GAMES ON AWS 2026

aws FOR GAMES

AI 에이전트로 재정의한 DB 운영 :

전문 DBA Agent 들을 서로 협업시켜 동작시키기

박현수

Solutions Architect

Amazon Web Services

신우성

DBA, DB운영셀

카카오게임즈

곽은경

DBA 리드

카카오게임즈

GS Neotek | MEGAZONE CLOUD Transform · Tomorrow · Together

LLM은 우리 DB를 모른다 — 세 가지 공백

모델이 똑똑해도 판단에 쓸 근거가 없으면, 그럴듯하게 틀립니다

WHY | 의미 컨텍스트 | 운영 컨텍스트 | 하네스



01

의미의 공백

"이 컬럼은 무엇을 뜻하나?"

- 질문은 업무 언어, 스키마는 약어와 코드
- 관계와 코드값의 의미는 문서가 아니라 데이터 안에
- 모델은 빈칸을 그럴듯한 이름으로 채운다

결과 **없는 컬럼, 틀린 JOIN으로 그럴듯한 SQL**



02

실행 시점의 공백

"지금 DB 안에서 무슨 일이?"

- 학습 데이터에는 지금의 세션·락·대기가 없음
- 엔진마다 진단 뷰가 다름
(DMV · pg_stat_* · performance_schema)
- 메트릭과 쿼리 근거가 다른 시스템에 흩어짐

결과 **"인덱스를 확인해 보세요" 같은 일반론**



03

경계의 공백

"어디까지 해도 되나?"

- 프롬프트 지시는 우회되거나 주입될 수 있음
- 에이전트가 쥔 권한 = 사고의 최대 반경
- 근거 없는 수치가 답에 섞여도 구분 불가

결과 **운영 DB에서 되돌리기 어려운 실수**

필요한 건 더 큰 모델이 아니라 — 모델에 넣을 컨텍스트와 모델을 감싸는 구조

세 공백을 메우는 세 개의 층

컨텍스트 두 층이 근거를 공급하고, 하네스가 모델이 할 수 있는 일을 정합니다

WHY | 의미 컨텍스트 | 운영 컨텍스트 | 하네스

의미 컨텍스트

SEMANTIC


OpenSearch


Neptune

이 데이터는 무엇이고, 어떻게 연결되나

테이블·컬럼 설명과 신뢰도 · 값으로 복원한 PK·FK
업무 개념 온톨로지(IS_A · REL) · 조인 그래프

운영 컨텍스트

OPERATIONAL


CloudWatch


RDS

지금, 무슨 일이 일어나고 있나

쿼리·세션·락·대기·실행계획 · 메트릭·로그·이벤트
같은 대상 · 같은 시간창으로 정렬

하네스 — 모델을 감싸는 구조



Amazon Bedrock

판단만 한다
도구 선택 · 해석 · 작문

1 결정적 순서

2 최소 권한 도구

3 실행 전 검사

4 사람의 승인

근거가 붙은 답

- 도구 — 어떤 도구로 봤는지
- 수치 — 어떤 값이었는지
- 시간창 — 언제의 데이터인지
- 신뢰도 — 얼마나 확실한지

판단 · 조치는 DBA

● Part 1 신우성 — 의미 컨텍스트를 게임 DB에 ● Part 2 곽은경 — 하네스를 프로덕션에 ● 지금 AWS — 설계 원리와 프로토타입 검증

DB 운영 AI 도입 — 5단계 방법론

모델을 고르기 전에 업무를 쪼개고, 컨텍스트와 구조를 먼저 설계합니다

WHY | 의미 컨텍스트 | 운영 컨텍스트 | 하네스

01
업무 분해
무엇을 맡길까

업무를 유닛으로 쪼개고
에이전트 몫과 사람 몫을 가른다

산출물

업무 유닛 목록
1차 타깃 하나

Part 2

02
의미 컨텍스트
이 데이터는 무엇인가

DB에서 설명 · 관계 · 개념을
복원하고 검수한다

산출물

카탈로그 · 조인 그래프
온톨로지



AWS · Part 1

03
운영 컨텍스트
지금 무슨 일인가

진단 질문마다 근거 도구를
엔진별 고정 SQL로

산출물

진단 도구 세트
데이터 소스 목록



AWS

04
하네스
어디까지 해도 되나

순서 · 권한 · 자격증명 · 승인을
프롬프트 밖 구조로

산출물

고정 그래프 · 도구 목록
승인 게이트



AWS · Part 2

05
측정 · 확장
얼마나 맞나

층별 지표와 실행 로그로 재고
유닛을 하나씩 늘린다

산출물

블라인드 평가셋
호출 로그 · 비용



AWS · Part 2

한 번에 다 하지 않는다 — 1차 타깃 하나로 01→05를 끝까지 돌리고, 그다음에 유닛을 늘린다

db2doc — 읽고, 관계를 찾고, 의미를 쓴다

DBA가 처음 보는 DB를 받았을 때 하는 일을 다섯 단계로 고정했습니다 — 원본 DB는 읽기만 합니다

WHY | 의미 컨텍스트 | 운영 컨텍스트 | 하네스

1 프로파일링

코드

테이블당 집계 쿼리 1번
NULL 비율 · 고유값 · 최소·최대
자주 나오는 값 상위 10개

통계 · 표본

2 관계 복원

코드 + LLM 제안

PK: 유일성을 전체 테이블로 확인
FK: 자식 값 ⊂ 부모 키 (값포함관계)
LLM이 제안한 FK도 값 검증 통과해야

PK · FK → ERD

3 설명 쓰기

LLM

코드값은 코드표를 실제 JOIN해 라벨
FK 순서대로 부모 테이블부터
통계 · 표본 · 키를 증거로 넘김

테이블 · 컬럼 설명

4 검증 · 신뢰도

코드 + LLM

연결된 테이블끼리 모순 탐지
걸린 테이블만 1회 다시 쓰기
근거 약하면 신뢰도 강등 → 검수

신뢰도 · 검수 큐

5 그래프 · 온톨로지

코드 + LLM

카탈로그를 스키마 그래프로 적재
LLM이 개념 · IS_A 추출
개념 관계는 복원 FK에 근거해야

ERD 그래프 · 온톨로지

입력 테이블 · 컬럼 이름, 타입, 그리고 데이터 — 문서 없는 DB가 보여 주는 것만 **출력** **설명 카탈로그 + ERD + 온톨로지**

설명은 3단계부터 — 그 전까지 코드가 데이터에서 근거를 모으고, LLM의 FK 제안도 값으로 검증한다

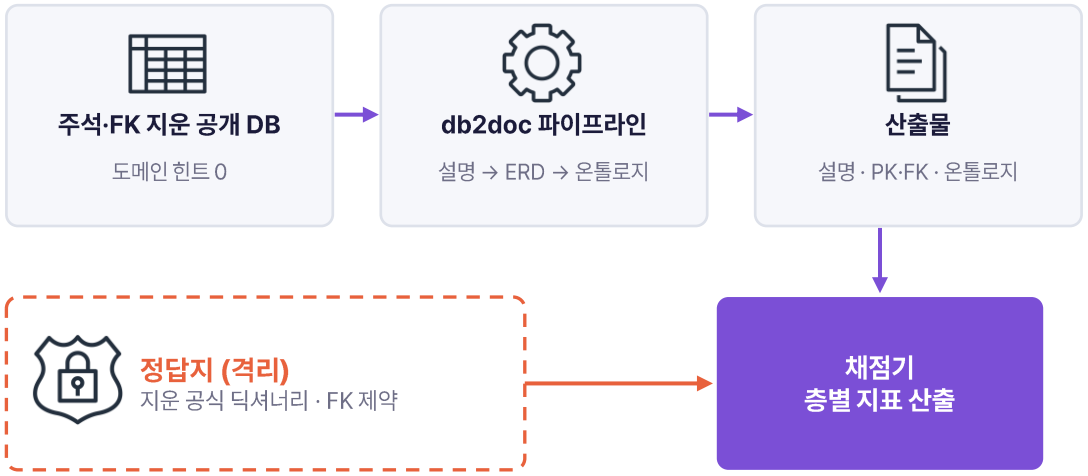
출처: github.com/blait/dbMetaToDoc v2/profiler.py(Stage 1) · relations.py(2) · describe.py(3) · catalog.py(4) · graph.py(7) · concepts.py(8) | OMOP CDM 공개 스키마로 검증

