

Record linking [Wikidata MCP lookup](#) sounds simple until you have to do it at scale, under scrutiny, with records that were never designed to line up cleanly. A person name can point to five plausible entities. An organization can change names, merge, split, or show up in one source under an acronym and in another under a formal legal title. A place can have variants across languages. Once you start linking local records to Wikidata QIDs, the real job is not search alone. It is judgment, evidence, and restraint.

That is where `kg_resolve` stands out in the Wikidata + Google Knowledge Graph MCP project. It is not just another search wrapper that floods an agent with candidates and hopes the model improvises. It is a resolution tool designed for one of the hardest parts of knowledge integration: deciding whether a local record should link to a specific Wikidata item, and making that decision in a way a human can inspect later.

The project itself is an open source MCP server and CLI, published as “Wikidata + Google Knowledge Graph MCP.” Its stated purpose is direct and practical: help AI agents search Wikidata, read selected facts, and link local records to Wikidata QIDs with inspectable evidence and explicit uncertainty when evidence is insufficient. That last phrase matters more than it first appears. Most bad linking systems fail not because they cannot find candidates, but because they act too confident when they should pause.

Why record linking needs more than search

If you have ever tried to reconcile a local catalog against Wikidata, you know the failure mode. Search returns a broad set of names that look plausible. The top hit often feels “good enough,” especially when an agent wants to complete a workflow. But “good enough” is not the standard for identity resolution.

Linking a local record to the wrong QID is expensive. It can contaminate downstream enrichment, leak incorrect metadata into user-facing systems, and create cleanup work that costs far more than the original link saved. In practical terms, one false positive can spread through indexing, recommendations, analytics, and internal joins before anyone notices. Search helps you discover options. Resolution decides whether any option is strong enough to trust.

The design of `kg_resolve` reflects that distinction. Instead of behaving like a broad retrieval function, it uses deterministic resolution logic with explicit outcomes. The documented outcomes are `AUTO_MATCH`, `HOLD`, `AMBIGUOUS`, and `NO_CANDIDATE`. Those labels are useful because they keep the task grounded. You are not pretending every lookup can end in a neat success. Sometimes there is a credible match. Sometimes there are several. Sometimes there is none. Sometimes the evidence exists, but not enough of it lines up.

That discipline makes a large difference in MCP for Wikidata workflows, where an agent may be operating inside Claude Code, Cursor, or Codex and trying to resolve entities as part of a larger process. A tool that says “I found some nearby names” is one thing. A tool that says “this record should be held for review because the evidence does not clear the threshold” is much more valuable.

What `kg_resolve` is doing in the MCP toolset

The documented tools in the project include `kg_search`, `kg_entity`, `kg_related`, `kg_resolve`, and `kg_status`. Taken together, they map to a sensible workflow.

`kg_search` helps surface likely entities. `kg_entity` reads selected facts about a known item. `kg_related` expands context around an item. `kg_status` gives health or service context. Then `kg_resolve` sits at the decision point, where a local record must either be linked, deferred, or rejected.

In practice, that matters because record linking is rarely a one-call problem. An agent may first search, then inspect a small fact set, then attempt resolution. Or it may go straight to resolution if the local record has enough structured information. The key is that resolution is not left implicit. It has its own tool and its own outcome model.

There is also a workflow advantage in having this exposed through MCP. The broader Wikidata MCP ecosystem is about giving LLMs standardized tools to explore and query Wikidata programmatically. This project adds a sharper operational use case: link records to Wikidata QIDs with evidence and bounded uncertainty, rather than treating the graph as a purely exploratory knowledge source.

The quiet strength of bounded search

One design choice in this project deserves more attention than it usually gets: bounded search. By default, the server returns three candidates, with up to five, rather than dumping a large raw result set.

That may sound like a small implementation detail. It is not. In entity linking work, too many candidates can actually make an agent worse. Large result sets encourage shallow pattern matching. The model sees a familiar label, latches onto it, and rationalizes the rest. A narrower candidate set forces higher quality ranking and sharper comparison.

I have seen this pattern in real reconciliation work, even outside MCP environments. Human reviewers often perform better when they compare three strong options than when they sift through thirty mediocre ones. The same principle applies here. Bounded search turns the problem from open-ended browsing into focused disambiguation.

This is especially relevant for MCP for google knowledge graph and wikidata use cases, where users may be tempted to treat every external source as additive proof. More candidates from more systems can feel reassuring. Often it just adds noise. The better approach is to present a few plausible entities, inspect specific facts, and make the uncertainty explicit when the match is not solid.

Deterministic outcomes are more important than they sound

The phrase “deterministic resolution logic” may read as dry implementation language, but it is one of the strongest signals in the project. Determinism means the tool is not improvising from call to call. Given the same inputs and context, you should expect the same resolution outcome.

