Memory and context: Helping agents remember

If you ask the agent, 'What did you just ask me?', it won't remember anything. Every message starts from scratch. That's the gap the memory fills – and that's the difference between a disposable tool and a true assistant.

In lesson 4, your research agent might search the web, query Wikipedia, and synthesize findings . But if you ask the agent, "What did I just ask you?", it won't remember anything. Every message starts from scratch. That's the gap that memory fills—and that's the difference between a disposable tool and a true assistant.

Why is memory important?

Without memory, every workflow execution operation is isolated. AI doesn't know:

1. What did the user ask 30 seconds ago?
2. What did it answer?
3. The context that the user provided earlier.

For single-task workflows like email sorting (lesson 3), this is fine. But for anything conversational—chatbots, support staff, personal assistants—memory is essential. Research shows that around 70% of users building AI workflows struggle with remembering context. This is the number one source of frustration.

Types of internal memory n8n

n8n offers four memory management methods, each with its own advantages and disadvantages:

Memory type	Storage	Durability	Queue mode	Best for
Simple Memory	RAM in progress	? Lost upon completion of execution	? Broken	For testing purposes only.
Window Buffer	RAM in progress	? Lost upon completion of execution	? Broken	Experiment with a limited context.
PostgreSQL	Database	? Long-term storage	? Active	Production chatbot
Redis	Cache in memory	? Long-term storage (configurable)	? Active	High throughput, real-time

Development process : Start with Simple Memory for testing ? switch to PostgreSQL for production environment ? add Redis if sub-millisecond read speeds are needed.

Let's examine each type.

Simple Memory: For testing purposes only.

Simple Memory stores conversation history in the workflow's runtime memory. This is the quickest way to set it up – just attach it to your AI Agent and it's ready to go.

Why shouldn't you use it in a production environment?

1. The conversation history disappears as soon as the execution process is complete.
2. If n8n restarts, all memory will be lost.
3. It doesn't work in queue mode - this is the n8n recommended setup for handling concurrent users.

Think of Simple Memory as a notebook that gets torn up after each conversation. Good for testing your prompts. Useless for real users.

PostgreSQL Memory: Default for production environments

PostgreSQL Memory stores conversations in a physical database. It is retained after restarts, supports concurrent access, and can be queried using SQL.

Establish:

1. You need a PostgreSQL database (n8n Cloud includes one, or you can run your own database).
2. Add a Postgres Chat Memory child node to your AI Agent.
3. Configure connection information (server, database, user, password)
4. Set **Session ID** - this is key to grouping messages into conversations.

The session ID is very important . It tells n8n which conversation to load. A chat trigger usually provides this ID via the command line, `{{ $json.sessionId }}`or you can get it from the user ID.

? **Quick test** : Two users chat with your bot simultaneously. Both store their conversations in PostgreSQL Memory. How does the agent keep their conversations separate?

Answer : Session ID. Each user receives a unique session ID. When User A sends a message, the system only loads User A's conversation history from PostgreSQL. User B's history is stored separately. Without a separate session ID, both users would share the same conversation – this would cause confusion and violate privacy.

Redis Memory: For speed

Redis Memory stores conversations in Redis – an extremely fast in-memory data store (reads in milliseconds).

It is ideal for:

1. The bot has high throughput, handling hundreds of conversations simultaneously.

2. Real-time applications where latency is a critical factor.
3. Conversations need to expire automatically (Redis supports TTL - time to live).

Establish:

Similar to PostgreSQL, this adds a **Redis Chat Memory** child node , provides your Redis connection, and sets a session ID. The main difference: You can set a TTL so that conversations automatically expire after a certain period of time (useful for support chats that don't require a permanent history).

For most workflows, PostgreSQL is the right choice. Redis is used when you're optimizing performance at scale.

Window Buffer: Context Limitation

Window Buffer isn't a type of storage—it's a strategy. It only retains the N most recent messages in the context window instead of the entire conversation history.

Why limit messages? LLM models have token limits. A conversation with 200 messages can exceed the model's context window, causing errors or silent truncation. The Window Buffer retains the most recent messages (e.g., the 20 most recent messages) so the agent always has recent context without exceeding the token limit.

You combine Window Buffer with a storage system:

1. Window Buffer + PostgreSQL = store everything, load the N most recent messages
2. Window Buffer + Redis = same idea, faster reading

Building a chatbot with persistent memory.

Upgrade your research agent from Lesson 4 with PostgreSQL Memory.

Step 1: Start with the agent from Lesson 4

Open the Multi-Tool Research Agent that you built in Lesson 4 (Activate chat ? AI Agent with tools).

Step 2: Add PostgreSQL Memory

1. Click on the AI ??Agent node
2. In the **Memory** section , add **the Postgres Chat Memory child node.**
3. Configure the connection to your PostgreSQL instance.
4. Set **Session ID Key** :`{{ $json.sessionId }}`

If you're using n8n Cloud, you can use n8n's built-in PostgreSQL. If you're self-hosting, connect to any PostgreSQL database.