이 방법이 쓸만한거 맞나? — 평가 설계

정답지를 파이프라인에서 떼어 두고, 층마다 따로 채점합니다

WHY | 의미 컨텍스트 | 운영 컨텍스트 | 하네스

문서를 지우고 복원시킨 뒤, 지운 문서로 채점한다



층마다 따로 잰다 — 아래 층의 오류는 위로 번진다

관계 — **PK F1 · FK F1**
찾은 것 중 맞은 비율(정밀도)과 정답 중 찾은 비율(재현율)의 조화평균

의미 — **컬럼 설명 의미 일치**
LLM 심사관이 정답과 뜻이 같은지 판정 (임베딩 유사도 병기)

활용 — **검증 질의 실행**
대표 질문 → SQL → 실제 DB 실행, 성공한 것만 저장

종합 — **S_overall**
층별 지표를 묶어 개선 이력을 추적하는 하나의 숫자

실행 검증의 함정

자기 출력으로 자기를 채점하지 않게

자기참조 방지
런 시작 전 기존 검증 질의를 비우고 빈 상태에서 생성

런 격리
few-shot 조회는 run_id 범위만 — 다른 DB 결과 혼입 방지

재시도 일관성
few-shot은 1회 조회 후 캐시 — 수정 재시도 중 예시 고정

출처: github.com/blait/dbMetaToDoc prepare/strip_docs.py(주석·FK 제거) · v2/score.py(채점) · db2doc_overview.pptx 26장 | 온톨로지는 점수 대상 아님 — 검증 질의로 간접 확인

검증 결과 — 두 번 개선하여 정확도를 의미있는 수준으로 높였습니다.

OMOP CDM 5.3 공개 스키마 · 37 테이블 · 396 컬럼 · 주석·FK 제거 · 도메인 힌트 0

WHY | 의미 컨텍스트 | 운영 컨텍스트 | 하네스

① 통계만

v1 baseline

PK F1	0.667
FK F1	0.394
컬럼 설명	0.931
종합 S	0.687

FK 정밀도 0.951 · 재현율 0.248
찾은 건 맞았지만 157개 중 39개

원인 — 0행 테이블 18개

값이 없으니 비교 불가

바꾼 것

→ 빈 테이블은 이름 규칙으로 PK·FK 후보 (낮은 신뢰도)
→ 선언된 키가 있으면 그대로 채택
→ 데이터 없는 설명은 신뢰도 강등
0.917 → 0.427

② 빈 테이블 보강

v1 개선

PK F1	0.912
FK F1	0.619
컬럼 설명	0.938
종합 S	0.840

PK 재현율 1.0
FK 재현율 0.471 — 아직 절반

원인 — 이름 안 닮은 FK

self-FK도 통계로는 못 찾음

바꾼 것

→ LLM이 FK 후보 제안 → 값포함 관계로 판정
→ preceding_ · parent_ 역할 접두사 해석
→ PK 유일성은 표본 아닌 전체 테이블로
→ 코드값 라벨을 실제 JOIN · 도메인 힌트 제거

③ LLM 제안 + 값 검증

v2 · 블라인드

PK F1	0.945
FK F1	0.963
컬럼 설명	0.953
종합 S	0.970

FK 157개 중 155개
정밀도 0.939 · 재현율 0.987

가장 큰 도약은 LLM이 후보를 내고, 데이터가 판정하는 구조에서
FK F1 0.619 → 0.963 · 설명은 처음부터 0.93 — 어려운 건 쓰기가 아니라 찾기

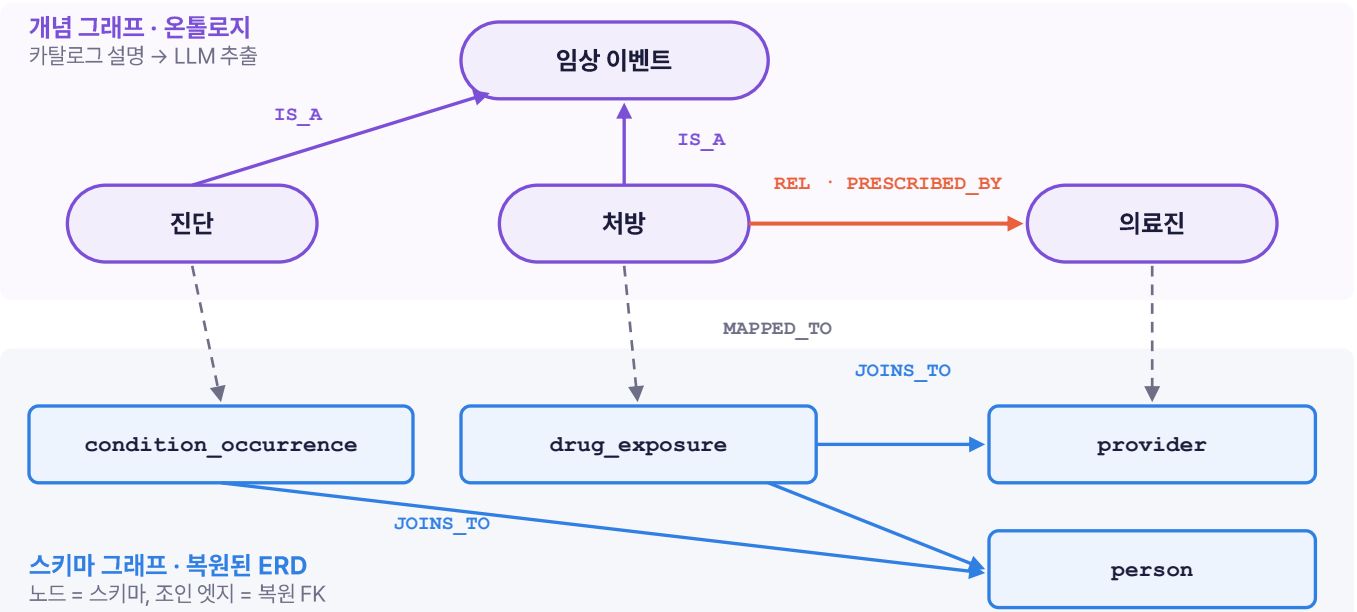
효과 없던 것도 기록
반복 정제(backprop) — 고친 설명 0건 → 기본 비활성

공개 스키마 1종(GiBleed) 결과 · 단계별 기여는 변경 묶음 단위로만 측정 | 출처: github.com/blait/dbMetaToDoc RESULTS.md · IMPROVEMENTS.md · v2/README.md 결과 표

두 개의 층 — 스키마 그래프와 온톨로지

아래는 스키마와 값으로 복원한 ERD, 위는 카탈로그에서 뽑은 업무 개념

WHY | 의미 컨텍스트 | 운영 컨텍스트 | 하네스



같은 연결, 두 가지 언어

스키마 `drug_exposure.provider_id → provider.provider_id (N:1)`

온톨로지 (처방) —PRESCRIBED_BY→ (의료진)

조인 엣지	선언 FK + 값포함관계 · 이름 · LLM 제안 — 값으로 검증
개념 · IS_A	LLM이 카탈로그 설명에서 추출 — 없는 테이블 참조는 버림
개념 관계	복원 FK 하나에 근거(via)해야 생성 — RELATED_TO 금지
카디널리티	LLM이 아니라 컬럼 통계 — distinct 비율 ≥ 0.99면 1:1

아래 층은 스키마와 값이, 위 층은 LLM이 — 단, 위 층의 모든 개념과 관계는 아래 층에 근거가 있어야 남는다

출처: github.com/blait/dbMetaToDoc v2/graph.py(Stage 7) · v2/concepts.py(Stage 8) · v2/relations.py(Stage 2) · OMOP CDM 예시 | 카카오게임즈의 개념 층 구현(쿠폰 · 이벤트 사양 · 리더보드)은 Part 1

GraphRAG — 벡터로 들어가고, 그래프로 넓힌다

질문 "임상 이벤트 종류별 건수를 알려줘" — 이 단어가 들어간 컬럼은 하나도 없습니다

WHY | 의미 컨텍스트 | 운영 컨텍스트 | 하네스

1 진입

벡터 검색

설명 임베딩 유사도로
가까운 테이블 · 개념 찾기

→

2 확장

개념 매칭 → IS_A

"임상 이벤트"의 하위 개념
진단 · 처방 · 검사 (1~3단계)

