

If you work with entity data long enough, you run into the same practical problem over and over. A name in a local record looks familiar, but not familiar enough. Is this the right person, place, company, or work? Does the record map cleanly to a Wikidata item, or are there two plausible candidates with nearly identical labels? Can you justify the link later, when someone asks why you matched this row to that QID?

That is the niche where the open source project often described as MCP for Google Knowledge Graph and Wikidata becomes useful. It is an MCP server and CLI called “Wikidata + Google Knowledge Graph MCP,” published on Smithery under [revanalex/wikidata-google-knowledge-mcp](https://smithery.io/revanalex/wikidata-google-knowledge-mcp/), with an MIT license. Its purpose is narrow in the best way: help AI agents search Wikidata, inspect selected facts, and resolve local records to Wikidata QIDs with evidence you can review and uncertainty made explicit when the evidence is not strong enough.

That focus matters. Many knowledge graph tools promise broad discovery, but day-to-day data work usually needs discipline more than breadth. You want a small candidate set, not a dump of every vaguely similar result. You want enough facts to decide, not an uncontrolled scrape of everything attached to an entity. You want a system that can say “hold” instead of bluffing confidence. This project leans into that mindset.

What this MCP server is actually for

The easiest way to understand the project is to stop thinking about it as a giant graph browser and start thinking about it as a resolver with guardrails. It can be used in MCP clients such as Claude Code, Cursor, and Codex. It supports searching Wikidata, fetching focused details about an entity, exploring related entities, checking service status, and attempting a deterministic resolution from your local record to a Wikidata item.

That last part is the heart of it. The server does not simply return a ranked pile of candidates and leave you there. It uses explicit outcomes such as `AUTO_MATCH`, `HOLD`, `AMBIGUOUS`, and `NO_CANDIDATE`. In practical terms, that gives you a language for the result. You are not forced into a yes or no when the underlying evidence does not justify one.

I have found that this kind of explicit uncertainty is what separates a useful matching workflow from a dangerous one. In real data, ambiguity is normal. Two films can share a title. A person may publish under multiple names. A local record may be missing the exact detail that would break a tie. A resolver that pretends otherwise creates cleanup work later.

Why the combination of Wikidata and Google Knowledge Graph is interesting

Wikidata is already a strong source for structured entity resolution. It is broad, public, and familiar to anyone who has worked with open linked data. Wikidata’s own MCP documentation also frames its MCP offering as a way for LLMs to explore and query Wikidata programmatically through the Wikidata API and Wikidata Query Service. So there is already a wider ecosystem forming around MCP for Wikidata.

What makes this project distinctive is that it pairs Wikidata workflows with an optional Google Knowledge Graph cross-check. Optional is the key word. Wikidata requires no account or API key in this setup. The Google Knowledge Graph Search API is not mandatory. That is a healthy design choice because it lets you start with the open source, zero-key path, then add a second signal only if your use case benefits from it.

The project is careful about how it treats that second signal. Google and Wikidata agreement is handled as provider concordance, not proof of identity. That sounds subtle, but it is exactly the right call. Two providers lining up on the same entity can increase confidence, especially when specific IDs align, but it is still not absolute proof. Anyone who has debugged entity mappings knows that “two systems agree” and “this is definitely the same thing” are not always identical statements.

The documented cross-check uses exact ID joins, specifically /m/ for Wikidata property P646 and /g/ for P2671. That gives the comparison a concrete basis. It is not vague semantic similarity. It is an exact join on identifiers that are already represented in Wikidata. Even so, the project keeps the epistemic brakes on, which is good engineering and good editorial judgment.

The design choice that will save you time: bounded search

One of the smartest details in the project is also one of the simplest. Search is bounded. By default, it returns 3 candidates, with up to 5, rather than handing back a large raw result set.

That may sound restrictive until you have spent a week reviewing fuzzy matches. Big result lists create a false sense of completeness while making human review slower. Small candidate sets push the resolver toward a clearer decision surface. Either the best candidate is visible quickly, or the case needs to be held for more evidence.

For local cataloging, CRM enrichment, editorial metadata, and similar workloads, this is usually what you want. The costly cases are not the obvious matches. The costly cases are the nearly-right ones, where too many candidates turn a quick review into a scavenger hunt. Bounded search keeps the system honest.

There is also a side benefit for agentic workflows. An MCP tool that emits a concise, inspectable set of choices is easier for an AI client to reason over than a flood of loosely relevant hits. Less noise means fewer accidental leaps.

