

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

## 8. SelfQueryRetriever

### 1) 셀프 쿼리

- **SelfQueryRetriever** = 자체적으로 질문을 생성하고 해결할 수 있는 기능을 갖춘 검색 도구
  - 사용자가 제공한 자연어 질의 → **query-constructing LLM chain** 사용 → 구조화된 질의 생성 → 기본 벡터 데이터 저장소 (VectorStore)에 적용 → 검색 수행
  - 저장된 문서의 내용에서 의미적으로 비교하는 것을 넘어 **사용자의 질의에서 문서의 메타데이터에 대한 필터를 추출 → 필터를 실행해 관련도** **니 문서 찾을 수 있음**
- 참고: [LangChain 이 지원하는 셀프 쿼리 검색기 목록](#)

### 2) 설정

```
# API 키를 환경변수로 관리하기 위한 설정 파일
from dotenv import load_dotenv
```

```
# API 키 정보 로드
load_dotenv()                                     # True
```

```
from langsmith import Client
from langsmith import traceable

import os

# 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_API_KEY') else "설정되지 않음" # API 키 값은 직접 출력하지 않음

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_API_KEY') and langchain_project:
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='{langchain_tracing_v2}')
```

- 셀 출력

```
--- LangSmith 환경 변수 확인 ---
✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='true')
✅ LangSmith 프로젝트: 'LangChain-practice'
✅ LangSmith API Key: 설정됨
-> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.
```

### 3) 샘플 데이터 생성



- 화장품 상품의 설명과 메타 데이터를 기반으로 유사도 검색이 가능한 벡터 저장소 구축하기

```
from langchain.embeddings import HuggingFaceEmbeddings

# 업그레이드된 임베딩 모델 사용하기: 다국어 고성능 모델 (1024차원)
multilingual_embeddings = HuggingFaceEmbeddings(
    model_name="sentence-transformers/paraphrase-multilingual-mpnet-base-v2", # 1024차원
    model_kwargs={'device': 'cpu'},
    encode_kwargs={'normalize_embeddings': True}
) # 9.7s
```

```
from langchain_chroma import Chroma
from langchain_core.documents import Document
from langchain_huggingface import HuggingFaceEmbeddings

# 화장품 상품의 설명과 메타데이터 생성
docs = [
    Document(
        page_content="수분 가득한 히알루론산 세럼으로 피부 속 깊은 곳까지 수분을 공급합니다.",
        metadata={"year": 2024, "category": "스킨케어", "user_rating": 4.7},
    ),
    Document(
        page_content="24시간 지속되는 매트한 피니시의 파운데이션, 모공을 커버하고 자연스러운 피부 표현이 가능합니다.",
        metadata={"year": 2023, "category": "메이크업", "user_rating": 4.5},
    ),
    Document(
        page_content="식물성 성분으로 만든 저자극 클렌징 오일, 메이크업과 노폐물을 부드럽게 제거합니다.",
        metadata={"year": 2023, "category": "클렌징", "user_rating": 4.8},
    ),
    Document(
        page_content="비타민 C 함유 브라이팅 크림, 칙칙한 피부톤을 환하게 밝혀줍니다.",
        metadata={"year": 2023, "category": "스킨케어", "user_rating": 4.6},
    ),
    Document(
        page_content="롱래스팅 립스틱, 선명한 발색과 촉촉한 사용감으로 하루종일 편안하게 사용 가능합니다.",
        metadata={"year": 2024, "category": "메이크업", "user_rating": 4.4},
    ),
    Document(
        page_content="자외선 차단 기능이 있는 톤업 선크림, SPF50+/PA++++ 높은 자외선 차단 지수로 피부를 보호합니다.",
        metadata={"year": 2024, "category": "선크어", "user_rating": 4.9},
    ),
]

# 벡터 저장소 생성
vectorstore = Chroma.from_documents(
    docs, multilingual_embeddings) # 24.3s
```

#### 4) SelfQueryRetriever

- **retriever** 인스턴스화: 문서가 지원하는 **메타데이터 필드**, 문서 내용에 대한 **간단한 설명**을 미리 제공 해야 함
- **AttributeInfo** 클래스 → 화장품 메타데이터 필드에 대한 정보 정의하기
  - **category** (카테고리)
    - 문자열 타입, 화장품의 카테고리
    - [스킨케어, 메이크업, 클렌징, 선크어] 중 하나의 값을 가짐
  - **year** (연도)
    - 정수 타입
    - 화장품이 출시된 연도
  - **user\_rating** (사용자 평점)
    - 실수 타입
    - 1~5 범위의 사용자 평점

```
from langchain.chains.query_constructor.base import AttributeInfo

# 메타데이터 필드 정보 생성
metadata_field_info = [
    AttributeInfo(
        name="category",
        description="The category of the cosmetic product. One of ['스킨케어', '메이크업', '클렌징', '선크어']",
        type="string",
    ),
]
```

```

        AttributeInfo(
            name="year",
            description="The year the cosmetic product was released",
            type="integer",
        ),
        AttributeInfo(
            name="user_rating",
            description="A user rating for the cosmetic product, ranging from 1 to 5",
            type="float",
        ),
    ],
    # 0.1s

```

- **SelfQueryRetriever.from\_llm()** 메서드 사용 → **retriever** 객체 생성
  - **llm**: 언어 모델
  - **vectorstore**: 벡터 저장소
  - **document\_contents**: 문서들의 내용 설명
  - **metadata\_field\_info**: 메타데이터 필드 정보

```

from langchain.retrievers.self_query.base import SelfQueryRetriever
from langchain_ollama import OllamaLLM

# LLM 정의하기
llm_llama = OllamaLLM(model="exaone3.5:7.8b", temperature=0)

# SelfQueryRetriever 생성
retriever = SelfQueryRetriever.from_llm(
    llm=llm_llama,
    vectorstore=vectorstore,
    document_contents="Brief summary of a cosmetic product",
    metadata_field_info=metadata_field_info,
    # verbose = True,
    # search_kwargs = {"k":1}
)
# 검색 결과에서 중복 답변 허용 X
# 0.8s

```

#### 4) Query 테스트

- 필터를 걸 수 있는 질의 입력 → 검색 수행하기

```

# Self-query 검색_1

retriever.invoke("평점이 4.8 이상인 제품을 추천해주세요")

```

- 검색\_1 (16.2s)

```
[Document(id='43171779-45b4-4fdb-bc35-f1ab632fa5f6', metadata={'year': 2024, 'category': '선크어', 'user_rating': 4.9}, text='선크어 제품 설명')]
```

```

# Self-query 검색_1

retriever.invoke("평점이 4.8이거나 그 이상인 제품을 추천해주세요")