That matters in two ways. First, it improves auditability. If a record was marked AMBIGUOUS yesterday and someone asks why, you need an answer that does not depend on a model’s shifting wording. Second, it supports batch processing. The CLI provides batch and evidence export commands, so consistency across many records is not a luxury. It is the baseline requirement.

The documented outcomes also encode good operational judgment:

- AUTO_MATCH means the evidence is strong enough for an automatic link.
- HOLD means the process should stop short of automatic assignment.
- AMBIGUOUS means more than one candidate remains plausible.
- NO_CANDIDATE means nothing credible surfaced for the record.

Those distinctions may look obvious on paper, but many systems collapse them into a single ranked answer. That is how bad links get normalized. A resolver that can say “no” or “not yet” is usually safer than one that always returns its favorite guess.

Evidence is the real product

The project emphasizes inspectable evidence. That wording is exactly right. In serious linking work, the link itself is not enough. You need to know why the link was made.

This is where the selected-fact retrieval support becomes useful. The server can retrieve selected facts, including ranks, qualifiers, and references on request. That may seem like a detail aimed at power users, but it gets at the heart of confidence. A plain property value can be helpful. A property with its rank, qualifiers, and references tells you much more about whether the statement is current, contextualized, or well supported.

Suppose an agent is trying to link a local organization record. A name match alone may be weak. But if selected facts align on expected type and supporting details, confidence increases. If qualifiers or references reveal a contextual mismatch, confidence drops. The point is not that `kg_resolve` magically knows truth. The point is that it operates in an environment where the facts behind the decision can be surfaced and reviewed.

That inspectability becomes even more valuable when the result is `HOLD` or `AMBIGUOUS`. A human reviewer does not want a vague explanation like “low confidence.” They want to see what candidates were considered, what facts aligned, and what remained unresolved. The project’s emphasis suggests exactly that style of workflow: machine-assisted linking, with enough evidence to support human oversight.

How the optional Google cross-check fits, and where it does not

One of the more careful parts of the project documentation is the optional Google cross-check. The server can use exact ID joins through `/m/` for Wikidata property `P646` and `/g/` for `P2671`. That is a precise statement, and it avoids a common mistake.

The project explicitly treats agreement between Google and Wikidata as provider concordance, not proof of identity. That distinction is easy to miss if you have not worked through messy identity data before. Two providers agreeing can be a strong sign. It is not the same thing as independently proving the local record refers to that entity.

That is why the phrase MCP for google knowledge graph should be used carefully here. The project is not an export of the Google Knowledge Graph, and it is not official software from Wikimedia or Google. The Google side is optional. Wikidata requires no account or API key, while the Google Knowledge Graph Search API is optional. In other words, the resolver is centered on Wikidata linking and can add a Google-based cross-check when the right identifiers are available.

That design is sensible. Exact joins on known external IDs are useful because they are concrete. They are not fuzzy semantic comparisons pretending to be certainty. At the same time, the documentation avoids overselling them. Concordance between providers can support confidence. It should not substitute for entity resolution logic.

In practice, this helps prevent a familiar error. Teams often treat a second source match as if it closes the case. But if both sources reflect the same historical confusion, duplication, or mismatch, agreement can be misleading. The project’s framing is cautious, and that caution is earned.

Why read-only matters for trust

There is another point in the documentation that may seem administrative but actually shapes how the tool can be safely adopted: it is read-only. The project does not edit Wikidata, Google, or user data.

That matters because record linking has very different risk profiles depending on whether the tool only reads data or also writes back. A read-only resolver can be inserted into review workflows, batch analyses, and enrichment pipelines with fewer governance concerns. It can recommend or assign a QID in your local process without mutating external systems.

From an operational perspective, read-only tooling is often the right first layer. It lets a team build confidence in linking quality before any thought of writeback enters the picture. Since this project is focused on search, fact retrieval, and linking support, the decision to stay read-only keeps the scope tight and defensible.

A realistic linking workflow with `kg_resolve`

The most productive way to think about `kg_resolve` is not as a magic linker, but as the decision engine inside a broader reconciliation loop. A local record comes in, the system looks for bounded candidates, selected facts are available when needed, and the resolver returns a structured outcome that determines the next action.

A practical flow often looks like this:

- a local record is submitted with the fields available in that local system
- the resolver evaluates bounded candidates against Wikidata
- selected facts are pulled when the match needs closer inspection
- the outcome routes the record to auto-linking, review, or rejection
- evidence is exported for audit or batch follow-up when needed

This is the kind of workflow that works in production because it respects uncertainty. It also maps cleanly onto MCP clients where an agent may need to combine search, retrieval, and decision-making without losing track of what was actually established.

