
- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: https://smith.langchain.com/hub/teddynote/summary-stuff-documents

---

SQL (SQLAlchemy)

> Structured Query Language (SQL)
>
> - 프로그래밍에 사용되는 도메인 특화 언어
> - 관계형 데이터베이스 관리 시스템(RDBMS)에서 데이터를 관리하거나 관계형 데이터 스트림 관리 시스템(RDSMS)에서 스트림 처리를 위해 설계
> - 엔티티와 변수 간의 관계를 포함하는 구조화된 데이터를 다루는 데 유용
>
> SQLAlchemy 는 MIT 라이선스에 따라 배포되는 Python 프로그래밍 언어용 오픈 소스 SQL 툴킷이자 객체 관계 매퍼(ORM)

- 노트북에서는 SQLAlchemy 가 지원하는 모든 데이터베이스에 채팅 기록을 저장할 수 있는 SQLChatMessageHistory 클래스에 대해 설명
- SQLite 이외의 데이터베이스와 함께 사용하려면 해당 데이터베이스 드라이버를 설치해야 함

```
# 환경변수 처리 및 클라이언트 생성
from langsmith import Client
from dotenv import load_dotenv

import os
import json

# 클라이언트 생성
api_key = os.getenv("LANGSMITH_API_KEY")
client = Client(api_key=api_key)
```

```
# LangSmith 추적 설정하기 (https:smith.langchin.com)
# LangSmith 추적을 위한 라이브러리 임포트
from langsmith import traceable

# LangSmith 환경 변수 확인

print("\n--- LangSmith 환경 변수 확인 ---")
langchain_tracing_v2 = os.getenv('LANGCHAIN_TRACING_V2')
langchain_project = os.getenv('LANGCHAIN_PROJECT')
langchain_api_key_status = "설정됨" if os.getenv('LANGCHAIN_A
org = "설정됨" if os.getenv('LANGCHAIN_ORGANIZATION') else "설

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='
    print(f"✅ LangSmith 프로젝트: '{langchain_project}'")
    print(f"✅ LangSmith API Key: {langchain_api_key_status
    print("  -> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요
else:
    print("❌ LangSmith 추적이 완전히 활성화되지 않았습니다. 다음을 확
    if langchain_tracing_v2 != "true":
        print(f"  - LANGCHAIN_TRACING_V2가 'true'로 설정되어 있
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("  - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")
```

"@traceable" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

- 셀 출력

```
--- LangSmith 환경 변수 확인 ---
✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='true')
✅ LangSmith 프로젝트: 'LangChain-prantice'
```

```
✅ LangSmith API Key: 설정됨
-> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.
```

```
import os
from dotenv import load_dotenv
import openai

from langchain_openai import ChatOpenAI

# .env 파일에서 환경변수 불러오기
load_dotenv()

# 환경변수에서 API 키 가져오기
api_key = os.getenv("OPENAI_API_KEY")

# OpenAI API 키 설정
openai.api_key = api_key

# OpenAI를 불러오기
# ✅ 디버깅 함수: API 키가 잘 불러와졌는지 확인
def debug_api_key():
    if api_key is None:
        print("❌ API 키를 불러오지 못했습니다. .env 파일과 변수명을 확인하세요.")
    elif api_key.startswith("sk-") and len(api_key) > 20:
        print("✅ API 키를 성공적으로 불러왔습니다.")
    else:
        print("⚠️ API 키 형식이 올바르지 않은 것 같습니다. 값을 확인하세요.")

# 디버깅 함수 실행
debug_api_key()
```

