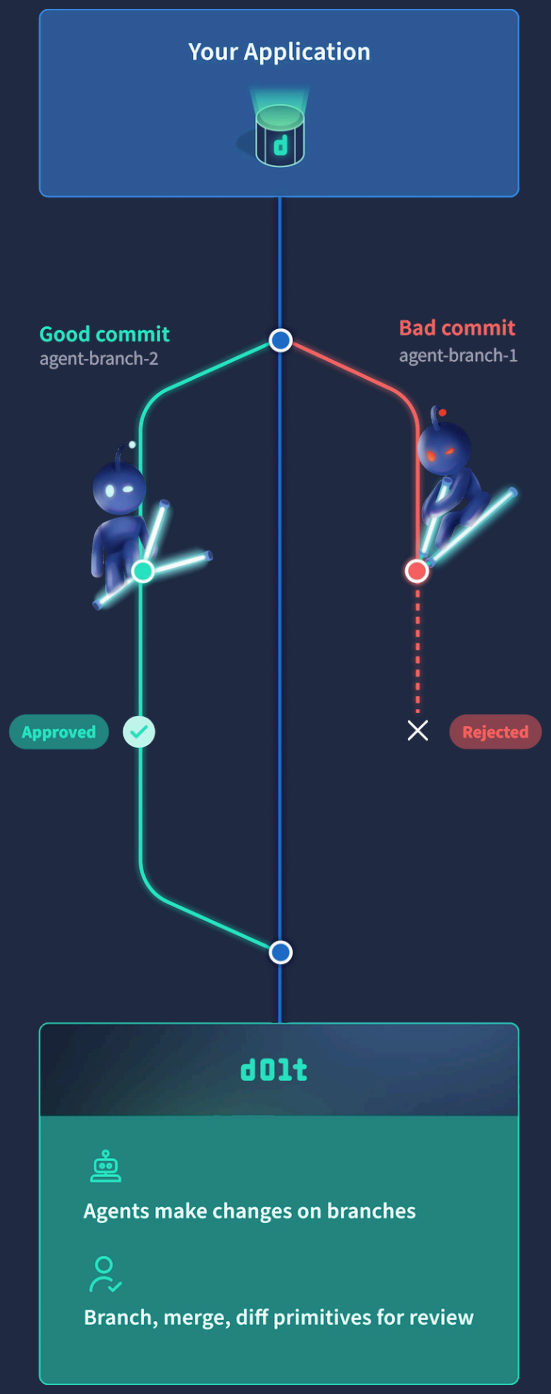
A WHITE PAPER FROM DOLTHUB

How to do agentic AI on your data ~~recklessly~~ safely.



Agents are coming for your database. This is the field manual for letting them in without losing everything you've spent years collecting.

> You trust agents to write code because version control gives you safety and traceability. If you had the same version control for everything else, you would let agents handle that too. Most of that 'everything else' lives in databases, not files.
>
> THE THESIS

An agent on your database is a loaded gun.

The dawn of practical agentic AI arrived in February 2025 with Claude Code. A little over a year later, agents are writing production code at scale across the industry. The trick that made it work — letting models loop, fail, and self-correct — is now being pointed at every other system in the stack. Including, dangerously, the database.

Code is forgiving. You can git reset --hard and pretend the last hour never happened. Data is not. An agent that updates 40,000 customer records with a malformed query has changed the world. There is no undo button.

And yet teams are connecting LLMs directly to production databases — because the demos look good and the velocity is intoxicating. The question is not whether agentic systems will touch your data. They already are. The question is whether you've built the infrastructure for it to happen without catastrophe.

FAILURE 01

No rollback

Production databases assume changes are intentional and final. An agent that mis-prompts itself into a destructive UPDATE has no native way to undo. Your DR plan is a nightly backup.

FAILURE 02

No diff

You can review a pull request before it merges. There is no equivalent for data changes. Reviewing what an agent did to 12,000 rows in your orders table is, currently, impossible.

FAILURE 03

No isolation

Multiple agents writing to the same tables produce race conditions, conflicting state, and emergent garbage. Branching exists for code. It does not exist for your data.

Every working agent stands on three pillars.

