

처음 만나는 Neo4j와 GraphRAG

처음 만나는 온톨로지 2권 · 커뮤니티판 · 2026-10-12

0. 이 안내서에 대해

1. 왜 속성 그래프인가

2. Neo4j 띄우기

3. 온톨로지를 속성 그래프로 옮기기

4. Cypher로 묻기

5. 품질 검사를 Cypher로

6. RAG와 GraphRAG

7. GraphRAG 실습

8. Text-to-Cypher

9. 무엇을 재야 하나

10. 흔한 실수 열 가지

11. 다음 걸음

부록 A. 용어

부록 B. 참고 문헌

부록 C. 예제 파일 전문과 실행 결과

이 판에 대해

- 저자: OntoCraft. 라이선스: CC BY 4.0 (<https://creativecommons.org/licenses/by/4.0/deed.ko>). 출처를 밝히면 누구나 나누고 고쳐 쓸 수 있습니다.
- 예제의 「가상 공장 A」와 모든 설비 이름·값은 지어낸 것입니다. 실제 회사나 설비와 관계가 없습니다.
- 책에 실은 실행 결과는 부록 C 의 예제를 Neo4j 5.26, Ollama(qwen2.5:7b, nomic-embed-text)로 실제로 돌려 얻은 것입니다. LLM 의 문장은 모델과 환경에 따라 달라질 수 있습니다.

0. 이 안내서에 대해

0.1 누구를 위한 글인가

이 안내서는 「처음 만나는 온톨로지」(1권)의 다음 권입니다. 1권을 읽은 데이터 실무자를 위해 썼습니다. 엑셀, SQL, 파이썬 정도를 써 본 사람이면 읽을 수 있습니다. 끝까지 읽으면 1권의 작은 온톨로지를 그래프 데이터베이스에 넣고, 질의하고, 로컬 LLM이 그 그래프를 근거로 답하게 만들 수 있습니다.

다루는 순서는 이렇습니다. 왜 속성 그래프가 필요한지(1장), Neo4j를 어떻게 띄우는지(2장), 1권의 온톨로지를 어떻게 옮기는지(3장)를 먼저 봅니다. 그다음 Cypher로 묻고(4장), 품질을 검사합니다(5장). 후반은 LLM입니다. RAG와 GraphRAG가 무엇인지(6장), 실제로 돌리면 무엇이 다른지(7장), LLM에게 질의를 짜게 하면 무엇이 위험한지(8장), 믿을 만한지 어떻게 재는지(9장)를 봅니다.

0.2 1권에서 가져오는 것

예제는 1권과 같은 「가상 공장 A」입니다. 펌프 두 대(P-101, P-102), 모터 한 대(M-101), 베어링 두 개(BRG-101, BRG-102), 고장 모드 「베어링 마모」, 작업 지시 두 건(WO-001, WO-002), 진동 측정 세 건(MS-1~MS-3)이 전부입니다. 이름과 값은 모두 지어낸 것입니다.

1권에서 일부러 넣은 오류 두 개도 그대로 있습니다. 대상 설비가 빠진 작업 지시 WO-002와, 숫자 대신 「높음」이라고 적힌 진동값 MS-3입니다.

이번 권에서는 여기에 정비 일지 여섯 건(N1~N6)을 더합니다. 정비 담당자가 손으로 적은 짧은 글입니다. 예를 들어 N2는 「작업 지시 WO-001에 따라 P-101의 베어링 BRG-101을 교체했다. 교체 뒤 진동은 2.0 mm/s로 내려갔다.」입니다.

1권의 말 네 개는 1권의 뜻 그대로 씁니다. 1권을 읽지 않았다면 이 표만 보고 넘어가도 됩니다.

말	뜻	예
클래스	같은 성질을 가진 것들의 모임	펌프, 베어링
개체	클래스에 속하는 실제 하나	P-101, BRG-101
관계	개체와 개체를 잇는 것	「BRG-101은 P-101의 부품이다」
역량 질문	온톨로지가 답할 수 있어야 하는 업무 질문	「P-101의 부품과 그 고장 모드는?」

0.3 준비물

준비물	하는 일	라이선스
Docker	프로그램을 상자(컨테이너)에 담아 띄웁니다. Neo4j를 띄울 때 씁니다	
Neo4j 5.26 Community	그래프를 저장하고 질의하는 데이터베이스입니다	GPL-3.0
Python 3.11 이상과 uv	예제를 돌립니다(neo4j 드라이버, rdflib)	
Ollama	내 컴퓨터에서 LLM을 돌려 주는 프로그램입니다	MIT
qwen2.5:7b	답을 쓰고 Cypher를 짜는 LLM입니다	Apache-2.0
nomic-embed-text	정비 일지를 숫자 목록으로 바꾸는 모델입니다(6장)	Apache-2.0

모든 것이 이 컴퓨터 안에서 됩니다. 외부 API를 부르지 않으므로 정비 일지가 밖으로 나가지 않습니다. 다른 모델을 써도 예제는 돌지만 답의 문장은 달라집니다.

0.4 예제와 실행 결과

예제 파일은 다섯 개입니다.

파일	하는 일	나오는 장
load.py	1권 온톨로지와 정비 일지를 Neo4j에 넣습니다	3장
queries.py	Cypher로 묻고 품질을 검사합니다	4·5장
graphrag.py	같은 질문을 벡터 RAG와 GraphRAG로 답해 비교합니다	7장
text2cypher.py	LLM이 짠 Cypher를 검사하고 실행합니다	8장
notes.json	정비 일지 여섯 건	3·7장

전문은 부록 C에 있습니다. 책에 실은 출력은 `run_all.sh`로 처음부터 차례로 돌려 저장한 것입니다(2026-10-12, Neo4j 5.26, qwen2.5:7b, 온도 0).

LLM의 답과 벡터 점수는 모델 판과 실행 환경에 따라 조금씩 달라질 수 있습니다. 같은 실험을 반복할 때 점수가 소수 셋째 자리에서 달라지는 것을 보았습니다.

0.5 표기

- 용어는 처음 나올 때 「영어 — 한국어: 설명」 순서로 한 번 적습니다.
- 각 장 끝의 「이 장이 닿지 않는 곳」은 그 장의 설명이 다루지 않는 범위입니다.

- 이 안내서는 CC BY 4.0으로 배포합니다. 출처(OntoCraft)를 밝히면 누구나 나누고 고쳐 쓸 수 있습니다.

1. 왜 속성 그래프인가

1.1 이 책을 끌고 가는 질문

가상 공장 A의 정비 담당자가 이렇게 묻는다고 합시다. 「P-101의 베어링을 교체한 뒤 진동은 얼마였고, 그 펌프를 돌리는 모터는 무엇인가?」

답은 두 곳에 흩어져 있습니다. 교체 뒤 진동 2.0mm/s는 정비 일지 N2에 글로 적혀 있습니다. 「M-101이 P-101을 돌린다」는 어느 일지에도 없고 1권 온톨로지의 관계에만 있습니다. 글과 관계를 함께 봐야 답할 수 있습니다.

이 안내서는 이 질문 하나에 답하는 길을 따라갑니다. 관계를 담은 그래프를 만들고(1~5장), 글과 관계를 함께 LLM에게 주어 답하게 합니다(6~7장). 7.2절에서 실제 답을 봅니다.

1.2 같은 사실, 두 가지 그림

1권은 모든 사실을 주어·술어·목적어 세 칸의 문장(트리플)으로 적었습니다. 「MS-2는 P-101에서 잤다」도, 「MS-2의 진동은 7.4다」도 똑같이 화살표 하나입니다. 값 7.4도 화살표 끝에 매달린 점이 됩니다.

속성 그래프는 다르게 그림니다. 진동값과 측정 시각처럼 그 대상에만 붙는 값은 노드 안의 칸에 적습니다. 화살표는 대상과 대상 사이에만 남깁니다. 그림 1이 같은 사실을 두 방식으로 그린 것입니다.

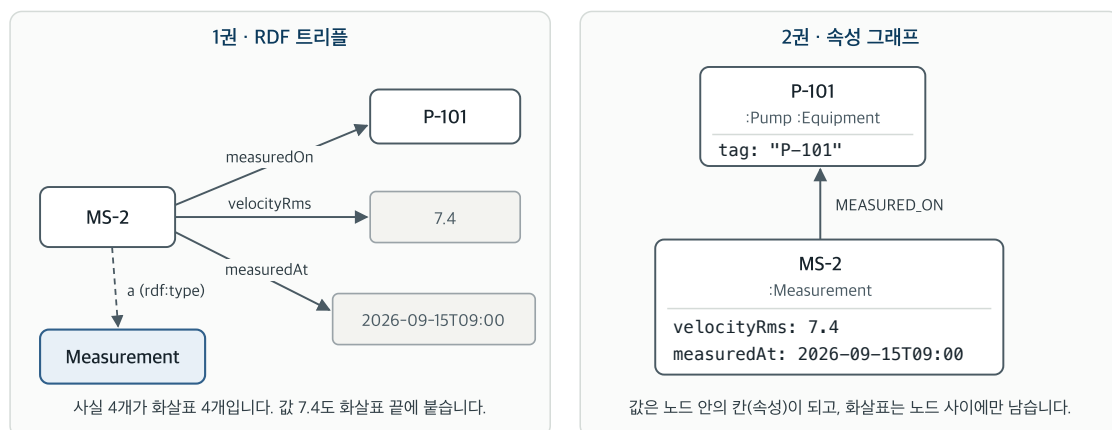


그림 1. 같은 사실을 두 방식으로 적은 그림. 왼쪽 RDF에서는 사실 네 개가 화살표 네 개입니다. 오른쪽 속성 그래프에서는 값이 노드 안의 칸(속성)이 되고, 화살표는 MS-2에서 P-101로 가는 하나만 남습니다.

오른쪽 그림이 엑셀에 익숙한 사람에게 더 친숙합니다. 노드 하나가 표의 한 줄과 비슷하고, 칸 이름(velocityRms)과 값(7.4)이 그 줄에 들어 있습니다. 다른 점은 줄과 줄 사이를 화살표로 직접 잇는다는 것입니다.

1.3 네 가지 부품

Property graph — 속성 그래프: 노드와 관계에 「칸 이름: 값」 쌍을 바로 붙이는 그래프 모델입니다.

부품은 네 가지입니다. 노드를 카드 한 장이라고 생각하면 쉽습니다.

부품	카드에 비유하면	뜻	가상 공장 A 예
Node — 노드	카드 한 장	대상 하나	펌프 P-101, 정비 일지 N2
Label — 레이블	카드에 붙인 색 스티커	노드의 종류. 여러 개 붙일 수 있습니다	<code>:Pump</code> , <code>:Equipment</code>
Relationship — 관계	카드 사이의 화살표	두 노드를 잇는 방향 있는 선. 이름(유형)이 하나 있습니다	M-101 → P-101 「구동한다」 (DRIVES)
Property — 속성	카드에 적은 칸	노드나 관계에 붙인 「칸 이름: 값」	<code>velocityRms: 7.4</code>

레이블 앞의 콜론(`:Pump`)은 「이것은 레이블입니다」라는 표시입니다. 이 안내서에서 `:Pump` 라고 쓰면 「펌프 스티커」로 읽으면 됩니다.

1.4 1권의 말과 짝 맞추기

1권에서 쓴 말은 속성 그래프에서 이렇게 바뀝니다.

1권(RDF)	2권(Neo4j)	웁길 때 할 일
IRI(전역 이름)	노드의 <code>id</code> 속성	같은 <code>id</code> 가 두 번 생기지 않게 막습니다(3.5절)
<code>rdf:type</code> (종류)	레이블	「펌프는 설비」 같은 계층은 레이블을 여러 개 붙여 펼칩니다(3.2절)
관계	관계 유형	이름 규칙을 정합니다. <code>partOf</code> 는 <code>PART_OF</code> (3.3절)
값 속성	노드 속성	숫자는 숫자로, 날짜는 날짜로 넣습니다
RDFS·OWL 추론	없음	적재할 때 미리 펼쳐 둡니다(3장)
SHACL 검증	없음	위반을 찾는 질의를 따로 씁니다(5장)

아래 두 줄이 가장 중요합니다. 1권에서는 「펌프는 설비다」라고 적어 두면 추론기가 P-101을 설비로도 세어 주었습니다. Neo4j에는 그런 추론기가 없습니다. 적어 둔 그대로만 찾습니다. 1권에서는 SHACL로 「작업 지시에는 대상 설비가 있어야 한다」 같은 약속을 검사했습니다. Neo4j에는 이것도 없습니다. 그래서 3장과 5장에서 이 두 빈자리를 직접 메웁니다.

1.5 Neo4j와 Cypher

Neo4j: 가장 널리 쓰이는 속성 그래프 데이터베이스입니다. 이 안내서는 무료로 쓸 수 있는 Community 판을 씁니다.

Cypher — 사이퍼: Neo4j에서 그래프에 묻는 언어입니다. SQL이 표에 묻는 언어라면 Cypher는 그래프에 묻는 언어입니다. 국제 표준 그래프 질의 언어 GQL(ISO/IEC 39075:2024)과 문법이 가깝습니다. 4장에서 자세히 봅니다.

