


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

✓ CH09 벡터저장소 (VectorStore)

- 벡터스토어 저장 단계 = RAG 의 4번째 단계
 - 이전 단계에서 생성된 임베딩 벡터들을 효율적으로 저장하고 관리하는 과정
 - 향후 검색 과정에서 벡터들을 빠르게 조회하고, 관련 문서를 신속하게 찾아내는 데 필수적
- 벡터스토어 저장의 필요성
 - 빠른 검색 속도: 임베딩 벡터들을 효과적으로 저장 • 색인화 → 대량의 데이터 중에서 관련된 정보를 빠르게 검색 가능
 - 스케일러빌리티
 - 데이터 지속적으로 증가 → 수용할 수 있는 충분한 스케일러빌리티 제공 해야 함
 - 효율적인 저장 구조 → 데이터베이스의 확장성 보장, 시스템의 성능 저하 없이 대규모 데이터 관리 가능
 - 의미 검색 (Semantic Search) 지원 → 사용자의 질문과 의미상으로 유사한 단락 조회
0 → 벡터스토어 지원 가능한 기능
 - 텍스트 자체가 저장되는 DB: 키워드 기반 검색에 의존해야 하는 한계성 O
 - 벡터스토어: 의미적으로 유사한 단락 검색 가능
- 예시: Q. "모바일 디바이스 상에서 동작하는 인공지능 기술을 소개한 기업명은?"
 사용자 질문

- 벡터스토어 중요성
 - 벡터스토어의 저장 단계 = RAG 시스템의 검색 기능과 직접적으로 연결 됨
 - 전체 시스템의 응답 시간과 정확성에 큰 영향 미침
 - → 데이터 관리 용이 → 필요 시 즉시 접근 가능 → 사용자에게 신속하고 정확한 정보 제공 가능



코드

- 코드 예시

```
from langchain_community.vectorstores import FAISS

# 단계 4: DB 생성(Create DB) 및 저장
# 벡터스토어 생성하기
vectorstore = FAISS.from_documents(documents=documents, embedding=embeddings)
```



참고

- [Embedding](#)
 - [LangChain VectorStores](#)
-



1. Chroma

- **Chroma**
 - 개발자의 생산성과 행복에 초점을 맞춘 AI 네이티브 오픈 소스 벡터 데이터베이스
 - **Apache 2.0**에 따른 라이선스 부여
 - 참고 링크
 - [Chroma LangChain 문서](#)
 - [Chroma 공식 문서](#)
 - [LangChain 지원 VectorStore 리스트](#)
-



1) 설정

- 먼저 **langchain-openai** 설치 → **필요한 환경 변수**를 **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

if langchain_tracing_v2 == "true" and
    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 대시보드에 로그인하세요.")
else:
    print("❌ LangSmith 추적이 완전히 활성화되지 않았습니다.")
    if langchain_tracing_v2 != "true":
        print("    - LANGCHAIN_TRACING_V2가 'true'로 설정되어 있지 않습니다.")
    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.
```

• 샘플 데이터셋 로드하기

- 오류 생길 경우: 사전에 VS Code 터미널에 설치할 것

```
pip install langchain-chroma
```

```
from langchain_community.document_loaders import TextLoader
from langchain_google_genai import GoogleGenerativeAIEmbeddings
```

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_chroma import Chroma
```

```
# 텍스트 분할
```

```
text_splitter = RecursiveCharacterTextSplitter(chunk_size=600, chunk_overlap=0)
```

```
# 텍스트 파일을 load → List[Document] 형태로 변환
```

```
loader1 = TextLoader("../09_VectorStore/data/nlp-keywords.txt")
```

```
loader2 = TextLoader("../09_VectorStore/data/finance-keywords.txt")
```

```
print(type(loader1))      # <class 'langchain_community.document_loaders.text.TextL
print(type(loader2))      # <class 'langchain_community.document_loaders.text.TextL
```

```
# 문서 분할
```

```
split_doc1 = loader1.load_and_split(text_splitter)
```

```
split_doc2 = loader2.load_and_split(text_splitter)
```

```
print(type(split_doc1))    # <class 'list'>
print(type(split_doc2))    # <class 'list'>
```

```
# 문서 개수 확인
```

```
print(f"첫번째 문서의 개수: {len(split_doc1)}")
```

```
print("\n", "="*25, "\n")
```

```
print(f"두번째 문서의 개수: {len(split_doc2)}")
```

