MCP Server for Database Querying

An MCP server exposes five controlled, read-only database tools. An AI client picks a tool; the server validates the request and returns bounded rows.

MCP is a protocol and does not write SQL. The LLM inside the client interprets your question; this server never trusts the SQL it receives.

- Goal: learn MCP, SQL, Python, SQLAlchemy and defensive design.
- Use cases: approved analytics questions, data exploration, internal helpers.

> Status: verified on 18 Nov 2026 (Windows 11, Python 3.12, MySQL 8.0): 27 validation tests pass, and `backend/smoke_test.py` called all five tools through the official MCP client against the sample database. A desktop AI client was not tested.

Download the project

The zip holds the backend, database scripts, tests, this website and the setup guide. Your `.env` and `.venv` are never included.

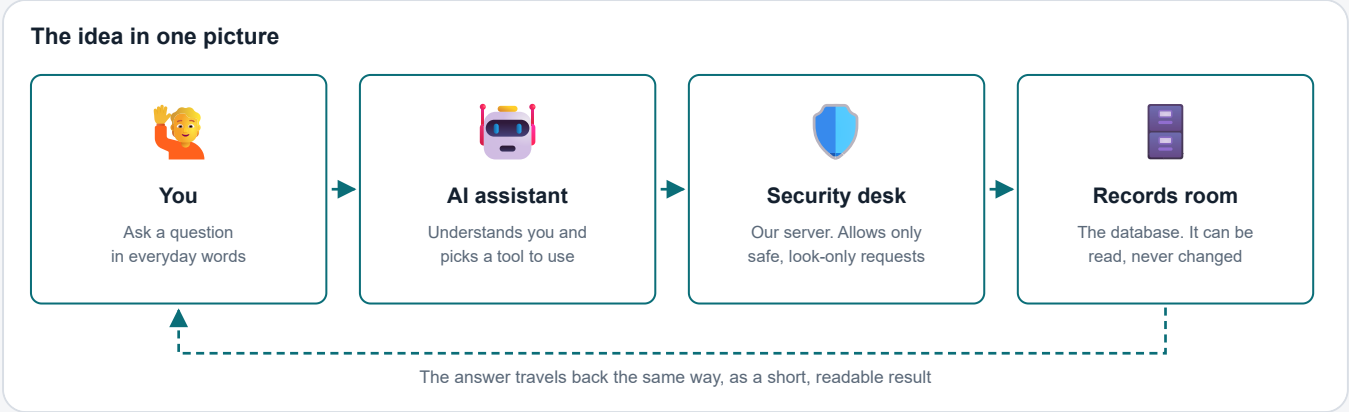
Free download: https://uncodemy.com/projects/mcp-server-database-query-project

The setup guide lists what to learn, what to install on macOS, every command in order with the result to expect, and common errors.

Visual guide

The same project explained in pictures. No technical background needed.