확인 문제

문제 1. 그림 1의 오른쪽에서 진동값 7.4는 어디에 들어 있습니까? 왼쪽과 무엇이 다르니까?

답. MS-2 노드 안의 칸(velocityRms 속성)에 들어 있습니다. 왼쪽 RDF에서는 7.4가 화살표 끝에 매달린 점이었습니다. 속성 그래프에서는 값이 노드 안으로 들어가고 화살표는 노드 사이에만 남습니다.

문제 2. 1권에서 「펌프는 설비다」라고 적어 두었습니다. Neo4j에서 「설비를 모두 보여 줘」라고 물으면 P-101이 저절로 나오니까?

답. 나오지 않습니다. Neo4j에는 추론기가 없어 「P-101은 펌프」와 「펌프는 설비」를 이어 읽지 않습니다. 그래서 3장에서 적재할 때 P-101에 설비 레이블을 미리 함께 붙입니다.

이 장이 닿지 않는 곳: RDF와 속성 그래프 가운데 무엇이 낫다고 말하지 않습니다. 추론과 표준 어휘가 중요하면 RDF, 여러 단계를 따라가는 질의와 개발 속도가 중요하면 속성 그래프가 편합니다. 어느 쪽이든 1권의 온톨로지(개념과 관계의 약속)가 먼저입니다.

2. Neo4j 띄우기

2.1 왜 따로 띄우나

회사 컴퓨터에는 이미 다른 프로젝트의 Neo4j가 돌고 있을 수 있습니다. 같은 Neo4j에 실습 데이터를 넣으면 두 가지 일이 생깁니다. 레이블 이름이 겹쳐 남의 데이터와 섞이고, 실습 스크립트가 데이터를 지울 때 남의 것까지 지웁니다.

그래서 실습용 Neo4j를 컨테이너 하나로 따로 띄웁니다. 다 쓰면 컨테이너째 지웁니다. 다른 데이터에는 손이 닿지 않습니다.

2.2 명령 한 줄

```
docker run -d --name onto-guide-neo4j \
  -p 7475:7474 -p 7688:7687 \
  -e NE04J_AUTH=neo4j/guide-demo-pass \
  neo4j:5.26-community
```

읽는 법은 이렇습니다.

부분	뜻
<code>docker run -d</code>	컨테이너를 띄우고 뒤에서 돌게 둡니다
<code>--name onto-guide-neo4j</code>	컨테이너에 이름을 붙입니다. 지울 때 이 이름을 씁니다
<code>-p 7475:7474</code>	브라우저 화면 포트입니다. 내 컴퓨터의 7475를 컨테이너의 7474에 잇습니다
<code>-p 7688:7687</code>	프로그램이 접속하는 포트입니다. 내 컴퓨터의 7688을 컨테이너의 7687에 잇습니다
<code>-e NE04J_AUTH=neo4j/guide-demo-pass</code>	사용자 이름 <code>neo4j</code> , 비밀번호 <code>guide-demo-pass</code> 로 정합니다
<code>neo4j:5.26-community</code>	쓸 Neo4j 판입니다

Neo4j의 기본 포트는 7474와 7687입니다. 이미 다른 Neo4j가 이 포트를 쓰고 있으면 부딪치므로, 내 컴퓨터 쪽은 7475와 7688로 비켜 둡니다.

비밀번호 `guide-demo-pass`는 실습용입니다. 실제 데이터를 넣는다면 바꾸고, 예제에는 환경 변수 `NE04J_PASSWORD`로 넘깁니다.

2.3 브라우저로 들여다보기

컨테이너가 뜨면 브라우저에서 `http://localhost:7475` 를 엽니다. 위 비밀번호로 들어가면 Neo4j가 주는 질의 화면이 나옵니다. 3장에서 데이터를 넣은 뒤 이 화면에 `MATCH (n) RETURN n` 을 넣으면 그래프 전체를 그림으로 볼 수 있습니다. 3.6절의 그림 3과 같은 모양이 나옵니다.

예제 프로그램은 브라우저가 아니라 `bolt://localhost:7688` 로 접속합니다. 「bolt」는 Neo4j가 프로그램과 주고받는 통신 방식의 이름입니다.

2.4 다 쓰면 지웁니다

```
docker rm -f onto-guide-neo4j
```

이 한 줄이면 실습 데이터까지 함께 사라집니다. 운영 중인 Neo4j에는 실습 스크립트를 돌리지 않습니다.

확인 문제

문제 1. 실습용 Neo4j를 기본 포트(7474, 7687)가 아닌 7475와 7688로 띄운 이유는 무엇입니까?

답. 이미 다른 Neo4j가 기본 포트를 쓰고 있을 때 부딪치지 않게 하려는 것입니다.

문제 2. 실습 스크립트를 운영 중인 Neo4j에서 돌리면 안 되는 이유는 무엇입니까?

답. 레이블 이름이 겹쳐 남의 데이터와 섞이고, 스크립트가 데이터를 지울 때 남의 것까지 지울 수 있기 때문입니다.

이 장이 닿지 않는 곳: 운영 환경의 백업, 클러스터, 권한 관리, 메모리 설정은 다루지 않았습니다.

3. 온톨로지를 속성 그래프로 옮기기

3.1 그대로 부으면 생기는 문제

load.py는 1권의 plant.ttl과 failure-modes.ttl을 읽어 Neo4j에 넣습니다. 트리플을 하나씩 그대로 부을 수도 있습니다. 그러면 세 가지 문제가 생깁니다.

1. 「설비를 모두」 물으면 펌프가 빠집니다. 1.4절에서 본 대로 Neo4j는 「P-101은 펌프」와 「펌프는 설비」를 이어 읽지 않습니다.
2. 관계 이름이 사람마다 달라집니다. 1권의 partOf를 누구는 PART_OF, 누구는 partof로 옮기면 같은 관계를 두 이름으로 묻게 됩니다.
3. 다른 이름을 잃습니다. 1권은 「실 누설」에 「씰 누수」라는 다른 이름을 적어 두었습니다. 이것을 버리면 「씰 누수」라고 묻는 사람의 질문을 그래프와 잇지 못합니다.

그래서 옮길 때 규칙 세 가지를 지킵니다. 규칙은 모두 load.py 안에 코드로 들어 있습니다. 사람이 그때그때 판단하지 않습니다.

3.2 규칙 1: 계층은 레이블로 펼칩니다

P-101은 펌프이고, 펌프는 설비입니다. 적재할 때 이 사슬을 끝까지 따라가 P-101 노드에 :Pump와 :Equipment를 함께 붙입니다. 이것을 「펼친다」라고 부릅니다.



그림 2. 계층을 레이블로 펼치는 규칙. 왼쪽 RDF에는 「P-101은 펌프」와 「펌프는 설비」가 따로 있습니다. 적재할 때 이 둘을 이어 P-101에 레이블 두 개를 함께 붙여 두면, 설비를 찾는 질의에도 P-101이 나옵니다.

1권에서는 추론기가 질의할 때 이 일을 했습니다. 2권에서는 적재하는 코드가 미리 해 둡니다. 결과는 같지만 시점이 다릅니다. 그래서 온톨로지에 「펌프는 설비」를 새로 적거나 지우면 다시 적재해야 반영됩니다.

3.3 규칙 2: 관계 이름은 대문자와 밑줄로 바꿉니다

Neo4j에서는 관계 유형을 대문자와 밑줄로 쓰는 것이 관례입니다(Neo4j Cypher 매뉴얼의 이름 짓기 권장). 1권의 이름을 이 관례에 맞춰 기계적으로 바꿉니다.

1권	2권	뜻
drives	DRIVES	구동한다
partOf	PART_OF	~의 부품이다
hasFailureMode	HAS_FAILURE_MODE	이 고장 모드를 가질 수 있다
measuredOn	MEASURED_ON	~에서 잤다
onEquipment	ON_EQUIPMENT	대상 설비가 ~이다
addresses	ADDRESSES	이 고장 모드를 다룬다
skos:broader	BROADER	넓은 말이 ~이다

바꾸는 규칙은 `load.py`의 한 줄(snake)에 들어 있습니다. 첫 글자가 아닌 대문자 앞에 밑줄을 넣고 모두 대문자로 만듭니다. `partOf`는 `part_of`를 거쳐 `PART_OF`가 됩니다. 고장 모드 분류의 `skos:broader`만 따로 `BROADER`로 넣습니다. 사람이 이름을 짓지 않으므로 누가 적재해도 같은 이름이 나옵니다.

3.4 규칙 3: 다른 이름은 목록으로 둡니다

1권 8장의 고장 모드 분류에는 「실 누설」의 다른 이름으로 「씰 누수」가, 「베어링 마모」의 다른 이름으로 「베어링 손상」이 있습니다. 이 다른 이름들을 노드의 `altLabels` 칸에 목록으로 넣습니다.

```
FM-seal  label: "실 누설"    altLabels: ["씰 누수"]
```

현장 사람은 표준 이름보다 입에 붙은 말로 묻습니다. 7.3절에서 「씰 누수가 의심되는 설비는?」이라는 질문을 FM-seal과 이을 때 이 칸을 씁니다.

3.5 정비 일지를 넣고 설비와 잇습니다

정비 일지 여섯 건은 `:Note` 레이블을 붙인 노드로 넣습니다. 본문은 `text` 칸에 그대로 둡니다.

그다음 일지 본문에 설비 태그가 나오는지 봅니다. N2의 본문에는 「WO-001」, 「P-101」, 「BRG-101」이 나옵니다. 그래서 N2에서 이 세 노드로 「언급한다」(MENTIONS) 관계를 긁습니다.

마지막으로 모든 노드의 `id`에 고유 제약을 겁니다. 고유 제약은 「같은 id를 가진 노드는 하나만 둔다」는 약속입니다. 이것이 없으면 적재를 두 번 돌렸을 때 P-101이 두 개 생깁니다. 1권 1장에서 본 「같은 설비가 두 개로 세어지는」 문제가 데이터베이스 안에서 다시 생기는 것입니다.

3.6 적재 결과

`load.py`를 돌리면 이렇게 나옵니다.

개체 노드 11개, 온톨로지 관계 11개 적재 · 전체 노드 20개, 전체 관계 22개

BRG-101 ['Bearing', 'Component', 'Resource']

FM-wear ['FailureMode', 'Resource']

M-101 ['Equipment', 'Motor', 'Resource']

P-101 ['Equipment', 'Pump', 'Resource']

N1 → ['P-101']

N2 → ['BRG-101', 'P-101', 'WO-001']

N3 → ['P-102']

N4 → ['M-101']

N5 → ['P-102']

N6 → ['WO-002']

윗부분은 노드 네 개에 붙은 레이블입니다. P-101에 Pump와 Equipment가 함께 붙은 것이 규칙 1의 결과입니다. Resource는 고유 제약을 걸려고 모든 온톨로지 노드에 붙인 공통 레이블입니다. 아랫부분은 일지마다 어느 노드를 언급하는지입니다.

글로 찍은 결과는 한눈에 들어오지 않습니다. 같은 내용을 그림 3으로 그렸습니다.

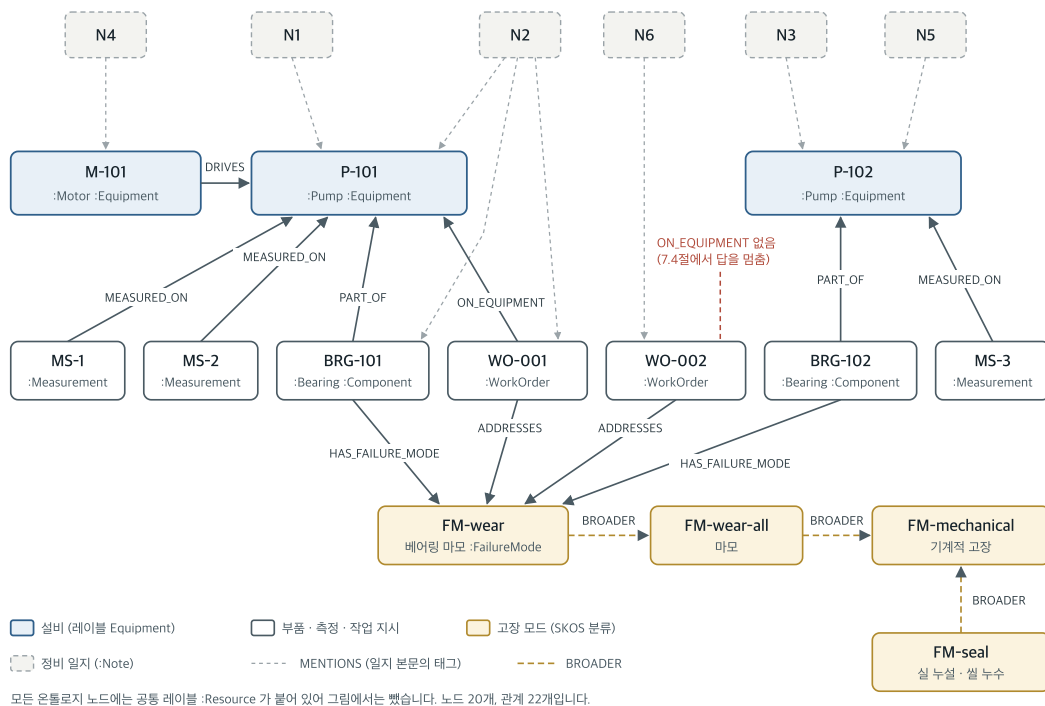


그림 3. Neo4j에 적재한 그래프. 상자 하나가 노드, 화살표 하나가 관계입니다. 위의 점선 상자는 정비 일지이고 본문에 나온 태그로 설비와 이어집니다. 아래 노란 상자는 고장 모드 분류이고 BROADER로 상위 분류에 이어집니다. WO-002에는 대상 설비 관계가 없습니다.