- 셀 출력

```
첫번째 문서의 개수: 11
```

```
=====
```

```
두번째 문서의 개수: 6
```

▼ 2) VectorStore 생성

- 벡터 저장소 생성 (`from_documents`): 문서 리스트 → 벡터 저장소 생성
- 매개변수
 - `documents` (`List[Document]`): 벡터 저장소에 추가할 문서 리스트
 - `embedding` (`Optional[Embeddings]`): 임베딩 함수. 기본값은 None
 - `ids` (`Optional[List[str]]`): 문서 ID 리스트. 기본값은 None
 - `collection_name` (`str`): 생성할 컬렉션 이름.

- `persist_directory` (Optional[str]): 컬렉션을 저장할 디렉토리. 기본값은 None
 - `client_settings` (Optional[chromadb.config.Settings]): Chroma 클라이언트 설정
 - `client` (Optional[chromadb.Client]): Chroma 클라이언트 인스턴스
 - `collection_metadata` (Optional[Dict]): 컬렉션 구성 정보. 기본값은 None
- **참고**
 - **`persist_directory`**
 - 지정 시 컬렉션이 해당 디렉토리에 저장
 - 지정되지 않으면 데이터는 메모리에 임시로 저장됨
 - **`from_texts`**: 내부적으로 해당 메서드 호출 → 벡터 저장소 생성
 - 문서
 - **`page_content`** = text
 - **`metadata`** = metadata
 - **반환값**
 - Chroma
 - 생성된 Chroma 벡터 저장소 인스턴스의 **`documents`** 매개변수로 **`Document`** 리스트 전달
 - **`embedding`**에 활용할 임베딩 모델 지정
 - **`namespace`**의 역할을 하는 **`collection_name`** 지정 가능

```
from langchain_google_genai import GoogleGenerativeAIEmbeddings

from dotenv import load_dotenv
import os

load_dotenv()

# API 키 확인
if not os.getenv("GOOGLE_API_KEY"):
    os.environ["GOOGLE_API_KEY"] = input("Enter your Google API key: ")
```

```
# Gemini 임베딩 모델 생성 (task_type 명시)
embeddings = GoogleGenerativeAIEmbeddings(
    model="models/gemini-embedding-001",
    task_type="retrieval_document",
    google_api_key=os.getenv("GOOGLE_API_KEY")
)
```

DB 생성 (3.1s 소요)

```
db = Chroma.from_documents(  
    documents=split_doc1,          # 문서 리스트 전달  
    embedding=embeddings,         # 임베딩 모델 지정  
    collection_name="my_db"       # 생성할 컬렉션 이름 설정  
)
```

- `persist_directory` 지정 시 → `disk`에 파일 형태로 저장

```
# 저장할 경로 지정  
DB_PATH = "../09_VectorStore/chroma_db/"
```

```
# 문서를 디스크에 저장하기  
# 저장시 persist_directory에 저장할 경로 지정하기  
persist_db = Chroma.from_documents(  
    split_doc1, embeddings, persist_directory=DB_PATH, collection_name="my_db"  
)
```

- `DB_PATH`에 저장된 데이터 로드해보기

```
# 디스크에서 문서 로드하기  
  
persist_db = Chroma(  
    persist_directory=DB_PATH,  
    embedding_function=embeddings,  
    collection_name="my_db",  
)
```

