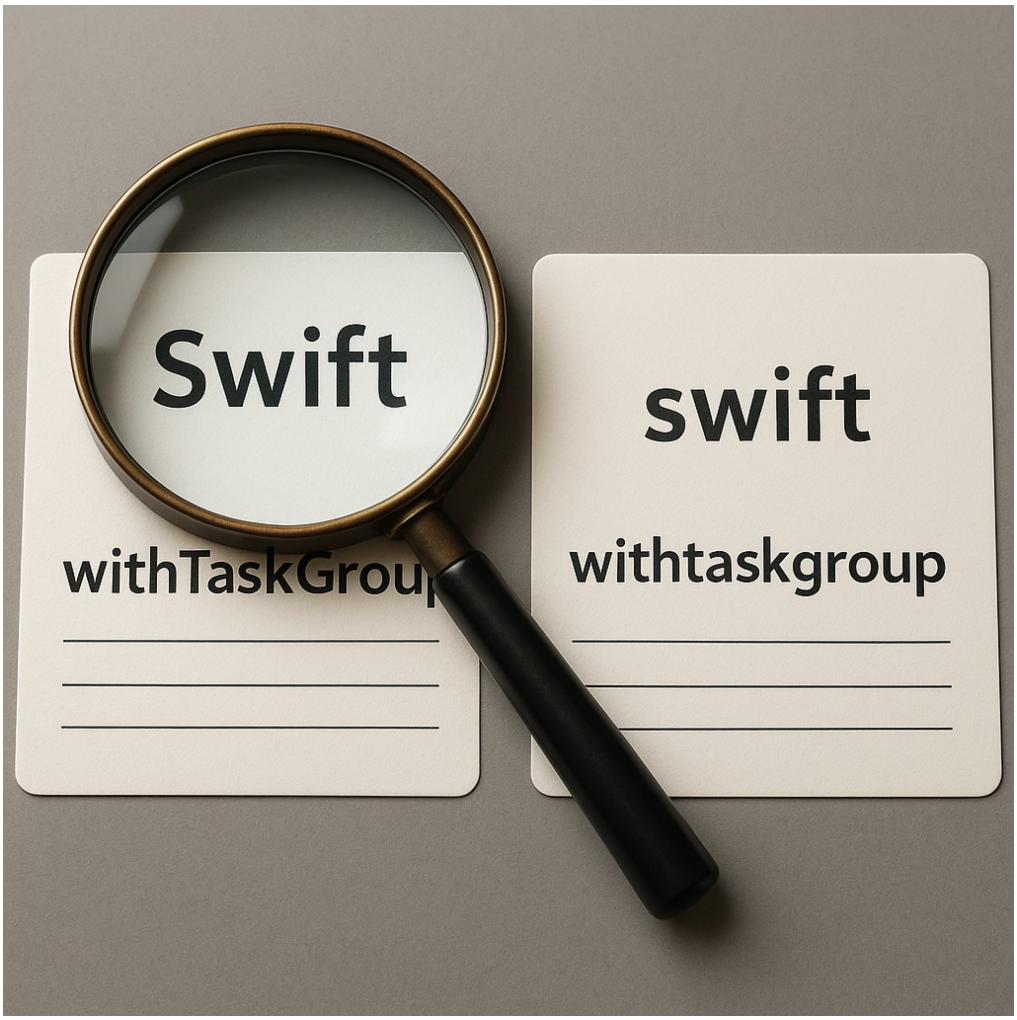

Cupertino v1.0.2: the duplicate that `LOWER(uri)` could not see

A few weeks ago I shipped cupertino v1.0.1 and closed the bug that was supposed to make it impossible for the same Apple page to appear twice in the local search index. The release notes carried a confident sentence: *"verified that the shipped v1.0.0 search.db has zero case-axis duplicate pairs."* The query that verified it returned empty across 405,782 rows. Done.

Today I shipped v1.0.2, which removes 128,142 of those rows. They were duplicates. They had been duplicates the whole time. The verification query was structurally incapable of seeing them.

Here is what happened, what v1.0.2 contains, and the three things I took away from the audit.

What v1.0.2 ships

Concrete numbers first.

	v1.0.0 / v1.0.1 bundle	v1.0.2 bundle	delta
`docs_metadata` rows	405,782	277,640	?128,142 (?31.6 %)
Frameworks	422	402	?20
`search.db` size	~3.2 GB	~2.4 GB	?0.8 GB (?25 %)
Case-axis duplicate clusters	61,257	0	gone

So a third of the database was the same Apple page indexed twice, once under `documentation/Swift/...` and once under `documentation/swift/...`, with slightly different URI hashes so the rows did not look identical to a casual audit.