```

- 검색\_1-ver.2 (14.8s)

```
[Document(id='43171779-45b4-4fdb-bc35-f1ab632fa5f6', metadata={'year': 2024, 'user_rating': 4.9, 'category': '선크어', 'text': '선크어 제품 설명'}, score=0.95), Document(id='d82f2168-06c5-46f2-a934-d174d09da1c1', metadata={'year': 2023, 'category': '클렌징', 'user_rating': 4.8}, text='클렌징 제품 설명')]
```

```

# Self-query 검색_2

retriever.invoke("2023년에 출시된 상품을 추천해주세요")

```

- Self-query 검색\_2 (9.8s)

```
[Document(id='f84e3ca8-d48d-4c77-9f6e-fc922acc4554', metadata={'year': 2023, 'user_rating': 4.6, 'category': '스킨케어', 'text': '스킨케어 제품 설명'}, score=0.95), Document(id='d82f2168-06c5-46f2-a934-d174d09da1c1', metadata={'year': 2023, 'user_rating': 4.8, 'category': '클렌징', 'text': '클렌징 제품 설명'}, score=0.9), Document(id='969fe0e9-09e2-4a87-915e-221b4a96f448', metadata={'year': 2023, 'category': '메이크업', 'user_rating': 4.7}, text='메이크업 제품 설명')]
```

```
# Self-query 검색_3
```

```
retriever.invoke("카테고리에서 '선헤어'와 일치하는 상품만 찾아주세요")
```

- Self-query 검색\_3 (17.4s)

```
[Document(id='f84e3ca8-d48d-4c77-9f6e-fc922acc4554', metadata={'user_rating': 4.6, 'year': 2023, 'category': '스킨케어'}, score=0.95),
Document(id='9e665b64-41ac-499b-baa5-8b491e30afb', metadata={'user_rating': 4.7, 'year': 2024, 'category': '스킨케어'}, score=0.95)]
```

```
retriever.invoke("카테고리별로 상품을 정리해주세요")
```

- 카테고리별 검색해보기 (50.0s)

```
[Document(id='43171779-45b4-4fdb-bc35-f1ab632fa5f6', metadata={'category': '선헤어', 'user_rating': 4.9, 'year': 2023}, score=0.95),
Document(id='d82f2168-06c5-46f2-a934-d174d09da1c1', metadata={'year': 2023, 'user_rating': 4.8, 'category': '클렌징폼'}, score=0.95),
Document(id='dd779b39-52d0-40c6-b4ca-02048816a2f0', metadata={'year': 2024, 'user_rating': 4.4, 'category': '메이크업'}, score=0.95),
Document(id='9e665b64-41ac-499b-baa5-8b491e30afb', metadata={'year': 2024, 'user_rating': 4.7, 'category': '스킨케어'}, score=0.95)]
```

```
retriever.invoke("카테고리에서 '선헤어'로 분류되는 상품을 출력해주세요")
```

- 카테고리에서 '선헤어'로 분류되는 상품 출력해보기 → 실패 (15.6s)

```
[Document(id='f84e3ca8-d48d-4c77-9f6e-fc922acc4554', metadata={'category': '스킨케어', 'year': 2023, 'user_rating': 4.6}, score=0.95),
Document(id='9e665b64-41ac-499b-baa5-8b491e30afb', metadata={'user_rating': 4.7, 'category': '스킨케어', 'year': 2024}, score=0.95)]
```

- input 으로 검색 → 모두 실패

```
# try_1
retriever.invoke(input="선헤어")

# try_2
retriever.invoke(input="선헤어", metadata={'category': '선헤어'}, search_type='filter')
```

```
-----
UnexpectedCharacters                               Traceback (most recent call last)
File ~/pyenv/versions/lc_env/lib/python3.13/site-packages/lark/lexer.py:665, in ContextualLexer.lex(self, lexer_state, parser_state)
    664         lexer = self.lexers[parser_state.position]
--> 665         yield lexer.next_token(lexer_state, parser_state)
    666 except EOFError:

File ~/pyenv/versions/lc_env/lib/python3.13/site-packages/lark/lexer.py:598, in BasicLexer.next_token(self, lexer_state, parser_state)
    597         allowed = {"<END-OF-FILE>"}
--> 598         raise UnexpectedCharacters(lexer_state.text, line_ctr.char_pos, line_ctr.line, line_ctr.column,
    599                                   allowed=allowed, token_history=lexer_state.last_token and [lexer_state.last_token],
    600                                   state=parser_state, terminals_by_name=self.terminals_by_name)
    602 value, type_ = res
```

UnexpectedCharacters: No terminal matches ',' in the current parser context, at line 1 col 12

```
eq(category, '\uc2a4\ud0a8\ucf00\uc5b4')
      ^
```

Expected one of:  
\* LPAR

Previous tokens: Token('CNAME', 'category')

During handling of the above exception, another exception occurred:

```
UnexpectedToken                                     Traceback (most recent call last)
File ~/pyenv/versions/lc_env/lib/python3.13/site-packages/langchain/chains/query_constructor/base.py:59, in StructuredQueryConstructor._construct_query
    58 else:
```

```

----> 59     parsed["filter"] = self.ast_parse(parsed["filter"])
        60 if not parsed.get("limit"):

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/lark/lark.py:655, in Lark.parse(self, text, start, on_
    638 """Parse the given text, according to the options provided.
    639
    640 Parameters:
(...)    653
    654 """
--> 655 return self.parser.parse(text, start=start, on_error=on_error)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/lark/parser_frontends.py:104, in ParsingFrontend.parse
    103 stream = self._make_lexer_thread(text)
--> 104 return self.parser.parse(stream, chosen_start, **kw)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/lark/parsers/lalr_parser.py:42, in LALR_Parser.parse(s
    41 try:
----> 42     return self.parser.parse(lexer, start)
    43 except UnexpectedInput as e:

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/lark/parsers/lalr_parser.py:88, in _Parser.parse(self,
    87     return InteractiveParser(self, parser_state, parser_state.lexer)
----> 88 return self.parse_from_state(parser_state)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/lark/parsers/lalr_parser.py:111, in _Parser.parse_from
    110     pass
--> 111     raise e
    112 except Exception as e:

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/lark/parsers/lalr_parser.py:100, in _Parser.parse_from
    99 token = last_token
--> 100 for token in state.lexer.lex(state):
    101     assert token is not None

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/lark/lexer.py:674, in ContextualLexer.lex(self, lexer_
    673     token = self.root_lexer.next_token(lexer_state, parser_state)
--> 674     raise UnexpectedToken(token, e.allowed, state=parser_state, token_history=[last_token], terminals_by_
    675 except UnexpectedCharacters:

UnexpectedToken: Unexpected token Token('COMMA', ',') at line 1, column 12.
Expected one of:
    * LPAR
Previous tokens: [Token('CNAME', 'category')]