→

3 수집

MAPPED_TO

하위 개념이 가리키는
실제 테이블 모으기

→

4 연결

JOINS_TO 경로

후보 테이블 쌍마다 1~4홉
→ 복원 FK 기반 조인 조건

그래프 탐색 — openCypher (구현 쿼리를 단순화)

```
// 2·3  상위 개념 → 하위 개념 → 테이블
MATCH (d:Concept)-[:IS_A*1..3]->(:Concept {name:'Clinical Event'})
MATCH (d)-[:MAPPED_TO]->(t:Table) RETURN t.name
// 4  후보 테이블 쌍 사이 조인 경로
MATCH p = (a:Table {name:$a})-[:JOINS_TO*1..4]-(b:Table {name:$b})
```

벡터 RAG만	"임상 이벤트"가 든 컬럼이 없어 놓친다 조인 조건은 LLM이 추측
GraphRAG	**상위 개념을 하위 테이블로 번역 (실측: 8개)** **조인 조건은 복원 FK 위의 경로로 제공**

벡터는 어디서 시작할지, 그래프는 어디까지 갈지 — 두 검색은 합집합으로, 결과는 근거와 함께 LLM에

출처: github.com/blait/dbMetaToDoc v2/text2sql.py(쿼리) · v2/SOLUTION.md("임상 이벤트" → 하위 8개 테이블 실측) · db2doc_overview.pptx 16–21장

의미 컨텍스트는 어디에 두나 — 역할별 저장소

원본은 하나, 검색·탐색용 사본은 원본에서 다시 만든다 — 고를 때는 역할부터



원본 · 검수 이력

무엇이 진짜인가?

Amazon Aurora · Amazon RDS (MySQL · PostgreSQL)

고르는 기준

- 사람이 고친 설명까지 담는 단일 진실
- 벡터 · 그래프 사본은 여기서 언제든 재생성



의미 검색 (벡터)

어떤 테이블이 관련 있나?

OpenSearch Serverless 벡터 컬렉션
Aurora PostgreSQL + pgvector

고르는 기준

- 키워드 + 벡터 하이브리드 검색이 필요하면 OpenSearch
- 이미 PostgreSQL을 운영 중이면 pgvector로 저장소 추가 없이



관계 탐색 (그래프)

어떻게 연결되나?

Amazon Neptune Analytics
Amazon Neptune Database

고르는 기준

- 다중 홉 조인 경로 — openCypher 한 줄 vs 재귀 CTE
- Neptune Analytics는 그래프 안의 벡터 검색도 openCypher로 호출



추론 · 임베딩

무엇을 뜻하나?

Amazon Bedrock
Claude · Titan Text Embeddings V2

고르는 기준

- 임베딩 모델 = 인덱스의 스키마 — 바꾸면 전체 재색인
- 모델 버전을 메타데이터에 함께 기록

AWS 프로토타입	RDS MySQL · OpenSearch Serverless · Neptune Analytics · Amazon Bedrock
카카오게임즈 (Part 1)	MySQL Meta Store · PostgreSQL pgvector · neo4j · Amazon Bedrock

저장소는 달라도
역할 분담은 같다

출처: db2doc_overview.pptx 28장(프로토타입 구성) · 고객 발표 3장(카카오게임즈 구성) · Neptune FAQ, Neptune Analytics 벡터 검색 문서, OpenSearch Serverless 벡터 컬렉션(2026-10-12 확인)

같은 질문, 엔진마다 다른 근거

질문은 같아도 근거를 꺼내는 SQL은 엔진마다 다릅니다 — 번역은 도구가 맡습니다

WHY | 의미 컨텍스트 | 운영 컨텍스트 | 하네스

진단 질문	진단 도구	SQL Server	PostgreSQL	MySQL	AWS 관리형
지금 오래 도는 쿼리는?	get_slow_queries	DMV exec_requests	pg_stat_activity	processlist	—
누가 누구를 막고 있나?	get_blocking_sessions	블로킹 체인 DMV	pg_blocking_pids()	data_locks 조인	—
자원을 가장 많이 쓴 쿼리는?	get_top_queries	Query Store 시간창 지정	pg_stat_statements 누적	performance_schema digest · 누적	Performance Insights AAS 기준 Top SQL
무엇을 기다리나?	get_wait_stats	Query Store 쿼리별 대기	activity 대기 스냅샷	누적 wait 이벤트	Performance Insights 대기 이벤트
어떻게 실행되나?	get_query_plan	플랜 캐시 XML	EXPLAIN (FORMAT JSON)	EXPLAIN FORMAT=JSON	—
인프라 · 이벤트는?	DBAOps에 위임 (A2A)	—	—	—	CloudWatch 지표 · 로그 RDS 이벤트·failover

진단 SQL은 코드에 고정

LLM은 '어떤 도구를 부를지'만 고른다 — 13종 도구, target 파라미터로 엔진별 SQL 자동 분기

같은 숫자도 시간의 의미가 다르다

Query Store는 시간창, pg·MySQL 누적 통계는 리셋 이후 누적 — 비교 전에 기준 시점부터 확인

출처: DBAOps-A2A-DBPerf-소개.pptx 5–10장 (AWS 프로토타입 DBPerf · DBAOps 도구 정의)

질문은 한 번, 전문가 에이전트는 둘

DBPerf(쿼리)와 DBAOps(인프라)가 A2A로 서로 위임해 하나의 리포트로 답합니다

WHY | 의미 컨텍스트 | 운영 컨텍스트 | 하네스

Slack · Web **"mssql 느린 쿼리 원인을 분석하고, 같은 시간대 서버 CPU 상황도 같이 봐줘"**

DBPerf 쿼리 성능 전문가 · 신규 개발

stdio MCP · 진단 도구 13종

SQL Server · PostgreSQL · MySQL

· 회귀 쿼리 · 블로킹 · 실행계획 · 대기를 DB에서 직접 조회
· 인프라 근거가 필요하면 → ask_dbaops_agent

A2A

양방향 위임
결과는 출처와 함께 인용

DBAOps 인프라 RCA 전문가 · 기존 자산

CloudWatch | Performance Insights | Amazon MSK | S3 로그

· 호스트 메트릭 · 로그 · AWS 이벤트로 RCA
· 쿼리 관점이 필요하면 → ask_perf_agent

표준으로 서로를 발견

agent card 엔드포인트 — 외부 시스템도 같은 A2A로 직접 연동

기존 에이전트는 무수정

A2A 서버 한 파일만 추가해 기존 DBAOps 그래프를 그대로 노출

위임 루프는 아직 프롬프트로

역할 경계 프롬프트로 무한 위임을 막음 — 위임 깊이 상한 같은 구조로 옮길 다음 후보

출력 하나의 통합 리포트 — 쿼리 원인 + 같은 시간대 인프라 상황, DBAOps 인용은 출처 표시

출처: DBAOps-A2A-DBPerf-소개.pptx 3–4·12–13·16장 (AWS 프로토타입) | '다음 후보'는 SA 제안이며 구현된 기능이 아님

규칙은 프롬프트가 아니라 구조에 둔다

AWS Well-Architected Agentic AI Lens의 권고와, 이를 받쳐 주는 AWS 구성 요소

WHY | 의미 컨텍스트 | 운영 컨텍스트 | 하네스

> **프롬프트 지시만으로 구현한 권한 통제는 충분하지 않다. 에이전트가 어떻게 그 호출에 이르렀든, 도구가 실행되기 전에 결정적인 외부 검사로 구현하라.**
>
> AWS Well-Architected Agentic AI Lens · AGENTSEC02-BP01 (번역) + 되돌리기 어려운 변경 작업은 사람의 검토(Human-in-the-loop)를 거치게 할 것

원칙	Lens가 짚는 안티패턴 → 권고	AWS 구성 요소
실행 전 인가	에이전트 스스로 호출이 적절한지 판단하게 둔다 → 모든 도구 호출을 실행 전에 정책으로 인가	**AgentCore Gateway + Policy** 단일 MCP 엔드포인트 · Cedar · default-deny, forbid 우선
최소 권한 도구	모든 도구를 일괄 허용한다 → 에이전트마다 필요한 도구만, 도구 과다 노출도 위험	**Policy in AgentCore · 시맨틱 도구 검색** 역할별 permit · 필요한 도구만 찾아 쓰는 내장 검색 도구
아이덴티티 전파	모든 호출이 에이전트 권한으로 실행된다 → 사용자 컨텍스트를 도구 호출까지 전파	**AgentCore Identity + AWS Secrets Manager** 워크로드 아이덴티티 · 토큰 볼트 · 자격증명 분리
관측 후 적용	정책을 바로 강제해 정상 호출까지 막는다 → 실트래픽으로 먼저 검증하고 강제	**LOG_ONLY → ENFORCE · AgentCore Observability** Amazon CloudWatch에서 에이전트 · 도구 호출 트레이스