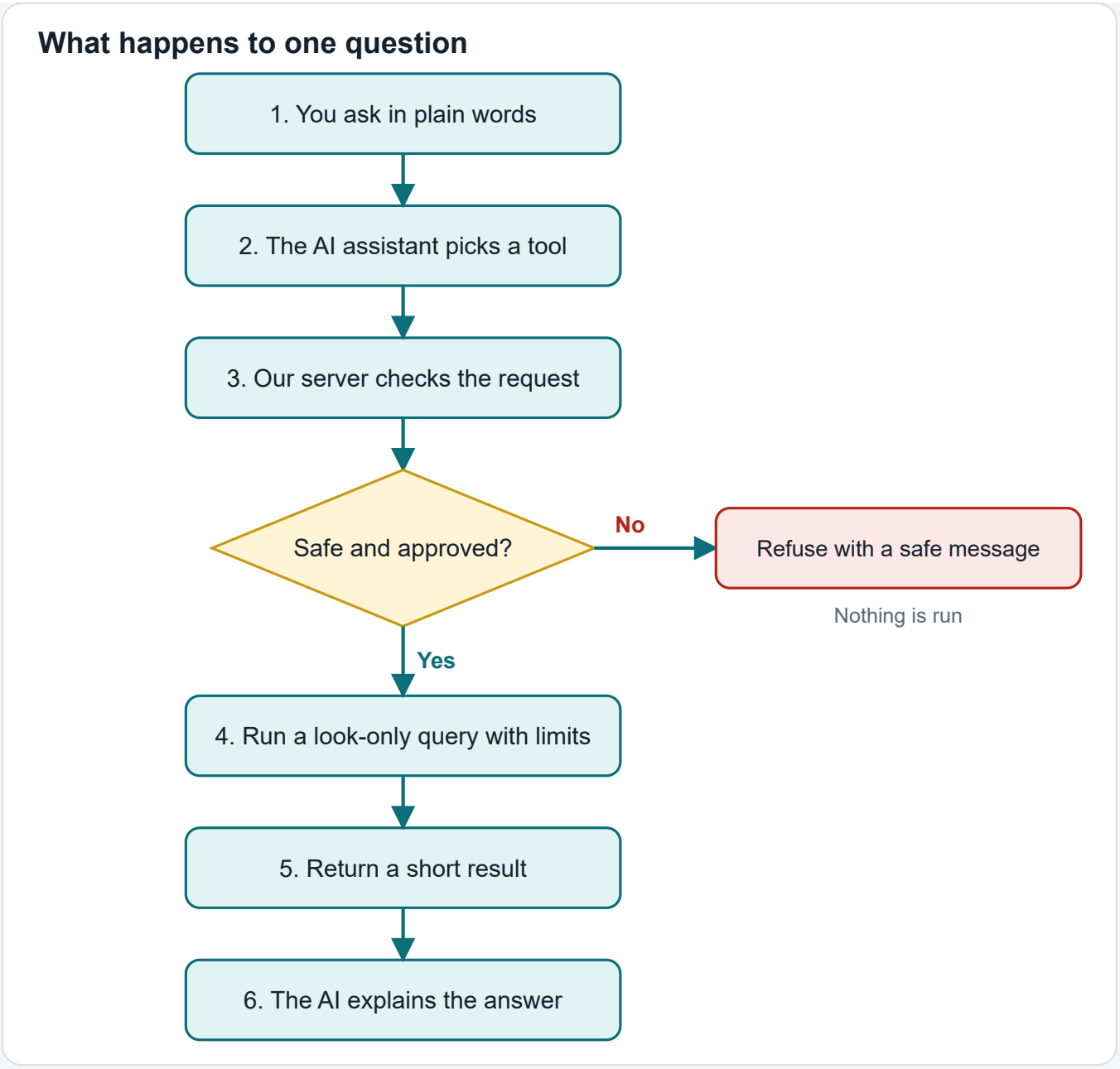
The idea in one picture



Think of a library with a very careful front desk.

You ask a helper (the AI assistant) a question in everyday words. The helper never walks into the records room itself. It goes through our security desk (the server), which only allows look-only questions about approved lists. The records room (the database) hands back just what was asked, and the helper explains it to you.

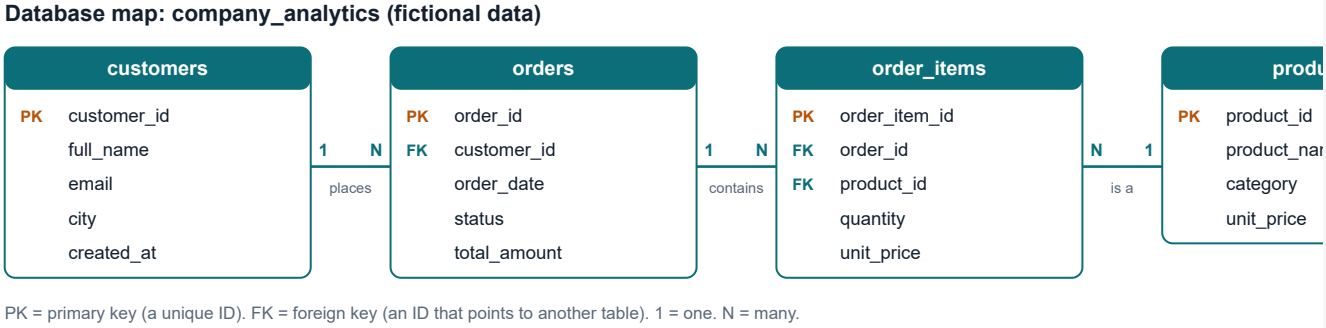
What happens to one question



Every question follows the same path.