```

The above exception was the direct cause of the following exception:

```

OutputParserException                                Traceback (most recent call last)
Cell In[20], line 1
----> 1 retriever.invoke(input="선택어")

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/retrievers.py:263, in BaseRetriever.inv
    261 kwargs_ = kwargs if self._expects_other_args else {}
    262 if self._new_arg_supported:
--> 263     result = self._get_relevant_documents(
    264         input, run_manager=run_manager, **kwargs_
    265     )
    266 else:
    267     result = self._get_relevant_documents(input, **kwargs_)

File ~/.pyenv/versions/lc_env/lib/python3.13/site-packages/langchain/retrievers/self_query/base.py:318, in SelfQu
    304 def _get_relevant_documents(
    305     self,
    306     query: str,
    307     *,
    308     run_manager: CallbackManagerForRetrieverRun,
    309 ) -> list[Document]:
    310     """Get documents relevant for a query.
    311
    312     Args:
    313         List of relevant documents
    314
    315     """
    316
    317

```

```

--> 318     structured_query = self.query_constructor.invoke(
319         {"query": query},
320         config={"callbacks": run_manager.get_child()},
321     )
322     if self.verbose:
323         logger.info("Generated Query: %s", structured_query)

```

File ~/.pyenv/versions/lc\_env/lib/python3.13/site-packages/langchain\_core/runnables/base.py:5710, in RunnableBinding.invoke

```

5703 @override
5704 def invoke(
5705     self,
5706     (...) 5708     **kwargs: Optional[Any],
5709 ) -> Output:
-> 5710     return self.bound.invoke(
5711         input,
5712         self._merge_configs(config),
5713         **{**self.kwargs, **kwargs},
5714     )

```

File ~/.pyenv/versions/lc\_env/lib/python3.13/site-packages/langchain\_core/runnables/base.py:3245, in RunnableSequence.\_run

```

3243         input_ = context.run(step.invoke, input_, config, **kwargs)
3244     else:
-> 3245         input_ = context.run(step.invoke, input_, config)
3246 # finish the root run
3247 except BaseException as e:

```

File ~/.pyenv/versions/lc\_env/lib/python3.13/site-packages/langchain\_core/output\_parsers/base.py:208, in BaseOutputParser.parse\_result

```

199 if isinstance(input, BaseMessage):
200     return self._call_with_config(
201         lambda inner_input: self.parse_result(
202             [ChatGeneration(message=inner_input)]
203         ),
204         input,
205         config,
206         run_type="parser",
207     )
--> 208 return self._call_with_config(
209     lambda inner_input: self.parse_result([Generation(text=inner_input)]),
210     input,
211     config,
212     run_type="parser",
213 )

```

File ~/.pyenv/versions/lc\_env/lib/python3.13/site-packages/langchain\_core/runnables/base.py:2089, in RunnableCallable.\_run

```

2085     child_config = patch_config(config, callbacks=run_manager.get_child())
2086     with set_config_context(child_config) as context:
2087         output = cast(
2088             "Output",
-> 2089             context.run(
2090                 call_func_with_variable_args, # type: ignore[arg-type]
2091                 func,
2092                 input_,
2093                 config,
2094                 run_manager,
2095                 **kwargs,
2096             ),
2097         )
2098 except BaseException as e:
2099     run_manager.on_chain_error(e)

```

File ~/.pyenv/versions/lc\_env/lib/python3.13/site-packages/langchain\_core/runnables/config.py:430, in call\_func\_with\_variable\_args

```

428 if run_manager is not None and accepts_run_manager(func):
429     kwargs["run_manager"] = run_manager
--> 430 return func(input, **kwargs)

```

File ~/.pyenv/versions/lc\_env/lib/python3.13/site-packages/langchain\_core/output\_parsers/base.py:209, in BaseOutputParser.parse\_result

```

199 if isinstance(input, BaseMessage):
200     return self._call_with_config(
201         lambda inner_input: self.parse_result(
202             [ChatGeneration(message=inner_input)]
203         ),
204         input,
205         config,
206         run_type="parser",
207     )
--> 208 return self._call_with_config(
209     lambda inner_input: self.parse_result([Generation(text=inner_input)]),
210     input,
211     config,
212     run_type="parser",
213 )

```

```

211     config,
212     run_type="parser",
213 )

```

```

File ~/pyenv/versions/lc_env/lib/python3.13/site-packages/langchain_core/output_parsers/base.py:254, in BaseOutputParser.parse(self, result)
238 @override
239 def parse_result(self, result: list[Generation], *, partial: bool = False) -> T:
240     """Parse a list of candidate model Generations into a specific format.
241
242     The return value is parsed from only the first Generation in the result, which
243     is assumed to be the most relevant.
244
245     """
246     return self.parse(result[0].text)

```

```

File ~/pyenv/versions/lc_env/lib/python3.13/site-packages/langchain/chains/query_constructor/base.py:67, in StructuredQueryConstructor._construct_query(self, query, filter)
65 except Exception as e:
66     msg = f"Parsing text\n{text}\n raised following error:\n{e}"
67     raise OutputParserException(msg) from e

```

```

OutputParserException: Parsing text
```json
{
  "query": "선크어",
  "filter": "eq(category, '\\uc2a4\\ud0a8\\ucf00\\uc5b4')"
}

```

#### Explanation:

- **Query:** "선크어" directly translates to "sun care" in English, which aligns with one of the category descriptions provided ('자외선 차단').
- **Filter:** Since the query focuses on identifying products within the 'sun care' category, the filter checks if the `category` attribute matches the Korean term for 'sun care'. Note that exact matching might require localization adjustments based on how categories are stored in the actual data source. Here, `\\uc2a4\\ud0a8\\ucf00\\uc5b4` represents 'sun care' in Unicode Korean characters as per the provided attribute descriptions. If exact Unicode matching isn't supported, a more straightforward Korean text match might be necessary depending on data storage conventions. raised following error: Unexpected token Token('COMMA', ',') at line 1, column 12. Expected one of:
  - LPAR Previous tokens: [Token('CNAME', 'category')]

For troubleshooting, visit: [https://python.langchain.com/docs/troubleshooting/errors/OUTPUT\\_PARSING\\_FAILURE](https://python.langchain.com/docs/troubleshooting/errors/OUTPUT_PARSING_FAILURE) ``