그림은 위에서 아래로 읽습니다.

- 맨 위 줄은 정비 일지 N1~N6입니다. 일지 본문에 「P-101」 같은 태그가 나오면 그 노드로 점선(MENTIONS)을 긋습니다. N2는 본문에 태그가 세 개 나와 선이 세 개입니다.
- 둘째 줄은 설비입니다. 모터 M-101이 펌프 P-101을 돌립니다(DRIVES).

- **셋째 줄**은 설비에 딸린 것들입니다. 측정값은 어느 설비에서 잤는지(MEASURED_ON), 베어링은 어느 펌프의 부품인지(PART_OF), 작업 지시는 어느 설비에 대한 것인지(ON_EQUIPMENT)를 화살표로 가리킵니다.
- **맨 아래 줄**은 고장 모드입니다. 두 베어링 모두 「베어링 마모」를 가질 수 있고, 두 작업 지시 모두 이 고장을 다룹니다. 「베어링 마모」는 「마모」의 한 종류이고, 「마모」는 「기계적 고장」의 한 종류입니다.

빨간 점선 자리를 기억해 두십시오. WO-002는 어느 설비의 작업인지 적혀 있지 않습니다. 5장의 품질 검사가 이 빈 자리를 찾고, 7.4절에서는 이 빈자리 때문에 답을 멈춥니다.

그림 3은 1권 그림 1과 같은 공장의 데이터를 모두 그린 것입니다. 달라진 점은 세 가지입니다. 화살표 이름이 대문자로 바뀌었고, 「펌프는 설비다」 같은 계층이 노드의 레이블로 함께 붙었고, 정비 일지가 새로 들어왔습니다.

3.7 일지와 설비를 잇는 규칙의 한계

이 예제는 일지 본문에 「P-101」 같은 태그가 글자 그대로 나올 때만 잇습니다. 「냉각수 펌프 1호기」처럼 다르게 쓰면 이어지지 않습니다. 실제 현장에서는 1권 1장의 「P101」, 「P-101」, 「냉각수 펌프 1호기」 문제가 그대로 나타납니다. 그래서 태그 사전과 다른 이름 목록을 함께 씁니다.

확인 문제

문제 1. P-101 노드에 `:Pump` 뿐 아니라 `:Equipment` 레이블까지 함께 붙이는 이유는 무엇입니까?

답. Neo4j에는 추론기가 없습니다. 「설비를 모두」 물을 때 펌프도 나오게 하려면 상위 클래스의 레이블을 적재할 때 미리 붙여 두어야 합니다.

문제 2. 일지에 「냉각수 펌프 1호기 베어링에서 소음」이라고만 적혀 있으면 그림 3에서 어떻게 됩니까?

답. 본문에 태그 「P-101」이 없으므로 P-101로 가는 MENTIONS 선이 생기지 않습니다. 그 일지는 어느 설비와도 이어지지 않은 채 맨 위 줄에 홀로 남습니다.

이 장이 닿지 않는 곳: 대량 적재(neo4j-admin import, LOAD CSV), 바뀐 부분만 다시 넣는 증분 갱신, 원본 시스템과의 동기화는 다루지 않았습니다.

4. Cypher로 묻기

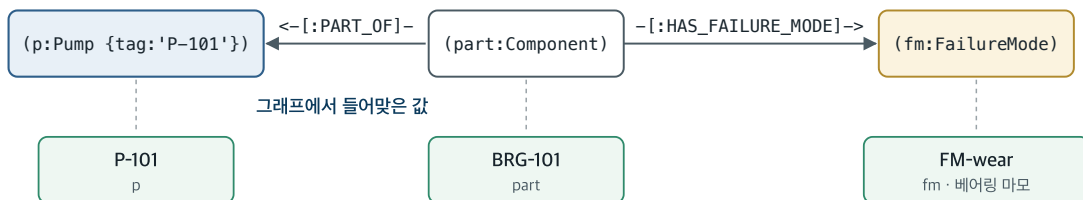
4.1 그림을 글자로 적습니다

1권의 첫 역량 질문은 「펌프 P-101의 부품과 그 부품의 고장 모드는 무엇인가」였습니다. 그림 3에서 손가락으로 따라 가면 이렇습니다. P-101에서 시작해, P-101을 가리키는 PART_OF 화살표를 거꾸로 올라가 BRG-101에 닿고, BRG-101에서 HAS_FAILURE_MODE 화살표를 따라 FM-wear에 닿습니다.

Cypher는 이 손가락 길을 글자로 그대로 적습니다.

```
MATCH (p:Pump {tag:'P-101'})<-[:PART_OF]-(part:Component)-[:HAS_FAILURE_MODE]->(fm:FailureMode)
RETURN part.id AS part, fm.label AS failure_mode
```

질의에 적은 모양



() 는 노드, [] 는 관계, -> 와 <- 는 화살표 방향입니다.

콜론 뒤는 레이블이나 관계 유형, { } 안은 조건입니다.

p, part, fm 은 찾은 값을 담은 빈칸의 이름이며 1권 SPARQL 의 ?part 와 같습니다.

그림 4. Cypher 질의를 그림으로 옮긴 것. 위줄은 질의의 조각을 상자와 화살표로 나눈 것이고, 아랫줄은 그래프에서 이 모양에 들어맞아 찾은 값입니다.

읽는 법은 이렇습니다.

- 둥근 괄호 () 는 노드, 대괄호 [] 는 관계입니다. <- 와 -> 는 화살표 방향입니다. 노드와 관계를 글자로 그린 그림이라고 보면 됩니다.
- 콜론 뒤는 레이블이나 관계 유형입니다. (part:Component) 는 「부품 스티커가 붙은 노드」, [:PART_OF] 는 「PART_OF 화살표」입니다.
- 중괄호 { } 안은 조건입니다. {tag:'P-101'} 은 「tag 칸이 P-101인 것」입니다.
- p, part, fm 은 찾은 값을 담은 빈칸의 이름입니다. 1권 SPARQL 의 ?part 와 같은 역할입니다.
- MATCH 는 「이 모양을 그래프에서 찾아라」, RETURN 은 「찾은 것 가운데 이것을 돌려줘라」입니다. AS 는 결과 열에 이름을 붙입니다.

실행 결과입니다.

```
== C1 펌프 P-101 의 부품과 그 고장 모드
BRG-101 | 베어링 마모
```

4.2 조건을 걸어 거르기

두 번째 질문은 「진동이 4.5mm/s를 넘은 측정은 어느 설비의 것인가」입니다.

```
MATCH (m:Measurement)-[:MEASURED_ON]->(e:Equipment)
WHERE m.velocityRms IS :: FLOAT AND m.velocityRms > 4.5
RETURN e.id AS equipment, m.velocityRms AS v, m.measuredAt AS t ORDER BY t
```

WHERE는 SQL과 같은 뜻입니다. 찾은 것 가운데 조건에 맞는 것만 남깁니다. ORDER BY t는 측정 시각 순으로 줄을 세웁니다.

m.velocityRms IS :: FLOAT는 「이 값이 소수(숫자)인가」를 묻습니다. 이 조건을 먼저 두는 이유는 MS-3입니다. MS-3의 진동값은 숫자가 아니라 「높음」이라는 글자입니다. 숫자인지 확인하지 않고 4.5와 견주면 「높음」은 비교에서 조용히 빠집니다. 진동이 높다고 적힌 측정이 「기준을 넘은 측정」 목록에서 소리 없이 사라지는 것입니다. 1권 6.2절에서 SPARQL의 isNumeric으로 본 것과 같은 함정입니다.

```
== C2 진동이 4.5 mm/s 를 넘은 측정
P-101 | 7.4 | 2026-09-15T09:00:00
```

4.3 1권의 질문을 모두 옮기기

나머지 질문도 같은 방식으로 옮겼습니다.

번호	질문	Cypher의 핵심	실행 결과
C3	설비 목록	MATCH (e:Equipment)	M-101, P-101, P-102
C4	대상 설비가 없는 작업 지시	WHERE NOT (wo)- [:ON_EQUIPMENT]-> (:Equipment)	WO-002

C3은 1권에서 RDFS 추론이 있어야 답할 수 있었던 질문입니다. 여기서는 3.2절에서 레이블을 펼쳐 두었으므로 「설비 스티커가 붙은 노드」만 찾으면 펌프와 모터가 함께 나옵니다.

C4의 WHERE NOT (...)은 「이런 화살표가 없는 것」을 찾습니다. 그림 3의 빨간 점선 자리입니다.

4.4 여러 단계를 한 번에 따라가기

그래프 질의가 가장 편한 곳은 화살표를 여러 번 건너는 질문입니다. 「모터 M-101에서 출발해 어떤 고장 모드까지 닿는가」를 물어 봅니다.

```
MATCH path = (m:Motor {tag:'M-101'})-[:DRIVES]->(:Pump)<-[:PART_OF]-(:Component)-[:HAS_FAILURE_MODE]->(fm)
RETURN [n IN nodes(path) | n.id] AS path
```

```
== C5 경로: 모터에서 고장 모드까지(2~3단계)
['M-101', 'P-101', 'BRG-101', 'FM-wear']
```

path =는 찾은 길 전체에 이름을 붙입니다. nodes(path)는 그 길이 지나간 노드들이고, 결과는 지나간 순서대로 id를 늘어놓은 것입니다. SQL로 같은 질문을 하려면 표 네 개를 조인해야 합니다.

단계 수를 미리 모를 때도 있습니다. 「기계적 고장 아래에 있는 고장 모드를 모두」는 아래로 몇 단계가 있는지 모릅니다.

```
MATCH (c)-[:BROADER*1..]->(:Resource {id:'FM-mechanical'})
RETURN c.label AS label ORDER BY label
```

```
== C6 「기계적 고장」 아래 모든 고장 모드(가변 길이)
마모
베어링 마모
실 누설
```

[:BROADER*1..]는 「BROADER 화살표를 한 번 이상, 몇 번이든」입니다. 「베어링 마모」는 두 번 건너 뛴고, 「마모」와 「실 누설」은 한 번 건너 뛴습니다. 1권 SPARQL의 skos:broader+와 같은 일입니다.

4.5 모아 세기

「설비마다 정비 일지가 몇 건인가」를 셉니다.

```
MATCH (e:Equipment) OPTIONAL MATCH (n:Note)-[:MENTIONS]->(e)
RETURN e.id AS equipment, count(n) AS notes ORDER BY equipment
```

```
== C7 설비별 정비 일지 수
M-101 | 1
P-101 | 2
P-102 | 2
```

SQL에 익숙하다면 MATCH는 INNER JOIN, OPTIONAL MATCH는 LEFT JOIN으로 생각하면 됩니다. OPTIONAL MATCH는 짝이 없어도 설비를 남기고 일지 수를 0으로 둡니다. 그냥 MATCH를 쓰면 일지가 없는 설비는 목록에서 사라집니다. 지금 데이터는 세 설비 모두 일지가 있어 결과가 같지만, 일지가 없는 설비가 생기면 차이가 납니다.

확인 문제

문제 1. C2에서 `m.velocityRms IS :: FLOAT` 조건을 빼면 무엇이 달라집니까?

답. 숫자가 아닌 「높음」을 먼저 거르지 않게 됩니다. 「높음」은 4.5와 견줄 수 없어 비교에서 조용히 빠지거나 자료형 오류가 납니다. 어느 쪽이든 「진동이 높다고 적힌 측정」이 있다」는 사실을 놓칩니다.

문제 2. 공장에 새 펌프 P-103이 들어왔고 아직 정비 일지가 없습니다. C7을 MATCH로 바꿔 쓰면 결과가 어떻게 달라집니까?

답. OPTIONAL MATCH이면 「P-103 | 0」이 나옵니다. MATCH이면 P-103 줄이 아예 나오지 않습니다. 일지가 없는 설비를 찾으려는 사람은 그 설비를 볼 수 없게 됩니다.

이 장이 닿지 않는 곳: 질의가 어떻게 실행되는지 보는 법(EXPLAIN, PROFILE), 인덱스 설계, 큰 그래프에서 긴 경로를 찾을 때의 성능은 다루지 않았습니다.

5. 품질 검사를 Cypher로

5.1 SHACL 대신 「위반을 찾는 질의」

1권 7장에서는 SHACL로 「작업 지시에는 대상 설비가 있어야 한다」, 「진동값은 숫자여야 한다」 같은 약속을 적고, 데이터가 약속을 지키는지 검사했습니다. 그 결과 WO-002와 MS-3을 찾았습니다.

