

Entity resolution looks simple until you have to trust it.

Anyone who has tried to map a local catalog, CRM, newsroom archive, research database, or product knowledge base to Wikidata knows where the pain starts. Names collide. Labels are incomplete. Dates are missing. Places change names. Organizations merge, split, or rebrand. A human can often sense the difference between a plausible match and a dangerous one. Software needs firmer rules.

That is what makes the resolution outcomes in the Wikidata + Google Knowledge Graph MCP interesting. Instead of pretending every query can be flattened into a neat yes or no, it returns explicit states: `AUTO_MATCH`, `HOLD`, `AMBIGUOUS`, and `NO_CANDIDATE`. Those labels are not cosmetic. They express a judgment boundary, and that boundary matters when an agent is linking records to Wikidata QIDs.

The project itself is an open-source MCP server and CLI, published as “Wikidata + Google Knowledge Graph MCP.” Its stated purpose is narrow in a good way: let AI agents search Wikidata, read selected facts, and link local records to Wikidata QIDs with inspectable evidence and explicit uncertainty when the evidence is not strong enough. It works with MCP clients such as Claude Code, Cursor, and Codex. Wikidata requires no account or API key in this setup, while the Google Knowledge Graph Search API is optional.

Those design choices tell you what kind of problem the tool is trying to solve. It is not trying to dump huge search result pages into an agent prompt and hope the model picks the right item. It is trying to create a bounded, inspectable resolution path.

Why these four outcomes matter

A lot of linking systems fail for the same reason. They push too hard toward automatic certainty. The cost looks low at first, because automatic linking feels efficient. The cleanup bill arrives later, when a person notices that several dozen records about one “John Williams” were linked to the wrong composer, conductor, or guitarist. By then, downstream exports, analytics, and user interfaces have already absorbed the error.

The four resolution outcomes create a more disciplined contract between the search process and the action that follows. A system that says “I found something” is not very useful if you cannot tell whether “something” means strong identity evidence, a tentative candidate, multiple plausible candidates, or nothing workable at all.

In practical terms, these outcomes let you route the record correctly. An `AUTO_MATCH` can proceed into linking workflows. A `HOLD` can wait for one more discriminating field. An `AMBIGUOUS` case can be sent to review or enrichment. A `NO_CANDIDATE` result can trigger alternate research rather than a forced match.

That sounds obvious, but it is still uncommon. Many pipelines bury these distinctions inside ad hoc scoring rules or opaque confidence percentages. The strength of the MCP approach here is that the uncertainty is first-class and explicit.

The shape of the server behind the labels

Before getting into the four statuses one by one, it helps to look at the server’s operating style.

The project emphasizes bounded search. By default, it returns three candidates, and it goes up to five, rather than flooding the caller with large raw result sets. That is not just a performance detail. It changes the

resolution experience. Bounded candidate sets force the agent and the human reviewer to look at a small number of serious possibilities instead of drowning in loosely related labels.

The documented MCP tools reflect that same focus. There is `kg_search` for candidate discovery, `kg_entity` for reading selected facts from a chosen entity, `kg_related`, `kg_resolve` for deterministic resolution, and `kg_status`. The CLI also provides batch workflows and evidence-export commands. That combination matters because entity resolution is rarely a single-call event in real work. You search, inspect, verify, and often need to export the evidence trail for auditing or review.

The selected-fact retrieval feature is another important piece. The server can return facts with ranks, qualifiers, and references on request. Anyone who has spent time with Wikidata knows why this helps. The bare presence of a property often is not enough. Dates need qualifiers, names may differ by language or period, and references can indicate whether a fact is strong enough to support identity checks. Resolution work gets better when evidence is inspectable at that level rather than flattened into a generic entity summary.

What AUTO_MATCH should mean in practice

AUTO_MATCH is the outcome people usually want, but it is also the one most likely to be abused if the system behind it is too eager.

In this MCP context, the important verified fact is that the resolution logic is deterministic and returns explicit outcomes, including AUTO_MATCH. That tells us the system is not merely producing a vague ranking. It is applying a repeatable decision process. Repeatability is essential when the same record needs to produce the same linking outcome tomorrow, next week, or in a batch rerun after other records have been processed.

A healthy interpretation of AUTO_MATCH is not “this feels right.” It is “the available evidence was sufficient under the system’s rules to link this local record to a specific Wikidata QID without human intervention.” The key phrase there is under the system’s rules. Deterministic does not mean infallible. It means inspectable and reproducible.

That distinction matters in production settings. If a batch of 10,000 local records is being linked and 7,200 come back as AUTO_MATCH, the success of the batch does not depend only on that number. It depends on whether each match can be defended later. When the server supports inspectable evidence and selected-fact retrieval, that defense becomes possible. A reviewer can examine which candidate was chosen and what facts were available at the time.

There is also a subtle workflow benefit. Because bounded search typically returns only a few candidates, an AUTO_MATCH result is easier to trust than one drawn from a huge result set with weak ranking separation. When the candidate pool is constrained and the evidence is visible, auto-linking becomes less mysterious.