The toolset, in plain English

The server exposes a focused set of documented MCP tools, and the CLI extends that with batch and evidence export capabilities. You do not need to memorize the names to understand the workflow, but it helps to know the shape of the toolbox.

- `kg_search` searches for candidate entities.
- `kg_entity` retrieves selected facts for a specific entity.
- `kg_related` explores related entities.
- `kg_resolve` attempts deterministic resolution from a local record to a Wikidata match.
- `kg_status` checks service status.

That is a compact surface area, and I think that is part of the appeal. A lot of graph tooling grows sideways until every action has three variants and five optional interpretation modes. Here, the verbs are straightforward. Search, inspect, relate, resolve, check status.

The CLI matters too. Batch work is where entity resolution stops being a demo and starts being operations. If you need to process a file of local names and export evidence for review, a command-line path is often the difference between “interesting” and “deployable.”

A sensible way to start, especially if you are new to this space

The best first experience is not to begin with a messy dataset. Start with a handful of records you already understand. Pick examples where the correct Wikidata item is obvious to you, then a few where you know ambiguity exists. That gives you a feel for how the resolver behaves when the answer is easy versus when it should hesitate.

A good starter sequence looks like this:

1. Use `kg_search` on a known entity and look at the small candidate set.
2. Use `kg_entity` to inspect selected facts for the most likely item.
3. Try `kg_resolve` on a local record and compare the explicit outcome with your own judgment.
4. If needed, add the optional Google cross-check and see whether it supports or fails to support the same mapping.
5. Export evidence when you need a review trail for other people.

That sequence teaches the right habits. You learn to treat search as a starting point, entity facts as decision support, and resolution outcomes as accountable states rather than mysterious scores.

What “selected facts” buys you in practice

The phrase “selected-fact retrieval” can sound minor until you have worked with large graph payloads. In this project, fact retrieval can include ranks, qualifiers, and references on request. That is exactly the sort of detail that makes an entity usable in a professional setting.

Suppose you are comparing two candidate entities with similar labels. A plain label and description may not settle it. Qualifiers can reveal timeframe, role, or geographic scope. Ranks can hint at preferred versus deprecated statements. References can tell you whether an assertion is lightly supported or well anchored within Wikidata’s model.

Even without inventing a specific domain scenario, you can see the pattern. Resolution work rarely fails because there is no data at all. It fails because the most important disambiguating detail is buried. A selected-fact approach surfaces the details that matter without forcing you to traverse the entire item manually.

There is also a governance angle here. If your team needs to defend a match, “the model thought so” is not enough. A fact set with qualifiers and references gives you something inspectable. You can show what was considered and where uncertainty remained.

Understanding the outcome states without overcomplicating them

The deterministic outcomes deserve more attention because they shape how you should integrate the server into real workflows.

`AUTO_MATCH` is the comfortable case. The resolver found enough evidence to make the link automatically.

`HOLD` is underrated. It tells you there may be a candidate, but the system should not finalize the mapping yet. In mature operations, holds are not failures. They are quality control working as intended.

`AMBIGUOUS` is different from hold in tone and in consequence. It signals that multiple plausible candidates remain in play. That often means a human needs a discriminating detail that the current record does not contain.

NO_CANDIDATE is also useful. It spares you from false matches created by the pressure to map everything to something.

What I like about this set is that it avoids performative certainty. It creates clean branches for downstream handling. Auto matched records can move forward. Held or ambiguous records can enter review. No-candidate records can be left unmapped without pretending they belong somewhere.

That kind of state discipline matters more than fancy ranking language. A lot of data debt begins when systems fail to distinguish “not enough evidence” from “probably yes.”

Where the optional Google check fits, and where it does not

It is tempting to treat the Google side of MCP for google knowledge graph and wikidata as a magic confidence booster. That would be a mistake. The project’s own framing is much more careful, and that restraint is one of its strengths.

The optional Google Knowledge Graph Search API can serve as a corroborating source when exact IDs align through the documented joins. If a Wikidata entity carries the relevant identifier and the Google side lines up through that path, you have stronger provider concordance. That may help move a borderline case toward confidence.

Still, concordance is not proof. It should not override contradictory local evidence. If your local record says one thing and the cross-check implies another, the right outcome may still be HOLD or AMBIGUOUS. Systems become unreliable when secondary evidence is used as a trump card instead of a support signal.