```
# Self-query 검색_4
```

```
retriever.invoke("카테고리가 메이크업과 일치하는 상품만 찾아주세요")
```

- 카테고리에서 메이크업으로는 검색이 가능 ( 50.3s )

```

[Document(id='9e665b64-41ac-499b-baa5-8b491e30afbf', metadata={'year': 2024, 'user_rating': 4.7, 'category': '스킨케어'}, text='수분 가득한 히알루론산 세럼으로 피부 속 깊은 곳까지 수분을 공급합니다.'),
 Document(id='f84e3ca8-d48d-4c77-9f6e-fc922acc4554', metadata={'year': 2023, 'user_rating': 4.6, 'category': '스킨케어'}, text='24시간 지속되는 매트한 피니시의 파운데이션, 모공을 커버하고 자연스러운 피부 표현이 가능합니다.'),
 Document(id='a1b2c3d4-e5f6-789a-bcde-fghijklmnopqr', metadata={'year': 2024, 'user_rating': 4.5, 'category': '메이크업'}, text='식물성 성분으로 만든 저자극 클렌징 오일, 메이크업과 노폐물을 부드럽게 제거합니다.'),
 Document(id='pqrstuvwxyz1234567890', metadata={'year': 2023, 'user_rating': 4.8, 'category': '스킨케어'}, text='수분 가득한 히알루론산 세럼으로 피부 속 깊은 곳까지 수분을 공급합니다.')
]

```

## 5) 한글 카테고리 이름을 Unicode로 변환해서 필터 생성함으로써 오류 발생

- 영어로 카테고리 변경해서 다시 시도

```

# 숫자 코드로 매핑하기
from langchain_core.documents import Document

# 카테고리를 영어로 변경!
docs = [
    Document(
        page_content="수분 가득한 히알루론산 세럼으로 피부 속 깊은 곳까지 수분을 공급합니다.",
        metadata={"year": 2024, "category": "skincare", "user_rating": 4.7}, # 영어!
    ),
    Document(
        page_content="24시간 지속되는 매트한 피니시의 파운데이션, 모공을 커버하고 자연스러운 피부 표현이 가능합니다.",
        metadata={"year": 2023, "category": "makeup", "user_rating": 4.5},
    ),
    Document(
        page_content="식물성 성분으로 만든 저자극 클렌징 오일, 메이크업과 노폐물을 부드럽게 제거합니다.",
        metadata={"year": 2024, "category": "skincare", "user_rating": 4.8},
    ),
    Document(
        page_content="수분 가득한 히알루론산 세럼으로 피부 속 깊은 곳까지 수분을 공급합니다.",
        metadata={"year": 2023, "category": "skincare", "user_rating": 4.6},
    )
]

```

```

        metadata={"year": 2023, "category": "cleansing", "user_rating": 4.8},
    ),
    Document(
        page_content="비타민 C 함유 브라이팅 크림, 칙칙한 피부톤을 환하게 밝혀줍니다.",
        metadata={"year": 2023, "category": "skincare", "user_rating": 4.6},
    ),
    Document(
        page_content="롱래스팅 립스틱, 선명한 발색과 촉촉한 사용감으로 하루종일 편안하게 사용 가능합니다.",
        metadata={"year": 2024, "category": "makeup", "user_rating": 4.4},
    ),
    Document(
        page_content="자외선 차단 기능이 있는 톤업 선크림, SPF50+/PA++++ 높은 자외선 차단 지수로 피부를 보호합니다.",
        metadata={"year": 2024, "category": "suncare", "user_rating": 4.9},
    ),
]

```

```

# 메타데이터 정의
metadata_field_info = [
    AttributeInfo(
        name="category",
        description="The category code: 1=스킨케어, 2=메이크업, 3=클렌징, 4=선크어",
        type="integer", # 정수 타입!
    ),
    AttributeInfo(
        name="year",
        description="The year the product was released",
        type="integer",
    ),
    AttributeInfo(
        name="user_rating",
        description="User rating from 1 to 5",
        type="float",
    ),
]

```

```

# 메타데이터 설명도 영어로
metadata_field_info = [
    AttributeInfo(
        name="category",
        description="The category of the cosmetic product. One of ['skincare', 'makeup', 'cleansing', 'suncare']",
        type="string",
    ),
    AttributeInfo(
        name="year",
        description="The year the cosmetic product was released",
        type="integer",
    ),
    AttributeInfo(
        name="user_rating",
        description="A user rating for the cosmetic product, ranging from 1 to 5",
        type="float",
    ),
]

```

```

# 벡터스토어 재생성
vectorstore2 = Chroma.from_documents(docs, multilingual_embeddings)

```

```

from langchain.retrievers.self_query.base import SelfQueryRetriever
from langchain_ollama import OllamaLLM
from langchain_community.vectorstores import Chroma
from langchain_community.embeddings import OllamaEmbeddings

```

```

# LLM 정의하기
llm_llama = OllamaLLM(model="llama3.1:8b", temperature=0)

```

```

# SelfQueryRetriever 재생성
retriever = SelfQueryRetriever.from_llm(
    llm=llm_llama,
    vectorstore=vectorstore2,
    document_contents="Brief summary of a cosmetic product",
    metadata_field_info=metadata_field_info,
    verbose=True, # 디버깅용
)

```

```

# 테스트 (한글 쿼리 가능 여부 확인하기)
results = retriever.invoke("선크림 제품 찾아줘")
print(f"✅ 검색 성공! {len(results)}개 찾음") # 14.9s


```

```


retriever.invoke("선크림 제품 찾아줘")


```




-  검색 성공! 1개 찾음 (8.9s)

```
[Document(id='aa61a1b7-357a-4144-824d-4a065bba2e0c', metadata={'category': '선풍기', 'user_rating': 4.9, 'year': 2024}, page_content='선풍기...')]
```

-  검색 성공! 0개 찾음 (14.0s)

```
# 테스트2
results = retriever.invoke("선풍기 제품 찾아줘")
print(f"
```

```
retriever.invoke("선풍기 제품 찾아줘")
```

-  검색 성공! 4개 찾음 (11.2s)