Step 3: Update the system prompt.

Add memory management instructions to your system prompt:

```
B?n là tr? lý nghiên c?u v?i kh? n?ng ghi nh? h?i tho?i. Khi ng??i dùng ??
t câu h?i ti?p theo: - Tham kh?o thông tin t? ph?n tr??c c?a cu?c h?i tho?
```

```
i - Không l?p l?i thông tin b?n ?ã cung c?p - N?u ng??i dùng nói "nh? tôi ?
ã ?? c?p" ho?c "nh? chúng ta ?ã th?o lu?n", hãy ki?m tra l?i trí nh? c?a b?
n Khi chào ?ón ng??i dùng quay l?i: - Xác nh?n cu?c h?i tho?i tr??c ?ó n?
u có liên quan - ??ng b?t ??u l?i t? ??u m?i l?n
```

Step 4: Check for continuity

Click on "Test workflow" and conduct a multi-round conversation:

1. "What is the population of Japan?"
2. "How does it compare to South Korea?"
3. Which country has a higher GDP per capita?

Without sufficient memory, the second question would fail – the chatbot wouldn't know what "that" is. With PostgreSQL Memory, the chatbot will load previous conversations and understand the context.

Now, close the conversation and reopen it (using the same session ID). Ask: "What were we talking about?". The chatbot will remember your discussion about Japan/Korea – because it's stored in the database.

? **Quick check** : Your chatbot works in the test environment but forgets conversations in the production environment. What's the first thing to check?

Answer : Session ID. During testing, Chat Triggers can generate a static session ID. In a production environment, each user needs a consistent, unique session ID—typically generated from their user ID or authentication token. If the session ID changes between messages or defaults to a random value, memory will load a new conversation each time.

Memory and token costs

The more memory there is, the more tokens are needed for each request. Every message in the conversation history is sent to the LLM as context. A conversation of 50 messages can add more than 5,000 tokens to each subsequent request.

Cost management strategies:

1. **Window Buffer** : Loads only the 10-20 most recent messages instead of the entire history.
2. **Summary Memory** : Periodically summarizes old messages into a concise summary (requires a separate LLM call).
3. **Selective Memory** : Only store messages containing important context (user preferences, decisions) - ignore casual messages.

For most use cases, a Window Buffer of 15-20 messages is the optimal balance between context awareness and cost.

Key points to remember

1. The memory is simply for testing purposes - data will disappear after execution and errors will occur in the queue.
2. PostgreSQL memory is the default for production environments—sustainable and queryable.
3. Redis Memory is designed for high-throughput scenarios where reading data in milliseconds is critical.
4. Window Buffer limits the number of messages that agents load – essential for managing token costs.
5. The session ID identifies which conversation the agent loads – if set incorrectly, all messages will start over from the beginning.
6. Keep your system prompts up-to-date to be aware of memory usage – let the agent know how to use your conversation history.

1. Question 1:

   You're building a customer support bot that handles 50 concurrent users. What type of memory should you use?

   1. A. Simple Memory - easiest to set up
   2. B. PostgreSQL Memory - robust, queryable, and queue-based.
   3. C. Window Buffer Memory with a 100-message buffer.

   EXPLAIN:

   PostgreSQL Memory stores conversations in the database, persists after restarts, supports concurrent access, and works perfectly in queue mode. Redis Memory also works (faster read speeds but poorer query capabilities). Simple Memory will not work in queue mode. Window Buffer Memory is a strategy (limited to the N most recent messages), not a storage system – you'll integrate it with PostgreSQL.

2. Question 2:

   A user reported that your chatbot repeated information they had already provided. What is the most likely cause?

   1. A. The LLM model is too small.
   2. B. The memory is not configured with a session ID, so each conversation starts over from the beginning - or the session ID changes between messages.
   3. C. Prompt system is too vague.

   EXPLAIN:

   Memory is locked by session ID. If the session ID is missing, changes, or defaults to a new value each time, the agent will start each conversation with a blank page. Check that your Chat Trigger is passing a consistent session ID (such as user ID or chat room ID) to the memory node. A static session ID means the agent remembers it; a random ID means it forgets it.

3. Question 3:

   Why is Simple Memory unsuitable for AI processes in a production environment?

1. A. Simple memory is slower than other types of memory.
2. B. Simple Memory stores conversations during the runtime of a process - data is lost when the process finishes execution, and it does not operate in queue mode.
3. C. Simple Memory can only store 10 messages.

EXPLAIN:

Simple Memory only works in memory. When the process execution ends, the conversation history disappears. Worse still, it's completely broken in queue mode (the production configuration recommended by n8n for concurrent processes). Use it only for quick testing, never for a production environment.

Submit your work

Training results

You have completed **0** questions.

-- / --

Review the lesson