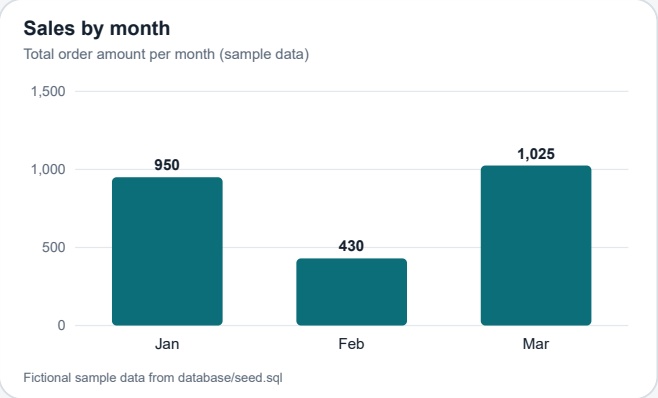
If a request looks risky, such as trying to change or delete information, it is refused and nothing is run. Safe requests run with limits on time and on how many rows come back.

How the information is organised

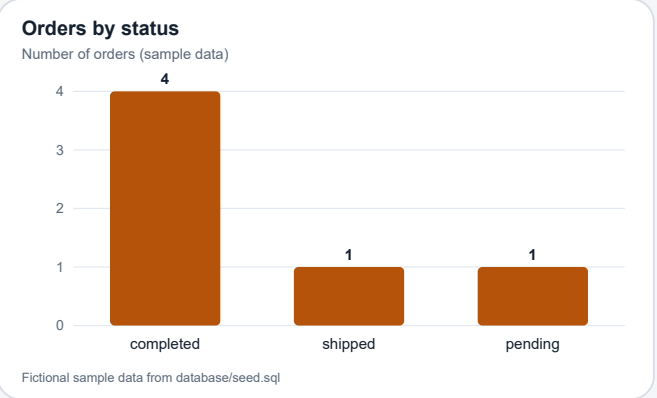


There are four lists (tables). Customers place orders, each order contains items, and each item is one product. A **primary key (PK)** is a unique ID, like a membership number. A **foreign key (FK)** is that ID reused in another list to link records, for example an order stores the customer ID so we know who placed it.

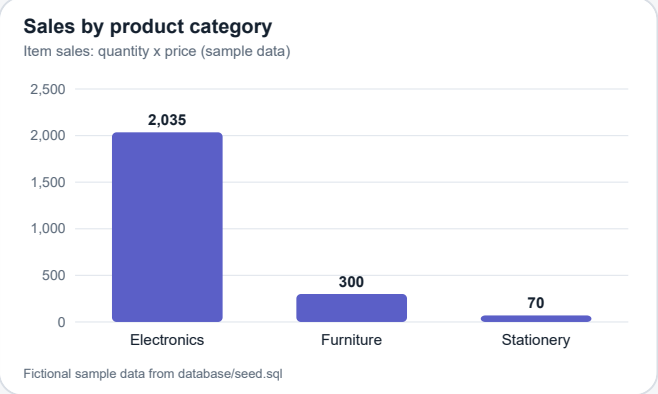
What answers could look like



Which month had the most sales?



How many orders are in each stage?



Which product category sells most?

> The server returns rows, not charts; an AI client could draw charts like these from those rows. These ones are generated by `website/build_images.py` from the fictional sample data, not from a live database.

Glossary

Search terms

MCP

Model Context Protocol: a standard way for AI clients to use tools and data.

Client / Server

The client is the AI app; the server is our Python program that offers tools.

Tool / Resource / Prompt

A tool is a callable function; a resource is readable data; a prompt is a reusable template. This project implements tools only.

