

Anyone who has spent time connecting messy real-world records to knowledge graphs learns the same lesson quickly: the hard part is rarely getting data, it is deciding what that data actually refers to. Names collide. Dates are partial. Organizations rebrand. Places change jurisdiction. A search box alone does not solve that problem, and a raw entity dump usually makes it worse.

That is why the pairing of `kg_search` and `kg_entity` is more useful than it looks at first glance in the Wikidata + Google Knowledge Graph MCP server. Used together, they support a practical workflow: search narrowly, inspect carefully, and only then attach a QID or read facts with confidence. In MCP for Wikidata, that pattern matters because the goal is not just retrieval. It is retrieval with enough structure and restraint that an agent, a developer, or an analyst can defend the result.

The project itself is fairly clear about what it is and what it is not. It is an open-source MCP server and CLI called Wikidata + Google Knowledge Graph MCP. It lets agents search Wikidata, read selected facts, and link local records to Wikidata identifiers with inspectable evidence and explicit uncertainty when the evidence is not good enough. It is read-only. It does not edit Wikidata, Google, or user data. It can be used in MCP clients such as Claude Code, Cursor, and Codex. Wikidata access does not require an account or API key, while the Google Knowledge Graph Search API is optional.

Those constraints are worth emphasizing because they shape the practical uses of `kg_search` and `kg_entity`. These tools are not meant to replace a full graph database or a custom reconciliation stack. They are useful when you need a disciplined front end to entity lookup and fact inspection, especially inside an MCP workflow where an agent needs bounded, explainable outputs rather than huge payloads.

Why these two tools matter together

On paper, `kg_search` sounds simple. It searches for candidate entities. `kg_entity` sounds equally simple. It reads facts about a chosen entity. In practice, that pairing solves a common operational problem: moving from uncertain text to a defensible graph identity.

The design choice that stands out is bounded search. By default, the server returns three candidates, and it caps the result set at five rather than dumping a large raw list. That may look restrictive if you are used to broad search interfaces, but in agent workflows it is a strength. A language model or a human reviewer does better with three plausible candidates and enough context to compare them than with fifty weak hits that invite hallucination or rushed selection.

Then `kg_entity` becomes the second half of the judgment process. Once a candidate looks promising, you can inspect selected facts, and on request include ranks, qualifiers, and references. That is a practical distinction. Many integrations stop at a label and a short description, which is often not enough when you are deciding whether two records truly refer to the same thing. A date without a qualifier can mislead. A claim without rank can look stronger than it is. A statement without references may still be useful, but it deserves a different level of trust.

That combination, search first and inspect second, is the backbone of responsible entity resolution.

A grounded use case: linking internal records to QIDs

The cleanest use of `kg_search` is record linking. Say you maintain a local catalog of authors, cultural institutions, companies, or places. Your local records probably contain a name, maybe a date or location,

and often a few notes. What they often do not contain is a stable global identifier.

This is where MCP for wikidata becomes practical rather than abstract. An agent can take the local name, run `kg_search`, and get a bounded set of likely candidates. If one candidate appears promising, the agent can call `kg_entity` and inspect the facts most relevant to that record. If the dates line up, the occupation or type matches, and the place context is consistent, you have a reasonable basis for linking the record to the Wikidata QID.

The important part is not speed. It is restraint. The project documentation emphasizes explicit outcomes such as `AUTO_MATCH`, `HOLD`, `AMBIGUOUS`, and `NO_CANDIDATE` in its resolution logic. Even if you are not calling `kg_resolve` directly, that philosophy should inform how you use `kg_search` and `kg_entity`. Some records deserve an automatic match. Others should clearly be held for review. In production data work, knowing when not to link is one of the highest-value behaviors you can build.

I have seen teams burn weeks cleaning up overconfident matches that came from broad search plus optimistic assumptions. A bounded candidate list followed by fact inspection is slower by a few seconds per record and faster by weeks over the life of a project.

Disambiguating people with common names

People are where naive search usually breaks first. One common name can produce multiple politicians, athletes, academics, and entertainers across different countries and decades. A label match alone is not evidence of identity.

