

ClickHouse

Giving Your AI Agent a Database Brain

A local SQL database your agent can actually query with chDB and Observe with Langfuse.

Mubashir (Moe) Masood · Architect @ ClickHouse

SQL
inside Python

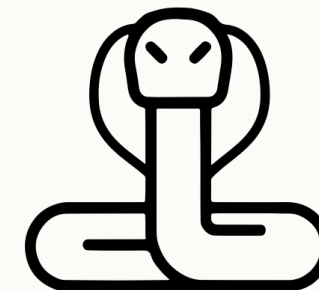


Mubashir Masood

@consultmasood

LinkedIn

- **AI Engineering Architect @ ClickHouse**
- **Ex-Principal AIML Architect @Snowflake**
- **Ex-Teradata, Ex- Massmutual**
- **Electrical and Computer Engineering, University of Houston**



ClickHouse

A familiar agent failure



Yesterday

It hit a constraint and found the fix that worked.



Today

Same task. It reads the prompt — not yesterday's fix.



Result

Repeats the failed plan.
Burns tool calls on retries.

The model isn't dumb. The right context just wasn't queryable in time.

Your agent's context is really a pile of data

Context engineering just means choosing what the agent sees. Four kinds of data:

Instructions

Rules, constraints, preferences.

Working data

Files, tables, API results, logs.

Memory

Decisions, facts, prior solutions.

Eval history

What worked, what failed, what changed.

Recall = find the few facts that matter *now*. A query, not a bigger prompt.

"Can't I just use what I already have?"

SQLite

Great for tiny state. No columnar scan, no vectors, no remote files.

Pandas

Great in memory. Dies past RAM. Can't query files you never loaded.

DuckDB

Closest match — but fewer formats and no mature cloud.

Vector DB

Similarity only. Can't filter + join + aggregate + audit in one query.

Vector search is just an index. Agent memory is the whole database around it.

ClickHouse in 60 seconds



Analytical database

Open-source. Fast SQL over huge tables: logs, events, metrics.



Columnar engine

Stores by column, so scans over millions of rows stay fast.



More than storage

A real engine: SQL, JSON, arrays, text search, vector distance.

OLAP & OLTP: built to scan billions of rows. Runs Anthropic, Cursor, Cloudflare, Shopee.

chDB: a rocket engine on a bicycle

The ClickHouse engine, embedded in your Python process — SQLite-simple, 100× faster for analytics.

```
# pip install chdb
from chdb import session as chs
sess = chs.Session("./agent")
sess.query("SELECT count() FROM file('log.pq')")
```

- Same engine: ClickHouse SQL, JOINS, vectors.
- Persistent: local tables on disk for memory.

your agent code



chDB · ClickHouse engine
runs in the same process



files · DataFrames · local tables

Local like SQLite. Analytical like ClickHouse.



chDB straps ClickHouse's engine onto Python.... like this.

chDB drops into your Python data stack



70+ data formats

Parquet, CSV, JSON, Arrow, ORC — read and write, no converters.



Zero-copy Pandas & Arrow

Query a DataFrame or Arrow table in place — no copy, no load.



Pandas-style API

Prefer DataFrames to raw SQL? chDB speaks that too.

It fits your existing Pandas / Arrow code, you don't rebuild around it.

Your Pandas Code

```
import pandas as pd

df = pd.read_csv("employee_data.csv")
filtered = df[
    (df['age'] > 25) &
    (df['salary'] > 50000)
]
grouped = filtered.groupby('city')['salary'] \
    .agg(['mean', 'sum', 'count'])
result = grouped.sort_values(
    'mean', ascending=False
).head(10)
```

Join anything with SQL

```
chdb.query("""
    SELECT name, age, purchase_total
    FROM url('https://api.co/users.parquet') users HTTP
    LEFT JOIN file('orders.csv') orders CSV
         ON users.uid = orders.uid
    LEFT JOIN Python(processed_df) enriched DataFrame
         ON enriched.uid = users.uid
""")
```

Three sources, one query — no load step.

same join, Pandas-style (runs on chDB):

```
import chdb.datastore as pd
(pd.read_parquet('https://api.co/users.parquet') HTTP
 .merge(pd.read_csv('orders.csv'), on='uid') CSV
 .merge(pd.DataFrame(enriched_df), on='uid')) DataFrame
```

What good agent memory needs

Memory the agent can filter, rank, and grow — not a prompt blob.