카카오게임즈가 이 원칙을 직접 구현한 방식은 Part 2 · 오른쪽 열은 선택지이며 사례가 사용한 서비스가 아님 | 출처: docs.aws.amazon.com/wellarchitected/latest/agentic-ai-lens/agentsec02-bp01.html

GAMES ON AWS 2026

aws FOR GAMES

카카오게임즈 DB운영팀

이제, 프로덕션에서 무엇이 달랐는지

PART 1

DB 설계 명세를 온톨로지로

신우성 · DBA

의미 컨텍스트를 게임 QA DB에 — 값으로 찾는 FK, 신뢰도로 줄인 검수, Text2SQL까지

의미 컨텍스트

PART 2

AI 에이전트로 재정의한 DB 운영

곽은경 · 팀장 · DB 운영셀

하네스를 프로덕션에 — 순서 · 기준 · 금지 · 반복, 운영 로그로 확인한 것

운영 컨텍스트

하네스

저는 원리와 프로토타입을, 두 분은 실제 환경에서 바뀐 것과 운영에서 확인한 것을 말씀드립니다.

GS Neotek | MEGAZONE CLOUD Transform · Tomorrow · Together

GAMES ON AWS 2026

aws FOR GAMES

DB 설계 명세를 온톨로지로

신우성

DBA, DB운영셀

카카오게임즈

GS Neotek | MEGAZONE CLOUD Transform · Tomorrow · Together

배경

운영 담당자의 데이터 추출 니즈는 상시 존재하고, 요청은 DB를 아는 소수에게 몰립니다

기획 · 운영 · CS · 사업 — 서비스 운영 담당자들의 데이터 추출 요청이 꾸준히 들어옵니다.
요청을 받은 쪽은 매번 스키마를 다시 파악하고, 같은 질문에 같은 탐색을 반복합니다.

주석이 없거나 낡았다

COMMENT를 끝까지 채운 DB는 드물고,
남아 있어도 스키마 변경을 따라오지 못함

관계가 선언되어 있지 않다

성능·샤딩 때문에 FK 제약을 걸지 않음
→ 관계는 코드와 사람 머릿속에만 존재

아는 사람이 남지 않는다

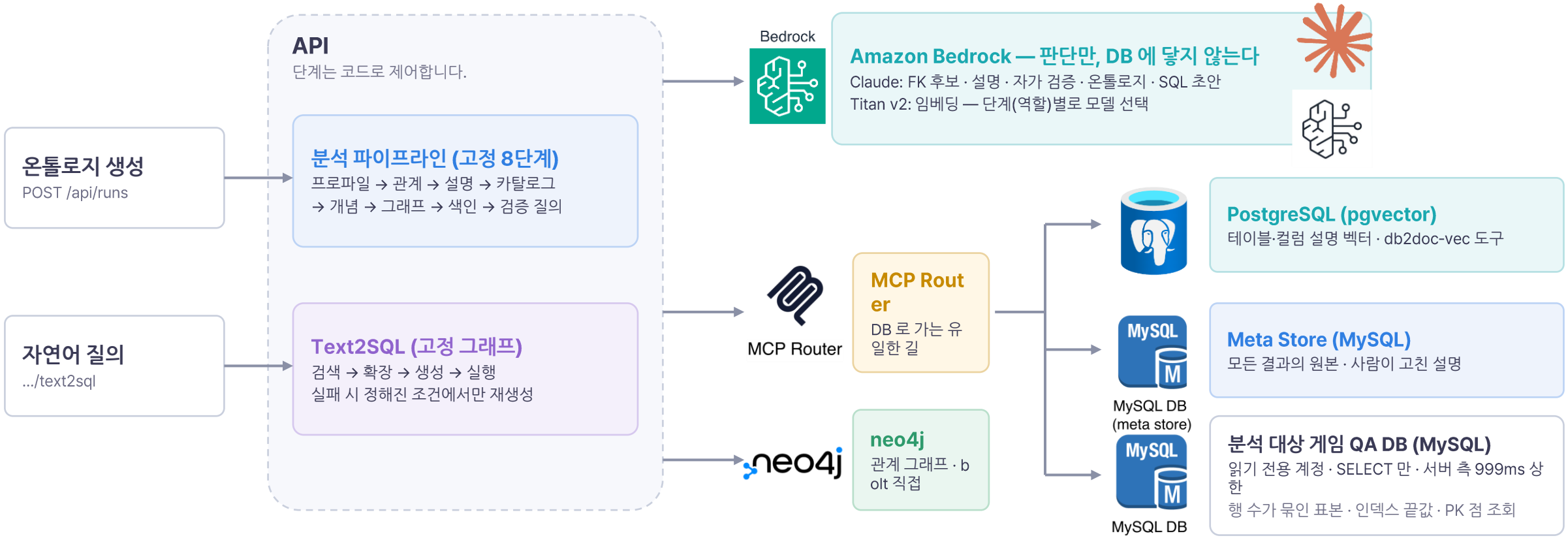
개발사·담당자·조직이 바뀌며
맥락이 문서가 아닌 인수인계로만 전달

그래서 요청이 올 때마다 데이터를 열어보고, 값을 맞춰보고, 테이블 의미를 추측하는 일을 반복합니다.

이 과정을 코드로 정해두고, 판단이 필요한 부분만 LLM 에 맡겨서 결과를 온톨로지로 남기려고 했습니다

아키텍처 — 온톨로지 구축부터 Text To SQL 까지

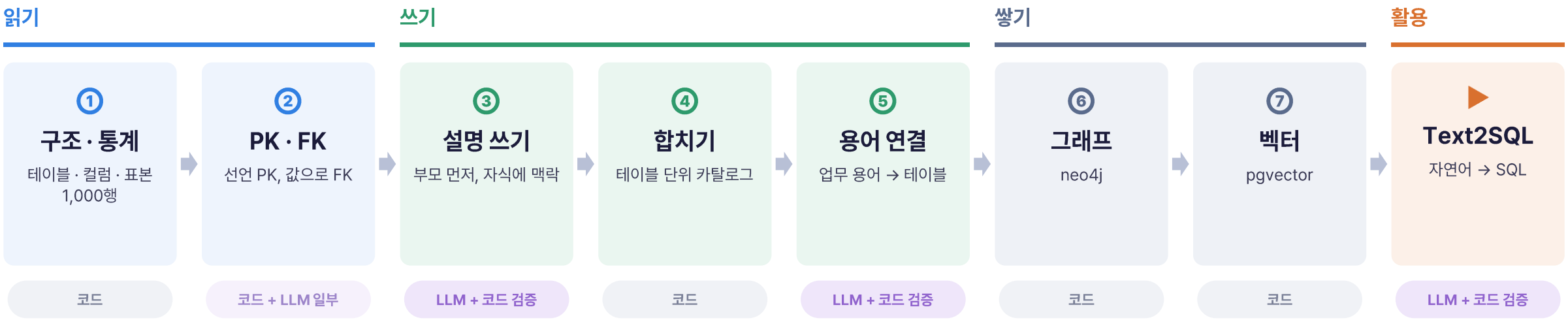
왼쪽 두 입구(온톨로지 생성 / 자연어 질의)에서 시작해 같은 메타데이터를 공유합니다



LLM 은 정해진 단계에서 판단만 하고, 검증과 DB 접근은 코드가 합니다

전체 흐름

크게 읽기 · 쓰기 · 쌓기 순서로 진행하고, LLM 은 정해진 단계에서만 사용합니다



에이전트가 아니라 고정 파이프라인이라서, 같은 입력이면 항상 같은 순서로 실행됩니다

① 구조와 통계 파악

information_schema 와 샘플 1,000행만 읽습니다

1-1 가져오는 것

테이블 · 컬럼 · 주석 · 타입 · 콜레이션

`information_schema.tables / columns`

대상 DB 에 부담 주지 않게

- 백업 · 테스트용 테이블은 분석에서 제외
- 읽기 전용 계정만 사용
- 쿼리마다 999ms 넘으면 중단

1-2 샘플 1,000행에서 개수 · 고유값 · 최소 · 최대

```
SELECT count(*)                AS n,
       count(c1)               AS nn_1,
       count(DISTINCT c1)      AS d_1,
       CAST(MIN(c1) AS CHAR)   AS mn_1,
       CAST(MAX(c1) AS CHAR)   AS mx_1    -- … 컬럼마다 반복
  FROM (SELECT c1, c2, … FROM db.t LIMIT 1000) s;
```

