# 데이터 표준화 라이브러리 사용자 매뉴얼

KANDLE Data Standardization Methodology

기준과 방법, 그리고 자동화를 통한 **효율성, 일관성, 품질 향상** 제시

**표준 정의**
업무 용어, 정의, 규칙 등 표준 기준 수립

**논리 모델링**
업무 관정의 개념 모델 설계 및 구조 정의

**물리모델링**
DB, 스키마, 테이블 등 물리 구조 설계

**표준화 진단**
데이터 품질 및 표준 적합성 진단 및 분석

**표준화 운영**
표준 적용, 모니터링, 지속적 개선 및 관리



# 차 례


들어가며 . . . . . 9

I 1부 · 데이터 표준화

1 표준화 개요 . . . . . 19
1.1 네 개의 사전이 담당하는 역할 . . . . . 20
1.2 이름이 결정되는 과정 . . . . . 20
1.3 사전 사이의 참조 관계 . . . . . 21
1.4 구성단어는 외래키로 보장되지 않는다 . . . . . 22
1.5 작업 순서 . . . . . 22
1.6 이 부의 구성 . . . . . 25

2 표준단어 . . . . . 27
2.1 사전의 구성 . . . . . 27
2.2 이름과 금지어 . . . . . 29
2.3 분류어와 상태 . . . . . 31
2.4 표준단어 검증 규칙 . . . . . 32

3 표준용어 . . . . . 35
3.1 구조와 구성 . . . . . 35
3.2 용어의 쓰임 . . . . . 36
3.3 이름이 만들어지는 방식 . . . . . 37
3.4 표준용어 검증 규칙 . . . . . 39

4 표준도메인 . . . . . 41
4.1 체계와 이름 . . . . . 41
4.2 길이 정하기 . . . . . 43
4.3 사전 사이의 관계 . . . . . 43
4.4 검증 규칙 . . . . . 44

5 표준코드 . . . . . 47
5.1 무엇을 코드로 만드는가 . . . . . 47
5.2 코드값 다루기 . . . . . 48
5.3 표준코드 검증 규칙 . . . . . 49

II 2부 · 데이터 모델링

6 데이터 모델링 개요 . . . . . 53
6.1 두 단계로 나누는 이유 . . . . . 53
6.2 두 모델의 대응 관계 . . . . . 54
6.3 사전이 모델링에서 하는 일 . . . . . 54
6.4 작업 순서 . . . . . 55
6.5 이 부의 구성 . . . . . 57

7 논리 데이터 모델링 . . . . . 59
7.1 이름은 표준용어에서 선택한다 . . . . . 59
7.2 엔터티 정의서 . . . . . 60
7.3 속성 정의서 . . . . . 61
7.4 관계 작성 . . . . . 62
7.5 검증 . . . . . 62
7.6 다음 단계 . . . . . 63

8 물리 데이터 모델링 . . . . . 65
8.1 논리에서 물리로 . . . . . 65
8.2 엔터티와 테이블은 1:1이 아니다 . . . . . 66
8.3 물리 단계에서 추가되는 항목 . . . . . 66
8.4 타입과 길이는 도메인이 정한다 . . . . . 67
8.5 외래키 작성 . . . . . 67
8.6 검증과 대조 . . . . . 68
8.7 다음 단계 . . . . . 69

9 산출물 작성 . . . . . 71
9.1 DM-LIB과 DM-DST . . . . . 71

9.2 파일명 규칙 . . . 73
9.3 양식 . . . 73
9.4 같은 사실이 여러 곳에 기록된다 . . . 75
9.5 산출물을 개정할 때 . . . 77

10 기본키와 외래키 . . . 79
10.1 두 축으로 나누어 기재한다 . . . 79
10.2 외래키 기록 . . . 81
10.3 DDL 생성 순서 . . . 82

III 3부 · 라이브러리

11 kdsm 라이브러리 . . . 85
11.1 무엇을 위한 라이브러리인가 . . . 86
11.2 라이브러리와 커맨드라인 . . . 87
11.3 설치와 구성 . . . 88
11.4 설계 메모 . . . 92
11.5 모듈이 제공하는 것 . . . 94

12 표준용어 추천 . . . 97
12.1 사전을 어디서 읽는가 . . . 97
12.2 분해 . . . 98
12.3 금지어 . . . 100
12.4 추천기 . . . 102
12.5 쓰는 법 . . . 107

13 검증 . . . 109
13.1 설계 원칙 . . . 111
13.2 사전의 출처 . . . 112
13.3 무엇을 검사하는가 . . . 113
13.4 결과 보기 . . . 121

14 DDL 생성과 대조 . . . 125
14.1 두 가지 방향 . . . 125
14.2 산출물 형식은 시트 구성으로 판별한다 . . . 126
14.3 파일 하나가 SQL 하나가 된다 . . . 126
14.4 출력 경로의 규칙 . . . 127
14.5 커맨드라인 옵션 . . . 127
14.6 라이브러리 인자 . . . 129

14.7 생성기의 판단 . . . 129
14.8 생성하기 전에 검증한다 . . . 133
14.9 정의서와 데이터베이스 대조 . . . 133

15 메타데이터 적재 . . . 137
15.1 메타데이터 파일의 적재 . . . 137
15.2 사용 방법 . . . 138
15.3 내부 동작 . . . 139
15.4 그 밖에 . . . 142
15.5 산출물 파일의 적재 . . . 144
15.6 일괄 적재하기 . . . 145

16 ERD 자동 작성 . . . 149
16.1 세 가지 원천 . . . 149
16.2 작성 대상과 출력 형식 . . . 153
16.3 영역 설정 . . . 153
16.4 전체 옵션 . . . 156
16.5 ERD 파일 출력 형식 . . . 157
16.6 Mermaid 소스 파일 . . . 158
16.7 렌더링 이미지 파일 . . . 163

17 산출물 내보내기 . . . 169
17.1 메타테이블에 없는 두 가지 . . . 170
17.2 표기를 옮기는 두 가지 . . . 172
17.3 데이터 표준 정의 산출물 . . . 173
17.4 논리 데이터 모델 산출물 . . . 173
17.5 물리 데이터 모델 산출물 . . . 174
17.6 카탈로그에서 내보내기 . . . 174
17.7 옵션 . . . 175
17.8 내보낸 결과를 확인하는 방법 . . . 176

