Ask a Question in Plain Language

Experimenter · Admin Click a saved-question button, filling in its blanks if it has any — or type a new question, complete in itself. Get a narrative plus downloadable tables and/or charts; a new question can be saved for the team.

Screen Ask (contextual) `ask-surface.html` Ask (global) `ask-global.html` saved questions managed on Saved questions `saved-questions.html · saved-questions-admin.html`

Two ways to ask click a saved button — filling its inputs, if it has any `nl_query.label · params` or type a new question, complete in itself `canonical_prompt`

Writes a saved question — only when you Save a NEW one `nl_query`

Key fields button caption `label` your normalized words `canonical_prompt` generated statements `derived_sqls` typed inputs `params` visibility `status = draft / approved / system` similarity vector `query_vector`

1 · Given / When / Then

Human wording first; the exact table/field in `mono`.

GIVEN	The asker is signed in as an **experimenter** or **admin**. There is data to query and, optionally, ingested documents. The Ask surface already shows the built-in and approved saved questions for this context as one-click buttons.
WHEN	They open **Ask**, either from a screen (scoped to that context, such as Results) or from the global Ask, and get an answer two ways. **(a) Reuse:** they click a saved-question button. If that question takes inputs, it shows a small form (for example, *model* = Opus 4.8); otherwise it runs on a single click. Reuse involves no typing and no AI step: the button binds the inputs into its stored `derived_sqls` and runs. **(b) Ask new:** they type a question in plain language, complete in itself, and the engine embeds it. A close saved match reuses that question's stored SQL, lifting any values the typed words supply — a match on a parameterized question runs with those values and never shows the inputs form. Otherwise the engine compiles the words into zero or more read-only parameterized SQL statements. Either way the query runs under the asker's own row permissions and returns a narrative plus zero or more result tables, each viewable as a downloadable table or chart. Alongside the answer to a newly typed question, and only there, a Save control offers to keep it: the asker gives it a `label` and chooses, per value, whether to expose it as a reusable input (`params`) or keep it fixed. A reused button, and a typed question that matched one, offer no Save — the question already exists.

THEN Reusing a saved button writes nothing; it just runs, with your inputs bound if it has any. Only Save on a new question creates one `nl_query` row (`status = draft` , owned by the asker). The asker manages their own drafts on the **Saved-questions Screen**: edit the caption, the inputs and the notes, share a draft read-only with teammates, or delete it. An admin reviews drafts on the same screen and approves the good ones, so `status` flips to `approved` and the button appears for everyone in that context; an admin can also take an approved question back, by unapproving it or by retiring it (see Govern a Saved Question). Seeded built-ins are `status = system` and are read-only to experimenters. One parameterized saved question serves every value: “results for model X” is prepared once, then bound to Opus, GPT-5, and so on. That is one button, not one per model.

writes `nl_query` (only when you Save a new question)