Why HOLD is often the smartest answer

In many real projects, HOLD ends up being more valuable than AUTO_MATCH, even if it feels slower.

A hold state acknowledges that the record might still be resolvable, but not safely with the current evidence. That is a very different message from “nothing exists” or “two candidates are equally plausible.” It says the search found enough structure to keep the case alive, but not enough to convert it into a link.

This is where people with cataloging or data operations experience tend to nod. Plenty of records are one field away from certainty. A birth year, headquarters city, official website, former name, or external identifier can break a tie or strengthen a tentative candidate into a reliable one. Sending those records into a hold queue preserves them for targeted enrichment instead of polluting your graph with weak matches.

The existence of HOLD also has cultural value inside teams. It gives analysts and developers permission not to overclaim. I have seen linking efforts go off the rails because every uncertain case was forced into either “matched” or “not found.” That binary framing rewards aggression. A hold state rewards restraint, which is usually the cheaper mistake.

When people talk about MCP for wikidata in operational terms, this is the kind of behavior that actually affects outcomes. The glamorous part is getting an agent to search and retrieve entities. The unglamorous part is deciding when not to link. HOLD is a mechanism for that discipline.

AMBIGUOUS is not failure, it is signal

If HOLD says “not enough evidence yet,” AMBIGUOUS says “more than one candidate remains genuinely plausible.”

That difference is important. Ambiguity is often a property of the source material, not a weakness in the resolver. Two people can share the same name, profession, and rough time period. Two organizations can use nearly identical labels in different jurisdictions. A place name can map to multiple settlements, districts, or historical entities. In those situations, a careful resolver should surface the ambiguity rather than hiding it behind a false winner.

The bounded candidate model helps here too. If the server is returning a small set of serious candidates, an AMBIGUOUS outcome means the conflict is concentrated and reviewable. Instead of asking someone to scroll through dozens of weak search hits, it asks them to decide among a few contenders with inspectable facts.

There is a practical effect on user trust. Reviewers tend to lose confidence **Wikidata MCP profile** in resolution tools when they notice obvious ambiguities being auto-linked. They gain confidence when the tool recognizes those cases and leaves them open. The credibility of AUTO_MATCH rises when AMBIGUOUS is used honestly.

In workflows involving MCP for google knowledge graph and wikidata, ambiguity can also become more visible rather than less. That may sound counterintuitive. People often assume that adding another provider will magically resolve uncertainty. Sometimes it does not. Sometimes it only confirms that multiple identity paths exist or that one provider’s representation does not eliminate the need for judgment. The project’s own documentation is careful on this point, and that caution is one of its strongest design signals.

NO_CANDIDATE should not be treated as dead end data

A NO_CANDIDATE outcome is easy to misread. Teams often hear “no candidate” as “the resolver failed” or “the item is absent from Wikidata.” Neither interpretation is guaranteed by the status alone.

What it safely tells you is narrower: within the resolver’s search and evidence process, no candidate was found that could be returned as a serious possibility. That still leaves several possibilities open. The local record may be too sparse. The label may be noisy. The entity may exist in Wikidata under a less expected name. Or it may truly be missing.

The disciplined response to NO_CANDIDATE is not to force a fallback match. It is to treat the result as a branch in the workflow. In a small editorial context, that may mean manual research. In a larger pipeline, it may mean collecting those records for later enrichment or possible item creation outside this tool's scope. The project is read-only and does not edit Wikidata, Google, or user data, so NO_CANDIDATE does not trigger creation behavior by itself.

This read-only posture is worth emphasizing because it keeps the boundary clean. The server helps agents search, inspect, and resolve. It does not mutate either source system. That is the right separation for cautious identity work.

Google cross-checks help, but they are not proof

The optional Google layer is the most likely part of the project to be oversold by people who only glance at it.

The documented behavior is quite specific. The server supports an optional Google cross-check using exact ID joins, specifically /m/ for Wikidata property P646 and /g/ for P2671. Just as important, the project treats Google and Wikidata agreement as provider concordance, not proof of identity.

That is exactly the right stance.

When MCP for google knowledge graph enters a Wikidata resolution workflow, it can be tempting to assume that agreement between providers closes the case. Sometimes it makes the case much stronger. But the documentation's warning matters: concordance is still concordance. It shows that two systems align on an identifier relationship. It does not magically erase all context or source quality questions.

In my experience, teams get into trouble when they confuse external alignment with final truth. A cross-check can be a high-value signal, especially when it uses exact identifier joins rather than fuzzy label matching. But identity work still benefits from inspectable facts. If a local record is weak, a concordant provider signal may support the match, yet you still want to know which entity facts were available and whether they fit the local record.

This also speaks to what the project is not. It is not an export of the Google Knowledge Graph, and it is not official software from Wikimedia or Google. That may sound like a disclaimer, but it has practical importance. It clarifies expectations. You are using a read-only tool that can consult Wikidata directly and can optionally use the Google Knowledge Graph Search API as an extra check. You are not getting a merged master truth layer.

