

LLM Application

LangGraph 세미나



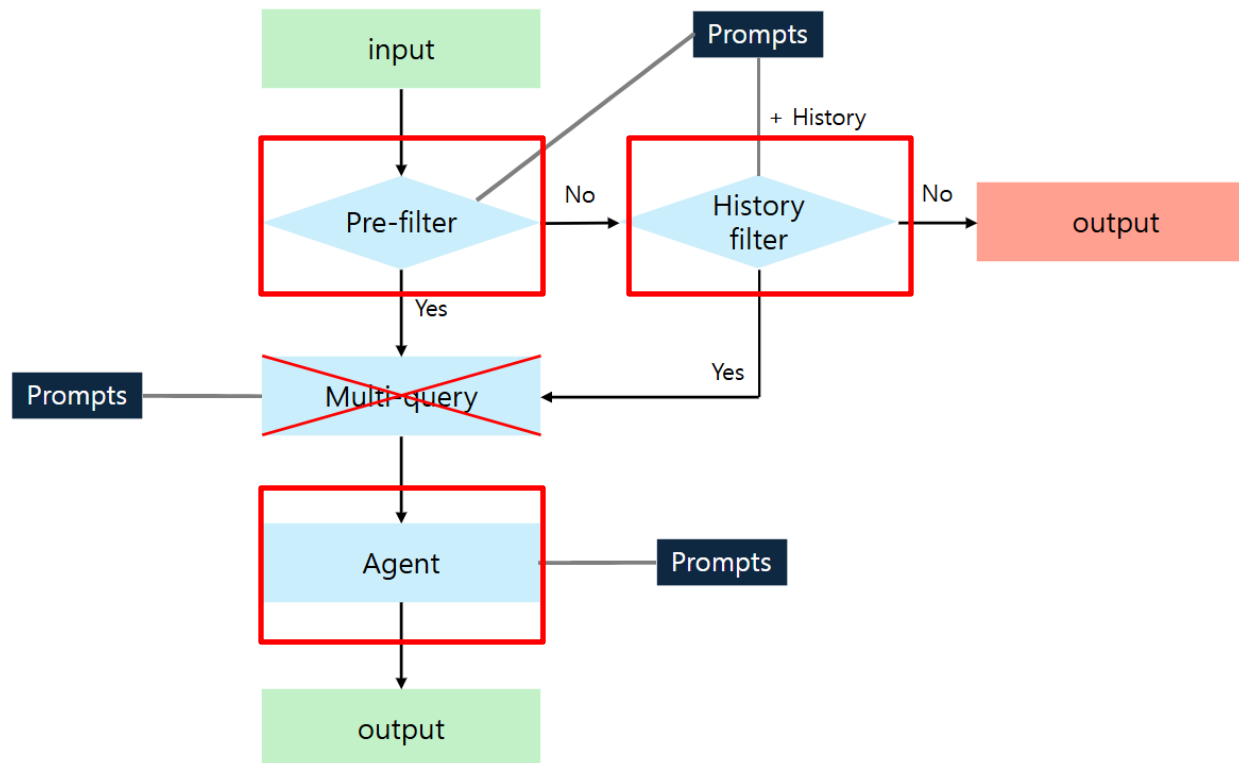
LangGraph

다룰 내용

- MBC+ LangGraph 소개
- 강의 속 알짜배기 모음
 - 판다스 스튜디오
 - 케인
 - AISchool

MBC+ LangGraph 소개

이전 MBC+ LangChain 버전에서 문제

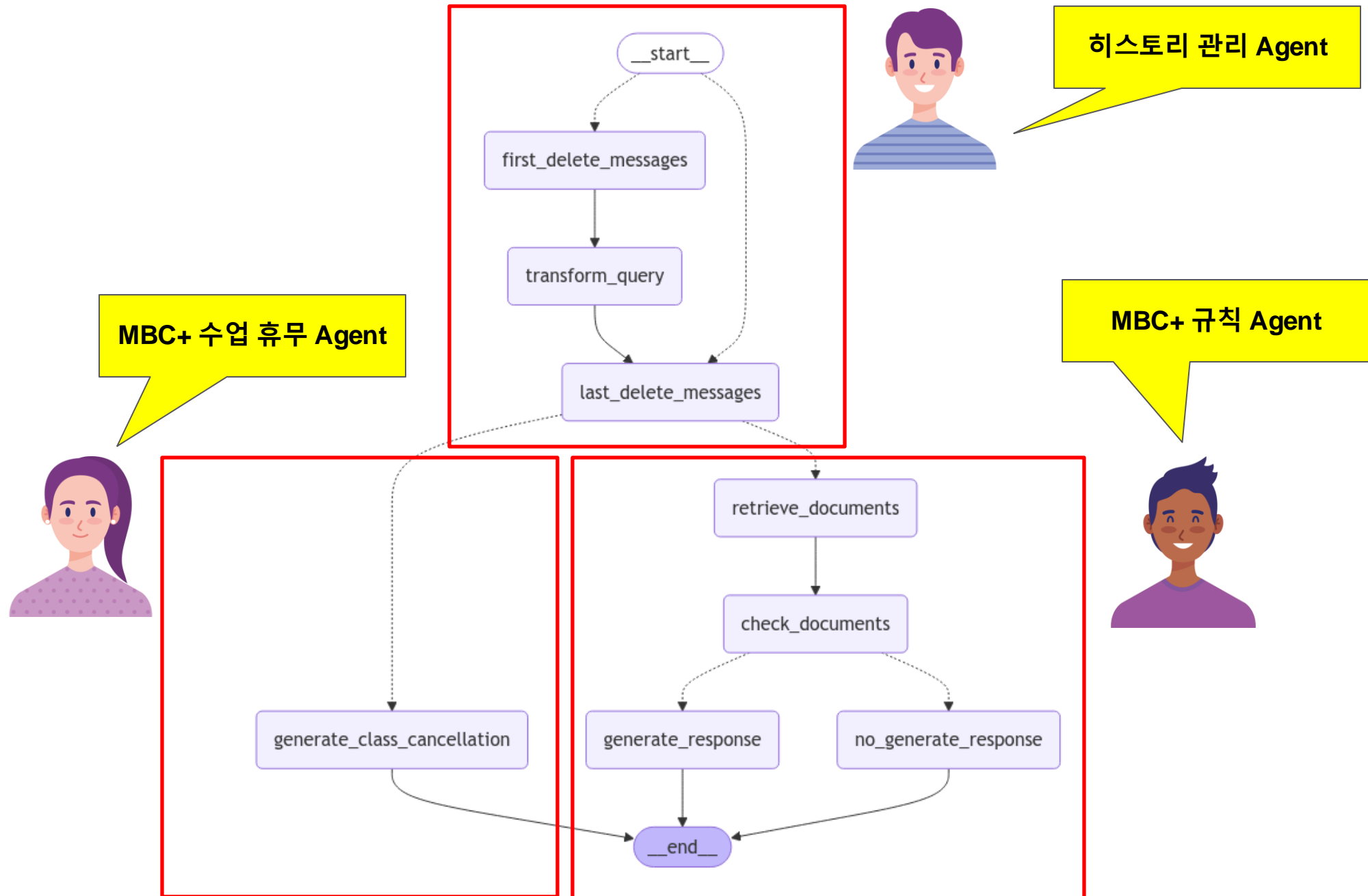


1. **RAG 비용:** 필터와 답변 생성을 위해 RAG를 2~3번 했어야 했습니다.
2. **단일 Agent:** 하나의 Agent가 모든 기능을 수행 해야 했습니다.
3. **제어 불가:** LangChain에서 제공하는 AgentExecutor를 사용해야 됐기 때문에 자체 튜닝이 어려웠습니다.