18 표준화 진단 . . . 179
18.1 무엇을 측정하는가 . . . 179
18.2 리포트 . . . 182
18.3 사전이 없는 스키마 . . . 187
18.4 이름 통계 . . . 192
18.5 사용 방법 . . . 194

19 커맨드라인 . . . . . 207
19.1 공통 규약 . . . . . 207
19.2 환경변수 . . . . . 208
19.3 명령 . . . . . 209
19.4 CI 연계 . . . . . 216

IV 4부 · 실무

20 환경 준비 . . . . . 221
20.1 처음 한 번의 순서 . . . . . 221
20.2 작업별 필요 도구 . . . . . 223
20.3 운영체제별 차이 . . . . . 224
20.4 1. uv와 kdsm 설치 (필수) . . . . . 225
20.5 2. PostgreSQL 클라이언트 (적재할 때) . . . . . 229
20.6 3. Node.js와 mermaid-cli (ERD를 그림으로) . . . . . 230
20.7 4. Quarto (매뉴얼을 빌드할 때) . . . . . 232
20.8 5. 선택: LLM과 Databricks . . . . . 232
20.9 6. 환경변수 . . . . . 233
20.10 7. 비밀을 노출하지 않고 지정한다 . . . . . 236
20.11 8. KDSM 시스템 환경 준비 . . . . . 239
20.12 처음부터 끝까지 . . . . . 243
20.13 흔한 실패 . . . . . 245

21 자주 발생하는 문제와 예방 . . . . . 247
21.1 원인별 유형 . . . . . 247
21.2 빠른 진단표 . . . . . 253

22 작업 흐름 . . . . . 255
22.1 처음부터 끝까지 . . . . . 256
22.2 자주 하는 일 . . . . . 258
22.3 점검 . . . . . 260
22.4 환경 구축과 남은 일 . . . . . 261

참고 자료 . . . . . 267

용어 사전 . . . . . 275

# 들어가며

이 매뉴얼은 KANDLE 데이터 표준화 방법론(KDSM)에 기반하여 수행하는 데이터 표준화 작업과 데이터 모델링 작업을 다룬다. 두 작업의 기준과 방법을 제시하고, `kdsm` 라이브러리로 산출물의 명세와 기준에 부합하는 산출물을 생성하거나 그 과정을 자동화하는 방법을 정리하였다.

## 자동화가 얻게 하는 것

자동화의 목적은 손이 덜 가게 하는 데에만 있지 않다. 세 가지를 함께 얻는다.

**효율성**
: 반복되는 작업을 사람이 수행하지 않는다. 표준단어의 영문약어를 정해 두면 물리명이 결정되고, 표준도메인을 정해 두면 데이터 타입과 길이가 결정된다. 정의서에서 DDL과 ERD를 생성하고, 데이터베이스에서 정의서를 다시 생성한다.

**일관성**
: 같은 사실이 여러 곳에 기록되면 어긋난다. 사람이 옮겨 기록하는 지점마다 불일치가 발생할 수 있으므로, 옮겨 기록할 수밖에 없는 지점에는 검증 규칙을 둔다. 검증 규칙이 산출물과 사전, 논리 모델과 물리 모델, 정의서와 운영 데이터베이스를 서로 대조한다.

**품질**
: 어긋난 부분을 사람이 발견하기 전에 프로그램이 알린다. 검증은 산출물을 작성하는 시점에, 대조는 데이터베이스에 반영한 뒤에, 진단은 이미 운영 중인 데이터베이스를 대상으로 수행한다. 세 시점이 서로 다른 것을 본다.

## 기준만으로는 판단이 서지 않는다

방법론 문서가 “무엇을 해야 하는가”를 다룬다면, 이 매뉴얼은 “그래서 어떻게 판단하는가”까지 다룬다. 표준화는 규칙을 정하는 일보다 경계에 있는 대상을 어느 쪽으로 분류할지 결정하는 일이 훨씬 많다. 보험사를 도메인어로 볼 것인가 일반어로 볼 것인가, `Y`/`N` 두 값만 가지는 항목도 표준코드로 정의할 것인가, 엔터티와 테이블이 1:1이 아닐 때 외래키를 설정할 것인가와 같은 질문이다. 이러한 판단 기준을 문서로 공유하지 않으면 담당자가 바뀔 때마다 같은 문제를 처음부터 다시 검토하게 된다.

자동화도 이 판단 위에서 성립한다. 프로그램이 대신할 수 있는 것은 정해진 기준을 적용하는 일이지 기준을 정하는 일이 아니다. 그러므로 이 매뉴얼은 각 규칙을 그렇게 정한 이유를 함께 기술한다.

## 이 문서를 읽는 사람

표 0.1. 독자별 읽을 곳

| 대상 | 읽을 곳 |
|---|---|
| 처음 환경을 구축하는 사람 | 4부 환경 준비(20장)부터 |
| 표준단어·용어를 등재하는 사람 | 1부 |
| 데이터 모델링 산출물을 작성하는 사람 | 2부 |
| `kdsm` 라이브러리로 자동화하는 사람 | 3부 |

처음 시작하는 경우에는 무엇을 설치해야 하는지부터 확인한다. 필수 도구는 uv 하나이며, 나머지는 수행하려는 작업에 따라 추가된다. 파일만 읽는 작업은 uv만으로 수행할 수 있다(20장).

## 구성

**1부 데이터 표준화**는 표준단어·표준용어·표준도메인·표준코드 네 사전의 규칙과, 각 규칙을 그렇게 정한 이유를 다룬다. 첫 장에서 관계도로 네 사전이 어떻게 연결되어 있는지 제시하고, 컬럼 이름 하나를 정하는 작업 순서도 함께 수록하였다(1장).

**2부 데이터 모델링**도 첫 장에서 두 모델의 대응 관계와 사전이 모델링에서 하는 일을 제시하고, 모델 하나를 완성하는 작업 순서를 순서도로 수록하였다(6장). 이어서 논리 모델링(7장)과 물리 모델링(8장)의 순서와 판단을 다루고, DM-DST-001부터 003까지 3종 산출물의 작성법을 다룬다. 시스템 관리자가 다루는 DM-LIB 6종과의 관계도 함께 다룬다(9.1절). 특히 기본키와 외래키를 어떻게 적어야 식별 관계까지 표현되는지가 핵심이다.

