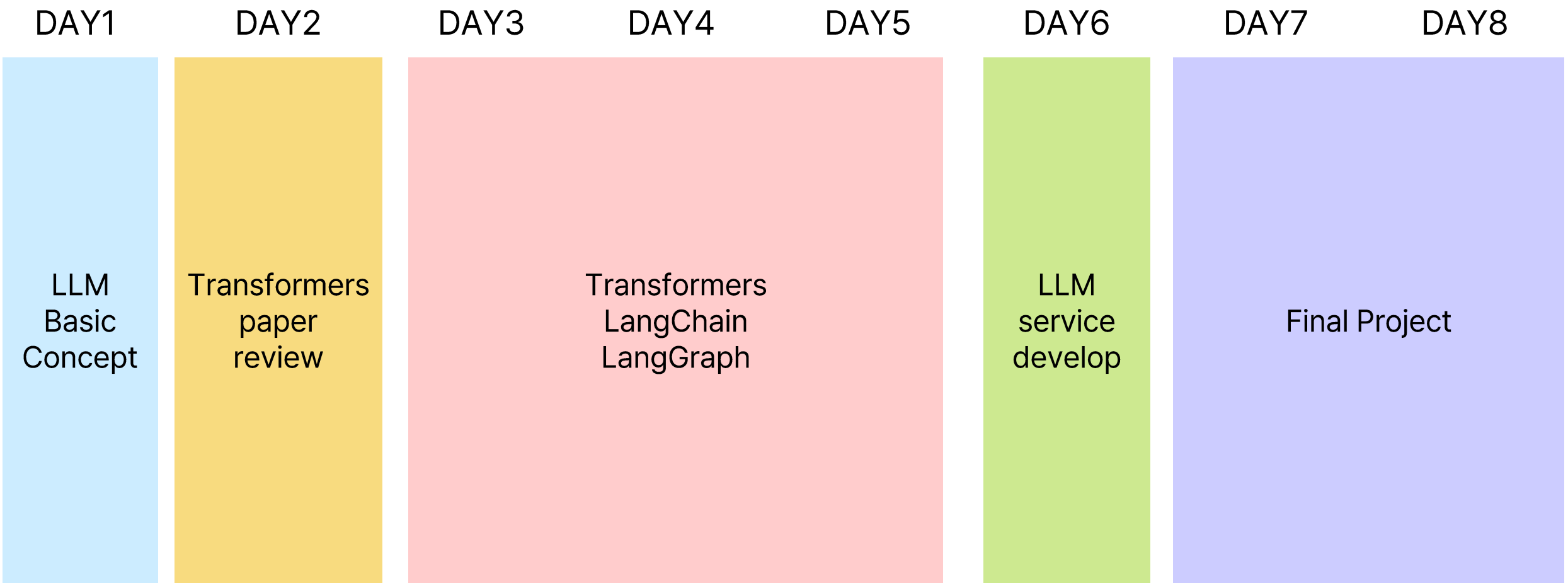
GEN AI 인텐시브 과정

강사 장철원

Section 0

코스소개



Section 0

목차

- ❑ vectorDB 개념
- ❑ chromaDB 기초

GEN AI 인텐시브 과정

Section 1. chromaDB

Section 1-1. vector DB의 개념

RDB vs NoSQL vs VectorDB

RDB

id	gender	age
1	male	25
2	female	42
3	male	63
4	female	21
5	male	32
6	female	38

NoSQL

```
{
    “_id”: 1,
    “gender”: “male”,
    “age”: 25,
    “contact”: {
        phone: “010-1234-5678”,
        email: “abcd@korea.com”
    },
    “hobby”: [“music”, “dance”]
},
{
    “_id”: 2,
    “gender”: “female”,
    “age”: 42,
    “contact”: {
        phone: “010-4321-8765”,
        email: “efgh@korea.com”
    },
    “hobby”: [“cook”, “book”]
},
```

VectorDB

임베딩 벡터 저장

[0.1, 0.2, 0.3, 0.6, 0.8]

[0.2, 0.6, 0.7, 0.7, 0.6]

[0.3, 0.3, 0.2, 0.5, 0.2]

[0.7, 0.5, 0.1, 0.2, 0.3]

[0.1, 0.2, 0.3, 0.6, 0.8]

[0.2, 0.6, 0.7, 0.7, 0.6]

[0.3, 0.3, 0.2, 0.5, 0.2]

[0.7, 0.5, 0.1, 0.2, 0.3]

Vector Database의 개념

- Vector Database란 데이터 항목들과 함께 벡터를 저장하는 데이터베이스를 의미
  - 이 때 벡터란 고정된 길이의 리스트를 의미하며, 리스트에는 여러 개의 숫자들이 포함됨
  - ex) [1, 2, 3, 4, 5]
  - LLM 분야에서는 워드 임베딩 방법으로 원문장을 벡터로 변환