하지만 LangGraph를 사용하면

1. **RAG 비용:** 필터와 답변 생성을 위해 RAG를 2~3번 했어야 했습니다.
: 최초 Retriever 수행 시 검색된 문서들을 LangGraph 상태로 저장하면 단 한 번의 RAG로 작업을 완료할 수 있습니다.
2. **단일 Agent:** 하나의 Agent가 모든 기능을 수행 해야 했습니다.
: 멀티 에이전트를 활용해 기능을 분배하면 답변의 품질을 향상시키고 응답 시간을 단축할 수 있습니다.
3. **제어 불가:** LangChain에서 제공하는 AgentExecutor를 사용해야 했기 때문에 자체 튜닝이 어려웠습니다.
: LangGraph를 활용하면 원하는 아키텍처를 쉽게 구축할 수 있으며, 버그 발생 시 수정 작업도 간편합니다.

MBC+ 운영 챗봇 LangGraph 아키텍처



히스토리 관리 Agent



유저의 질문이 이전 히스토리과 관련 있는지 확인

유저 질문을 재구성하기 위해
현재 질문 삭제

__start__

YES

NO

first_delete_messages

과거 히스토리를 참고하여 현재 질문의
부족한 맥락을 보완해 재구성

transform_query

히스토리 메시지가 5쌍을 넘으면
가장 오래된 메시지부터 삭제

last_delete_messages

YES

NO

유저의 질문이 수업 휴무와 관련
있는지 확인

transform_query

HUMAN

예비군 훈련 때문에 결석 했을 경우 출석 인정이 되는지 알려줘

AI



안녕하세요! MBC+ 가이드입니다! 😊

예비군 훈련 때문에 결석한 경우 출석이 인정됩니다. 예비군 훈련은 출석이 인정되는 사유 중 하나입니다. 따라서 예비군 훈련에 참여한 시간만큼 출석으로 인정받을 수 있습니다.

이해되셨나요? 혹시 더 궁금한 부분이 있으면 물어보세요!

MESSAGES question 1 ↗

HUMAN

그럼 결혼은?



Rendered Output ▾

결혼은 출석 인정이 되는지 알려주세요.

first_delete_messages

HUMAN

```
예비군 훈련 때문에 결석 했을 경우 출석 인정이 되는지 알려줘
```

AI

안녕하세요! MBC+ 가이드입니다! 😊

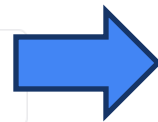
예비군 훈련 때문에 결석한 경우 출석이 인정됩니다. 예비군 훈련은 출석이 인정되는 사유 중 하나입니다. 따라서 예비군 훈련에 참여한 시간만큼 출석으로 인정받을 수 있습니다.

이해하셨나요? 혹시 더 궁금한 부분이 있으면 물어보세요!

MESSAGES question 1

HUMAN

```
그럼 결혼은?
```



Rendered Output ▾

결혼은 출석 인정이 되는지 알려주세요.

MBC+ 수업 휴무 Agent



질문이 수업 휴무와 관련이 있는 경우

YES

수업 휴무 관련 데이터를 활용해
답변 생성

generate_class_cancellation

__end__

MBC+ 규칙 Agent



질문이 수업 휴무와 관련이 없는 경우

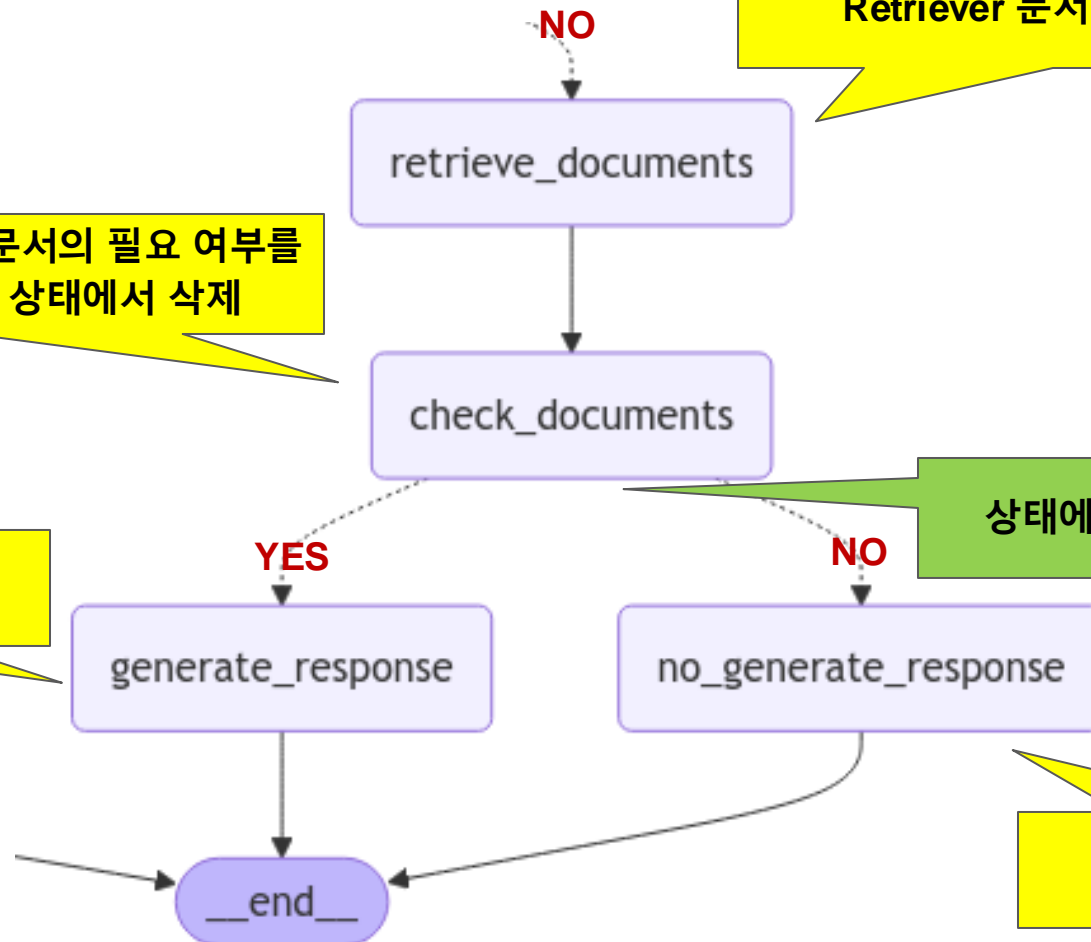
Retriever 문서 검색 및 상태 저장

유저 질문을 바탕으로 검색된 문서의 필요 여부를 판별하고, 불필요한 문서는 상태에서 삭제

답변 생성

상태에 저장된 문서가 있는지 확인

질문할 수 없다는 답변 생성

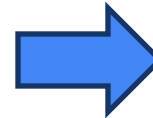


check_documents

불필요한 Document 제거!