```
[Document(metadata={'year': 2024, 'category': '선풍기', 'user_rating': 4.9}, page_content='선풍기...'),
 Document(metadata={'category': '선풍기', 'user_rating': 4.9, 'year': 2024}, page_content='선풍기...'),
 Document(metadata={'year': 2024, 'user_rating': 4.9, 'category': '선풍기'}, page_content='선풍기...'),
 Document(metadata={'year': 2024, 'user_rating': 4.9, 'category': '선풍기'}, page_content='선풍기...')]
```

- 4개의 결과 검색된 이유 확인해보기 (9.5s)

```
[Document(metadata={'year': 2024, 'category': '선풍기', 'user_rating': 4.9}, page_content='선풍기...'),
 Document(metadata={'category': '선풍기', 'user_rating': 4.9, 'year': 2024}, page_content='선풍기...'),
 Document(metadata={'year': 2024, 'user_rating': 4.9, 'category': '선풍기'}, page_content='선풍기...'),
 Document(metadata={'year': 2024, 'user_rating': 4.9, 'category': '선풍기'}, page_content='선풍기...')]
```

- 한글 쿼리가 되지 않는 이유 살펴보기
  - 원인: LLM이 선풍기를 선풍기로 매핑하지 못함

```
# ❌ 실패
retriever.invoke("선풍기 제품 찾아줘") # 0개

# ✅ 성공
retriever.invoke("선풍기 제품 찾아줘") # 4개
```

- ◦ 증거: Chroma가 같은 문서를 여러번 저장함

```
{'year': 2024, 'category': '선풍기', 'user_rating': 4.9},
{'category': '선풍기', 'user_rating': 4.9, 'year': 2024},
{'year': 2024, 'user_rating': 4.9, 'category': '선풍기'},
{'user_rating': 4.9, 'category': '선풍기', 'year': 2024}
```

- ◦ 해결책: 중복을 방지하기 위해 각 문서에 고유 ID 부여하기

- error 방지 위한 조치
  - Chroma DB 파일이 다른 프로세스에서 열려있음을 방지하기
  - 파일 권한 문제 (읽기 전용으로 잠겨있지 않은지 확인하기)
  - Jupyter 커널이 이전 연결을 유지하지 않도록 하기

```
# =====
# 🚨 응급처치: Jupyter 커널 재시작 전에 실행!
# =====

import shutil
import os
import gc

# 1. 모든 Chroma 연결 해제
try:
    if 'vectorstore_clean' in globals():
        del vectorstore
    if 'vectorstore2' in globals():
        del vectorstore2
    gc.collect() # 가비지 컬렉션 강제 실행
    print("✅ 기존 연결 해제 완료")
except:
    pass

# 2. DB 디렉토리 강제 삭제
chroma_db_path = "./chroma_cosmetic_db"

# 여러 경로 시도
possible_paths = [
    "./chroma_cosmetic_db",
    "./chroma_db",
    "./chroma",
    "./cosmetics_clean"
]

for path in possible_paths:
    if os.path.exists(path):
        try:
            shutil.rmtree(path)
            print(f"✅ {path} 삭제 완료")
        except Exception as e:
            print(f"⚠️ {path} 삭제 실패: {e}")

print("\n🔄 이제 Jupyter 커널을 재시작하세요!")
print("   메뉴: Kernel -> Restart Kernel")
```

• 셀 출력 → **커널 재시작하기**

```
✅ 기존 연결 해제 완료

🔄 이제 Jupyter 커널을 재시작하세요!
메뉴: Kernel -> Restart Kernel
```

✓ 6) 각 문서에 고유 ID 부여해 다시 벡터스토어 생성하기

```
# 각 문서에 고유 ID 부여해 다시 시도하기

from langchain_chroma import Chroma
from langchain_core.documents import Document
from langchain_huggingface import HuggingFaceEmbeddings
from langchain.chains.query_constructor.base import AttributeInfo
from langchain.retrievers.self_query.base import SelfQueryRetriever
from langchain_openai import ChatOpenAI
import uuid

# 1. 임베딩 모델 생성
multilingual_embeddings = HuggingFaceEmbeddings(
    model_name="sentence-transformers/paraphrase-multilingual-mpnet-base-v2",
    model_kwargs={'device': 'cpu'},
    encode_kwargs={
        'normalize_embeddings': True,
    }
)

# 12.5s
```

```
# 2. 중복 방지: 각 문서에 고유 ID 부여

docs_unique = [
    Document(
        page_content="수분 가득한 히알루론산 세럼으로 피부 속 깊은 곳까지 수분을 공급합니다.",
        metadata={
            "id": "doc_001",
            "year": 2024,
            "category": "skincare",
        }
    )
]

# 고유 ID!
```

```

        "user_rating": 4.7
    },
),
Document(
    page_content="24시간 지속되는 매트한 피니시의 파운데이션, 모공을 커버하고 자연스러운 피부 표현이 가능합니다.",
    metadata={
        "id": "doc_002",
        "year": 2023,
        "category": "makeup",
        "user_rating": 4.5
    },
),
Document(
    page_content="식물성 성분으로 만든 저자극 클렌징 오일, 메이크업과 노폐물을 부드럽게 제거합니다.",
    metadata={
        "id": "doc_003",
        "year": 2023,
        "category": "cleansing",
        "user_rating": 4.8
    },
),
Document(
    page_content="비타민 C 함유 브라이트닝 크림, 칙칙한 피부톤을 환하게 밝혀줍니다.",
    metadata={
        "id": "doc_004",
        "year": 2023,
        "category": "skincare",
        "user_rating": 4.6
    },
),
Document(
    page_content="롱래스팅 립스틱, 선명한 발색과 촉촉한 사용감으로 하루종일 편안하게 사용 가능합니다.",
    metadata={
        "id": "doc_005",
        "year": 2024,
        "category": "makeup",
        "user_rating": 4.4
    },
),
Document(
    page_content="자외선 차단 기능이 있는 톤업 선크림, SPF50+/PA++++ 높은 자외선 차단 지수로 피부를 보호합니다.",
    metadata={
        "id": "doc_006",
        "year": 2024,
        "category": "suncare",
        "user_rating": 4.9
    },
),
],

```

```

# 3. ✅ 타임스탬프로 고유한 경로 생성 (충돌 방지!)
import time

timestamp = int(time.time())
chroma_db_path_new = f"./chroma_cosmetic_{timestamp}"

print(f"📁 새 DB 경로: {chroma_db_path_new}")

```

- 📁 새 DB 경로: ./chroma\_cosmetic\_1759394599

```

# 4. ✅ persist_directory 없이 생성 (메모리 모드)
# 또는 새 경로에 생성
vectorstore_memory = Chroma.from_documents(
    documents=docs_unique,
    embedding=multilingual_embeddings,
    collection_name=f"cosmetics_{timestamp}",           # 고유 이름
    # persist_directory=None, # 메모리 모드 (권장!)
    # 또는:
    #persist_directory=chroma_db_path_new,               # 새 경로
)