- 셀 출력

```
✅ API 키를 성공적으로 불러왔습니다.
```

사용 방법

- `storage`를 사용하기 위해 필요한 두 가지
  - **`session_id`** - 사용자 이름, 이메일, 채팅 ID 등과 같은 **세션의 고유 식별자**
  - **`connection`**
    - 데이터베이스 연결을 지정하는 문자열
    - **SQLAlchemy의 `create_engine` 함수에 전달됨**

```
from langchain_community.chat_message_histories import SQLChatMessageHistory

# SQLChatMessageHistory 객체를 생성하고 세션 ID와 데이터베이스 연결 파일을 설정
chat_message_history = SQLChatMessageHistory(
    session_id="sql_history",
    connection="sqlite:///sqlite.db"
)
```

```
# 사용자 메시지 추가
chat_message_history.add_user_message(
    "안녕? 만나서 반가워. 내 이름은 앨리스야. 나는 랭체인 개발자야. 앞으로 잘 부탁해!"
)

# AI 메시지 추가
chat_message_history.add_ai_message("안녕 앨리스, 만나서 반가워. 나도 잘 부탁해!")
```

- **`chat_message_history.messages`** - 저장된 대화 내용 확인하기

```
# 채팅 메시지 기록의 메시지들

chat_message_history.messages
```

- 셀 출력

```
[HumanMessage(content='안녕? 만나서 반가워. 내 이름은 테디야. 나는 랭체인 개발자야. 앞으로 잘 부탁해!', additional_kwargs={}, respo
AIMessage(content='안녕 테디, 만나서 반가워. 나도 잘 부탁해!', additional_kwargs={}, response_metadata={}),
HumanMessage(content='안녕? 만나서 반가워. 내 이름은 앨리스야. 나는 랭체인 개발자야. 앞으로 잘 부탁해!', additional_kwargs={}, resp
AIMessage(content='안녕 앨리스, 만나서 반가워. 나도 잘 부탁해!', additional_kwargs={}, response_metadata={})]
```

```
print(type(chat_message_history))       # <class 'langchain_community.chat_message_histories.sql.SQLChatMessageHistory'>
```

Chain에 적용

- 이 메세지 기록 클래스를 **LCEL Runnabels** 와 결합시키기

```
from langchain_core.prompts import (
    ChatPromptTemplate,
    MessagesPlaceholder,
)
from langchain_core.runnables.history import RunnableWithMessageHistory
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser


prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "You are a helpful assistant."),                    # 시스템 메시지
        MessagesPlaceholder(variable_name="chat_history"),             # 대화 기록을 위한 Placeholder
        ("human", "{question}"),                                       # 질문
    ]
)

# LLM 생성
llm = ChatOpenAI(
    #temperature=0,
    openai_api_key=api_key,
    model="gpt-4o-mini",
    )

# chain 생성
chain = prompt | llm | StrOutputParser()
```

- **sqlite.db**`에서 대화내용을 가져오는 함수 만들기

```
def get_chat_history(user_id, conversation_id):
    return SQLChatMessageHistory(
        table_name=user_id,
        session_id=conversation_id,
        connection="sqlite:///sqlite.db",
    )
```

- **config_fields** 설정 = 대화정보 조회 시 참고 정보로 활용
  - **user_id** : 사용자 ID
  - **conversation_id** : 대화 ID

```
from langchain_core.runnables.utils import ConfigurableFieldSpec
```

```
config_fields = [
    ConfigurableFieldSpec(
        id="user_id",
        annotation=str,
        name="User ID",
        description="Unique identifier for a user.",
        default="",
        is_shared=True,
    ),
    ConfigurableFieldSpec(
        id="conversation_id",
        annotation=str,
        name="Conversation ID",
        description="Unique identifier for a conversation.",
        default="",
        is_shared=True,
    ),
]


chain_with_history = RunnableWithMessageHistory(
    chain,
    get_chat_history,                                   # 대화 기록 가져오는 함수 설정
    input_messages_key="question",                      # 입력 메시지의 키를 "question"으로 설정
    history_messages_key="chat_history",                # 대화 기록 메시지의 키를 "history"로 설정
    history_factory_config=config_fields,               # 대화 기록 조회시 참고할 파라미터 설정
)
```

- **configurable** 키 아래에 **user_id**, **conversation_id** 의 key-value 쌍 설정하기

```
# config 설정

config = {"configurable": {"user_id": "user1", "conversation_id": "conversation1"}}
```

