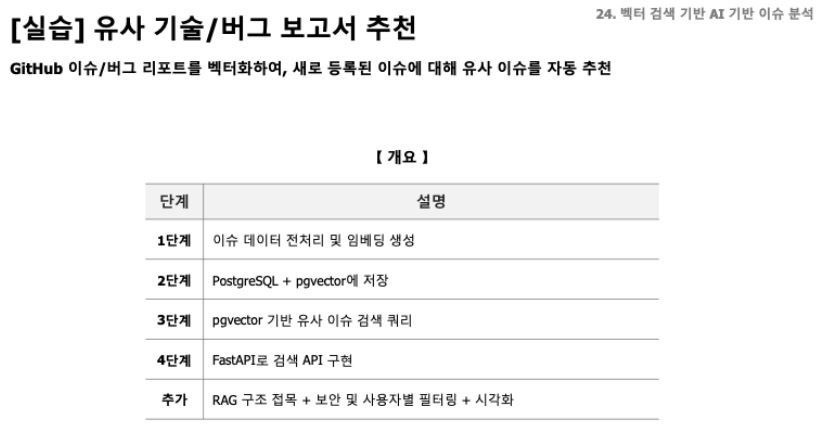
DBMS 및 SQL 활용 #5

▼ 문제



▼ 실습 개요

- 실습 목적
  - 텍스트 데이터(GitHub Issues)를 임베딩 생성하여 PostgreSQL + pgvector에 저장하고, 이를 기반으로 유사 이슈 검색을 수행하며, 시각화를 통해 데이터 구조를 이해하고 접근 제어를 적용한 뒤, RAG 구조를 접목해 자동 요약 구현
- 실습 설계
  - 임베딩 생성: SentenceTransformer("all-MiniLM-L6-v2")
  - 유사 이슈 검색: 코사인 유사도 + REST API 형태로 검색 기능 제공
  - 시각화: PCA + KMeans
  - 접근 제어: RLS
  - 자동 요약 구현: RAG + "gpt-4o-mini"