1-3 자주 나오는 값 상위 10개

```
SELECT CAST(v AS CHAR) AS v, count(*) AS c
  FROM (SELECT c1 AS v FROM db.t LIMIT 1000) s
 WHERE v IS NOT NULL
 GROUP BY v
 ORDER BY c DESC
 LIMIT 10;
```

집계는 전부 LIMIT 1000 으로 자른 샘플 안에서만 합니다

② PK · FK 파악

타입 → 범위 → 값 → 키 밀도 순으로 거르고, 빽빽한 키는 단서로 판정합니다

① 타입 · 콜레이션 일치
부모 = PK · UNIQUE 단일 컬럼

예 →

② 범위 밖 ≤ 20% ?
고아 행 고려 · 문자열 키는 생략

예 →

③ 일치율 ≥ 80% ?
자식 표본을 부모 키에 조회

예 →

③ 고유값 ≥ 20개 ?
자식 표본의 서로 다른 값

예 →

④ 부모 키 밀도
행 수 ÷ 값 범위

아니오 → **탈락**

아니오 → **탈락**

아니오 → **작은 코드표**
이름 2점 이상이면 검수

밀도 < 0.5

밀도 ≥ 0.5

4-1 듬성듬성한 키

→ 자동 FK

범위 안이고 80% 이상 맞는데 빈 번호가 많은 키라면, 우연으로는 이만큼 맞을 수 없음

4-2 빽빽한 키 (대개 auto_increment) → 단서 점수로 판정

컬럼명 2점	부모 테이블명(접두사 뗀 것 포함) + 접미사 · 부모 PK 이름 그대로 · 역할어 + 부모 PK	**4점 이상 · 부모 하나 → 자동 FK**
주석 2점	흔한 말(아이디 · 번호 · 그룹 …)을 뺀 핵심어가 부모 PK 주석과 같음	**4점 이상 · 부모 여럿 → 검수** 자식 컬럼 단위로 후보를 묶어서
테이블명 1점	자식 · 부모 테이블명에 공통 단어 (접두사 제외)	**2~3점 → 검수**
인덱스 1점	자식 컬럼에 인덱스가 있음	**0~1점 → 버림**

자동으로 FK 가 되는 건 두 경우뿐입니다 듬성듬성한 키, 또는 빽빽해도 단서 4점 이상 + 부모 하나. 그 외에는 검수로 넘기거나 버립니다

③ 설명 쓰기 (작성 순서)

설명은 AI 가 쓰고, 어떤 순서로 무엇을 넘겨줄지는 코드에서 정합니다

① 코드값 확인

FK 를 따라가서 코드값의 실제 의미 확인

부모 테이블에 저장된 이름을 그대로 가져옴

`job_code = 10 → WARRIOR`

② 부모 먼저

부모 테이블부터 작성

기존 주석(description)이 있으면 그걸 우선 사용

③ 자식 작성

부모 정보를 같이 넘겨서 작성

부모 테이블 설명과 참조 키 컬럼 설명을 함께 전달

④ 모순 검토

부모 1개와 자식들을 묶어서 비교

어긋나는 부분이 있으면 해당 테이블만 다시 작성

작성 순서 예시

`code_master`
코드표 (코드값의 의미)

④ 모순 검토 범위

`character`
부모 — 먼저 작성

`character_item`
자식 — 부모 설명을 받아서 작성

AI 가 추측으로 채우지 않도록, 코드값 · 기존 주석 · 부모 설명을 먼저 넘겨줍니다

③ 설명 쓰기 (신뢰도)

설명마다 신뢰도를 매기고, 근거가 부족한 건 따로 표시합니다

데이터로 확인된 만큼만 점수

seller_type 60%
값이 1 하나뿐이라 60% 에서 막힘

trade_uid 95%
값으로 확인된 FK 라 높게

seller_type	int NOT NULL	distinct 0% null 0% 예: 1 상위 값 1 (299)	판매 주체의 유형 코드(1:플레이어, 2:길드). 이 데이터셋에서는 전부 1(플레이어)로 채워져 있다. 원본 주석 · 판매주체 유형(1:플레이어, 2:길드)	60%
trade_uid market_product_views.trade...	bigint NOT NULL	distinct 100% null 0% 예: 9999999999999999	거래(매물)를 식별하는 고유 ID. market_product_views.trade_uid를 참조하여 해당 상품 매물과 연결되는 실제 조인 키이다. (값이 shape 마스킹되어 실제 값은 확인 불가) 원본 주석 · 거래 유니크 아이디	95%

근거가 부족하면 낮게

40%대 · 미검증
통계 없이 주석만 보고 쓴 설명

→ 이것만 사람이 검토

start_bid	bigint NOT NULL	경매의 시작 입찰가(최저 호가). 원본 주석 · 경매 시작가	42%	미검증
terminated_at	datetime NOT NULL	경매가 실제로 종료 처리된 일시. 원본 주석 · 종료 처리 일시	40%	미검증
updated_at	datetime NOT NULL	레코드가 마지막으로 갱신된 일시. 원본 주석 · 갱신 일시	44%	미검증

전부 보지 않고
낮은 것만 검수

사람이 고친 설명은 다시 돌려도 유지됨

신뢰도는 검수 순서 정답 확률이라기보다, 사람이 먼저 볼 순서를 정하는 값입니다

공개 스키마 블라인드 테스트: 테이블 100% / 컬럼 94%

④ 합치기 · ⑤ 인간의 언어로 연결

테이블을 개념으로 묶고, 개념끼리의 관계를 만들어 Neo4j 에 저장합니다

5-1 테이블을 묶어 대표 개념 만들기 — LLM

- 개념마다 영어 이름 · 한국어 이름 · 동의어 · 설명
- 한 테이블이 여러 개념에 들어가거나, 어디에도 안 들어갈 수 있음

5-2 개념 → 상위 개념 (IS_A) — LLM

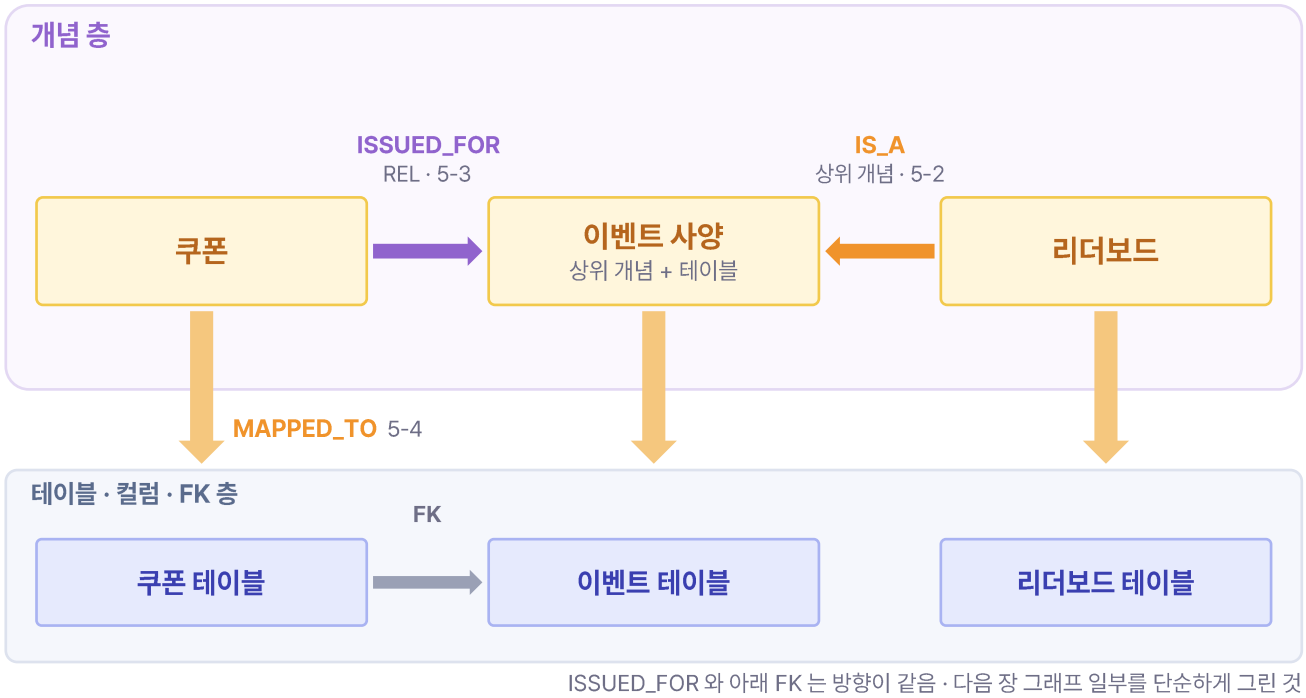
- 분명할 때만 묶음 · 모든 개념이 상위 개념을 갖지는 않음
- 상위 개념도 테이블과 연결될 수 있음 (예: 이벤트 사양)