- 불러온 `VectorStore`에서 저장된 데이터 확인하기

```
# 저장된 데이터 확인  
  
persist_db.get()
```

- 셀 출력

```
{'ids': ['e2bcb7fd-40bf-4e86-8eeb-0b195ca8da5d',  
        '614593b7-71fb-42e4-a954-e967f460c2e5',  
        '15e1303c-c7dc-4872-a826-392223220c0a',  
        '9e2d6752-2210-42c0-a2e1-0667f9b971ee',  
        '38a6d754-0ee0-4aff-843f-f62a03663e18',  
        'cea859e8-2584-4930-b82f-64abdf973757',  
        '9aa2bf6a-bb4c-4160-b886-f6989dabd265',  
        '9f805e9b-edc4-462e-871f-a627bd98f4d3',  
        '1c057883-26c0-4b59-906e-0c55091716b7',  
        '5706ef24-0425-4483-8578-cc7b26138b70',  
        '977bf0ba-3bef-4029-a501-b9c0fedbf5e2']},
```

```

'embeddings': None,
'documents': ['Semantic Search\n\n정의: 의미론적 검색은 사용자의 질의를 단순한 키워드 매칭을
'정의: 토큰라이저는 텍스트 데이터를 토큰으로 분할하는 도구입니다. 이는 자연어 처리에서 데이터를 전처리
'정의: CSV(Comma-Separated Values)는 데이터를 저장하는 파일 형식으로, 각 데이터 값은 쉼표로
'정의: HuggingFace는 자연어 처리를 위한 다양한 사전 훈련된 모델과 도구를 제공하는 라이브러리입니다
'정의: Word2Vec은 단어를 벡터 공간에 매핑하여 단어 간의 의미적 관계를 나타내는 자연어 처리 기술입니
'정의: 오픈 소스는 소스 코드가 공개되어 누구나 자유롭게 사용, 수정, 배포할 수 있는 소프트웨어를 의미
'정의: TF-IDF는 문서 내에서 단어의 중요도를 평가하는 데 사용되는 통계적 척도입니다. 이는 문서 내 단
"정의: DataFrame은 행과 열로 이루어진 테이블 형태의 데이터 구조로, 주로 데이터 분석 및 처리에 사용
'GPT (Generative Pretrained Transformer)\n\n정의: GPT는 대규모의 데이터셋으로 사전 훈련
'정의: 키워드 검색은 사용자가 입력한 키워드를 기반으로 정보를 찾는 과정입니다. 이는 대부분의 검색 엔진
'정의: 멀티모달은 여러 종류의 데이터 모드(예: 텍스트, 이미지, 소리 등)를 결합하여 처리하는 기술입니
'uris': None,
'included': ['metadatas', 'documents'],
'data': None,
'metadatas': [{ 'source': '../09 VectorStore/data/nlp-keywords.txt'},
{ 'source': '../09 VectorStore/data/nlp-keywords.txt'},
{ 'source': '../09 VectorStore/data/nlp-keywords.txt'},
{ 'source': '../09 VectorStore/data/nlp-keywords.txt'},
{ 'source': '../09 VectorStore/data/nlp-keywords.txt'},
{ 'source': '../09 VectorStore/data/nlp-keywords.txt'},
{ 'source': '../09 VectorStore/data/nlp-keywords.txt'},
{ 'source': '../09 VectorStore/data/nlp-keywords.txt'},
{ 'source': '../09 VectorStore/data/nlp-keywords.txt'},
{ 'source': '../09 VectorStore/data/nlp-keywords.txt'}]}

```

- `collection_name` 을 다르게 지정할 경우: 아무런 결과 얻지 못함 (저장된 데이터가 없으므로)

디스크에서 문서 로드하기

```

persist_db2 = Chroma(
    persist_directory=DB_PATH,
    embedding_function=embeddings,
    collection_name="my_db2",
)

```

저장된 데이터 확인해보기

```

persist_db2.get()

```

```

{'ids': [],
 'embeddings': None,
 'documents': [],
 'uris': None,
 'included': ['metadatas', 'documents'],
 'data': None,
 'metadatas': []}

```

- 셀 출력

```
{'ids': [],  
'embeddings': None,  
'documents': [],  
'uris': None,  
'included': ['metadatas', 'documents'],  
'data': None,  
'metadatas': []}
```