▼ 전체 코드

```
# sc2.ipynb
import pandas as pd
from sentence_transformers import SentenceTransformer
import os
from dotenv import load_dotenv

os.chdir("/Users/yshmbid/Documents/home/github/SQL")  # set path
load_dotenv()

# 1. CSV 불러오기 및 임베딩 생성
# 데이터 불러오기
df = pd.read_csv("github_issues_large.csv")  # issueId, title, description, tags 포함

# 임베딩 모델 로드
model = SentenceTransformer("all-MiniLM-L6-v2")

# title + description을 합쳐서 임베딩 생성
df["text"] = df["title"].astype(str) + " " + df["description"].astype(str)
```

```
df["embedding"] = df["text"].apply(lambda x: model.encode(x).tolist())
df
```

```
# 2. PostgreSQL + pgvector 테이블 생성
-- 확장 설치
CREATE EXTENSION IF NOT EXISTS vector;

-- 테이블 생성
DROP TABLE IF EXISTS issues;
CREATE TABLE issues (
    id SERIAL PRIMARY KEY,
    title TEXT,
    description TEXT,
    embedding vector(384)
);
```

```
# sc2.ipynb
# 3. Python에서 데이터 적재
import psycopg2
import json

conn = psycopg2.connect(
    host="localhost",
    port=5432,
    database="postgres",
    user="postgres",
    password=os.getenv("PG_PASSWORD"),
)
cur = conn.cursor()

for _, row in df.iterrows():
    cur.execute(
        "INSERT INTO issues (title, description, embedding, user_id) VALUES (%s, %s, %s, %s)",
        (row["title"], row["description"], row["embedding"], "alice")
    )

conn.commit()
cur.close()
conn.close()

# 4. pgvector 기반 유사 이슈 검색
import psycopg2
import os
import json

def search_similar(query, topk=5):
    conn = psycopg2.connect(
        host="localhost",
        port=5432,
        database="postgres",
        user="postgres",
        password=os.getenv("PG_PASSWORD"),
    )
    cur = conn.cursor()
```

```
    # 쿼리 임베딩
    q_emb = model.encode(query).tolist()
    q_emb_str = "[" + ",".join(map(str, q_emb)) + "]"

    # pgvector 검색
    cur.execute(
        f"""
        SELECT id, title, description
        FROM issues
        ORDER BY embedding ⇔ %s::vector
        LIMIT %s;
        """,
        (q_emb_str, topk),
    )
    results = cur.fetchall()
    cur.close()
    conn.close()
    return results

print(search_similar("memory leak on login"))

# 5. FastAPI 검색 API
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Query(BaseModel):
    text: str
    topk: int = 5

@app.post("/search")
def search(query: Query):
    results = search_similar(query.text, query.topk)
    return {"query": query.text, "results": results}

# 6. 시각화 (PCA + KMeans)
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# 임베딩 행렬
X = list(df["embedding"])

# PCA 2D
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# KMeans
kmeans = KMeans(n_clusters=5, random_state=42).fit(X_reduced)
labels = kmeans.labels_

# 시각화
plt.figure(figsize=(8,6))
```

```
plt.scatter(X_reduced[:,0], X_reduced[:,1], c=labels, cmap="tab10")
plt.title("Issue Clusters (PCA + KMeans)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()

# 7. 시각화2: 대표 키워드(Top-N 단어) 추출 후 라벨링
import re
from collections import Counter
import numpy as np

# 텍스트 전처리 함수
def tokenize(text):
    text = text.lower()
    text = re.sub(r"[^a-z0-9\s]", "", text)  # 알파벳/숫자만
    return text.split()

# 불용어 정의 (원하면 nltk stopwords로 확장 가능)
stopwords = set(["the", "and", "is", "to", "in", "of", "for", "on", "a", "an", "with", "by", "from", "at", "this", "th
at"])

# 각 클러스터별 텍스트 모으기
cluster_texts = {i: [] for i in range(5)}  # n_clusters=5
for idx, label in enumerate(labels):
    tokens = tokenize(df.iloc[idx]["title"] + " " + df.iloc[idx]["description"])
    tokens = [t for t in tokens if t not in stopwords and len(t) > 2]
    cluster_texts[label].extend(tokens)

# 클러스터별 Top-N 키워드 추출
top_keywords = {}
for label, words in cluster_texts.items():
    counter = Counter(words)
    top_keywords[label] = [w for w, _ in counter.most_common(3)]  # 상위 3개

print("클러스터 대표 키워드:", top_keywords)

# 클러스터 중심 계산
centers = kmeans.cluster_centers_

# 시각화 + 라벨
plt.figure(figsize=(8,6))
plt.scatter(X_reduced[:,0], X_reduced[:,1], c=labels, cmap="tab10")

for i, center in enumerate(centers):
    keywords = ", ".join(top_keywords[i])
    plt.text(center[0], center[1], keywords, fontsize=10, weight="bold", ha="center")

plt.title("Issue Clusters (PCA + KMeans + Keywords)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

```
-- 8. PostgreSQL에서 테이블 구조와 보안 정책을 세팅
-- 사용자 컬럼 추가
```

```
ALTER TABLE issues ADD COLUMN user_id TEXT;

-- RLS 활성화
ALTER TABLE issues ENABLE ROW LEVEL SECURITY;

-- Row-Level Security 정책
CREATE POLICY user_issues_policy
ON issues
FOR SELECT
USING (user_id = current_user);
```

```
# app.py
# 9. RAG 구조 접목
import psycopg2
from fastapi import FastAPI
from pydantic import BaseModel
from openai import OpenAI
from sentence_transformers import SentenceTransformer
import os
from dotenv import load_dotenv

os.chdir("/Users/yshmbid/Documents/home/github/SQL")
load_dotenv()

# FastAPI 앱
app = FastAPI()

# OpenAI 클라이언트
client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# SentenceTransformer 임베딩 모델 로드
model = SentenceTransformer("all-MiniLM-L6-v2")