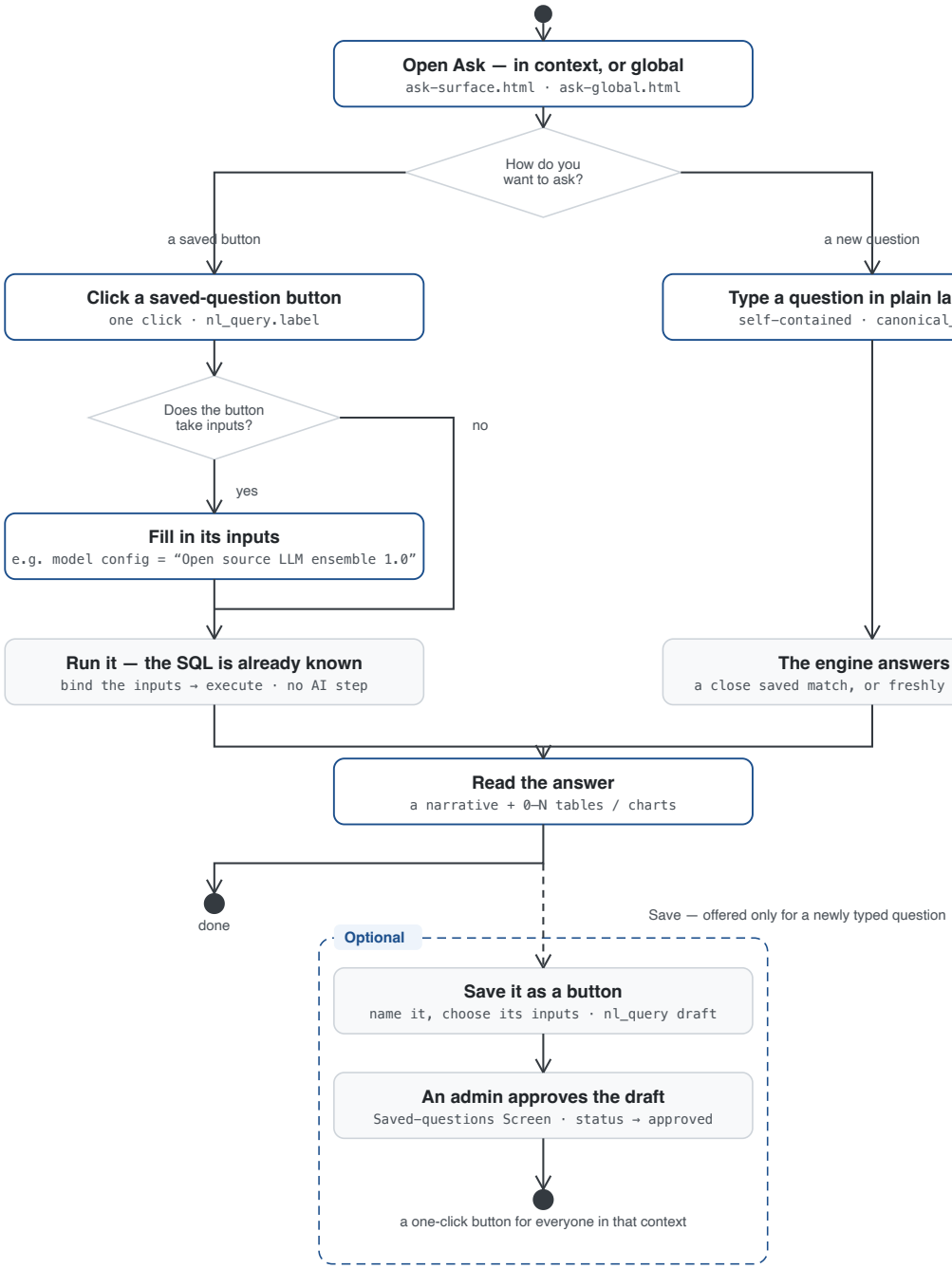
Variations and exceptions

IF…	THEN…
The saved button they click has no inputs (for example, “Top 5 models”)	It runs on the click; no form is shown.
The saved button takes inputs	A small form asks for them first; the values are bound into the stored SQL and it runs.
They type a question that matches a saved one (similarity at or above `ASK_MATCH_THRESHOLD`)	The engine reuses that question's stored SQL, lifting the values from their words. No inputs form, and no Save offered — the question already exists.
The typed question matches nothing	The agent compiles fresh read-only SQL and answers; Save is offered alongside the answer.
The typed question asks for a change — an insert, update, delete, or anything that is not a read	Refused. Only `SELECT` and `WITH` are compiled or run; the answer says the engine only reads.
The words cannot be mapped to safe SQL, or ask about data the schema does not hold	The engine says so and answers with narrative only, no tables, rather than guessing.
A matched saved question's `schema_fingerprint` is stale — the tables or columns it reads have changed	Its SQL is regenerated from `canonical_prompt` and flagged for re-approval; it is not run silently.
A saved question has been retired by an admin (`retired_at` set)	It is absent from every Ask surface and from matching until an admin restores it.
A result exceeds `ASK_ROW_CAP` rows	The first `ASK_ROW_CAP` rows are returned, marked truncated; the answer says so and asks them to narrow the question.

Acceptance test — *Given* a signed-in asker on the Ask surface, *When* they click a parameterized saved button and fill its inputs, *Then* it runs with those inputs bound, returns a narrative plus zero or more downloadable tables and/or charts, and writes no `nl_query` row; *and When* they instead type a new question and click Save, *Then* one `nl_query` draft is created that an admin can approve into a shared button; *and* every query, reused or freshly compiled, runs under the asker's own row-level security, so no answer contains a row they could not see on a screen.