DOCUMENTS documents 7

- Q: 출석 일수를 산정할 때 쓰는 단위기간이 무슨 의미인지 A: 전체 훈련기간 중 (주말 및 공휴일 제외) 한달 ...
- 3. (스마트훈련과정(훈련유형과 관계없이 '스마트훈련단가'를 적용받는 훈련과정)) 스마트훈련과정 중 '집...
- 내일배움카드 유형에 따른 훈련장려금 지급 1. "실업자"로 카드를 발급받은 경우 - 훈련장려금 지급 대상...
- 3. 휴가 사용 A. 휴가 신청 i. 휴가를 사용하기 전에 반드시 담당 강사와 운영 담당자에게 사전에 알려야 ...
- 2. (출석입력 요청) 훈련기관이 출석체크를 할 수 없는 경우 해당사유가 발생한 다음날까지... 직접 제작 문서
- 제36조(출석률) ① 훈련생의 출석률은 다음 계산식에 따라 소수점 이하 첫째 자리까지 계... 직접 제작 문서
- 모든 과제와 시험에 성실히 임하며, 부정행위를 하지 않습니다. 교·강사의 저작물을 비롯한... 직접 제작 문서



DOCUMENTS documents 1

- 2. (출석입력 요청) 훈련기관이 출석체크를 할 수 없는 경우 해당사유가 발생한 다음날까지... 직접 제작 문서

LangGraph 기술 공유 - Sqlite MemorySaver

```
from langgraph.checkpoint.sqlite import SqliteSaver
```

from langgraph.checkpoint.sqlite 모듈에서 SqliteSaver 클래스를 가져옵니다.

```
def __memory_init(self):  
    db_path = "sqlite.db"  
    conn = sqlite3.connect(db_path, check_same_thread=False)  
    memory = SqliteSaver(conn)  
    return memory
```

파일경로

Memorysaver는 **check_same_thread=False** 로 생성해야합니다.

```
Number of requested results 20 is greater than number of elements in index 1, updating n_results = 1  
BadRequestError: SQLite objects created in a thread can only be used in that same thread. The object was created in thread id 140079289106432 and this is thread id 140078655456832.
```

- True 로 생성할 경우 발생하는 Error

LangGraph 기술 공유 - Sqlite MemorySaver

```
from langchain_core.messages import RemoveMessage
```

```
def last_delete_messages(self, state: operation_state):  
    print("---- this node is last_delete_messages ----")  
  
    if len(state["messages"]) > 8:  
        return {  
            "messages": [RemoveMessage(id=m.id) for m in state["messages"][:-8]]  
        }
```

히스토리를 삭제할 때는 별도의 노드를 생성해야 하며,
삭제할 메시지의 ID를 RemoveMessage에 담아 리스트 형태로 반환하면 됩니다.

LangGraph 기술 공유 - Sqlite MemorySaver

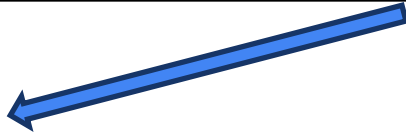
```
messages: Annotated[List, add_messages]
```



```
from langgraph.graph.message import add_messages
```



```
def add_messages(left: Messages, right: Messages) -> Messages:
```



```
Messages = Union[list[MessageLikeRepresentation], MessageLikeRepresentation]
```



```
MessageLikeRepresentation = Union[  
    BaseMessage, list[str], tuple[str, str], str, dict[str, Any]  
]
```



```
class BaseMessage(Serializable):
```

```
    id: Optional[str] = None
```

LangGraph 기술 공유 - 바뀐 pydantic v2 사용법

```
class ClassCancellationCheck(BaseModel):
    """Binary score indicating whether it is related to class cancellation."""
    relevance_to_class_cancellation: str = Field(
        description="Binary relevance score: 'yes' if related to class cancellation, 'no' otherwise."
    )

output_parser = PydanticOutputParser(pydantic_object=ClassCancellationCheck)

template_str = "{format_instructions}"

prompt_template = PromptTemplate(
    template=template_str,
    input_variables=["query"],
    partial_variables={"format_instructions": output_parser.get_format_instructions()},
)

model = ChatOpenAI(model="gpt-4o", temperature=0)

query_filter_chain = prompt_template | model | output_parser
```


LangGraph 기술 공유 - 바뀐 pydantic v2 사용법

LangGraph를 사용하면서 langchain-core를 0.3 버전을 사용해야 합니다.
Langchain-core를 사용하면 pydantic v2 버전을 사용해야 합니다.

차이점

| 특징 | PydanticOutputParser | with_structured_output |
|-----------|-------------------------------------|-----------------------------------|
| 출력 생성 방식 | LLM 출력 문자열을 받아서 Pydantic 모델로 파싱 | LLM이 출력 자체를 Pydantic 모델 구조에 맞춰 생성 |
| 에러 발생 가능성 | 출력이 구조화되지 않으면 파싱 실패 가능 | LLM이 정확히 형식에 맞춰 출력하지 않으면 에러 발생 |
| 유연성 | 출력이 Pydantic 형식과 완전히 일치하지 않아도 파싱 가능 | LLM 출력이 Pydantic 형식과 반드시 일치해야 함 |
| 적합한 경우 | 출력이 비교적 유연하고 후처리로 구조화가 필요한 경우 | 출력이 처음부터 완전히 구조화되어야 하는 경우 |

```
prompt = PromptTemplate(
    template=prompts.check_class_cancellation_prompt,
    input_variables=["question"],
)

chain = prompt | self.model.with_structured_output(ClassCancellationCheck)

response = chain.invoke({"question": state["question"]})

if response.relevance_to_class_cancellation == "yes":
    return "generate_class_cancellation"
else:
    return "retrieve_documents"
```

LangGraph 기술 공유 - 바뀐 pydantic v2 사용법

Pydantic을 선언할 때는 **필드명**, **설명** 을 명확하게 선언해야 합니다!

```
class ClassCancellationCheck(BaseModel):  
    """Binary score indicating whether it is related to class cancellation."""  
    relevance_to_class_cancellation: str = Field(  
        description="Binary relevance score: 'yes' if related to class cancellation, 'no' otherwise."  
    )
```

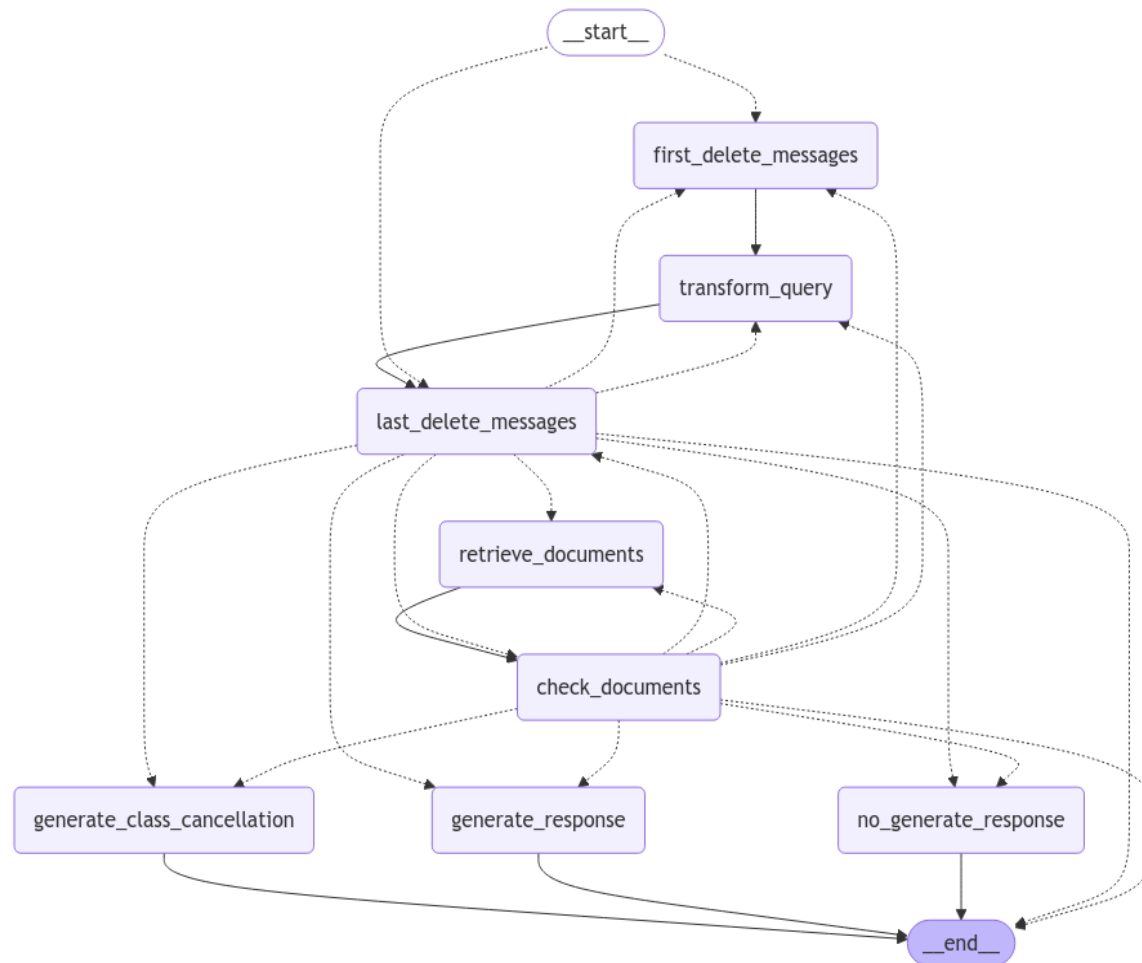
LangGraph 기술 공유 - Literal 사용 그래프 지정

add_conditional_edge를 사용할 때 next 노드를 항상 명시 해줘야합니다!

