

기술

같은 데이터로 지식그래프를 두 번 만들면, 다른 그래프가 나옵니다

방동걸 (주)웨이스 대표이사

2026년 9월 12일

지식그래프를 쓰는 검색 증강 생성(Graph RAG)을 붙이려고 할 때, 대부분의 안내는 이렇게 시작합니다. 문서를 언어 모델에 넣고 개체와 관계를 뽑아 그래프를 만들라는 것입니다. 웨이스가 VEXPLOR에 넣을 그래프를 공장 데이터로 만들다 처음 막힌 문제에서 시작한 글입니다.

그 방식에는 두 가지 문제가 따라옵니다. 하나는 같은 문서를 두 번 넣으면 다른 그래프가 나온다는 것이고, 다른 하나는 그래프를 다 만들 때까지 몇 시간에서 며칠이 걸린다는 것입니다.

두 번째 문제가 특히 무겁습니다. 인덱싱이 끝나야 질문할 수 있다면, 방금 들어온 값에 대해서는 물어볼 수 없습니다. 주문이 조금 전에 바뀌었는지, 설비가 5분 전에 멈췄는지를 묻는 상황에서 이 지연은 기능 자체를 없앱니다.

이 글은 그 두 문제를 그래프를 만드는 시점을 옮겨서 푸는 구성을 정리한 것입니다.

01

확률적으로 뽑은 그래프가 흔들리는 이유

언어 모델로 개체와 관계를 뽑는 방식은 본질적으로 추정입니다. 같은 입력에도 실행마다 결과가 달라질 수 있고, 원문에 없는 관계가 생기기도 합니다. 이것을 그래프 환각이라고 부릅니다.

문장 요약이라면 표현이 조금 달라져도 넘어갈 수 있습니다. 그런데 그래프는 다릅니다. 관계 하나가 잘못 생기면 그 관계를 타고 가는 모든 질의의 답이 바뀝니다. 틀린 문장은 한 문장이 틀리고, 틀린 엣지는 그 뒤를 전부 틀리게 합니다.

여기에 더 근본적인 문제가 하나 있습니다. 기업 데이터는 대부분 이미 관계형 데이터베이스에 구조화돼 있습니다. 주문과 고객은 이미 외래키로 연결돼 있고, 설비와 공정도 이미 표로 묶여 있습니다. 그런데 앞의 방식은 그 구조를 비정형 텍스트로 풀어서 다시 언어 모델에 넣고 관계를 추정하게 합니다.

이미 확정된 관계를 지워서 추정 문제로 바꾸는 셈입니다. 연산도 낭비지만, 원본에 있던 무결성도 함께 잃습니다.

02

그래프를 읽을 때 만들지 않고, 쓸 때 만든다

여기서 시점을 옮길 수 있습니다. 데이터가 들어온 뒤에 그래프를 추출하는 대신, 데이터가 들어오는 그 순간에 그래프를 함께 만듭니다.

거래 한 건이 발생하면 관계형 데이터베이스에 한 번 저장하고, 같은 트랜잭션 안에서 지식그래프에도 노드와 엣지로 한 번 더 저장합니다. 두 저장 명령을 **동기적으로** 실행합니다.

이렇게 하면 두 문제가 동시에 사라집니다.

문제	왜 사라지는가
그래프 환각	추정이 없다 — 규칙이 변환한다
인덱싱 지연	별도 인덱싱 단계가 없다
무결성 훼손	원본을 텍스트로 풀지 않는다

추정이 없으니 두 번 만들어도 같은 그래프가 나옵니다. 그리고 쓰기가 끝난 시점에 이미 그래프가 있으니, 방금 들어온 값을 그 자리에서 물어볼 수 있습니다.

그러면 규칙은 누가 만드는가

이 방식이 성립하려면 「무엇을 노드로 하고 무엇을 엣지로 할 것인가」가 미리 정해져 있어야 합니다. 그 규칙을 **메타모델**이라고 부르고, 자연어로 적은 요구사항에서 개체를 뽑아 만듭니다.

자연어 처리는 **규칙을 만들 때 한 번만** 쓰입니다. 설계 단계에서 쓰고, 데이터가 들어올 때마다 쓰지 않습니다. 이것이 시점을 옮긴다는 말의 실제 내용입니다.



데이터가 들어온 뒤 따로 정리하지 않고, 저장되는 그 순간에 규칙으로 함께 만듭니다.

