SumizAI .com Your Second Brain, Powered by Intelligence.
https://sumizai.com/blog/how-search-works-in-sumizai.html

UNDER THE HOOD

What does SumizAI search when you search?

Search is where a personal knowledge base is judged, because it is the only feature you use while frustrated. **SumizAI indexes the whole note** — including the original conversation — and returns the fragment that matched rather than a filename.

Updated: 2026-08-14 **8** questions 4 min read

Q. What parts of a note are searchable?

Four, with different weights: the **title**, the **summary**, the **body**, and the **original question and answer** stored as source material.

That fourth one is the reason the search feels different in use. Half the time you do not remember what the note is called or how it was summarised — you remember a phrase from the conversation, an example the model used, a number it quoted. That text is indexed too.

The weighting means a title match still outranks a passing mention deep in a transcript, so precision is not sacrificed to get the recall.

Q. Does it handle inflection and missing accents?

Accents, yes and deliberately. Text is normalised before indexing, and the query is normalised with the identical expression — writing Polish without diacritics is normal typing, not a mistake.

That identical-expression detail is not pedantry. If the query normalised differently from the index, the database would quietly stop using the index and the search would get slow rather than wrong, which is the harder failure to notice.

Inflection is handled by prefix matching on candidates and context rather than by a stemming dictionary. The dictionary in use does not know Polish inflection, so *notatka* would never match *notatkami* — matching on the prefix closes that gap.

Q. Why show a fragment instead of just the note?

Because a list of titles makes you open five notes to find out which one you meant.

A result carries the matching fragment with your phrase highlighted, so you can tell from the list whether this is the note where you worked out the cost model or the one where you mentioned it in passing.

It also tells you something about the note's structure — a hit inside the source material reads differently from a hit in the summary, and you can see which you got.

Q. Can I search inside the table of contents?

Yes, and it behaves differently from list search on purpose. There is a search field above the chapter tree, and typing in it **narrows the tree to the branches that matched**.

So you get structure rather than a flat list: you can see that four of your hits are under one chapter and one is somewhere unrelated, which is often the actual answer to what you were looking for.

From a note or from the list of notes there is also a link straight back to the note's place in the table of contents, so you can go from a hit to its neighbourhood in one step.

Q. How does search relate to what the AI sees?

Closely. When you ask a question in a vault, the retrieval that finds the most relevant notes is the same machinery.

Your question travels with the table of contents and the full text of up to five notes, inside a 24 000-character budget. The five are chosen by relevance to the question.

There is one important difference in how the queries are built. Precise search uses *and* between your words, because you meant all of them. Finding candidates from a long piece of text uses *or*, because demanding every word appear would return nothing.

Q. What if search stops finding notes I know exist?

Reindex. The index is derived data — searchable columns computed from the note content — and the files are authoritative.

The reindex operation rebuilds the index from the contents of the storage. If the index and the files disagree, the file wins; there is no scenario in which reindexing loses a note.

There is a single code path that writes indexed content, precisely so no write can update three of the four searchable columns and quietly leave the fourth stale.

Q. Is search scoped to a vault?

Yes, like everything else. A vault is a boundary for search exactly as it is for the model's context.

This is enforced structurally rather than by remembering to add a filter: the repositories do not expose a way to query notes without an owner and a vault.

The practical effect is that search stays sharp as your total note count grows, because you are always searching one domain rather than everything you have ever written.

Q. What is search deliberately not doing?

It is not semantic. There are no embeddings and no vector database; retrieval is lexical, with trigram similarity to catch near-misses.

For this application that is the right trade. The semantic step happens where it is most valuable — a model reading the shortlisted candidates and judging them — rather than being spread across an index that has to be maintained, migrated and re-embedded every time the model changes.

It also keeps the whole thing fast and inspectable. You can see why a result matched.