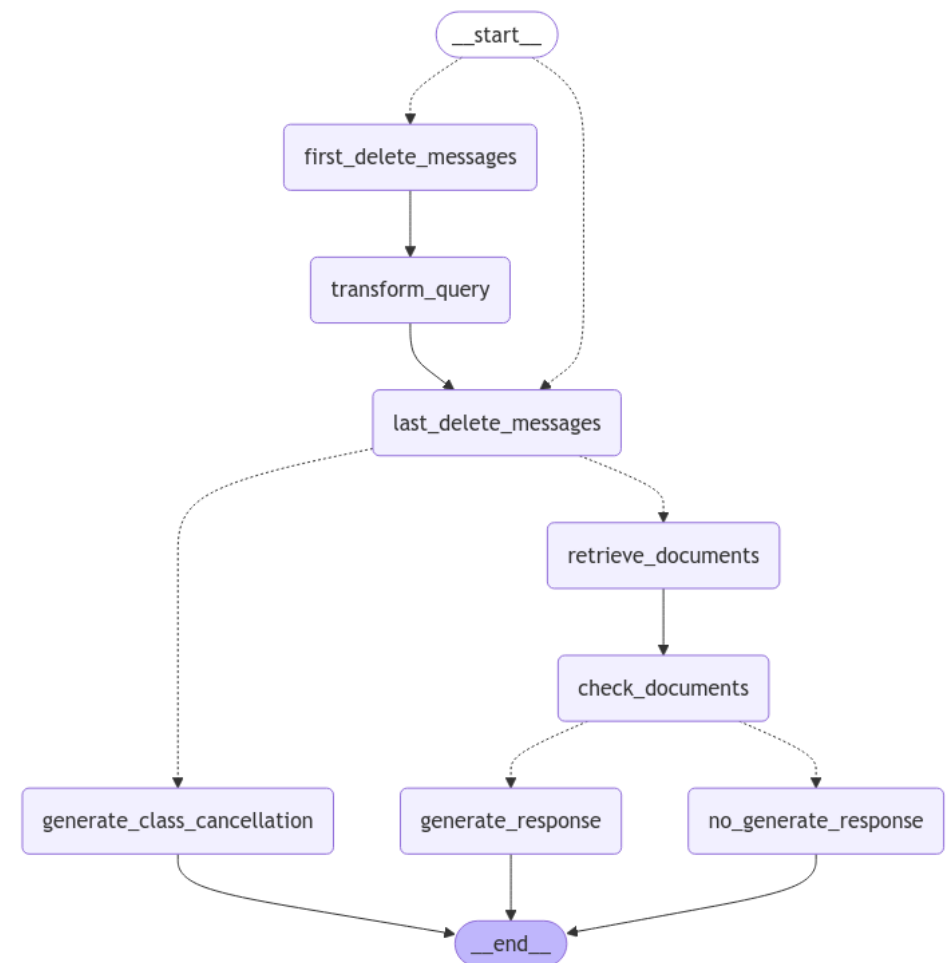
```
def check_class_cancellation(self, state: operation_state) -> Literal["Node_A", "Node_B"]:  
    print("---- this node is check_class_cancellation ----")  
  
    prompt = PromptTemplate(  
        template=prompts.check_class_cancellation_prompt,  
        input_variables=["question"],  
    )  
  
    chain = prompt | self.model.with_structured_output(ClassCancellationCheck)  
    response = chain.invoke({"question": state["question"]})  
  
    if response.relevance_to_class_cancellation == "yes":  
        return "Node_A"  
    else:  
        return "Node_B"
```

LangGraph 기술 공유 - Literal 사용 그래프 지정

지정X



지정O



결과



내일 수업 취소됐어?

불필요한 토큰을 사용하지 않기 때문에 속도 개선과 토큰 절감을 할 수 있습니다!



안녕하세요! 😊

내일은 2024년 11월 29일 금요일입니다. 공휴일 정보에 따르면 내일은 공휴일이 아니므로 수업이 정상적으로 진행됩니다. 📅

수업에 참석해 주세요! ✨



내일 수업 취소하고 싶어



안녕하세요! MBC+ 가이드입니다! 😊

내일 수업을 취소하고 싶으시다면, 우선적으로 운영진과 상담이 필요합니다. 디스코드 내 'MBC+ 과정운영팀'에게 과정 취소 상담과 관련한 내용을 DM으로 보내시면 빠르게 도움을 받으실 수 있습니다.

혹시 디스코드 사용이 어려우시면, 아래의 연락처로 문의해 주세요:

- 문의 전화: 02-730-4519
- 이메일: info@kdt-mbcplus.com

이해되셨나요? 혹시 더 궁금한 부분이 있으면 물어보세요! 😊



LangGraph

human: 내일 수업 취소... ai: 안녕하세요! M...

2024. 11. 28. 오후 11...

⌚ 6.32s



6,824

\$0.0187025



LangGraph

human: 내일 수업 취소... ai: 안녕하세요! 😊 ...

2024. 11. 28. 오후 11...