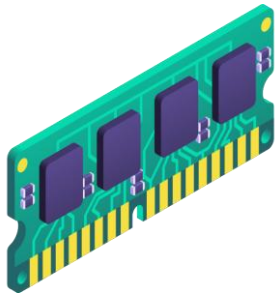
- 벡터는 수학적으로는 고차원 공간 속 데이터를 의미
- 각 차원은 데이터의 피처에 해당하며, 공간속에 나타나는 벡터의 위치는 해당 벡터의 특성을 나타냄.
- 문서, 이미지, 오디오 등 여러 종류의 데이터들을 벡터화 할 수 있음

- 데이터를 벡터 형태로 저장함으로서 유사도가 높은 데이터를 빠르게 찾을 수 있음
- RAG에 활용

Vector Database의 종류

이름	장점	단점
Chroma	오픈소스, 빠르고 가벼움, 로컬 환경에서도 쉽게 실행 가능, 파이썬 API 제공	대규모 데이터셋을 처리하기는 어려울 수도, 분산 시스템을 지원하지 않음
Faiss	Meta(페이스북)에서 개발, GPU 가속 지원으로 빠른 검색 성능, 대규모 데이터셋에서도 높은 성능, 파이썬 API 지원	단순 벡터 검색에 최적화되어 있고, 메타데이터 검색 기능 부족, 분산 시스템을 지원하지 않음, 사용법이 어려움.
Milvus	오픈소스, 대규모 벡터 검색 유용, 분산 시스템 기본 지원, 쿠버네티스 기반 확장 가능, CPU/GPU 최적화 지원	설정 및 운영이 복잡할 수 있음, 단순한 벡터 검색 용도로는 과도한 툴, 로컬 보다는 클러스터 환경에서 최적화됨
Qdrant	오픈소스, 러스트 기반이라 성능과 안정성이 뛰어남	인지도가 낮음, GPU 가속 지원 불가
Weaviate	오픈소스, RESTful API 및 GraphQL 지원, 확장성이 뛰어나며 쿠버네티스 지원	설정이 복잡하고 학습 곡선이 높음, 오버헤드가 커서 작은 프로젝트에는 과도한 툴.
Pinecone	클라우드 기반, 관리 부담이 적음, 빠른 검색 성능과 확장성	클라우드 서비스이므로 유료, 오픈소스가 아니므로 내부 구조 변경 불가, 로컬 환경 실행 불가
pgvector	PostgreSQL extension, PostgreSQL 기반이라 기존 데이터베이스와 쉽게 통합 가능, SQL활용 가능	대규모 벡터 검색시 성능 부족, 속도가 느림

FAISS vs ChromaDB vs Qdrant



FAISS	ChromaDB	Qdrant
인메모리 기반	경량 DB	서버형 DB
임베딩 실험용	로컬 테스트, 경량 앱	장기운영, 다중 사용자

GEN AI 인텐시브 과정

Section 1. chromaDB

Section 1-2. ChromaDB 설치

ChromaDB 설치

```
(base) C:\Users\stoic>conda activate py3_11_9

(py3_11_9) C:\Users\stoic>pip install chromadb
```

GEN AI 인텐시브 과정

Section 1. chromaDB

Section 1-3. ChromaDB 기초

라이브러리 불러오기 & 클라이언트 경로 설정

[1]:
```
import chromadb
```

chromaDB 라이브러리 불러오기

데이터 영구 저장

[2]:
```
client = chromadb.PersistentClient(path="./data")
```

chromaDB 클라이언트 생성

데이터를 path 경로에 저장

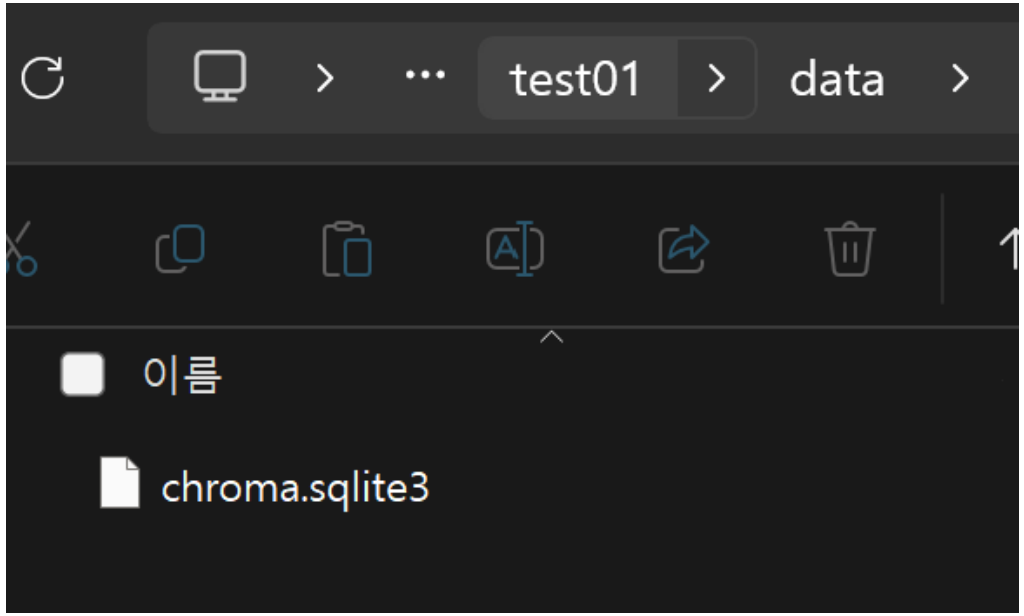
PersistentClient 대신 chromadb.Client() 사용하면 RAM에만 저장되고 프로그램 종료시 삭제