LLM

The language model in or behind the client. It chooses tools and writes SQL.

SQL / Query

The language for reading relational data; a query is one request.

Schema, Table, Row, Column

The structure of the data: a table holds rows (records) made of columns (fields).

Primary key (PK)

A column that uniquely identifies a row.

Foreign key (FK)

A column that points to a key in another table.

ORM / SQLAlchemy

An ORM maps tables to code objects. SQLAlchemy also offers a Core layer for plain SQL, which we use.

Driver (PyMySQL)

The library that talks to MySQL on SQLAlchemy's behalf.

Connection pool

A small set of reusable database connections.

Environment variable

A setting supplied outside the code, e.g. from `.env`.

Validation

Checking input is acceptable before using it.

Parameterized query

SQL with placeholders such as `:city`; values are sent separately so they cannot change the SQL structure.

Allowlist

A list of what is permitted; everything else is refused.

stdio transport

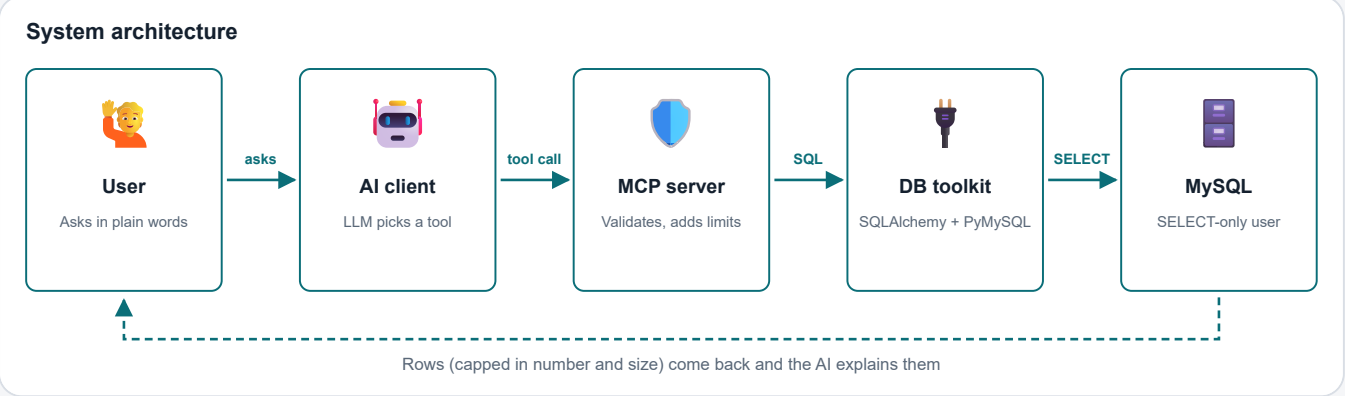
The client starts the server and talks to it over standard input/output.

Stack

Part	Role	Alternative
Python	Backend language	Node.js / TypeScript
MCP Python SDK (FastMCP)	Protocol and tool registration	Hand-written protocol (not advised)
MySQL	Data store	PostgreSQL, SQLite
SQLAlchemy 2 + PyMySQL	Connections and execution	Driver only
sqlglot	SQL parsing for validation	sqlparse (weaker structure)
HTML/CSS/JS	This static site	React (not needed)

Architecture

Five parts work together. The AI client understands your question; our MCP server is the only door to the database and checks every request.



Solid arrows go forward as requests. The dashed arrow brings the answer back.

User

AI client

Asks in everyday language. Never touches the database directly.

An MCP-compatible app. Its LLM reads the tool list and picks a tool and arguments.

`MCP server`

Our Python program. Checks the request, applies limits and returns safe errors.