Coding agents work because they sit inside three concentric guard rails: a **capable model**, **version control**, and **tests**. Strip any one of them out and the whole loop collapses.

01

A capable model

The hype center of AI, and arguably the least leveraged of the three. Models matter — but state-of-the-art versus last-quarter matters less than people think. Pick one. Move on.

02

Version control

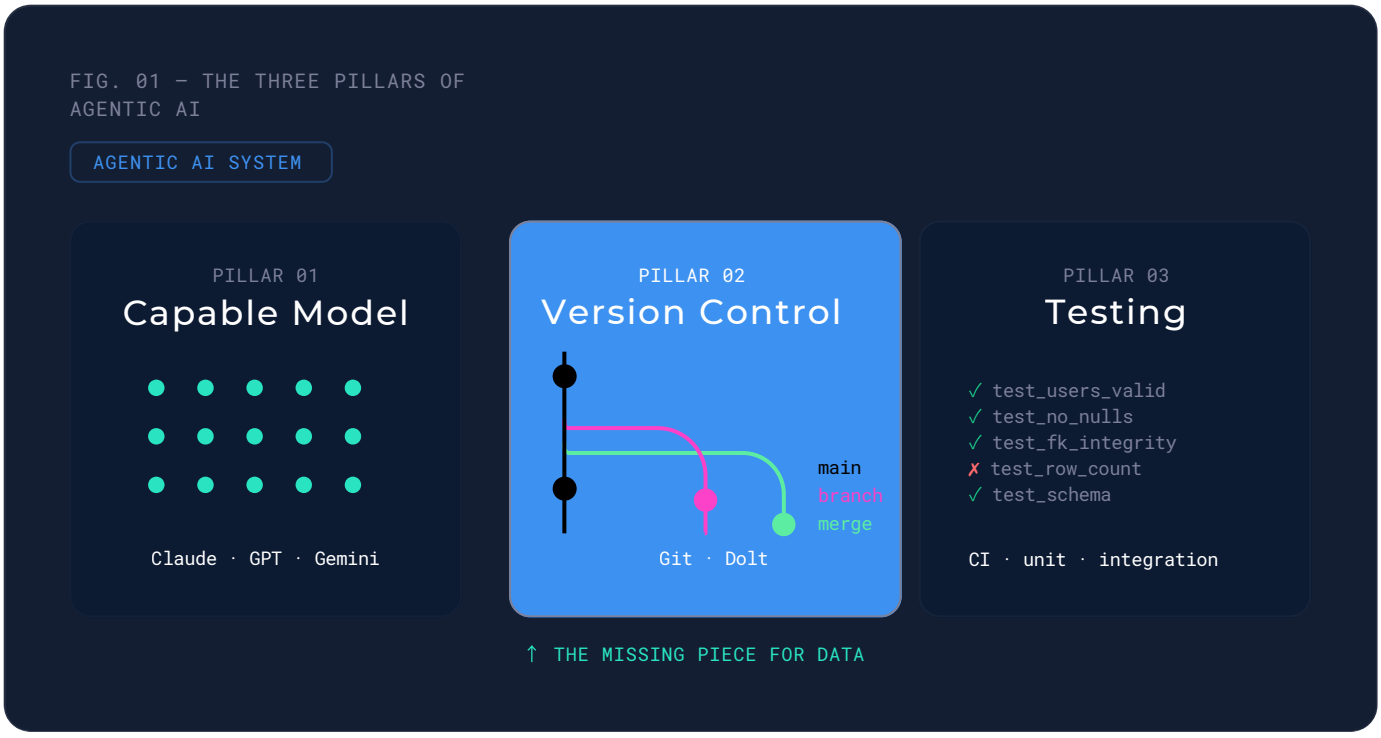
Branches. Diffs. Merges. Rollback. The only reason coding agents are tolerable is that everything they touch is undoable. Without this layer, every agent run is a high-wire act with no net.

03

Testing

The most underrated of the three. Tests give agents something to loop against. Without verifiable output, agents wander. With it, they self-correct. Tests at least double an agent's productivity.

