

People often treat search and resolution as if they were the same operation. In practice, they solve different problems, and confusing them usually creates messy downstream data. That distinction matters a lot in the Wikidata + Google Knowledge Graph MCP server, where `kg_search` and `kg_resolve` sit close together conceptually but behave very differently.

If you work with MCP for Wikidata, or more specifically with MCP for google knowledge graph and wikidata, the easiest mistake is to assume that anything returning candidate entities is already doing identity resolution. It is not. One tool helps you explore. The other helps you decide, within explicit limits, whether a local record can be linked to a Wikidata QID with enough evidence to trust the result.

That difference sounds subtle until you try to use the output in a pipeline.

Search is for discovery, resolution is for decision

`kg_search` is best understood as a search interface shaped for agents. You provide a query, and the tool returns a bounded set of likely candidates. The project documentation makes that bounded behavior a design feature rather than an implementation detail. By default, it returns 3 candidates, with a maximum of 5, instead of flooding the caller with a long result [Knowledge Graph MCP lookup](#) set.

That tells you a great deal about its intended use. This is not a bulk export tool, and it is not trying to mimic a full search engine interface. It is giving an MCP client just enough context to continue the conversation or the workflow.

`kg_resolve`, by contrast, is not merely asking, “What entities look relevant?” It is asking a harder question: “Can this specific local record be matched to a Wikidata entity, and if so, how confident should we be?” The project describes its resolution logic as deterministic, with explicit outcomes such as `AUTO_MATCH`, `HOLD`, `AMBIGUOUS`, and `NO_CANDIDATE`. That language is the giveaway. Search produces options. Resolution produces an adjudicated state.

In day-to-day data work, that gap is the difference between a researcher scanning possibilities and a system assigning identity.

What `kg_search` is designed to do well

Search is where you start when the human or agent still needs orientation. Maybe the query is a person's name, an organization, a book title, or a place name with limited context. You do not yet know whether the record refers to a unique thing or one of several near-matches.

With `kg_search`, the value is not just that it finds entities. The value is that it keeps the result set intentionally small. In many agent workflows, too many candidates are worse than too few. A giant search response encourages vague comparisons, hallucinated certainty, and brittle follow-up prompts. A bounded search result is easier to inspect, easier to rank in context, and easier to hand off to `kg_entity` for closer reading.

That bounded design also changes how you should think about failure. If `kg_search` does not return the entity you expected, it does not necessarily mean the entity does not exist. It means the top bounded candidates for that query did not produce the expected result. That is a search interpretation problem, not yet a resolution verdict.

I have seen this pattern cause confusion in entity-linking projects. Someone enters a sparse organization name, gets three plausible candidates, and assumes that if the exact legal entity is missing, there is no match available. That is too strong a conclusion. Search results are query-shaped. Resolution decisions are evidence-shaped.

What kg_resolve is trying to protect you from

Identity resolution has a different failure mode from search. With search, the risk is inconvenience or extra manual review. With resolution, the risk is a false link that looks authoritative and then spreads across systems.

The project's explicit outcomes are useful because they encode restraint. AUTO_MATCH signals that the system can make a deterministic match. HOLD implies more review is needed. AMBIGUOUS tells you the evidence points to multiple viable candidates. NO_CANDIDATE tells you that nothing suitable was found under the method's rules.

That is an operationally mature model. In many pipelines, the worst behavior is not missing a match. The worst behavior is silently choosing one when the evidence is weak. A held record can be reviewed. A bad auto-link can contaminate analytics, customer records, content graphs, or reconciliation jobs.

This is where kg_resolve differs most sharply from kg_search. Search can be useful even when it is broad or uncertain. Resolution is only useful when uncertainty is surfaced clearly enough that a caller can act on it responsibly.

Why the two tools belong together

Although the tools have different purposes, they complement each other closely. In a practical workflow, you often search first to understand the candidate space, then inspect one or more candidates, and only then attempt or accept resolution. The server's other tools make that pattern clearer: kg_entity lets you read selected facts, and kg_related helps explore nearby entities. kg_status gives operational visibility. The CLI extends this into batch and evidence-export commands.