질문 해보기

- **chain_with_history** 객체의 **invoke** 메서드 호출 → 질문에 대한 답변 생성하기
- **invoke** 메서드: **질문 딕셔너리** + **config 설정** 이 전달됨

```
# 질문과 config를 전달해서 실행해보기

chain_with_history.invoke({"question": "안녕 반가워, 내 이름은 앨리스야"}, config)
```

- 셀 출력 (1.3s)

```
'안녕하세요, 앨리스! 만나서 반가워요. 오늘 무엇을 이야기하고 싶으신가요?'
```

```
# 후속 질문 해보기

chain_with_history.invoke({"question": "내 이름이 뭐라고?"}, config)
```

- 셀 출력 (0.8s)

```
'당신의 이름은 앨리스라고 하셨습니다. 맞나요?'
```

- 같은 **user_id**, 다른 **conversation_id** 가지도록 설정해보기

```
# config 재설정
config = {"configurable": {"user_id": "user1", "conversation_id": "conversation2"}}         # 다른 conversation_id
```

```
# 질문과 config를 전달해 실행하기
chain_with_history.invoke({"question": "내 이름이 뭐라고?"}, config)
```

- 셀 출력 (1.4s)

```
'죄송하지만, 당신의 이름을 알 수 있는 방법이 없습니다. 이름에 대해 알려주시면, 더 나은 대화를 나눌 수 있을 것 같습니다! 어떻게 도와드릴까요?'
```

다른 예시로 test

- **탐정 AI**
  - 사용자가 범죄 사건의 세부 사항을 말하고, AI 가 그걸 기억하며 추리 하는 시나리오
  - **AI 가 이전 정보를 제대로 기억 하는지 확인 가능**

- **시나리오 흐름**
  - **용의자 알렉스, 장소 도서관, 증거 피 묻은 책** 정보를 히스토리에 저장
  - 두 번째 질문에서 AI 가 용의자를 기억 하며 분석
  - 세 번째 질문에서 AI 가 이전 장소/증거 정보를 기억해 **도서관에서 발견됐다** 처럼 답변 = **기억 테스트 가능**
  - 네 번째 질문 이후: **새 정보 추가 + 전체 맥락 유지 하는지 확인해보기**

```
# 새로운 유저와 대화 세션 지정하기

config = {
    "configurable": {
        "user_id": "user3",                                 # 새로운 사용자 ID
        "conversation_id": "conversation3"                  # 새로운 대화 세션 ID
        }
    }


# 첫 질문

response = chain_with_history.invoke(
    {"question": "탐정님, 살인 사건이야. 용의자는 알렉스, 장소는 도서관, 증거는 피 묻은 책이야."},
    config
)
print(response)
```

- 셀 출력 (3.9s)

```
안녕하세요! 이 사건을 해결하기 위해 몇 가지 질문을 해보겠습니다.

1. **알렉스의 알리바이**: 사건이 발생한 시간에 알렉스는 어디에 있었나요? 혹시 다른 목격자가 있나요?
2. **피 묻은 책**: 피 묻은 책은 어떤 책인지, 그 책이 사건과 어떤 관련이 있는지 알고 있나요? 혹시 피해자가 자주 읽던 책인가요?
3. **도서관의 CCTV**: 도서관에 CCTV가 설치되어 있다면, 사건 당시의 녹화 영상은 확인할 수 있나요?
4. **피해자에 대한 정보**: 피해자는 누구이며, 알렉스와의 관계는 어떤가요? 그들 사이에 갈등이 있었나요?
5. **추가 증거**: 현장에서 발견된 다른 증거나, 용의자에 대한 추가적인 정보가 있나요?

이 정보를 바탕으로 더 깊이 사건을 분석해보겠습니다.
```

```
# 이어서 질문하기

response2 = chain_with_history.invoke(
    {"question": "용의자의 이름이 뭐야?"},
    config
)
print(response2)
```

- 셀 출력 (1.6s)

용의자의 이름은 알렉스입니다. 추가로 알고 싶은 사항이 있으시면 말씀해 주세요! 사건을 더 깊이 파악하는 데 도움이 됩니다.

```
# 이어서 질문하기

response3 = chain_with_history.invoke(
    {"question": "용의자가 왜 의심되는 거지?"},
    config
)
print(response3)
```