# 요청 모델
class Query(BaseModel):
    text: str
    topk: int = 5
    user_id: str | None = None  # 사용자 ID 필드 (옵션)

# pgvector 기반 검색 함수
def search_similar(query, topk=5, user_id=None):
    conn = psycopg2.connect(
        host="localhost",
        port=5432,
        database="postgres",
        user="postgres",
        password=os.getenv("PG_PASSWORD"),
    )
    cur = conn.cursor()

    q_emb = model.encode(query).tolist()
    q_emb_str = "[" + ",".join(map(str, q_emb)) + "]"

    if user_id:
```

```
        cur.execute(
            """
            SELECT id, title, description
            FROM issues
            WHERE user_id = %s
            ORDER BY embedding <=> %s::vector
            LIMIT %s;
            """,
            (user_id, q_emb_str, topk),
        )
    else:
        cur.execute(
            """
            SELECT id, title, description
            FROM issues
            ORDER BY embedding <=> %s::vector
            LIMIT %s;
            """,
            (q_emb_str, topk),
        )

    results = cur.fetchall()
    cur.close()
    conn.close()
    return results

# RAG API
@app.post("/search_rag")
def search_rag(query: Query):
    # 1) pgvector 검색
    results = search_similar(query.text, query.topk, query.user_id)

    # 2) 프롬프트 구성
    context_text = "\n".join([f"- {r[1]}: {r[2]}" for r in results])
    prompt = f"""
    새로운 이슈: "{query.text}"
    아래는 DB에서 검색된 유사 이슈들입니다:
    {context_text}

    위 유사 이슈들을 참고해서,
    1) 공통된 문제 요약
    2) 잠재적인 원인
    3) 해결 방향 (가능하다면)
    을 간단히 정리해줘.
    """

    # 3) GPT 호출
    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {"role": "system", "content": "You are a helpful assistant for issue tracking."},
            {"role": "user", "content": prompt}
        ],
    )
```

```
    summary = response.choices[0].message.content

    return {
        "query": query.text,
        "results": results,
        "summary": summary
    }
```

```
# Terminal 1
$ uvicorn app:app --reload --host 0.0.0.0 --port 8000

# Terminal 2
curl -X POST "http://127.0.0.1:8000/search_rag" \
  -H "Content-Type: application/json" \
  -d '{"text": "memory leak on login", "topk": 5, "user_id": "alice"}'
```

▼ 코드 - 4. pgvector 기반 유사 이슈 검색

```
# 4. pgvector 기반 유사 이슈 검색
import psycopg2
import os
import json

def search_similar(query, topk=5):
    conn = psycopg2.connect(
        host="localhost",
        port=5432,
        database="postgres",
        user="postgres",
        password=os.getenv("PG_PASSWORD"),
    )
    cur = conn.cursor()

    # 쿼리 임베딩
    q_emb = model.encode(query).tolist()
    q_emb_str = "[" + ",".join(map(str, q_emb)) + "]"

    # pgvector 검색
    cur.execute(
        f"""
        SELECT id, title, description
        FROM issues
        ORDER BY embedding ⇔ %s::vector
        LIMIT %s;
        """,
        (q_emb_str, topk),
    )
    results = cur.fetchall()
    cur.close()
    conn.close()
    return results

print(search_similar("memory leak on login"))
```

- search_similar() 목적
  - GitHub 이슈 자동 추천
    - 새로 등록된 이슈에 대해 과거 유사 이슈를 추천
  - 새로운 이슈가 들어오면 기존 데이터베이스(issues 테이블)에 저장된 이슈들 중에서 내용이 비슷한 이슈들을 자동으로 검색.
- conn = psycopg2.connect()
  - PostgreSQL DB(postgres)에 연결하고 issues 테이블 준비
- q_emb = model.encode(query).tolist()
  - SentenceTransformer 모델로 쿼리(입력된 문장)을 384차원 임베딩 벡터로 변환
- ORDER BY embedding ⇔ %s::vector
  - issues 테이블 내 이슈의 임베딩과 입력 벡터(q_emb) 사이 **코사인 거리** 계산
- LIMIT %s;
  - 가장 가까운 것부터 LIMIT %s 개 (topk=5) 가져오기.
- results = cur.fetchall()
  - 결과((id, title, description) 형식 튜플 리스트)

▼ 코드 - 5. FastAPI 검색 API