collection 생성

```
[3]: collection = client.get_or_create_collection("test_data01")
```

test_data01이라는 데이터 그룹을 불러오고, 만약 없다면 생성
ChromaDB는 데이터를 collection 단위로 관리함

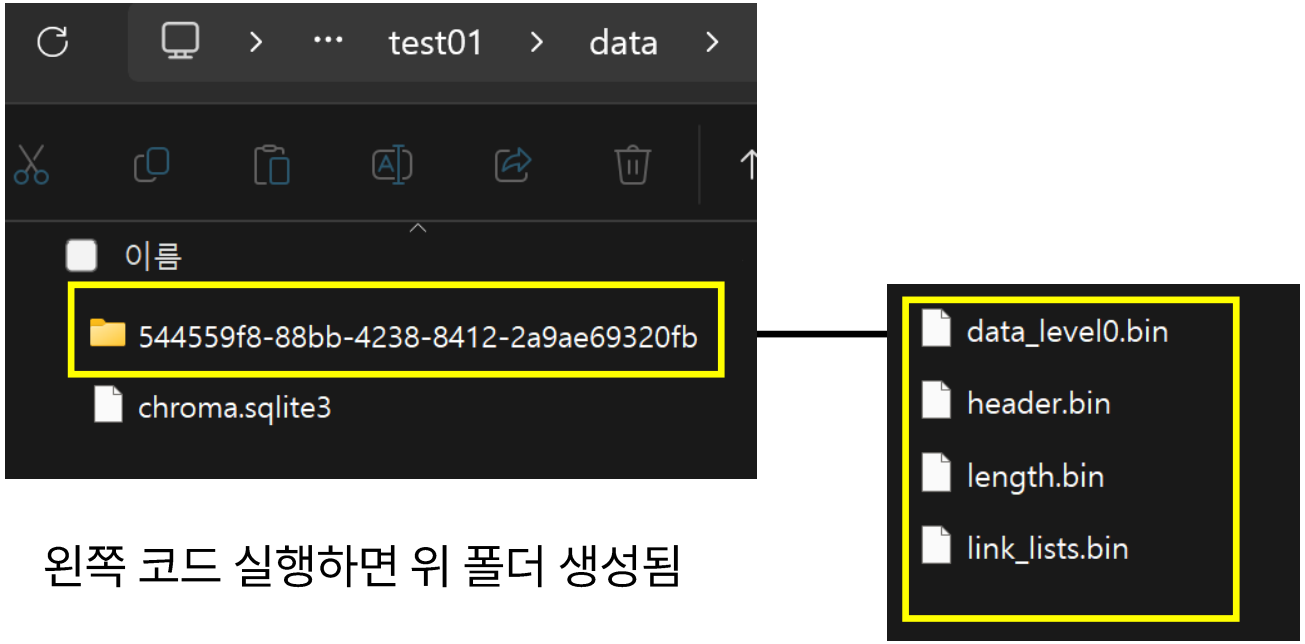
위 코드를 실행하고 test01/data 폴더를 확인하면
sqlite3 파일을 확인할 수 있음



데이터 추가하기(insert)

```
collection.add(
    ids = ["doc1", "doc2"],
    embeddings = [
        [0.1, 0.2, 0.3],
        [0.4, 0.5, 0.6]
    ],

    metadatas = [
        {"text": "안녕하세요"},
        {"text": "날씨가 좋네요"}
    ]
)
```



왼쪽 코드 실행하면 위 폴더 생성됨

데이터 추가하기(insert)

반드시 해당 변수명 사용해야함
ids
embeddings
metadatas

```
collection.add(
    ids = ["doc1", "doc2"],
    embeddings = [
        [0.1, 0.2, 0.3],
        [0.4, 0.5, 0.6]
    ],

    metadatas = [
        {"text": "안녕하세요"},
        {"text": "날씨가 좋네요"}
    ]
)
```

RDB에서 PRIMARY KEY 에 해당 index와 비슷해보이지만 RDB와 달리 정수(int)를 사용하지 않는 이유는 문서 ID를 사람이 이해하기 쉽게 만들기 위함