#print(f"✅ 벡터스토어 생성 완료: {vectorstore_fresh._collection.count()}개 문서")
print(f"✅ 메모리 벡터스토어 생성: {vectorstore_memory._collection.count()}개")

```

- ✅ 벡터스토어 생성: 6개 (0.9s)

```

# 5. 메타데이터 정의

```

```

metadata_field_info = [
    AttributeInfo(
        name="category",

```

```

        description="Product category: 'skincare' (스킨케어), 'makeup' (메이크업), 'cleansing' (클렌징), 'suncare' (선풀어)",
        type="string",
    ),
    AttributeInfo(
        name="year",
        description="Release year",
        type="integer",
    ),
    AttributeInfo(
        name="user_rating",
        description="User rating (1.0-5.0)",
        type="float",
    ),
),
]

```

```

# 6. LLM
llm_openai = ChatOpenAI(
    model="gpt-4o-mini",
    temperature=0,
)

```

```

# 7. Retriever

retriever_fixed = SelfQueryRetriever.from_llm(
    llm=llm_openai,
    vectorstore=vectorstore_memory,
    document_contents="Brief summary of a cosmetic product",
    metadata_field_info=metadata_field_info,
    enable_limit=True,          # k 파라미터 활성화
    verbose=True,              # 디버깅용
)
# 0.3s

```

```

# 8. 후처리 함수 (중복 제거 + k 제한)
def deduplicated_search(retriever, query, k=3):
    """중복 제거된 검색 결과 반환"""

    results = retriever.invoke(query)

    # ID 기반 중복 제거
    seen_ids = set()
    unique_results = []

    for doc in results:
        doc_id = doc.metadata.get("id")
        if doc_id not in seen_ids:
            seen_ids.add(doc_id)
            unique_results.append(doc)

    # k개만 반환
    return unique_results[:k]

```

```

# =====
# 🚀 테스트!
# =====

print("\n" + "=" * 60)
print("테스트 1: 한글 쿼리 (선풀어)")
print("=" * 60)
results1 = deduplicated_search(retriever_fixed, "선풀어 제품을 찾아줘", k=3)
print(f"✅ {len(results1)}개 검색됨\n")
for i, doc in enumerate(results1, 1):
    print(f"{i}. [{doc.metadata['category']}] {doc.page_content[:40]}...")
    print(f"📅 {doc.metadata['year']}년 | ⭐ {doc.metadata['user_rating']}")

print("\n" + "=" * 60)
print("테스트 2: 2024년 제품")
print("=" * 60)
results2 = deduplicated_search(retriever_fixed, "2024년에 출시된 제품", k=3)
print(f"✅ {len(results2)}개 검색됨\n")
for i, doc in enumerate(results2, 1):
    print(f"{i}. [{doc.metadata['category']}] {doc.page_content[:40]}...")
    print(f"📅 {doc.metadata['year']}년 | ⭐ {doc.metadata['user_rating']}")

print("\n" + "=" * 60)
print("테스트 3: 평점 4.7 이상")
print("=" * 60)
results3 = deduplicated_search(retriever_fixed, "평점이 4.7 이상인 제품", k=3)
print(f"✅ {len(results3)}개 검색됨\n")
for i, doc in enumerate(results3, 1):
    print(f"{i}. [{doc.metadata['category']}] {doc.page_content[:40]}...")
    print(f"📅 {doc.metadata['year']}년 | ⭐ {doc.metadata['user_rating']}")

```

```
print("\n🎉 모든 테스트 완료!")
```

• test ( 3.3s )

```
=====
테스트 1: 한글 쿼리 (선택어)
=====
```

✅ 3개 검색됨

1. [makeup] 24시간 지속되는 매트한 피니시의 파운데이션, 모공을 커버하고 자연스러운...  
📅 2023년 | ⭐ 4.5
2. [suncare] 자외선 차단 기능이 있는 톤업 선크림, SPF50+/PA++++ 높은 자...  
📅 2024년 | ⭐ 4.9
3. [skincare] 비타민 C 함유 브라이팅 크림, 칙칙한 피부톤을 환하게 밝혀줍니다....  
📅 2023년 | ⭐ 4.6

```
=====
테스트 2: 2024년 제품
=====
```

✅ 3개 검색됨

1. [suncare] 자외선 차단 기능이 있는 톤업 선크림, SPF50+/PA++++ 높은 자...  
📅 2024년 | ⭐ 4.9
2. [makeup] 롱래스팅 립스틱, 선명한 발색과 촉촉한 사용감으로 하루종일 편안하게 사용...  
📅 2024년 | ⭐ 4.4
3. [skincare] 수분 가득한 히알루론산 세럼으로 피부 속 깊은 곳까지 수분을 공급합니다....  
📅 2024년 | ⭐ 4.7

```
=====
테스트 3: 평점 4.7 이상
=====
```

✅ 3개 검색됨

1. [suncare] 자외선 차단 기능이 있는 톤업 선크림, SPF50+/PA++++ 높은 자...  
📅 2024년 | ⭐ 4.9
2. [cleansing] 식물성 성분으로 만든 저자극 클렌징 오일, 메이크업과 노폐물을 부드럽게 ...  
📅 2023년 | ⭐ 4.8
3. [skincare] 수분 가득한 히알루론산 세럼으로 피부 속 깊은 곳까지 수분을 공급합니다....  
📅 2024년 | ⭐ 4.7

🎉 모든 테스트 완료!

## 7) Query 테스트

```
# Self-query_1
```

```
print("\n" + "=" * 60)
print("Self-query_1: 평점이 4.8 이상인 제품")
print("=" * 60)
results1 = deduplicated_search(retriever_fixed, "평점이 4.8 이상인 제품", k=3)
print(f"✅ {len(results1)}개 검색됨\n")
for i, doc in enumerate(results1, 1):
    print(f"{i}. [{doc.metadata['category']}] {doc.page_content[:40]}...")
    print(f"📅 {doc.metadata['year']}년 | ⭐ {doc.metadata['user_rating']}")
```

• Self-query\_1 ( 0.9s )

```
=====
Self-query_1: 평점이 4.8 이상인 제품
=====
```

✅ 2개 검색됨

1. [suncare] 자외선 차단 기능이 있는 톤업 선크림, SPF50+/PA++++ 높은 자...  
📅 2024년 | ⭐ 4.9
2. [cleansing] 식물성 성분으로 만든 저자극 클렌징 오일, 메이크업과 노폐물을 부드럽게 ...  
📅 2023년 | ⭐ 4.8