**3부 라이브러리**는 `kdsm` 패키지로 추천·검증·DDL 생성·적재·진단을 자동화하는 방법이다. 장의 차례는 22.1의 작업 순서를 따른다(22장). 사전은 메타파일과 DB 테이블 양쪽에서 읽을 수 있고, 사전에 없는 조각은 LLM으로부터 등재 초안을 받을 수 있다. LLM은 Claude API에 직접 연결하거나 AWS Bedrock을 경유한다(12.4.3.1절).

메타테이블이 기준이 된 뒤에는 방향이 바뀐다. 문서를 수작업으로 맞추는 대신 데이터베이스에서 DM-DST 산출물 셋을 다시 생성한다(17장). 생성한 문서는 버전 `0.0.0`, 작성자 공란으로 기록된다. 아직 사람이 검토하지 않았다는 표시이다.

같은 메타테이블에서 ERD도 생성한다(16장). 간선을 외래키가 아니라 속성의 참조 항목에서 도출하는데, 외래키로는 설정할 수 없는 관계가 있기 때문이다. DBMS에 테이블을 만들기 전이라면 물리 모델링 산출물을 그대로 읽어 그린다(16.1절).

**4부 실무**는 환경 준비, 자주 발생하는 문제와 예방 방법, 그리고 처음부터 끝까지의 작업 흐름이다.

이미 구축된 데이터베이스가 표준에서 얼마나 벗어나 있는지 측정하는 진단은 3부에 있다(18장). PostgreSQL·MySQL·Databricks Unity Catalog를 대상으로 한다. 표준 사전이 없는 스키마도 진단한다. 이때는 스키마 내부에서 서로 일치하지 않는 부분을 찾는다.

본문 뒤에는 참고 자료와 용어 사전을 두었다. 뜻을 모르는 낱말을 만나면 용어 사전에서 찾는다. 이 매뉴얼이 특정한 뜻으로 사용하는 낱말을 모아 두었다.

## 구성 개요

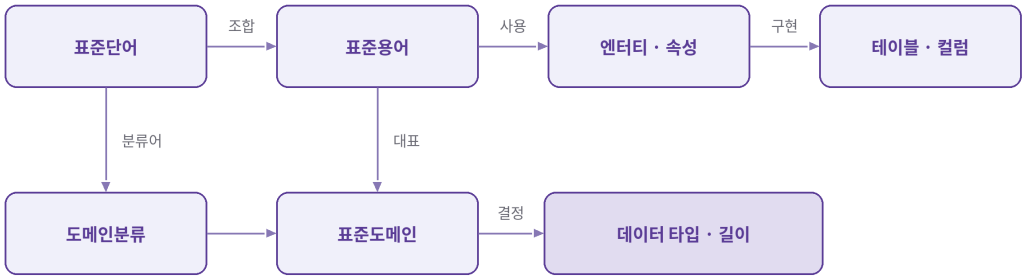


**그림 0.1.** 표준화의 골격

표준화의 골격은 이 흐름이다. 표준단어를 조합하여 표준용어를 만들고, 표준용어로 엔터티와 속성의 이름을 정하며, 이를 테이블과 컬럼으로 구현한다. 데이터 타입과 길이는 설계자가 개별적으로 정하지 않고 표준도메인이 결정한다.

핵심은 **한 번 정한 것을 여러 곳에 반복하여 기록하지 않는 것**이다. 앞의 세 가지가 모두 이 원칙에서 나온다. 앞 단계에서 정한 것이 뒤 단계를 결정하므로 사람이 다시 정할 일이 없고, 다시 정하지 않으므로 어긋날 일도 없다. 그럼에도 옮겨 기록할 수밖에 없는 지점이 남으며, 그 자리에 검증 규칙을 둔다.

## 현재 상태

ℹ 기준일 2026-08-21

**표 0.2.** 2026-09-29 현재 규모

| 항목 | 규모 |
|---|---|
| 표준단어 | 4,193건 (확정 4,143 · 폐기 50) |
| 표준용어 | 102건 |
| 표준도메인 | 139건 (8그룹 · 53분류) |
| 표준코드 | 6종 · 코드값 26건 |
| 물리 테이블 | 14개 ( `kdsm_meta` 스키마) · 메타파일 3벌 |
| 검증 규칙 | 104종 |
| 대상 DBMS | PostgreSQL · MySQL · Databricks (생성 · 적재 · 조회 · 대조 · 진단) |

이 사전은 KDSM 시스템 관리자가 구축한 것이며, 사용자는 여기에 자신의 표준을 등재하여 사용한다. 위 규모는 매뉴얼의 예시가 어떤 데이터를 바탕으로 하는지를 밝히기 위한 것이다.

폐기 표준단어 50건에는 모두 대체어가 연결되어 있다. 따라서 사용할 수 없는 단어를 입력하면 대신 사용할 단어를 함께 제시받는다(2.2.2.2절).

## 문장 작성 기준

이 매뉴얼의 모든 문장은 다음을 지킨다.

1. 문법에 맞는 올바른 문장을 사용한다. 비문을 쓰지 않는다.
2. 비어·속어·은어를 쓰지 않는다.
3. 도서 출판에 적절한 문어체를 사용한다. 구어체 서술과 은유적 표현을 피한다.
4. 평어체로 기술한다. 서술은 `-이다`·`-한다`로 끝맺고, `-입니다`·`-합니다`와 같은 경어체를 쓰지 않는다.

네 번째 기준은 문체의 성격에서 나온 것이다. 경어체는 듣는 사람을 높이는 격식체로 공식 발표나 고객 응대에 쓰고, 평어체는 높임의 구분 없이 사실을 적는 기본형으로 책과 논문, 기사에 쓴다. 이 매뉴얼은 객관적 사실을 기록하는 문서이므로 평어체를 따른다.

지시와 권고도 평어체 서술로 적는다. "지우지 마세요"는 "지우지 않는다"로, "참고하세요"는 "참고한다"로 기술한다.

두 번째와 세 번째 기준은 비유적 구어 표현에도 적용된다. 뜻이 통하더라도 문어체 서술로 바꾸어 기술한다.