- 벡터 저장소 생성 ②
- 참고
 - **ids**: 미제공 → **UUID** 자동 생성
- 반환값
 - 생성된 벡터 저장소 인스턴스

```
# 문자열 리스트로 생성  
db2 = Chroma.from_texts(  
    ["안녕하세요. 정말 반갑습니다.", "제 이름은 엘리스입니다."],  
    embedding=embeddings,  
)
```

```
# 데이터 조회해보기  
db2.get()
```

- 셀 출력

```
{'ids': ['78f79150-0676-4bf1-a434-31d2e84a12ba',  
'2af7b3f8-94bd-46c3-b6e7-8ce90797e97c'],  
'embeddings': None,  
'documents': ['안녕하세요. 정말 반갑습니다.', '제 이름은 엘리스입니다.'],  
'uris': None,  
'included': ['metadatas', 'documents'],  
'data': None,  
'metadatas': [None, None]}
```


- **similarity_search** 메서드

- **Chroma** 데이터베이스에서 유사도 검색 수행
- 주어진 쿼리와 가장 유사한 문서들을 반환

- 매개변수

- **query** (str): 검색할 쿼리 텍스트
- **k** (int, 선택적): 반환할 결과의 수 (**기본값 = 4**)
- **filter** (Dict[str, str], 선택적): 메타데이터로 필터링 (기본값 = None)

- 참고

- **k 값 조절**: 원하는 수의 결과 얻을 수 있음
- **filter 사용**: 특정 메타데이터 조건에 맞는 문서만 검색 가능
- 이 메서드는 **점수 정보없이 문서만 반환함**
- 점수 정보도 필요한 경우: **similarity_search_with_score** 직접 사용

- **반환값**: List [Document] = 쿼리 텍스트와 **가장 유사한 문서의 리스트**

```
db.similarity_search("TF IDF 에 대하여 알려줘")
```

- 셀 출력 (0.6s)

```
[Document(id='57005c56-b9a8-4d48-9dd1-c8e093be0bac', metadata={'source': '..../09
Document(id='213aac84-a1cd-401d-94f5-fd0f1236bdb6', metadata={'source': '..../09
Document(id='adb0e0c1-7787-4c8a-92fc-18ee6089d482', metadata={'source': '..../09
Document(id='d464f668-71ee-40a7-af5a-52de76c525f0', metadata={'source': '..../09
```

- **k 값에 따라 검색 결과의 개수 지정 가능**

```
db.similarity_search("TF IDF 에 대하여 알려줘", k=2)
```

- 셀 출력 (0.4s)(2개만 검색)

```
[Document(id='57005c56-b9a8-4d48-9dd1-c8e093be0bac', metadata={'source': '..../09
Document(id='213aac84-a1cd-401d-94f5-fd0f1236bdb6', metadata={'source': '..../09
```

- **filter** 에 **metadata** 정보를 활용해 검색 결과 필터링해보기

filter 사용

```
db.similarity_search(
    "TF IDF 에 대하여 알려줘",
    filter={"source": "../09_VectorStore/data/nlp-keywords.txt"},
    k=2
)
```

query = 검색 쿼리 텍스트
metadata = 검색 결과의 메타데이터
k = 검색 결과 개수 지정

- 셀 출력 (0.3s) (metadata 필터링까지 적용)

```
[Document(id='57005c56-b9a8-4d48-9dd1-c8e093be0bac', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'}, score=0.85),
 Document(id='213aac84-a1cd-401d-94f5-fd0f1236bdb6', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'}, score=0.75)]
```

- **filter** 에서 다른 **source** 를 사용해 검색한 결과 확인해보기

filter 사용

```
db.similarity_search(
    "TF IDF 에 대하여 알려줘",
    filter={"source": "../09_VectorStore/data/finance-keywords.txt"},
    k=2
)
```

[]

4) 벡터 저장소에 문서 추가해보기

- **add_documents** 메서드: 벡터 저장소에 문서를 추가 or 업데이트
- 매개변수
 - **documents** (**List** [**Document**]): 벡터 저장소에 추가할 문서 리스트
 - **kwargs**: 추가 키워드 인자
 - **ids**: 문서 **ID** 리스트 → 제공할 경우 문서의 **ID** 보다 우선시됨
- 참고
 - **add_texts** 메서드 구현되어 있어야 함

