database
 - > demo
 - > sakila
 - Tables
 - > actor
 - > address
 - > category
 - > city 64K
 - > country
 - > customer
 - > customers
 - > film 272K
 - > film_actor
 - > film_category
 - > film_text

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

Editor Builder Query

Drag tables
on to the
palette

The screenshot displays a database management interface. At the top, three tables are shown in a diagrammatic view:

- 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

Relationships are indicated by lines with circular icons: a line connects 'customer c' to 'store s' (on 'customer c' side), and another line connects 'customer c' to 'rental r' (on 'customer c' side).

At the bottom, the 'Query' editor shows the following SQL query:

```
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
```

The interface also includes a 'Results 1' pane with a table structure:

#	Column or Expression	Ali
1	*	

Navigation and tool icons are visible on the left and bottom edges of the window.

Select desired columns

Change join type

The screenshot shows a database query editor with a query plan diagram and a SQL query. The query plan diagram shows three tables: customer c, store s, and rental r. The customer c table is highlighted in orange, store s in yellow, and rental r in yellow. The SQL query is:

```
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
```

The query editor interface includes a toolbar on the left, a top toolbar, and a bottom toolbar. The bottom toolbar shows 'Results 1' and 'Query X'.

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

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!





Thank you for your attention!



[DBEaver.com](https://www.dbeaver.com)

David.Stokes@DBEaver.com
<https://www.linkedin.com/in/davidmstokes/>

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