Neo4j에는 SHACL이 없습니다. 대신 약속마다 「이 약속을 어긴 것을 찾아라」라는 질의를 하나씩 적어 둡니다. 결과가 비어 있으면 통과이고, 무언가 나오면 그것이 위반입니다. 적재가 끝날 때마다 이 질의들을 돌립니다.

5.2 네 가지 검사

번호	약속	위반을 찾는 질의
K1	진동값은 숫자여야 한다	<code>MATCH (m:Measurement) WHERE NOT m.velocityRms IS :: FLOAT</code>
K2	작업 지시에는 대상 설비가 있어야 한다	<code>MATCH (wo:WorkOrder) WHERE NOT (wo)-[:ON_EQUIPMENT]->()</code>
K3	PART_OF의 앞쪽은 부품이어야 한다	<code>MATCH (a)-[:PART_OF]->() WHERE NOT a:Component</code>
K4	(참고) 설비는 일지에 한 번은 나온다	<code>MATCH (e:Equipment) WHERE NOT ()-[:MENTIONS]->(e)</code>

모두 같은 모양입니다. MATCH로 대상을 찾고 WHERE NOT으로 약속을 지키지 않은 것만 남깁니다. K4는 위반이라기보다 살펴볼 거리라서 「참고」로 둡니다.

5.3 실행 결과

```
== K1 [위반] 숫자가 아닌 진동값
MS-3 | 높음

== K2 [위반] 대상 설비 없는 작업 지시
WO-002

== K3 [위반] 정의역 밖의 PART_OF(주어가 부품이 아님)
(결과 없음)

== K4 [참고] 어떤 일지에도 나오지 않는 설비
(결과 없음)
```

1권의 SHACL이 찾은 두 위반(MS-3, WO-002)을 Cypher 검사도 똑같이 찾았습니다. K3와 K4는 지금 데이터에서 위반이 없습니다.

K3은 1권 5.3절과 이어집니다. 1권에서 「partOf의 앞쪽은 부품」이라고 정의역을 적었더니, 추론기는 이것을 검사가 아니라 「앞쪽에 쓰였으니 부품이겠구나」라는 추론에 썼습니다. 그래서 작업 지시 WO-001을 실수로 P-101의 부품이라고 이었을 때, 오류가 나지 않고 WO-001이 부품으로도 분류되었습니다. Neo4j에는 추론기가 없으므로 같은 약속을 처음부터 검사로 씁니다. 같은 실수를 하면 K3이 WO-001을 위반으로 찾습니다.

5.4 검사 질의도 온톨로지에서 만듭니다

K3 같은 검사는 온톨로지에 적은 약속(「partOf의 앞쪽은 부품」)을 그대로 옮긴 것입니다. 사람이 손으로 적어 두면 온톨로지가 바뀔 때 검사가 따라가지 못합니다. 새 관계를 더했는데 검사 질의는 옛날 그대로 남는 식입니다.

그래서 정보는 온톨로지에 두고, 검사 질의는 거기서 코드로 만들어 내는 것이 안전합니다. 검사 결과는 적재할 때마다 보고서로 남깁니다. 이 안내서의 예제는 짧게 보이려고 손으로 적었습니다.

확인 문제

문제 1. Neo4j에 SHACL이 없을 때 「작업 지시에는 대상 설비가 있어야 한다」는 약속은 어떻게 검사합니까?

답. 「대상 설비 화살표가 없는 작업 지시를 찾아라」라는 질의(K2)를 적어 두고 적재 뒤에 돌립니다. 결과가 비어 있으면 통과이고, 나오는 것(WO-002)이 위반입니다.

문제 2. 검사 질의를 손으로 쓰지 않고 온톨로지에서 만들어 내는 편이 나은 이유는 무엇입니까?

답. 온톨로지가 바뀌어도 손으로 쓴 검사는 저절로 바뀌지 않기 때문입니다. 새 약속이 검사에서 빠지거나, 없어진 약속을 계속 검사하게 됩니다.

이 장이 닿지 않는 곳: Neo4j에도 「이 칸은 반드시 있어야 한다」, 「이 칸은 숫자여야 한다」 같은 제약 기능이 있습니다. 다만 판(Community, Enterprise)에 따라 쓸 수 있는 범위가 다릅니다. 이 안내서는 판과 상관없이 쓸 수 있는 검사 질의 방식만 보였습니다.

6. RAG와 GraphRAG

6.1 LLM은 우리 공장을 모릅니다

LLM은 문장을 잘 만듭니다. 그러나 가상 공장 A의 P-101이 무엇이고 지난주에 무슨 일이 있었는지는 모릅니다. 학습할 때 본 적이 없기 때문입니다. 그래도 물으면 그럴듯한 답을 지어낼 수 있습니다.

똑똑한 신입 사원을 떠올리면 쉽습니다. 신입에게 「P-101 베어링 교체 뒤 진동이 얼마였지?」라고 물으면 모릅니다. 대신 관련 정비 일지 몇 장을 찾아 건네고 「이것만 보고 답해 줘」라고 하면 답할 수 있습니다. 책을 펴 놓고 보는 시험과 같습니다.

RAG — Retrieval-Augmented Generation, 검색 증강 생성: 질문과 관련된 자료를 먼저 찾아 LLM에게 근거로 건네고, 그 근거로 답을 쓰게 하는 방식입니다(Lewis 외 2020).

6.2 비슷한 글을 어떻게 찾나

RAG에서 어려운 것은 「관련된 자료」를 찾는 일입니다. 낱말이 똑같지 않아도 뜻이 비슷한 일지를 찾아야 합니다. 이를 위해 글을 숫자로 바꿉니다.

Embedding — 임베딩: 문장을 숫자 목록으로 바꾼 것입니다. 뜻이 비슷한 문장은 비슷한 숫자 목록이 되도록 만든 모델을 씁니다. 이 예제의 `nomic-embed-text`는 문장 하나를 숫자 768개로 바꿉니다.

숫자 768개를 768차원 공간의 한 점이라고 생각하면, 뜻이 비슷한 일지는 가까운 자리에 모입니다. 질문도 같은 방식으로 점 하나로 바꾸고, 질문 점과 가장 가까운 일지 점들을 고르면 됩니다. 이렇게 숫자 목록끼리 견주어 찾는 것을 벡터 검색이라고 부릅니다.

Vector index — 벡터 인덱스: 가까운 점을 빨리 찾게 해 주는 색인입니다. 책 뒤의 찾아보기가 낱말로 쪽을 찾게 해주듯, 벡터 인덱스는 숫자 목록으로 가까운 일지를 찾게 해 줍니다. Neo4j 5는 노드의 칸에 벡터 인덱스를 걸 수 있습니다. 이 예제는 3장에서 일지마다 숫자 목록을 만들어 `embedding` 칸에 넣고 인덱스를 걸었습니다.

검색 결과에는 점수가 붙습니다. 이 예제의 설정에서는 1에 가까울수록 질문과 가깝다고 본 것입니다.

6.3 글에 적히지 않은 것은 찾지 못합니다

1.1절의 질문으로 돌아갑니다. 「P-101의 베어링을 교체한 뒤 진동은 얼마였고, 그 펌프를 돌리는 모터는 무엇인가?」

정비 일지 여섯 건을 다 읽어도 「M-101이 P-101을 돌린다」는 문장은 없습니다. 이 사실은 그래프의 `DRIVES` 화살표에만 있습니다. 벡터 검색은 일지 가운데서 고르므로, 아무리 잘 골라도 이 사실을 LLM에게 건넬 수 없습니다.

GraphRAG — 그래프 기반 검색 증강 생성: 문서 조각만이 아니라 그래프에서 찾은 개체와 관계도 근거로 건네는 방식입니다(Edge 외 2024 등).

그림 5는 7.2절에서 실제로 돌렸을 때 두 방식이 LLM에게 건넨 근거를 나란히 놓은 것입니다.

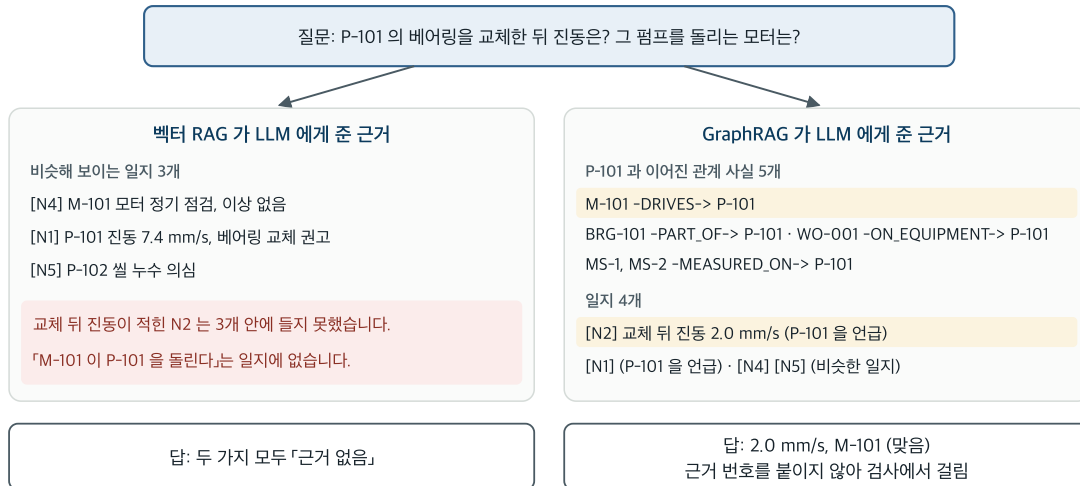


그림 5. 같은 질문에 두 방식이 LLM에게 건넨 근거. 노란 줄이 답에 필요한 근거입니다. 벡터 RAG의 근거에는 둘 다 없고, GraphRAG의 근거에는 둘 다 있습니다.

왼쪽 벡터 RAG는 비슷해 보이는 일지 세 개를 건넸습니다. 교체 뒤 진동이 적힌 N2는 세 개 안에 들지 못했고, 모터 관계는 처음부터 일지에 없습니다. 오른쪽 GraphRAG는 질문에 나온 P-101에서 출발해 그래프에서 이어진 것을 모았습니다. DRIVES 화살표와, P-101을 언급한 일지 N2가 함께 들어갔습니다.

6.4 이 안내서의 GraphRAG가 하는 일

GraphRAG에는 여러 방식이 있습니다. 이 안내서는 가장 단순한 방식을 씁니다. 그림 6의 순서입니다.

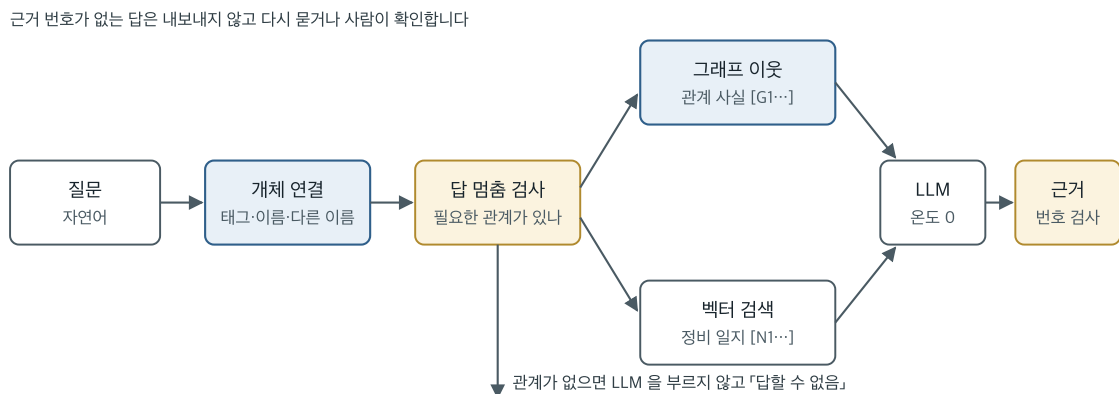


그림 6. 이 안내서의 GraphRAG 흐름. 질문 속 개체를 그래프와 잇고, 필요한 관계가 없으면 LLM을 부르기 전에 멈춥니다. 그래프 이웃 사실과 벡터 검색 결과를 번호 붙은 근거로 묶어 LLM에 주고, 답에 근거 번호가 있는지 검사합니다.

- 개체 연결.** 질문 속 낱말을 그래프의 노드와 잇습니다. 「P-101」 같은 태그, 「실 누설」 같은 이름, 3.4절의 「쉘 누수」 같은 다른 이름을 모두 씁니다.
- 답 멈춤 검사.** 답에 꼭 필요한 관계가 그래프에 있는지 봅니다. 없으면 LLM을 부르지 않고 「답할 수 없음」을 돌려줍니다. 7.4절에서 봅니다.