5-3 개념 → 개념 관계 (REL, 단방향) — LLM · 이름만

- 테이블이 FK로 이어져 있을 때만, 그 테이블들의 개념끼리 관계를 만든다

5-4 Neo4j 에 저장 — 코드

- 개념 층과 그 아래 테이블 · 컬럼 · FK 층을 같이 저장
- 두 층은 MAPPED_TO (개념 → 테이블 · 핵심 컬럼) 로 연결

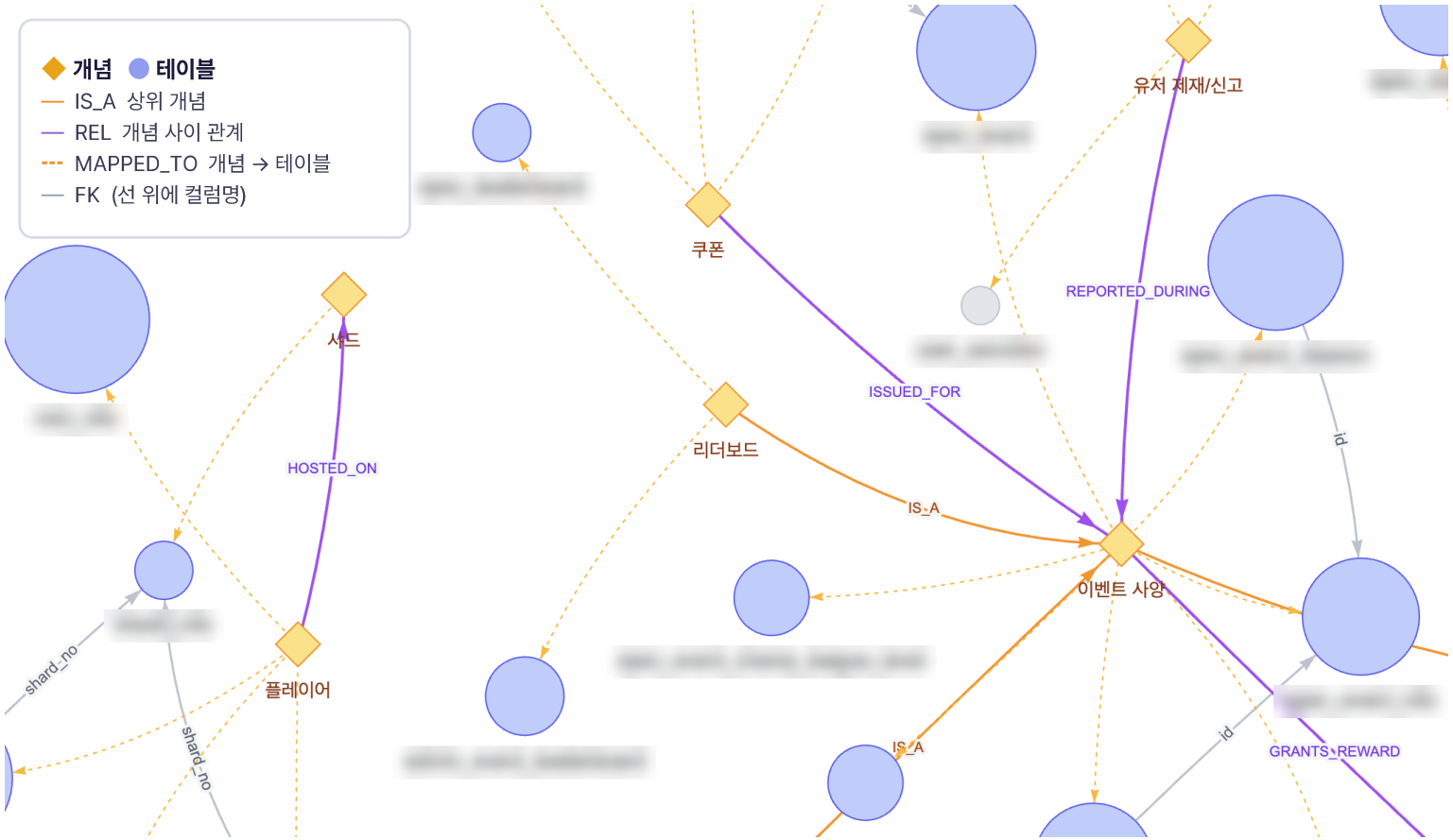


ISSUED_FOR 와 아래 FK 는 방향이 같음 · 다음 장 그래프 일부를 단순하게 그린 것

④ 하나로 합치기 ③ 에서 쓴 설명을 테이블 단위로 합쳐 카탈로그를 만들고, ⑤ 는 이 카탈로그를 보고 개념을 만듭니다

⑥ 그래프 · ⑦ 벡터

그래프는 조인 경로를, 벡터는 관련 있는 테이블을 찾을 때 씁니다



조인 경로 찾기

⑥ neo4j — 그래프

⑤ 에서 만든 개념 층과 테이블 · FK 층이 한 그래프에 들어 있음

관련 테이블 찾기

⑦ pgvector — 벡터

컬럼 문서 설명 · 소속 테이블 설명 앞부분 · FK 대상 · 코드의 뜻

테이블 문서 설명 · 연결된 개념의 한국어 이름 · 동의어

설명을 고치면 그 문서만 다시 임베딩

실제로 저장된 벡터 (일부)

```
[-0.017033733,0.009207424,0.06482026,0.031857684,0.08949616,
,-0.026149083,0.03738214,-0.07955214,-0.0006963114,0.018967291
56,0.06482026,0.029095458,-0.021545371,-0.01740203,0.001945068
186763,-0.058190916,0.035908952,-0.022650262,-0.0011164001,-0
.014179432,0.026701529,0.05782262,-0.0011451733,-0.012153799,
```

원본은 Meta Store (MySQL) · 고친 설명 포함

활용 — Text2SQL

예시 질문: "전사 캐릭터가 많이 가진 아이템은?"

1 검색

벡터 검색 + 개념 매칭,
상위 점수 후보만 남김

2 확장

neo4j에서 최단 조인 경로
추정 FK는 (추정) 표시

3 생성

스키마·조인경로·코드값·
검증 예시로 SELECT 작성

4 실행

이름 대조 → 가드 → 실행
가망 없는 오류면 멈춤

```
SELECT c.code_name AS 아이템명, COUNT(*) AS 보유수
FROM game.character_item ci
JOIN game.character    ch ON ci.char_id   = ch.char_id
JOIN game.code_master c  ON ci.item_code = c.code_id
WHERE ch.job_code = 10    -- 전사
  AND ch.deleted  = 0     -- 삭제 캐릭터 제외
GROUP BY 1 ORDER BY 2 DESC
```

환각이 줄어드는 이유

초록색 JOIN 절은 LLM이 추측한 게 아니라
neo4j에서 받아온 검증된 경로입니다.

조건값은 실제 코드값을 보고 쓰고,
없는 컬럼은 DB에 가기 전에 막습니다.