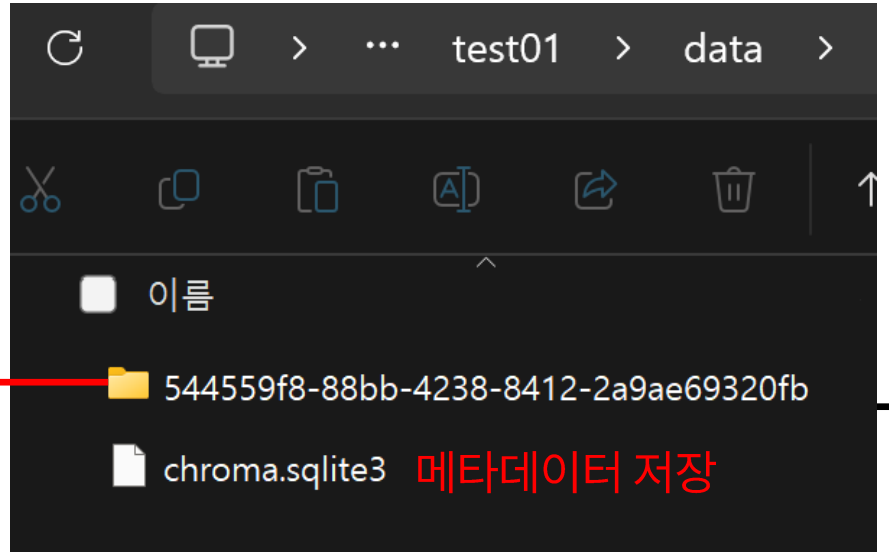
원문장의 임베딩 벡터

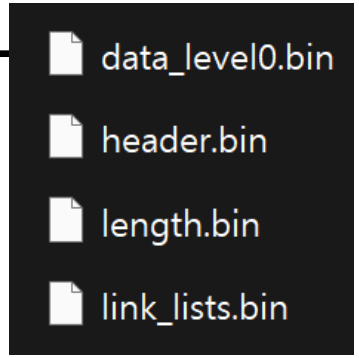
원문장을 메타데이터 형태로 저장함으로써 추후 원래 문장 확인 가능

데이터 추가하기(insert)

collection(test_data01) 생성할 때 자동으로 부여된 UUID

해당 폴더는 해당 컬렉션에 저장된 벡터 데이터를 위한 저장소





엠베딩 벡터 저장

메타데이터 저장

벡터 길이 저장

벡터 검색 인덱스

ChromaDB 기초

[5]:

```
query = [[0.1, 0.2, 0.3]]
```

쿼리 임베딩 벡터

[6]:

```
result = collection.query(query_embeddings = query, n_results=1)
print(result)
```

collection에서 가장 유사한 벡터 찾기

임베딩 벡터 나오게 하고 싶으면? 다음페이지 참고

```
{'ids': [['doc1']], 'embeddings': None, 'documents': [[None]], 'uris': None, 'data': None, 'm
etadatas': [[{'text': '안녕하세요'}]], 'distances': [[0.0]], 'included': [<IncludeEnum.distanc
es: 'distances'>, <IncludeEnum.documents: 'documents'>, <IncludeEnum.metadatas: 'metadata
s'>]}
```

[7]:

```
result_text = result['metadatas'][0][0]['text']
print(result_text)
```

text 결과 확인

```
안녕하세요
```

임베딩 벡터도 불러오고 싶은 경우

[23]:
```
result2 = collection.query(query_embeddings = query, n_results=1, include=["embeddings", "metadatas"])
result2
```

[23]:
```
{'ids': [['doc1']],
 'embeddings': [array([[0.1       , 0.2       , 0.30000001]])],
 'documents': None,
 'uris': None,
 'data': None,
 'metadatas': [[{'text': '안녕하세요'}]],
 'distances': None,
 'included': [<IncludeEnum.embeddings: 'embeddings'>,
  <IncludeEnum.metadatas: 'metadatas'>]}
```

ChromaDB 기초

[11]:
```
all_data = collection.get()
all_data
```
collection 데이터 내부 전체 데이터 확인

[11]:
```
{'ids': ['doc1', 'doc2'],
 'embeddings': None,
 'documents': [None, None],
 'uris': None,
 'data': None,
 'metadatas': [{'text': '안녕하세요'}, {'text': '날씨가 좋네요'}],
 'included': [<IncludeEnum.documents: 'documents'>,
  <IncludeEnum.metadatas: 'metadatas'>]}
```

[10]:
```
collection.get(ids=["doc1"])
```
doc1만 확인

[10]:
```
{'ids': ['doc1'],
 'embeddings': None,
 'documents': [None],
 'uris': None,
 'data': None,
 'metadatas': [{'text': '안녕하세요'}],
 'included': [<IncludeEnum.documents: 'documents'>,
  <IncludeEnum.metadatas: 'metadatas'>]}
```

Q RDB처럼 상위 데이터만 확인하는 SELECT * FROM table LIMIT 1000; 같은 명령어 없나요?