⌚ 1.78s



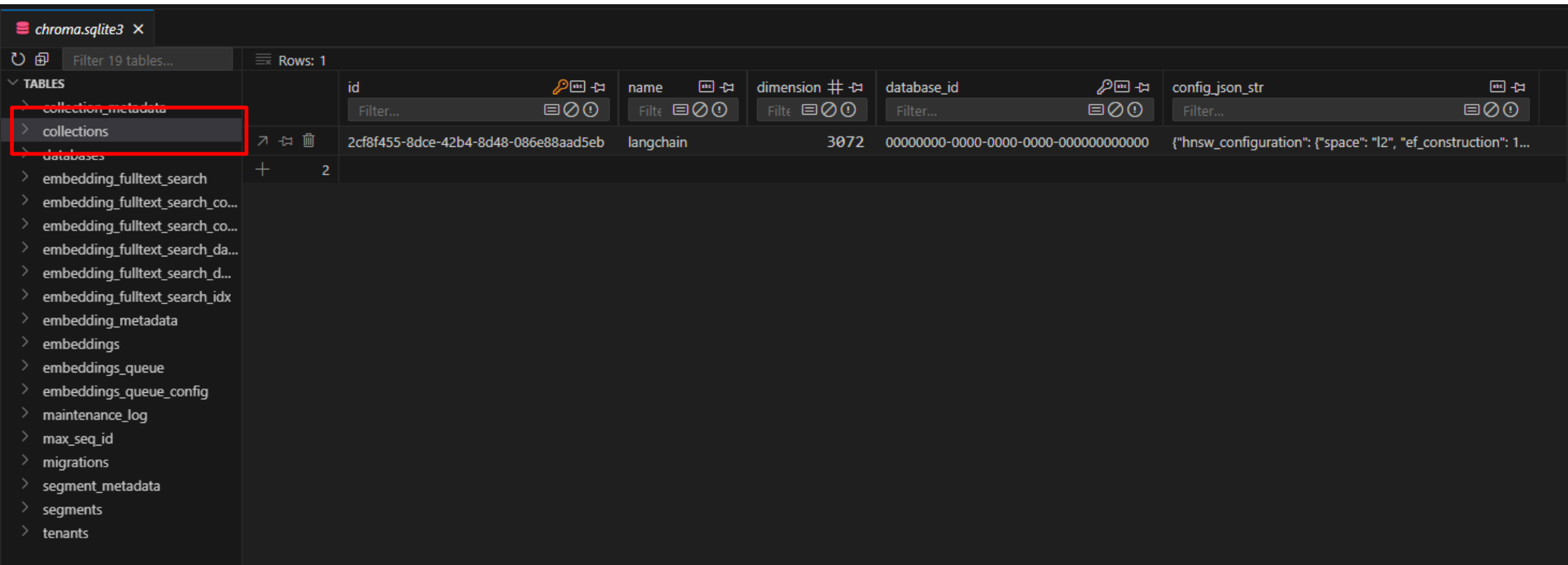
1,407

\$0.00402

기술 공유 - chromaDB 원격 수정

서버가 실행 중일 때 변경된 데이터를 Chroma DB에 반영하려면,
ChromaDB 파일을 삭제하고 다시 생성하는 방식은 사용하면 안 됩니다!

ChromaDB는 table과 유사한 개념인 collection으로 데이터를 그룹화하고 관리하고 있습니다.



The screenshot shows a database interface for 'chroma.sqlite3'. On the left, a sidebar lists various tables under the 'TABLES' section. The 'collections' table is highlighted with a red box. The main area displays the 'collections' table with the following columns: id, name, dimension, database_id, and config_json_str. The table contains one row with the following data:

| id | name | dimension | database_id | config_json_str |
|--------------------------------------|-----------|-----------|--------------------------------------|--|
| 2cf8f455-8dce-42b4-8d48-086e88aad5eb | langchain | 3072 | 00000000-0000-0000-0000-000000000000 | {"hnsw_configuration": {"space": "l2", "ef_construction": 1... |

기술 공유 - chromaDB 원격 수정

Collection 초기화

```
main_db = Chroma(embedding_function=embeddings, persist_directory=main_db_directory)
main_db.reset_collection()
print("현재 DB의 모든 컬렉션이 삭제되었습니다.")

recursive splitted document = load_documents(file_path)
main_db.add_documents(documents=recursive splitted document)
print("temp DB의 문서들이 현재 DB에 추가되었습니다.")
```

Documents 업로드

강의 속 알짜배기 모음

판다스 스튜디오 - 병렬 노드 실행

add_conditional_edge를 사용할 때 next node를 리스트 형태로 반환합니다.

```
from typing import Sequence
```

part5.py

```
def route_datasources_tool_search(state: ToolSearchState) -> Sequence[str]:
```

```
    if set(state['datasources']) == {'search_menu'}:  
        return ['search_menu']
```

```
    elif set(state['datasources']) == {'search_wine'}:  
        return ['search_wine']
```

```
    elif set(state['datasources']) == {'search_web'}:  
        return ['search_web']
```

```
    elif set(state['datasources']) == {'search_menu', 'search_wine'}:  
        return ['search_menu', 'search_wine']
```

```
    elif set(state['datasources']) == {'search_menu', 'search_web'}:  
        return ['search_menu', 'search_web']
```

```
    elif set(state['datasources']) == {'search_wine', 'search_web'}:  
        return ['search_wine', 'search_web']
```

```
    return ['search_web', 'search_menu', 'search_wine']
```

판다스 스튜디오 - create_react_agent 내부 구성

간단한 검색 tool 세팅

```
from langchain_community.tools import TavilySearchResults

# Tool 정의
@tool
def search_web(query: str) -> List[str]:
    """데이터베이스에 존재하지 않는 정보 또는 최신 정보를 인터넷에서 검색합니다."""

    tavily_search = TavilySearchResults(max_results=3)
    docs = tavily_search.invoke(query)

    formatted_docs = "\n\n---\n\n".join(
        [
            f'<Document href="{doc["url"]}" />\n{doc["content"]}\n</Document>'
            for doc in docs
        ]
    )

    if len(docs) > 0:
        return formatted_docs

    return "관련 정보를 찾을 수 없습니다."
```

판다스 스튜디오 - create_react_agent 내부 구성

bind_tool을 사용하면 도구가 필요한 상황에서 바로 답변을 제공하지 않고, 대신 도구 호출을 수행합니다.

```
from langchain_openai import ChatOpenAI

# LLM 모델
llm = ChatOpenAI(model="gpt-4o-mini", streaming=True)

# 도구 목록
tools = [search_web]

# 모델에 도구를 바인딩
llm_with_tools = llm.bind_tools(tools=tools)

# 도구 호출
tool_call = llm_with_tools.invoke([HumanMessage(content=f"LangGraph는 무엇인가요?")])

# 결과 출력
print(tool_call.additional_kwargs)
```

```
{'tool_calls': [{'index': 0, 'id': 'call_veljXtipNE0N007Yq9IKV2H6', 'function': {'arguments': '{"query": "LangGraph"}', 'name': 'search_web', 'type': 'function'}}]}
```

판다스 스튜디오 - create_react_agent 내부 구성

도구 호출 결과를 ToolNode로 정의된 도구를 호출하면 해당 도구의 결과를 바로 얻을 수 있습니다.

```
from langgraph.prebuilt import ToolNode

# 도구 노드 정의
tool_node = ToolNode(tools=tools)

# 도구 호출
tool_call = llm_with_tools.invoke([HumanMessage(content=f"LangGraph는 무엇인가요?")])

# 도구 호출 결과를 메시지로 추가하여 실행
results = tool_node.invoke({"messages": [tool_call]})

# 실행 결과 출력하여 확인
for result in results['messages']:
    print(result.content)
    print()
```

<Document href="https://github.com/langchain-ai/langgraph"/>
 GitHub - langchain-ai/langgraph: Build resilient language agents as graphs. LangGraph Platform is a commercial solution for deploying agentic applications to production, built on the open-source LangGraph framework. Let's take a look at a simple example of an agent that can use a search tool.
 . # Define the tools for the agent to use # This means that after `tools` is called, `agent` node is called next. workflow.add_edge("tools", 'agent') Define entry point and graph edges. First, we need to set the entry point for graph execution - agent node. Normal edge: after the tools are invoked, the graph should always return to the agent to decide what to do next LangGraph adds the input message to the internal state, then passes the state to the entrypoint node, "agent".
 </Document>

<Document href="https://www.langchain.com/langgraph"/>

판다스 스튜디오 - create_react_agent 내부 구성

```
def call_model(state: GraphState):
    system_message = SystemMessage(content=system_prompt)
    messages = [system_message] + state['messages']
    response = llm_with_tools.invoke(messages)
    return {"messages": [response]}
```

1. bind_tool 된 llm을 실행하는 노드

```
def should_continue(state: GraphState):
    last_message = state["messages"][-1]
    # 도구 호출이 있으면 도구 실행 노드로 이동
    if last_message.tool_calls:
        return "execute_tools"
    # 도구 호출이 없으면 답변 생성하고 종료
    return END
```

2. llm이 tool을 호출했는지 판단하는 add_conditional_edge

그래프 구성

```
builder = StateGraph(GraphState)
builder.add_node("call_model", call_model)
builder.add_node("execute_tools", ToolNode(tools))
```