```
# 5. FastAPI 검색 API
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Query(BaseModel):
    text: str
    topk: int = 5

@app.post("/search")
def search(query: Query):
    results = search_similar(query.text, query.topk)
    return {"query": query.text, "results": results}
```

- search() 목적
  - 새로운 이슈 텍스트 쿼리를 입력받아 DB에서 비슷한 이슈들을 API로 반환하는 REST API 형태 서비스 구현
    - search_similar 함수를 API 서버화해서 다른 애플리케이션이나 프론트엔드에서 호출하게 만든다.
- app = FastAPI()
  - FastAPI 앱 생성
- class Query(BaseModel)
  - 요청 모델 BaseModel 정의
  - text: 검색할 이슈 내용
  - topk: 유사 이슈 몇 개까지 추천할지 (디폴트 5)
- @app.post("/search")
  - /search 경로에 POST 요청이 들어오면 실행
- search_similar(query.text, query.topk)

    - 부적으로 `search_similar()` 함수를 호출해서 DB 검색 수행
- return {"query": query.text, "results": results}
    - JSON 형식으로 결과 반환

▼ 코드 - 6. 시각화 (PCA + KMeans)

```
# 6. 시각화 (PCA + KMeans)
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# 임베딩 행렬
X = list(df["embedding"])

# PCA 2D
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# KMeans
kmeans = KMeans(n_clusters=5, random_state=42).fit(X_reduced)
labels = kmeans.labels_

# 시각화
plt.figure(figsize=(8,6))
plt.scatter(X_reduced[:,0], X_reduced[:,1], c=labels, cmap="tab10")
plt.title("Issue Clusters (PCA + KMeans)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

- 목적
    - GitHub 이슈 텍스트의 의미적 구조를 PCA로 시각화
    - GitHub 이슈 텍스트를 임베딩 벡터로 변환해서 차원 축소(PCA)로 2차원에 투영하고 군집화(KMeans)로 유사한 이슈들을 그룹핑한 결과를 Scatter Plot으로 시각화 → 이슈들이 어떤 패턴으로 묶이고 있는지를 파악하기.
- X = list(df["embedding"])
    - X: 각 이슈를 SentenceTransformer로 변환한 384차원 벡터
- PCA(n_components=2) → pca.fit_transform(X)
    - 384차원 벡터를 주성분 분석(PCA)으로 2차원(PC1, PC2)에 압축
- KMeans(n_clusters=5, random_state=42).fit(X_reduced)
    - PCA로 축소된 좌표를 바탕으로 KMeans 클러스터링 (클러스터 개수: 5)

▼ 코드 - 9. RAG 구조 접목

```
 # 9. RAG 구조 접목
import psycopg2
from fastapi import FastAPI
from pydantic import BaseModel
from openai import OpenAI
from sentence_transformers import SentenceTransformer
import os
from dotenv import load_dotenv
```

```
os.chdir("/Users/yshmbid/Documents/home/github/SQL")
load_dotenv()

# FastAPI 앱
app = FastAPI()

# OpenAI 클라이언트
client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# SentenceTransformer 임베딩 모델 로드
model = SentenceTransformer("all-MiniLM-L6-v2")

# 요청 모델
class Query(BaseModel):
    text: str
    topk: int = 5
    user_id: str | None = None  # 사용자 ID 필드 (옵션)

# pgvector 기반 검색 함수
def search_similar(query, topk=5, user_id=None):
    conn = psycopg2.connect(
        host="localhost",
        port=5432,
        database="postgres",
        user="postgres",
        password=os.getenv("PG_PASSWORD"),
    )
    cur = conn.cursor()

    q_emb = model.encode(query).tolist()
    q_emb_str = "[" + ",".join(map(str, q_emb)) + "]"

    if user_id:
        cur.execute(
            """
            SELECT id, title, description
            FROM issues
            WHERE user_id = %s
            ORDER BY embedding ⇔ %s::vector
            LIMIT %s;
            """,
            (user_id, q_emb_str, topk),
        )
    else:
        cur.execute(
            """
            SELECT id, title, description
            FROM issues
            ORDER BY embedding ⇔ %s::vector
            LIMIT %s;
            """,
            (q_emb_str, topk),
        )

    results = cur.fetchall()
```