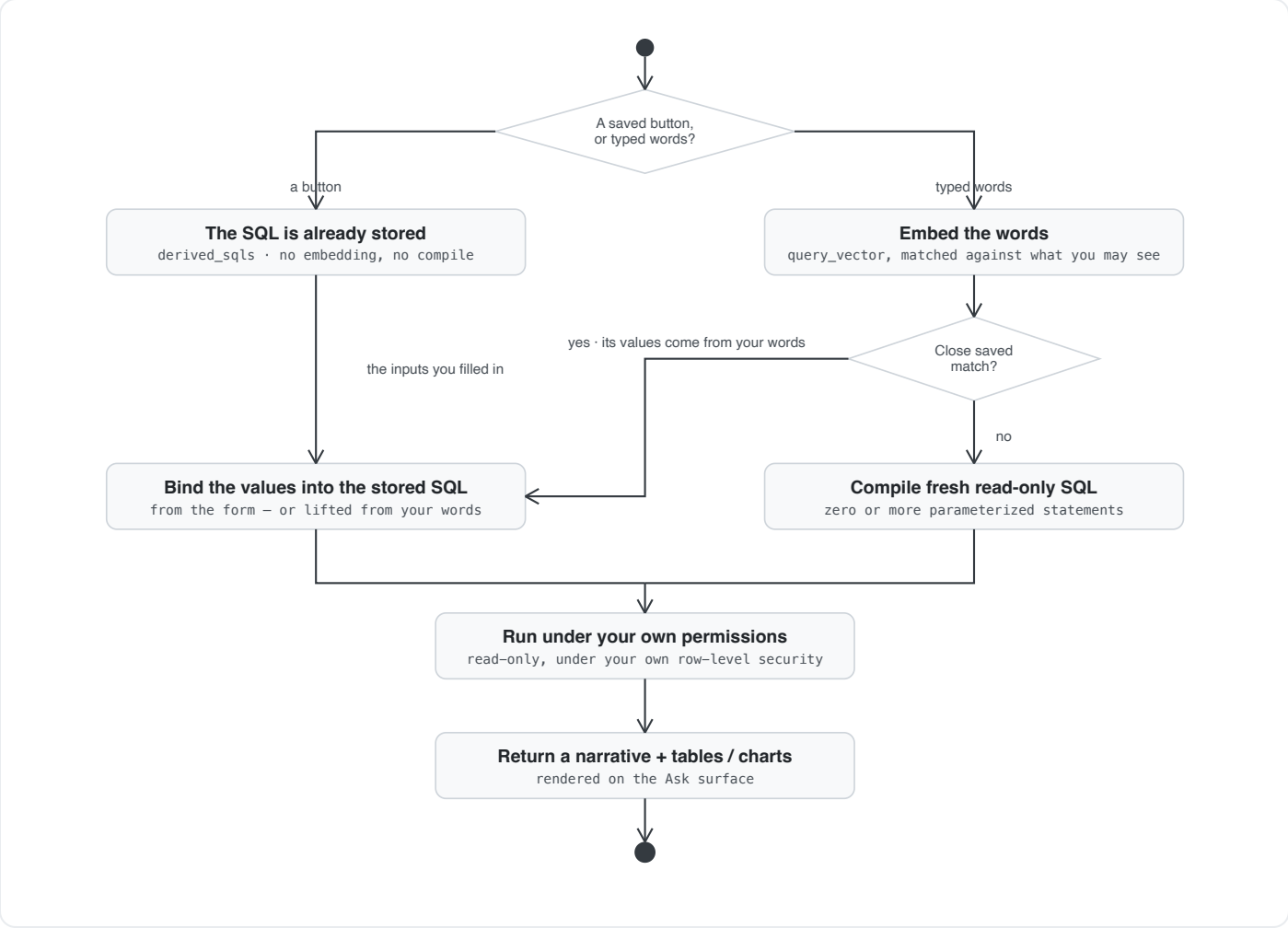
2 · The Journey — Pick a Saved Question, or Type a New One

Two entry points that meet at the answer. A button shows an inputs form only if it takes inputs; a typed question is complete in itself. Save appears alongside the answer, only for a newly typed question.



3 · Behind the Scenes — How a Question Becomes an Answer

A button already knows its SQL. Typed words are embedded; a close match reuses stored SQL with the values lifted from the words, and only an unmatched question is compiled.



A matched typed question and a clicked button take the same bind step — the difference is only where the values come from. Every run is scoped by row-level security and read-only.

4 · In Plain English

The Ask panel answers questions about the data in plain language, and most of the time you do not even type. The common questions are already there as labeled buttons, such as “Runs over budget” and “Groundedness by nudge.” You click one and read the answer.

Some buttons ask for a detail first, such as a model name or a threshold, so one button like “Results for a model” works for every model: you pick the model, and it fills the blank and runs. Buttons with nothing to fill in just answer on the click. None of this involves the AI or any typing, because the button already knows its query.

If nothing fits, you type your own question the way you would say it out loud. The app checks whether a saved question already covers it. If not, it quietly turns your words into a safe read-only lookup, runs it as you (so you only see what you are allowed to), and shows a written answer plus any tables, each of which you can flip to a chart.

A brand-new question is not kept unless you Save it. You give it a button name and decide whether a value in it should become a fill-in-the-blank. Your saved questions start private, and you look after them yourself on the Saved-questions page. An admin can promote the good ones into shared buttons, take one back if it stops earning its place, and a handful ship built-in from day one.

Fits the schema cleanly. A saved question is one `nl_query` row: `label` (the button caption), `canonical_prompt` (your normalized words), `derived_sqls` (zero or more parameterized statements), `params` (the typed inputs the button prompts for), `status` (draft, approved, or system), and `query_vector` for matching typed questions. Clicking a button binds `params` into `derived_sqls` and runs, with no write. The compile-and-run engine is middle-tier.