`kg_search` helps by limiting the field to a small candidate set. That makes it feasible to compare candidates rather than just picking the first one. Then `kg_entity` gives you what actually matters: the selected facts that can distinguish one person from another. If you request ranks, qualifiers, and references when needed, you can inspect the strength and context of those claims rather than reading them as flat facts.

For example, if your local record has a birth year, nationality, and profession, those become your filters in practice. You search the name, inspect the top few candidates, and compare the birth date, occupation, and related context. If one candidate lines up cleanly and the others do not, you can proceed. If two candidates both fit partially, the right answer may be to stop and mark the case as ambiguous.

That sounds mundane, but this is exactly where MCP for google knowledge graph and wikidata earns its keep. The value is not in pretending ambiguity does not exist. The value is in making ambiguity inspectable and operationally manageable.

Verifying organizations and institutions

Organizations introduce a different kind of mess. They merge, split, rename, or operate through subsidiaries. A local record might use an old name while Wikidata reflects the current one. Sometimes the local record refers to a department or campus while the graph entity refers to the larger institution.

In this situation, `kg_search` is useful as a first pass because it narrows likely candidates without flooding the reviewer. Then `kg_entity` helps you inspect the kind of facts that expose structure: what sort of entity it is, what names or aliases seem relevant, and whether the surrounding facts fit the record you are trying to identify.

The practical trick here is to resist the urge to overinterpret a near match. A name similarity plus a shared city is not always enough. If the local record points to a research center and the candidate entity is the

parent university, you may have found related context rather than the right identity. A careful read of selected facts often reveals that distinction early.

This is also where the project's read-only design is helpful. Because the server does not edit anything, you can safely use it as an inspection layer inside a larger curation process. That keeps the workflow honest. Search, inspect, decide. No accidental writes. No hidden side effects.

Building evidence-driven enrichment workflows

One of the most practical uses of `kg_entity` is not entity resolution at all. It is targeted enrichment. Once you already know the QID, you can pull selected facts for a downstream task. The phrase "selected facts" matters because it implies discipline. You are not trying to ingest all available claims. You are asking for the facts that support a specific business or research need.

Suppose you have already linked a set of local records to QIDs. Now you want to enrich them with a small set of verifiable attributes for display, matching, or routing. Calling `kg_entity` for those QIDs gives you a controlled way to inspect what is there and, when needed, include ranks, qualifiers, and references. That can help you avoid common mistakes such as treating deprecated or weakly supported claims as if they were canonical.

In practical data operations, this matters a lot. The difference between "has a date" and "has a preferred-ranked date with context" is not academic. It affects user trust, conflict handling, and whether your support team gets pulled into edge cases.

The same principle applies if an MCP client is helping a researcher or analyst. Rather than dumping broad graph content into a conversation, it is often better to retrieve a small, relevant set of facts tied to a known entity and inspect them with proper context. That tends to produce better reasoning and fewer leaps.

Where Google Knowledge Graph fits, and where it does not

The project includes optional Google cross-checking. That is useful, but it needs to be understood correctly. The documented join logic uses exact IDs, specifically `/m/` for Wikidata property P646 and `/g/` for P2671. The project also states clearly that agreement between Google and Wikidata is provider concordance, not proof of identity.

That is exactly the right posture. In real reconciliation work, cross-provider agreement can raise confidence, especially when identifiers line up exactly. But it is not independent proof that two entities are the same in every practical sense. Providers can inherit each other's errors, model things differently, or reflect different levels of granularity.

So if you are working with MCP for google knowledge graph, the best use of Google here is as a check, not a shortcut. `kg_search` and `kg_entity` still do the core work. The optional Google layer can support confidence when exact ID joins exist, but it should not replace factual inspection.

This distinction is especially important for teams tempted to market "knowledge graph matching" as if it were infallible. It is not. The good news is that this project does not make that mistake. It leaves room for uncertainty and pushes users toward inspectable evidence.

A sensible workflow inside an MCP client

In day-to-day use, these tools make the most sense when they are treated as part of a short decision loop inside an MCP client such as Claude Code, Cursor, or Codex. The loop is simple enough to become habit.