**표 0.3.** 문장 표현 기준

| 쓰지 않는 표현 | 바꿔 쓰는 표현 |
|---|---|
| 논리 모델을 DBMS에 앉힌다 | 논리 모델을 DBMS에 적용한다 |
| 사전 넷으로 이루어진다 | 네 개의 사전으로 구성된다 |
| 규칙에 걸린다 | 규칙 위반으로 보고된다 |
| 도면을 뽑는다 | ERD를 생성한다 |
| 값이 갈린다 | 값이 서로 다르다 |
| 낱말로 쪼갠다 | 낱말 단위로 분리한다 |
| 눈으로 훑는다 | 육안으로 확인한다 |
| 조용히 사라진다 | 안내 없이 누락된다 |
| 판이 어긋나면 탈이 난다 | 버전이 일치하지 않으면 문제가 발생한다 |
| 이름 칸이 비어 있다 | 이름 항목이 비어 있다 |
| 기본키가 부딪친다 | 기본키가 중복된다 |
| 재는 자 | 비교 기준 |
| 메타 테이블에 넣는다 | 메타 테이블에 적재한다 |
| 규칙을 돌린다 | 규칙을 실행한다 |
| 지적이 쏟아진다 | 위반이 다수 보고된다 |
| 진짜 문제가 묻힌다 | 실제 문제를 식별하기 어렵다 |
| 적재에서 터진다 | 적재 시점에 제약 위반이 발생한다 |
| 사전에 얹는다 | 사전에 결합한다 |
| 세 벌·일곱 벌 | 3종·7종 |

| 쓰지 않는 표현 | 바꿔 쓰는 표현 |
|---|---|
| 갈래를 가린다 | 종류를 판별한다 |
| 조용히 무시한다 | 안내 없이 무시한다 |
| 아예 제외한다 | 전부 제외한다 |
| 한눈에 보기 | 구성 개요 |
| 잴 수 없음 | 측정 불가 |
| 견줄 자가 없었다 | 비교 기준이 없었다 |
| 손봐야 할 컬럼 | 조치가 필요한 컬럼 |
| 어긋난 곳 | 불일치 |
| 걸린 컬럼 | 영향 컬럼 |
| 지적 87건의 갈래 | 지적 87건의 구분 |
| 제목「가장 큰 것」 | 제목「최댓값」 |
| 줄을 세운다 | 정렬한다 |
| 잘려 나간다 | 제외된다 |
| 그 밖에 3개 | 이외 3개 |
| 몇 번 썼습니다 | 몇 회 사용되었다 |
| 2가지 타입 | 2종의 타입 |
| 일감으로 쓰시면 됩니다 | 작업 목록으로 사용할 수 있다 |
| 값이 큽니다 | 효과가 크다 |
| 사전이 불어납니다 | 사전이 증가한다 |
| 무엇을 하면 되는가 | 조치 사항 |
| 왜 그런가 | 판단 근거 |
| 어떻게 셌는가 | 산출 기준 |
| 점수의 속 | 점수의 구성 |
| 노릇을 한다 | 역할을 수행한다 |
| 숨을 수 있다 | 구별되지 않는다 |
| 고쳐 주지는 않는다 | 자동으로 수정하지 않는다 |
| 두 갈래로 나뉜다 | 두 종류로 구분된다 |
| 다섯 갈래 | 다섯 가지 구분 |
| 통째로 깨진다 | 전체가 표시되지 않는다 |
| 바탕에 묻힌다 | 배경과 구별되지 않는다 |
| 배경을 깐다 | 배경을 적용한다 |
| 한눈에 보이지 않는다 | 즉시 확인하기 어렵다 |

| 쓰지 않는 표현 | 바꿔 쓰는 표현 |
|---|---|
| 그 자리에서 확인한다 | 해당 위치에서 확인한다 |
| 조회하여 되돌린다 | 조회하여 복원한다 |
| 모든 쪽에 넣는다 | 모든 쪽에 삽입한다 |
| 제목 블록을 얹는다 | 제목 블록을 배치한다 |
| 알려 주는 형태 | 제공하는 형태 |
| 줄기 · 곁 상자 | 주 흐름 · 병행 상자 |
| 묶음선 | 연결선 |
| 길목에 놓인다 | 경로에 배치한다 |
| 앞엣것 · 뒤엣것 | 전자 · 후자 |
| 적재로 흘러든다 | 적재로 연결된다 |
| 맞대어 본다 | 대조한다 |
| 견준다 | 대조한다 |
| 네 축 | 네 가지 대조 대상 |

## 표기 규칙

- `고정폭 글꼴`: 코드, 파일명, 컬럼명, 명령어
- **굵은 글꼴**: 핵심 용어의 첫 등장
- 규칙 이름은 `word.abbr-unique`와 같이 점으로 구분하여 표기한다

콜아웃 네 종류를 다음과 같이 구분하여 사용한다.

> ! 중요
>
> 반드시 지켜야 할 판단 기준이다. 위반했을 때 어떤 문제가 발생하는지 함께 기술한다.

> ⚠ 주의
>
> 실제로 문제가 되었던 사항이다. 같은 실수를 방지하기 위한 경고이다.

> 요령
>
> 실무에서 도움이 되는 방법이나 참고 사항이다.

> i 참고
>
> 배경 설명이나 부가 정보이다.

## 문서 빌드

```
cd document/manual
quarto render                  # HTML + PDF
quarto render --to html
quarto render --to typst       # PDF만
quarto preview                 # 수정하면서 확인할 때
```

PDF는 LaTeX이 아니라 Typst로 조판한다. Quarto가 Typst를 내장하고 있어 별도 설치가 필요 없다.

> **폰트**
>
> 인쇄용 폰트를 `assets/fonts/print`에 동봉했으므로 따로 설치하지 않아도 된다. 본문은 `KoPubWorldBatang_Pro`, 제목은 `KoPubWorldDotum_Pro`, 코드는 `D2Coding`을 사용한다. HTML의 코드 폰트는 `assets/fonts/web`의 woff2를 사용한다.
>
> 폰트를 찾지 못하는 경우에는 `Noto Sans/Serif CJK KR`, `Apple SD Gothic Neo`, `Malgun Gothic` 순으로 대체되므로 조판이 손상되지는 않는다. 폰트 목록은 `assets/typst-header.typ`에서 변경한다.

