

처음 만나는 온톨로지

데이터 실무자를 위한 입문 · 커뮤니티판 · 2026-10-12

0. 이 안내서에 대해

1. 왜 온톨로지인가

2. 온톨로지의 부품

3. 범위부터: 역량 질문

4. RDF와 Turtle로 적기

5. RDFS와 OWL: 계층과 추론

6. SPARQL로 묻기

7. SHACL로 검증하기

8. 남이 만든 것 다시 쓰기

9. 속성 그래프, 그리고 LLM과 함께 쓰기

10. 흔한 실수 열 가지

11. 다음 걸음

부록 A. 용어

부록 B. 참고 문헌

부록 C. 예제 파일 전문

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 `rdflib`, `pySHACL`, `owlrl` 로 실제로 돌려 얻은 것입니다.

0. 이 안내서에 대해

0.1 누구를 위한 글인가

이 안내서는 데이터를 다루지만 온톨로지는 처음인 실무자를 위해 썼습니다. 엑셀, SQL, 파이썬 정도를 써 본 사람이면 읽을 수 있습니다. 끝까지 읽으면 작은 온톨로지를 직접 적고, 질의하고, 검증할 수 있습니다.

다루는 순서는 이렇습니다. 왜 필요한지(1장), 무엇으로 이루어지는지(2장), 범위를 어떻게 정하는지(3장)를 먼저 봅니다. 그다음 RDF로 적고(4장), 계층과 추론을 붙이고(5장), SPARQL로 묻고(6장), SHACL로 검증합니다(7장). 끝으로 남의 온톨로지를 다시 쓰는 법(8장), 속성 그래프와 LLM과의 관계(9장), 흔한 실수(10장)를 봅니다.

0.2 예제는 하나로 이어집니다

모든 장의 예제는 「가상 공장 A」의 설비 정비 데이터 하나로 이어집니다. 펌프 두 대, 모터 한 대, 베어링 두 개, 고장 모드 하나, 작업 지시 두 건, 진동 측정 세 건이 전부입니다. 이름과 값은 모두 지어낸 것입니다.

일부러 오류도 두 개 넣었습니다. 대상 설비가 빠진 작업 지시 하나와, 숫자 대신 「높음」이라고 적힌 진동값 하나입니다. 7장에서 이 오류를 찾습니다.

예제 파일 네 개(`plant.ttl`, `shapes.ttl`, `failure-modes.ttl`, `run.py`)의 전문은 부록 C에 있습니다. 책에 실은 실행 결과는 모두 `run.py`를 `rdflib` 7, `pySHACL`, `owlrl`로 실제로 돌려 얻은 것입니다.

```
pip install rdflib pyshacl owlrl
python run.py
```

0.3 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 온톨로지인가

1.1 데이터 실무자가 매일 겪는 문제

가상 공장 A의 정비 담당자가 이렇게 묻는다고 합시다. 「지난달 진동이 높았던 펌프 가운데, 베어링 마모로 작업 지시가 나간 것은 무엇인가?」

이 질문에 답하려면 세 곳의 데이터를 이어야 합니다. 진동 측정 시스템, 설비 대장, 정비 작업 지시입니다. 실제 현장에서는 세 곳이 같은 설비를 다르게 부릅니다. 측정 시스템은 「P101」, 설비 대장은 「P-101」, 작업 지시는 「냉각수 펌프 1 호기」라고 적는 식입니다. 「펌프」가 무엇인지도 시스템마다 다릅니다. 어떤 곳은 모터를 포함해 펌프로 부르고, 어떤 곳은 따로 셉니다.

사람은 이 차이를 머릿속으로 맞춰 가며 답합니다. 그런데 그 맞춤 규칙은 문서로 남지 않고 담당자가 바뀌면 사라집니다. 질의를 짜는 사람마다 맞춤도 조금씩 다릅니다.

1.2 온톨로지가 하는 일

Ontology — 온톨로지: 한 분야에서 쓰는 개념과 그 개념 사이의 관계를, 사람과 기계가 같은 뜻으로 읽도록 명시적으로 적어 둔 것입니다.

온톨로지는 위의 맞춤 규칙을 데이터 밖이 아니라 데이터 옆에 적어 둡니다. 「펌프는 설비의 한 종류다」, 「베어링은 설비의 부품이다」, 「작업 지시는 대상 설비를 하나 이상 가진다」 같은 문장입니다. 이렇게 적어 두면 세 가지가 달라집니다.

1. **같은 뜻을 공유합니다.** 「설비」를 물으면 펌프와 모터가 함께 나옵니다. 질의를 짜는 사람이 계층을 외우지 않아도 됩니다(5장).
2. **연결을 따라 묻습니다.** 측정 → 설비 → 부품 → 고장 모드 → 작업 지시로 이어지는 질문을 한 번에 던집니다(6장).
3. **규칙을 검사합니다.** 「대상 설비 없는 작업 지시」 같은 위반을 데이터가 들어올 때 찾습니다(7장).

1.3 데이터베이스 스키마와 무엇이 다른가

관계형 데이터베이스의 스키마도 구조를 적습니다. 차이는 목적에 있습니다. 스키마는 한 시스템이 데이터를 저장하고 꺼내기 위한 구조입니다. 온톨로지는 여러 시스템이 같은 개념을 같은 뜻으로 쓰기 위한 약속입니다.

그래서 온톨로지는 테이블 하나의 열이 아니라 개념에서 출발합니다. 그리고 「이 개념은 저 개념의 한 종류다」 같은 의미 관계를 기계가 추론에 쓸 수 있게 적습니다.

확인 문제

문제 1. 정비 기록 시스템이 「P101」, 설비 대장이 「P-101」이라고 적을 때 생기는 문제를 한 문장으로 적으십시오.

답. 같은 설비가 두 개로 세어지거나 이어지지 않아, 측정과 정비 기록을 잇는 질문에 답하지 못합니다.

문제 2. 온톨로지를 쓰지 않는 편이 나은 경우를 하나 드십시오.

답. 시스템이 두세 개뿐이고 질문이 고정되어 있을 때입니다. 조인 테이블 하나가 더 싸고 빠릅니다.

이 장이 닿지 않는 곳: 온톨로지가 모든 데이터 통합 문제의 답은 아닙니다. 시스템이 두세 개이고 질문이 고정되어 있다면 잘 짠 조인 테이블 하나가 더 싸고 빠릅니다. 온톨로지는 질문이 계속 늘고, 시스템이 바뀌고, 뜻을 맞춰야 하는 사람이 많을 때 값을 합니다.

2. 온톨로지의 부품

2.1 다섯 가지 부품

부품	뜻	가상 공장 A 예
Class — 클래스	같은 성질을 가진 것들의 모임	설비, 펌프, 베어링, 고장 모드
Individual — 개체(인스턴스)	클래스에 속하는 실제 하나	P-101, BRG-101, WO-001
Object property — 관계	개체와 개체를 잇는 것	부품이다(partOf), 구동한다(drives)
Datatype property — 값 속성	개체에 숫자·글자·날짜 값을 붙이는 것	진동 속도, 측정 시각, 태그
Axiom — 공리(제약)	항상 참이라고 약속한 문장	「펌프는 설비의 한 종류», 「partOf의 주어는 부품」

2.2 계층: 「종류」와 「부분」을 섞지 않습니다

온톨로지에서 가장 자주 쓰는 관계는 두 가지입니다.