- 문서
 - `page_content` = `text`
 - `metadata` = `metadata`
- 문서의 `ID`
 - 문서의 `ID`가 사용되는 경우: 문서에 `ID`가 있고, `kwargs`에 `ID`가 제공되지 않는 경우
 - `kwargs`의 `ID`와 문서의 수가 일치하지 않는 경우 `ValueError` 발생
- 예외
 - `NotImplementedError`: `add_texts` 메서드가 구현되지 않은 경우 발생

```
from langchain_core.documents import Document

# page_content, metadata, id 지정
db.add_documents(
    [
        Document(
            page_content="안녕하세요! 이번엔 도큐먼트를 새로 추가해 볼게요",
            metadata={"source": "mydata.txt"},
            id="1",
        )
    ]
) # ['1']
```

```
# id=1 로 문서 조회

db.get("1")
```

- 셀 출력 (0.1s)

```
{'ids': ['1'],
 'embeddings': None,
 'documents': ['안녕하세요! 이번엔 도큐먼트를 새로 추가해 볼게요'],
 'uris': None,
 'included': ['metadatas', 'documents'],
 'data': None,
 'metadatas': [{'source': 'mydata.txt'}]}
```

-
- `add_texts` 메서드 = 텍스트 임베딩, 벡터 저장소에 추가
 - 매개변수

- **text** (Iterable [str]): 벡터 저장소에 추가할 텍스트 리스트
- **metadatas** (Optional [List [dict]]): 메타데이터 리스트 (*기본값 = None*)
- **ids** (Optional [List [str]]): 문서 ID 리스트 (*기본값 = None*)
- **참고**
 - **ids** 미제공시: **UUID** 사용 → 자동 생성
 - 임베딩 함수 설정 시 텍스트를 임베딩함
 - **메타데이터 제공된 경우**:
 - **분리 처리**: 메타데이터가 있는 텍스트, 없는 텍스트
 - 메타데이터가 없는 텍스트의 경우 = **빈 딕셔너리**
 - 컬렉션에 **upsert** 수행 → 텍스트, 임베딩, 메타데이터 추가
- **반환값**: List [str] = 추가된 텍스트의 ID 리스트
- **예외**
 - **ValueError**: 복잡한 메타데이터로 인한 오류 발생시
 - 필터링 방법 안내 메시지와 함께 발생 기존 아이디에 추가하는 경우 **upsert** 가 수행
 - 기존의 문서는 대체됨

```
# 신규 데이터 추가
# 이때 기존의 id=1 의 데이터는 덮어씀

db.add_texts(
    ["이전에 추가한 Document 를 덮어쓰겠습니다.", "덮어쓴 결과가 어떨까요?"],
    metadatas=[{"source": "mydata.txt"}, {"source": "mydata.txt"}],
    ids=["1", "2"],
)
# ['1', '2']
```

```
['1', '2']
```

```
# id=1 조회

db.get(["1"])
```

- 셀 출력

```
{'ids': ['1'],
 'embeddings': None,
 'documents': ['이전에 추가한 Document 를 덮어쓰겠습니다.'],
 'uris': None,
 'included': ['metadatas', 'documents'],
 'data': None,
```

```
'metadatas': [{'source': 'mydata.txt'}]}
```

```
# id=2 조회
```

```
db.get(["2"])
```

- 셀 출력

```
{'ids': ['2'],  
'embeddings': None,  
'documents': ['덮어쓴 결과가 어떤가요?'],  
'uris': None,  
'included': ['metadatas', 'documents'],  
'data': None,  
'metadatas': [{'source': 'mydata.txt'}]}
```