03

태그 네 가지로 탐색 범위를 미리 정한다

메타모델에서 각 개체에는 의미 태그가 붙습니다. 태그는 장식이 아니라 데이터 처리 규칙이고, 넷이 각각 다른 일을 합니다.

태그	정하는 것
구조	데이터를 노드·엣지로 바꾸는 규칙
검색	탐색 진입점과 우선순위
로직	따라가도 되는 연결
보안	민감 정보의 노출 범위

구조 태그가 앞에서 말한 변환 규칙입니다. 같은 거래 데이터라도 무엇을 노드로 세울지는 도메인마다 다르고, 그 결정이 여기 적힙니다.

검색 태그는 그래프 탐색을 어디서 시작할지 정합니다. 여기에 재미있는 설계가 하나 들어갑니다 — 진입점으로 지정된 개체의 속성 값만 임베딩 벡터로 바꿔 노드 속성에 함께 저장합니다. 그래서 질의가 들어오면 벡터 유사도로 시작점을 찾고, 그다음부터는 그래프를 따라갑니다.

벡터 검색과 그래프 검색을 양자택일로 놓지 않는다는 뜻입니다. 벡터는 「어디서 시작할까」에 강하고 그래프는 「그래서 무엇과 연결돼 있나」에 강한데, 각자 잘하는 일만 맡깁니다.

로직 태그는 따라가도 되는 연결을 미리 정합니다. 그래프 탐색은 범위를 안 정하면 따라갈 것이 금세 불어나는데, 도메인 규칙상 의미 없는 방향을 처음부터 빼 둡니다.

보안 태그가 마지막입니다. 서브그래프를 만든 뒤 이 태그가 붙은 속성을 **마스킹**하고 나서 언어 모델에 넣습니다. 즉 **원래 값이 모델에 들어가지 않습니다**. 모델에게 「말하지 말라」고 부탁하는 것과, 애초에 주지 않는 것은 다른 일입니다.



태그 넷이 각각 변환 규칙·진입점·따라갈 연결·가릴 값을 정합니다.

04

덤으로 따라오는 것 — 토큰이 줄어듭니다

탐색 범위를 미리 정하면 비용 쪽에서도 이득이 생깁니다.

그래프 탐색에 범위 제한이 없으면 질의 하나에 관련 없는 노드까지 딸려 옵니다. 그렇게 만들어진 서브그래프를 그대로 언어 모델에 넣으면 **읽지 않아도 될 내용에 토큰이 나갑니다**. 입력이 길어지니 응답도 느려집니다.

로직 태그가 허용한 연결만 따라가면 질의와 관련된 부분만 남습니다. **모델에 넣는 입력이 줄어드니 비용이 줄고 응답이 빨라집니다**. 검색 품질을 위해 만든 제한이 비용도 함께 낮춥니다.

한 가지 덧붙이면, 벡터 검색이 약한 곳이 여기서도 드러납니다. 벡터 유사도는 의미가 가까운 것을 찾는 도구라, 「3개월 안에 계약이 끝나면서 아직 정산이 안 된 건」처럼 조건을 만족해야 하는 질의에는 잘 듣지 않습니다. 그 조건은 유사도가 아니라 구조로 판정할 문제이기 때문입니다. 관계가 이미 그래프에 확정돼 있으면 그런 질의는 탐색으로 풀립니다.

05

그래프를 만드는 순서를 정리하면

1. 자연어 요구사항에서 개체를 뽑아 메타모델을 만들고 태그를 붙인다 (설계 시점, 한 번)
2. 거래가 발생하면 관계형 데이터베이스와 지식그래프에 동시에 쓴다 (운영 시점, 매번)
3. 질의가 오면 벡터로 진입점을 찾고 로직 태그가 허용한 연결만 따라가 서브그래프를 만든다
4. 보안 태그가 붙은 속성을 가린 뒤 언어 모델에 넣어 답을 만든다

언어 모델이 하는 일이 4번으로 좁혀졌습니다. 그래프를 만드는 2번과 탐색 범위를 정하는 3번에는 추정이 들어가지 않습니다. 모델이 잘하는 일(말을 이해하고 말을 만드는 것)에만 모델을 쓰고, 규칙이 잘하는 일에는 규칙을 둡니다.



추정이 들어가는 단계를 양끝으로 좁혔습니다.