- **is-a — 종류 관계**: 「펌프는 설비의 한 종류다.」 하위 클래스의 모든 개체는 상위 클래스의 개체이기도 합니다. RDF에서는 `rdfs:subClassOf`로 적습니다.
- **part-of — 부분 관계**: 「베어링 BRG-101은 펌프 P-101의 부품이다.」 베어링은 펌프의 한 종류가 아닙니다. 별도의 관계(예제의 `ex:partOf`)로 적습니다.

두 관계를 섞으면 수를 잘못 셉니다. 베어링을 펌프의 하위 클래스로 적으면 「펌프가 몇 대인가」를 물을 때 베어링까지 펌프로 셉니다.

2.3 이웃 개념과의 차이

이름	무엇을 적는가	온톨로지와의 차이
Taxonomy — 분류 체계	개념의 위아래 계층	계층만 있고 관계와 제약이 없습니다
Thesaurus — 시소러스	같은 말, 넓은 말, 좁은 말, 관련어	낱말 사이 관계이며 개체를 다루지 않습니다. SKOS로 적습니다
Data model — 데이터 모델	한 시스템의 저장 구조	시스템에 묶여 있고 추론을 하지 않습니다
Knowledge graph — 지식 그래프	개체와 관계의 실제 데이터	온톨로지가 뜻을 정하고, 지식 그래프는 그 뜻에 맞춰 쌓인 사실입니다

확인 문제

문제 1. 「베어링은 펌프다」라고 적으면 어떤 질의가 틀린 답을 냅니까?

답. 「펌프가 몇 대인가」 같은 질의가 베어링까지 펌프로 셉니다. 종류와 부분 관계를 섞었기 때문입니다.

문제 2. 「베어링 6205」를 클래스로 돌지 개체로 돌지는 무엇이 정합니까?

답. 답해야 할 질문이 정합니다. 형번별 집계기 필요하다면 클래스, 날개 추적이 필요하다면 개체로 둡니다.

이 장이 닿지 않는 곳: 클래스와 개체의 경계는 목적에 따라 달라집니다. 「베어링 6205」를 클래스(형번)로 돌지 개체(제품 하나)로 돌지는 질문이 정합합니다. 이 판단은 3장의 역량 질문에서 시작합니다.

3. 범위부터: 역량 질문

3.1 무엇을 넣고 무엇을 뺄 것인가

온톨로지를 처음 만들 때 가장 흔한 실패는 너무 많이 넣는 것입니다. 「설비에 관한 모든 것」을 적으려다 끝나지 않습니다. 그래서 범위를 질문으로 정합니다.

Competency Question — 역량 질문(CQ): 온톨로지가 답할 수 있어야 하는 업무 질문입니다. Grüninger와 Fox(1995)가 온톨로지의 범위를 정하고 평가하는 수단으로 제안했습니다.

3.2 가상 공장 A의 역량 질문

번호	질문	필요한 것
CQ1	펌프 P-101의 부품과 그 부품의 고장 모드는 무엇인가	부품이다, 고장 모드를 가질 수 있다
CQ2	진동이 기준을 넘은 측정은 어느 설비의 것인가	측정, 측정 대상, 진동 값, 시각
CQ3	공장의 설비는 모두 몇 대인가(펌프와 모터 포함)	설비 계층
CQ4	대상 설비가 적히지 않은 작업 지시는 무엇인가	작업 지시, 대상 설비

이 네 질문이 2장의 클래스와 관계를 정했습니다. 질문에 필요 없는 개념(예: 구매 단가, 제조사 주소)은 넣지 않았습니다.

3.3 질문을 요구 요소로 나누기

질문 하나를 「필요한 클래스와 관계」로 나눠 적으면 빠진 것이 보입니다. CQ2를 나누면 측정(클래스), 측정 대상(관계), 진동 값(값 속성), 측정 시각(값 속성) 네 요소가 나옵니다. 넷 중 하나라도 없으면 답할 수 없습니다.

나는 요소마다 「스키마에 정의가 있는가」와 「실제 데이터에 값이 있는가」를 따로 표시합니다. 정의가 있어도 데이터에 값이 없으면 질문에 답하지 못합니다. 이 차이는 생각보다 자주 생깁니다.

3.4 만드는 순서: 일곱 단계

Noy와 McGuinness(2001)의 「Ontology Development 101」은 처음 온톨로지를 만드는 사람을 위한 일곱 단계를 제시합니다. 가상 공장 A에 적용하면 다음과 같습니다.

단계	할 일	가상 공장 A에서
1 범위	분야와 쓰임을 정하고 역량 질문을 적습니다	정비 담당자의 질문 CQ1~CQ4
2 재사용 검토	이미 있는 온톨로지와 표준을 찾습니다	측정은 SOSA, 분류는 SKOS를 검토(8장)
3 용어 열거	필요한 낱말을 모두 적습니다	설비, 펌프, 모터, 베어링, 고장, 작업 지시, 진동, 태그
4 클래스와 계층	클래스를 정하고 위아래를 잇습니다	펌프·모터는 설비의 하위 클래스
5 속성	관계와 값 속성을 정합니다	부품이다, 구동한다, 진동 속도
6 제약	개수, 자료형, 정의역·치역을 정합니다	작업 지시의 대상 설비는 하나 이상
7 개체	실제 개체를 넣습니다	P-101, BRG-101, WO-001

단계는 한 번에 끝나지 않습니다. 개체를 넣다 보면(7단계) 빠진 속성이 보이고(5단계), 다시 역량 질문으로 돌아가 범위를 고칩니다.

확인 문제

문제 1. CQ2를 요구 요소로 나누면 무엇이 나오니까?

답. 측정(클래스), 측정 대상(관계), 진동 값(값 속성), 측정 시각(값 속성)입니다.

문제 2. 「스키마에 정의가 있다」와 「데이터에 값이 있다」를 따로 세는 이유는 무엇입니까?

답. 정의가 있어도 실제 데이터에 값이 없으면 질문에 답하지 못하기 때문입니다.

이 장이 닿지 않는 곳: 역량 질문은 범위를 정하는 도구이지 정확성을 보장하지 않습니다. 질문이 스키마로 표현된다는 것과, 실제로 질의를 돌려 맞는 답을 받는다는 것은 다릅니다. 6장에서 질문을 실제 질의로 돌려 확인합니다.

4. RDF와 Turtle로 적기

4.1 모든 사실은 세 칸짜리 문장입니다

RDF — Resource Description Framework, 자원 기술 프레임워크: 사실을 주어·술어·목적어 세 칸의 문장(트리플)으로 적는 W3C 데이터 모델입니다.

「베어링 BRG-101은 펌프 P-101의 부품이다」는 다음 트리플 하나입니다.

주어	술어	목적어
BRG-101	partOf	P-101

IRI — 국제화 자원 식별자: 주어, 술어, 목적어에 붙이는 전역 이름입니다. `https://example.org/plant/P-101`처럼 씁니다. 다른 시스템과 섞여도 이름이 겹치지 않게 하려는 것입니다.

4.2 Turtle로 적기

Turtle — 터틀: RDF를 사람이 읽기 쉽게 적는 문법입니다. 접두어(@prefix)로 긴 IRI를 줄여 씁니다. 예제 `plant.ttl`의 일부입니다.

```
@prefix ex:    <https://example.org/plant/> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .

ex:P-101    a ex:Pump      ; ex:tag "P-101" ; rdfs:label "냉각수 공급 펌프"@ko .
ex:M-101    a ex:Motor     ; ex:tag "M-101" ; ex:drives ex:P-101 .
ex:BRG-101  a ex:Bearing   ; ex:partOf ex:P-101 ; ex:hasFailureMode ex:FM-wear .

ex:MS-2 a ex:Measurement ; ex:measuredOn ex:P-101 ;
  ex:velocityRms 7.4 ;
  ex:measuredAt "2026-09-15T09:00:00"^^xsd:dateTime .
```

읽는 법은 이렇습니다.