- 셀 출력 (3.7s)

  알렉스가 용의자로 의심되는 이유는 다음과 같은 요소들로 추측할 수 있습니다:

  1. **피 묻은 책**: 사건 현장에서 발견된 피 묻은 책이 알렉스와 관련이 있을 가능성이 있습니다. 예를 들어, 그 책이 알렉스의 소유물이거나 알렉스가 자

  2. **동기**: 알렉스와 피해자 사이에 어떤 갈등이나 불화가 있었을 경우, 이는 알렉스를 용의자로 만드는 중요한 요인이 될 수 있습니다. 그들이 과거에 어

  3. **알리바이 여부**: 사건 발생 시간에 알렉스가 다른 곳에 있었다는 강력한 알리바이가 없으면, 그는 의심을 받을 수 있습니다. 알리바이가 불확실하다면

  4. **목격자 진술**: 도서관에 있었던 목격자들이 알렉스를 봤다는 진술이 있다면, 그의 행동이 의심스럽다면 추가적인 의혹이 가해질 수 있습니다.

  이러한 요소들이 알렉스를 의심하게 만드는 주된 이유라고 할 수 있습니다. 사건을 더 조사하고, 다른 증거나 목격자의 진술이 있다면 알렉스의 혐의를 더 깊이

```
# 이어서 질문하기

response4 = chain_with_history.invoke(
    {"question": "그 증거는 어디에서 발견됐어?"},
    config
)
print(response4)
```

- 셀 출력 (3.9s)

  피 묻은 책은 사건의 장소인 도서관에서 발견되었습니다. 이는 사건의 중요한 증거로, 아래와 같은 점들을 고려해야 할 필요가 있습니다:

  1. **위치**: 피 묻은 책이 도서관의 어떤 특정 장소(예: 독서 공간, 서가 등)에서 발견되었는지. 이 위치가 알렉스나 피해자와 어떤 관련이 있는지를 살피

  2. **책의 내용**: 피가 묻은 책의 제목이나 주제가 사건과 관련이 있을 수 있습니다. 피해자가 자주 읽던 책이라면 그 또한 의심스러운 요소가 될 수 있습

  3. **증거 수집**: 책에서 채취한 피의 유전자 분석 결과가 용의자 알렉스와 일치한다면, 이는 더욱 결정적인 증거가 됩니다.

  4. **발견 경위**: 이 책이 도서관에서 발견된 경위는 어떻게 되는지, 누가 발견했으며 어떤 상황에서 발견되었는지에 대한 조사도 중요합니다.

  이러한 요소들을 바탕으로 사건을 파악하면 알렉스가 용의자로 의심되는 이유를 더 명확하게 이해할 수 있습니다. 추가적인 정보가 있다면 공유해 주세요! 사건

```
# 이어서 질문하기

response5 = chain_with_history.invoke(
    {"question": "새 증인이 나타났어. 이름은 베티야. 알렉스와 다퉜대."},
    config
)
print(response5)
```

- 셀 출력 (4.7s)

  베티라는 새 증인이 등장하여 알렉스와의 다툼이 있었다는 사실은 사건에 중요한 추가 정보를 제공합니다. 이 내용을 바탕으로 몇 가지 질문과 고려할 점이 있

  1. **다툼의 이유**: 알렉스와 베티가 왜 다퉜는지, 그 다툼이 어떤 내용을 포함하고 있었는지 파악해야 합니다. 감정적으로 격해질만한 이유가 있었다면, 이

  2. **다툼의 시기**: 이 다툼이 사건 발생 전후 언제 있었는지도 중요합니다. 사건 발생 직전에 다툴 경우, 알렉스가 감정적으로 불안정했을 가능성이 있습니

3. **베티의 진술**: 베티의 진술이 신뢰할 수 있는지, 혹은 다른 목격자와 일치하는 부분이 있는지를 확인해야 합니다. 그녀가 어떤 상황에서 다퉜는지 구체

4. **알렉스의 반응**: 베티와 다퉜던 알렉스의 행동이나 반응을 다른 사람이나 그 당시 상황과 비교해 보아야 합니다. 그가 평소와 다르게 행동했는지 확인