- id=1, id=2 조회한 결과

	코드	결과
id=1로 문서 조회	db.get("1")	{'ids': ['1'], 'embeddings': None, 'documents': ['안녕하세요! 이번엔 도큐먼트를 새로 추가해 볼게요'], 'uris': None, 'included': ['metadatas', 'documents'], 'data': None, 'metadatas': [{'source': 'mydata.txt'}]}
id=2로 문서 조회	db.get("2")	{'ids': ['2'], 'embeddings': None, 'documents': ['덮어쓴 결과가 어떤가요?'], 'uris': None, 'included': ['metadatas', 'documents'], 'data': None, 'metadatas': [{'source': 'mydata.txt'}]}

✓ 5) 벡터 저장소에서 문서 삭제

- **delete** 메서드: 벡터 저장소에서 지정된 ID의 문서 삭제
- 매개변수
 - **ids** (Optional [List [str]]): 삭제할 문서의 ID 리스트 (기본값 = None)
- 참고
 - 해당 메서드는 내부적으로 컬렉션의 **delete** 메서드 호출
 - **ids = None** → 아무 작업도 수행하지 않음

- 반환값 = None

```
# id=1 삭제
```

```
db.delete(ids=["1"])
```

```
# 문서 조회
```

```
db.get(["1", "2"])
```

- 셀 출력 (id=1의 값이 삭제된 것을 확인할 수 있음)

```
{'ids': ['2'],  
'embeddings': None,  
'documents': ['덮어쓴 결과가 어떨까요?'],  
'uris': None,  
'included': ['metadatas', 'documents'],  
'data': None,  
'metadatas': [{'source': 'mydata.txt'}]}
```

```
# where 조건으로 metadata 조회
```

```
db.get(where={"source": "mydata.txt"})
```

- 셀 출력

```
{'ids': ['2'],  
'embeddings': None,  
'documents': ['덮어쓴 결과가 어떨까요?'],  
'uris': None,  
'included': ['metadatas', 'documents'],  
'data': None,  
'metadatas': [{'source': 'mydata.txt'}]}
```

6) 초기화 (reset_collection)

- reset_colltion 메스드 = 벡터 저장소의 컬렉션 초기화

```
# 컬렉션 초기화
```

```
db.reset_collection()
```

```
# 초기화 후 문서 조회
```

```
db.get()
```

```
{'ids': [],  
 'embeddings': None,  
 'documents': [],  
 'uris': None,  
 'included': ['metadatas', 'documents'],  
 'data': None,  
 'metadatas': []}
```

- 셀 출력 (아무것도 검색되지 않음)

```
{'ids': [],  
 'embeddings': None,  
 'documents': [],  
 'uris': None,  
 'included': ['metadatas', 'documents'],  
 'data': None,  
 'metadatas': []}
```

7) 벡터저장소를 검색기(Retriever)로 변환

- **as_retriever** 메서드 = 벡터 저장소를 기반으로 **VectorStoreRetriever** 생성
- 매개변수
 - **kwargs**: 검색 함수에 전달할 **키워드 인자**
 - **search_type** (Optional [str]): 검색 유형
 - **similarity**
 - **mmr**
 - **similarity_score_threshold**
 - **search_kwargs** (Optional [Dict]): 검색 함수에 전달할 추가 인자
 - **k**: 반환할 문서 수 (**기본값 = 4**)
 - **score_threshold**: 최고 유사도 임계값

- **fetch_k**: MMR 알고리즘에 전달할 문서 수 (**기본값 = 20**)
- **lambda_mult**: MMR 결과의 다양성 조절 (**0~1, 기본값 = 0.5**)
- **filter**: 문서 메타데이터 필터링

- **반환값**: **VectorStoreRetriever** = 벡터 저장소 기반 검색기 인스턴스 **DB** 생성

```
# DB 생성 (1.4s 소요)

db = Chroma.from_documents(
    documents=split_doc1 + split_doc2,
    embedding=embeddings,
    collection_name="nlp",
)
```

- 기본 값으로 설정된 4개의 문서를 유사도 검색을 수행해서 조회해보기

```
retriever = db.as_retriever()
retriever.invoke("Word2Vec 에 대하여 알려줘")
```