- `a`는 영어 관사가 아니라 Turtle의 약속입니다. 「~의 한 종류다」를 뜻하는 `rdfs:type`을 줄여 쓴 것입니다. 그래서 `ex:P-101 a ex:Pump`는 원래 다음과 같은 문장이고, 「P-101은 펌프다」로 읽습니다.

```
ex:P-101 rdfs:type ex:Pump .
```

칸	값	뜻
주어	<code>ex:P-101</code>	무엇에 대한 이야기인가: 펌프 P-101

칸	값	뜻
술어	<code>a(= rdf:type)</code>	둘의 관계: 「~의 한 종류다」
목적어	<code>ex:Pump</code>	무엇과 이어지는가: 펌프라는 클래스

- `;` 는 같은 주어로 다음 술어를 이어 씁니다.
- "냉각수 공급 펌프"@ko 는 한국어 글자, 7.4 는 숫자, `^^xsd:dateTime` 은 날짜·시각 형식을 뜻합니다.

예제 파일 전체는 88개 트리플입니다.

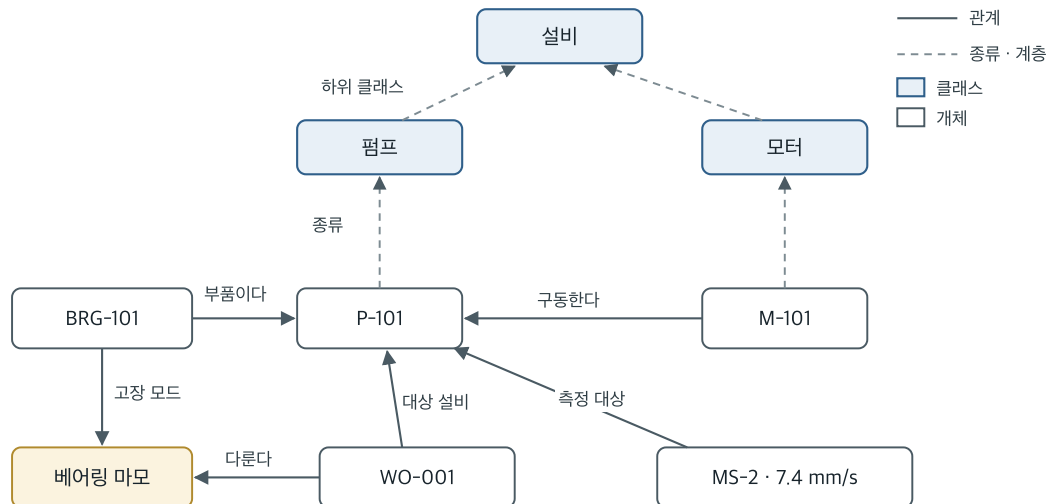


그림 1. 가상 공장 A 예제 그래프의 일부. 파란 상자는 클래스, 흰 상자는 개체입니다. 점선은 종류와 계층(`rdf:type`, `rdfs:subClassOf`), 실선은 관계입니다.

4.3 이름 짓기 규칙

- 클래스는 대문자로 시작하는 단수 명사(Pump), 관계와 속성은 소문자로 시작하는 동사구(partOf, measuredOn)로 씁니다.
- 사람이 읽는 이름은 IRI가 아니라 `rdfs:label`에 언어 태그와 함께 둡니다. IRI는 바뀌지 않는 식별자로 둡니다.
- 현장 태그("P-101")는 IRI에 넣더라도 따로 값 속성(`ex:tag`)으로도 둡니다. 예제의 `ex:P-101 a ex:Pump ; ex:tag "P-101"` . 에서 앞의 `ex:P-101` 은 IRI이고 뒤의 `"P-101"` 은 값 속성입니다.

IRI는 주민등록번호처럼 다른 데이터와 잇는 열쇠라서 한번 정하면 바꾸지 않습니다. 태그는 명찰처럼 현장 사람이 부르는 이름이라 바뀔 수 있습니다. 공장이 태그 체계를 바꿔 P-101이 「CW-P-101」이 되었다고 해 봅시다. 태그를 IRI 안에만 두었다면 IRI를 바꿔야 하고, 그러면 이 IRI로 이어진 정비 기록과 측정값의 연결이 끊깁니다. 태그를 값 속성으로 따로 두었다면 IRI는 그대로 두고 `ex:tag` 값만 바꾸면 됩니다. 연결은 그대로이고 새 태그로도 찾을 수 있습니다.

확인 문제

문제 1. `ex:P-101 a ex:Pump`에서 주어, 술어, 목적어는 각각 무엇입니까?

답. 주어는 `ex:P-101` (펌프 P-101), 술어는 `a`, 목적어는 `ex:Pump` (펌프 클래스)입니다. `a`는 `rdf:type` (「~의 한 종류다」)을 줄인 말이므로 이 줄은 「P-101은 펌프다」라는 문장입니다.

문제 2. 현장 태그를 IRI와 별도의 값 속성으로도 두는 이유는 무엇입니까?

답. IRI는 다른 데이터와 잇는 바뀌지 않는 열쇠이고, 태그는 바뀔 수 있는 이름이기 때문입니다. 태그 체계가 바뀌어도(예: P-101 → CW-P-101) IRI를 그대로 두어 연결을 지키고, 태그 값만 고쳐 새 이름으로도 찾을 수 있게 합니다.

이 장이 닿지 않는 곳: RDF는 Turtle 외에도 JSON-LD, N-Triples, RDF/XML 같은 직렬화가 있습니다. 같은 그래프를 다르게 적는 방법일 뿐이며 이 안내서는 Turtle만 씁니다.

5. RDFS와 OWL: 계층과 추론

5.1 RDFS로 계층을 적습니다

RDFS — RDF Schema, RDF 스키마: 클래스, 하위 클래스, 속성의 정의역과 치역을 적는 기본 어휘입니다.

```
ex:Pump a owl:Class ; rdfs:subClassOf ex:Equipment .
ex:Motor a owl:Class ; rdfs:subClassOf ex:Equipment .
ex:partOf rdfs:domain ex:Component ; rdfs:range ex:Equipment .
```

5.2 추론을 실제로 보기

데이터에는 P-101이 「펌프」라고만 적혀 있고 「설비」라고는 적혀 있지 않습니다. 그래서 「설비를 모두 보여 달라」(CQ3)를 그냥 물으면 빈 결과가 나옵니다. RDFS 추론을 한 뒤 다시 물으면 계층을 따라 답이 나옵니다. `run.py`의 실제 출력입니다.

```
== Q3 설비(Equipment) 목록(추론 전)

== Q3 설비(Equipment) 목록(RDFS 추론 후)
ex:M-101
ex:P-101
ex:P-102
```

Reasoner — 추론기: 적힌 공리에서 적히지 않은 사실을 이끌어 내는 프로그램입니다. 이 예제는 파이썬 `owlrl`의 RDFS 추론을 썼습니다.