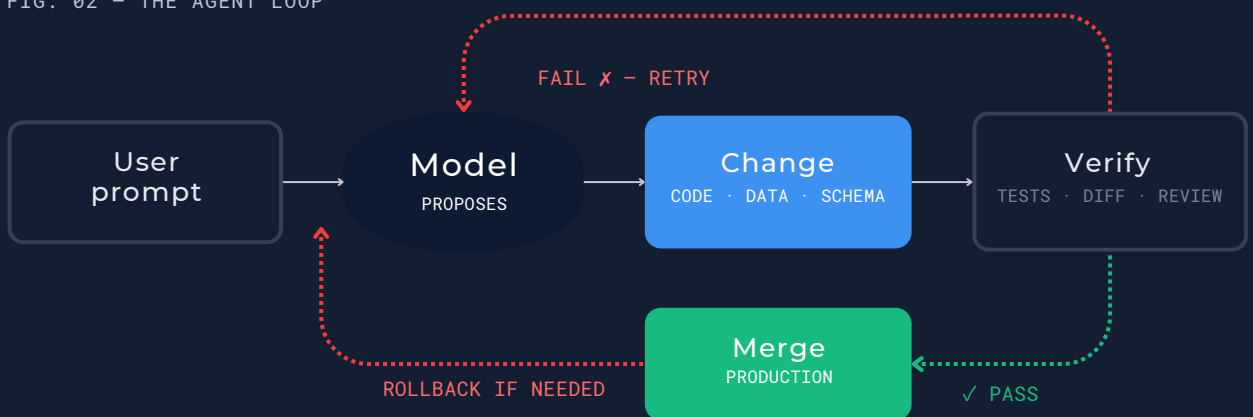
FIG. 01 — THE THREE PILLARS OF AGENTIC AI



The loop is everything. The loop needs guardrails.

An agent is not a single inference. It is a closed loop: propose a change, verify the change, repeat. The verification step is what makes agentic AI actually agentic — and it is what most teams forget when they extend this pattern from code to data.

FIG. 02 — THE AGENT LOOP



For code, this loop has been refined for two decades: Git provides version control, CI provides testing, and the model fills in the middle.
For data — until recently — no such loop existed. The middle pillar was missing. The right pillar was rudimentary. Most agentic-data deployments today are running on one pillar and a prayer.

WHY BRANCHES ARE NON-NEGOTIABLE

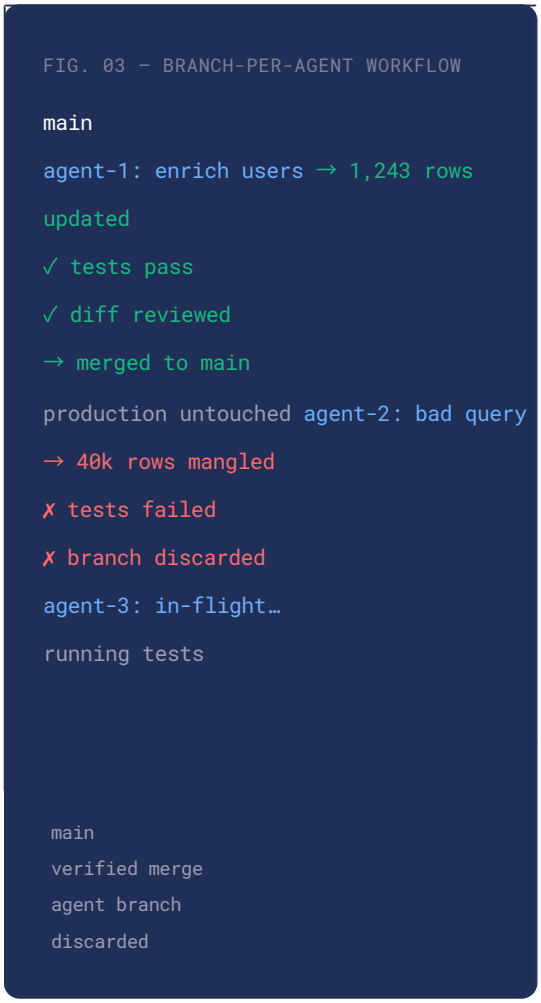
“

Without version control, you would be relying on perfect prompts and perfectly correct edits which seem impossible for current and maybe even future models.

The persistence layer is the database. That's where agents need to land.