A ChromaDB는 데이터를 순서대로 가져오는 것보다 유사한 데이터를 빠르게 찾는것에 집중

현재는 RDB에서의 LIMIT과 같은 명령어는 없으나 향후 추가 될지도..?

데이터 추가

[12]:
```
collection.add(
    ids = ["doc3"],
    embeddings = [
        [0.7, 0.8, 0.9]
    ],

    metadatas = [
        {"text": "안녕히 가세요"}
    ]
)
```

[13]:
```
all_data = collection.get()
all_data
```

[13]:
```
{'ids': ['doc1', 'doc2', 'doc3'],
 'embeddings': None,
 'documents': [None, None, None],
 'uris': None,
 'data': None,
 'metadatas': [{'text': '안녕하세요'}, {'text': '날씨가 좋네요'}, {'text': '안녕히 가세요'}],
 'included': [<IncludeEnum.documents: 'documents'>,
  <IncludeEnum.metadatas: 'metadatas'>]}
```

데이터 삭제

[14]:
```
collection.delete(ids=["doc3"])
```

[24]:
```
all_data = collection.get(include=["embeddings", "metadatas"])
all_data
```

"embedings", "metadatas" 정보 함께 출력

[24]:
```
{'ids': ['doc1', 'doc2'],
 'embeddings': array([[0.1       , 0.2       , 0.30000001],
        [0.40000001, 0.5       , 0.60000002]]),
 'documents': None,
 'uris': None,
 'data': None,
 'metadatas': [{'text': '안녕하세요'}, {'text': '날씨가 좋네요'}],
 'included': [<IncludeEnum.embeddings: 'embeddings'>,
  <IncludeEnum.metadatas: 'metadatas'>]}
```

parquet 파일로 저장

[25]:
```python
import pandas as pd
```

[30]:
```python
df = pd.DataFrame({
    "ids": all_data["ids"],
    "embeddings": [embedding for embedding in all_data["embeddings"]],
    "text": [meta["text"] for meta in all_data["metadatas"]]
})
```

[31]:
```python
df.head()
```

[31]:

	ids	embeddings	text
0	doc1	[0.10000000149011612, 0.20000000298023224, 0.3...	안녕하세요
1	doc2	[0.4000000059604645, 0.5, 0.6000000238418579]	날씨가 좋네요

parquet 파일로 저장 & parquet 파일 불러오기

```
[33]: df.to_parquet("./data/test_data01.parquet", index=False)
```

```
[35]: df2 = pd.read_parquet("./data/test_data01.parquet")
```

```
[36]: df2.head()
```

[36]:

	ids	embeddings	text
0	doc1	[0.10000000149011612, 0.20000000298023224, 0.3...	안녕하세요
1	doc2	[0.4000000059604645, 0.5, 0.6000000238418579]	날씨가 좋네요

GEN AI 인텐시브 과정

Section 1. chromaDB

Section 1-4. 기존 데이터 임베딩 후 chromaDB 삽입

Section

기존데이터임베딩후chromaDB 삽입

라이브러리 불러오기

```
import chromadb
import pandas as pd
from sentence_transformers import SentenceTransformer
```

```
df = pd.read_csv('./test_file.csv', encoding='utf8')
```

```
df
```

	sentence
0	안녕하세요.
1	누구세요?
2	안녕히 가세요.

```
sc_list = df['sentence'].tolist()
```

```
sc_list
```

```
['안녕하세요.', '누구세요?', '안녕히 가세요.']
```

Section

기존데이터임베딩후chromaDB 삽입

문장 임베딩

```
model = SentenceTransformer("sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2")
```

```
ids_list = []
ebd_list = []
text_list = []
ith = 0
for sc in sc_list:
    idx = f"doc{ith+1}"
    ids_list.append(idx)
    ith += 1

    text_dic = {"text": sc}
    text_list.append(text_dic)

    embedding = model.encode(sc).tolist()
    ebd_list.append(embedding)
```

```
ids_list
```

```
['doc1', 'doc2', 'doc3']
```

```
print(ebd_list)
```

```
[[0.12071584910154343, 0.16374550759792328, 0.27759042382240295, 0.28737086057662964, -0.1
5490840375423431, -0.3444209098815918, 0.1318168193101883, 0.06661806255578995, -0.5285354
9343109, 0.1576467752456665, 0.28771552443504333, -0.2034401297569275, -0.2240461409091949
607, -0.11048044264316559, 0.07415991276502609, 0.044156335294246674, -0.2010980248451233,
```

Section

기존데이터임베딩후chromaDB 삽입

chromaDB 설정

```
print(text_list)
```

```
[{'text': '안녕하세요.'}, {'text': '누구세요?'}, {'text': '안녕히 가세요.'}]
```

```
client = chromadb.PersistentClient(path="./data")
collection = client.get_or_create_collection("test_data02")
```

```
collection.add(
    ids = ids_list,
    embeddings = ebd_list,
    metadatas = text_list
)
```