The fix is one canonicalization call inside `URLUtilities.filename(_:)`. The bundle is a full reindex of 404,729 JSON pages, 12 hours 36 minutes wall-clock on an M4 Max. The schema version bumps from 12 to 13 so old databases get rejected at open with a "rebuild required" message that points at `cupertino setup`, which downloads the new clean bundle in seconds.

Two other things ride along in this release:

> **Post-redirect URL canonicalization** (#277), contributed by @Vignesh-Thangamariappan in PR #278. When Apple issues a 301/302 redirect (e.g. `professional_video_applications` to `professional-video-applications`), the crawler now files the page under the post-redirect URL instead of the request URL. Five regression tests plus an integration test against a mock redirect server. Thank you, Vignesh. This was a subtle bug to find and a careful PR to author.
>
> **HTML link augmentation for sparse-references pages** (#203, PR #281). The crawler now unions HTML `<a href>` links with DocC's JSON references on pages where the JSON link count is below a configurable threshold. Catches operator overloads, legacy numeric-ID symbols, REST sub-paths, and the handful of frameworks Apple serves only as HTML.

Earlier groundwork on `#200` (case-axis dedup at index time) came from @imwyvern, who supplied the crawler queue dedup, the dedup helper, and the URI alignment in PR #201. v1.0.2 builds on that work; the change here is finding the layer it missed.

The bug

Apple serves the same documentation page under multiple URL casings. The reference graph inside Apple's DocC JSON is sometimes `documentation/Swift/withTaskGroup(of:returning:isolation:body:)` and sometimes the all-lowercase variant. Cupertino treats both as crawl targets. Pre-#283, the indexer would also treat them as separate pages, producing two rows in `docs_metadata`, two rows in `docs_fts`, two answers in the search results.

Wesley's PR #201 in v1.0.1 fixed `URLUtilities.normalize(_:)` to lowercase the path, which was supposed to collapse these at queue time. It did, partially. There was a layer below the queue that #201 did not cover.

`URLUtilities.filename(from:)` is the function that turns a URL into an on-disk filename and, by extension, into the URI key used in `docs_metadata`. For URLs containing parens or colons (so, any Swift method signature with named parameters), it appends an 8-hex disambiguator hash. That hash was computed from the *raw* URL string, before the function's internal lowercasing. So the same Apple page, fetched from two case-variant URLs, produced two different hash suffixes:

```
documentation/Swift/withTaskGroup(of:returning:isolation:body:)
  -> documentation_swift_withtaskgroup_of_returning_isolation_body_b9f71892

documentation/swift/withtaskgroup(of:returning:isolation:body:)
  -> documentation_swift_withtaskgroup_of_returning_isolation_body_969a5400
```

Different filename, different URI, two rows in the database, two cards in your search results.

The verification trap

This is the part that bothers me.

The v1.0.1 closure of #200 was not careless. It included a verification query, run against the actual shipped DB, with the explicit goal of confirming the bug was gone:

```
SELECT LOWER(uri), COUNT(*)
  FROM docs_metadata
  GROUP BY LOWER(uri)
  HAVING COUNT(*) > 1;
-- (returned empty across 405,782 rows)
```

Empty result, ship it. Felt rigorous. Was wrong.

The query is structurally incapable of finding the bug. The URIs in `docs_metadata` carry an 8-hex suffix that varies per source URL, so the case-variant pair `documentation_swift_withtaskgroup_..._b9f71892` and `documentation_swift_withtaskgroup_..._969a5400` lowercase to two *different* strings. `GROUP BY LOWER(uri)` cannot collapse them. It returns empty. It will always return empty no matter how broken the dedup is, because the column itself is wrong for the question.

The correct query is on `docs_structured.url`, which preserves the original Apple URL with its original casing:

```
SELECT 'clusters', COUNT(*) FROM (
  SELECT LOWER(url) FROM docs_structured
    GROUP BY LOWER(url) HAVING COUNT(*) > 1
);
-- clusters | 61257
```

Two columns, two completely different stories. One verification you would write because the index key obviously *is* the URI. One verification you would write because that is the column a human would look at to spot the duplicate.

This generalizes. The verification query for a data correctness fix has to come from the column where the bug would *show up to a reader*, not the column the data is *keyed by*. They look interchangeable until you realize that the key has been transformed by code that may itself be where the bug lives. If you verify in the transformed space, you verify against the same transformation that may be broken.

A simpler version of the same lesson: ask yourself which column a customer would email you about, not which column the code uses as its primary key.

What I learned

Three takeaways, in order of how much they cost me.

1. "Already fixed in vN" claims with a clean test are very compelling and very capable of being wrong.