> **표지 이미지를 넣으려면**
>
> `assets/images/cover.png`를 두고 `assets/typst-show.typ`의 `book.with()` 인자를 주석에 기재된 대로 변경하면 된다. 현재는 메타데이터로 표지를 조판한다.

## 조판 구성

표 0.4. 조판 구성 파일

| 파일 | 역할 |
|---|---|
| `_quarto.yml` | 프로젝트·출력 형식 설정 |
| `assets/typst-header.typ` | Typst 조판 (색상·폰트·헤딩·표·코드·목차) |
| `assets/typst-show.typ` | orange-book 템플릿 호출 오버라이드 |
| `assets/typst-code-fix.typ` | 코드 블록 개행·토큰 테두리 교정 |
| `assets/custom.scss` | HTML 테마 |
| `assets/fonts/` | 인쇄용(print)·웹용(web) 폰트 |
| `fix-korean-crossref.py` | 교차참조 어순 교정 (후처리) |

## 표 작성 규칙

표에는 캡션·식별자·열 너비를 함께 지정한다. 열 너비를 지정하지 않으면 Typst가 내용에 맞춰 자동으로 계산하는데, 한글 셀이 길면 열이 좁아져 읽기 어려워진다.

```
| 코드 | 구분 | 정의 |
|---|---|---|
| 01 | 일반어 | 업무나 일상에서 의미를 전달하는 일반적인 단어 |
```

```
: 단어구분 코드와 건수 {#tbl-word-section-example tbl-colwidths="[10,14,76]"}
```

식별자를 지정해 두면 본문에서 `@tbl-word-section-example`처럼 참조할 수 있다.

> **교차참조 어순**
>
> Quarto는 한국어에서도 “섹션 1.2”와 같이 영어 어순으로 참조를 생성한다. 한국어에서는 “1.2 절”이 자연스러우므로 렌더링이 끝난 뒤 후처리로 변경한다. PDF는 `typst-header.typ`의 `show ref` 규칙이 같은 처리를 수행한다.

# I

# 1부 · 데이터 표준화


1 표준화 개요 . . . . . 19
2 표준단어 . . . . . 27
3 표준용어 . . . . . 35
4 표준도메인 . . . . . 41
5 표준코드 . . . . . 47

CHAPTER 1

# 표준화 개요

KDSM의 데이터 표준은 네 개의 사전으로 구성된다. 표준단어, 표준용어, 표준도메인, 표준코드이다.

네 개의 사전은 나란히 놓인 것이 아니라 하나의 흐름으로 연결되어 있다. 단어를 조합하여 용어를 만들고, 용어가 도메인을 참조하여 데이터 타입과 길이가 정해지며, 값이 미리 정해진 경우에는 코드를 참조한다. 따라서 단어 하나를 수정하면 그 단어를 사용하는 용어가 영향을 받고, 그 용어로 이름을 부여한 컬럼까지 영향을 받는다.

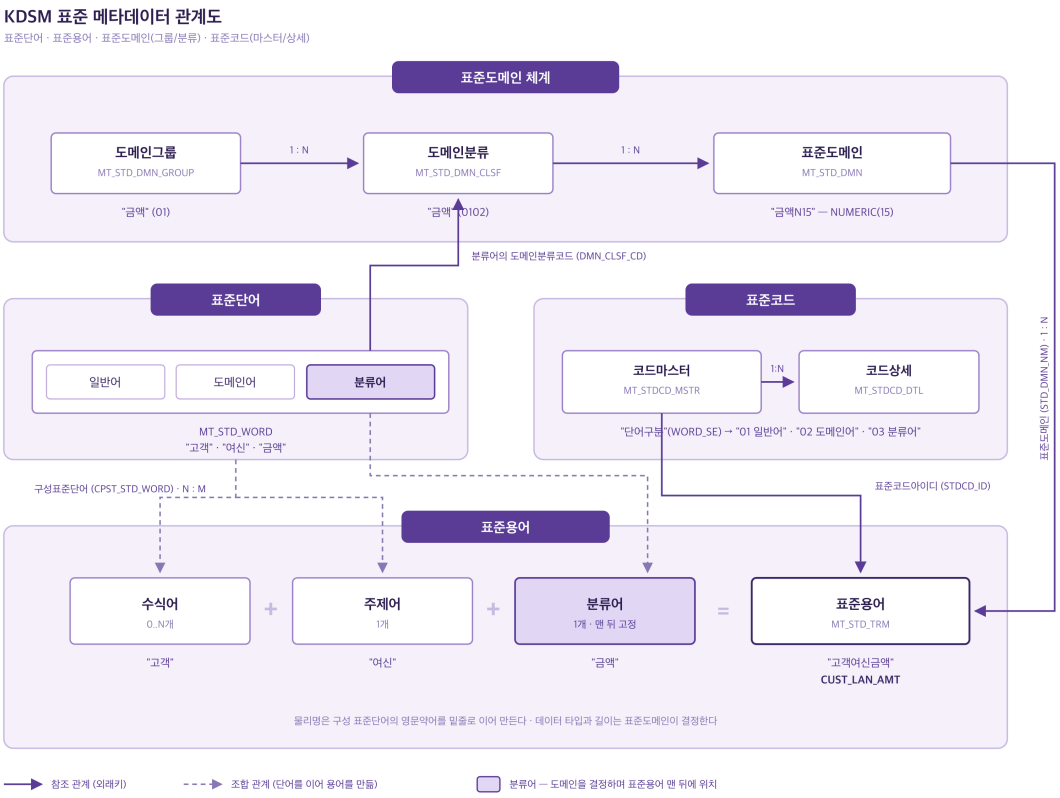


그림 1.1. KDSM 표준 메타데이터 관계도

## 1.1 네 개의 사전이 담당하는 역할

표 1.1. 네 개의 사전

| 사전 | 역할 | 물리 테이블 |
|---|---|---|
| 표준단어 | 이름의 최소 단위. 한글명에 영문명과 영문약어를 연결한다 | `MT_STD_WORD` |
| 표준용어 | 단어를 조합하여 만든 이름. 컬럼명이 되는 단위 | `MT_STD_TRM` |
| 표준도메인 | 데이터 타입과 길이를 결정한다. 3단 체계로 구분된다 | `MT_STD_DMN` 외 2 |
| 표준코드 | 값의 목록을 정의한다 | `MT_STDCD_MSTR` 외 1 |

## 1.2 이름이 결정되는 과정

`고객여신금액`이라는 컬럼 이름이 어떻게 결정되는지 따라가 보면 네 사전의 관계를 명확하게 파악할 수 있다.

