SPSync

AI on SharePoint data architecture guide (PDF)

Agents and retrieval on SharePoint list data through one SQL replica, when to use embeddings and when to use SQL, and what the replica leaves out.

The first AI project on SharePoint data usually starts with one agent calling Microsoft Graph directly. It works in the demo. Then a second team builds a second agent with its own app registration, both start reading the same lists every few minutes, and the tenant's resource-unit budget goes on repeat reads of data that has barely changed. Each agent also has to learn that a person column holds a lookup id, and that past 5,000 items a query that filters or sorts on a non-indexed column can fail. The alternative we recommend is to read SharePoint once into a database and have every agent query that database.

The four layers, in SharePoint terms

Layer | What it is | Who owns it

System of record | The SharePoint lists and libraries people already edit | The business, unchanged

Reader | SPSync, holding the only Graph connection, the throttling budget and each list's delta token | One app registration, one admin grant

Replica | One database with a normalised schema, refreshed on your sync interval | Your database team

Agents | Copilot Studio agents and assistants with SQL tools | Each team, with its own database login

The point of the split is that agents never touch Graph. SPSync makes every SharePoint call and keeps the throttling and delta state. Adding a fifth agent costs one more database login and no extra SharePoint traffic.

The replica keeps SharePoint's structure in a form an agent can reason about. Each list is a table keyed on sp_item_id. A lookup is {col}_id plus {col}_value, a person is a user id that joins to sp_users, and a multi-select column is a child table. The system columns sp_modified and sp_deleted_at let an agent ask what changed this week, including what was deleted, which a search index alone can't answer.

Retrieval or SQL

Most questions people put to an agent about list data have exact answers, such as how many open requests there are or which suppliers are late. Those belong in SQL. A retrieval system returns the passages that look most similar to the question, and an agent that counts rows from retrieved passages gets the count wrong whenever the answer is spread across more rows than it retrieved.

Embeddings earn their place on free text. A Description or Resolution column written by hundreds

of people is where "find incidents like this one" needs similarity rather than a where clause.

Question | Use | Where it runs
How many, which, what total, what changed | SQL | Any SPSync destination with a SQL engine
Find items that read like this one | Vector search | PostgreSQL with pgvector, or Azure AI Search
Mixed, for example similar incidents at one site this year | Vector search with a SQL filter | PostgreSQL with pgvector

SPSync generates embeddings during the sync for two destinations, pgvector and Azure AI Search. On pgvector each row gets one embedding built from the text columns you choose, with a hash beside it so only rows whose text changed are embedded again. Elasticsearch, OpenSearch, Qdrant, Weaviate and LanceDB receive the replicated columns and no vector, so bring your own embedding step there.

Giving an agent tools over the replica

An agent needs a small set of SQL tools, covering a list of tables, a description of each table and a read-only query. SPSync ships no agent. Point any Postgres or SQL Server MCP server at the replica with a dedicated login, or run spsync mcp serve, SPSync's own read-only MCP server, and the agent gets those tools without an Entra app of its own.

Bound what the agent reads in the database. Give its login select on the tables it needs, make the session read only, set a statement timeout, and use row-level security where different agents should see different rows. The Postgres starter kit has the SQL for each step. Your database's own query log then records what each agent read. If the agent is written in LangChain, LlamaIndex, Semantic Kernel, CrewAI, AutoGen or n8n, the framework guide has the tested code for each one.

Context engineering and agent memory, in replica terms

Context engineering is the current name for deciding what reaches a model's context window and in what shape. For list data the replica supplies three useful pieces: a schema whose table and column names an agent can read before it writes a query, exact rows returned by that query in place of a pasted export, and sp_synced_at, so the agent can say how old its answer is. What it leaves out is listed below, and those gaps belong in the agent's instructions as much as in this guide.

Agent memory means the notes an agent keeps between sessions, such as what a user asked last week or which report it produced. The replica is not that store. It holds shared facts that SPSync updates on every sync, and the agent's login can only read it. Keep memory in a separate schema or database the agent may write to, and join it to the replica tables by sp_item_id when the agent needs to remember something about a particular item.

What the replica doesn't hold today

Plan around these before you promise anything to the business. Each row names the usual workaround.

Not in the replica | What to do instead

File contents from document libraries | SPSync copies library metadata only. Use Azure AI Search's SharePoint indexer or your own extraction for the text

SharePoint item permissions | Rows arrive without sharing settings. See the next section

Managed metadata columns | The term's GUID and label come across; the term hierarchy does not

A write path back to SharePoint | SPSync is read only. If an agent proposes changes, write them to a staging table a person reviews and applies

Library file name, URL and size | Mapped, and still being verified on a live tenant

SharePoint permissions are not carried into the replica

SPSync reads each item's fields and not its sharing settings. A row for an item that only HR can open in SharePoint sits in the same table as every other row, and any login with SELECT on that table can read it. Because the replica has no item permissions, Azure AI Search's security trimming fields stay empty too.

Decide the boundary in the database. Replicate only the lists an agent should see, give each agent its own login, expose views in place of whole tables, and add row-level security where two audiences share a table. Don't tell users the assistant respects SharePoint sharing. Tell them which database role it uses and what that role can read.

Freshness

The replica is as current as the sync interval, which can be as short as one minute with application permissions. Each delta round costs one resource unit per list when nothing has changed, so a short interval is cheap. A reconcile, weekly by default, compares the replica with SharePoint as a quality check and recovers anything missed during an outage on your side.

For a worked agent setup, see connecting an AI assistant to SharePoint data with SQL and the agent data layer use case, which joins three lists in one query. Retrieval projects start from RAG on SharePoint lists.

Full, up to date version: https://spsync.io/resources/ai-on-sharepoint-data-guide/