Section

기존데이터임베딩후chromaDB 삽입

결과 확인

```
query_sc = "반갑습니다"
```

```
query_ebd = [ model.encode(query_sc).tolist() ]
```

```
result = collection.query(query_embeddings = query_ebd, n_results=1)
result
```

```
{'ids': [['doc1']],
 'embeddings': None,
 'documents': [[None]],
 'uris': None,
 'data': None,
 'metadatas': [[{'text': '안녕하세요.'}]],
 'distances': [[11.225251501029351]],
 'included': [<IncludeEnum.distances: 'distances'>,
  <IncludeEnum.documents: 'documents'>,
  <IncludeEnum.metadatas: 'metadatas'>]}
```

```
result_text = result['metadatas'][0][0]['text']
print(result_text)
```

```
안녕하세요.
```

ChromaDB

Section 2. Chunking

Section 2-1. Chunking

Section

Chunking

라이브러리 불러오기 & 모델 설정

```
from langchain_chroma import Chroma
from langchain.schema import Document
from langchain_huggingface import HuggingFaceEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter
```

```
with open("./data/doc.txt", "r", encoding="utf-8") as f:
    raw_text = f.read()
```

```
raw_text
```

```
'Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다. PM 장비의 필터는 주 1회 정기적으로 교체해야 합니다. 웨이퍼 투입 전 챔버 내부의 온도 안정화가 필요합니다. 불량률이 2%를 초과할 경우 원인 분석 보고서를 제출해야 합니다. 클린룸 입장 전에는 반드시 정전기 방지복을 착용해야 합니다. 포토 공정 시 PR 코팅 두께는 1.5μm 이상 유지해야 합니다. 검사 장비는 매일 초기화 후 기능 점검을 실시합니다. 주간 생산 계획은 매주 월요일 오전 9시에 확정됩니다. X선 검사 장비는 비정상 신호 발생 시 즉시 사용 중지합니다. 로더 설비의 진공 펌프는 월 1회 이상 윤활 상태를 점검합니다. 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다. 동일 Lot 내에서 불량이 5개 이상 발생하면 전수 검사 실시합니다. 물류 이송 로봇은 경로 장애 감지 시 자동 우회합니다. 이온 주입 공정 후 열처리 시간은 최소 30분 이상 확보합니다. 클린룸 청소는 일 2회, 장비 청소는 주 1회 이상 실시합니다. 신입 엔지니어는 공정별 교육을 모두 이수한 후 장비 조작이 가능합니다. 공정 레시피 변경 시 Change History 문서 작성을 필수로 합니다. 수율 개선안은 월간 품질 회의에서 발표되어야 합니다. Mask alignment 오류가 0.2μm를 초과하면 장비 점검을 실시합니다. MES 시스템에 기록된 공정 로그는 6개월간 보관됩니다. 웨이퍼 저장 캐리어는 주기적으로 오염도를 측정해야 합니다. 장비 재가동 시 Warm-up 절차를 반드시 이행합니다. Test wafer는 생산 wafer 투입 전에 반드시 시뮬레이션합니다. 제조 Lot ID는 공정마다 바코드로 자동 추적됩니다. 소자 특성 분석 결과는 전자문서 시스템에 등록합니다. 야간조 작업자는 작업 시작 전 점검 체크리스트를 반드시 확인합니다. 공정 조건이 기준에서 벗어난 경우 자동 알람이 발생합니다. 생산 중단 요청은 생산 관리팀 승인 후 진행할 수 있습니다. 사내 기술 포럼 자료는 R&D 인트라넷에 주기적으로 업데이트됩니다. 정전 발생 시 UPS 시스템으로 30분간 운영이 가능합니다.'
```

Chunking

```
# 2. 텍스트 청킹
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=300,
    chunk_overlap=50,
    separators=["\n", ".", " "]
)
chunks = text_splitter.create_documents([raw_text])
```