SELECT 전용(파서 판정) · LIMIT · 999ms 상한

예시로는 실제 결과가 나온 검증 질의 중에서, 이번 질문과 테이블이 많이 겹치는 3개만 넣습니다

정리

게임 QA DB 로 파일럿을 끝까지 돌려봤습니다 (분석부터 Text2SQL 까지)

01

FK 는 값으로 찾는다

FK 선언이 없어도 타입과 실제 값으로
관계를 찾고, 애매한 건 보류

02

신뢰도로 검수 범위를 줄인다

설명마다 신뢰도를 붙여서,
사람은 낮은 것만 검토

03

순서는 코드, 판단은 LLM

고정 파이프라인 · 읽기 전용 · 같은 입력이면
같은 순서로 실행

가능성

- 데이터 추출 요청을 담당자가 직접 처리해서, DBA 를 거치는 일을 줄임
- 사내 에이전트나 Claude Code 가 MCP 로 조인 경로 · 테이블 의미를 조회
- 스키마를 바꿀 때 어떤 개념 · 질의에 영향이 가는지 미리 확인

다음 단계

- 다시 분석할 때 이전 구조와 비교해서, 바뀐 테이블만 다시 분석
- 사람이 고친 설명은 유지하고, 바뀐 부분은 알림으로
- 검수가 끝난 설명은 DB 주석에도 반영해서 스키마 자체를 최신화

사람 머릿속에 있던 DB 명세를 온톨로지로 남기는 것 — db2doc, 카카오게임즈 DB운영셀

GAMES ON AWS 2026

aws FOR GAMES

AI 에이전트로 재정의한 DB 운영

판단은 DBA가, 조회는 에이전트가

곽은경

DBA

카카오게임즈

GS Neotek | MEGAZONE CLOUD Transform · Tomorrow · Together

아젠다

DBA 업무 AI 에이전트 적용 과정

1 질문

왜 시작했나

업무를 먼저 정의했다

2 구조

프로세스의 구성 흐름

통제 수단이 계층마다 다르다

3 하네스- 4가지

① **순서**
→ 그래프

② **기준**
→ 설정

③ **금지**
→ 도구 목록

④ **반복**
→ 게이트 · 캐시

4 결론

구조로 옮기지 못한 것

= 사람의 자리

프롬프트에 두면 확률적, 구조에 두면 결정적 — 오늘 발표를 꿰는 한 문장

왜 시작했나

1 질문
AI가 DBA 업무를 대신할 수 있을까? 어디까지 가능할까?

2 정의
DBA 업무를 유닛으로 분류

3 타깃
의도는 업무 외 시간의 알람과 이슈 해결

결론 먼저

AI가 DBA를 대신한 게 아니다.

업무 규칙을 모델이 어길 수 없는 구조로 옮기고, 비용과 시간 안에서 돌게 했다.

옮기지 못한 셋 — 기준 · 승인 · 조치 — 이 사람의 자리로 남았다.

프로덕션 적용률 — DBA 업무 29개 중 4개

14%

업무를 먼저 정의했다

정의가 곧 설계가 됐다

A-유닛
물리·논리 설계
Sizing · Capacity · 파라미터

A-유닛
운영 관리
계정·권한 · Object · 용량

A-유닛
성능 관리
SQL 튜닝 · 인덱스 · 아키텍처

A-유닛
백업·복구
전략 수립 · 복구 테스트

A-유닛
관제·감시
24/7 Slack 알람 · 대시보드

A-유닛
ETL
추출·이관·가공 · 파이프라인

A-유닛
운영툴 개발
DBGW · 자동화 도구

K01~K16
Kafka 운영
16노드 · 커넥터 · 컨슈머 랙

DBA 29개 (A01~A29) · Kafka 16개 (K01~K16) · 유닛 U1~U8 — 전부 리포에 실재 1차 타깃: 업무 외 시간 알람의 1차 분석

프로세스의 구성

Slack → 라우터 → 에이전트 → mcp_router — 에이전트가 아는 주소는 하나



라우터 뒤에서 데이터를 가져오는 것

자체 MCP 도구 19종 + 서드파티 stdio

서드파티는 필요한 도구만 노출 — 에이전트는 이 중 화이트리스트에 있는 것만 본다

틀 - 하네스

공통 층과 하지 않는 일

모든 에이전트는 base_agent를 상속한다 — 규칙은 공통 층에 한 번만 만들고, 업무별 에이전트가 그대로 물려받는다



※ 판정 기준만은 물려주지 않는다 — 에이전트마다 따로 둔다 (Kafka 기준 ≠ DB 기준)

층마다 하지 않는 일

에이전트

자격증명을 갖지 않는다

DB 드라이버 · AWS SDK import 0

라우터

임계값을 모른다

판정 기준은 DBA의 설정이 정한다

도구

판정하지 않는다

수치만 올린다 · 실패와 0건을 구분

① 순서 — LangGraph 고정 엣지

SINGLE vs PIPELINE

single

대화형 · 탐색

구현	`LangGraph create_react_agent`
순서	LLM이 매 스텝 결정
프롬프트	탐색용 — "사고 순서이지 정해진 양식이 아니다"
종료	모델이 끝났다고 할 때
모델	대부분 standard

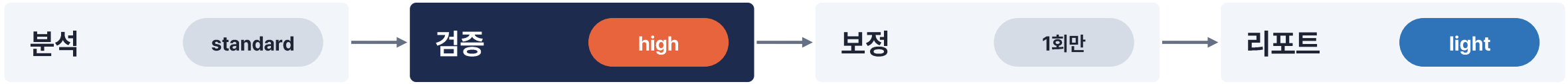
pipeline

분석 · 리포트

구현	`LangGraph StateGraph 4노드`
순서	분석 → 검증 → (보정) → 리포트 고정
프롬프트	단계별 고정 프롬프트
종료	그래프 끝에서
모델	검증 high · 리포트 light

① 순서 — LangGraph 고정 엣지 이점

LLM에게 맡기지 않아도 되는 결정을 코드로



1 검증을 건너뛸 수 없다

분석 → 검증이 엣지로 묶여, 모든 답이 검증을 거친다

ReAct는 모델이 "충분하다"고 하면 검증 없이 끝난다

2 단계마다 모델·프롬프트를 따로

검증 = high · 리포트 = light · 실패해도 등급을 내리지 않는다

검증 1회 ≈ 7K · 리포트 1회 ≈ 8K — 작고 예측 가능

3 답의 형태가 일관된다

DBA의 순서(근거 수집 → 판정 → 정리)가 그대로 고정

같은 유형의 질문에 같은 순서·형식 — 비교하고 쌓아 두기 좋다

4 디버깅·테스트가 노드 단위

틀린 답이 어느 노드에서 틀렸는지 로그로 바로 갈린다

ReAct는 경로가 매번 달라 "어디서 틀렸나"를 재현하기 어렵다

② 기준 — 기준은 설정에

알람은 "지금 봐라", 에이전트의 기준은 "얼마나 위험한가"

Prometheus · Loki
알람 임계치

→

Alertmanager
일률적 기준으로 발송

→

Slack 채널
`{alert_name} : {db_id}`

→

에이전트 1차 분석
리소스·노드별 기준으로 재판정

알람 임계값은 에이전트가 모른다 — 알럿 메시지 첫 줄만 파싱

thresholds.yaml 실제 설정

```
# 우선순위: consumer_groups > topics > default
consumer_groups:
  # S3 싱크 — 배치 적재라 렉이 상시 수십만
  GroupA:
    { info: 500000, warning: 2000000, critical: 5000000 }
  # 정산 — 09-18 1,501,290 / 10-04 833,825
  GroupB:
    { info: 500000, warning: 2000000, critical: 4000000 }
  # 현재 3,658 — default에서도 info. 축만 잡아 둔다
  GroupC:
    { info: 50000, warning: 200000, critical: 1000000 }
```

→ 오른쪽 결과

실환경 실행 결과 `kafka_lag_risk` · **GroupA** `threshold_source: file`

총 렉

891,373건 → **info** 위험 아님

파티션 최대 48,556건 (20개 파티션) · 이 그룹·토픽 전용 임계값이 매칭됨

89만 건 → GroupA 기준 info 구간 (50만 이상 · 200만 미만)

기준은 코드 저장소 밖 `thresholds.yaml` · `policies/sentinel` → **재배포 없이 DBA가 고친다** · 판정 기준 출처 로그: file 15회 · builtin 0건

③ 금지 — 단일 진입점과 화이트리스트

"쓰지 마라"고 말하지 않았다. 부를 수 있는 목록에서 뺐다

1

통로는 하나

아는 주소는 `MCP_ROUTER_URL` 하나

DB 드라이버 · AWS SDK import **0개**

자격증명은 라우터 프로세스에만

→

2

목록에 있는 것만

화이트리스트 `tool_targets.py`

dba 에이전트는 Kafka 도구를 아예 못 본다

env 없으면 미노출 · 쓰기 도구는 없다

→

3

그래서 36일 동안

도구 호출 4,301회 중 쓰기 **0건**

모르는 DB 알럿 **45건** 분석 전 차단

모르는 대상은 추측하지 않는다

LLM에게 쓰기와 자격증명은 "하면 안 되는 일"이 아니라 없는 일이다.