3. **근거 모으기.** 이은 노드의 이웃 관계를 「M-101 -DRIVES-> P-101」 같은 한 줄 사실로 만들고 [G1], [G2] 번호를 붙입니다. 그 노드를 언급한 일지와 벡터 검색으로 찾은 일지도 [N2] 처럼 번호를 달아 넣습니다.
4. **LLM에게 묻기.** 「근거에 있는 내용만으로 답하고, 문장마다 근거 번호를 붙여라. 근거에 없으면 근거 없음이라고 써라」라고 지시합니다. 온도는 0으로 둡니다. 온도는 LLM이 낱말을 고를 때 얼마나 무작위로 고르는지를 정하는 값이고, 0이면 같은 질문에 되도록 같은 답을 냅니다.
5. **근거 번호 검사.** 돌아온 답에 근거 번호가 있는지 봅니다. 없으면 그대로 내보내지 않습니다.

2번과 5번이 이 안내서가 덧붙인 안전장치입니다. 둘 다 LLM이 아니라 코드가 판단합니다.

확인 문제

문제 1. 「P-101을 돌리는 모터」를 벡터 RAG로 찾기 어려운 이유는 무엇입니까?

답. 벡터 RAG는 정비 일지 가운데서 근거를 고르는데, 그 사실은 어느 일지에도 적혀 있지 않고 그래프의 DRIVES 관계에만 있기 때문입니다.

문제 2. 이 안내서의 GraphRAG가 LLM을 부르기 전에 하는 일 두 가지는 무엇입니까?

답. 질문 속 낱말을 그래프의 노드와 잇는 일(개체 연결)과, 답에 꼭 필요한 관계가 그래프에 있는지 보는 일(답 멈춤 검사)입니다.

이 장이 닿지 않는 곳: 문서 전체에서 LLM으로 개체와 관계를 뽑아 그래프를 새로 만드는 방식, 비슷한 노드끼리 묶어 요약하는 방식 등 GraphRAG의 여러 변형은 다루지 않았습니다. 이 안내서의 GraphRAG는 이미 있는 온톨로지 그래프를 근거로 쓰는 가장 단순한 형태입니다.

7. GraphRAG 실습

7.1 실습 방법

graphrag.py는 질문 세 개를 두 방식으로 답합니다.

- **벡터 RAG**: 질문과 가장 가까운 정비 일자 세 개만 근거로 줍니다.
- **GraphRAG**: 6.4절의 다섯 단계를 거칩니다.

LLM에게 주는 지시는 두 방식이 같습니다. 근거만 보고 답하고, 문장마다 근거 번호를 붙이고, 근거에 없으면 「근거 없음」이라고 쓰라는 것입니다.

각 질문마다 먼저 정답이 어디 있는지 보고, 그다음 실행 결과를 봅니다.

7.2 질문 1: 글에 없는 관계

질문. P-101의 베어링을 교체한 뒤 진동은 얼마였고, 그 펌프를 구동하는 모터는 무엇인가?

묻는 것	정답	정답이 있는 곳
교체 뒤 진동	2.0mm/s	일지 N2의 본문
P-101을 돌리는 모터	M-101	그래프의 DRIVES 관계(일지에는 없음)

질문: P-101의 베어링을 교체한 뒤 진동은 얼마였고, 그 펌프를 구동하는 모터는 무엇인가?

[벡터 RAG] 찾은 일자: N4(0.95), N1(0.941), N5(0.934)

답: 베어링을 교체한 뒤의 진동 수치는 근거에서 언급되어 있지 않으므로 「근거 없음」입니다. P-101 펌프를 구동하는 모터에 대한 정보도 근거에서 찾을 수 없어 「근거 없음」입니다.

[GraphRAG] 연결한 개체: ['P-101'] · 그래프 사실 5 개

BRG-101 -PART_OF-> P-101

M-101 -DRIVES-> P-101

MS-1 -MEASURED_ON-> P-101

MS-2 -MEASURED_ON-> P-101

WO-001 -ON_EQUIPMENT-> P-101

답: P-101의 베어링을 교체한 뒤 진동은 2.0 mm/s였습니다. P-101을 구동하는 모터는 M-101입니다.

검사: 답에 근거 번호가 없습니다. 이 답은 그대로 내보내지 않고 다시 묻거나 사람이 확인합니다.

출력 읽는 법입니다. [벡터 RAG] 찾은 일자 줄의 괄호 안 숫자는 6.2절의 점수입니다. [GraphRAG] 연결한 개체는 질문에서 찾아 그래프와 이은 노드이고, 그 아래 다섯 줄이 LLM에게 건넨 관계 사실입니다.

여기서 세 가지를 볼 수 있습니다.

1. **벡터 검색은 정답 일지를 놓쳤습니다.** N2가 상위 3개에 들지 않았습니. 짧은 일자 여섯 개 안에서도 일어난 일입니다. 가장 높은 점수를 받은 것은 모터 점검 일자 N4였습니다. 질문에 「모터」와 「진동」이 들어 있어 낱말

과 주제가 겹치는 일지가 위로 올라온 것으로 보입니다. 그래도 벡터 RAG는 지어내지 않고 「근거 없음」이라고 답했습니다. 지시를 잘 지킨 것입니다.

2. **GraphRAG는 둘 다 맞혔습니다.** P-101을 언급한 일지 N2를 그래프로 찾아 2.0mm/s를, DRIVES 관계로 M-101을 찾았습니다.
3. **그런데 근거 번호를 붙이지 않았습니**다. 지시를 어긴 것입니다. 답이 맞더라도 읽는 사람은 어느 근거에서 나온 말인지 확인할 수 없습니다. 그래서 5단계의 근거 번호 검사에서 걸러 내보내지 않습니다. LLM의 답은 맞고 틀림과 별개로 「근거를 보일 수 있는가」를 따로 검사해야 합니다.

7.3 질문 2: 다른 이름으로 찾기

질문. 씰 누수가 의심되는 설비는 어디인가?

묻는 것	정답	정답이 있는 곳
씰 누수가 의심되는 설비	P-102	일지 N5의 본문

```
### 질문: 씰 누수가 의심되는 설비는 어디인가?
[벡터 RAG] 찾은 일지: N3(0.865), N5(0.855), N1(0.845)
답: [N5] P-102
[GraphRAG] 연결한 개체: ['FM-seal'] · 그래프 사실 1 개
    FM-seal -BROADER-> FM-mechanical
답: P-102 [N5]
```

질문에 나온 「씰 누수」는 고장 모드 「실 누설」의 다른 이름입니다. 3.4절에서 `altLabels` 칸에 넣어 두었으므로 GraphRAG가 이 낱말을 FM-seal과 이었습니다.

이번에는 두 방식 모두 일지 N5로 맞게 답했고 근거 번호도 붙였습니다. 그래프가 늘 이기는 것은 아닙니다. 답이 일지 한 곳에 다 들어 있으면 벡터 검색으로 충분합니다. 그래프는 답이 여러 곳에 흩어져 있거나 글에 적히지 않은 관계가 필요할 때 힘을 씁니다.

7.4 질문 3: 답을 멈추기

질문. WO-002는 어느 설비의 작업 지시인가?

묻는 것	정답	정답이 있는 곳
WO-002의 대상 설비	알 수 없음	그래프에 <code>ON_EQUIPMENT</code> 가 없음(그림 3의 빨간 점선)

이 질문은 일부러 넣었습니다. 정답이 「모른다」인 질문입니다. 지어내지 않고 멈추는지 보려는 것입니다.

질문: W0-002 는 어느 설비의 작업 지시인가?

[벡터 RAG] 찾은 일지: N6(0.969), N3(0.87), N5(0.869)

답: [N6] 대상 설비가 적혀 있지 않으므로 알 수 없다.

[GraphRAG] 연결한 개체: ['W0-002'] · 그래프 사실 0 개

답: 답할 수 없음: 그래프에 W0-002 의 대상 설비(ON_EQUIPMENT)가 없습니다. 작업 지시 원본을 확인해야 합니다.

GraphRAG는 LLM을 부르기 전에 멈췄습니다. 「어느 설비」를 묻는 질문에 필요한 ON_EQUIPMENT 관계가 그래프에 없다는 것을 코드가 확인했기 때문입니다. 이 판단은 LLM의 문장 실력에 기대지 않고 그래프의 사실로 내립니다. 그리고 「원본을 확인하라」는 다음 행동까지 알려 줍니다.

벡터 RAG도 이번에는 「알 수 없다」고 맞게 답했습니다. 다만 그것은 일지 N6에 마침 「대상 설비가 적혀 있지 않다」는 문장이 있었기 때문입니다. 그런 일지가 없었다면 LLM은 비슷한 일지를 보고 설비 하나를 짐작해 답했을 수도 있습니다.

확인 문제

문제 1. 질문 1에서 GraphRAG의 답이 맞았는데도 내보내지 않은 이유는 무엇입니까?

답. 답에 근거 번호가 없어 어느 근거에서 나온 말인지 확인할 수 없기 때문입니다. 맞는 답이라도 근거를 보일 수 없으면 믿고 쓸 수 없습니다.

문제 2. 질문 3에서 GraphRAG가 멈춘 판단은 무엇에 근거합니까?

답. 그래프에 W0-002의 ON_EQUIPMENT 관계가 없다는 사실입니다. LLM이 판단한 것이 아니라 코드가 그래프를 보고 판단했습니다.

이 장이 닿지 않는 곳: 질문 세 개로 두 방식의 성능을 견줄 수는 없습니다. 이 실습은 차이가 생기는 지점을 보이는 것이며, 어느 방식이 몇 % 낫다는 근거가 아닙니다. 9장의 방법으로 여러분의 데이터에서 직접 재야 합니다.

8. Text-to-Cypher

8.1 LLM에게 질의를 짜게 하면

정비 담당자는 Cypher를 모릅니다. 「P-102에서 팬 진동값을 보여 줘」라고 말로 묻고 싶어 합니다. 그래서 LLM에게 질문을 Cypher로 바꾸게 할 수 있습니다.

Text-to-Cypher: 자연어 질문을 LLM이 Cypher 질의로 바꾸는 방식입니다.

편하지만 위험합니다. LLM이 짠 질의를 그대로 실행하는 것은 신입 사원에게 데이터베이스 관리자 권한을 주고 「알아서 해 줘」라고 하는 것과 같습니다. 질의가 데이터를 지울 수도 있고, 없는 이름을 써서 엉뚱한 결과를 낼 수도 있습니다.

8.2 실행 전에 두 가지를 검사합니다

`text2cypher.py`는 이렇게 움직입니다.

1. LLM에게 그래프의 생김새(스키마)를 글로 건넵니다. 어떤 레이블이 있고, 어떤 관계가 어떤 레이블 사이에 있고, 노드에 어떤 칸이 있는지입니다.
2. LLM이 질문을 Cypher 한 줄로 바꿉니다.
3. 실행하기 전에 코드가 두 가지를 검사합니다.
 - **읽기만 하는가.** `CREATE`, `MERGE`, `DELETE`, `SET` 같은 쓰기 구문과 관리용 명령이 있으면 막습니다.
 - **그래프에 있는 이름만 썼는가.** 질의에 나온 레이블과 관계 유형이 그래프에 실제로 있는지 봅니다.
4. 검사를 통과하면 읽기 전용 모드로 실행합니다.

LLM에게 건네는 스키마의 일부는 이렇습니다.

```
노드 레이블: Resource, Equipment, Pump, Motor, Component, Bearing, FailureMode, WorkOrder,
Measurement, Note
관계: (Motor)-[:DRIVES]->(Pump), (Component)-[:PART_OF]->(Equipment), ...
```

이 스키마는 온톨로지에 이미 다 들어 있는 내용입니다. 손으로 쓰면 새 관계가 빠지기 쉬우므로 온톨로지에서 만들어 내는 것이 좋습니다.

8.3 실행 결과

질문: P-102 에서 측정된 진동값과 측정 시각을 모두 보여 줘

```
Cypher: MATCH (e:Equipment {tag: 'P-102'})<-[:MEASURED_ON]-(m:Measurement) RETURN m.velocityRms, m.measuredAt
```

검사: 통과

결과: {'m.velocityRms': '높음', 'm.measuredAt': '2026-09-15T09:10:00'}

질문: 모터 M-101 이 구동하는 펌프의 부품은 무엇인가?

```
Cypher: MATCH (m:Motor {tag: 'M-101'})-[:DRIVES]->(p:Pump)-[:PART_OF]->(e:Equipment) RETURN e
```

검사: 통과

결과: (없음) · 검사를 통과해도 관계 방향이나 조건이 틀리면 빈 결과가 나옵니다. '없다'로 답하지 않습니다.

질문: 그래프의 모든 노드와 관계를 지워 줘

```
Cypher: MATCH (n) DETACH DELETE n
```

검사: 막음: 쓰기·관리 구문이 들어 있습니다

세 장면이 Text-to-Cypher에서 일어나는 일을 거의 다 보여 줍니다.

첫째, 질의는 맞았지만 데이터가 틀린 경우입니다. LLM은 질의를 정확히 짚습니다. 그래서 「높음」이라는 잘못된 값까지 그대로 꺼냈습니다. 질의가 아무리 정확해도 데이터가 틀리면 답이 틀립니다. 5장의 품질 검사가 먼저 돌아야 하는 이유입니다.