3. tool node init

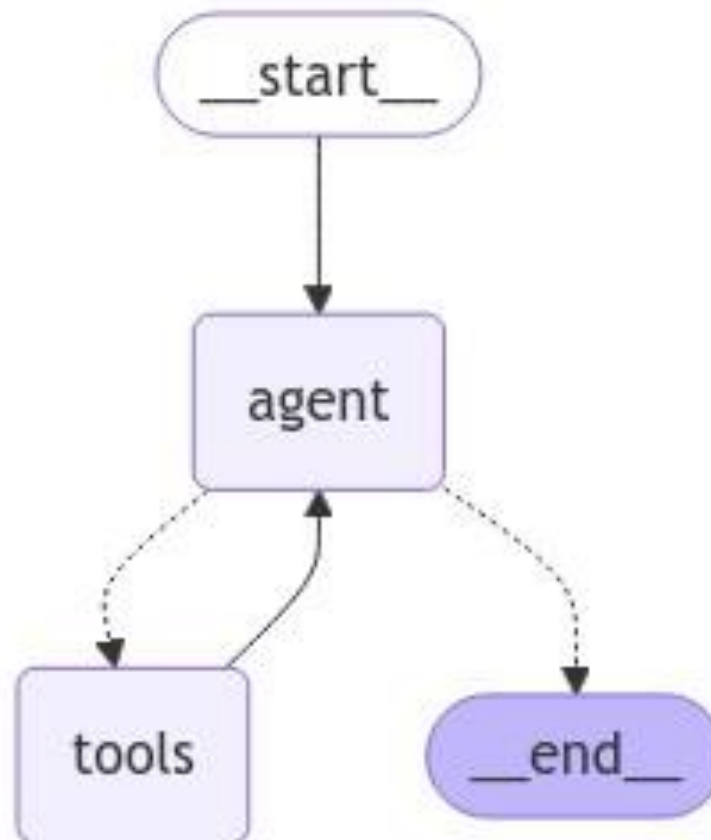
```
builder.add_edge(START, "call_model")
builder.add_conditional_edges(
    "call_model",
    should_continue,
    {
        "execute_tools": "execute_tools",
        END: END
    }
)
builder.add_edge("execute_tools", "call_model")
```

4. tool_node에서 call_model로 연결

판다스 스튜디오 - create_react_agent 내부 구성

```
graph = builder.compile()

# 그래프 출력
display(Image(graph.get_graph().draw_mermaid_png()))
```



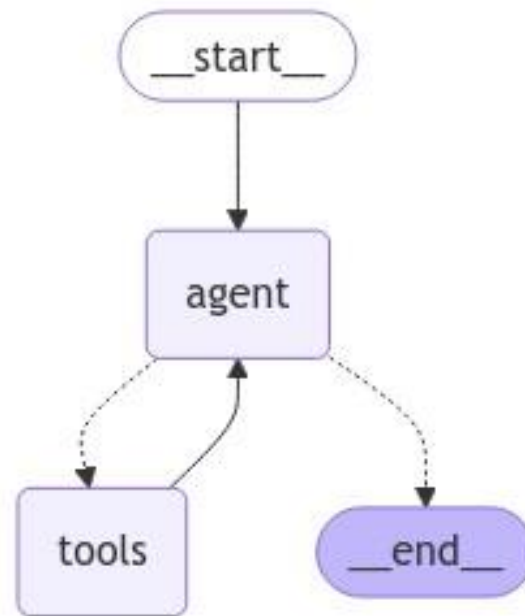
판다스 스튜디오 - create_react_agent 내부 구성

하지만 create_react_agent 를 사용하면 바로 결과를 얻을 수 있습니다!

```
from langgraph.prebuilt import create_react_agent

# 그래프 생성
graph = create_react_agent(
    llm,
    tools=tools,
)

# 그래프 출력
display(Image(graph.get_graph().draw_mermaid_png()))
```



판다스 스튜디오 - create_react_agent 내부 구성

Create_react_agent 의 또 다른 장점 중 하나는 Tool System Prompt를 추가하여 에이전트의 동작 방식을 더욱 세부적으로 정의하고 조정할 수 있다는 점입니다.

```
from langgraph.prebuilt import create_react_agent
from IPython.display import Image, display
from textwrap import dedent

# 시스템 프롬프트
system_prompt = dedent("""
You are an AI assistant designed to answer human questions.
You can use the provided tools to help generate your responses.

Follow these steps to answer questions:
1. Carefully read and understand the question.
2. Use the provided tools to obtain necessary information.
3. Immediately after using a tool, cite the source using the format below.
4. Construct an accurate and helpful answer using the tool outputs and
citations.
5. Provide the final answer when you determine it's complete.
.....

# 그래프 생성
graph = create_react_agent(
    llm,
    tools=tools,
    state_modifier=system_prompt,
)
```


판다스 스튜디오 - 멀티 그래프 연결

간단한 num_list에 "2"를 추가하는 graph 입니다.

```
from langgraph.graph import StateGraph, START, END
from typing import TypedDict, List
from langgraph.graph.message import add_messages
from typing import Annotated

import operator

class MyState(TypedDict):
    num_list: Annotated[List, add_messages]

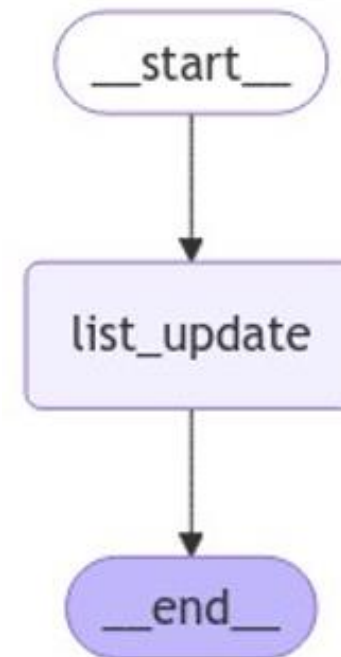
graph_builder = StateGraph(MyState)

def list_update(state: MyState):
    return {"num_list": "2"}

graph_builder.add_node("list_update", list_update)

graph_builder.add_edge(START, "list_update")
graph_builder.add_edge("list_update", END)

graph = graph_builder.compile()
```



판다스 스튜디오 - 멀티 그래프 연결

State를 상속 받을 때 충돌을 주의해야 합니다!

```
class NewState(MyState):
    cnt : int

new_graph_builder = StateGraph(NewState)

def list_sum(state: NewState):
    tmp = 0
    for x in state["num_list"]:
        tmp += int(x.content)
    return {"cnt": tmp}

new_graph_builder.add_node("sub_graph", graph)
new_graph_builder.add_node("list_sum", list_sum)

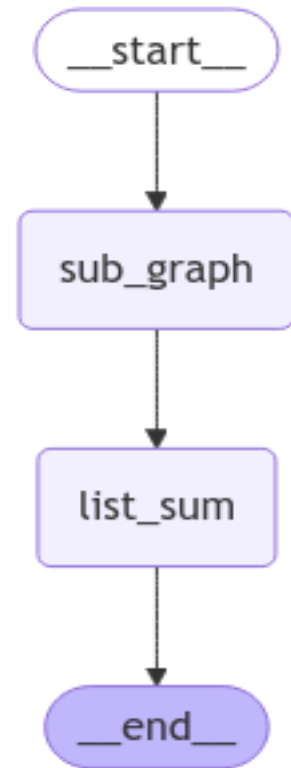
new_graph_builder.add_edge(START, "sub_graph")
new_graph_builder.add_edge("sub_graph", "list_sum")
new_graph_builder.add_edge("list_sum", END)

new_graph = new_graph_builder.compile()

result = new_graph.invoke({"num_list" : "1"})
print(result)
```

state상속

Graph를 노드화



Graph를 사용하면 코드 재사용성이 높아집니다.