The architecture suggests a deliberate separation of concerns. Search retrieves possibilities. Entity retrieval exposes facts, including ranks, qualifiers, and references on request. Resolution applies deterministic logic and returns an explicit outcome. Those are not interchangeable phases, and blending them too early usually leads to overconfident matching.

In other words, if you use kg_search where you needed kg_resolve, you are likely to overread suggestion as certainty. If you use kg_resolve where you really needed kg_search, you may force a decision before you have enough context.

A concrete way to think about the difference

A simple comparison helps:

Tool	Primary job	Typical output shape	Best used when
kg_search	Find likely candidate entities	Small bounded candidate set	You are exploring possibilities
kg_resolve	Decide whether a local record can be linked	Deterministic outcome with match state	You need an actionable identity decision

That may look obvious on paper, but it becomes much more important once you build a repeatable process. Search answers, "What might this be?" Resolution answers, "Can I safely link this now?"

The role of evidence in each tool

The project's emphasis on inspectable evidence is one of its strongest design choices. That matters for both tools, but in different ways.

With `kg_search`, evidence is often indirect. A candidate appears because the query resembles labels or known forms strongly enough to place it near the top. The user still has work to do. They may need to open the entity, compare facts, or inspect related entities before deciding whether the search result is actually the target.

With `kg_resolve`, evidence has to support a decision path. Even when the system reaches `AUTO_MATCH`, the point is not blind automation. The point is a match with a rationale sturdy enough to survive review. The project also states that when evidence is insufficient, uncertainty should be explicit. That is a critical promise for data governance.

This becomes especially relevant when names are common, transliterated, abbreviated, or reused across jurisdictions. A plain search may surface the right answer among several siblings. A resolution tool must know when those siblings are too close to call.

Where Google Knowledge Graph fits, and where it does not

The server includes optional Google cross-checking, which is easy to misunderstand if you are not careful. The documented behavior is specific: it can do exact ID joins using `/m/` for Wikidata property `P646` and `/g/` for `P2671`. The project is also careful to frame Google and Wikidata agreement as provider concordance, not proof of identity.

That distinction matters enormously.

For `kg_search`, optional Google context may enrich your understanding of a candidate landscape, especially in workflows that already touch MCP for google knowledge graph. But search remains search. A Google-facing hint does not turn a search result into a verified entity link.

For `kg_resolve`, the cross-check can strengthen confidence in some cases because the same identifiers align across providers. Yet the project explicitly avoids overselling that concordance. Agreement between providers is useful evidence, but it is not metaphysical truth. Providers can share stale mappings, inherit the same public assumptions, or simply agree on a wrong linkage. Treating concordance as supporting evidence rather than final proof is the right call.

That restraint is one reason the server feels more operational than promotional. It acknowledges the appeal of multi-provider agreement without pretending that agreement erases ambiguity.

Why bounded search changes agent behavior

One detail that deserves more attention is the bounded search result. Returning 3 candidates by default, up to 5, sounds modest, but it has strong implications for MCP clients such as Claude Code, Cursor, and Codex.

When an agent receives a huge candidate set, it tends to narrate patterns too quickly. It may anchor on superficial signals and construct a story around the first plausible entity. A bounded result narrows that

temptation. It forces the next action to be more deliberate, often by calling `kg_entity` to inspect selected facts or by deferring to `kg_resolve` if the task is really reconciliation rather than browsing.

In practical terms, `kg_search` in this server is not trying to be exhaustive. It is trying to be useful inside a decision flow.

I think that is one of the more mature design choices in modern MCP tooling. Exhaustiveness sounds attractive until you watch an agent drown in options. Bounded outputs, especially around identity tasks, are often the difference between a tractable workflow and a noisy one.

The hidden danger of using search results as if they were resolved identities

This is where many implementations go wrong. Someone builds a script, queries by name, takes the first hit, and writes the QID back into a record. It works beautifully on easy cases and fails quietly on the ones that matter.