- 셀 출력 (기본 값으로 설정된 4개의 문서를 유사도 검색으로 조회)

```
[Document(id='9174a219-9bba-41d6-a08c-9c2fe712d441', metadata={'source': '..../09
Document(id='ec2f75fb-88fd-4936-b929-40ed4bdea955', metadata={'source': '..../09
Document(id='0762048e-c63b-4f3c-bf0c-2ca289c6f959', metadata={'source': '..../09
Document(id='b05272b0-7f0d-4917-9672-2bc53a6d1b6f', metadata={'source': '..../09
```

- 다양성이 높은 더 많은 문서 검색해보기

- **k**: 반환할 문서 수 (**기본값 = 4**)
- **fetch_k**: MMR 알고리즘에 전달할 문서 수 (**기본값 = 20**)
- **lambda_mult**: MMR 결과의 다양성 조절 (**0~1, 기본값 = 0.5**)

```
retriever = db.as_retriever(
    search_type="mmr",
    search_kwargs={"k": 6,
                   "lambda_mult": 0.25,
                   "fetch_k": 10}
)
retriever.invoke("Word2Vec 에 대하여 알려줘")
```

검색 유형
반환할 문서 수
결과의 다양성 조절
MMR 알고리즘에 전달할 문서 수

- 셀 출력

```
[Document(id='9174a219-9bba-41d6-a08c-9c2fe712d441', metadata={'source': '..../09
```



```
Document(id='0762048e-c63b-4f3c-bf0c-2ca289c6f959', metadata={'source': '../09
Document(id='b05272b0-7f0d-4917-9672-2bc53a6d1b6f', metadata={'source': '../09
Document(id='1642794d-38de-4bde-a521-3b9606e768dd', metadata={'source': '../09
Document(id='1695c7d3-65e5-4b66-b9c4-a786e1a33787', metadata={'source': '../09
Document(id='1bc73306-0be6-4c34-8ce2-090d73fd04bb', metadata={'source': '../09
```

- MMR 알고리즘을 위해 더 많은 문서를 가져오되 **상위 2개만 반환**

```
retriever = db.as_retriever(
    search_type="mmr",
    search_kwargs={
        "k": 2,
        "fetch_k": 10})

retriever.invoke("Word2Vec 에 대하여 알려줘")
```

상위 2개만 반환

- 셀 출력 (상위 2개만 가져오도록 출력)

```
[Document(id='9174a219-9bba-41d6-a08c-9c2fe712d441', metadata={'source': '../09
Document(id='0762048e-c63b-4f3c-bf0c-2ca289c6f959', metadata={'source': '../09
```

- 특정 임계값 이상의 유사도를 가진 문서만 검색해보기

```
retriever = db.as_retriever(
    search_type="similarity_score_threshold",
    search_kwargs={"score_threshold": 0.8})

# 출력해보기
retriever.invoke("Word2Vec 에 대하여 알려줘")
```

검색 유형 설정하기
유사도 임계값 설정하기

- 셀 출력 (특정 임계값 이상의 **유사도** 를 가진 문서만 검색)

```
[Document(id='9174a219-9bba-41d6-a08c-9c2fe712d441', metadata={'source': '../09
```

- 가장 유사한 단일 문서만 검색해보기

```
retriever = db.as_retriever(search_kwargs={"k": 1})
```

유사도가 높은 문서 1개만

```
retriever.invoke("Word2Vec 에 대하여 알려줘")
```

- 셀 출력

```
[Document(id='9174a219-9bba-41d6-a08c-9c2fe712d441', metadata={'source': '../09
```

- 특정 메타데이터 필터 적용해보기

```
retriever = db.as_retriever(
    search_kwargs={
        "filter": {"source": "../09_VectorStore/data/finance-keywords.txt"},
        "k": 2}
)

retriever.invoke("ESG 에 대하여 알려줘")
```

- 셀 출력

```
[Document(id='135ddd04-f01c-4374-8888-aef61c932bad', metadata={'source': '../09
Document(id='cca43bd4-0754-426c-97d9-d16c02e8ffce', metadata={'source': '../09
```

-
- next: **멀티모달 이용해보기**
-