이때 300은 토큰 수가 아니라 character 수

chunk_overlap -> 문맥 이해

```
chunks
```

```
[Document(metadata={}, page_content='Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다. PM 장비의 필터는 주 1회 정기적으로 교체해야 합니다. 웨이퍼 투입 전 챔버 내부의 온도 안정화가 필요합니다. 불량률이 2%를 초과할 경우 원인 분석 보고서를 제출해야 합니다. 클린룸 입장 전에는 반드시 정전기 방지복을 착용해야 합니다. 포토 공정 시 PR 코팅 두께는 1.5μm 이상 유지해야 합니다. 검사 장비는 매일 초기화 후 기능 점검을 실시합니다. 주간 생산 계획은 매주 월요일 오전 9시에 확정됩니다'),
 Document(metadata={}, page_content='. 주간 생산 계획은 매주 월요일 오전 9시에 확정됩니다. X선 검사 장비는 비정상 신호 발생 시 즉시 사용 중지합니다. 로더 설비의 진공 펌프는 월 1회 이상 윤활 상태를 점검합니다. 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다. 동일 Lot 내에서 불량이 5개 이상 발생하면 전수 검사 실시합니다. 물류 이송 로봇은 경로 장애 감지 시 자동 우회합니다. 이온 주입 공정 후 열처리 시간은 최소 30분 이상 확보합니다. 클린룸 청소는 일 2회, 장비 청소는 주 1회 이상 실시합니다'),
 Document(metadata={}, page_content='. 클린룸 청소는 일 2회, 장비 청소는 주 1회 이상 실시합니다. 신입 엔지니어는 공정별 교육을 모두 이수한 후 장비 조작이 가능합니다. 공정 레시피 변경 시 Change History 문서 작성을 필수로 합니다. 수율 개선안은 월간 품질 회의에서 발표되어야 합니다. Mask alignment 오류가 0.2μm를 초과하면 장비 점검을 실시합니다. MES 시스템에 기록된 공정 로그는 6개월간 보관됩니다. 웨이퍼 저장 캐리어는 주기적으로 오염도를 측정해야 합니다. 장비 재가동 시 Warm-up 절차를 반드시 이행합니다'),
 Document(metadata={}, page_content='. 장비 재가동 시 Warm-up 절차를 반드시 이행합니다. Test wafer는 생산 wafer 투입 전에 반드시 시뮬레이션합니다. 제조 Lot ID는 공정마다 바코드로 자동 추적됩니다. 소자 특성 분석 결과는 전자문서 시스템에 등록합니다. 야간조 작업자는 작업 시작 전 점검 체크리스트를 반드시 확인합니다. 공정 조건이 기준에서 벗어난 경우 자동 알람이 발생합니다. 생산 중단 요청은 생산 관리팀 승인 후 진행할 수 있습니다. 사내 기술 포럼 자료는 R&D 인트라넷에 주기적으로 업데이트됩니다'),
 Document(metadata={}, page_content='. 사내 기술 포럼 자료는 R&D 인트라넷에 주기적으로 업데이트됩니다. 정전 발생 시 UPS 시스템으로 30분간 운영이 가능합니다.')]
```

데이터 저장

```
# 3. 임베딩 모델 로딩
embedding_model = HuggingFaceEmbeddings(
    model_name="sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2"
)
```

```
# 4. ChromaDB에 저장
vectorstore = Chroma.from_documents(
    documents=chunks,
    embedding=embedding_model,
    persist_directory="./chromaDB_chunked"
)
```

Section
Chunking

데이터 불러오기

```
# 5.저장된 Chroma 불러오기
vectorstore = Chroma(
    persist_directory="./chromaDB_chunked",
    embedding_function=embedding_model
)
```

```
#  6.벡터DB에서 전체 문서 객체 가져오기
collection = vectorstore._collection
docs = collection.get()  # 모든 document ID, documents, metadatas, embeddings 가져옴
```

```
docs
```

```
{'ids': ['4f5f5715-0070-4f44-b1e2-6245772120cd',
  '8ed731df-63f9-4375-9ed9-ec3abce95a07',
  '0a2e0520-1f71-4736-b6c0-aaa6b9a23ab7',
  'c60e36e3-dd12-48c1-90e5-9fe6a6a147af',
  '4b70ee46-2e56-4531-9a70-4966f8c366a4'],
 'embeddings': None,
 'documents': ['Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다. PM 장비의 필터는 주 1회
화가 필요합니다. 불량률이 2%를 초과할 경우 원인 분석 보고서를 제출해야 합니다. 클린룸 입장 전에는 
1.5μm 이상 유지해야 합니다. 검사 장비는 매일 초기화 후 기능 점검을 실시합니다. 주간 생산 계획은 매
  '. 주간 생산 계획은 매주 월요일 오전 9시에 확정됩니다. X선 검사 장비는 비정상 신호 발생 시 즉시
점검합니다. 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다. 동일 Lot 내에서 불량이 5개 이상 발생
우회합니다. 이온 주입 공정 후 열처리 시간은 최소 30분 이상 확보합니다. 클린룸 청소는 일 2회, 장비
  '. 클린룸 청소는 일 2회, 장비 청소는 주 1회 이상 실시합니다. 신입 엔지니어는 공정별 교육을 모두
ry 문서 작성을 필수로 합니다. 수율 개선안은 월간 품질 회의에서 발표되어야 합니다. Mask alignment
된 공정 로그는 6개월간 보관됩니다. 웨이퍼 저장 캐리어는 주기적으로 오염도를 측정해야 합니다. 장비 
  '. 장비 재가동 시 Warm-up 절차를 반드시 이행합니다. Test wafer는 생산 wafer 투입 전에 반드시 
소자 특성 분석 결과는 전자문서 시스템에 등록합니다. 야간조 작업자는 작업 시작 전 점검 체크리스트를
생합니다. 생산 중단 요청은 생산 관리팀 승인 후 진행할 수 있습니다. 사내 기술 포럼 자료는 R&D 인트라
  '. 사내 기술 포럼 자료는 R&D 인트라넷에 주기적으로 업데이트됩니다. 정전 발생 시 UPS 시스템으로 3
 'uris': None,
 'data': None,
 'metadatas': [None, None, None, None, None],
 'included': [<IncludeEnum.documents: 'documents'>,
  <IncludeEnum.metadatas: 'metadatas'>]}
```