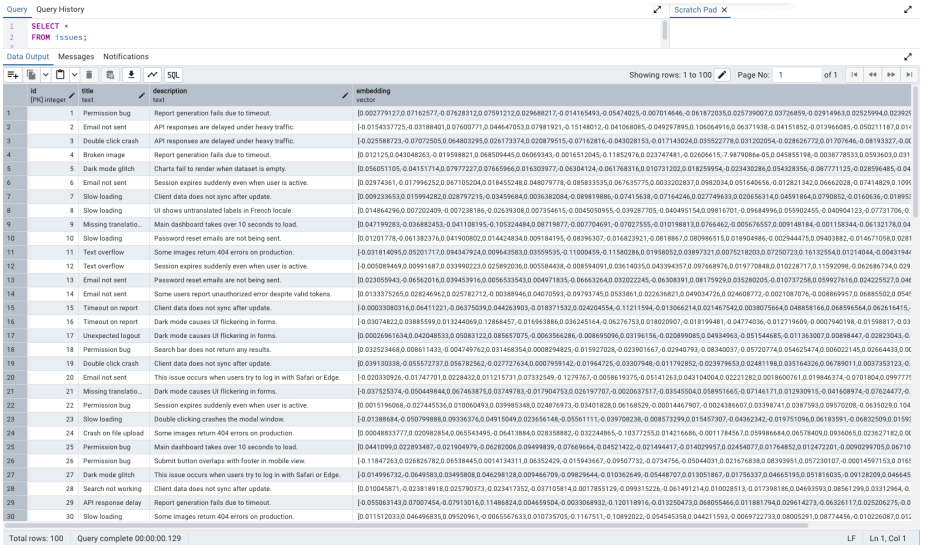
```
    cur.close()
    conn.close()
    return results

# RAG API
@app.post("/search_rag")
def search_rag(query: Query):
    # 1) pgvector 검색
    results = search_similar(query.text, query.topk, query.user_id)

    # 2) 프롬프트 구성
    context_text = "\n".join([f"- {r[1]}: {r[2]}" for r in results])
    prompt = f"""
    새로운 이슈: "{query.text}"
    아래는 DB에서 검색된 유사 이슈들입니다:
    {context_text}

    위 유사 이슈들을 참고해서,
    1) 공통된 문제 요약
    2) 잠재적인 원인
    3) 해결 방향 (가능하다면)
    을 간단히 정리해줘.
    """

    # 3) GPT 호출
    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {"role": "system", "content": "You are a helpful assistant for issue tracking."},
            {"role": "user", "content": prompt}
        ],
    )

    summary = response.choices[0].message.content

    return {
        "query": query.text,
        "results": results,
        "summary": summary
    }
```

- 목적
  - pgvector로 유사 이슈 검색 후 LLM(GPT)에 전달해서 자동 보고서를 생성
- results = search_similar()
  - 입력 텍스트를 임베딩 벡터로 변환하고 ssues 테이블에서 **코사인 거리 기반**으로 가장 유사한 Top-K 이슈 검색했다.
- prompt = f"""새로운 이슈: "{query.text}" 아래는 DB에서 검색된 유사 이슈들입니다: {context_text} 위 유사 이슈들을 참고해서 ... """
  - 검색된 이슈들 {context_text}을 GPT에 그대로 전달.
  - 읽고 공통점, 원인, 해결책을 정리하라"는 지시문 추가.
- client.chat.completions.create(model="gpt-4o-mini", messages=[{"role": "system", "content": "You are a helpful assistant for issue tracking."},{"role": "user", "content": prompt}],))
  - gpt 호출