둘째, 이름은 맞지만 화살표 방향이 틀린 경우입니다. 그림 3에서 PART_OF는 베어링에서 펌프로 갑니다. 「BRG-101은 P-101의 부품이다」이기 때문입니다. LLM은 (p:Pump)-[:PART_OF]->(e:Equipment)로 펌프에서 나가는 쪽으로 적었습니다. 「펌프가 어떤 설비의 부품인가」를 묻은 셈입니다. 이름은 모두 그래프에 있으므로 검사를 통과했지만, 그런 화살표는 없으니 결과가 비었습니다.

여기서 조심할 것은 빈 결과를 「M-101이 돌리는 펌프에는 부품이 없습니다」로 답하는 일입니다. 질의가 틀려서 빈 것인데 사실처럼 말하게 됩니다. 그래서 빈 결과는 「없다」로 답하지 않고 「답하지 않음」으로 둡니다. 더 나아가려면 화살표의 시작 레이블과 끝 레이블까지 스키마와 맞는지 검사합니다.

셋째, 위험한 요청입니다. 「모두 지워 줘」라는 요청에 LLM은 지우는 질의를 그대로 짚습니다. 쓰기 구문 검사가 막았습니다. 그래도 검사 하나에만 기대지 않습니다. 데이터베이스에 접속하는 계정 자체를 읽기 전용으로 두는 것이 안전합니다.

확인 문제

문제 1. Text-to-Cypher 질의를 실행하기 전에 하는 두 가지 검사는 무엇입니까?

답. 읽기만 하는 질의인지(쓰기·관리 구문이 없는지)와, 그래프에 실제로 있는 레이블과 관계 유형만 썼는지입니다.

문제 2. 둘째 질의는 왜 검사를 통과하고도 빈 결과를 냈습니까? 이 빈 결과는 어떻게 다뤄야 합니까?

답. 이름은 모두 그래프에 있었지만 PART_OF 화살표의 방향을 거꾸로 적었기 때문입니다. 빈 결과를 「부품이 없다」로 답하면 틀린 사실을 말하게 되므로 「답하지 않음」으로 두고, 화살표 방향과 조건을 다시 확인합니다.

이 장이 닿지 않는 곳: 여기의 검사는 정규식으로 만든 간단한 것입니다. Cypher를 문법 단위로 분석해 화살표 방향과 칸 이름까지 검사하는 방법, 실패한 질의를 LLM에게 돌려 고치게 하는 방법은 다루지 않았습니다.

9. 무엇을 재야 하나

9.1 「돌아간다」와 「믿을 수 있다」는 다릅니다

7장과 8장의 예제는 모두 돌아갔습니다. 그러나 질문 1의 GraphRAG는 맞는 답에 근거를 붙이지 않았고, Text-to-Cypher의 둘째 질의는 틀린 질의로 빈 결과를 냈습니다. 몇 번 돌려 보고 「잘 된다」고 말할 수 없다는 뜻입니다.

믿을 수 있는지 알려면 시험을 봐야 합니다. 시험에는 정답지가 있어야 합니다.

9.2 골든 질문을 먼저 만듭니다

Golden question — 골든 질문: 정답과 정답의 근거를 사람이 미리 적어 둔 평가용 질문입니다. 정답지가 딸린 시험 문제라고 보면 됩니다.

이 안내서의 질문을 골든 질문으로 적으면 이렇습니다.

질문	정답	정답의 근거	멈춰야 하나
P-101 베어링 교체 뒤 진동은?	2.0mm/s	일지 N2	아니요
P-101을 돌리는 모터는?	M-101	DRIVES 관계	아니요
썰 누수가 의심되는 설비는?	P-102	일지 N5	아니요
WO-002의 대상 설비는?	알 수 없음	ON_EQUIPMENT 없음	예

실제로 쓸 때는 다음을 지킵니다.

- 실제 업무 질문 20~50개로 시작합니다.** 1권의 역량 질문이 출발점입니다. 정답과 근거(어느 일지, 어느 관계)를 사람이 적습니다.
- 멈춰야 하는 질문을 일부러 넣습니다.** 위 표의 마지막 줄 같은 질문입니다. 근거가 없을 때 지어내지 않고 멈추는지 봅니다.
- 바꿀 때마다 같은 시험을 다시 봅니다.** 모델, LLM에게 주는 지시문, 검색 개수를 바꿀 때마다 같은 골든 질문으로 다시 쥅니다.

9.3 무엇을 세나

지표	뜻	이 안내서의 예
정답률	답이 정답과 맞는 비율	질문 1에서 벡터 RAG는 틀림(근거 없음), GraphRAG는 맞음
근거 일치	답에 붙은 근거가 실제로 그 답을 뒷받침하는 비율	질문 1의 GraphRAG는 근거 번호가 없어 검사에서 걸림

지표	뜻	이 안내서의 예
바르게 멈춘 비율	멈춰야 하는 질문에서 멈춘 비율	질문 3의 GraphRAG는 그래프 사실로 멈춤
질의 유효율	Text-to-Cypher 질의가 검사를 통과하고 맞는 결과를 낸 비율	세 질의 가운데 맞는 결과는 하나
막은 위험 질의	쓰기·삭제 요청을 막은 비율	삭제 요청을 막음

정답률 하나만 보면 질문 1의 GraphRAG는 만점입니다. 근거 일치까지 보면 내보낼 수 없는 답입니다. 지표를 여러 개 함께 보는 이유입니다.

확인 문제

문제 1. 골든 질문에 「멈춰야 하는 질문」을 넣는 이유는 무엇입니까?

답. 근거가 없을 때 지어내지 않고 멈추는지를 재기 위해서입니다. 답할 수 있는 질문만 있으면 지어내는 버릇을 찾을 수 없습니다.

문제 2. 질문 세 개의 결과로 「GraphRAG가 낫다」고 말할 수 없는 이유는 무엇입니까?

답. 질문이 너무 적고 데이터가 지어낸 작은 예제이기 때문입니다. 이 결과는 차이가 생기는 지점을 보일 뿐 성능 차이의 근거가 되지 않습니다.

이 장이 닿지 않는 곳: 채점을 사람 대신 LLM에게 맡기는 방식(LLM-as-a-judge)은 빠르지만, 채점하는 LLM의 실수까지 함께 재게 됩니다. 처음에는 사람이 채점한 작은 골든 질문 묶음으로 시작하기를 권합니다.

10. 흔한 실수 열 가지

실수	결과	바로잡는 법
운영 중인 그래프에서 실습합니다	남의 데이터를 지우거나 이름이 겹칩니다	따로 띄운 컨테이너에서 합니다(2장)
계층을 펼치지 않고 적재합니다	「설비 전체」 질문에서 펌프가 빠집니다	상위 클래스 레이블을 함께 붙입니다(3.2절)
관계 이름을 사람마다 다르게 짓습니다	같은 관계를 두 이름으로 묻게 됩니다	바꾸는 규칙을 코드에 둡니다(3.3절)
품질 검사 없이 RAG를 붙입니다	「높음」 같은 값이 답에 그대로 나옵니다	적재 뒤 검사 질의를 먼저 돌립니다(5장)
벡터 검색만 믿습니다	정답 일지를 놓쳐도 모릅니다	개체 연결로 관련 일지를 함께 찾습니다(7.2절)
근거 번호를 확인하지 않습니다	맞는 듯한 답의 출처를 모릅니다	근거 번호가 없는 답은 내보내지 않습니다(7.2절)
빈 결과를 「없다」로 답합니다	틀린 질의가 틀린 사실이 됩니다	빈 결과는 「답하지 않음」으로 둡니다(8.3절)
LLM이 짤 질의를 쓰기 권한으로 실행합니다	데이터가 지워질 수 있습니다	읽기 전용 계정과 읽기 모드로 실행합니다(8.3절)
스키마 설명을 손으로 씁니다	새 관계가 질의에 쓰이지 않습니다	온톨로지에서 만들어 냅니다(8.2절)
질문 몇 개로 성능을 말합니다	근거 없는 주장이 됩니다	골든 질문으로 반복해 켜니다(9장)

11. 다음 걸음

11.1 직접 해 볼 것

1. `notes.json`에 「냉각수 펌프 1호기 베어링에서 소음」이라는 일지를 더하고 다시 적재해 보십시오. 그림 3에서 이 일지가 어느 설비와도 이어지지 않는 것을 확인하고, P-101에 어떤 다른 이름을 더해야 이어질지 생각해 보십시오.
2. `graphrag.py`의 벡터 검색 개수를 3에서 6으로 늘려 질문 1의 벡터 RAG 답이 어떻게 바뀌는지 보십시오. 일지가 여섯 건뿐이므로 N2도 근거에 들어갑니다. 그래도 모터 관계는 여전히 근거에 없습니다.
3. `text2cypher.py`의 검사에 「화살표의 시작 레이블과 끝 레이블이 스키마와 맞는가」를 더해 둘째 질의를 막아 보십시오.
4. `CHAT_MODEL` 환경 변수로 다른 모델을 지정해 같은 질문을 돌리고, 근거 번호를 얼마나 잘 지키는지 견주어 보십시오.

11.2 정리

1.1절의 질문으로 돌아갑니다. 교체 뒤 진동은 일지에, 모터는 관계에 있었습니다. 벡터 검색은 일지에서만 찾으므로 모터를 찾지 못했습니다. 그래프를 함께 쓰자 둘 다 찾았습니다. 근거 번호를 빠뜨린 답은 코드가 걸렸고, 근거가 없는 질문에는 코드가 답을 멈췄습니다.

이 안내서가 보이려 한 것은 한 가지입니다. LLM은 그래프를 대신하지 않습니다. 그래프는 LLM의 답을 받쳐 주고, 근거가 없을 때 멈추게 합니다. 그리고 그 그래프의 이름과 관계, 검사 규칙의 정보는 1권에서 만든 온톨로지입니다.

부록 A. 용어

용어	뜻
속성 그래프	노드와 관계에 「칸 이름: 값」 쌍을 바로 붙이는 그래프 모델
노드	그래프의 대상 하나. 카드 한 장
레이블	노드의 종류 표시. 여러 개 붙일 수 있음. 카드의 색 스티커
관계	두 노드를 잇는 방향 있는 선. 이름(유형)이 하나
관계 유형	관계의 이름. DRIVES, PART_OF 등
속성	노드나 관계에 붙인 「칸 이름: 값」
고유 제약	같은 값을 가진 노드를 하나만 두게 하는 약속
펼치기	계층(「펌프는 설비」)을 적재할 때 레이블로 미리 붙여 두는 일
Cypher	Neo4j의 그래프 질의 언어
GQL	ISO/IEC 39075:2024 그래프 질의 언어 표준
임베딩	문장을 뜻이 반영된 숫자 목록으로 바꾼 것
벡터 검색	숫자 목록끼리 견주어 뜻이 가까운 글을 찾는 일
벡터 인덱스	가까운 숫자 목록을 빨리 찾게 하는 색인
RAG	질문과 관련된 자료를 찾아 LLM에게 근거로 건네는 방식
GraphRAG	그래프의 개체와 관계도 근거로 건네는 RAG
개체 연결	질문 속 낱말을 그래프의 노드와 잇는 일
온도	LLM이 낱말을 얼마나 무작위로 고르는지 정하는 값. 0이면 되도록 같은 답
Text-to-Cypher	자연어 질문을 LLM이 Cypher로 바꾸는 방식
골든 질문	정답과 근거를 사람이 미리 적어 둔 평가용 질문

부록 B. 참고 문헌

- Neo4j, Inc. Neo4j Cypher Manual (5.x). <https://neo4j.com/docs/cypher-manual/current/>
- Neo4j, Inc. Vector indexes, Neo4j Cypher Manual (5.x). <https://neo4j.com/docs/cypher-manual/current/indexes/semantic-indexes/vector-indexes/>
- ISO/IEC 39075:2024. Information technology, Database languages, GQL.
- Lewis, P. 외 (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS 2020.
- Edge, D. 외 (2024). From Local to Global: A Graph RAG Approach to Query-Focused Summarization. arXiv:2404.16130.
- W3C (2009). SKOS Simple Knowledge Organization System Reference. <https://www.w3.org/TR/skos-reference/>
- Qwen Team (2024). Qwen2.5 모델 카드(Qwen/Qwen2.5-7B-Instruct, Apache-2.0). <https://huggingface.co/Qwen/Qwen2.5-7B-Instruct>
- Nomic AI (2024). nomic-embed-text 모델 카드(Apache-2.0). <https://huggingface.co/nomic-ai/nomic-embed-text-v1.5>
- OntoCraft (2026). 처음 만나는 온톨로지: 데이터 실무자를 위한 입문(1권, CC BY 4.0).

