제품 개요

목표

- 자연어 질의(NL)를 받아 **연결된 RDB**에서 안전한 **SQL**을 생성·실행하고, **표(테이블)**와 **시각화(차트)**로 결과를 반환.
- 거대한 스키마(수천 개 테이블)에서도 전체 DDL을 LLM 컨텍스트에 통째로 넣지 않고, **Neo4j**에 저장된 **스키마 그래프 + 벡터 유사도**로 **관련 서브스키마만 증강(RAG)** 하여 정확도·안정성을 확보.
- **SQL 안전성(Only-SELECT)**, 소스 근거(어떤 테이블/관계가 선택됐는지)와 재현성(프롬프트/스키마 스냅샷 로그)을 제공.

비즈니스 가치

- 사내 방대한 데이터웨어하우스/레거시 OLTP 스키마 위에서 셀프서비스 분석·탐색 지원.
- 데이터팀 의존도/대기시간 감소, 현업의 탐색·프로토타이핑 속도 향상.
- 거버넌스(PII 보호, 쿼리 화이트리스트/레이트리미트, 실행 감사로그) 내장.

비범위(Non-Goals)

- DDL/DML(INSERT/UPDATE/DELETE) 자동 실행.
- 멀티-DB 트랜잭션 조인(초기 버전은 단일 커넥션/단일 엔진).
- 실시간 대시보드 스트리밍(초기에는 요청-응답 기반 정적 그래프).

주요 사용자 / 시나리오

- **현업 분석가**: “지난달 카드 승인 취소 사유 Top10과 거래액 추세 보여줘.”
- **데이터PM/제품기획**: “신규 가입자 대비 유료 전환률 Cohort.”
- **CS/운영**: “아이템 A 반품 주문 건의 고객 등급 분포.”

각 시나리오에서 사용자는 NL로 요청 → 후보 테이블/컬럼 서브그래프 자동 선택 → SQL 생성·검증·실행 → 표 + 추천 차트 반환.

시스템 아키텍처(개요)

1. **스키마 인제스천(DDL→Graph)**
   - 소스 RDB(예: Postgres/MySQL/Oracle)의 DDL 및 카탈로그 메타데이터를 파싱.
   - Neo4j에 **노드**(Table, Column, Index), **관계**(HAS_COLUMN, FK_TO 등)로 적재.
   - Table/Column 설명문을 임베딩 → **Neo4j 벡터 인덱스** 생성.
2. **질의 파이프라인(NL→SQL)**
   - NL 쿼리 임베딩 → Neo4j에서 **벡터 유사도**로 관련 Table/Column 후보 k개 검색.
   - 후보 사이의 **관계 경로 탐색**(FK 경로, 조인 경로 스코어링)로 **서브스키마** 구성.
   - LangChain 프롬프트에 “서브스키마(텍스트화된 제약)”와 “NL 쿼리”를 제공해 **초안 SQL** 생성.
   - **SQL 검증기**(SQLGlot/정규식/화이트리스트)로 Only-SELECT, 위험 키워드 차단, LIMIT 보장.
   - 실행 후 **표 + 시각화 스펙(Vega-Lite JSON 추천)** 반환.
3. **API/앱**
   - FastAPI 기반 `/ask` REST (또는 SSE).

- 결과 캐시/샘플링, 쿼리 로그+근거(사용된 서브스키마 스냅샷, 임베딩 매치 근거)를 반환.

상세 요구사항

기능 요구사항(FR)

