생성형 AI 기반 RAG 개발자 과정

Chat History 적용

Session 23 / 33 — Day 3

"그건 얼마야?" — 대화가 이어지는 RAG

박수현 · (사)한국정보공학기술사회 · 2026

LLM 이론 · LangChain · ChromaDB CRUD · Agent — 4일 핸즈온

출처: Day 3 Chat History

이번 시간을 마치면

🧵 대화 유지

- **session** 별 대화 이력
- 이전 대화 **활용**
- Context Window 관리

🔁 history-aware

- 후속 질문 **재작성**
- 대화 맥락 + 검색 결합
- 독립 질문 ↔ 후속 질문

> 🎯 목표: 후속 질문("그럼 기간은?")까지 정확히 답하는 **대화형 RAG** 구현.

출처: Day 3 Chat History

"그건 며칠이야?" — 후속 질문의 검색 실패

문제

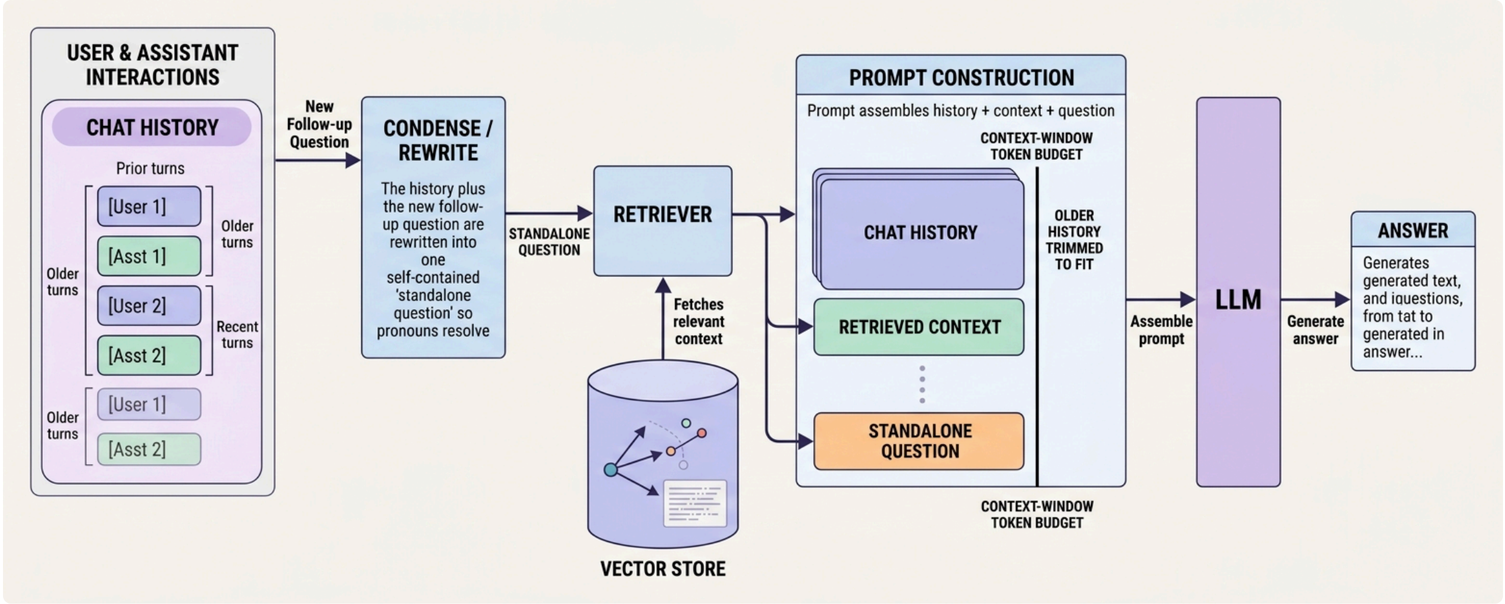
- 사용자: "환불 돼요?" → (답변)
- 사용자: "**그럼 며칠**이에요?"
- 검색기에는 "그럼 며칠"만 전달 → **무엇의** 며칠인지 미상
- → 무관한 chunk 검색

해결 방향

- 검색 **이전에** 후속 질문을
- 대화 이력으로 **독립 질문**으로 재작성
- "그럼 며칠?" → "환불 가능 기간은 며칠?"
- 재작성된 질문으로 검색 → 정확도 확보

> 📁 참고: `7.RAG/5.conversational/5.1_followup_problem.py`

후속 질문을 독립 질문으로



🖼️ 대화이력 인지 RAG · 이력+질문→독립 질문 재작성→Retriever→context→Prompt→LLM→답변

💡 핵심은 **검색 이전 단계의 질문 재작성**(history-aware)이다. 이력을 검색기에 직접 주입하지 않고, 질문 자체를 독립적으로 완결시킨다.