        - gpt-4o-mini 모델이 프롬프트(messages)를 바탕으로 보고서 형태의 답변 생성.
    - return {"query": query.text, "results": results, "summary": summary}
        - results: pgvector에서 검색된 유사 이슈 5개 (id, title, description)
        - summary: GPT가 자동으로 생성한 보고서

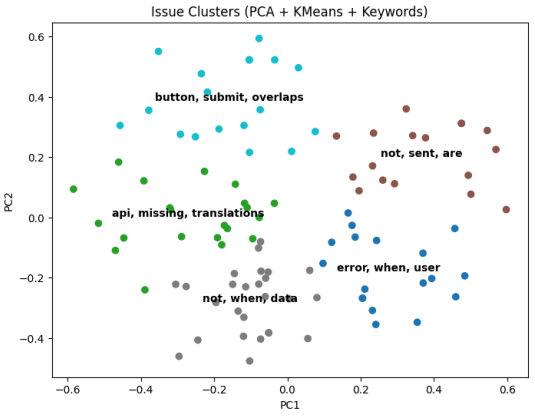
▼ 실행 결과 & 해석

▼ 데이터 적재 확인



▼ GitHub 이슈 텍스트 클러스터링 및 시각화

- 결과



클러스터 대표 키워드: {0: ['error', 'when', 'user'], 1: ['api', 'missing', 'translations'], 2: ['not', 'sent', 'are'], 3: ['not', 'when', 'data'], 4: ['button', 'submit', 'overlaps']}

- PCA+KMeans 클러스터링으로 얻은 군집의 Top-3 키워드 확인 결과 키워드는 다음과 같았습니다
    - 하늘색 군집 → "button, submit, overlaps"
    - 초록색 군집 → "api, missing, translations"
    - 파란색 군집 → "error, when, user"
    - 갈색 군집 → "not, sent, are"
    - 회색 군집 → "not, when, data"

- 결과 해석
  - 군집의 의미는 다음과 같이 예상됩니다.
    - 하늘색 군집 → UI/버튼 관련 이슈
    - 초록색 군집 → API/번역 누락 관련 이슈
    - 파란색 군집 → 사용자 동작 시 에러 발생 관련 이슈
    - 갈색 군집 → 전송/통신 관련 이슈
    - 회색 군집 → 데이터/조건 처리 문제 관련 이슈