1. Use `kg_search` with the local name or phrase to get a bounded candidate set.
2. Select the most plausible candidate and inspect it with `kg_entity`.
3. Compare the returned facts with the local evidence you actually have.
4. Accept the link, hold it for review, or mark it ambiguous.
5. Only after identity is stable, use facts for enrichment or downstream reasoning.

That sequence looks almost too obvious to mention, yet many failed graph integrations skip step three. They jump from search to acceptance. The problem is not usually bad tooling. It is bad discipline. The strength of this MCP server is that its design encourages discipline through bounded results and explicit uncertainty.

Why bounded search is better than “more results”

A lot of developers assume more search results are inherently safer because they reduce the risk of missing the right candidate. In practice, broad result sets often create a different risk: they encourage superficial review. If a human or an agent sees too many options, it may latch onto the first plausible one or manufacture a justification.

Returning three candidates by default, with a maximum of five, creates a healthier review environment. It forces the search layer to stay selective and the review layer to stay focused. That is especially helpful in MCP settings, where responses need to remain compact enough for an agent to reason over them without losing track of the decision criteria.

There are trade-offs, of course. A bounded list may omit a low-ranked but correct candidate in difficult cases. That means some records will land in `NO_CANDIDATE` or remain unresolved until the query is improved. But that is usually a better failure mode than false certainty. In curation and linking projects, a missed match can be revisited. A wrong match can contaminate analytics, user interfaces, and downstream exports for a long time.

Selected facts are better than indiscriminate facts

Another practical strength of `kg_entity` is the idea of retrieving selected facts rather than treating the entity page as a data dump. This matters for both performance and judgment.

When people first work with graph data, they often assume more claims mean more truth. Experienced teams know that more claims usually mean more interpretation work. Claims can differ by rank. They can rely on qualifiers for meaning. They can carry references of varying quality or none at all. Reading them without structure invites mistakes.

By supporting ranks, qualifiers, and references on request, `kg_entity` gives you a way to ask for more context when the decision requires it. That is a useful middle ground. You can stay light for straightforward cases and go deeper when a record is contested, incomplete, or high value.

There is also a subtle operational benefit. Targeted retrieval reduces noise in the MCP exchange itself. If an agent only needs enough context to confirm or reject a candidate, selected facts make that possible without swamping the conversation with irrelevant properties.

Good fit, poor fit, and realistic expectations

These tools are strongest in workflows where identity needs to be inspected rather than assumed. They are a good fit for local record linking, research support, entity verification, and careful enrichment. They are less suited to use cases that expect unrestricted exploration, bulk graph export, or automated certainty in highly ambiguous domains.

That is not a criticism. It is a sign of a well-scoped tool. The project openly states that it is not official Wikimedia or Google software, and it is not an export of the Google Knowledge Graph. Those limits help set expectations. The server provides a practical interface for search, [Google Knowledge Graph MCP schema](#) fact retrieval, and cautious linking. It does not claim to settle every identity question on its own.

For teams already working with the broader Wikidata MCP ecosystem, this narrower, evidence-oriented approach can be valuable. Wikidata's own MCP documentation frames standardized tools for exploring and querying Wikidata programmatically via the Wikidata API and Query Service. This project sits comfortably alongside that broader context by focusing on bounded search, selected facts, and explicit uncertainty.

The real payoff: fewer bad links, better review

If I had to reduce the practical value of `kg_search` and `kg_entity` to one sentence, it would be this: they make it easier to say "probably yes," "probably no," and "not enough evidence yet" with a straight face.

That may not sound glamorous, but it is exactly what good data systems need. Entity work lives or dies on judgment. Tools that encourage overreach create cleanup projects. Tools that structure uncertainty create durable workflows.

For anyone exploring MCP for wikidata, or comparing options for MCP for google knowledge graph and wikidata, that is the useful lens. Do not ask whether search is available. Search is easy. Ask whether the search is bounded, whether the facts are inspectable, whether qualifiers and references are available when needed, and whether the workflow leaves room for ambiguity without collapsing into guesswork.

`kg_search` and `kg_entity` pass that test because they are modest in the right ways. They do not promise omniscience. They support careful linking, selective enrichment, and evidence-driven review. In real-world knowledge graph work, those are not secondary features. They are the features that keep the whole system trustworthy.