```
# Self-query_2
```

```
print("\n" + "=" * 60)
print("Self-query_2: 2023년에 출시된 상품")
print("=" * 60)
results1 = deduplicated_search(retriever_fixed, "2023년에 출시된 상품", k=5) # 최대 5개까지
print(f"✅ {len(results1)}개 검색됨\n")
for i, doc in enumerate(results1, 1):
    print(f"{i}. [{doc.metadata['category']}] {doc.page_content[:40]}...")
    print(f"📅 {doc.metadata['year']}년 | ⭐ {doc.metadata['user_rating']}")
```

#### • Self-query\_2 (1.5s)

```
=====
Self-query_2: 2023년에 출시된 상품
=====
✅ 3개 검색됨

1. [cleansing] 식물성 성분으로 만든 저자극 클렌징 오일, 메이크업과 노폐물을 부드럽게 ...
📅 2023년 | ⭐ 4.8
2. [makeup] 24시간 지속되는 매트한 피니시의 파운데이션, 모공을 커버하고 자연스러운...
📅 2023년 | ⭐ 4.5
3. [skincare] 비타민 C 함유 브라이팅 크림, 칙칙한 피부톤을 환하게 밝혀줍니다....
📅 2023년 | ⭐ 4.6
```

```
# Self-query_3
```

```
print("\n" + "=" * 60)
print("Self-query_3: suncare 상품")
print("=" * 60)
results1 = deduplicated_search(retriever_fixed, "suncare 상품", k=1)
print(f"✅ {len(results1)}개 검색됨\n")
for i, doc in enumerate(results1, 1):
    print(f"{i}. [{doc.metadata['category']}] {doc.page_content[:40]}...")
    print(f"📅 {doc.metadata['year']}년 | ⭐ {doc.metadata['user_rating']}")
```

#### • Self-query\_3 (1.2s)

```
=====
Self-query_3: suncare 상품
=====
✅ 1개 검색됨

1. [suncare] 자외선 차단 기능이 있는 톤업 선크림, SPF50+/PA++++ 높은 자...
📅 2024년 | ⭐ 4.9
```

#### • 복합 필터

```
# Self-query_4
```

```
print("\n" + "=" * 60)
print("Self-query_4: makeup 상품 & 평점이 4.5 이상인 상품")
print("=" * 60)
results1 = deduplicated_search(retriever_fixed, "makeup 상품 & 평점이 4.5 이상인 상품", k=1)
print(f"✅ {len(results1)}개 검색됨\n")
for i, doc in enumerate(results1, 1):
    print(f"{i}. [{doc.metadata['category']}] {doc.page_content[:40]}...")
    print(f"📅 {doc.metadata['year']}년 | ⭐ {doc.metadata['user_rating']}")
```

```
# Self-query_4
```

```
print("\n" + "=" * 60)
print("Self-query_4: makeup 상품 중 평점이 4.5 이상인 상품")
print("=" * 60)
results1 = deduplicated_search(retriever_fixed, "반드시 makeup 상품이면서 평점이 4.5 이상인 상품")
print(f"✅ {len(results1)}개 검색됨\n")
for i, doc in enumerate(results1, 1):
    print(f"{i}. [{doc.metadata['category']}] {doc.page_content[:40]}...")
    print(f"📅 {doc.metadata['year']}년 | ⭐ {doc.metadata['user_rating']}")
```

- **k** = 가져올 문서의 수 조정
  - **SelfQueryRetriever** 파라미터 중 **enable\_limit = True** 설정

```
retriever = SelfQueryRetriever.from_llm(
    llm=llm_openai,
    vectorstore=vectorstore_memory,
    document_contents="Brief summary of a cosmetic product",
    metadata_field_info=metadata_field_info,
    enable_limit=True,          # 검색 결과 제한 기능 활성화
    search_kwargs={"k": 2},     # k의 값 = 2 → 검색 결과를 2개로 제한
)
```

- 2023년도 출시된 상품은 3개가 있지만 **k** 값을 2개로 지정 → **2개만 반환하도록 함**

```
# Self-query_5

print("\n" + "=" * 60)
print("Self-query_5: 2023년에 출시된 상품")
print("=" * 60)
results1 = deduplicated_search(retriever, "2023년에 출시된 상품")
print(f"✅ {len(results1)}개 검색됨\n")
for i, doc in enumerate(results1, 1):
    print(f"{i}. [{doc.metadata['category']}] {doc.page_content[:40]}...")
    print(f"📅 {doc.metadata['year']}년 | ⭐ {doc.metadata['user_rating']}")
```

- Self-query\_5 ( 1.7s )

```
=====
Self-query_5: 2023년에 출시된 상품
=====
✅ 2개 검색됨

1. [cleansing] 식물성 성분으로 만든 저자극 클렌징 오일, 메이크업과 노폐물을 부드럽게 ...
📅 2023년 | ⭐ 4.8
2. [makeup] 24시간 지속되는 매트한 피니시의 파운데이션, 모공을 커버하고 자연스러운...
📅 2023년 | ⭐ 4.5
```

- 코드 ( **search\_kwargs** )로 명시적으로 지정하지 않고, **query** 에서 **1개**, **2개** 등의 숫자를 사용해 검색 결과를 제한할 수 있음

```
retriever2 = SelfQueryRetriever.from_llm(
    llm=llm_openai,
    vectorstore=vectorstore_memory,
    document_contents="Brief summary of a cosmetic product",
    metadata_field_info=metadata_field_info,
    enable_limit=True,
)
```

```
# Self-query_6 ver.1
retriever2.invoke("2023년에 출시된 상품 1개를 추천해주세요")
```

```
# Self-query_6 ver.2

print("\n" + "=" * 60)
print("Self-query_6: 2023년에 출시된 상품 1개를 추천해주세요")
print("=" * 60)
results1 = deduplicated_search(retriever2, "2023년에 출시된 상품 1개를 추천해주세요")
print(f"✅ {len(results1)}개 검색됨\n")
for i, doc in enumerate(results1, 1):
    print(f"{i}. [{doc.metadata['category']}] {doc.page_content[:40]}...")
    print(f"📅 {doc.metadata['year']}년 | ⭐ {doc.metadata['user_rating']}")
```

- Self-query\_6 ver.1 ( 2.4s )

```
[Document(id='996155f0-cf85-4761-b1bf-3a75a4006d7e', metadata={'id': 'doc_003', 'category': 'cleansing', 'year':
```

- Self-query\_6 ver.2 ( 1.9s )