06

동시에 저장한다면, 먼저 정해야 할 것 셋

여기까지 읽으면 바로 떠오르는 질문이 있습니다. 두 저장소에 동시에 쓰면 느려지지 않는가, 한쪽만 실패하면 어떻게 되는가입니다. 구성을 잡을 때 먼저 정해야 하는 것이 셋입니다.

첫째, 한쪽이 실패했을 때의 규칙입니다. 관계형 데이터베이스에는 들어갔는데 그래프 쪽이 실패하면 둘이 어긋납니다. 어긋난 그래프는 틀린 답을 일관되게 내놓아 발견이 늦어질 수 있습니다. 같은 트랜잭션으로 묶어 함께 되돌릴지, 그래프 쪽을 뒤따라 맞출지를 먼저 정합니다. 되돌리기로 정했다면 원본 쓰기가 그래프 쓰기에 묶인다는 뜻이므로, 그 대가를 알고 선택해야 합니다.

둘째, 쓰기 지연을 어디까지 받아들일지입니다. 저장이 한 번에서 두 번으로 늘어나니 쓰기는 분명히 느려집니다. 대신 읽기 쪽에서 인덱싱 시간이 통째로 사라집니다. 비용을 쓰기 쪽으로 옮기는 구성이라고 보는 편이 정확하고, 쓰기가 초당 수만 건 들어오는 환경이라면 이 교환이 맞지 않을 수 있습니다.

셋째, 어디까지 그래프로 만들지입니다. 모든 엔티티와 모든 외래키를 노드·엣지로 옮기면 그래프가 원본만큼 커지고 탐색도 느려집니다. 질의에 실제로 쓰이는 관계만 고르는 판단이 필요하고, 이 판단이 곧 구조 태그를 설계하는 일입니다.

셋 다 「자동으로 되는」 항목이 아닙니다. 이 방식이 편의를 주는 대신 요구하는 것이 바로 이 결정들입니다.

07

이 구성이 답하지 못하는 것

이 방식에는 값을 치러야 하는 곳이 있습니다.

- **메타모델을 먼저 만들어야 합니다.** 문서를 던져 넣으면 알아서 그래프가 생기는 편의는 없습니다. 무엇을 노드로 할지가 요구사항에 들어 있어야 하고, 그 설계가 틀리면 확정적으로 틀린 그래프가 쌓입니다.
- **관계가 이미 구조화돼 있는 데이터에 맞습니다.** 계약서 더미나 회의록처럼 관계가 글 안에만 있는 데이터라면 결국 추출이 필요합니다.
- **스키마가 바뀌면 메타모델도 바뀌야 합니다.** 규칙 기반의 대가입니다.

즉 이것은 모든 Graph RAG를 대체하는 방식이 아니라, 이미 데이터베이스에 관계가 있고 방금 들어온 값을 물어야 하는 환경에 맞는 구성입니다. 제조·물류·금융처럼 거래가 계속 발생하는 쪽이 그렇습니다.

08

지금 쓰는 Graph RAG에서 확인해 볼 세 가지

지금 Graph RAG를 검토 중이시라면 이 셋을 먼저 재 보시길 권합니다.

1. 같은 데이터로 그래프를 두 번 만들면 같은 그래프가 나오니까? 다르다면 그 차이가 답을 바꿀 수 있는지 봅니다.
2. 데이터가 들어오고 나서 질문할 수 있기까지 얼마나 걸립니까? 그 지연이 업무가 요구하는 시간보다 길면, 검색 품질을 올려도 쓰이지 않습니다.
3. 언어 모델에 들어가는 내용에 가려야 할 값이 섞여 있습니까? 가리는 일을 모델에게 맡기고 있는지, 넣기 전에 빼고 있는지 확인합니다.

여러분의 시스템에서는 그래프가 언제 만들어집니까?

WACE(웨이스)는 공장이 스스로 판단하고 멈추지 않게 하는 제조 AI 운영체제 VEXPLOR를 만드는 회사다.



원문 blog.vexplor.com/post/deterministic-graph-at-write-time

발행 2026년 9월 12일 · 방동걸 (주)웨이스 대표이사



원문



홈페이지



링크드인

회사 주식회사 웨이스 (WACE Inc.) · 대표이사 방동걸 ·
사업자등록번호 439-87-02852

주소 (18462) 경기 화성시 동탄구 동탄영천로 150, 에이동(A동)
1702·1706호 (영천동, 실리콘엘리동탄)

연락처 전화 070-4730-0629 · contact@wace.me

홈페이지 wace.me · linkedin.com/company/wace-inc