5.3 정의역은 검사가 아니라 추론입니다

입문자가 가장 자주 헷갈리는 지점입니다. `ex:partOf`의 정의역(domain)을 「부품」으로 적었습니다. 많은 사람이 이것을 「partOf의 주어는 부품이어야 한다」는 검사로 읽습니다. 그런데 RDFS에서는 반대입니다. 「partOf의 주어로 쓰였으니 그것은 부품이다」라고 추론합니다.

작업 지시 WO-001을 실수로 `ex:partOf ex:P-101`로 이은 뒤 추론하면, 추론기는 오류를 내지 않고 WO-001을 「부품」으로도 분류합니다. `run.py`의 출력입니다.

```
== Q5 잘못된 partOf 뒤 WO-001의 클래스(RDFS 추론 후, ex: 만)
ex:Component
ex:WorkOrder
```

Domain / Range — 정의역 / 치역: 관계의 주어와 목적어가 속하는 클래스입니다. 이것을 검사로 쓰고 싶다면 7장의 SHACL을 씁니다.

5.4 OWL은 더 많은 것을 말합니다

OWL — Web Ontology Language, 웹 온톨로지 언어: RDFS보다 표현력이 큰 W3C 온톨로지 언어입니다(OWL 2, 2012). 대표적인 표현은 다음과 같습니다.

표현	뜻	예
<code>owl:disjointWith</code>	두 클래스는 겹치지 않는다	펌프이면서 모터일 수는 없다
<code>owl:inverseOf</code>	역관계	「부품이다」의 역은 「부품을 가진다」
<code>owl:TransitiveProperty</code>	전이 관계	부품의 부품도 부품이다
<code>owl:someValuesFrom</code>	존재 제약	작업 지시는 적어도 하나의 대상 설비를 가진다
<code>owl:equivalentClass</code>	조건으로 정의한 클래스	「진동 경보 설비」 = 기준을 넘은 측정이 있는 설비

OWL은 열린 세계 가정 — Open World Assumption 을 따릅니다. 「작업 지시는 대상 설비를 적어도 하나 가진다」고 적어 두어도, 대상 설비가 안 적힌 작업 지시를 오류로 보지 않습니다. 「아직 모르는 대상 설비가 어딘가 있다」고 해석하기 때문입니다. 이 차이 때문에 「데이터에 빠진 값을 찾는 일」은 OWL이 아니라 SHACL이 말합니다.

확인 문제

문제 1. 추론 전에는 「설비」 목록이 비었다가 추론 뒤 세 개가 나온 이유는 무엇입니까?

답. 데이터에는 「펌프」, 「모터」로만 적혀 있고, 「펌프는 설비의 하위 클래스」라는 공리로 추론기가 「설비」를 이끌어 냈기 때문입니다.

문제 2. 정의역을 「부품」으로 적은 관계에 작업 지시를 주어로 쓰면 RDFS 추론기는 어떻게 합니까?

답. 오류를 내지 않고 그 작업 지시를 「부품」으로도 분류합니다. 정의역은 검사가 아니라 추론입니다.

이 장이 닿지 않는 곳: OWL의 기술 논리, 프로파일(EL, QL, RL), 추론의 계산 복잡도는 다루지 않았습니다. 실무에서는 필요한 추론만 고르는 편이 성능과 예측 가능성 모두에 좋습니다.

6. SPARQL로 묻기

6.1 질의는 빈칸이 있는 트리플입니다

SPARQL — SPARQL 질의 언어: RDF 그래프에 질문을 던지는 W3C 표준 질의 언어입니다(SPARQL 1.1, 2013). 물음표로 시작하는 변수에 맞는 값을 찾습니다.

CQ1 「펌프 P-101의 부품과 그 고장 모드」는 다음과 같습니다.

```
PREFIX ex: <https://example.org/plant/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?part ?fmLabel WHERE {
  ?part ex:partOf ex:P-101 ; ex:hasFailureMode ?fm .
  ?fm rdfs:label ?fmLabel .
}
```

```
== Q1 펌프 P-101 의 부품과 그 고장 모드
ex:BRG-101 | 베어링 마모
```

6.2 걸러 내기와 정렬

CQ2 「진동이 기준을 넘은 측정」입니다. 여기서 4.5mm/s는 이 예제에서 정한 값이며, 실제 기준은 설비 종류와 설치 조건에 따라 따로 정해야 합니다.

```
SELECT ?eq ?v ?t WHERE {
  ?m ex:measuredOn ?eq ; ex:velocityRms ?v ; ex:measuredAt ?t .
  FILTER(isNumeric(?v) && ?v > 4.5)
} ORDER BY ?t
```

```
== Q2 진동이 4.5 mm/s 를 넘은 측정
ex:P-101 | 7.4 | 2026-09-15T09:00:00
```

`isNumeric(?v)`를 넣은 이유가 있습니다. 예제에는 진동값이 「높음」으로 적힌 측정(MS-3)이 있습니다. 숫자가 아닌 값은 비교에서 빠지므로, 이 질의는 MS-3을 조용히 놓칩니다. 질의가 답을 냈다고 데이터가 깨끗하다는 뜻은 아닙니다. 7장에서 이 측정을 찾습니다.

6.3 없는 것을 찾기

CQ4 「대상 설비가 없는 작업 지시」는 `FILTER NOT EXISTS`로 묻습니다.

```
SELECT ?wo ?label WHERE {  
  ?wo a ex:WorkOrder ; rdfs:label ?label .  
  FILTER NOT EXISTS { ?wo ex:onEquipment ?any }  
}
```

== Q4 대상 설비가 없는 작업 지시
ex:W0-002 | 대상 설비를 적지 않은 작업 지시

6.4 역량 질문을 시험으로 만들기

네 개의 역량 질문을 질의로 적어 두고, 데이터가 바뀔 때마다 돌려서 기대한 답이 나오는지 봅니다. 이것이 온톨로지의 회귀 시험입니다. 답이 비거나 달라지면 스키마나 데이터 가운데 어디가 바뀌었는지 찾습니다.

확인 문제

문제 1. CQ2 질의가 MS-3을 결과에 넣지 않은 이유는 무엇이고, 그것이 왜 위험한니까?

답. 값이 숫자가 아니어서 `isNumeric` 조건에서 빠졌습니다. 질의는 오류를 알리지 않고 조용히 결과에서 빼므로 데이터 문제를 놓칩니다.

문제 2. 역량 질문을 질의로 적어 두면 무엇에 쓸 수 있습니까?

답. 데이터와 스키마가 바뀔 때마다 돌리는 회귀 시험입니다.

이 장이 닿지 않는 곳: 여러 그래프를 묶어 묻는 연합 질의(SERVICE), 갱신(INSERT, DELETE), 성능 조정은 다루지 않았습니다.

7. SHACL로 검증하기

7.1 데이터가 약속을 지키는지 검사합니다

SHACL — Shapes Constraint Language, **형태 제약 언어**: RDF 데이터가 정한 모양(필수 값, 자료형, 개수)을 지키는지 검사하는 W3C 표준입니다(2017). OWL과 달리 닫힌 세계처럼 검사합니다. 값이 없으면 위반입니다.