Memories are rows

Filter, rank, and audit them
with plain SQL.



Recall = Similarity + Filter

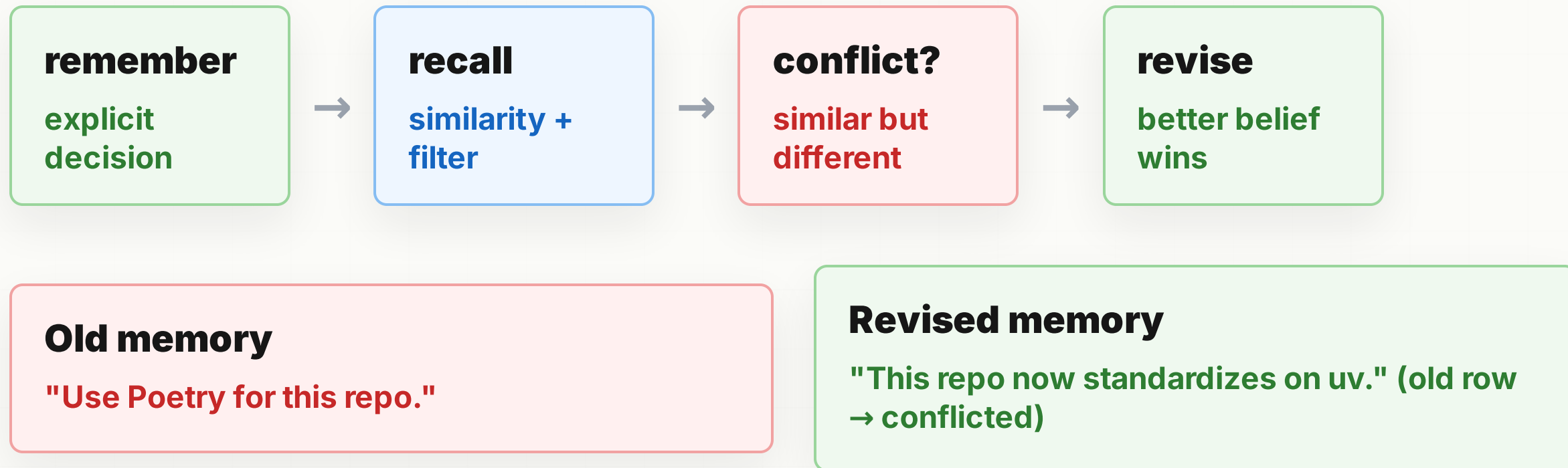
Meaning and rules in one
query.



Local now, cloud later

Same SQL from laptop to
ClickHouse.

Agent memory has a lifecycle



Recall is one SQL query

Rank memories by meaning, keep the active ones, pinned first.

cosineDistance

how close two meanings are, as a number
(0 = identical)

Need project, tags, or task-history? Add WHERE / JOIN lines to the same query.

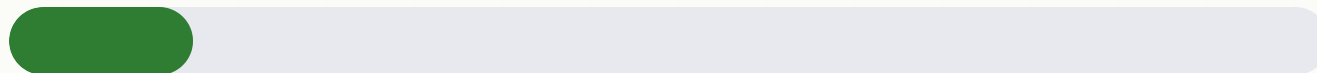
```
SELECT
  content,
  1 - cosineDistance(embedding, query_vec) AS meaning
FROM memories
WHERE status = 'active'
ORDER BY pinned DESC, meaning DESC
LIMIT 8
```

same recall, Pandas-style:

```
import chdb.datastore as pd
m = pd.DataStore(
    table="memories", database="./agent_context")
m = m.assign(
    meaning=1 - pd.F.cosineDistance(
        m.embedding, pd.F.array(*query_vec)))
m[m.status == "active"].sort_values(
    ["pinned", "meaning"], ascending=False).head(8)
```

Recall sits inside the agent loop

in-process recall



instant

remote recall in the loop



network tax



In the loop

Recall runs many times per turn — jitter becomes think-time jitter.