출처: Day 3 Chat History

session_id 로 대화를 격리 (스니펫)

```
from langchain_core.chat_history import InMemoryChatMessageHistory
from langchain_core.runnables.history import RunnableWithMessageHistory

sessions = {}   # {session_id: InMemoryChatMessageHistory}

def get_session_history(session_id):
    return sessions.setdefault(session_id, InMemoryChatMessageHistory())

# 세션 이력의 로드·저장을 자동화 — 호출 시 session_id 만 지정
conversational_rag = RunnableWithMessageHistory(
    rag_chain, get_session_history,
    input_messages_key="input",
    history_messages_key="chat_history",
    output_messages_key="answer")
```

- `RunnableWithMessageHistory` 가 세션별 이력 **자동 로드·저장**
- `InMemoryChatMessageHistory` 는 휘발 → 실서비스는 **Redis/DB** 로 교체

> 🔑 다중 사용자는 `session_id` 로 격리하며, 라우트는 세션 ID 만 전달한다.

출처: Day 3 Chat History

두 체인 결합 — LangChain 헬퍼 (스니펫)

```
from langchain.chains import (
    create_history_aware_retriever, create_retrieval_chain)
from langchain.chains.combine_documents import create_stuff_documents_chain

# ① 검색: 이력 기반 쿼리 재작성 → retriever
history_aware = create_history_aware_retriever(llm, retriever, rewrite_prompt)
# ② 답변: 검색 문서 {context} + chat_history + input
doc_chain = create_stuff_documents_chain(llm, qa_prompt)
# 두 체인 결합 — 검색·답변 양쪽이 history 를 참조
rag_chain = create_retrieval_chain(history_aware, doc_chain)
```

- **두 단계**에서 이력 사용: 검색 단계(쿼리 재작성) + 답변 단계(맥락 반영)
- `qa_prompt` 에 `MessagesPlaceholder("chat_history")` 로 이력 자리 확보

> 📁 전체: `7.RAG/5.conversational/5.3_full_conversational_rag.py`

출처: Day 3 Chat History

이력이 길어지면 — 잘라내거나 요약

구조적 상한

- 모델 context window (예: 128K)
- 이력 + 근거 + 질문이 모두 수용되어야 함

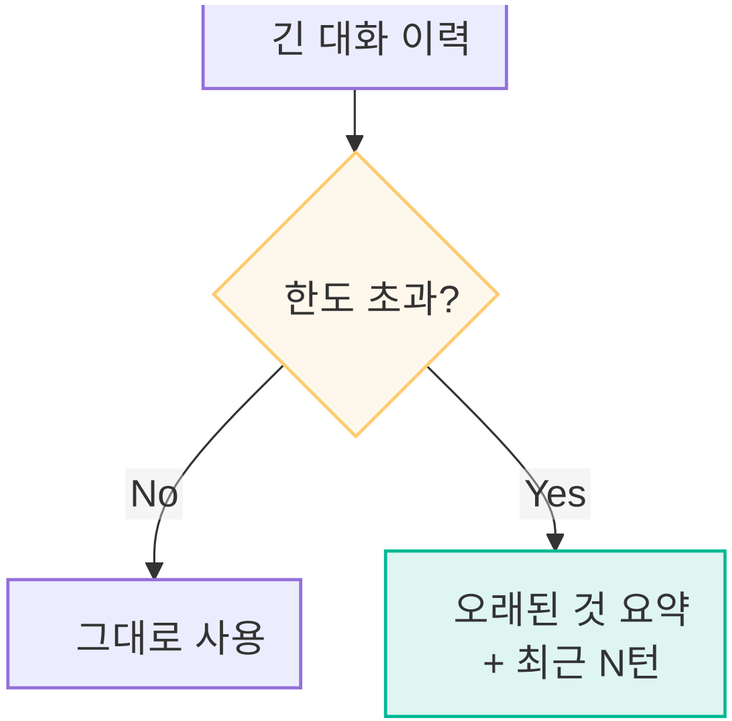
전략 3가지

- **최근 N턴**만 유지 (sliding window) — 단순, 원거리 맥락 손실
- 오래된 대화 **요약** 압축 — 맥락 보존, 요약 비용 발생
- **토큰 예산** 기반 — 정밀, 토큰 계산 필요

> 💡 history-aware 패턴은 질문을 **독립 쿼리로 재작성**하므로, 검색 정확도 측면에서 긴 이력이 불필요하다. 짧은 이력만으로 충분하다.

출처: Day 3 Chat History

한도 초과 시 — 요약 + 최근 N턴



한도 미만이면 이력 그대로, 초과하면 오래된 대화는 요약하고 최근 N턴만 유지

핵심 5가지

1. ✅ 후속 질문은 **앞 대화를 지시** → 원문 그대로 검색하면 실패
2. ✅ **history-aware** — 검색 전에 질문을 독립 질문으로 재작성
3. ✅ **session_id** 로 대화 이력 격리 (실서비스는 DB)
4. ✅ 이력은 **두 곳**에서 — 검색(쿼리 재작성) + 답변(맥락 반영)
5. ✅ 이력이 길면 **최근 N턴 / 요약 / 토큰 예산**으로 context 관리