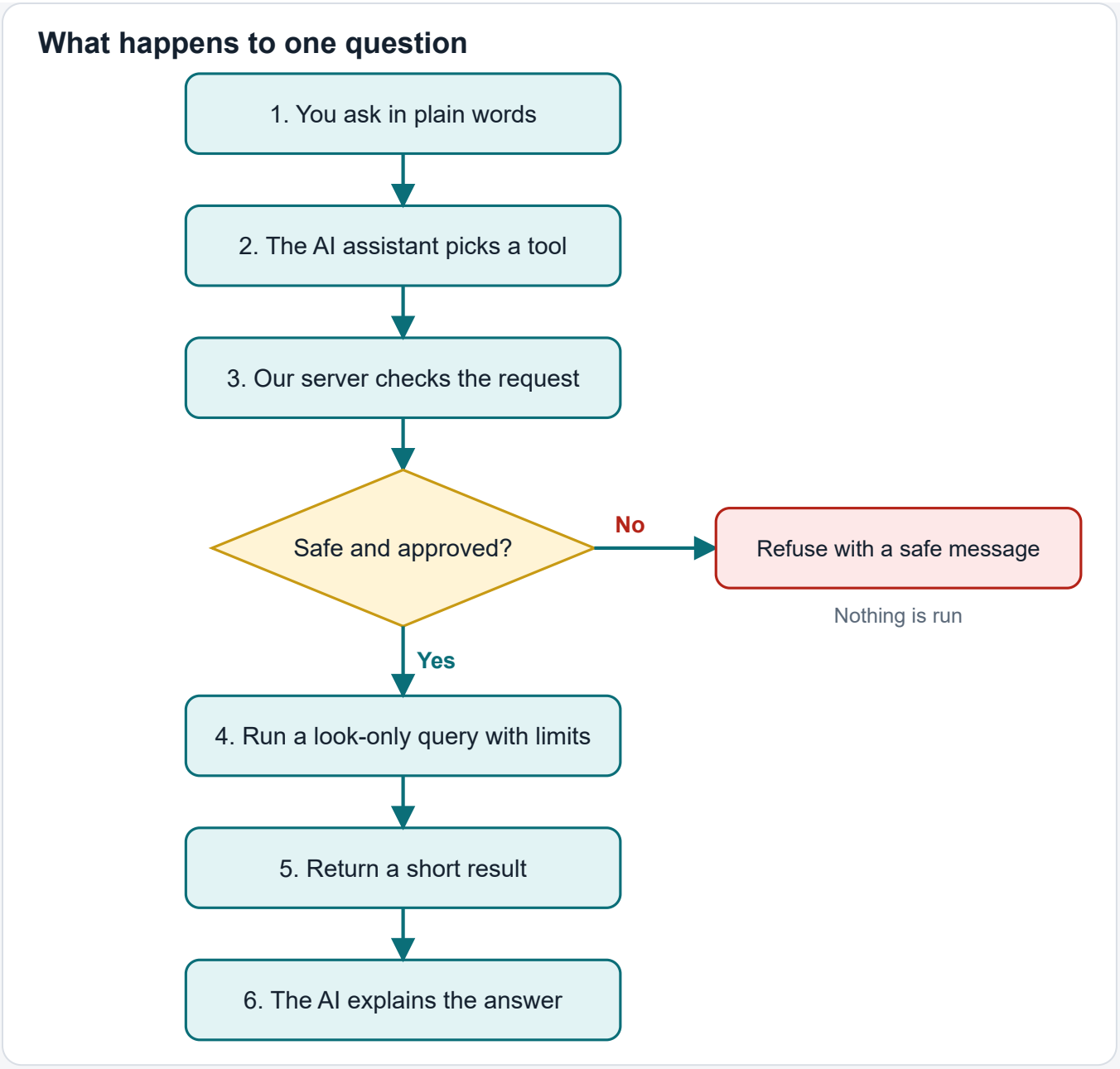
`DB toolkit`

SQLAlchemy runs the query through the PyMySQL driver, with timeouts and a small connection pool.