Retries hurt twice

A timed-out call is retried — each retry re-sends the whole prompt.



Local wins

In-process recall is orders of magnitude faster. No tail, no retries.

Start local, scale to ClickHouse

Day 1

chDB session on a laptop
or agent runtime



Day 100

ClickHouse Server or
Cloud when memory is
shared or large

```
# Day 1 – query local memory
sess.query("SELECT content FROM memories WHERE project = 'checkout'")

# Day 100 – repoint the same name at the cluster (SELECT unchanged)
sess.query("""CREATE OR REPLACE VIEW memories AS
             SELECT * FROM remote('cloud:9440', db, memories)""")
```

Honest note: remote reads pay the network cost again.



Ship agents you can trust in production

Langfuse is the open-source agent observability and evaluation platform, built on ClickHouse.

Trace

Every input, output, tool call, latency, and cost.

Evaluate

Catch quality drops before a prompt or model change ships.

Monitor

Alert on cost, latency, and quality while the agent is live.

Engineering challenges for AI agents

Rapid iteration gets stuck on manual QA, operational risk, and missing governance.

Undetected quality regressions

Model degradations and hallucinations show up as user churn, not as alerts.

High debugging overhead

Finding the bad step across prompts, tool calls, and retrieval is still manual.

High-risk iteration

Without regression tests, a prompt or model update is a business risk.

Unmonitored performance drift

Latency, cost-per-token, and quality slip until users feel it.

langfuse.com/academy

GETTING AI AGENTS PRODUCTION READY

Agent observability and the AI engineering loop

Full visibility into what your agent actually did: Every input, output, tool call, latency and cost, step by step. The critical infrastructure enabling teams to transition from local development to scalable, production-ready AI.

● ONLINE · WHILE YOUR AGENT IS LIVE

Tracing

Monitor real user behavior.
Accelerate root-cause analysis
with visual, searchable traces.

Monitoring & alerts

Proactively monitor and alert.
Detect cost, latency, and
quality deviations via online
evals.

● OFFLINE · BEFORE YOU SHIP A CHANGE

Datasets

Add failure cases
or generate
golden datasets.

Experiments

Catch quality drops
systematically before
deploying.

Evals

LLM-as-a-judge
and deterministic
metrics.

Prompt management Ship prompt, model or config changes without a redeploy.

langfuse.com/academy

HOW LANGFUSE WORKS

The same loop, as a product

Tracing

trace · online

Every request and intermediate step, searchable and visual. Debug in minutes instead of digging through logs.

Datasets & experiments

build + experiment · offline

Turn real production traces into regression test sets. Catch a quality drop before it ships, not after.

Prompt management

connective tissue across every step

Version and deploy prompt or config changes without a redeploy and roll back instantly when something breaks. Domain experts can iterate without an engineering release.

Monitoring & alerts

monitor · online

Cost, latency and quality by agents, model, prompt version or user segment — proactive notification instead of customer complaint.

Evaluations

evaluate · offline

LLM-as-judge, custom scorers, human review and user feedback on live traffic or a curated test set.

