Introduction to SQL with AI

SQL with AI Cheat Sheet

Learn SQL online at www.DataCamp.com

Why use AI to write SQL?

SQL (“structured query language”) is the standard programming language for analyzing data in databases. Using AI to write SQL speeds up analyses, reduces errors, and lets analysts focus on insight rather than syntax.

> Dataset

We’ll use data from the “A Song of Ice and Fire” book series, stored as the table `asoiaf_books`.

position_in_series	title	pages	release_date	era
1	A Game of Thrones	694	1996-08-06	1990s
2	A Clash of Kings	768	1998-11-16	1990s
3	A Storm of Swords	973	2000-08-08	2000s
4	A Feast for Crows	753	2005-10-17	2000s
5	A Dance with Dragons	1016	2011-07-12	2000s
6	The Winds of Winter	NULL	NULL	Unreleased
7	A Dream of Spring	NULL	NULL	Unreleased

> Principles of Good SQL Prompting

1. State the goal clearly
Clearly describe the question you want answered so the AI understands why the query is being written.
Bad: Write a SQL query about the books.
Good: Find the average number of pages for books that have been released.

2. Provide database context
Tell the AI which table and columns to use so it doesn’t have to guess.
Bad: Show me all the books sorted by release date.
Good: Using the asoiaf_books table with columns title and release_date, return all book titles sorted by release date.

3. Specify the SQL dialect
Different databases have different syntax; some AI tools may require you to name the SQL dialect.
Bad: Write SQL to get the longest book.
Good: Write a PostgreSQL query to return the title and page count of the longest book in asoiaf_books.

4. Define assumptions explicitly
Make implicit rules explicit so the AI handles the data the way you expect.
Bad: Count how many books have been released.
Good: Count how many books have been released, treating books with NULL release_date as unreleased.

5. Describe the desired output
Specify exactly what columns, sorting, or aggregation you want in the final result.
Bad: Find books after the year 2000.
Good: Return title and release_date for books released after 2000, sorted by release date ascending.

6. Include edge cases
Call out missing values or unusual data so the query behaves correctly.
Bad: Find the average number of pages.
Good: Find the average number of pages, excluding books where pages is NULL.

7. Constrain the solution
Adding constraints helps the AI produce simpler, more appropriate SQL.
Bad: Get the most recent book.
Good: Using a single SELECT query, return the most recently released book by release_date

8. Ask for explanations when needed
Ask for a brief explanation to help you validate the logic and learn SQL faster.
Bad: Write the SQL and explain everything.
Good: Write the SQL and briefly explain how NULL values are handled in the filter.

> Select Columns

Select columns by name

Prompt: Get the title and number of pages for all books **Generated SQL:**

```
SELECT title, pages
FROM asoiaf_books;
```

Select all columns

Using * tells SQL to return every column in the table. You can return more than one column by listing them in the SELECT clause, separated by commas.

Prompt: Show all information for every book. Generated SQL:

```
SELECT *
FROM asoiaf_books;
```

*Some databases like DuckDB allow FROM table or TABLE table, but SELECT * FROM table works everywhere.*

Order results

ORDER BY controls the order in which rows are returned.

Prompt: Get all book titles ordered by their position in the series. **Generated SQL:**

```
SELECT title
FROM asoiaf_books
ORDER BY position_in_series;
```

Limit results

Limiting results is useful when you only want a subset of rows, such as the first few records.

Prompt: Get the first 3 books in the series. **Generated SQL (most dialects):**

```
SELECT title
FROM asoiaf_books
ORDER BY position_in_series
LIMIT 3;
```

Generated SQL (SQL Server):

```
SELECT TOP 3 title
FROM asoiaf_books
ORDER BY position_in_series;
```

Select distinct values

DISTINCT removes duplicate values and returns only unique entries.

Prompt: Get the unique book eras. **Generated SQL:**

```
SELECT DISTINCT era
FROM asoiaf_books;
```

Create column aliases

AS renames a column in the query output, making results easier to read.

Prompt: Get the title and number of pages, and rename pages to page_count. **Generated SQL:**

```
SELECT title, pages AS page_count
FROM asoiaf_books;
```

> Filter Rows

Filter rows on numeric criteria

Filter with equal to

Use = to return rows where a numeric value matches exactly.

Prompt: Get books with exactly 768 pages. **Generated SQL:**

```
SELECT title, pages
FROM asoiaf_books
WHERE pages = 768;
```

Filter with not equal to

Use <> (or != in most dialects) to exclude a specific numeric value.

Prompt: Get books that do not have 768 pages. **Generated SQL:**

```
SELECT title, pages
FROM asoiaf_books
WHERE pages <> 768;
```

<> is the official SQL standard for not equal to, but all the most popular SQL dialects support !=

Filter with greater than

Use > to return rows where a numeric value is above a threshold.

Prompt: Get books with more than 800 pages. **Generated SQL:**

```
SELECT title, pages
FROM asoiaf_books
WHERE pages > 800;
```

Filter with less than or equal to

Use <= to include values equal to or below a threshold.