`MySQL`

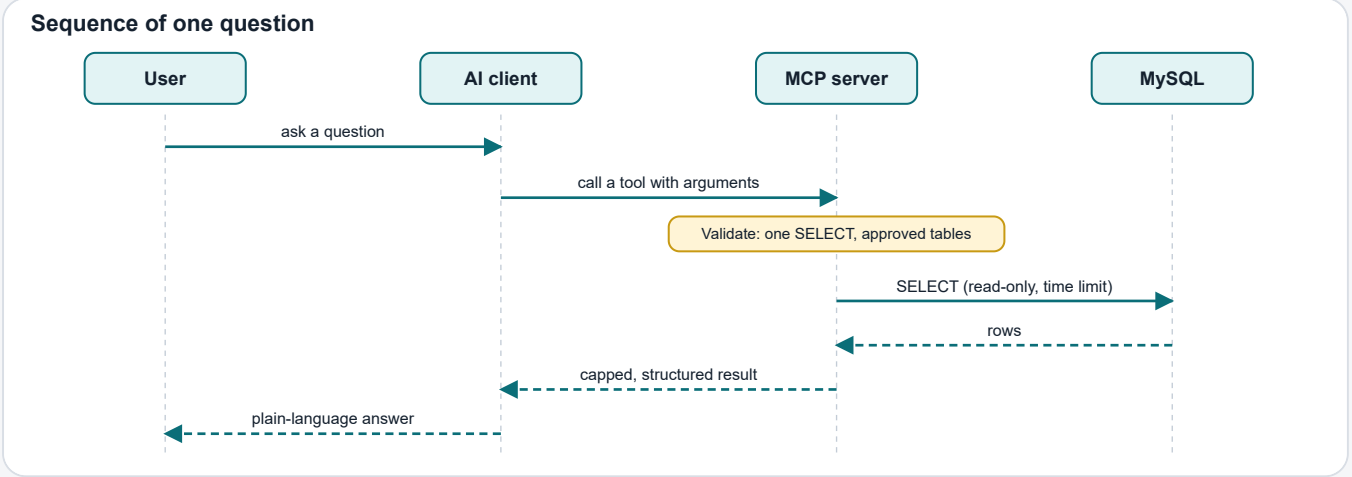
Stores the data. The server logs in as a SELECT-only user.

One question, step by step



Unsafe requests stop at the diamond. Nothing is run.

Sequence



Time runs downward. Dashed arrows are results coming back.

1. The user asks; the client hands the question to its LLM.
2. The client calls a tool on the server with arguments.
3. The server validates before anything touches MySQL.
4. Only a validated SELECT is sent, with a time limit.
5. MySQL returns rows; the server caps them and returns structured data; the client explains it.

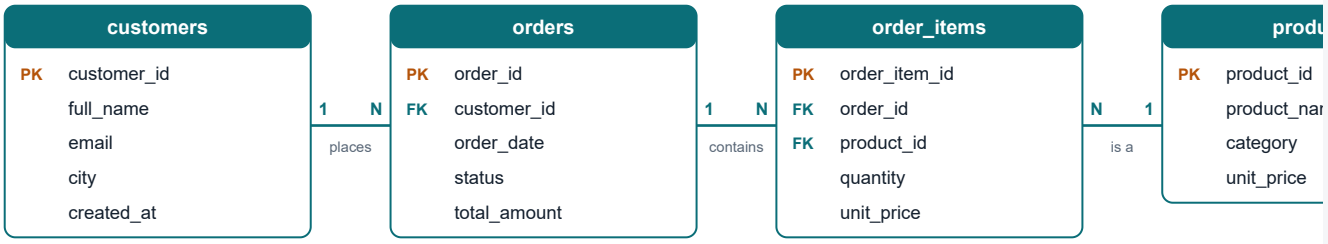
▼ Text version of the diagrams

```
User -> AI client (LLM picks tool) -> MCP server (validate) -> SQLAlchemy/PyMySQL ->
MySQL -> bounded rows -> MCP server -> AI client -> explanation to user
```

Database and ER

The sample database `company_analytics` models a small online shop. All people and values are fictional.

Database map: company_analytics (fictional data)



PK = primary key (a unique ID). FK = foreign key (an ID that points to another table). 1 = one. N = many.

Relationships: one customer has many orders; one order has many items; one product appears in many items. **PK** uniquely identifies a row; **FK** points to a row in another table.

`customers`

People who buy.

- PK `customer_id`
- `full_name`
- `email (unique)`
- `city`
- `created_at`

`products`

Things for sale.