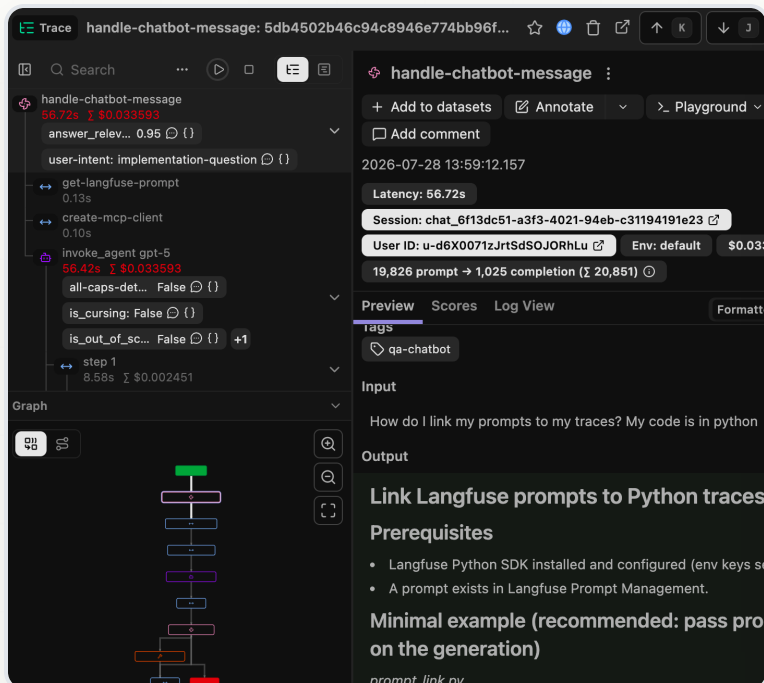
OpenTelemetry-native

140+ integrations like Langchain etc.

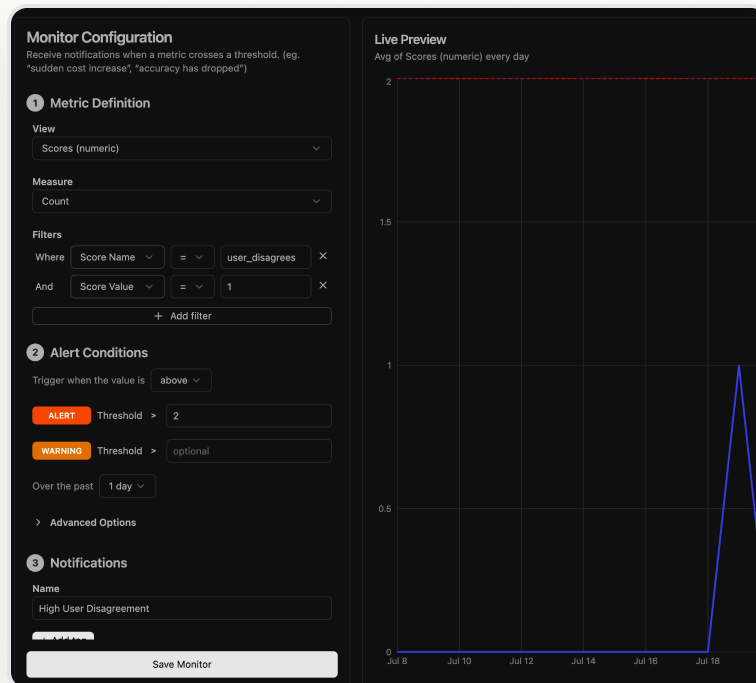
any model provider

open-source core · cloud or self-hosted

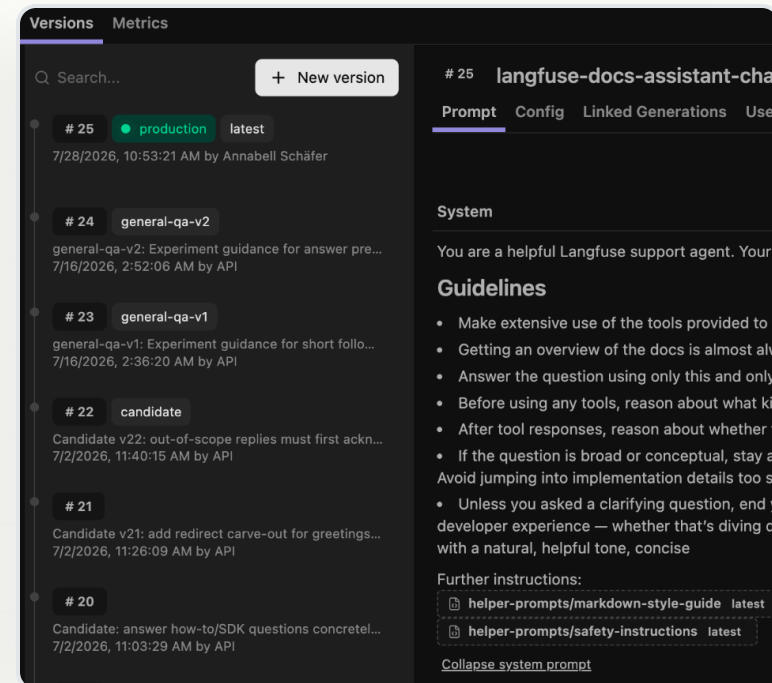
From a bad answer to the exact step that caused it



Tracing. Nested LLM calls, tool calls, and retrieval, with cost and latency.



