

Teams that work with entity data usually hit the same wall sooner or later. Search is easy enough. Reliable retrieval is not. You can find ten plausible entities for a name like "Mercury" in seconds, but narrowing that result to the right record, extracting only the facts you need, and preserving enough evidence for later review is where systems often become messy.

That is the problem space where the open source project commonly described as *Wikidata + Google Knowledge Graph MCP* sits. Published as an MCP server and CLI under [revanalex/wikidata-google-knowledge-mcp](https://github.com/revanalex/wikidata-google-knowledge-mcp), it is designed to let agents search Wikidata, read selected facts, and link local records to Wikidata QIDs with inspectable evidence. It also makes room for uncertainty, which is one of its most useful qualities. When evidence is insufficient, it does not pretend otherwise.

The phrase *selected fact retrieval* matters here. Many data integrations fail because they pull far too much material, too early, and with too little structure. A good retrieval layer should do the opposite. It should keep search bounded, fetch only what is needed, expose the shape of the evidence, and make resolution outcomes explicit.

Why selected fact retrieval is more valuable than broad entity dumps

In practical data work, broad entity exports are seductive. A developer sees a rich source like Wikidata and wants the full record. That works for exploration. It tends to work badly in production workflows.

When a system retrieves everything, three problems appear at once. First, downstream agents or applications have to decide which claims matter, often without enough context. Second, the volume of material creates noise, especially when an entity has many statements, aliases, and historical variations. Third, the audit trail becomes fuzzy. You know the system saw "something" on the entity, but not exactly which claims drove the match or decision.

Selected fact retrieval solves that by narrowing the scope. Instead of starting with a complete entity dump, the workflow centers on specific facts. The server documented for MCP for google knowledge graph and wikidata is built around that discipline. It supports retrieval of selected facts and can include ranks, qualifiers, and references on request. That last detail is not cosmetic. In data reconciliation work, qualifiers and references often determine whether a fact is merely plausible or operationally trustworthy.

A simple example makes the distinction clearer. Suppose a local record says "Springfield", category unknown, with one field showing a population estimate and another field hinting at a state or province. Pulling every available fact from multiple Springfield entities adds confusion. Pulling a small set of candidate entities first, then requesting only the properties needed to distinguish them, is a much saner route. It is faster to review, easier to test, and more defensible when someone asks why a match was made.

What this MCP server actually does

The project is not a general export of the Google Knowledge Graph, and it is not official software from Wikimedia or Google. It is read only. It does not edit Wikidata, Google, or user data. Those boundaries are worth stating plainly because they shape expectations. This is a retrieval and resolution layer, not a synchronization engine and not a write back system.

Its stated purpose is direct and useful. It allows AI agents to search Wikidata, read selected facts, and link local records to Wikidata QIDs with inspectable evidence. It also supports an optional Google cross check.

The wording matters because the cross check is framed as provider concordance, not proof of identity. That is mature design. Agreement between providers can strengthen confidence, but it is not the same thing as identity verification.

The project can be used through MCP clients such as Claude Code, Cursor, and Codex. Wikidata access does not require an account or API key. The Google Knowledge Graph Search API is optional. From an operational standpoint, that lowers friction for teams that want to start with Wikidata only and add Google checks later if needed.

For broader context, Wikidata itself documents an MCP offering that gives standardized tools for language models to explore and query Wikidata programmatically via the Wikidata API and Wikidata Query Service. The specialized server discussed here sits in a narrower lane. It is not trying to expose all of Wikidata. It is shaping a focused workflow around search, evidence, fact selection, and entity resolution.

Bounded search is a design choice, not a limitation

One of the smartest parts of the project is its bounded search behavior. By default it returns three candidates, with up to five, rather than large raw result sets.

That sounds modest until you have spent time reviewing entity matches in real systems. Large candidate pools create a false sense of completeness while making review harder. A bounded result set forces the system to present its best candidates rather than spraying possibilities and leaving the hard work to a human or another model.

There is also a quiet but important discipline in that design. Search is treated as a step toward resolution, not an invitation to roam endlessly through neighboring entities. In production pipelines, especially when records are processed in batches, bounded candidate lists are easier to reason about. They reduce token waste in agent contexts. They shorten inspection time for humans. They also make failure modes clearer. If the right entity is not among the top few candidates, the system can escalate to a hold or ambiguity state rather than pretending that candidate number seventeen is a serious contender.

I have seen reconciliation efforts collapse under their own generosity. A search layer returns twenty results, then another process tries to infer confidence from thin differences in labels or snippets. The outputs look sophisticated until someone examines ten borderline cases and discovers that the ranking logic is not stable enough to justify automatic acceptance. Limiting the candidate set is a small choice with outsized consequences.