The problem is not that search is bad. The problem is that the script has skipped the contract that `kg_resolve` is trying to provide. Search can return highly plausible candidates even when the evidence needed for exact identity is missing. For a well-known entity with a distinctive name, the top result may be enough in practice. But systems fail at scale on the long tail, where names collide, metadata is sparse, and local records contain errors.

That is why deterministic statuses matter. A record marked `AMBIGUOUS` is not a nuisance. It is a healthy refusal to guess. A record marked `HOLD` prevents accidental certainty. A record marked `NO_CANDIDATE` lets you branch into remediation instead of fabricating confidence.

This is especially important if you plan to batch-process records. The CLI's support for batch and evidence export suggests the project expects industrial use cases, not just interactive lookups. In batch mode, every silent false positive becomes expensive.

When to reach for each tool

A practical rule of thumb works well here:

- Use `kg_search` when the task is exploratory and you still need to understand the candidate space.
- Use `kg_resolve` when the task is record linkage and the output must carry an explicit decision state.
- Use `kg_entity` after either one if you need to inspect selected facts, including ranks, qualifiers, and references.
- Treat optional Google cross-checking as corroboration, not identity proof.

That pattern may sound conservative, but conservative is exactly what you want in entity resolution. Discovery should be permissive. Linking should be strict.

The Wikidata angle matters more than it first appears

For broader context, Wikidata's own MCP documentation describes a standardized way for LLMs to explore and query Wikidata programmatically through the Wikidata API and the Wikidata Query Service. That wider ecosystem matters because it frames user expectations. Many people approach MCP for Wikidata expecting

rich exploration and query capabilities. They may not immediately distinguish between querying, searching, and resolving.

This server narrows the focus in a useful direction. It is read-only, it does not edit Wikidata, Google, or user data, and it is explicit that it is neither official Wikimedia software nor official Google software. Those boundaries are healthy. They signal that the server is built for retrieval, inspection, and linking workflows, not for authority over the underlying knowledge bases.

Within that boundary, `kg_search` and `kg_resolve` play complementary but non-overlapping roles. One helps you navigate the graph. The other helps you decide when a local record can be anchored to it.

A realistic example without overclaiming

Imagine you have an internal catalog record with a title or name that could refer to several entities. You start with `kg_search` because you need to see the likely candidates. The server gives you a short, bounded list. One candidate looks promising, but another has a similar label. At that point, a careful workflow would inspect selected facts through `kg_entity`, perhaps paying attention to qualifiers or references if the distinction depends on time, role, or jurisdiction.

Now suppose you need to write a QID back into the catalog. That is where `kg_resolve` becomes the right tool. If the logic supports an `AUTO_MATCH`, great. If it returns `AMBIGUOUS`, you have learned something equally valuable: the record should not be auto-linked. If it returns `HOLD`, you know the system is waiting for stronger evidence. If it returns `NO_CANDIDATE`, you do not have enough to link safely under the documented method.

The important point is that the same underlying entity might appear in both workflows, but the question you ask is different. Search asks for options. Resolution asks for permission to commit.

Why this distinction improves trust

Trust in knowledge workflows rarely comes from impressive retrieval alone. It comes from knowing when the system will stop short of certainty. That is what separates a useful search tool from a dependable resolution tool.

The Wikidata + Google Knowledge Graph MCP project appears to [Wikidata MCP](#) understand that. It combines search, fact retrieval, related-entity exploration, deterministic resolution, and optional provider concordance checks, while still keeping the boundaries clear. Wikidata requires no account or API key in this setup. Google Knowledge Graph Search API support is optional. The stack is practical, inspectable, and intentionally read-only.

For teams using MCP for wikidata, that matters because not every lookup needs to become a match, and not every match should begin as a search result. If you keep `kg_search` and `kg_resolve` mentally separate, your workflows stay cleaner. Your evidence trails get easier to defend. Your bad links drop.

That is the real difference between the two tools. `kg_search` helps you find candidates worth considering. `kg_resolve` helps you decide, with explicit uncertainty when needed, whether consideration can become commitment.