1. **DDL→Neo4j 모델링**
   - 노드 타입
     - `Table{name, schema, db, description, tags, vector}`
     - `Column{name, dtype, nullable, description, vector}`
     - `ForeignKey{name, on_update, on_delete}` (선택)
     - `Index{name, columns, unique}` (선택)
   - 관계
     - `(Table)-[:HAS_COLUMN]->(Column)`
     - `(Column)-[:FK_TO]->(Column)` (타겟은 보통 PK 컬럼)
     - `(Table)-[:FK_TO_TABLE]->(Table)` (옵션, 탐색 최적화용)
   - **벡터 인덱스**: Table.vector, Column.vector 각각 생성 (Cosine/Euclidean 선택 가능).
2. **임베딩 & 검색**
   - NL 쿼리 임베딩 → Neo4j **vector similarity**로 Top-k Table/Column 후보 검색.
   - 후보 테이블 간 **최단 FK 경로**(n-hop) 탐색으로 조인 가능성 점수화.
   - 스코어 합성: α·(텍스트 유사도) + β·(경로 존재/길이) + γ·(컬럼 매칭 다양성).
3. **서브스키마 구성 & LLM 프롬프팅**
   - 선택된 테이블/컬럼, FK 제약, PK, 카디널리티 힌트(선택), 샘플 값 프리뷰(선택적·PII 마스킹).
   - LangChain **Runnable** 파이프라인으로 프롬프트 템플릿/출력 파서 구성.
   - **가드레일**: "사용 가능 테이블/컬럼 목록", "허용 함수/키워드", "JOIN 키 후보"를 명시.
4. **SQL 검증 & 실행**
   - **정책**: Only `SELECT`; `INSERT/UPDATE/DELETE/ALTER/DROP/TRUNCATE` 등 금지.
   - `LIMIT` 기본 부여(예: 1000행) 및 상한.
   - `SQLGlot` 로 파싱해 금칙어/세미콜론 다중문 차단, 서브쿼리 개수 제한.
   - 실행 전 **샌드박스/읽기 전용** 연결 사용(원 DB의 read-only role).
   - 실행 시간 타임아웃(예: 30초) 및 행수 상한(예: 10만) 방지.
5. **결과 반환 & 시각화**
   - 표 데이터(배열/Arrow/Parquet 옵션)와 함께
     - 자동 추천 **Vega-Lite** 스펙 1~3개(막대/선/파이/분포) 제안.
     - 사용자가 차트 유형을 선택하면 해당 스펙 반환.
6. **출처/근거(Explainability)**
   - 어떤 Table/Column이 왜 선택됐는지:
     - 유사도 점수, 경로 스냅샷, 프롬프트에 포함된 서브스키마 텍스트를 응답에 첨부.
7. **학습/피드백 루프(옵션)**
   - 사용자가 SQL/차트를 수정→승인 시 **우선순위 룰/프롬프트 힌트**로 축적(도메인 용어→테이블 별칭 사전, 조인 우선경로 등).

비기능 요구사항(NFR)

- **성능**: P95 전체 왕복 3~6초(임베딩/검색/LLM/쿼리 실행 포함, 10M행 기준 샘플링).
- **보안/거버넌스**:
  - DB 읽기 전용 롤, PII 컬럼 마스킹 규칙(예: 이메일/전화번호 해싱), 테이블 접근 ACL.
  - API 토큰/기관 싱글사인온 연동(추후).
- **가용성**: 무중단 배포(uv + 프로세스 매니저), Neo4j Aura/클러스터 고려.
- **관측성**: 쿼리 로그, LLM 토큰 사용량, 벡터 검색 점수, 실패 원인 태깅.

데이터 모델(Neo4j)

노드/관계 정의(예시 Cypher)

```
CREATE CONSTRAINT table_key IF NOT EXISTS FOR (t:Table) REQUIRE (t.db,
t.schema, t.name) IS NODE KEY; CREATE CONSTRAINT column_key IF NOT EXISTS FOR
(c:Column) REQUIRE (c.fqn) IS UNIQUE; // db.schema.table.column // 벡터 인덱스
(Neo4j 5.x) CREATE VECTOR INDEX table_vec_index IF NOT EXISTS FOR (t:Table) ON
(t.vector) OPTIONS { indexConfig: { `vector.dimensions`: 1536,
`vector.similarity_function`: 'cosine' } }; CREATE VECTOR INDEX
column_vec_index IF NOT EXISTS FOR (c:Column) ON (c.vector) OPTIONS {
indexConfig: { `vector.dimensions`: 1536, `vector.similarity_function`:
'cosine' } };
```

적재(예시)

- `Table{db, schema, name, description, tags, vector}`
- `Column{fqn, name, dtype, nullable, description, vector}`
- `(t)-[:HAS_COLUMN]->(c)`
- `(c1)-[:FK_TO{constraint: 'fk_orders_customer'}]->(c2)`
- (선택) `(t1)-[:FK_TO_TABLE]->(t2)`

임베딩: Table/Column의 `name + description + 샘플 컬럼값(옵션)` 을 결합하여 모델(예: text-embedding-3-large 또는 로컬 임베딩)로 생성.

파이프라인 설계

1) 인제스천(DDL→Graph)

- **입력**: `information_schema` / `pg_catalog` / `INFORMATION_SCHEMA` 조회 + DDL Export.
- **처리**:
  1. 테이블/컬럼/키/인덱스 추출
  2. 설명문(코멘트) 및 비식별 샘플값(옵션) 수집
  3. 임베딩 생성 → Neo4j upsert
- **출력**: 최신 스키마 그래프 + 벡터 인덱스 리프레시.
- **주기**: 매일/온디맨드(스키마 변경 감지 시 증분).