예제의 검증 규칙 `shapes.ttl`입니다.

```
ex:WorkOrderShape a sh:NodeShape ;
  sh:targetClass ex:WorkOrder ;
  sh:property [ sh:path ex:onEquipment ; sh:minCount 1 ;
    sh:message "작업 지시에는 대상 설비가 하나 이상 있어야 합니다" ] .

ex:MeasurementShape a sh:NodeShape ;
  sh:targetClass ex:Measurement ;
  sh:property [ sh:path ex:velocityRms ; sh:datatype xsd:decimal ; sh:minCount 1 ;
    sh:message "진동 값은 숫자(xsd:decimal)여야 합니다" ] .
```

7.2 실행 결과

pySHACL로 검사한 실제 출력입니다. 일부러 넣은 오류 두 개를 모두 찾았습니다.

```
== SHACL 검증: conforms = False
Results (2):
Focus Node: ex:MS-3
Value Node: Literal("높음")
Result Path: ex:velocityRms
Message: 진동 값은 숫자(xsd:decimal)여야 합니다
Focus Node: ex:W0-002
Result Path: ex:onEquipment
Message: 작업 지시에는 대상 설비가 하나 이상 있어야 합니다
```

6장의 질의가 조용히 놓친 MS-3을 SHACL은 위반으로 잡았습니다. 질의는 답을 찾는 도구이고, SHACL은 데이터가 약속을 지켰는지 보는 도구입니다. 둘은 서로를 대신하지 않습니다.

7.3 OWL과 SHACL, 언제 무엇을

하고 싶은 일	쓸 것
적하지 않은 사실을 이끌어 낸다(펌프는 설비다)	RDFS·OWL 추론
빠진 값, 틀린 자료형, 개수 위반을 찾는다	SHACL

하고 싶은 일	쓸 것
두 클래스가 겹치면 모순으로 본다	OWL(owl:disjointWith)
위반 목록을 사람에게 보고서로 준다	SHACL 검증 결과

확인 문제

문제 1. OWL의 존재 제약으로는 「대상 설비 없는 작업 지시」가 오류로 나오지 않는 이유는 무엇입니까?

답. OWL은 열린 세계 가정을 따라, 적히지 않은 대상 설비를 「모르는 것」으로 보기 때문입니다.

문제 2. SHACL 검증과 SPARQL 질의는 각각 무엇을 위한 도구입니까?

답. SPARQL은 답을 찾는 도구, SHACL은 데이터가 정한 약속을 지켰는지 검사하는 도구입니다.

이 장이 닿지 않는 곳: SHACL의 SPARQL 기반 규칙, 규칙 심각도 구분, 대용량 검증 성능은 다루지 않았습니다.
검증 규칙도 데이터처럼 버전을 매겨 관리하는 편이 좋습니다.

8. 남이 만든 것 다시 쓰기

8.1 처음부터 다 만들지 않습니다

설비, 측정, 단위, 시간, 출처 같은 개념은 이미 많은 공개 온톨로지와 표준 어휘가 정의해 두었습니다. 다시 쓰면 다른 시스템과 바로 이어지고, 설계 실수도 줄어듭니다.

어휘	다루는 것	쓸 곳
SKOS	개념 체계와 용어(같은 말, 넓은 말, 좁은 말)	고장 모드 분류, 동의어 사전
SOSA/SSN	센서, 관측, 측정 결과	진동 측정
QUDT	물리량과 단위	mm/s, °C 같은 단위
OWL-Time	시각과 기간	측정 시각, 정비 기간
PROV-O	데이터의 출처와 생성 과정	이 값은 어느 시스템에서 왔는가
BFO, IOF Core	상위 개념과 산업 공통 개념	설비, 기능, 역할의 공통 틀

8.2 이름만 보고 가져오지 않습니다

다시 쓸 때 가장 흔한 실수는 이름이 맞는 개념을 찾으려 바로 가져오는 것입니다. 이름이 같아도 뜻이 다를 수 있습니다. 가져오기 전에 세 가지를 읽습니다.

1. **관계의 방향.** 정의역과 치역, 설명 주석을 읽어 「누가 누구에게 하는 관계인가」를 확인합니다. 이름만 보면 「측정한다」의 주어가 측정 과정 같아도, 정의상 주어는 계측 장치일 수 있습니다.
2. **클래스의 정의 방식.** 어떤 표준은 「설비」를 물건의 종류가 아니라 「설비 역할을 맡은 물건」으로 정의합니다. 이런 클래스 아래에 「펌프」를 종류로 선언하면 원래 정의와 어긋납니다.
3. **데이터에 실제로 값이 있는가.** 표준에 칸이 있어도 실제 데이터 파일에는 그 칸이 비어 있는 경우가 많습니다. 3.3절처럼 「정의가 있다」와 「값이 있다」를 따로 셉니다.

8.3 재사용의 수준

수준	방법	언제
이름만 빌림	그 IRI를 그대로 쓰되 파일은 가져오지 않음	몇 개 개념만 필요할 때
대응만 적음	내 클래스와 표준 클래스를 <code>skos:exactMatch</code> , <code>closeMatch</code> 로 이음	내 설계를 유지하면서 다른 시스템과 이을 때
일부 추출	필요한 모듈만 떼어 가져옴	표준이 너무 클 때

수준	방법	언제
전체 가져옴	<code>owl:imports</code>	표준의 추론까지 필요할 때

8.4 실습: 고장 모드를 SKOS로 분류하기

고장 모드처럼 「넓은 말과 좁은 말», 「다른 이름」이 중요한 목록은 클래스보다 SKOS 개념 체계로 적는 편이 쉽습니다. 예제 `failure-modes.ttl`의 일부입니다.

```
ex:FM-mechanical a skos:Concept ; skos:prefLabel "기계적 고장"@ko .
ex:FM-wear-all a skos:Concept ; skos:prefLabel "마모"@ko ; skos:broader ex:FM-mechanical .
ex:FM-wear a skos:Concept ; skos:prefLabel "베어링 마모"@ko ;
  skos:altLabel "베어링 손상"@ko , "brg wear"@en ; skos:broader ex:FM-wear-all .
ex:FM-seal a skos:Concept ; skos:prefLabel "실 누설"@ko ; skos:altLabel "씰 누수"@ko ;
  skos:broader ex:FM-mechanical .
```

`skos:broader+`는 넓은 말 관계를 여러 단계 따라갑니다. 「기계적 고장」 아래의 모든 고장 모드와, 현장에서 쓰는 다른 이름 「씰 누수」로 찾은 결과입니다.

```
== Q6 「기계적 고장」 아래의 모든 고장 모드(broader+)
마모
베어링 마모
실 누설

== Q7 「씰 누수」라는 다른 이름으로 찾은 개념
ex:FM-seal | 실 누설
```

현장 기록에 「씰 누수」라고 적혀 있어도 정보 이름 「실 누설」로 모입니다. 동의어 사전을 별도 표로 관리하지 않아도 됩니다.

확인 문제

문제 1. 표준 어휘의 관계를 가져오기 전에 확인할 세 가지는 무엇입니까?

답. 관계의 방향(정의역·치역과 설명 주석), 클래스의 정의 방식(종류인지 역할인지), 실제 데이터에 값이 있는지입니다.

문제 2. 「씰 누수」로 기록된 고장을 정보 이름으로 모으려면 SKOS의 무엇을 씁니까?

답. `skos:altLabel`(다른 이름)과 `skos:prefLabel`(정보 이름)입니다.

이 장이 닿지 않는 곳: 상위 온톨로지(BFO 등)를 채택할지는 팀의 숙련도와 목적에 달려 있습니다. 채택하면 다른 표준과의 정렬이 쉬워지지만 학습 비용이 큼니다. 이 안내서의 예제는 상위 온톨로지 없이 썼습니다.

9. 속성 그래프, 그리고 LLM과 함께 쓰기

9.1 RDF와 속성 그래프

현장에서는 RDF 대신 **Property graph** — 속성 그래프(Neo4j 등)를 많이 씁니다. 노드와 관계에 키-값 속성을 바로 붙이는 모델입니다.