If the three-pillar framework is correct, the question becomes:how do we extend it past the editor? Most applications are not code — they're data. The persistence layer of nearly every system is a database. Bringing version control and testing there is what makes the whole pattern work for everything.

A version-controlled SQL database, run as the persistence layer behind your agents.



01 **Branch per agent**

Each agent connects to a branch or clone of your production data. It can read everything, write freely, and never touch the live tables.

02 **Diff before merge**

When the agent finishes, compare its branch to production with queryable, row-level diffs. Humans or other agents review. Approve or reject.

03 **Tests as guardrails**

Schema constraints prevent obvious damage. Unit tests assert business rules. CI workflows run on every commit. The agent loops until it passes — or gets thrown away.

04 **Rollback anytime**

Bad merge? `doltreset` to any prior commit. Your data history is now an audit log, not a forensic exercise.

05 **Parallelism is native**

Run dozens or hundreds of agents in parallel on independent branches. Merge the winners. Discard the losers. Conflicts are surfaced, not silently overwritten.

Five steps to safe agentic AI on your data.

You can build most of this yourself. Here's the order to do it in.
Start where you are, ship in increments.

01

Pick the database you can branch.

Your current Postgres or MySQL does not have branches.
You have three options:
(a) use Dolt as a drop-in MySQL-or-Postgres-compatible replacement,
(b) build a completely isolated replica environment, or
(c) accept that you cannot safely run agents against your data.
We recommend (a).

```
# Dolt speaks MySQL and Postgres wire protocols.
# Point your existing app at a Dolt instance and
# most code keeps working unchanged.
dolt sql-server --host=0.0.0.0 --port=3306
```

02

Give every agent its own branch.

Treat the agent like a contract or on probation. It gets a workspace. It does not get the keys to the production database. When the agent connects, it connects on a freshly branched copy — its writes are isolated, its mistakes are local, and the rest of your system has no idea it exists.

```
CALL DOLT_CHECKOUT ('-b', 'agent/enrich-leads-2026-05-12');
--agent now writes here. main is untouched.
```

dolthub

03

Write tests like you mean it.

This is the step everyone skips and then regrets. Agents need verifiable output. Schema constraints catch the cheap mistakes; assertions on row counts, distributions, foreign-key integrity, and business invariants catch the expensive ones. The more tests, the longer your agent can run unsupervised.

```
-- Dolt unit tests: assert agent did not break anything.
SELECT assert(
 (SELECT COUNT(*) FROM users WHERE email IS NULL) = 0,
  'no users may have null email'
);
```

04

Diff.Review. Merge — or throw away.

When the agent says it's done, you (or another agent) inspect what changed. Dolt exposes diffs as query able system tables, so you can SQL your way through "what did this agent actually touch?" with row-level precision. Good diff? Merge. Bad diff? Reset the branch. Either way, production was never at risk.

```
SELECT * FROM dolt_diff_users
WHERE to_commit = 'HEAD'
  AND diff_type = 'modified';
--Every row the agent changed, queryable.
```

05

Automate the review loop.

Once one agent runs safely, run ten. Use CI to gate every branch merge: tests pass, diffs pass policy, optional human approval for high-stakes tables. The pattern is the same one that made GitHub Actions ubiquitous for code — except now it works on the data layer where most of your application's risk actually lives.

dolthub

WHERE WE COME IN

You can build this. We've already built it.

Dolt is the first and only version-controlled SQL database. We've spent years on the hard parts — the storage engine, the diff algorithms, the merge semantics, the MySQL and Postgres compatibility. The playbook above runs on top of it.

If you'd rather not figure out the corners yourself, we're here. The team that wrote the framework is the team that designed the database.

Three ways to start.

Pick whichever fits where you are in the journey

Try Dolt yourself
Free, open source.
Install in a minute.
GitHub →

Talk to the team
Active Discord.
Real engineers,
real answers.
Discord →

Bring us in
For teams running
agents on critical data.
Contact us →

dolthub

The version-controlled SQL database. Built for humans, designed for agents, ready for whatever you put on top of it.

READ MORE

Three Pillars of Agentic AI
Documentation
Engineering Blog

COMMUNITY

Git
Discord
Contact