2) NL 질의 → 서브스키마 검색

- 쿼리 임베딩 → K 개 Table/Column 후보:

```
// Table 후보 CALL db.index.vector.queryNodes('table_vec_index', $k,
$query_embedding) YIELD node, score RETURN node AS table, score; // Column
후보 CALL db.index.vector.queryNodes('column_vec_index', $k,
$query_embedding) YIELD node, score RETURN node AS column, score;
```

- 후보 테이블들 간 FK 경로 탐색(k-hop 제한, 예: 3):

```
MATCH (t1:Table {name:$a}), (t2:Table {name:$b}), p = shortestPath((t1)-
[:FK_TO_TABLE*..3]-(t2)) RETURN p, length(p) AS hops
```

- **스코어 합성** 후 상위 N개의 테이블+관련 컬럼을 **서브스키마**로 선정.

3) LLM 프롬프트(핵심 요소)

- "사용 가능 테이블/컬럼/조인키" 목록.
- "허용 함수/키워드/예약어 금지" 룰.
- "성능 힌트": 우선 필터 컬럼/기간, GROUP BY 제한.
- **출력 포맷 강제**: SQL만, 백틱·주석 금지.

4) SQL 검증·안전장치

- SQLGlot 파싱 → DML/DDL 키워드 탐지 시 거부.
- 다중문/세미콜론 금지.
- LIMIT 자동 부여 및 상한.
- 조인 수/서브쿼리 깊이 제한(예: depth ≤ 3).
- ACL: 허용 테이블/스키마만 사용.

5) 실행 & 결과 후처리

- Python DB Driver(예: asyncpg/psycopg/mysqlclient).
- **샘플링/추출 전략**: 큰 테이블은 먼저 COUNT/Explain 후 범위 제한 or 시간 파티션 최신분.
- 결과를 **표(JSON rows) + 추천 Vega-Lite** 스펙으로 반환.

API 설계 (초안)

POST /ask

Request

```
{ "question": "지난달 카테고리별 매출 추이 보여줘", "db_key": "dw-readonly",
"visual_pref": ["line","bar"], "limit": 1000 }
```

Response

```
{ "sql": "SELECT date_trunc('day', o.order_date) AS d, c.category,
SUM(o.amount) AS revenue ...", "table": { "columns":
["d","category","revenue"], "rows": [[...], ...] }, "charts": [ { "title": "매출
```

```
일별 추이", "vega_lite": { "$schema":"https://vega.github.io/schema/vega-
lite/v5.json", "mark":"line", "encoding":{ ... } } } ], "provenance": {
"tables": ["sales.orders","sales.categories"], "columns":
["orders.order_date","orders.amount","categories.category"], "neo4j_paths": [
"orders ->(FK)-> categories" ], "vector_matches": [
{"node":"Table:sales.orders","score":0.81},
{"node":"Column:orders.amount","score":0.79} ], "prompt_snapshot_id": "ps_2025-
10-09_... " }, "perf": { "embedding_ms": 40, "graph_search_ms": 80, "llm_ms":
900, "sql_ms": 320 } }
```


`GET /tables?search=...`

- 간단한 메타데이터 검색/오토컴플리트.

`POST /feedback`

- 사용자가 SQL/차트 수정본과 메모를 보내면 도메인 사전/우선경로에 반영.

기술 스택 / 실행

- 언어: Python 3.11+
- **LLM/임베딩**: (플러그형) OpenAI, Azure OpenAI, 로컬 임베딩(영업망/보안 요구 시 교체).
- **Orchestration**: LangChain (RunnableGraph, Neo4jVector/Neo4jGraphRetriever 사용).
- **DB**: 대상 RDB(예: Postgres), 메타는 Neo4j 5.x.
- **API**: FastAPI
- **런타임/패키징**: **uv**
  - 설치/실행 예:

```
uv init uv add fastapi uvicorn neo4j langchain langchain-community
sqlglot psycopg[binary] pydantic vega-datasets uv run uvicorn
app.main:app --host 0.0.0.0 --port 8000
```

모듈 구조(제안)

```
app/
  main.py                # FastAPI 엔트리
  deps.py                # DI: Neo4j driver, DB pools, embedding client
  routers/
    ask.py               # /ask 엔드포인트
    meta.py              # /tables, /columns
    feedback.py          # /feedback
  core/
    embedding.py         # 임베딩 클라이언트
```

```
    graph_search.py      # 벡터검색+경로탐색+서브스키마 구성
    prompt.py            # LangChain 템플릿/체인
    sql_guard.py         # SQL 검증/정책
    sql_exec.py          # 실행 & 샘플링/타임아웃
    viz.py               # Vega-Lite 추천기
  ingest/
    ddl_extract.py       # RDB 카탈로그→중간모델
    to_neo4j.py          # Neo4j upsert + 벡터 인덱스 관리
  config.py
```

핵심 로직 스니펫(요약)