One subtle advantage of this structure is that `NO_CANDIDATE` is not a failure state in the same way many people assume. Often it is exactly the right answer. If the local record cannot be credibly tied to an existing Wikidata item from the evidence at hand, the safest output is to say so. That preserves data quality and avoids a false sense of completeness.

Where `kg_resolve` helps most

Not every entity linking problem benefits equally from this sort of resolver. The project is particularly well suited to cases where local records are meaningful enough to compare against Wikidata, but not so richly identified that a trivial exact ID match already exists.

That middle zone is common. Internal collections, research datasets, editorial systems, and content repositories often have partial metadata: a title or label, maybe a type, perhaps one or two descriptive fields. Those records are exactly where naive search can overreach and where deterministic resolution with evidence can save time.

It also helps in environments where decisions need to be explainable to someone other than the original operator. If a cataloging team, product owner, or data steward asks why a QID was attached, “the model

thought it looked right” will not survive the meeting. A clear outcome model plus inspectable evidence usually will.

For teams exploring MCP for wikidata specifically, this is an important distinction. Basic query access to Wikidata is valuable, but it does not automatically solve local reconciliation. `kg_resolve` addresses that narrower, tougher operational problem.

Edge cases that still require judgment

No resolver eliminates edge cases. The project documentation does not claim to do so, and that is a strength. Based on the verified features, there are several classes of hard cases where a `HOLD` or `AMBIGUOUS` result should be expected and respected.

The first is genuine name collision. Some labels are shared by many entities, and bounded search will still surface multiple plausible candidates. The second is sparse local metadata. If a record gives only a short name with little context, there may not be enough evidence to cross the threshold. The third is drifting identity, where organizations rebrand or places change administrative status over time. Selected facts with qualifiers and ranks can help, but they do not erase ambiguity. The fourth is asymmetry between sources, especially when optional Google concordance is available for one candidate path but not another. That can support a decision, but it should not bully the resolver into a false certainty.

These are not weaknesses. They are exactly the situations where explicit uncertainty is the responsible outcome. In my experience, systems become more trusted, not less, when they visibly refuse to overclaim.

Why the CLI matters alongside MCP

The MCP server side will attract most of the attention, especially from users in Claude Code, Cursor, or Codex. But the CLI matters just as much for serious data work. The documented batch and evidence-export commands turn the project from an interactive helper into something that can support operational reconciliation.

That matters for two reasons. First, many linking jobs begin as backlogs, not live requests. You may need to process hundreds or thousands of local records before any agent-driven workflow enters the picture. Second, exported evidence supports review, sampling, and quality control. Even when the resolver performs well, teams usually want periodic checks. A batchable CLI makes that routine rather than exceptional.

This is one place where the project feels grounded in practical use rather than demo design. Search alone demos well. Evidence export and batch processing are what people ask for after the pilot, when they realize someone has to own the links.

The project’s scope is narrow in a good way

A lot of tools in the knowledge graph space suffer from ambition creep. They try to be a graph browser, an ETL layer, a validator, a writer, a search engine, and a universal entity matcher all at once. This project’s documented scope is tighter. It helps agents search Wikidata, read selected facts, and link local records to Wikidata QIDs, with optional Google cross-checking and explicit uncertainty.

That restraint is part of why `kg_resolve` is credible. It is not trying to settle every knowledge problem. It is trying to make one hard operational task safer and more inspectable.

The distinction also matters when people search for MCP for google knowledge graph and wikidata. It is easy to assume a combined label means a merged authority or a comprehensive broker across both ecosystems. That is not what is documented. What exists is a read-only MCP server and CLI that works primarily with Wikidata, optionally uses the Google Knowledge Graph Search API, and treats cross-provider agreement carefully.

What makes kg_resolve worth using

At the core, kg_resolve supports record linking by combining a few choices that are easy to appreciate only after you have cleaned up bad links before: bounded candidates, deterministic outcomes, selected facts on demand, inspectable evidence, and explicit uncertainty.

None of those choices are flashy. Together, they are exactly what mature linking workflows need.

A resolver that caps candidates at three by default is telling you it values quality over noise. A resolver that returns AMBIGUOUS instead of bluffing is telling you it respects the difference between search and identity. A resolver that can expose facts with ranks, qualifiers, and references is telling you the evidence should survive human review. And a resolver that can optionally cross-check exact Google IDs while still refusing to call concordance proof is showing unusual discipline.

For teams working with MCP for Wikidata, that is the right kind of sophistication. Not more complexity, just better boundaries. The result is a tool that helps local records meet Wikidata on terms that are practical, inspectable, and far less likely to create expensive mistakes later.