- PK `product_id`
- `product_name`
- `category`
- `unit_price`

`orders`

`order_items`

One purchase by one customer.

PK order_id
FK customer_id
order_date
status
total_amount

One product line inside an order.

PK order_item_id
FK order_id
FK product_id
quantity
unit_price

Indexes on city, category, customer, date and the foreign keys speed up lookups. Example question: total spend per customer.

```
SELECT c.full_name, SUM(o.total_amount) AS spent
FROM customers c JOIN orders o ON o.customer_id = c.customer_id
GROUP BY c.full_name ORDER BY spent DESC
```

Tools

Tool	Input	Output
list_tables	none	approved table names
describe_table	table_name	column names, types, nullability
query_database	sql, parameters	columns, rows, row_count, truncated
get_table_sample	table_name, limit (1-50)	sample rows
get_database_stats	none	row count per table

Files

```
backend/app/server.py       MCP entry point, tool registration, logging
backend/app/config.py       .env settings, approved tables
backend/app/database.py     engine, timeout, row and size caps
backend/app/security.py     SQL validation
backend/app/tools/          metadata.py, query.py
backend/tests/              validation tests
backend/smoke_test.py       calls every tool through a real MCP client
```

```
database/                   schema.sql, seed.sql, create_readonly_user.sql
website/                    index.html, css/styles.css, js/main.js
```

Complete code for every file is in the Source code section.

Complete source code

Every file used to build this project. Choose a group, pick a file, then copy it. The code is generated from the real files by `website/build_code.py`; after editing any file, run `python3 website/build_code.py` to refresh this section.

backend/app/server.py (83 lines)

```
"""MCP server entry point (stdio transport). Run: python -m app.server"""
import atexit
import functools
import logging
import time
from typing import Any
from mcp.server.fastmcp import FastMCP
from .config import load_settings
from .database import Database, DatabaseError
from .security import QueryRejected
from .tools import metadata, query

logging.basicConfig(level=logging.INFO, format="%(asctime)s %(levelname)s %(message)s")
log = logging.getLogger("mcp-db")
mcp = FastMCP("company-analytics-db")
_db: Database | None = None


def db() -> Database:
    """Create the engine lazily so the server starts even if MySQL is down."""
    global _db
```

Commands

Run from the project root unless noted.

```
python3 --version
```

Checks Python (3.10+). Error: "command not found" means install Python.

```
python3 -m venv .venv && source .venv/bin/activate
```

Creates and enters an isolated environment; the prompt shows `(.venv)`.

```
pip install -r backend/requirements.txt
```

Installs dependencies. Error: version conflicts mean checking Python version.

```
cp .env.example .env
```

Creates your private settings file. Set DB_PASSWORD.

```
mysql -u root -p < database/schema.sql
mysql -u root -p < database/seed.sql
mysql -u root -p < database/create_readonly_user.sql
```

Creates tables, loads fictional rows, creates the SELECT-only user. Error: "mysql: command not found" means install MySQL.

```
cd backend && pytest -q
```

Runs the validation tests; no database needed.

```
cd backend && python -m app.server
```

Starts the stdio server. It prints nothing and waits for a client; that is normal.

Client config

Example for Claude Desktop. Replace the placeholders; config format varies by client and version, so check its docs.

```
{"mcpServers":{"company-analytics-db":{
  "command":"/ABS/PATH/mcp-server-database-query-project/.venv/bin/python",
  "args":["-m","app.server"],
  "env":{"PYTHONPATH":"/ABS/PATH/mcp-server-database-query-project/backend"}}}}
```

Examples

Question	Tool and arguments
What data is available?	list_tables
Show customers in Delhi	query_database: `SELECT full_name FROM customers WHERE city = :c`, `{"c":"Delhi"}`
How big is each table?	get_database_stats

With the seed data, the Delhi query returns Asha Example and Dev Test (checked with the smoke test).

Security

- SELECT-only MySQL user: the main boundary.
- Parser validation: one SELECT, approved tables, no INTO, locks, variables or SLEEP-style functions.
- Bound parameters, row cap, response cap, execution timeout.
- Safe errors; logs contain tool name, duration and status only.