먼저 단어를 선택한다. 표준단어 사전에서 `고객`, `여신`, `금액`을 선택한다. 세 단어는 성격이 서로 다르다.

```
고객   수식어    0..N개
여신   주제어    1개
금액   분류어    1개 · 맨 뒤 고정
```

세 단어를 순서대로 연결하면 표준용어 `고객여신금액`이 된다. 물리명은 각 단어의 영문약어를 밑줄로 연결하여 만든다.

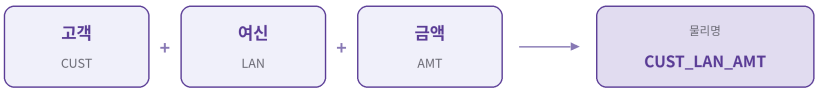


**그림 1.2.** 구성 단어의 영문약어를 연결하여 물리명을 만든다

여기까지가 이름이다. 타입과 길이는 아직 정해지지 않았다.

이를 결정하는 것은 맨 뒤의 분류어이다. `금액`이 도메인분류 `금액(0102)`을 참조하고, 그 아래의 표준도메인 `금액N15`가 `NUMERIC(15)`를 결정한다. 설계자가 임의로 15자리를 지정하는 것이 아니라, 분류어를 선택하는 순간 자동으로 결정된다.

> **!** 분류어가 맨 뒤에 오는 이유
>
> 이름의 마지막 단어가 그 값의 성격을 나타내기 때문이다. `고객여신금액`은 금액이고 `고객여신일자`는 날짜이다. 앞의 단어가 무엇이든 마지막 단어가 성격을 결정한다.
>
> 이 위치가 고정되어 있어야 이름만으로 타입을 판단할 수 있으며, 프로그램도 같은 방법으로 판단할 수 있다. 분류어가 맨 뒤에 있지 않으면 이 규칙이 성립하지 않는다.

값이 미리 정해진 경우에는 표준코드도 함께 참조한다. `고객등급코드`가 이에 해당한다. 코드마스터가 코드의 성격을 정의하고, 코드상세가 개별 값을 담는다.

## 1.3 사전 사이의 참조 관계

관계도의 실선은 참조(외래키)를, 점선은 조합을 나타낸다. 참조는 값이 대상 사전에 등재되어 있어야 한다는 뜻이고, 조합은 이 값이 대상 사전의 이름들을 연결하여 만든 문자열이라는 뜻이다.

**표 1.2.** 사전 사이의 참조

| 가리키는 곳 | 가리키는 대상 | 뜻 |
|---|---|---|
| `MT_STD_WORD.STD_DMN_CLSF_CD` | 도메인분류 | 분류어가 어떤 도메인분류인지 |
| `MT_STD_TRM.CPST_STD_WORD_CN` | 표준단어 | 이 용어를 구성하는 단어. `+`로 연결하여 기록한다 |
| `MT_STD_TRM.STD_DMN_NM` | 표준도메인 | 이 용어의 타입과 길이 |
| `MT_STD_TRM.STDCD_ID` | 코드마스터 | 값이 코드로 정해진 경우에만 |
| `MT_STD_DMN.STD_DMN_CLSF_CD` | 도메인분류 | 이 도메인이 속한 분류 |
| `MT_STD_DMN_CLSF.STD_DMN_GROUP_CD` | 도메인그룹 | 이 분류가 속한 그룹 |

## 1.4 구성단어는 외래키로 보장되지 않는다

`CPST_STD_WORD_CN`은 `고객 + 여신 + 금액`과 같이 하나의 항목에 여러 값을 연결하여 기록한 문자열이다. 값이 하나가 아니므로 외래키를 설정할 수 없다.

따라서 데이터베이스가 이 관계를 보장하지 못한다. 등재되지 않은 단어를 기록하여도 그대로 적재된다. 대신 `term.words-registered` 규칙이 `+`를 기준으로 분리하여 하나씩 사전과 대조한다(3장).

값을 여러 개 기록하는 항목은 모두 이러한 성격을 가진다. 데이터베이스가 차단하지 못하므로 검증을 실행하지 않으면 잘못된 값이 그대로 남는다. 이 관계는 논리 모델에서는 관계선으로 존재하지만 물리 모델에서는 외래키가 되지 않으므로, 논리 관계선의 수와 물리 외래키의 수가 서로 다른 것이 정상이다.

## 1.5 작업 순서

읽는 순서와 일하는 순서는 다르다. 실제로는 컬럼 이름 하나를 정할 때마다 아래 절차를 한 차례 수행한다.