질문	RDF	속성 그래프
관계에 값 붙이기	별도 표현이 필요(RDF-star, 관계를 개체로)	관계에 바로 붙임
전역 식별자	IRI가 기본	직접 정해야 함
추론·표준 검증	RDFS·OWL·SHACL 표준이 있음	표준이 약해 코드로 구현
질의 언어	SPARQL	Cypher, GQL(ISO/IEC 39075:2024)

선택은 질문이 합니다. 추론과 표준 어휘 재사용이 중요하다면 RDF, 경로 탐색과 개발 생산성이 중요하다면 속성 그래프가 편합니다. 어느 쪽이든 온톨로지(개념과 관계의 약속)는 필요합니다. 속성 그래프에서는 그 약속을 레이블과 관계 유형의 목록, 그리고 검사 코드로 지킵니다.

같은 질문을 Cypher로

6장의 CQ1을 속성 그래프(Neo4j)에 옮기면 다음과 같습니다. 노드 레이블과 관계 유형은 예제의 클래스와 관계 이름을 그대로 썼습니다. 이 Cypher는 이 안내서에서 실행하지 않았으며 형식을 보이기 위한 것입니다.

```
// CQ1: 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, fm.label;

// CQ4: 대상 설비가 없는 작업 지시
MATCH (wo:WorkOrder)
WHERE NOT (wo)-[:ON_EQUIPMENT]->(:Equipment)
RETURN wo.id, wo.label;
```

속성 그래프에는 추론이 없으므로, CQ3처럼 「설비」로 모두 찾으려면 펌프 노드에 `:Equipment` 레이블도 함께 붙여 두어야 합니다. 온톨로지의 계층을 적재할 때 레이블로 펼쳐 넣는 이유입니다.

9.2 LLM은 온톨로지를 대신하지 않습니다

LLM(대규모 언어 모델)은 문장을 이해하고 만들지만, 회사의 설비가 무엇이고 어떻게 이어져 있는지는 모릅니다. 그래서 둘을 함께 씁니다.

- **RAG — Retrieval-Augmented Generation, 검색 증강 생성:** 질문과 관련된 문서를 찾아 LLM에 함께 주고 답하게 하는 방식입니다(Lewis 외 2020).
- **GraphRAG — 그래프 기반 검색 증강 생성:** 문서 대신 또는 문서와 함께 그래프에서 관련 개체와 관계를 찾아 주는 방식입니다(Edge 외 2024 등).

온톨로지는 여기서 세 가지 일을 합니다.

1. **어휘를 고정합니다.** LLM이 만든 질의나 추출 결과가 온톨로지에 있는 클래스와 관계만 쓰는지 검사합니다.
2. **근거를 남깁니다.** 답에 쓴 사실이 그래프의 어느 트리플에서 왔는지 붙입니다.
3. **답을 멈춥니다.** 질문에 필요한 관계가 그래프에 없거나 검증되지 않았으면, 그럴듯한 답 대신 「답할 수 없음」과 그 이유를 냅니다.

확인 문제

문제 1. 속성 그래프에서 CQ3처럼 상위 클래스로 찾으려면 적재할 때 무엇을 해야 합니까?

답. 펌프 노드에 :Equipment 레이블도 함께 붙이는 식으로 온톨로지의 계층을 레이블로 펼쳐 넣습니다.

문제 2. LLM과 함께 쓸 때 온톨로지가 하는 세 가지 일은 무엇입니까?

답. 어휘를 고정하고, 답의 근거를 남기고, 필요한 관계가 없거나 검증되지 않았으면 답을 멈춥니다.

이 장이 닿지 않는 곳: LLM으로 온톨로지 초안을 만드는 일, 문서에서 개체와 관계를 자동 추출하는 일은 빠르게 발전하고 있습니다. 다만 그 결과도 7장의 검증과 사람의 검토를 거쳐야 합니다. 이 안내서는 특정 도구의 성능을 비교하지 않았습니다.

10. 흔한 실수 열 가지

실수	결과	바로잡는 법
범위 없이 시작합니다	끝나지 않고 쓰이지 않습니다	역량 질문 5~10개로 시작합니다(3장)
종류와 부분을 섞습니다	수를 잘못 셉니다	is-a와 part-of를 다른 관계로 둡니다(2.2절)
사람용 이름을 IRI에 넣고 바꿉니다	링크가 끊깁니다	IRI는 고정, 이름은 <code>rdfs:label</code> 예(4.3절)
정의역을 검사로 믿습니다	잘못된 데이터가 엉뚱한 클래스로 분류됩니다	검사는 SHACL로 합니다(5.3절)
OWL로 빠진 값을 찾으려 합니다	열린 세계라 위반이 나오지 않습니다	SHACL의 minCount를 씁니다(5.4절)
단위를 값에 섞어 적습니다(「7.4mm/s」)	비교와 계산이 안 됩니다	숫자와 단위를 나눠 적습니다(QUDT)
질의 결과가 나오면 끝났다고 봅니다	숫자 아닌 값 같은 오류를 놓칩니다	질의와 검증을 함께 돌립니다(6.2절)
이름만 보고 표준을 가져옵니다	뜻이 어긋난 채 쓰입니다	방향, 정의 방식, 실제 값을 확인합니다(8.2절)
출처를 남기지 않습니다	값이 어디서 왔는지 모릅니다	PROV-O로 출처를 붙입니다
버전을 매기지 않습니다	무엇이 바뀌었는지 모릅니다	온톨로지와 검증 규칙에 버전과 변경 기록을 둡니다

11. 다음 걸음

11.1 직접 해 볼 것

1. 예제 `plant.ttl`에 「모터 M-102가 펌프 P-102를 구동한다」를 더하고 CQ3을 다시 돌려 보십시오.
2. `ex:drives`의 정의역을 모터로 두었습니다. 펌프를 주어로 잘못 쓰면 추론 결과가 어떻게 되는지 5.3절처럼 확인해 보십시오.
3. 「측정 시각은 반드시 있어야 한다」는 SHACL 규칙을 더하고, 시각을 뺀 측정 하나를 넣어 위반이 나오는지 보십시오.
4. 고장 모드를 SKOS 개념 체계로 바꿔 「마모」 아래에 「베어링 마모」를 두어 보십시오.

11.2 쓸 만한 도구

도구	용도
Protégé	온톨로지 편집과 추론기 실행(무료)
rdflib, pySHACL, owlrl	파이썬에서 RDF 읽기, 질의, 검증, 추론
Apache Jena Fuseki	SPARQL 서버
Neo4j	속성 그래프 데이터베이스
ontocraft-check	한국어 보고서를 내는 온톨로지·데이터 검사기(Apache-2.0, PyPI)

11.3 함께 볼 자료

OntoCraft는 한국어 산업 용어 등록부를 CC BY 4.0으로 공개합니다(<https://ontocraft.com/terms>). 설비 보전, 반도체, 해양, 항만, 방산 등의 용어 정의와 동의어를 SKOS 형식으로 내려받을 수 있습니다.

