SumizAI .com Your Second Brain, Powered by Intelligence.
https://sumizai.com/blog/how-sumizai-handles-duplicate-notes.html

UNDER THE HOOD

How does SumizAI stop your library filling with near-duplicates?

Ask a model about retrieval quality four times over three months and you get four notes about retrieval quality — each slightly better than the last, none of them complete, all of them findable. **That is the failure mode SumizAI is built to prevent**, and it costs one extra step before every save.

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

Q. What runs before a new note is saved?

A two-stage check, cheap first and expensive second.

Stage one is retrieval: full-text search and trigram similarity over the notes in the vault produce a shortlist of candidates. This is pure database work and takes milliseconds even over thousands of notes.

Stage two is judgement: an auxiliary model reads the new material against a candidate and returns **a score between 0 and 1, with a written reason**. The split matters — asking a model to compare against every note would be unaffordable, and asking a database to understand that two paragraphs make the same point does not work.

Q. Why use two different search methods for the candidates?

Because they fail in opposite directions and cover for each other.

Full-text search is precise about words and blind to the ones you did not type. Trigram similarity matches on character patterns, so it catches typos, inflected forms and near-misses that exact matching drops.

One detail that matters for inflected languages: text is normalised for the index — accents stripped — and the query uses exactly the same normalisation. Writing Polish without diacritics is the rule rather than a typo, and if the two sides disagreed the database would silently stop using the index.

Q. What does the score actually mean?

It is a judgement of whether these two things are *the same topic*, not whether they share vocabulary. That is the job you cannot hand to a database.

The written justification is as useful as the number. “Both describe the chunking trade-off, the new one adds cost figures” tells you what a merge would do; “both mention chunking but one is about tooling” tells you why it should not happen.

The score is compared against the vault's threshold. Above it, the vault's mode decides what happens next.

Q. What are the three modes, and when is each right?

Ask is the default and shows you the candidate with its score and reason. The decision can be deferred — it does not block you from carrying on. Use it while you are learning what the app considers the same topic.

Always merge folds new material in automatically. This suits a vault that is a long study of a small number of concepts, where the outcome you want is a handful of deep notes rather than a hundred shallow ones.

Always create keeps everything separate. This suits a vault that is a log, where two entries about the same subject on different days are genuinely two entries.

The threshold applies in every mode, because it is the definition of “the same topic” that the mode then acts on.

Q. What does a merge do, precisely?

It appends. The new material joins the existing note, and the new question and answer are added to that note's source material alongside the ones already there.

It never edits or deletes what was there. This is a structural guarantee rather than a promise: the service and the repository have no update or delete operation for source material at all, so no code path can do it by accident.

The result is a note that grows in depth and keeps its receipts. You can always read back what each contributing conversation actually said, in its own words, rather than a synthesis you have to take on trust.

Q. Why is append-only worth this much care?

Because a merged note is a claim assembled from several conversations, and an assembled claim is only as trustworthy as the record underneath it.

Without the originals, a merge is lossy in a way you cannot detect later. The summary reads fine, and the nuance that a particular conclusion only held for one specific configuration is gone — and you have no way of knowing it was ever there.

With them, the note has two layers: the current best understanding on top, and the evidence underneath. That is what makes it safe to keep merging for years.

Q. What happens to candidates that score below the threshold?

They are not discarded. A note that is related but not the same is exactly what a link is for, and near-misses from this stage feed the linking step.

So the check produces two useful outputs from one piece of work: the strong matches become merge decisions, and the weaker ones become the connections between notes.

This is why the threshold is worth tuning rather than maximising. Set it very high and you get duplicates; set it very low and you merge things that should have stayed distinct and lose the links that would have connected them.

Q. What if I merge two notes and regret it?

Retitle the merged note and, if you want them apart again, split it by hand — everything you need is still in the file, because the source material from both conversations is preserved in full.

This is the practical payoff of append-only: a merge is not a destructive edit, so the worst case is a note that has more in it than you wanted rather than one that has lost something.

And because notes are ordinary Markdown files in a folder, a split is a text-editor operation. You are not fighting a database schema to undo a decision.