Alerts. Get notified when cost or quality drifts.

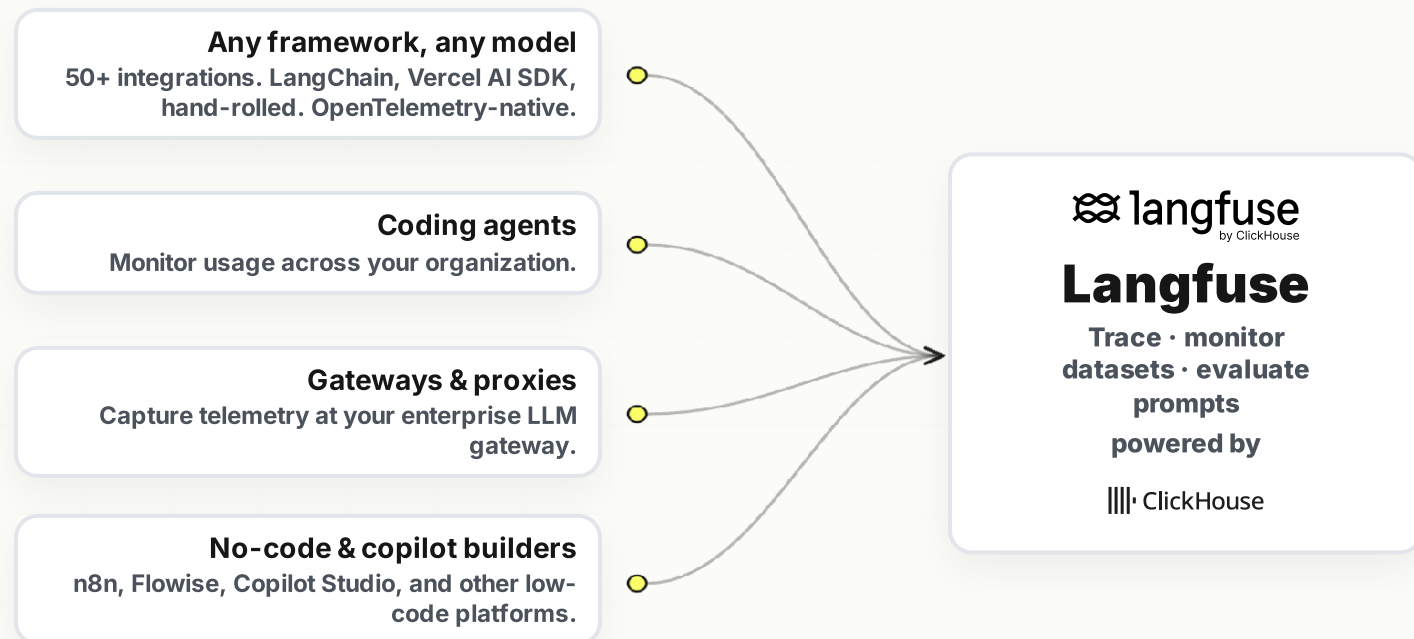


Prompts & review. Versioning and human scoring in one place.