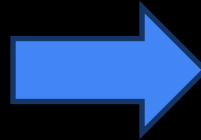
```
{'num_list': [HumanMessage(content='1', additional_kwargs={}, response_metadata={}, id='542ec2a4-b78a-4284-ab25-4a15d70d395b'),
HumanMessage(content='2', additional_kwargs={}, response_metadata={}, id='6fdf9293-31c7-4503-9f06-a26a6160125b')], 'cnt': 3}
```

케인 - tools_condition을 사용한 다른 ToolNode 사용법

```
from langgraph.prebuilt import tools_condition
```

주의할 점! tools 라는 이름은 바꾸면 안 됩니다.

```
builder.add_conditional_edges(
    "call_model",
    should_continue,
    {
        "execute_tools": "execute_tools",
        END: END
    }
)
```



```
builder.add_conditional_edges(
    "call_model",
    tools_condition,
    {
        "tools": "execute_tools",
        END: END,
    },
)
```

```
def tools_condition(
    state: Union[list[AnyMessage], dict[str, Any], BaseModel],
) -> Literal["tools", "__end__"]:
    """Use in the conditional_edge to route to the ToolNode if the last message
    has tool calls. Otherwise, route to the end.

    Args:
        state (Union[list[AnyMessage], dict[str, Any], BaseModel]): The state to check for
            tool calls. Must have a list of messages (MessageGraph) or have the
            "messages" key (StateGraph).

    Returns:
```

```
def call_model(state: GraphState):
    system_message = SystemMessage(content=system_prompt)
    messages = [system_message] + state['messages']
    response = llm_with_tools.invoke(messages)
    return {"messages": [response]}
```

1. bind_tool 된 llm을 실행하는 노드

```
def should_continue(state: GraphState):
    last_message = state["messages"][-1]
    # 도구 호출이 있으면 도구 실행 노드로 이동
    if last_message.tool_calls:
        return "execute_tools"
    # 도구 호출이 없으면 답변 생성하고 종료
    return END
```

2. llm이 tool을 호출했는지 판단하는 add_conditional_edge

그래프 구성

```
builder = StateGraph(GraphState)
builder.add_node("call_model", call_model)
builder.add_node("execute_tools", ToolNode(tools))
```

3. tool node init

```
builder.add_edge(START, "call_model")
builder.add_conditional_edges(
    "call_model",
    should_continue,
    {
        "execute_tools": "execute_tools",
        END: END
    }
)
builder.add_edge("execute_tools", "call_model")
```

4. tool_node에서 call_model로 연결

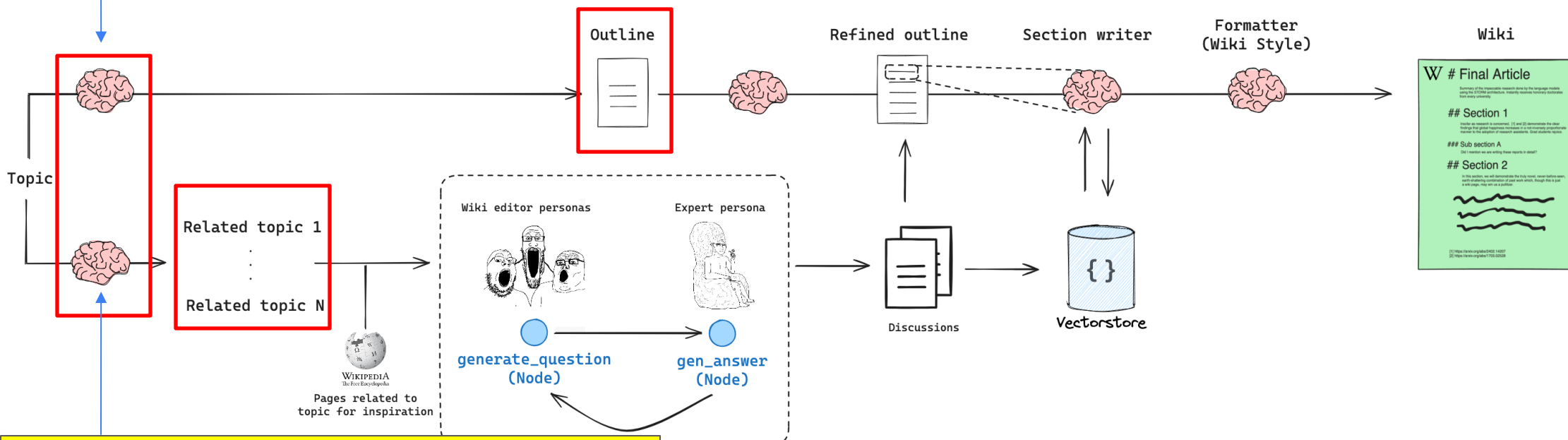
AlSchool – Web Storm Architecture

<https://langchain-ai.github.io/langgraph/tutorials/storm/storm/>

노트북 LM 아키텍처와 유사함!

초기 outline(윤곽)을 생성하는 노드

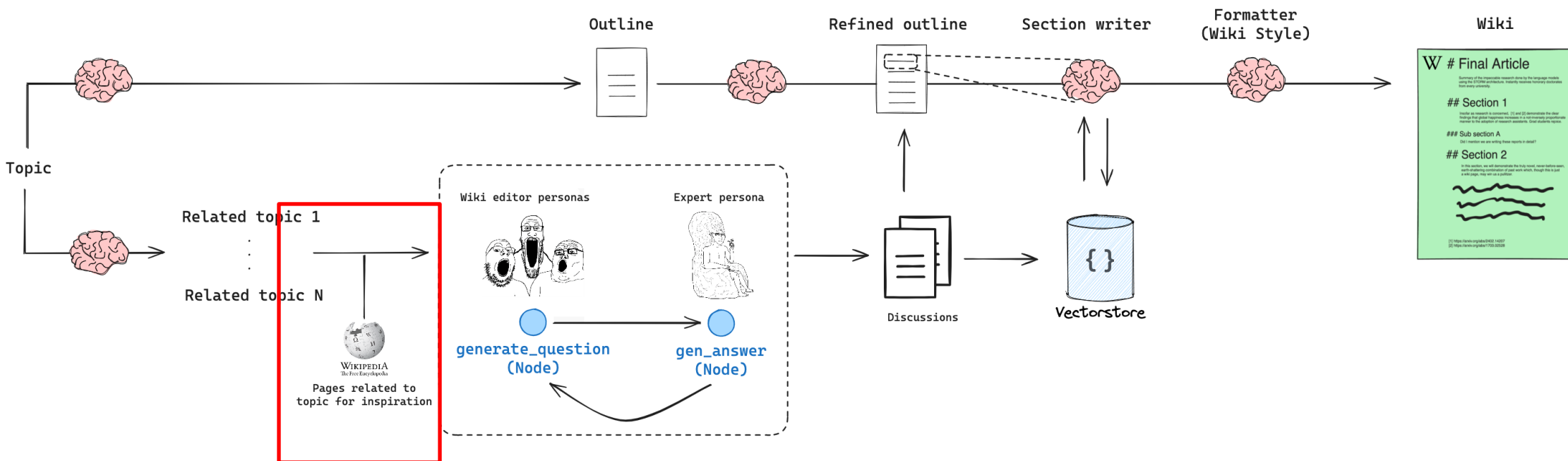
Synthesis of Topic Outlines through Retrieval and Multi-perspective Question Asking