```
=====
Self-query_6: 2023년에 출시된 상품 1개를 추천해주세요
=====
```

✅ 1개 검색됨

1. [cleansing] 식물성 성분으로 만든 저자극 클렌징 오일, 메이크업과 노폐물을 부드럽게 ...  
📅 2023년 | ⭐ 4.8

```
# Self-query_7 ver.1
retriever.invoke("2023년에 출시된 상품 2개를 추천해주세요")
```

```
# Self-query_7 ver.2

print("\n" + "=" * 60)
print("Self-query_7: 2023년에 출시된 상품 2개를 추천해주세요")
print("=" * 60)
results1 = deduplicated_search(retriever2, "2023년에 출시된 상품 2개를 추천해주세요")
print(f"✅ {len(results1)}개 검색됨\n")
for i, doc in enumerate(results1, 1):
    print(f"{i}. [{doc.metadata['category']}] {doc.page_content[:40]}...")
    print(f"📅 {doc.metadata['year']}년 | ⭐ {doc.metadata['user_rating']}")
```

- Self-query\_7 ver.1 (1.2s)

```
[Document(id='996155f0-cf85-4761-b1bf-3a75a4006d7e', metadata={'category': 'cleansing', 'year': 2023, 'user_rating': 4.8}),
 Document(id='7e44e475-ceb6-4ba3-be2b-d48216669931', metadata={'year': 2023, 'id': 'doc_002', 'category': 'makeup', 'user_rating': 4.5})]
```

- Self-query\_7 ver.2 (1.3s)

```
=====
Self-query_7: 2023년에 출시된 상품 2개를 추천해주세요
=====

✅ 2개 검색됨

1. [cleansing] 식물성 성분으로 만든 저자극 클렌징 오일, 메이크업과 노폐물을 부드럽게 ...
📅 2023년 | ⭐ 4.8
2. [makeup] 24시간 지속되는 매트한 피니시의 파운데이션, 모공을 커버하고 자연스러운...
📅 2023년 | ⭐ 4.5
```

## 8) 더 깊게 들어가기

- 처음부터 **retriever** 재구성 가능
  - **query-construction chain** 생성에서부터 시작함
- [참고 튜토리얼](#)
- **query\_constructor chain** 생성
  - **get\_query\_constructor\_prompt** 함수 사용 → 쿼리 생성기 프롬프트 가져오기

```
from langchain.chains.query_constructor.base import (
    StructuredQueryOutputParser,
    get_query_constructor_prompt,
)

# 쿼리 생성기 프롬프트 가져오기
prompt = get_query_constructor_prompt(
    "Brief summary of a cosmetic product",          # 문서 내용 설명
    metadata_field_info,                            # 메타데이터 필드 정보
)

# StructuredQueryOutputParser 를 생성
output_parser = StructuredQueryOutputParser.from_components()

# query_constructor chain 생성
```



```
query_constructor = prompt | llm_openai | output_parser
```

- `query_constructor.invoke()` 메서드 호출 → 주어진 쿼리에 대한 처리 수행

```
query_output = query_constructor.invoke(  
    {  
        # 쿼리 생성기 호출 → 주어진 질문에 대한 쿼리 생성  
        "query": "2023년도에 출시한 상품 중 평점이 4.5 이상인 상품중에서 스킨케어 제품을 추천해주세요"  
    }  
)  
  
# 1.5s
```

- 생성된 쿼리 확인해보기

```
# 쿼리 출력해보기  
  
query_output.filter.arguments
```

- 생성된 쿼리 출력해보기

```
[Comparison(comparator=<Comparator.EQ: 'eq'>, attribute='year', value=2023),  
Comparison(comparator=<Comparator.GTE: 'gte'>, attribute='user_rating', value=4.5),  
Comparison(comparator=<Comparator.EQ: 'eq'>, attribute='category', value='skincare')]
```

## 9) 구조화된 쿼리 변환기를 사용해 구조화된 쿼리로 변환하기

- `structured query translator`
  - 일반적인 `StructuredQuery` 객체를 사용중인 `vector store` 구문에 맞는 메타데이터 필터로 변환하는 역할 담당

```
from langchain.retrievers.self_query.chroma import ChromaTranslator  
  
retriever3 = SelfQueryRetriever(  
    query_constructor=query_constructor, # 이전에 생성한 query_constructor chain을 지정  
    vectorstore=vectorstore_memory, # 벡터 저장소를 지정  
    structured_query_translator=ChromaTranslator(), # 쿼리 변환기  
)
```

- `retriever.invoke()` 메서드 사용 → 주어진 질문에 대한 답변 생성하기

```
retriever3.invoke(  
    "2023년도에 출시한 상품 중 평점이 4.5 이상인 상품중에서 스킨케어 제품을 추천해주세요" # 질문  
)
```

- Structured query translator (2.2s)

```
[Document(id='6768ce36-89aa-4bd3-a81b-5da51c1b1bbe', metadata={'year': 2023, 'id': 'doc_004', 'category': 'skince
```

```
# Structured query translator  
  
print("\n" + "=" * 60)  
print("Structured query translator: 2023년도에 출시한 상품 중 평점이 4.5 이상인 상품중에서 스킨케어 제품을 추천해주세요")  
print("=" * 60)  
results3 = deduplicated_search(retriever3, "2023년도에 출시한 상품 중 평점이 4.5 이상인 상품중에서 스킨케어 제품을 추천해주세요")  
print(f"✅ {len(results3)}개 검색됨\n")  
for i, doc in enumerate(results3, 1):  
    print(f"{i}. [{doc.metadata['category']}] {doc.page_content[:40]}...")  
    print(f"📅 {doc.metadata['year']}년 | ⭐ {doc.metadata['user_rating']}")
```

- Structured query translator (2.2s)

```
[Document(id='6768ce36-89aa-4bd3-a81b-5da51c1b1bbe', metadata={'year': 2023, 'id': 'doc_004', 'category': 'skince
```

- 깔끔하게 출력하기 (1.4s)

=====

Structured query translator: 2023년도에 출시한 상품 중 평점이 4.5 이상인 상품중에서 스킨케어 제품을 추천해주세요

=====

✔

 1개 검색됨

1.

 [skincare] 비타민 C 함유 브라이팅 크림, 칙칙한 피부를 환하게 밝혀줍니다....

2023년 | 

★

 4.6

- 
- next: 시간 가중 벡터저장소 검색기 (TimeWeightedVectorStoreRetriever)
-