부록 C. 예제 파일 전문과 실행 결과

C.1 notes.json

```
[
  {"id": "N1", "date": "2026-09-15", "text": "P-101 냉각수 공급 펌프에서 진동 7.4 mm/s 를 측정했다. 베어링 쪽에서 소음이 들려 베어링 교체를 권고한다."},
  {"id": "N2", "date": "2026-09-16", "text": "작업 지시 W0-001 에 따라 P-101 의 베어링 BRG-101 을 교체했다. 교체 뒤 진동은 2.0 mm/s 로 내려갔다."},
  {"id": "N3", "date": "2026-09-15", "text": "P-102 진동 측정값이 숫자로 기록되지 않고 현장 표기로 '높음'만 남았다. 다시 측정해야 한다."},
  {"id": "N4", "date": "2026-08-20", "text": "M-101 모터 정기 점검. 절연 저항과 전류 모두 이상 없음."},
  {"id": "N5", "date": "2026-09-20", "text": "P-102 주변 바닥에 물기가 있어 썬 누수가 의심된다. 다음 정기 점검 때 확인한다."},
  {"id": "N6", "date": "2026-09-18", "text": "베어링 마모 점검을 요청하는 작업 지시 W0-002 가 들어왔으나 대상 설비가 적혀 있지 않다."}
]
```

C.2 common.py

```

"""2권 예제의 공통 설정입니다. 모든 이름과 값은 지어낸 것입니다."""
import json, os, pathlib, urllib.request
from neo4j import GraphDatabase

HERE = pathlib.Path(__file__).parent
BOOK1 = HERE.parent.parent / "community" / "examples"           # 1권 예제(plant.ttl, failu
                        re-modes.ttl)
URI = os.environ.get("NEO4J_URI", "bolt://localhost:7688")
AUTH = ("neo4j", os.environ.get("NEO4J_PASSWORD", "guide-demo-pass"))
OLLAMA = os.environ.get("OLLAMA_URL", "http://localhost:11434")
EMBED_MODEL = "nomic-embed-text"      # 768차원
CHAT_MODEL = os.environ.get("CHAT_MODEL", "qwen2.5:7b")

def driver():
    return GraphDatabase.driver(URI, auth=AUTH)

def run(cypher, **params):
    with driver() as d:
        records, _, _ = d.execute_query(cypher, params, database_="neo4j")
        return [r.data() for r in records]

def ollama(path, body):
    req = urllib.request.Request(OLLAMA + path, data=json.dumps(body).encode(), headers=
        {"Content-Type": "application/json"})
    with urllib.request.urlopen(req, timeout=300) as r:
        return json.loads(r.read())

def embed(text):
    return ollama("/api/embeddings", {"model": EMBED_MODEL, "prompt": text})["embeddin
        g"]

def chat(system, user):
    out = ollama("/api/chat", {"model": CHAT_MODEL, "stream": False,
        "options": {"temperature": 0, "seed": 7},
        "messages": [{"role": "system", "content": system}, {"rol
            e": "user", "content": user}]}))
    return out["message"]["content"].strip()

```

C.3 load.py

```

"""1권의 RDF(plant.ttl, failure-modes.ttl)와 정비 일지를 Neo4j 속성 그래프로 옮깁니다.
- 온톨로지의 클래스 계층은 레이블로 펼칩니다(펌프 노드에 :Pump:Equipment).
- 관계 이름은 온톨로지의 속성 이름을 대문자 뱀 표기로 바꿉니다(partOf → PART_OF).
- 정비 일지는 :Note 노드로 넣고, 본문에 나오는 태그로 :MENTIONS 를 잇고, 임베딩을 붙입니다.
"""

```

```

import json, re
from rdflib import Graph, RDF, RDFS, Literal, URIRef
from rdflib.namespace import SKOS
from common import BOOK1, HERE, run, embed

EX = "https://example.org/plant/"
local = lambda u: str(u).replace(EX, "")
snake = lambda name: re.sub(r"(?!^)(?=[A-Z])", "_", name).upper()

g = Graph().parse(BOOK1 / "plant.ttl").parse(BOOK1 / "failure-modes.ttl")

def ancestors(c):
    out, todo = [], [c]
    while todo:
        x = todo.pop()
        for sup in g.objects(x, RDFS.subClassOf):
            if sup not in out:
                out.append(sup); todo.append(sup)
    return out

run("MATCH (n) DETACH DELETE n")
for stmt in ["CREATE CONSTRAINT node_id IF NOT EXISTS FOR (n:Resource) REQUIRE n.id IS UNIQUE",
             "CREATE CONSTRAINT note_id IF NOT EXISTS FOR (n:Note) REQUIRE n.id IS UNIQUE"]:
    run(stmt)

# 개체: 클래스가 example.org 안에 있는 것만(스키마 선언은 건너뛴)
classes = {s for s in g.subjects(RDF.type, None) if str(s).startswith(EX)}
n_nodes = 0
for s in classes:
    types = [t for t in g.objects(s, RDF.type) if str(t).startswith(EX)]
    if not types:
        continue
    labels = {local(t) for t in types} | {local(a) for t in types for a in ancestors(t)}
    props = {"id": local(s)}
    for p, o in g.predicate_objects(s):
        if isinstance(o, Literal) and p != RDF.type:
            key = "label" if p == RDFS.label else local(p)
            v = o.toPython()
            if hasattr(v, "isoformat"):
                v = v.isoformat()
            elif type(v).__name__ == "Decimal":
                v = float(v)

```

```

        props[key] = v
    run(f"MERGE (n:Resource {{id:$id}}) SET n += $props SET n:{':'.join(sorted(label
        s))}", id=props["id"], props=props)
    n_nodes += 1

# 고장 모드 개념(SKOS)도 같은 노드로 합치고 다른 이름(altLabel)을 붙입니다
for c in g.subjects(RDF.type, SKOS.Concept):
    alts = [str(a) for a in g.objects(c, SKOS.altLabel)]
    pref = str(next(g.objects(c, SKOS.prefLabel)))
    run("MERGE (n:Resource {id:$id}) SET n:FailureMode, n.label=coalesce(n.label,$pref),
        n.altLabels=$alts", id=local(c), pref=pref, alts=alts)
    for b in g.objects(c, SKOS.broader):
        run("MATCH (a:Resource {id:$a}) MERGE (b:Resource {id:$b}) MERGE (a)-[:BROADER]-
            >(b)", a=local(c), b=local(b))

# 관계
n_rels = 0
for s, p, o in g:
    if isinstance(o, URIRef) and str(o).startswith(EX) and str(s).startswith(EX) and p !
        = RDF.type and p.startswith(EX):
        run(f"MATCH (a:Resource {{id:$a}}), (b:Resource {{id:$b}}) MERGE (a)-[:{snake(lo
            cal(p))}]>(b)", a=local(s), b=local(o))
        n_rels += 1

# 정비 일자: 태그가 본문에 나오면 MENTIONS 로 잇고 임베딩을 붙입니다
notes = json.load(open(HERE / "notes.json"))
tags = [r["id"] for r in run("MATCH (n:Resource) RETURN n.id AS id")]
for nt in notes:
    run("MERGE (n:Note {id:$id}) SET n.date=$date, n.text=$text, n.embedding=$emb", id=n
        t["id"], date=nt["date"], text=nt["text"], emb=embed(nt["text"]))
    for t in tags:
        if re.search(rf"(?![A-Za-z0-9-]){re.escape(t)}(?![A-Za-z0-9-])", nt["text"]):
            run("MATCH (n:Note {id:$n}), (r:Resource {id:$r}) MERGE (n)-[:MENTIONS]->
                (r)", n=nt["id"], r=t)
run("CREATE VECTOR INDEX note_embedding IF NOT EXISTS FOR (n:Note) ON n.embedding "
    "OPTIONS {indexConfig: {\`vector.dimensions\`: 768, \`vector.similarity_function\`: 'cos
    ine'}}")
run("CALL db.awaitIndexes(60)")

c = run("MATCH (n) WITH count(n) AS nodes MATCH ()-[r]->() RETURN nodes, count(r) AS rel
    s")[0]
print(f"개체 노드 {n_nodes}개, 온톨로지 관계 {n_rels}개 적재 · 전체 노드 {c['nodes']}개, 전체 관계 {c
    ['rels']}개")
for r in run("MATCH (n:Resource) WHERE n.id IN ['P-101','M-101','BRG-101','FM-wear'] RET
    URN n.id AS id, labels(n) AS labels ORDER BY id"):
    print(" ", r["id"], sorted(r["labels"]))
for r in run("MATCH (n:Note)-[:MENTIONS]->(r) RETURN n.id AS note, collect(r.id) AS ment
    ions ORDER BY note"):
    print(" ", r["note"], "→", sorted(r["mentions"]))

```

C.4 queries.py

"""1권의 역량 질문을 Cypher 로 묻고, SHACL 과 같은 품질 검사를 Cypher 로 합니다."""

```
from common import run
```

```
Q = {
```

```
    "C1 펌프 P-101 의 부품과 그 고장 모드": ""
```

```
    MATCH (p:Pump {tag:'P-101'})<-[:PART_OF]-(part:Component)-[:HAS_FAILURE_MODE]->(fm:FailureMode)
```

```
    RETURN part.id AS part, fm.label AS failure_mode""",
```

```
    "C2 진동이 4.5 mm/s 를 넘은 측정": ""
```

```
    MATCH (m:Measurement)-[:MEASURED_ON]->(e:Equipment)
```

```
    WHERE m.velocityRms IS :: FLOAT AND m.velocityRms > 4.5
```

```
    RETURN e.id AS equipment, m.velocityRms AS v, m.measuredAt AS t ORDER BY t""",
```

```
    "C3 설비 목록(레이블로 펼친 계층 덕분에 추론 없이 나옴)": ""
```

```
    MATCH (e:Equipment) RETURN e.id AS equipment, [l IN labels(e) WHERE l <> 'Resource'] AS labels ORDER BY equipment""",
```

```
    "C4 대상 설비가 없는 작업 지시": ""
```

```
    MATCH (wo:WorkOrder) WHERE NOT (wo)-[:ON_EQUIPMENT]->(:Equipment)
```

```
    RETURN wo.id AS work_order, wo.label AS label""",
```

```
    "C5 경로: 모터에서 고장 모드까지(2~3단계)": ""
```

```
    MATCH path = (m:Motor {tag:'M-101'})-[:DRIVES]->(:Pump)<-[:PART_OF]-(c:Component)-[:HAS_FAILURE_MODE]->(fm)
```

```
    RETURN [n IN nodes(path) | n.id] AS path""",
```

```
    "C6 「기계적 고장」 아래 모든 고장 모드(가변 길이)": ""
```

```
    MATCH (c)-[:BROADER*1..]->(:Resource {id:'FM-mechanical'}) RETURN c.label AS label ORDER BY label""",
```

```
    "C7 설비별 정비 일지 수": ""
```

```
    MATCH (e:Equipment) OPTIONAL MATCH (n:Note)-[:MENTIONS]->(e)
```

```
    RETURN e.id AS equipment, count(n) AS notes ORDER BY equipment""",
```

```
}
```

```
CHECKS = {
```

```
    "K1 [위반] 숫자가 아닌 진동값": "MATCH (m:Measurement) WHERE NOT m.velocityRms IS :: FLOAT RETURN m.id AS id, m.velocityRms AS value",
```

```
    "K2 [위반] 대상 설비 없는 작업 지시": "MATCH (wo:WorkOrder) WHERE NOT (wo)-[:ON_EQUIPMENT]->() RETURN wo.id AS id",
```

```
    "K3 [위반] 정의역 밖의 PART_OF(주어가 부품이 아님)": "MATCH (a)-[:PART_OF]->() WHERE NOT a:Component RETURN a.id AS id, labels(a) AS labels",
```

```
    "K4 [참고] 어떤 일지에도 나오지 않는 설비": "MATCH (e:Equipment) WHERE NOT ()-[:MENTIONS]->(e) RETURN e.id AS id",
```

```
}
```

```
def show(title, rows):
```

```
    print("\n==", title)
```

```
    if not rows:
```

```
        print("    (결과 없음)")
```

```
    for r in rows:
```

```
        print("    ", " | ".join(str(v) for v in r.values()))
```

```
for k, q in Q.items():
```

```
    show(k, run(q))
```

```
for k, q in CHECKS.items():
```

```
    show(k, run(q))
```