④ 반복 — confirm 게이트

모델은 분류만, 비싼 길로 갈지는 코드가



0건
pipeline 직행
명시 키워드 외의 PIPELINE은 전부 확인을 거친다

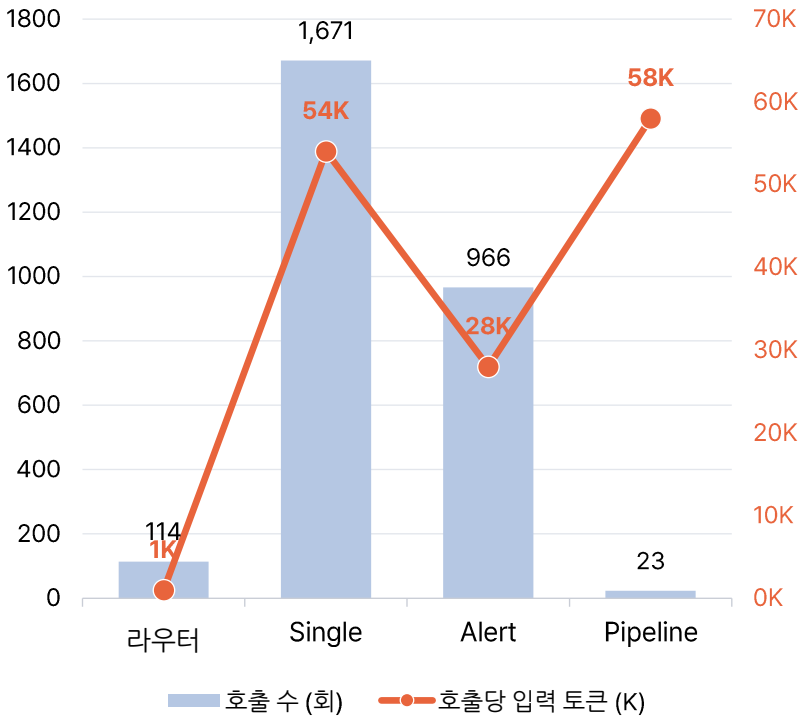
2 / 12
사람이 "예"한 것만 통과
10건은 single로 되돌아갔다

185토큰
라우터 판정 1회
과잉 PIPELINE 판정을 허락한 비용

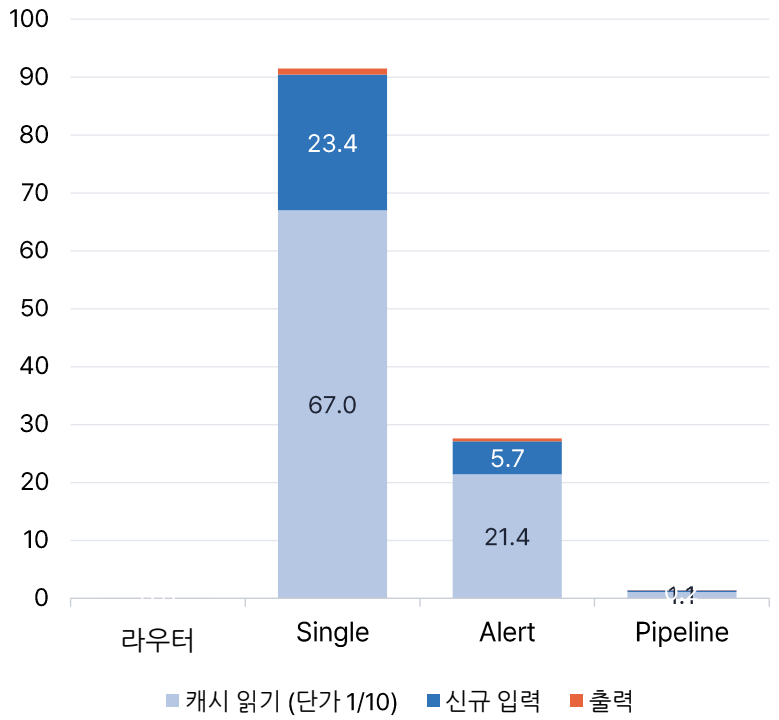
④ 반복 — 입력 토큰 캐시

비싼 건 모드가 아니라 루프다

호출 수(막대) vs 호출당 입력 토큰(선)



토큰 (백만) — 캐시 읽기 75 · 79 · 83%



운영 가능 조건 — 비용과 시간

16회 / 요청

Single 54K × 1,671 · Pipeline 58K × 23

1/3

실효 과금 입력 32% — 캐시 읽기 75.7%

14% / 60초

호출의 14%가 끊긴 연결에서 60초 소모 → 라우터 타임아웃 60→10초

스텝 상한 — 아직 없음

레슨런 — 구조로 옮기지 못한 것이 사람의 자리로 남았다

가설·실측은 에이전트, 판단·조치는 DBA

업무 개선 — 가설·실측 시간을 AI에 맡겼다

기존	알람 수신	가설 수립	실측 (메트릭·쿼리)	판단	조치
지금	알람 수신	가설·실측 → 에이전트 근거(도구명·수치·시간창)까지		판단 → DBA	조치 → DBA

가장 오래 걸리던 구간 = 장애 자체가 아니라 근거를 모으는 시간 → 이 구간이 줄어 DBA는 판단·조치에 집중

구조로 옮기지 못한 세 가지 = 사람

기준

"무엇이 문제인가"는 YAML에 쓰기 전에 사람이 정한다

승인

게이트가 묻고 사람이 답한다 — 12번 중 10번 "아니오"

조치

실행 도구를 만들지 않았다 — 권고까지 에이전트, 실행은 사람

aws FOR GAMES

Thank you

AI가 DBA를 대신하진 못했습니다.
대신 DBA가 새벽에 깨는 횟수를 줄였습니다.

GS Neotek | MEGAZONE CLOUD
Transform · Tomorrow · Together

Appendix — 하네스: 규칙이 놓인 자리

요청이 내려가고 응답이 올라오는 길 위에, 네 규칙이 각각 어느 박스에 있는가

① 순서 규칙 → 그래프 엣지 | ② 기준 규칙 → DBA의 YAML | ③ 금지 규칙 → 도구 목록 밖 | ④ 반복 규칙 → 게이트 · 캐시

색 = 규칙. 아래 각 박스가 어느 규칙을 구현하는지 색으로 표시

요청 경로 (내려감)

④ router(SINGLE/PIPELINE) → 코드가 confirm으로 강등
LLM은 두 단어만 · 명시 키워드 외 PIPELINE은 사람이 "예"해야 진입

↓

① 지침 프롬프트 — md 48개 (LLM 투입 37)
_common.md가 매 호출 동일 접두사 → 캐시 읽기 75.7% (④)

↓

에이전트 — single(ReAct) / pipeline(StateGraph 엣지)
분석 → 검증 → 리포트 순서는 엣지가 강제 · 단계별 티어는 코드, 티어→모델은 환경변수

↓

③ 에이전트 층 통제
노출 화이트리스트 · 미노출 호출 감지 · 승인 게이트

↓

mcp_router — 단일 진입점
자격증명 보유 · DB 주소·포트는 응답에 담지 않는다

↓

③ 도구 층 통제 — 자체 MCP 19종
운영 쓰기 19 → 1 · 인자 경계 · 범위 거부 · read-only 강제

→

응답 경로 (올라옴)

데이터 소스 → 수치 · 집계만
도구는 판정하지 않는다 — 실패와 0건을 구분

↑

② 판정 기준 — DBA가 정한다
thresholds.yaml · 정책 YAML · 베이스라인 — 모르면 "확인 못 함"

↑

①·② 검증 — high 모델
근거 없음 · 단정 · 모순 → 보정 1회 · 순서는 엣지가 강제

↑

Slack 답변
도구명 + 수치 + 시간창

↑

실행 로그 — 모든 규칙의 결함을 발견한 곳
LLM 2,774회 · 도구 4,301회 (36일) — 이 그림이 맞는지는 로그로만 확인된다