5. **증거와의 연결**: 베티의 진술이 피 묻은 책이나 다른 증거와 어떻게 연결될 수 있는지도 고려해야 합니다. 알렉스가 베티와의 다툼 이후에 피해자와의

베티의 증언이 사건 해결의 열쇠가 될 수 있으므로, 그녀의 진술을 정확하게 수집하고 분석하는 것이 중요합니다. 추가적인 정보나 다른 증언이 있다면 공유해

```
# 이어서 질문하기

response6 = chain_with_history.invoke(
    {"question": "지금까지의 정보 중 중요한 내용만 정리해줘."},
    config
)
print(response6)
```

- 셀 출력 (4.6s)

현재까지의 사건 관련 중요한 정보를 아래와 같이 정리하였습니다:

1. **사건 개요**:
- **사건 유형**: 살인 사건
- **장소**: 도서관
- **주요 증거**: 피 묻은 책

2. **용의자**:
- **이름**: 알렉스
- **의심 이유**:
  - 피 묻은 책이 알렉스와 관련이 있을 가능성
  - 피해자와의 관계 및 갈등이 있었을 가능성
  - 사건 발생 시간에 대한 알리바이 불확실성

3. **새 증인**:
- **이름**: 베티
- **다툼 정보**: 알렉스와 베티가 다퉜음.
  - 다툼의 이유, 시기 및 내용이 사건에서 중요할 수 있음.
  - 베티의 진술이 사건에 대한 정보 제공 가능성이 높음.

4. **증거 조사를 위한 고려 사항**:
- 피 묻은 책의 위치 및 내용
- 베티의 진술 신뢰도 및 알렉스의 반응
- 알렉스와 베티 사이의 다툼과 사건 간의 연결성

이 정보를 바탕으로 사건을 더 깊이 분석하고, 가능성 있는 의혹이나 다른 증거들을 확인하며 조사를 이어나가는 것이 중요합니다. 추가적인 정보나 질문이 있

```
# 이어서 질문하기

response7 = chain_with_history.invoke(
    {"question": "베티에게 무엇을 확인하면 좋을지 질문 목록을 적어줘."},
    config
)
print(response7)
```

- 셀 출력 (4.9s)

베티에게 확인하고 물어볼 질문 목록은 아래와 같습니다:

1. **다툼의 이유**:
- 알렉스와 다툰 이유는 무엇인가요?
- 그 다툼이 어떤 상황에서 발생했는지 설명해 줄 수 있나요?

2. **다툼의 시간**:
- 다툼이 사건 발생 전에, 혹은 사건이 일어난 직후였나요?
- 다툼이 발생한 정확한 시간이나 날짜를 기억하시나요?

3. **구체적인 행동**:
- 알렉스가 다툼 중에 어떤 행동을 보였나요? 격한 반응이나 위협적인 행동이 있었나요?
- 다툼 후 알렉스가 어떻게 행동했나요?

4. **증거 관련**:
- 다툼 당시 도서관에 다른 사람들이 있었나요? 만약 있었다면 그들이 목격한 내용도 있을까요?
- 다툼 중에 알렉스가 피해자에 대해 언급한 것이 있었는지요?

5. **알렉스의 감정**:
- 알렉스의 감정 상태는 어땠나요? 예를 들어, 분노했거나 불안해 보였나요?
- 알렉스가 다툼 후에 혹시 특이한 행동이 있었나요?

6. **연관성**:
- 다툼이 사건에 어떤 관련이 있을 수 있다고 생각하시나요?
- 사건과 관련하여 아는 다른 정보나 목격자가 있나요?

7. **신뢰성**:
- 다툼의 상황을 누구에게 전달했나요? 신뢰할 수 있는 사람들이 있나요?

이 질문들을 통해 베티의 진술을 명확하게 확인하고, 알렉스와 피해자 간의 관계 및 사건 발생과의 연관성을 더 깊이 이해할 수 있을 것입니다. 추가 질문이

- *next: RunnableWithMessageHistory에 ChatMessageHistory추가*