The statuses in operational terms

The cleanest way to think about the four outcomes is as routing instructions for downstream work.

- AUTO_MATCH means the deterministic resolver found enough evidence to link the record to a specific Wikidata QID.
- HOLD means the case remains promising, but the evidence is not sufficient for a safe automatic link.
- AMBIGUOUS means multiple candidates remain plausible and the resolver should not pick one.
- NO_CANDIDATE means the search and resolution process did not produce a viable candidate to inspect or link.

That framing is more useful than treating the statuses as abstract labels. In a batch run, each status implies a different next action. In an interactive MCP client, each status tells the agent what to do next, whether that

is fetch entity facts, request more local data, pause for review, or stop trying to force a connection.

How the MCP tools fit the decision cycle

The toolset makes more sense when you stop seeing it as a generic search API and start seeing it as a compact resolution workbench.

A typical cycle begins [Wikidata MCP](#) with `kg_search`, where bounded candidate retrieval prevents result sprawl. If a candidate looks promising, `kg_entity` can retrieve selected facts, and when needed those facts can include ranks, qualifiers, and references. `kg_resolve` is where the deterministic status gets assigned. `kg_status` rounds out the workflow by exposing the state clearly, and `kg_related` can help when the resolver needs broader context around an entity.

The CLI matters for different reasons. Batch and evidence-export commands are where this kind of tool becomes operational rather than merely demonstrative. If your team has ever tried to explain to an editor, librarian, or compliance reviewer why a machine linked a record to a given QID, you know that evidence export is not a luxury. It is what turns a black-box guess into a reviewable act.

Here is where the broader Wikidata MCP context is useful. Wikidata's own MCP documentation describes standardized tools for LLMs to explore and query Wikidata programmatically via the Wikidata API and Wikidata Query Service. That larger ecosystem matters because it normalizes the idea that an LLM should use tools to inspect structured data rather than hallucinate its way through entity identity. The Wikidata + Google Knowledge Graph MCP sits inside that same practical movement, but with a narrower resolution-oriented emphasis.

Bounded search is a design choice with real consequences

It is easy to underestimate the significance of returning three candidates by default, with up to five.

Search systems often signal seriousness by returning more. In entity resolution, more can be worse. Long result lists create cognitive drag for human reviewers and noise for agents. They also encourage weak ranking assumptions, where the first result may look authoritative simply because it is first.

A bounded set changes the discipline of the interaction. It says the server will try to surface a small number of meaningful possibilities, not every remotely matching label. That works especially well when combined with deterministic statuses. If the server cannot narrow the field enough, it can say `AMBIGUOUS` or `NO_CANDIDATE` rather than pretending abundance equals quality.

I have seen projects improve simply by reducing candidate volume. Reviewers make better choices when they compare two or three strong options rather than skimming twenty. Agents also reason more reliably over compact evidence than over sprawling search output. For anyone evaluating MCP for google knowledge graph and wikidata, this is one of the details worth paying attention to. It is not flashy, but it reflects real experience with how resolution work succeeds or fails.

Sensible ways to use these statuses in production

If you are integrating this MCP server into a real workflow, the healthiest pattern is to treat the statuses as policy triggers, not just diagnostic output.

- Allow only `AUTO_MATCH` to create automatic links in your main graph.
- Send `HOLD` and `AMBIGUOUS` into a review or enrichment queue with exported evidence attached.

- Preserve NO_CANDIDATE records for later reprocessing rather than discarding them.
- Keep the Google cross-check optional and interpret agreement as support, not proof.
- Audit a sample of AUTO_MATCH outcomes regularly to make sure your trust remains earned.

Those are governance choices as much as technical ones. The tool gives you explicit uncertainty categories. The real test is whether your pipeline respects them.

The quiet value of inspectable uncertainty

There is a broader lesson in these four labels.

For years, a lot of software around knowledge graphs and linked data has suffered from one of two habits. Either it exposed raw complexity and left users to sort it out, or it hid complexity behind confidence theater. This MCP server takes a better middle path. It narrows the candidate space, retrieves selected facts when needed, and exposes a deterministic outcome that still leaves room for uncertainty.

That is why AUTO_MATCH, HOLD, AMBIGUOUS, and NO_CANDIDATE deserve attention. They are not merely return values in an API response. They express a philosophy of identity work: link when the evidence is good enough, pause when it is not, show ambiguity when it exists, and admit when no real candidate has surfaced.

For anyone working with MCP for wikidata, that is a mature stance. For anyone bringing in MCP for google knowledge graph as a supporting signal, it is an even more necessary one. Cross-provider checks can strengthen confidence, but they do not erase the need for judgment. The project's insistence on inspectable evidence, bounded search, deterministic logic, and explicit uncertainty is what makes these statuses meaningful rather than decorative.

When entity resolution goes wrong, it usually goes wrong because a system was too eager to sound certain. These four outcomes push in the opposite direction. They ask the resolver to earn certainty, and when it cannot, to say so plainly. That is not just good engineering. It is good data stewardship.