▼ pgvector 기반 유사 이슈 검색 → RAG 구조 접목으로 자동 보고서 생성

- 결과

```
(skala) yshmbid:SQL yshmbid$ curl -X POST "http://127.0.0.1:8000/search_rag" \
>   -H "Content-Type: application/json" \
>   -d '{"text": "memory leak on login", "topk": 5, "user_id": "alice"}'
{"query":"memory leak on login","results":[[360,"Memory leak","Client data does not sync after update."
],[358,"Memory leak","Search bar does not return any results."],[331,"Memory leak","App crashes when up
loading a file larger than 50MB."],[333,"Memory leak","Text overflows from button boundaries."],[365,"M
emory leak","Some images return 404 errors on production."]],"summary":"### 1) 공통된 문제 요약\n모든
이슈에서 '메모리 누수(memory leak)'가 발생하고 있으며, 이는 애플리케이션의 성능 저하 및 비정상적인 종료
를 유발할 수 있는 문제입니다. 이러한 누수는 사용자 인터페이스 요소와 데이터 처리와 관련된 특정 기능에서
 발생하고 있습니다.\n\n### 2) 잠재적인 원인\n- **데이터 관리 불량**: 클라이언트 데이터나 파일 업로드 시
 불필요한 데이터가 메모리에 남아있어 누수가 발생할 수 있습니다.\n- **비동기 처리 문제**: 비동기 작업 이
후 리소스 해제가 이루어지지 않으면 메모리가 계속 남아있게 됩니다.\n- **UI 요소 관리**: UI 요소의 상태를
 적절히 관리하지 않거나 이벤트 리스너를 적절히 제거하지 않으면 메모리 누수가 발생할 수 있습니다.\n- **
자원 낭비**: 사용하지 않는 이미지나 데이터가 메모리에 상주하는 경우 비효율적인 자원 관리가 누수로 이어
질 수 있습니다.\n\n### 3) 해결 방향\n- **메모리 프로파일링**: 메모리 사용량을 모니터링하고 누수가 발생
하는 부분을 식별하기 위해 프로파일링 도구를 사용합니다.\n- **코드 리뷰 및 리팩토링**: 메모리 관리와 자
원 할당을 적절히 처리하고 있는지 코드 리뷰를 통해 점검하고, 필요 시 리팩토링을 진행합니다.\n- **테스트
 및 검증**: 다양한 사용 시나리오를 테스트하여 특정 상황에서 메모리 누수가 발생하지 않는지 확인합니다.\n-
 **정리 작업 구현**: 비동기 작업 완료 후에 자원을 반환하는 코드 추가 및 UI 요소와 관련된 리스너나 상태
를 적절히 정리하는 로직을 구현합니다."}(skala) yshmbid:SQL yshmbid$
```

  - 검색 결과

```
"results":[
 [360,"Memory leak","Client data does not sync after update."],
 [358,"Memory leak","Search bar does not return any results."],
 [331,"Memory leak","App crashes when uploading a file larger than 50MB."],
 [333,"Memory leak","Text overflows from button boundaries."],
 [365,"Memory leak","Some images return 404 errors on production."]
]
```

    - 모두 "Memory leak" 키워드가 포함된 유사 이슈들이 반환되었습니다.
  - GPT 요약 결과

```
### 1) 공통된 문제 요약
모든 이슈는 메모리 누수(memory leak)와 관련됨.
→ 성능 저하, 비정상 종료 유발.

### 2) 잠재적인 원인
- 데이터 관리 불량 (메모리 해제 안됨)
- 비동기 처리 문제
- UI 요소 이벤트 리스너 관리 미흡
- 자원 낭비(불필요한 이미지/데이터 상주)

### 3) 해결 방향
- 메모리 프로파일링 도구로 추적
- 코드 리뷰 및 리팩토링
- 다양한 테스트 시나리오 검증
- 비동기 작업 종료 후 자원 반환, UI 리스너 정리
```

- 결과 해석

- 검색 결과
  - pgvector가 반환한 5개의 이슈는 전부 "Memory leak" 키워드를 포함하고 있긴 하지만, 구체적인 현상은 데이터 동기화, 검색 실패, 대용량 파일 업로드 시 크래시, UI 문제, 이미지 로딩 실패 등 조금씩 다릅니다.
  - 표면적으로는 전부 "Memory leak"으로 라벨링 되어 있지만, **실제 현상은 다양한데** 이건 DB 적재 시 title에 "Memory leak"이 과도하게 많이 들어갔거나, 이슈 라벨링이 제대로 구분되지 않았기 때문에 검색 결과가 다소 "동질적인 이름, 이질적인 내용"이 된 것으로 보입니다.
- GPT 요약 결과
  - GPT는 검색된 이슈들의 공통 키워드 "Memory leak"을 중심으로 묶어서 “메모리 관리 문제”라는 상위 카테고리로 정리했습니다. 실제로는 "Text overflow"나 "Image 404" 같은 UI/네트워크 오류도 결과에 포함돼 있는데, GPT는 이들을 전부 메모리 누수 관점으로 해석했습니다.
  - 검색 컨텍스트를 우선시해서 일관된 이야기로 맞추는 경향 때문에 일부 이슈가 억지로 “memory leak” 범주 안에 포함된 것일 수 있습니다.