Vendor-agnostic architecture

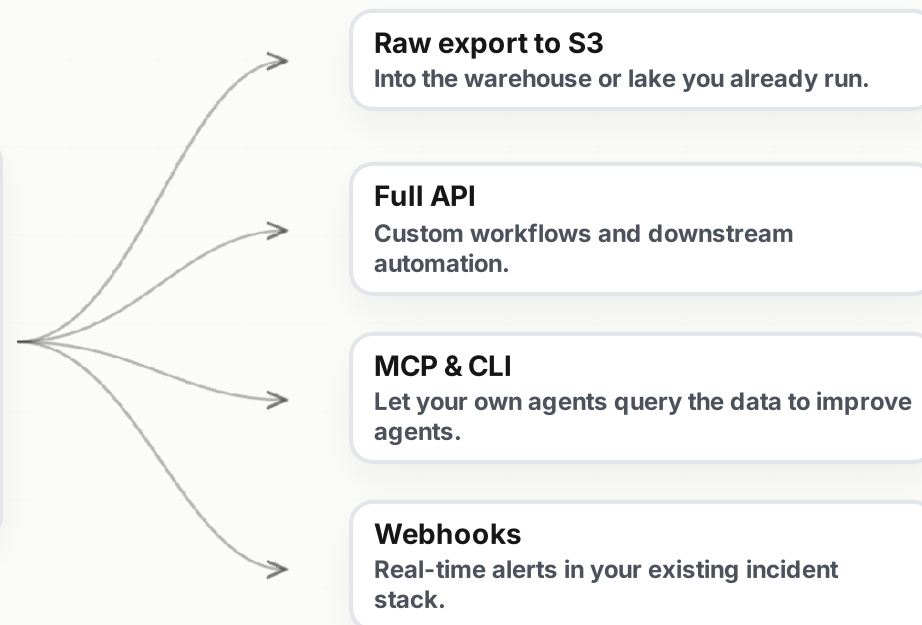
Seamless integration on the way in. Complete data ownership on the way out.

Universal ingestion



langfuse.com/integrations

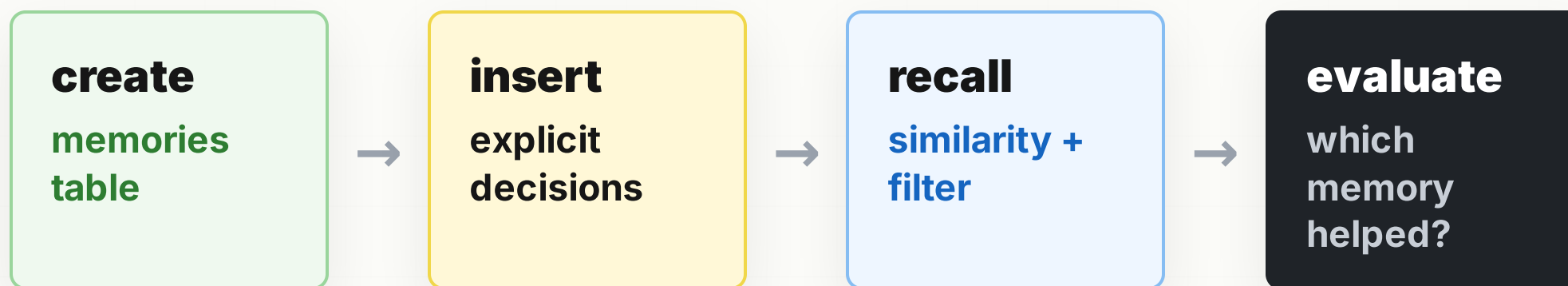
Unconstrained exports



langfuse.com/docs/api-and-data-platform

Meet ClickHouse

That whole pattern — lifecycle, recall, local-first — packaged as an open-source project on chDB. Build the core yourself:



```
sess = chs.Session("./agent_context")
sess.query("CREATE TABLE memories (...) ENGINE = MergeTree ORDER BY id")
sess.query("INSERT INTO memories VALUES (...)")
sess.query("""
    SELECT content FROM memories
    WHERE created_at > now() - INTERVAL 30 DAY -- filter
    ORDER BY cosineDistance(embedding, ?) LIMIT 8 -- similarity
    """)
```

Embeddings: **Qwen/Qwen3-Embedding-0.6B** via sentence-transformers, runs locally.

Three takeaways

01**Context is data.**

Filter it, rank it, join it, evaluate it.

02**Recall is a query.**

Semantic meaning + simple rules, in one SQL.

03**Local first.**

chDB now; ClickHouse when shared or large.

```
pip install chdb
```

github.com/chdb-io/chdb · github.com/langfuse/langfuse · github.com/clickhouse/clickhouse

**Q&A**