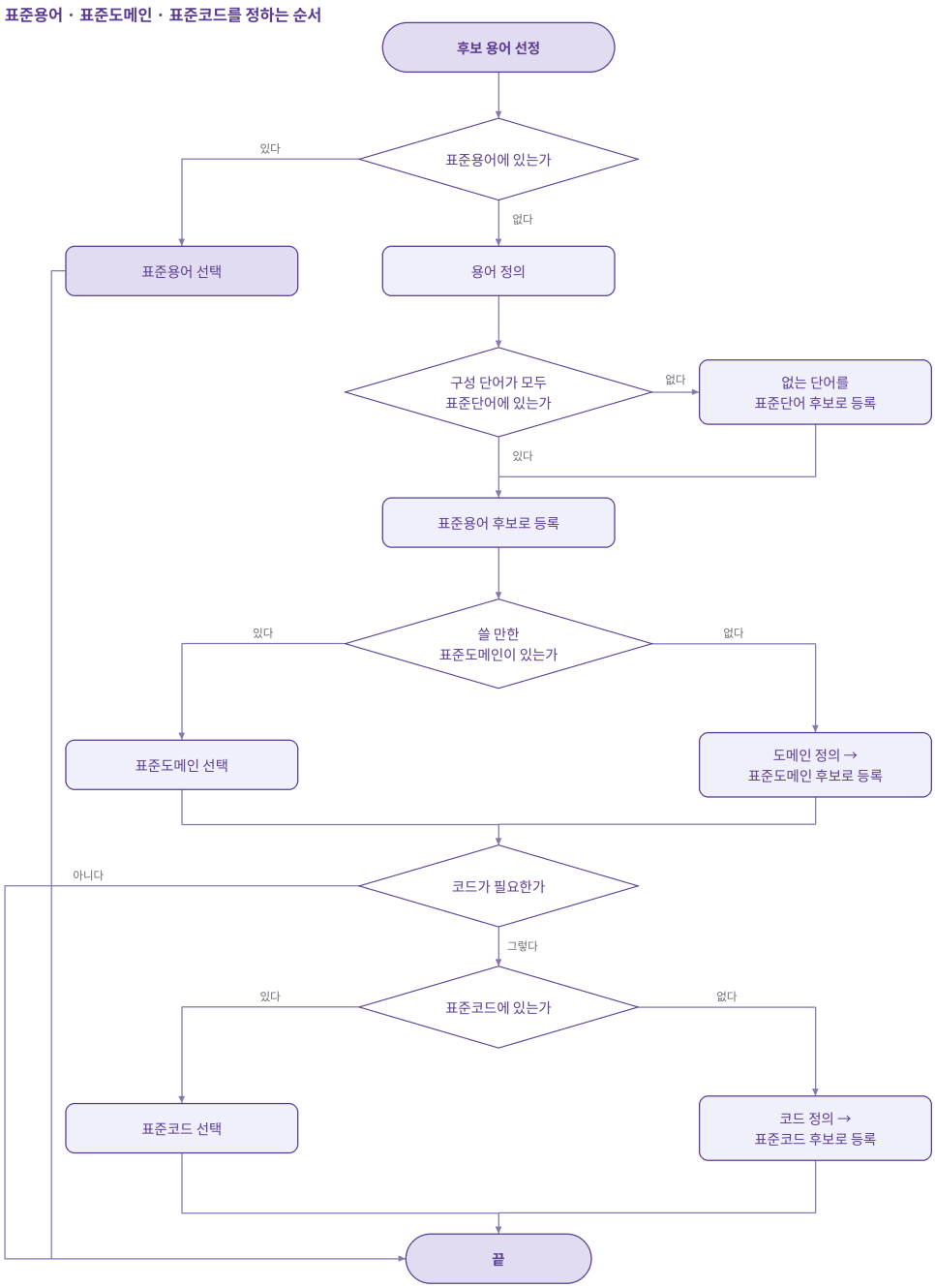


**그림 1.3.** 표준용어·표준도메인·표준코드를 정하는 순서

### 1.5.1 두 가지 경로

분기는 맨 위의 하나뿐이다. 찾는 용어가 이미 표준용어에 등재되어 있으면 왼쪽 경로로 진행하여 즉시 종료된다.

이것이 표준화의 가치이다. 표준용어 하나를 선택하면 도메인이 함께 결정되고, 코드까지 정해진 용어라면 코드도 함께 결정된다. 타입의 자릿수나 값의 목록을 다시 정의할 필요가 없다.

등재되어 있지 않은 경우에만 오른쪽 경로로 진행한다. 이때는 세 개의 사전을 차례로 거치면서 "등재되어 있으면 선택하고, 없으면 후보로 등록한다"를 반복한다.

표 1.3. 단계별 두 가지 처리

| 단계 | 있으면 | 없으면 |
|---|---|---|
| 표준용어 | 선택하고 종료한다 | 용어를 정의한다 |
| 표준단어 | 그대로 사용한다 | 등재되지 않은 단어만 후보로 등록한다 |
| 표준도메인 | 선택한다 | 도메인을 정의하여 후보로 등록한다 |
| 표준코드 | 선택한다 | 코드를 정의하여 후보로 등록한다 |

**! 등재가 아니라 후보 등록이다**

새로 정의한 것은 모두 후보로 등록한다. 즉시 사전에 등재하지 않는다.

한 사람이 임의로 만든 단어가 즉시 표준이 되면, 다른 사람이 같은 뜻의 단어를 다시 만들어도 이를 막을 수 없다. 사전은 여러 사람이 함께 사용하는 것이므로 등재에는 검토를 거친다. 검토가 진행되는 동안에도 작업은 후보 상태로 진행할 수 있다.

**2단계 검색은 도구가 대신한다**

"표준용어에 있는가"를 육안으로 확인하는 것은 4,000건이 넘는 사전에서는 적절한 방법이 아니다. `kdsm suggest`가 한글 이름을 단어 단위로 분리하여 후보를 점수와 함께 제시한다(12장).

등재되지 않은 단어가 포함되어 있으면 이를 함께 제시하므로, 위 순서도의 "없는 단어를 표준단어 후보로 등록" 단계까지 한 번에 확인할 수 있다.

```
kdsm suggest 고객여신금액 --from-db
```

`--from-db`는 적재된 사전에서 읽는다는 뜻이다. 아직 적재하기 전이라면 `--meta`로 메타파일을 지정한다(표 12.9).

**i 코드가 필요한지 판단하는 기준**

값이 미리 정해진 목록에서만 선택되는 경우가 코드이다. `고객등급`, `상품유형`, `처리상태`가 이에 해당한다.

반대로 금액·일자·이름과 같이 값의 범위가 제한되지 않으면 코드가 아니다. 분류어가 `코드`로 끝나는 것이 유용한 판단 근거이지만, 그것만으로 결정하지는 않는다. 값의 목록을 열거할 수 있는지가 기준이다.

## 1.6 이 부의 구성

**표 1.4.** 1부의 구성

| 장 | 다루는 내용 |
|---|---|
| 2장 | 이름의 최소 단위를 정의하는 방법. 영문명과 영문약어, 이음동의어와 금칙어 |
| 3장 | 단어를 조합하여 이름을 만드는 방법. 물리명이 계산되는 규칙 |
| 4장 | 데이터 타입과 길이를 결정하는 3단 체계 |
| 5장 | 값의 목록을 정의하는 방법과 코드로 정할 대상의 판단 기준 |

차례로 읽으면 앞에서 살펴본 과정과 같은 순서이다. 단어를 정의하고, 그것으로 용어를 만들며, 용어에 도메인을 지정하고, 값이 정해진 용어에 코드를 지정한다.