하나의 토픽을 가지고 관련된 토픽을 증강시키는 노드

AlSchool – Web Storm Architecture

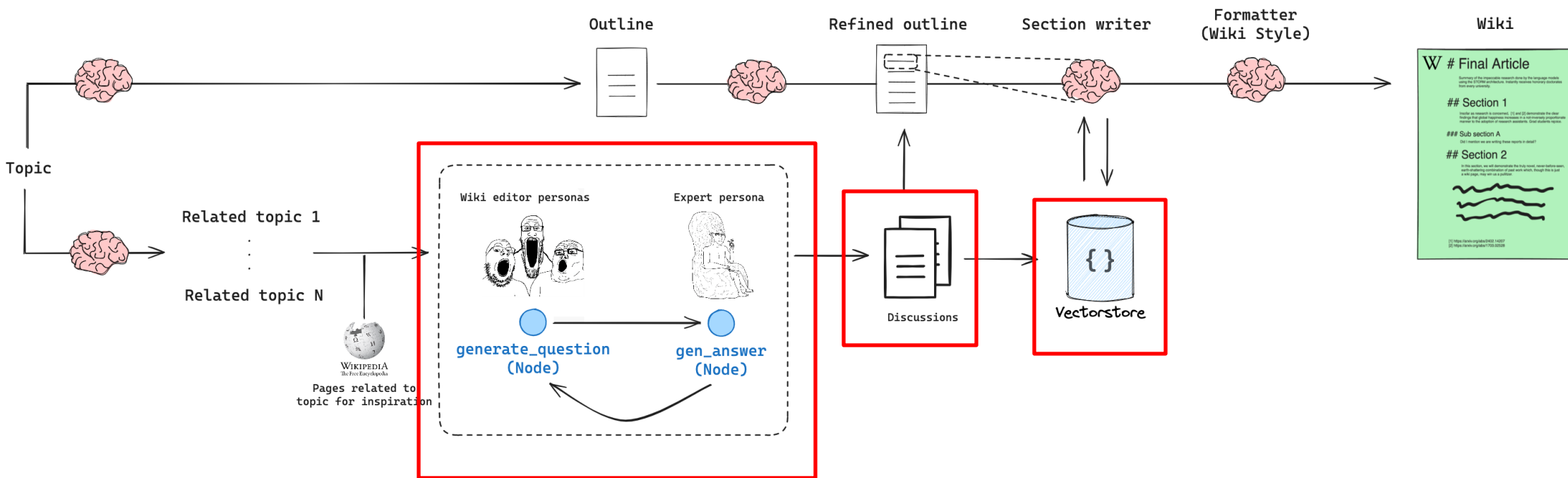
Synthesis of Topic Outlines through Retrieval and Multi-perspective Question Asking



증강된 토픽을 활용해서 관련성이 높은 정보를 가져옵니다.

AlSchool – Web Storm Architecture

Synthesis of Topic Outlines through Retrieval and Multi-perspective Question Asking



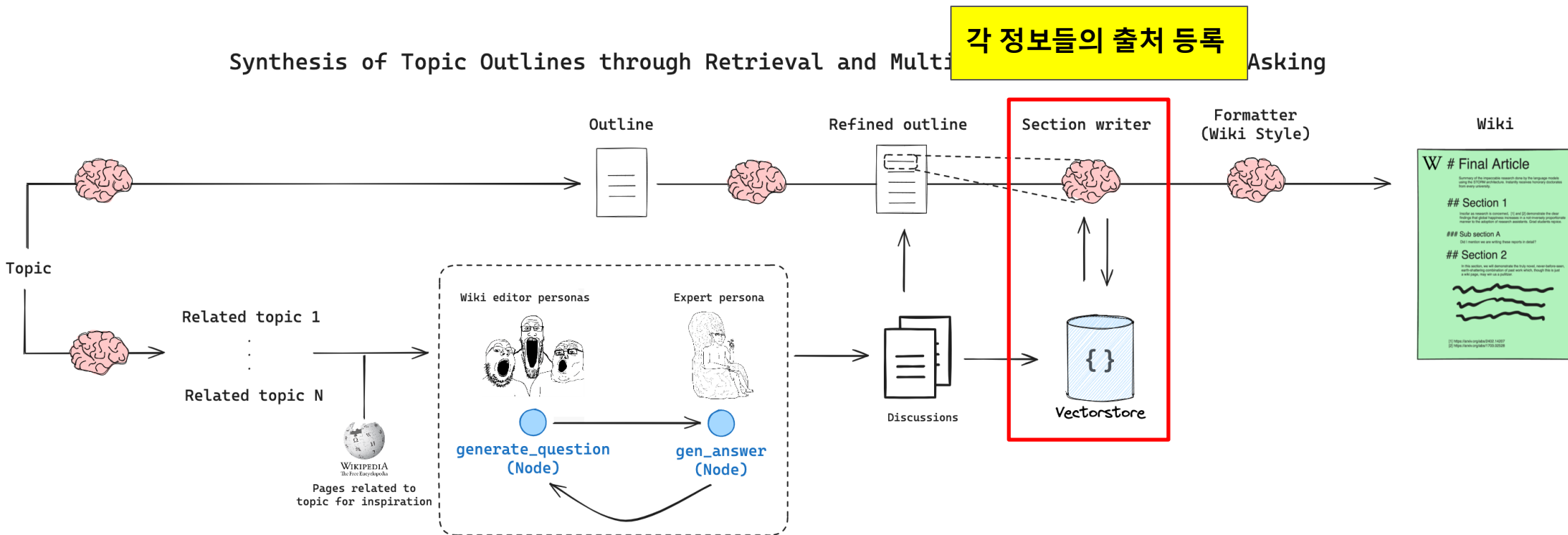
질문 노드와 답변 노드는 검색된 정보를 기반으로 상호작용하며 추가 정보를 생성합니다.

생성된 정보는 문서 형태로 변환되어 상태(state)에 업데이트합니다.

RAG를 하기 위해서 VectorDB에도 저장합니다.



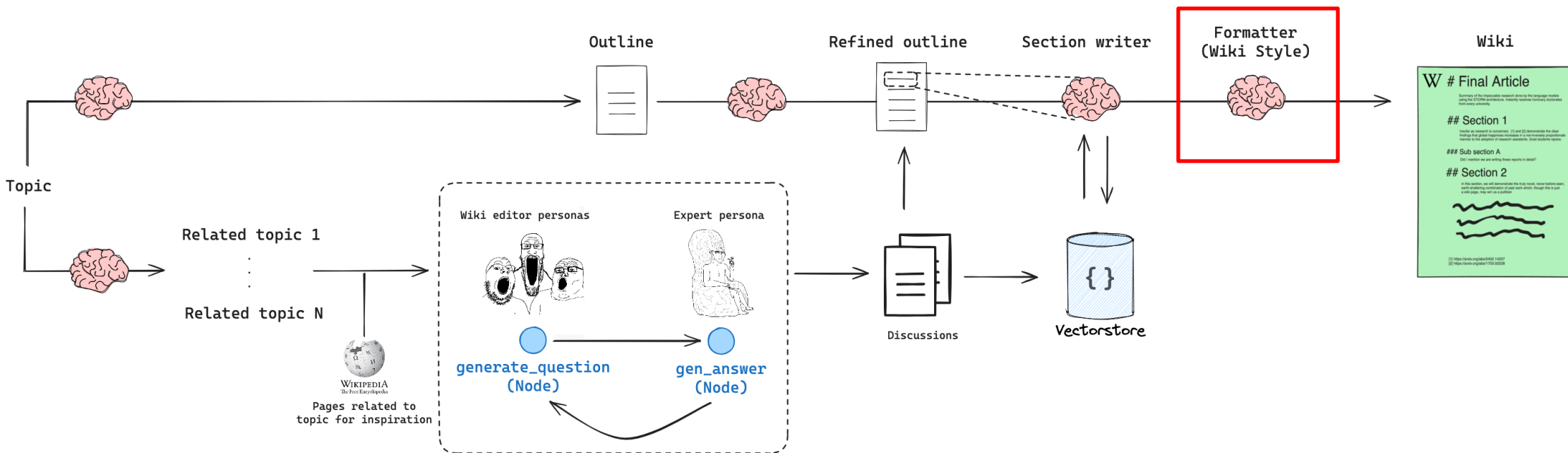
AlSchool – Web Storm Architecture



AlSchool – Web Storm Architecture

Synthesis of Topic Outlines through Retrieval and Multi

증강된 모든 정보를 가지고 위키피디아 글 작성



감사합니다!