LangChain 체인(개요)

```
from langchain_core.prompts import ChatPromptTemplate from
langchain_core.output_parsers import StrOutputParser from langchain_openai
import ChatOpenAI sql_prompt = ChatPromptTemplate.from_template(""" You are a
senior SQL engineer. Generate a single SELECT statement only. Use ONLY the
tables/columns listed below and follow the constraints. User question:
{question} Allowed schema: {schema_text} Constraints: - SELECT only; no CTE if
not needed; no comments; LIMIT {limit} max. - Use join keys suggested:
{join_hints} - Prefer indexed filter columns if available. Return ONLY SQL,
nothing else. """) llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)
sql_chain = sql_prompt | llm | StrOutputParser()
```

그래프 검색(개요)

```
def retrieve_subschema(neo4j, query_embedding, k=10): tables =
neo4j.query_vector("table_vec_index", query_embedding, k) columns =
neo4j.query_vector("column_vec_index", query_embedding, k) candidates =
rank_and_connect(neo4j, tables, columns) # FK 경로 탐색+스코어 return
render_schema_text(candidates) # 프롬프트용 텍스트
```

SQL 가드

```
import sqlglot FORBIDDEN =
{"INSERT","UPDATE","DELETE","ALTER","DROP","TRUNCATE","CREATE","GRANT","REVOKE"}
def validate_sql(sql: str, limit_max=1000): tree = sqlglot.parse_one(sql,
read="postgres") if any(token.upper() in FORBIDDEN for token in
tree.find_all(lambda n: n.token_type)): raise ValueError("Forbidden keyword") #
LIMIT 강제/상한 # 조인/서브쿼리 깊이 체크 등 추가 return True
```

시각화 추천(요약)

- 스키마:
  - 시간열 존재 + 수치 → **line**
  - 범주 + 수치 합계 → **bar**
  - 비율 → **pie** (행수 ≤ 10)
- Vega-Lite 스펙 자동 생성:

```
{ "$schema": "https://vega.github.io/schema/vega-lite/v5.json", "data":
{"name":"table"}, "mark":"bar", "encoding":{"x":
{"field":"category","type":"nominal"}, "y":
{"field":"revenue","type":"quantitative"}} }
```

보안/거버넌스

- **DB 권한**: 읽기 전용 계정, Row-Level Security(가능 시).
- **PII 마스킹**: 설정 기반으로 열 이름/정규식 매칭해 마스킹.
- **쿼리 레이트 리미트**: 사용자·토큰 단위.
- **감사로그**: 입력 질의, 서브스키마 스냅샷, 생성 SQL, 실행 계획 요약, 결과 통계.

테스트 전략

- **유닛**: DDL 파서, 그래프 경로 탐색, SQL 가드.
- **통합**: End-to-end 질의 셋(의도→정답 SQL), 허위양성/누락률 측정.
- **성능**: 스키마 5k 테이블 기준 벡터 검색<100ms, FK 탐색<200ms 목표.
- **휴먼인더루프**: 오답 SQL 수동 교정 → 룰/사전 업데이트.

단계적 로드맵

1. **MVP (4~6주)**
   - Postgres 1개 소스 지원, Neo4j 스키마 그래프/벡터, 기본 RAG→SQL, 표/단일 차트.
   - 기본 보안(Only-SELECT, LIMIT, 마스킹).
2. **V1**
   - 조인 경로 스코어링 고도화(통계/카디널리티 힌트), 피드백 루프, 캐시/샘플링 전략.
3. **V1.5**
   - 다중 DB 커넥션, 메타 ACL, 쿼리 스케줄/북마크, 간단한 대시보드.
4. **V2**
   - 온프레미스 임베딩·LLM 대체, 고급 거버넌스(열 수준 권한, PII 카탈로그 연동), 비용/토큰 최적화.

운영/배포

- **uv** 기반 실행:

```
uv run uvicorn app.main:app --host 0.0.0.0 --port 8000
```

- 설정(`.env` / `config.py`):
  - `NEO4J_URI` , `NEO4J_USER` , `NEO4J_PASSWORD`
  - `DB_URI` , `DB_USER` , `DB_PASSWORD` (read-only)
  - `EMBEDDING_PROVIDER` , `LLM_PROVIDER`
- 컨테이너화(선택): `uv` 러너 + FastAPI 이미지.

리스크 & 대응

- **스키마 드리프트**: 증분 인제스천/변경감지, 인덱스 리빌드 자동화.
- **오답 SQL**: 엄격한 가드, 휴먼 승인 모드, 빠른 피드백 반영.
- **성능 이슈**: 큰 테이블 샘플링/파티션 프룬, FK 경로 캐시.
- **보안**: 마스킹 누락 방지 위한 테스트/룰 베이스, 감사 로그 정기점검.