The heart of the matter: facts, ranks, qualifiers, references

Wikidata is powerful partly because many statements are not flat assertions. Claims can have rank, qualifiers, and references. Any system that treats all facts as equal flattens the part of Wikidata that makes it useful.

This MCP for wikidata supports selected fact retrieval with those details available on request. That means an agent or application can ask not only for a fact but for the surrounding context that governs how the fact should be interpreted.

Rank matters when multiple statements exist for the same property. In the abstract, that sounds like an implementation detail. In practice, it can prevent simple mistakes. If an entity has several statements for a value over time, rank may signal which one is preferred. Without rank, an automated process may treat all

statements as equivalent and either choose arbitrarily or concatenate conflicting values into unusable output.

Qualifiers matter because they sharpen meaning. A bare property value often leaves essential context unstated. Was that office held during a particular period? Does that population figure apply to a specific date? Is a name an official name, a former name, or a native label in a given language? The difference between a correct resolution and a misleading one often lives in the qualifier, not in the main statement.

References matter for obvious reasons, but their practical value is sometimes underappreciated. In a review workflow, references turn a claim from "the graph says so" into "the graph says so, and here is the evidentiary trail you can inspect." That does not magically guarantee truth, but it changes the quality of review. Analysts can decide whether a linked source is current enough, specific enough, or trustworthy enough for the use case at hand.

This is where selected fact retrieval earns its keep. You do not need all references for all claims. You need the references for the claims that drive your decision.

Resolution with named outcomes is a major operational advantage

Another strong part of the design is deterministic resolution with explicit outcomes such as AUTO_MATCH, HOLD, AMBIGUOUS, and NO_CANDIDATE.

Those labels might look straightforward, yet they address one of the chronic weaknesses in entity linking systems: hidden uncertainty. A lot of tooling behaves as if every record must produce a winner. That pressure leads to overmatching. Overmatching is expensive. It pollutes local records, undermines trust, and creates cleanup work that is far harder than the original review would have been.

The explicit outcomes in this project create room for judgment:

- AUTO_MATCH when the evidence is sufficient for deterministic acceptance
- HOLD when a result is promising but should not pass without review
- AMBIGUOUS when multiple candidates remain plausible
- NO_CANDIDATE when the search did not produce a defensible option

This is one of only a few places where a short list clarifies more than paragraphs alone. The operational value is that each outcome suggests a different next step. An AUTO_MATCH can feed a downstream enrichment process. A HOLD can enter a review queue. An AMBIGUOUS case may prompt the system to request another identifying attribute. A NO_CANDIDATE outcome can be logged without poisoning the graph with a bad link.

Determinism also matters. If the same input produces different outcomes on different runs without any underlying data change, your QA process becomes unpleasant fast. Stable resolution logic is not glamorous, but it is what makes batch linking trustworthy.

The optional Google cross check, handled with restraint

The project documents an optional Google cross check using exact ID joins, specifically /m/ for Wikidata property P646 and /g/ for P2671. This is one of the most carefully framed parts of the design.

A lot of systems treat provider agreement as a shortcut to certainty. This one does not. It explicitly treats Google and Wikidata agreement as provider concordance rather than proof of identity. That is a distinction

experienced practitioners tend to appreciate because multi source agreement can still encode the same upstream error, the same stale mapping, or the same category confusion.

Still, concordance is valuable. If a local record is linked to a Wikidata entity and that entity has an exact join aligning with a Google identifier, that is a meaningful signal. It can support confidence, improve explainability, and sometimes help catch edge cases where labels are similar but identities differ.

The key is the use of exact joins. Exact joins are much cleaner than loose text similarity across providers. Once you start cross checking by names alone, you open the door to collisions, transliteration quirks, organization renamings, and all the old problems you were trying to escape. Joining on established identifiers is a more disciplined approach.

For teams exploring MCP for google knowledge graph, that optionality is useful. You can begin with Wikidata only, which requires no account or API key, then layer in the Google Knowledge Graph Search API when concordance adds enough value to justify the extra dependency.

The available tools point to a practical workflow

The documented MCP tools are `kg_search`, `kg_entity`, `kg_related`, `kg_resolve`, and `kg_status`. The CLI also includes batch and evidence export commands.

Even without making assumptions beyond those names, the workflow pattern is fairly clear. Search identifies candidates. Entity retrieval fetches focused details. Related entity exploration provides context when labels alone are not enough. Resolve applies the deterministic outcome logic. Status helps operational visibility. Batch support matters for scale. Evidence export matters for review, debugging, and compliance style record keeping.