부록 A. 용어

용어	뜻
온톨로지	개념과 관계를 사람과 기계가 같은 뜻으로 읽도록 명시적으로 적은 것
클래스	같은 성질을 가진 것들의 모임
개체(인스턴스)	클래스에 속하는 실제 하나
관계(객체 속성)	개체와 개체를 잇는 것
값 속성(데이터 속성)	개체에 숫자·글자·날짜를 붙이는 것
공리	항상 참이라고 약속한 문장
트리플	주어·술어·목적어 세 칸의 사실
IRI	전역에서 겹치지 않는 이름
정의역 / 치역	관계의 주어와 목적어가 속하는 클래스
추론기	적힌 공리에서 적히지 않은 사실을 이끌어 내는 프로그램
열린 세계 가정	적혀 있지 않은 것을 거짓이 아니라 모르는 것으로 보는 가정
역량 질문(CQ)	온톨로지가 답해야 할 업무 질문
SPARQL	RDF 질의 언어
SHACL	RDF 데이터 검증 언어
SKOS	개념 체계와 용어를 적는 어휘
지식 그래프	온톨로지의 뜻에 맞춰 쌓인 개체와 관계의 데이터
속성 그래프	노드와 관계에 키-값 속성을 바로 붙이는 그래프 모델
GraphRAG	그래프에서 찾은 개체와 관계를 LLM에 함께 주는 검색 증강 생성

부록 B. 참고 문헌

- W3C (2014). RDF 1.1 Concepts and Abstract Syntax. <https://www.w3.org/TR/rdf11-concepts/>
- W3C (2014). RDF 1.1 Turtle. <https://www.w3.org/TR/turtle/>
- W3C (2014). RDF Schema 1.1. <https://www.w3.org/TR/rdf-schema/>
- W3C (2012). OWL 2 Web Ontology Language Primer (Second Edition). <https://www.w3.org/TR/owl2-primer/>
- W3C (2013). SPARQL 1.1 Query Language. <https://www.w3.org/TR/sparql11-query/>
- W3C (2017). Shapes Constraint Language (SHACL). <https://www.w3.org/TR/shacl/>
- W3C (2009). SKOS Simple Knowledge Organization System Reference. <https://www.w3.org/TR/skos-reference/>
- W3C (2017). Semantic Sensor Network Ontology (SOSA/SSN). <https://www.w3.org/TR/vocab-ssn/>
- W3C (2013). PROV-O: The PROV Ontology. <https://www.w3.org/TR/prov-o/>
- Grüninger, M., Fox, M. S. (1995). Methodology for the Design and Evaluation of Ontologies. IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing.
- Noy, N. F., McGuinness, D. L. (2001). Ontology Development 101: A Guide to Creating Your First Ontology. Stanford KSL-01-05.
- Guarino, N., Welty, C. (2002). Evaluating Ontological Decisions with OntoClean. Communications of the ACM 45(2).
- ISO/IEC 21838-2:2021. Top-level ontologies, Part 2: Basic Formal Ontology (BFO).
- Drobnjakovic, M. 외 (2022). The Industrial Ontologies Foundry (IOF) Core Ontology. FOMI 2022, CEUR Vol. 3240.
- 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.
- ISO/IEC 39075:2024. Information technology — Database languages — GQL.

부록 C. 예제 파일 전문

C.1 plant.ttl

가상 공장 A 의 설비 정비 예제 데이터입니다. 모든 이름과 값은 지어낸 것입니다.

@prefix ex: <https://example.org/plant/> .

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

@prefix owl: <http://www.w3.org/2002/07/owl#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

----- 스키마(클래스와 속성) -----

ex:Equipment a owl:Class ; rdfs:label "설비"@ko .

ex:Pump a owl:Class ; rdfs:subClassOf ex:Equipment ; rdfs:label "펌프"@ko .

ex:Motor a owl:Class ; rdfs:subClassOf ex:Equipment ; rdfs:label "모터"@ko .

ex:Component a owl:Class ; rdfs:label "부품"@ko .

ex:Bearing a owl:Class ; rdfs:subClassOf ex:Component ; rdfs:label "베어링"@ko .

ex:FailureMode a owl:Class ; rdfs:label "고장 모드"@ko .

ex:WorkOrder a owl:Class ; rdfs:label "작업 지시"@ko .

ex:Measurement a owl:Class ; rdfs:label "측정"@ko .

ex:partOf a owl:ObjectProperty ; rdfs:domain ex:Component ; rdfs:range ex:Equipment ; rdfs:label "~의 부품이다"@ko .

ex:drives a owl:ObjectProperty ; rdfs:domain ex:Motor ; rdfs:range ex:Pump ; rdfs:label "구동한다"@ko .

ex:hasFailureMode a owl:ObjectProperty ; rdfs:domain ex:Component ; rdfs:range ex:FailureMode ; rdfs:label "고장 모드를 가질 수 있다"@ko .

ex:addresses a owl:ObjectProperty ; rdfs:domain ex:WorkOrder ; rdfs:range ex:FailureMode ; rdfs:label "다룬다"@ko .

ex:onEquipment a owl:ObjectProperty ; rdfs:domain ex:WorkOrder ; rdfs:range ex:Equipment ; rdfs:label "대상 설비"@ko .

ex:measuredOn a owl:ObjectProperty ; rdfs:domain ex:Measurement ; rdfs:range ex:Equipment ; rdfs:label "측정 대상"@ko .

ex:velocityRms a owl:DatatypeProperty ; rdfs:domain ex:Measurement ; rdfs:range xsd:decimal ; rdfs:label "진동 속도 RMS(mm/s)"@ko .

ex:measuredAt a owl:DatatypeProperty ; rdfs:range xsd:dateTime ; rdfs:label "측정 시각"@ko .

ex:tag a owl:DatatypeProperty ; rdfs:label "설비 태그"@ko .

----- 인스턴스 -----

ex:P-101 a ex:Pump ; ex:tag "P-101" ; rdfs:label "냉각수 공급 펌프"@ko .

ex:P-102 a ex:Pump ; ex:tag "P-102" ; rdfs:label "냉각수 예비 펌프"@ko .

ex:M-101 a ex:Motor ; ex:tag "M-101" ; ex:drives ex:P-101 .

ex:BRG-101 a ex:Bearing ; ex:partOf ex:P-101 ; ex:hasFailureMode ex:FM-wear .

ex:BRG-102 a ex:Bearing ; ex:partOf ex:P-102 ; ex:hasFailureMode ex:FM-wear .

ex:FM-wear a ex:FailureMode ; rdfs:label "베어링 마모"@ko .

ex:W0-001 a ex:WorkOrder ; ex:onEquipment ex:P-101 ; ex:addresses ex:FM-wear ; rdfs:label "P-101 베어링 교체"@ko .

ex:W0-002 a ex:WorkOrder ; ex:addresses ex:FM-wear ; rdfs:label "대상 설비를 적지 않은 작업 지시"@ko .

```
ex:MS-1 a ex:Measurement ; ex:measuredOn ex:P-101 ; ex:velocityRms 2.1 ; ex:measuredAt
"2026-09-01T09:00:00"^^xsd:dateTime .
ex:MS-2 a ex:Measurement ; ex:measuredOn ex:P-101 ; ex:velocityRms 7.4 ; ex:measuredAt
"2026-09-15T09:00:00"^^xsd:dateTime .
ex:MS-3 a ex:Measurement ; ex:measuredOn ex:P-102 ; ex:velocityRms "높음" ; ex:measuredAt
"2026-09-15T09:10:00"^^xsd:dateTime .
```