A good mental model is this: use the Google layer to strengthen a case, not to rescue a weak one. If the underlying local record is too sparse or too messy, an extra provider does not automatically solve the problem. It just gives you another angle to inspect.

What this project is not

The boundaries are refreshingly explicit. The server is not official Wikimedia or Google software. It is not an export of the Google Knowledge Graph. It is read-only. It does not edit Wikidata, Google, or user data.

Those disclaimers are not legal filler. They tell you how to position the tool operationally. This is a resolution and inspection layer, not a synchronization system and not a write-back pipeline. If your workflow needs actual edits in Wikidata or updates to internal systems, that belongs elsewhere in your architecture.

Read-only behavior is often a virtue during adoption. Teams are much more willing to test a resolver when it cannot mutate source systems. You can evaluate match quality, export evidence, and measure ambiguity rates before you decide how mappings should be persisted downstream.

A practical reading of the CLI angle

The CLI’s batch and evidence-export commands are easy to underestimate, but they reveal the project’s real audience. This is not only for one-off interactive lookups in an MCP client. It is also for repeated, traceable record handling.

In most organizations, the hard part of entity resolution is not finding one correct QID for one famous entity. It is processing hundreds or thousands of mixed-quality records and leaving an audit trail that another analyst can inspect later. Evidence export addresses that operational need directly.

That kind of workflow also changes how you think about success. You are not aiming for a dramatic all-or-nothing automation rate. You are aiming for a healthy split where clear matches are handled deterministically, uncertain cases are isolated quickly, and reviewers spend time only where human judgment is actually needed.

Trade-offs you should expect before you adopt it

Every resolver reflects trade-offs, and this project is no exception. Its bounded search means you get tighter candidate sets, but you also accept that some long-tail or poorly labeled entities may require more deliberate follow-up. In practice, I would still choose bounded search for most production use because review efficiency tends to matter more than speculative recall.

The selected-fact model is similarly pragmatic. You get focused evidence, ranks, qualifiers, and references on request, but you are not treating the tool as a full graph dump. That keeps interactions leaner, especially for agent workflows, though it means you should know when a case needs deeper manual investigation outside the tool.

The deterministic outcome states are excellent for workflow design, but they may feel conservative if you are used to looser search products that always hand you something. Conservative is usually better when the cost of a wrong entity link is high.

The optional Google component introduces another trade-off. It can add corroboration, but it is still just corroboration. If you expect it to function like a final arbiter, you will be disappointed, and you should be. The project is right not to claim more.

How this fits into the broader MCP ecosystem

There is now a broader story around MCP for Wikidata, with Wikidata's own documentation describing standardized tools for LLMs to explore and query Wikidata programmatically. This project sits slightly differently. It is not trying to be the universal gateway to everything in Wikidata. It is oriented toward practical search, selected fact inspection, related entity exploration, and most importantly, resolution to QIDs with inspectable evidence.

That makes it a good fit when your end goal is a defensible entity link rather than general graph exploration. If your use case is "help an agent browse Wikidata broadly," you may compare multiple options in the ecosystem. If your use case is "map this local record to the correct Wikidata item and tell me when not to trust the match," the value proposition here is sharper.

I also think the publication context says something useful. Being published **here** on Smithery and designed for clients like Claude Code, Cursor, and Codex puts it directly in the path of how many teams are now experimenting with agent tooling. That matters because entity resolution is one of the tasks where agents need strong boundaries. A resolver that emits compact candidates, selected evidence, and explicit uncertainty fits that environment well.

A realistic first use case

If I were introducing a team to MCP for google knowledge graph and wikidata, I would not sell it as a grand knowledge integration platform. I would frame it as a careful assistant for linking local records to Wikidata, with optional provider concordance through Google when available.

The first pilot would be modest. Take a small sample of records, run them through search and resolve, inspect the evidence on a subset, and note how often you see each outcome state. Pay special attention to HOLD and AMBIGUOUS. Those cases teach you the limits of your local data quality faster than any marketing copy ever will.

That is also where MCP for wikidata becomes more than a phrase. You are not just plugging a language model into a graph. You are giving it a constrained protocol and a resolver that knows when to stop. In my experience, that restraint is what turns an interesting prototype into a trustworthy piece of infrastructure.

The project's choices reflect a mature understanding of real entity work: bounded search instead of candidate sprawl, selected facts instead of indiscriminate payloads, deterministic outcomes instead of hand-wavy confidence, optional Google concordance instead of overclaiming identity, and read-only behavior instead of risky automation. For anyone getting started, those are the right defaults to learn from.