- Limits: parser and MySQL may differ; valid but heavy queries can still be slow; hosted LLMs see returned rows.

Advantages, disadvantages and alternatives

Advantages

- **Easier data exploration.** People can ask approved questions in plain language instead of writing SQL for every need.
- **One standard interface.** MCP lets different compatible AI clients use the same server without custom integrations.
- **Safer by design.** A SELECT-only user, parser checks, row caps and timeouts reduce the chance of accidental damage.
- **Controlled access.** Only approved tables are exposed, so private tables stay hidden.
- **Low cost to run.** It runs locally on a laptop with no paid service and no model training.
- **Good learning project.** It combines Python, SQL, database design, AI tool use and security in one place.

Disadvantages

- **The AI can be wrong.** It may misread a question or choose the wrong tool, so answers should be checked.
- **Valid SQL can still be costly.** A read-only query can be slow or heavy; timeouts and row caps only limit the damage.
- **Validation has limits.** A parser cannot promise to match MySQL in every case, so the database permissions remain the real safety net.
- **Client support varies.** Setup steps and config formats differ by client and version.
- **Ongoing maintenance.** Schema changes, new tables and permission updates need work; outages return errors.
- **Privacy.** Rows returned to a hosted model leave your machine, which may break company or legal rules.

Alternatives

Alternative	Better when	Trade-off
SQL and a database GUI	Users know SQL and want full control	No natural language; skill needed
REST API	An app needs fixed, predictable endpoints	Every new question needs new code
Text-to-SQL web app without MCP	Only one app needs database access	Not reusable by other AI clients
BI dashboard	Teams need governed, repeatable reports	Less flexible for one-off questions
SQLite instead of MySQL	A tiny local demo with minimal setup	Fewer permission controls

Troubleshooting

▼ `ModuleNotFoundError: No module named 'app'`

Likely cause: The command ran from the wrong folder.

Fix: Run from `backend/`, or set PYTHONPATH to the backend path.

```
cd backend
python -m app.server
```

▼ `No module named 'sqlglot'` **(or mcp, sqlalchemy, pytest)**

Likely cause: The virtual environment is not active or the install was skipped.

Fix: Activate the environment and install again from the project root.

```
source .venv/bin/activate
```

```
pip install -r backend/requirements.txt
```

▼ `Access denied for user 'mcp_readonly'`

Likely cause: The password in `.env` differs from the one in `create_readonly_user.sql`.

Fix: Make both match, then recreate the user.

```
mysql -u root -p < database/create_readonly_user.sql
```

▼ **Tool returns 'Database is unavailable'**

Likely cause: MySQL is stopped, the host or port is wrong, or the query timed out.

Fix: Start MySQL and check the DB_ values in `.env`.

```
brew services start mysql
```

▼ **Client shows no tools**

Likely cause: A wrong absolute path in the client config, or the client was not fully restarted.

Fix: Check the python path and PYTHONPATH, save, then quit and reopen the client.

▼ `ImportError` **for FastMCP**

Likely cause: The installed MCP SDK version differs from what the code expects.

Fix: Compare the import in `server.py` with the installed SDK's documentation and update it.

```
pip show mcp
```

▼ A query is rejected

Likely cause: The validator allows one SELECT on approved tables only. UNION, INTO, locks, variables and SLEEP are blocked.

Fix: Rewrite it as one simple SELECT using the four approved tables and :name placeholders.

▼ Download button does nothing

Likely cause: `website/downloads/` has not been built.

Fix: Build the files.

```
python3 website/build_code.py
python3 website/build_zip.py
```

Interview Q&A

▼ Does MCP generate SQL?

No. The client's LLM does; the server validates it.

▼ Why a read-only user?

It limits damage even if validation fails.

▼ Why parameterize?

Values travel separately, so they cannot alter the query structure.

▼ Why cap results?

It protects memory, the client's context and privacy.

▼ **What if MySQL is down?**

The tool returns a generic unavailable error without details.