C.2 shapes.ttl

```
# 가상 공장 A 예제의 SHACL 검증 규칙입니다.
@prefix ex: <https://example.org/plant/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:WorkOrderShape a sh:NodeShape ;
  sh:targetClass ex:WorkOrder ;
  sh:property [
    sh:path ex:onEquipment ;
    sh:minCount 1 ;
    sh:message "작업 지시에는 대상 설비가 하나 이상 있어야 합니다" ;
  ] .

ex:MeasurementShape a sh:NodeShape ;
  sh:targetClass ex:Measurement ;
  sh:property [
    sh:path ex:velocityRms ;
    sh:datatype xsd:decimal ;
    sh:minCount 1 ;
    sh:message "진동 값은 숫자(xsd:decimal)여야 합니다" ;
  ] .
```

C.3 failure-modes.ttl

```
# 가상 공장 A 의 고장 모드 분류 체계(SKOS)입니다. 분류와 이름은 예제용으로 지어낸 것입니다.
@prefix ex:    <https://example.org/plant/> .
@prefix skos:  <http://www.w3.org/2004/02/skos/core#> .

ex:FailureModeScheme a skos:ConceptScheme ;
    skos:prefLabel "가상 공장 A 고장 모드 분류"@ko .

ex:FM-mechanical a skos:Concept ; skos:inScheme ex:FailureModeScheme ;
    skos:prefLabel "기계적 고장"@ko ; skos:topConceptOf ex:FailureModeScheme .
ex:FM-wear-all a skos:Concept ; skos:inScheme ex:FailureModeScheme ;
    skos:prefLabel "마모"@ko ; skos:broader ex:FM-mechanical .
ex:FM-wear a skos:Concept ; skos:inScheme ex:FailureModeScheme ;
    skos:prefLabel "베어링 마모"@ko ; skos:altLabel "베어링 손상"@ko , "brg wear"@en ;
    skos:broader ex:FM-wear-all .
ex:FM-seal a skos:Concept ; skos:inScheme ex:FailureModeScheme ;
    skos:prefLabel "실 누설"@ko ; skos:altLabel "씰 누수"@ko ;
    skos:broader ex:FM-mechanical .
```

C.4 run.py

```

"""커뮤니티판 예제 실행: 질의 결과와 검증 결과를 그대로 출력합니다. 책에 실는 출력은 이 스크립트에서 나온 것입니다.
실행: uv run --with rdflib --with pyshacl --with owlrl python run.py
"""

import pathlib
from rdflib import Graph
import owlrl
from pyshacl import validate

HERE = pathlib.Path(__file__).parent
g = Graph().parse(HERE / "plant.ttl")
print("트리플 수:", len(g))

Q = {
    "Q1 펌프 P-101 의 부품과 그 고장 모드": """
PREFIX ex: <https://example.org/plant/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?part ?fmLabel WHERE {
    ?part ex:partOf ex:P-101 ; ex:hasFailureMode ?fm .
    ?fm rdfs:label ?fmLabel .
}""",
    "Q2 진동이 4.5 mm/s 를 넘은 측정": """
PREFIX ex: <https://example.org/plant/>
SELECT ?eq ?v ?t WHERE {
    ?m ex:measuredOn ?eq ; ex:velocityRms ?v ; ex:measuredAt ?t .
    FILTER(isNumeric(?v) && ?v > 4.5)
} ORDER BY ?t""",
    "Q3 설비(Equipment) 목록(추론 전)": """
PREFIX ex: <https://example.org/plant/>
SELECT ?e WHERE { ?e a ex:Equipment } ORDER BY ?e""",
    "Q4 대상 설비가 없는 작업 지시": """
PREFIX ex: <https://example.org/plant/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?wo ?label WHERE {
    ?wo a ex:WorkOrder ; rdfs:label ?label .
    FILTER NOT EXISTS { ?wo ex:onEquipment ?any }
}""",
}

def show(graph, name, q):
    print("\n==", name)
    for row in graph.query(q):
        print(" ", " | ".join(str(x).replace("https://example.org/plant/", "ex:") for x
            in row))

for k, q in Q.items():
    show(g, k, q)

# RDFS 추론 뒤 Q3 다시
inf = Graph()

```

```

for t in g: inf.add(t)
owlrl.DeductiveClosure(owlrl.RDFS_Semantics).expand(inf)
show(inf, "Q3 설비(Equipment) 목록(RDFS 추론 후)", Q["Q3 설비(Equipment) 목록(추론 전)"])

conforms, _, text = validate(g, shacl_graph=Graph().parse(HERE / "shapes.ttl"), inference="none")
print("\n== SHACL 검증: conforms =", conforms)
for line in text.splitlines():
    if line.strip().startswith(("Focus Node", "Message", "Value Node", "Result Path", "Results (")):
        print("  ", line.strip().replace("https://example.org/plant/", "ex:"))

# 정의역(domain)은 검사가 아니라 추론입니다: 작업 지시를 잘못 partOf 로 이으면 추론기는 그것을 「부품」으로 분류합니다.
bad = Graph()
for t in g: bad.add(t)
bad.update("PREFIX ex: <https://example.org/plant/> INSERT DATA { ex:W0-001 ex:partOf ex:P-101 }")
owlrl.DeductiveClosure(owlrl.RDFS_Semantics).expand(bad)
show(bad, "Q5 잘못된 partOf 뒤 W0-001 의 클래스(RDFS 추론 후, ex: 만)",
      "PREFIX ex: <https://example.org/plant/> SELECT ?c WHERE { ex:W0-001 a ?c FILTER(STRSTARTS(STR(?c), STR(ex:))) } ORDER BY ?c")

# SKOS 분류 체계: 「기계적 고장」 아래의 모든 개념을 broader 를 거꾸로 따라가며(전이) 찾고, 다른 이름(altLabel)으로도 찾습니다.
fm = Graph().parse(HERE / "failure-modes.ttl")
show(fm, "Q6 「기계적 고장」 아래의 모든 고장 모드(broader+)", ""
      "PREFIX ex: <https://example.org/plant/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?label WHERE { ?c skos:broader+ ex:FM-mechanical ; skos:prefLabel ?label } ORDER BY ?label""")

show(fm, "Q7 「씰 누수」라는 다른 이름으로 찾은 개념", ""
      "PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?c ?pref WHERE { ?c skos:altLabel "씰 누수"@ko ; skos:prefLabel ?pref }""")

```

C.5 run.py 실행 결과

트리플 수: 88

== Q1 펌프 P-101 의 부품과 그 고장 모드

ex:BRG-101 | 베어링 마모

== Q2 진동이 4.5 mm/s 를 넘은 측정

ex:P-101 | 7.4 | 2026-09-15T09:00:00

== Q3 설비(Equipment) 목록(추론 전)

== Q4 대상 설비가 없는 작업 지시

ex:W0-002 | 대상 설비를 적지 않은 작업 지시

== Q3 설비(Equipment) 목록(RDFS 추론 후)

ex:M-101

ex:P-101

ex:P-102

== SHACL 검증: conforms = False

Results (2):

Focus Node: ex:MS-3

Value Node: Literal("높음")

Result Path: ex:velocityRms

Message: 진동 값은 숫자(xsd:decimal)여야 합니다

Focus Node: ex:W0-002

Result Path: ex:onEquipment

Message: 작업 지시에는 대상 설비가 하나 이상 있어야 합니다

== Q5 잘못된 partOf 뒤 W0-001 의 클래스(RDFS 추론 후, ex: 만)

ex:Component

ex:WorkOrder

== Q6 「기계적 고장」 아래의 모든 고장 모드(broader+)

마모

베어링 마모

실 누설

== Q7 「씰 누수」라는 다른 이름으로 찾은 개념

ex:FM-seal | 실 누설