Prompt: Get books with 800 pages or less. **Generated SQL:**

```
SELECT title, pages
FROM asoiaf_books
WHERE pages <= 800;
```

Filter with BETWEEN

BETWEEN selects values within an inclusive range.

Prompt: Get books with between 700 and 900 pages. **Generated SQL:**

```
SELECT title, pages
FROM asoiaf_books
WHERE pages BETWEEN 700 AND 900
```

Filter rows on text criteria

Filter with equal to

Use = to match an exact text value.

Prompt: Get books with the title ‘A Storm of Swords’. **Generated SQL:**

```
SELECT *
FROM asoiaf_books
WHERE title = ‘A storm of swords’;
```

Filter with IN

IN matches a column against a list of possible values.

Prompt: Get all books from the 1990s and 2000s. **Generated SQL:**

```
SELECT title, era
FROM asoiaf_books
WHERE era IN ('1990s','2000s');
```

Filter with LIKE

LIKE matches text patterns using wildcards such as %.

Prompt: Get books with titles that contain ‘of’. **Generated SQL:**

```
SELECT title, pages
FROM asoiaf_books
WHERE title LIKE '%of%';
```

Filter rows on logical criteria

Filter with NOT IN

NOT IN excludes rows that match a list of values.

Prompt: Get books that are released. **Generated SQL:**

```
SELECT title, era
FROM asoiaf_books
WHERE era NOT IN ('Unreleased');
```

Filter with AND

AND requires all conditions to be true.

Prompt: Get books from the 2000s with more than 800 pages. **Generated SQL:**

```
SELECT title, pages
FROM asoiaf_books
WHERE era = '2000s';
```

Filter with OR

OR returns rows that meet at least one condition.

Prompt: Get books that are unreleased or have more than 900 pages. **Generated SQL:**

```
SELECT title, pages, era
FROM asoiaf_books
WHERE era = ‘Unreleased’
OR pages > 900;
```

Filter rows with missing values

Filter with IS NULL

Use IS NULL to find missing values.

Prompt: Get books without a release date. **Generated SQL:**

```
SELECT title
FROM asoiaf_books
WHERE release_date IS NULL;
```

> Aggregate Data

Calculate summary statistics

Count all rows

Use = to return rows where a numeric value matches exactly.

Prompt: Count the total number of books. **Generated SQL:**

```
SELECT COUNT(*) AS total_books
FROM asoiaf_books;
```

Sum values

SUM adds together values in a numeric column.

Prompt: Get the total number of pages across all books. **Generated SQL:**

```
SELECT SUM(pages) AS total_pages
FROM asoiaf_books;
```

Average values

AVG calculates the mean of a numeric column.

Prompt: Get the average number of pages per book. **Generated SQL:**

```
SELECT AVG(pages) AS average_pages
FROM asoiaf_books;
```

Find the minimum value

MIN returns the smallest (or earliest) value in a column.

Prompt: Get the earliest release date. **Generated SQL:**

```
SELECT MIN(release_date) AS earliest_release_date
FROM asoiaf_books;
```

Find the maximum value

MAX returns the largest (latest) value in a column.

Prompt: Get the latest release date. **Generated SQL:**

```
SELECT MAX(release_date) AS latest_release_date
FROM asoiaf_books;
```

Most SQL dialects automatically ignore NULL values in SUM, AVG, MIN, and MAX, which is why unreleased books do not affect these results.

> Complex aggregations

Count rows by group

Use COUNT(*) with GROUP BY to count rows per category.

Prompt: Count the number of books per era. **Generated SQL:**

```
SELECT era COUNT(*) AS books_per_era
FROM asoiaf_books
GROUP BY era;
```

Sum values by group

Use SUM with GROUP BY to total a numeric column for each group.

Prompt: Get the total number of pages per era. **Generated SQL:**

```
SELECT era SUM(pages) AS total_pages
FROM asoiaf_books
GROUP BY era;
```

Average values by group

Use AVG with GROUP BY to calculate an average for each group.

Prompt: Get the average number of pages per era. **Generated SQL:**

```
SELECT era AVG(pages) AS average_pages
FROM asoiaf_books
GROUP BY era;
```

Find the earliest value per group and sort

Use MIN with GROUP BY to find the earliest release date in each group, and ORDER BY to sort the results.

Prompt: For each era, get the earliest release date, ordered from earliest to latest. **Generated SQL:**

```
SELECT era MIN(release_date) AS earliest_release_date
FROM asoiaf_books
GROUP BY era
ORDER BY earliest_release_date;
```

Filter rows before grouping

Prompt: Get the average number of pages per era, excluding unreleased books. **Generated SQL:**

```
SELECT era AVG(pages) AS average_pages
FROM asoiaf_books
WHERE era IS NOT NULL
GROUP era;
```

Filter rows after grouping

HAVING filters after grouping (whereas WHERE filters rows before grouping).

Prompt: Show eras where the average number of pages is greater than 800. **Generated SQL:**

```
SELECT era AVG(pages) AS average_pages
FROM asoiaf_books
GROUP BY era
HAVING AVG(pages) > 800;
```