Section

Chunking

데이터 일부만 불러오기

```python
# 7. 최대 5개 문서만 가져오기
docs = vectorstore._collection.get(limit=5)

for i in range(len(docs["documents"])):
    print(f"Document {i+1}: {docs['documents'][i]}")
```

```
Document 1: Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다. PM 장비의 필터는 주 1회 정기적으로 교체해야 합니다. 웨이퍼 투입 전 챔버 내부의 온도 안정화가 필요합니다. 불량률이 2%를 초과할 경우 원인 분석 보고서를 제출해야 합니다. 클린룸 입장 전에는 반드시 정전기 방지복을 착용해야 합니다. 포토 공정 시 PR 코팅 두께는 1.5μm 이상 유지해야 합니다. 검사 장비는 매일 초기화 후 기능 점검을 실시합니다. 주간 생산 계획은 매주 월요일 오전 9시에 확정됩니다
Document 2: . 주간 생산 계획은 매주 월요일 오전 9시에 확정됩니다. X선 검사 장비는 비정상 신호 발생 시 즉시 사용 중지합니다. 로더 설비의 진공 펌프는 월 1회 이상 윤활 상태를 점검합니다. 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다. 동일 Lot 내에서 불량이 5개 이상 발생하면 전수 검사 실시합니다. 물류 이송 로봇은 경로 장애 감지 시 자동 우회합니다. 이온 주입 공정 후 열처리 시간은 최소 30분 이상 확보합니다. 클린룸 청소는 일 2회, 장비 청소는 주 1회 이상 실시합니다
Document 3: . 클린룸 청소는 일 2회, 장비 청소는 주 1회 이상 실시합니다. 신입 엔지니어는 공정별 교육을 모두 이수한 후 장비 조작이 가능합니다. 공정 레시피 변경 시 Change History 문서 작성을 필수로 합니다. 수율 개선안은 월간 품질 회의에서 발표되어야 합니다. Mask alignment 오류가 0.2μm를 초과하면 장비 점검을 실시합니다. MES 시스템에 기록된 공정 로그는 6개월간 보관됩니다. 웨이퍼 저장 캐리어는 주기적으로 오염도를 측정해야 합니다. 장비 재가동 시 Warm-up 절차를 반드시 이행합니다
Document 4: . 장비 재가동 시 Warm-up 절차를 반드시 이행합니다. Test wafer는 생산 wafer 투입 전에 반드시 시뮬레이션합니다. 제조 Lot ID는 공정마다 바코드로 자동 추적됩니다. 소자 특성 분석 결과는 전자문서 시스템에 등록합니다. 야간조 작업자는 작업 시작 전 점검 체크리스트를 반드시 확인합니다. 공정 조건이 기준에서 벗어난 경우 자동 알람이 발생합니다. 생산 중단 요청은 생산 관리팀 승인 후 진행할 수 있습니다. 사내 기술 포럼 자료는 R&D 인트라넷에 주기적으로 업데이트됩니다
Document 5: . 사내 기술 포럼 자료는 R&D 인트라넷에 주기적으로 업데이트됩니다. 정전 발생 시 UPS 시스템으로 30분간 운영이 가능합니다.
```

임베딩 벡터 확인

```python
# 8. 임베딩 벡터 출력
docs = vectorstore._collection.get(limit=3, include=["embeddings", "documents"])

for i, embedding in enumerate(docs["embeddings"]):
    print(f"\n문서 {i+1} 임베딩 벡터 (길이: {len(embedding)}):")
    print(embedding[:10], "...")
```

```
문서 1 임베딩 벡터 (길이: 384):
[-0.09894919  0.33043957 -0.03778113  0.06803279  0.17861922  0.03154641
 -0.12783037 -0.10195233 -0.07134267 -0.08900272] ...

문서 2 임베딩 벡터 (길이: 384):
[-0.0974468   0.13622653  0.02063832 -0.13876298  0.04451955 -0.05115731
  0.0061339   0.03803405  0.07322869 -0.1191432 ] ...

문서 3 임베딩 벡터 (길이: 384):
[-0.15061809  0.21071179  0.02546037 -0.16733345  0.02144328  0.10268474
 -0.22503944 -0.02251757 -0.08299647 -0.07703963] ...
```

감사합니다.

Q & A