C.5 graphrag.py

"""같은 질문을 벡터 RAG(정비 일지만 검색)와 GraphRAG(일지 + 그래프 관계)로 답해 비교합니다.
그래프에 필요한 관계가 없으면 LLM 을 부르지 않고 답을 멈춥니다. LLM 은 로컬 Ollama 모델이고 온도는 0 입니다."""

```
import re, sys
```

```
from common import run, embed, chat, CHAT_MODEL
```

```
SYSTEM = ("너는 정비 기록 도우미다. 아래 [근거]에 있는 내용만으로 한국어로 짧게 답한다. "
```

```
"문장마다 근거 번호를 대괄호로 붙인다(예: [N2], [G1]). 근거에 없는 것은 '근거 없음'이라고 쓴다.")
```

```
def vector_notes(q, k=3):
```

```
    return run("CALL db.index.vector.queryNodes('note_embedding', $k, $e) YIELD node, score "
```

```
                "RETURN node.id AS id, node.text AS text, round(score, 3) AS score", k=k,  
                e=embed(q))
```

```
def link_entities(q):
```

```
    """질문에 나온 태그·이름·다른 이름을 그래프 노드와 잇습니다."""
```

```
    rows = run("MATCH (n:Resource) RETURN n.id AS id, coalesce(n.label, '') AS label, coa  
                lesce(n.altLabels, []) AS alts")
```

```
    found = []
```

```
    for r in rows:
```

```
        names = [r["id"], r["label"]] + list(r["alts"])
```

```
        if any(nm and re.search(rf"(?![A-Za-z0-9-])\{re.escape(nm)\}(?![A-Za-z0-9-])", q)  
               for nm in names):
```

```
            found.append(r["id"])
```

```
    return found
```

```
def graph_facts(ids):
```

```
    rows = run("""UNWIND $ids AS i MATCH (a:Resource {id:i})-[r]-(b)
```

```
                WHERE NOT b:Note RETURN DISTINCT a.id AS a, type(r) AS rel, startNode  
                (r).id AS s, b.id AS b, coalesce(b.label, '') AS bl""", ids=ids)
```

```
    facts = sorted({f"{r['s']} -{r['rel']}-> {r['b']} if r['s']==r['a'] else r['a']}" if  
                    r['s']==r['a'] else f"{r['b']} -{r['rel']}-> {r['a']}" for r in rows})
```

```
    return facts
```

```
def notes_for(ids):
```

```
    return run("UNWIND $ids AS i MATCH (n:Note)-[:MENTIONS]->(:Resource {id:i}) RETURN D  
                ISTINCT n.id AS id, n.text AS text ORDER BY id", ids=ids)
```

```
def guard(q, ids):
```

```
    """답에 꼭 필요한 관계가 그래프에 없으면 답을 멈춥니다(예: 작업 지시의 대상 설비)."""
```

```
    if "어느 설비" in q or "대상 설비" in q:
```

```
        for i in ids:
```

```
            if run("MATCH (w:WorkOrder {id:$i}) RETURN count(w) AS c", i=i)[0]["c"]:
```

```
                if not run("MATCH (:WorkOrder {id:$i})-[:ON_EQUIPMENT]->(e) RETURN e.id  
AS e", i=i):
```

```

        return f"답할 수 없음: 그래프에 {i}의 대상 설비(ON_EQUIPMENT)가 없습니다. 작업
        지시 원본을 확인해야 합니다."
    return None

def vector_rag(q):
    ns = vector_notes(q)
    ctx = "\n".join(f"[{n['id']}] {n['text']}" for n in ns)
    return [f"[{n['id']}]({n['score']})" for n in ns], chat(SYSTEM, f"[근거]\n{ctx}\n\n[질문]
    {q}")

def graph_rag(q):
    ids = link_entities(q)
    stop = guard(q, ids)
    if stop:
        return ids, [], stop
    facts = graph_facts(ids)
    ns = {n["id"]: n for n in notes_for(ids)}
    for n in vector_notes(q):
        ns.setdefault(n["id"], n)
    ctx = "\n".join([f"[G{i+1}] {f}" for i, f in enumerate(facts)] + [f"[{k}] {v['text']}" for k, v in sorted(ns.items())])
    return ids, facts, chat(SYSTEM, f"[근거]\n{ctx}\n\n[질문] {q}")

QUESTIONS = [
    "P-101의 베어링을 교체한 뒤 진동은 얼마였고, 그 펌프를 구동하는 모터는 무엇인가?",
    "셀 누수가 의심되는 설비는 어디인가?",
    "W0-002는 어느 설비의 작업 지시인가?",
]

if __name__ == "__main__":
    print("LLM:", CHAT_MODEL)
    for q in QUESTIONS:
        print("\n### 질문:", q)
        hits, a = vector_rag(q)
        print("[벡터 RAG] 찾은 일지:", ", ".join(hits))
        print("  답:", a.replace("\n", " "))
        ids, facts, a = graph_rag(q)
        print("[GraphRAG] 연결한 개체:", ids, ". 그래프 사실", len(facts), "개")
        for f in facts:
            print("  ", f)
        print("  답:", a.replace("\n", " "))
        if facts and not re.search(r"\[(N|G)\d+\]", a):
            print("  검사: 답에 근거 번호가 없습니다. 이 답은 그대로 내보내지 않고 다시 묻거나 사람이 확인합
            니다.")

```

C.6 text2cypher.py

"""Text-to-Cypher: LLM 이 질문을 Cypher 로 바꾸고, 실행 전에 두 가지를 검사합니다.

(1) 읽기 전용인가(쓰기·관리 구문 금지) (2) 그래프에 있는 레이블·관계 유형만 썼는가. 실행은 읽기 모드로 한다."""

```
import re
```

```
from common import driver, run, chat, CHAT_MODEL
```

```
labels = {r["l"] for r in run("CALL db.labels() YIELD label RETURN label AS l")}
```

```
rels = {r["r"] for r in run("CALL db.relationshipTypes() YIELD relationshipType RETURN relationshipType AS r")}
```

```
SCHEMA = """노드 레이블: Resource, Equipment, Pump, Motor, Component, Bearing, FailureMode, WorkOrder, Measurement, Note
```

```
관계: (Motor)-[:DRIVES]->(Pump), (Component)-[:PART_OF]->(Equipment), (Component)-[:HAS_FAILURE_MODE]->(FailureMode),
```

```
(WorkOrder)-[:ON_EQUIPMENT]->(Equipment), (WorkOrder)-[:ADDRESSES]->(FailureMode), (Measurement)-[:MEASURED_ON]->(Equipment),
```

```
(Note)-[:MENTIONS]->(Resource), (FailureMode)-[:BROADER]->(FailureMode)
```

```
속성: 모든 Resource 는 id, 설비는 tag, 이름은 label, Measurement 는 velocityRms, measuredAt, Note 는 date, text"""
```

```
SYSTEM = "너는 Neo4j Cypher 작성기다. 주어진 스키마만 써서 질문에 답하는 Cypher 한 개만 출력한다. 설명과 코드 블록 표시는 쓰지 않는다."
```

```
WRITE = re.compile(r"\b(CREATE|MERGE|DELETE|DETACH|SET|REMOVE|DROP|LOAD\s+CSV|FOREACH)\b|CALL\s+(dbms|apoc)", re.I)
```

```
def check(cy):
```

```
    if WRITE.search(cy):
```

```
        return "막음: 쓰기·관리 구문이 들어 있습니다"
```

```
    used_l = set(re.findall(r"\\s*\\w*\\s*:\\s*([A-Za-z_]+)", cy)) # 괄호 안의 레이블
```

```
    used_r = set(re.findall(r"\\s*\\w*\\s*:\\s*([A-Za-z_]+)", cy)) # 대괄호 안의 관계 유형
```

```
    bad = sorted((used_l - labels) | (used_r - rels))
```

```
    return f"막음: 그래프에 없는 이름 {bad}" if bad else "통과"
```

```
def ask(q):
```

```
    cy = chat(SYSTEM, f"[스키마]\n{SCHEMA}\n\n[질문] {q}").strip().strip("`").removeprefix("cypher").strip()
```

```
    verdict = check(cy)
```

```
    rows = []
```

```
    if verdict == "통과":
```

```
        with driver() as d:
```

```
            records, _, _ = d.execute_query(cy, database_="neo4j", routing_="r")
```

```
            rows = [r.data() for r in records]
```

```
    return cy, verdict, rows
```

```
if __name__ == "__main__":
```

```
    print("LLM:", CHAT_MODEL)
```

```
    for q in ["P-102 에서 측정한 진동값과 측정 시각을 모두 보여 줘",
```

```
              "모터 M-101 이 구동하는 펌프의 부품은 무엇인가?",
```

```
              "그래프의 모든 노드와 관계를 지워 줘"]:
```

```
        cy, verdict, rows = ask(q)
```

```

print("\n### 질문:", q)
print("  Cypher:", " ".join(cy.split()))
print("  검사:", verdict)
for r in rows:
    print("  결과:", r)
if verdict == "통과" and not rows:
    print("  결과: (없음) · 검사를 통과해도 관계 방향이나 조건이 틀리면 빈 결과가 나옵니다. 「없다」로  
    답하지 않습니다.")

```

C.7 run_all.sh

```

#!/bin/sh
# 2권 예제를 처음부터 차례로 돌려 출력 파일을 만듭니다. 필요: Neo4j(bolt://localhost:7688), Ollama(qwe  
    n2.5:7b, nomic-embed-text)

set -e
cd "$(dirname "$0")"
PY="uv run --quiet --with neo4j --with rdflib python"
$PY load.py > load_output.txt
$PY queries.py > queries_output.txt
$PY graphrag.py > graphrag_output.txt
$PY text2cypher.py > text2cypher_output.txt
echo done

```

C.8 load_output.txt

```

개체 노드 11개, 온톨로지 관계 11개 적재 · 전체 노드 20개, 전체 관계 22개
BRG-101 ['Bearing', 'Component', 'Resource']
FM-wear ['FailureMode', 'Resource']
M-101 ['Equipment', 'Motor', 'Resource']
P-101 ['Equipment', 'Pump', 'Resource']
N1 → ['P-101']
N2 → ['BRG-101', 'P-101', 'W0-001']
N3 → ['P-102']
N4 → ['M-101']
N5 → ['P-102']
N6 → ['W0-002']

```

C.9 queries_output.txt

== C1 펌프 P-101 의 부품과 그 고장 모드
BRG-101 | 베어링 마모

== C2 진동이 4.5 mm/s 를 넘는 측정
P-101 | 7.4 | 2026-09-15T09:00:00

== C3 설비 목록(레이블로 펼친 계층 덕분에 추론 없이 나옴)
M-101 | ['Equipment', 'Motor']
P-101 | ['Equipment', 'Pump']
P-102 | ['Equipment', 'Pump']

== C4 대상 설비가 없는 작업 지시
W0-002 | 대상 설비를 적지 않은 작업 지시

== C5 경로: 모터에서 고장 모드까지(2~3단계)
['M-101', 'P-101', 'BRG-101', 'FM-wear']

== C6 「기계적 고장」 아래 모든 고장 모드(가변 길이)
마모
베어링 마모
실 누설

== C7 설비별 정비 일지 수
M-101 | 1
P-101 | 2
P-102 | 2

== K1 [위반] 숫자가 아닌 진동값
MS-3 | 높음

== K2 [위반] 대상 설비 없는 작업 지시
W0-002

== K3 [위반] 정의역 밖의 PART_OF(주어가 부품이 아님)
(결과 없음)

== K4 [참고] 어떤 일지에도 나오지 않는 설비
(결과 없음)

C.10 graphrag_output.txt

LLM: qwen2.5:7b

질문: P-101 의 베어링을 교체한 뒤 진동은 얼마였고, 그 펌프를 구동하는 모터는 무엇인가?

[벡터 RAG] 찾은 일지: N4(0.95), N1(0.941), N5(0.934)

답: 베어링을 교체한 뒤의 진동 수치는 근거에서 언급되어 있지 않으므로 「근거 없음」입니다. P-101 펌프를 구동하는 모터에 대한 정보도 근거에서 찾을 수 없어 「근거 없음」입니다.

[GraphRAG] 연결한 개체: ['P-101'] · 그래프 사실 5 개

BRG-101 -PART_OF-> P-101

M-101 -DRIVES-> P-101

MS-1 -MEASURED_ON-> P-101

MS-2 -MEASURED_ON-> P-101

W0-001 -ON_EQUIPMENT-> P-101

답: P-101의 베어링을 교체한 뒤 진동은 2.0 mm/s였습니다. P-101을 구동하는 모터는 M-101입니다.

검사: 답에 근거 번호가 없습니다. 이 답은 그대로 내보내지 않고 다시 묻거나 사람이 확인합니다.

질문: 썰 누수가 의심되는 설비는 어디인가?

[벡터 RAG] 찾은 일지: N3(0.865), N5(0.855), N1(0.845)

답: [N5] P-102

[GraphRAG] 연결한 개체: ['FM-seal'] · 그래프 사실 1 개

FM-seal -BROADER-> FM-mechanical

답: P-102 [N5]

질문: W0-002 는 어느 설비의 작업 지시인가?

[벡터 RAG] 찾은 일지: N6(0.969), N3(0.87), N5(0.869)

답: [N6] 대상 설비가 적혀 있지 않으므로 알 수 없다.

[GraphRAG] 연결한 개체: ['W0-002'] · 그래프 사실 0 개

답: 답할 수 없음: 그래프에 W0-002 의 대상 설비(ON_EQUIPMENT)가 없습니다. 작업 지시 원본을 확인해야 합니다.

C.11 text2cypher_output.txt

LLM: qwen2.5:7b

질문: P-102 에서 측정한 진동값과 측정 시각을 모두 보여 줘

Cypher: MATCH (e:Equipment {tag: 'P-102'})<-[:MEASURED_ON]-(m:Measurement) RETURN m.velocityRms, m.measuredAt

검사: 통과

결과: {'m.velocityRms': '높음', 'm.measuredAt': '2026-09-15T09:10:00'}

질문: 모터 M-101 이 구동하는 펌프의 부품은 무엇인가?

Cypher: MATCH (m:Motor {tag: 'M-101'})-[:DRIVES]->(p:Pump)-[:PART_OF]->(e:Equipment) RETURN e

검사: 통과

결과: (없음) · 검사를 통과해도 관계 방향이나 조건이 틀리면 빈 결과가 나옵니다. 「없다」로 답하지 않습니다.

질문: 그래프의 모든 노드와 관계를 지워 줘

Cypher: MATCH (n) DETACH DELETE n

검사: 막음: 쓰기·관리 구문이 들어 있습니다