I do not think Wesley or I were reckless. The #200 fix was real and good. The verification query passed. The release notes were faithful to what the query said. Everyone behaved sensibly. The query was still the wrong query.

What I want to internalize: in any audit that asserts "we checked the data and it's clean", the question is not just "what did the query return", but "if the bug were present, would this exact query show it". For data bugs, that means: which column does the bug live in, and is that the column the verifier looked at.

For this PR, I added a docs entry that explicitly names the wrong query and the right query, side by side, as a permanent footnote. Anyone reading the v1.0.0 release notes in five years can follow the link to see why the empty result was misleading.

2. The migration you built and then deleted is still good engineering.

I spent a real amount of time on an in-place v12 to v13 migration. The plan was: walk `docs_metadata`, recompute each URI through the post-#283 helper, group by canonical URI, pick the survivor by latest `last_crawled`, delete losers, rename survivors across `docs_metadata` + `docs_structured` + `docs_fts` + `doc_symbols` + `doc_imports` + `doc_code_examples`, all in one transaction with FK off. 270 lines of code, nine unit tests, three integration tests against a v12 fixture DB. The unit tests passed. The integration test passed on a 10-row fixture in milliseconds.

Then I ran it against my real 405k-row search.db. First attempt: over an hour, never committed, journal at 314 MB, killed it. Second attempt with prepared-statement reuse: marginally faster, journal at 415 MB at the 1h mark, killed it. Third attempt with `PRAGMA journal_mode = MEMORY` plus `synchronous = OFF`: another hour, never observably finished.

Meanwhile, `cupertino setup` downloads the pre-built v1.0.2 bundle in about thirty seconds. That is the path. Always was. The architecture already had the answer; the migration was solving a problem that did not need to be solved.

I deleted the migration before the tag. All 270 lines, all twelve tests. The CHANGELOG notes the draft existed and was removed, because that fact matters for anyone wondering why the codebase has a v12-to-v13 throw block but no migration body to call. The bug fix lives entirely in `URLUtilities.filename(_:)`. The data fix lives entirely in the new bundle. The schema bump is a one-line `Int32` that forces the upgrade path.

What I learned: write the migration code if you have to think through the algorithm, but ship the path that the architecture wants you to ship. The investment was not wasted. It was the thing that forced me to fully understand what the dedup boundary actually looks like across all six tables that hold URIs.

3. Reindexing 404,729 pages costs you an evening of wall-clock and removes one third of your data.

The bundle reindex took 12 hours 36 minutes on an M4 Max. I watched a stale-build SwiftPM lock kill three attempts before the fourth one ran clean. I babysat the cupertino-rel binary signing through Apple's notarization queue. I watched the search.db file grow from 425 MB to 2.41 GB in

27-second progress increments, all night.

The first time I ran `cupertino search "withTaskGroup"` against the new bundle it returned a single result, lowercase, canonical. The same query against the v1.0.1 bundle returns two: one capital S, one lowercase. That moment is worth twelve hours.

What also changed in the cupertino skill

The cupertino skill (the bundle of instructions that ships in `skills/cupertino/SKILL.md` for LLMs that integrate with cupertino) got a "Two rules" block at the top of its body. The previous version had query strategy + verification guidance scattered throughout. The new version makes the two things that matter the first thing the calling model sees:

1. Any Apple-related question goes through `cupertino search` first. Do not reach for training-data memory of Apple APIs.
2. After drafting an answer, verify that the code exists on Apple **and** that the pattern is appropriate. Existence catches invented method names. Appropriateness catches deprecated symbols that cupertino indexes alongside current ones (e.g. `UIWebView` when `WKWebView` exists).

The token cost of doing this is small (about 5 % overhead for cite-as-you-go, a few hundred tokens per verification re-search) and the failure mode it prevents is the most common LLM Apple-API hallucination class.

Upgrade

If you already have cupertino installed:

```
brew update && brew upgrade cupertino
cupertino setup
```

The first command pulls the new universal binary (signed and notarized via the GitHub Actions release workflow). The second downloads the new database bundle (669 MB compressed, 3.3 GB on disk). Your old v12 search.db will be rejected at open with a "rebuild required" message and replaced.

If you are installing for the first time:

```
bash <(curl -sSL
https://raw.githubusercontent.com/mihaelamj/cupertino/main/install.sh)
```

Thanks

To @Vignesh-Thangamariappan for PR #278 (post-redirect canonicalization, #277). To @imwyvern for the upstream #200 work that v1.0.2 builds on. To everyone who filed issues against the v1.0.0 corpus and made #283 surface in the first place.

Cupertino is free, open-source, local-first, and built by one person on her own time.

Built by Mihaela. I also work on Codeweaver.