A common pattern in knowledge work is to start with a small, explainable interactive loop before moving to batch operation. That is where a CLI often shines. Analysts or developers can try a few problematic records, inspect outcomes, and export evidence before trusting the system with thousands of rows. Once the confidence model is understood, batch mode becomes much safer.

I have found that evidence export is often the difference between a tool that engineers like and a tool that organizations actually adopt. If reviewers cannot see what happened, they assume the system is making guesses in a black box. If they can inspect the candidate set, selected facts, and any supporting references, skepticism drops and feedback becomes more useful.

Where selected fact retrieval helps most

Selected fact retrieval is not equally valuable in every scenario. It is especially strong in workflows where the cost of a wrong link is high and the cost of carrying uncertainty is acceptable.

Think of catalog enrichment, content tagging, internal knowledge systems, editorial review, or data normalization before analytics. In these settings, a wrong entity match can cascade into bad recommendations, misleading labels, or corrupted joins. A system that can say "hold this record" is better than one that makes an elegant mistake.

It is also helpful when local records are sparse. Sparse records often have just enough information to confuse a naive search process. A title, one date, maybe a location, and little else. In that setting, selected fact retrieval allows a system to ask for exactly the few discriminating facts that matter rather than flooding the context window with everything known about the candidate entity.

By contrast, if the use case only needs broad exploratory search and no durable linking, full entity browsing may be perfectly fine. The discipline of selected fact retrieval pays off most when records need to be linked, reviewed, and revisited later.

Trade offs and edge cases worth respecting

No retrieval strategy removes ambiguity from the world. Names collide. Entities change. Sources disagree. A read only system can surface evidence, but it cannot fix problems in the underlying graph.

One trade off is obvious. Bounded search improves focus but may miss edge cases that only appear deeper in a ranking. That is acceptable if the resolution logic can gracefully return NO_CANDIDATE or HOLD and let a human intervene. It is less acceptable if [Take a look at the site here](#) users expect every obscure entity to appear in the first few hits automatically. The design suggests a preference for precision and inspectability over exhaustive recall in a single step.

Another trade off sits in qualifiers and references. They are valuable, but they add complexity. If every retrieval request asks for full contextual detail, the workflow can become heavier than necessary. Good practice is to request richer evidence when the decision demands it, not by default for every trivial match.

The Google cross check brings its own judgment call. Concordance is useful, but only when teams remember what it is and what it is not. If users treat cross provider alignment as proof, they will overestimate certainty. The project explicitly warns against that interpretation, and that restraint is one of its strengths.

Finally, the system is read only. That is a feature for some organizations and a limitation for others. It keeps the scope clean and the risk lower, but it also means remediation happens elsewhere. If a local workflow discovers an issue, someone still needs a separate path for fixing internal data or contributing corrections upstream.

How to think about this in agent workflows

The current wave of MCP adoption has made one old problem visible again: giving agents too much unfiltered data leads to worse outcomes, not better ones. Agents tend to perform best when tools return structured, bounded, high signal information.

That is why this project's approach fits MCP so well. The combination of bounded search, selected fact retrieval, explicit resolution states, and optional evidence export creates a narrower and more reliable tool contract. Instead of asking an agent to infer identity from a large, messy payload, the server helps shape the decision space.

A disciplined agent workflow using MCP for google knowledge graph and wikidata might look something like this:

- search for the top few candidates rather than a long result list
- retrieve only the facts needed to discriminate among those candidates
- inspect ranks, qualifiers, or references when the match is close or contested
- resolve to a named outcome and preserve the evidence for review
- use Google concordance only as a supporting signal when exact joins exist

That sequence is not complicated, which is part of the appeal. It mirrors how careful humans tend to work when they are not rushed: narrow the field, inspect the decisive evidence, and stop short of certainty when

certainty is not justified.

A better standard for entity retrieval

The most encouraging thing about this project is not any single tool name or integration point. It is the standard of behavior it implies. Search should be bounded. Facts should be selected, not dumped. Evidence should be inspectable. Uncertainty should be explicit. Cross provider agreement should support judgment, not replace it.

For anyone evaluating MCP for wikidata or looking at MCP for google knowledge graph as part of an entity resolution stack, those are healthy principles to build around. They align with the realities of knowledge graphs rather than pretending those graphs are clean, final, or self interpreting.

In daily practice, selected fact retrieval is less flashy than broad search and more useful than it first appears. It shortens review cycles, improves auditability, reduces overmatching, and gives both humans and agents a better chance of staying honest about what they know. That is usually what separates a promising demo from a system that survives contact with real records.