시간이 부족한 경우에는 표준용어부터 읽어도 무방하다. 실제 작업량이 가장 많은 부분이며, 앞뒤 단계가 왜 그렇게 구성되었는지도 그 과정에서 자연스럽게 드러난다.

네 사전을 모두 갖춘 뒤에 모델링을 시작하는 것은 아니다. 모델링에 필요한 이름이 사전에 없으면 그 자리에서 후보로 등록하고 작업을 이어 간다. 두 부의 관계는 6.3절 에 있다.

CHAPTER 2

# 표준단어

표준단어는 데이터 표준의 최소 단위이다. 표준용어는 표준단어의 조합으로만 만들어지므로, 여기서 정한 내용이 이후 단계 전체에 전파된다.

물리 테이블은 `MT_STD_WORD`, 메타파일 시트도 같은 이름이다.

## 2.1 사전의 구성

사전이 어떤 항목으로 구성되어 있고, 단어를 어떤 성격으로 구분하는지부터 살펴본다.

### 2.1.1 컬럼 구성

**표 2.1.** 표준단어 컬럼 구성

| 컬럼 | 한글명 | 설명 |
|---|---|---|
| `STD_WORD_NM` | 표준단어이름 | 사전의 기본키 |
| `WORD_ENG_NM` | 단어영문이름 | 영문 전체명. 표준용어의 영문명이 여기에서 유도된다 |
| `WORD_ENG_ABBR_NM` | 단어영문약어이름 | 영문 약어. 물리명이 여기에서 유도된다 |
| `WORD_EXPLN` | 단어설명 | 정의 및 설명 |
| `STD_DMN_CLSF_CD` | 도메인분류코드 | 분류어만 가진다 |
| `ALLO_SYNM_CN` | 이음동의어내용 | 쉼표로 구분 |
| `PHWRD` | 금칙어 | 쉼표로 구분 |
| `WORD_SE_CD` | 단어구분코드 | 01 일반어 / 02 도메인어 / 03 분류어 |

| 컬럼 | 한글명 | 설명 |
|---|---|---|
| STTS_CD | 상태코드 | 01 후보 / 02 확정 / 03 폐기 |

### 2.1.2 단어구분

단어구분은 단어의 역할을 구분한 것이다. 현재 분포는 표 2.2 과 같다.

표 2.2. 단어구분 코드와 건수

| 코드 | 구분 | 정의 | 건수 |
|---|---|---|---|
| 01 | 일반어 | 업무나 일상에서 의미를 전달하는 일반적인 단어 | 2,937 |
| 02 | 도메인어 | 특정 업무 영역이나 산업의 전문 용어 | 669 |
| 03 | 분류어 | 데이터의 형식과 타입을 결정하는 단어. 표준용어 맨 뒤에 위치 | 587 |

#### 2.1.2.1 분류 기준은 좁게 적용한다

도메인어와 일반어의 경계를 정하는 일이 가장 어렵다. 이 기준은 **좁게 적용한다**. 특정 업무영역이나 제도에서 정의된 전문용어만 도메인어(02)로 두고, 판단이 명확하지 않으면 일반어(01)로 분류한다.

> ! 판단이 애매하면 일반어
>
> 도메인어를 넓게 적용하면 해당 단어를 도메인어로 분류한 근거를 매번 설명해야 한다. 좁게 적용하면 기준이 단순해지고, 이후 특정 업무의 전문용어임이 분명해졌을 때 도메인어로 변경하면 된다. 도메인어를 일반어로 내리는 것보다 일반어를 도메인어로 올리는 편이 위험이 적다.

구체적으로 정한 것은 표 2.3 와 같다.

표 2.3. 도메인어·일반어 판단 사례

| 계열 | 예 | 구분 | 이유 |
|---|---|---|---|
| 업종 + 사 | 보험사, 증권사, 카드사 | 01 일반어 | 업종을 가리키는 일반적 조어이지 제도 용어가 아니다 |
| 문서명 + 서 | 청구서, 보증서, 계약서 | 01 일반어 | 문서 종류를 가리키는 일반명사다 |
| 제도 용어 | 세금계산서, 전자세금계산서 | 02 도메인어 | 세법이 정의한 특정 문서다 |
| 금융 업무 | 여신, 수신, 담보 | 02 도메인어 | 금융업에서 뜻이 고정된 용어다 |

"세금계산서"와 "청구서"의 차이가 이 기준을 잘 보여준다. 청구서는 여러 분야에서 널리 사용하는 일반적인 용어이지만, 세금계산서는 부가가치세법이 형식과 요건을 정해 둔 특정 문서이다.

#### 2.1.2.2 어근과 파생어의 일관성

어근이 같은 단어는 구분을 일치시킨다. “기본키”와 “대체키”를 도메인어로 두었다면 어근인 “키”도 도메인어여야 한다. 어근만 일반어로 남아 있으면 조합의 성격이 중간에 달라진다.

## 2.2 이름과 금지어

영문약어는 물리명의 재료이며, 이음동의어와 금칙어는 사용할 수 없는 말을 지정한다. 두 가지 모두 이름을 정하는 작업에 해당한다.

### 2.2.1 영문약어

물리명이 영문약어의 조합으로 만들어지므로, 약어는 SQL 식별자로 사용할 수 있어야 한다.

- 대문자로 시작하고 대문자·숫자만 사용 (`[A-Z][A-Z0-9]*`)
- 사전 전체에서 유일해야 함

약어가 중복되면 서로 다른 두 용어가 같은 물리명을 갖게 되어 컬럼명이 충돌한다. 검증 규칙 `word.abbr-unique`가 이를 검출한다.

> 약어를 정할 때
>
> 기존 약어와의 충돌을 먼저 확인해야 한다. 이후에 약어를 변경하면 해당 단어를 사용하는 모든 표준용어의 물리명이 변경되고, 이미 생성한 테이블의 컬럼명까지 함께 변경된다.

### 2.2.2 이음동의어와 금칙어

이음동의어는 해당 표현 대신 지정된 표준단어를 사용하도록 안내하는 장치이다.

> ⚠ 이음동의어는 표준단어가 아니다
>
> 이음동의어로 지정된 말은 표준단어로 사용할 수 없다. 사실상 금칙어와 동일하게 취급한다.

이음동의어는 유효한 표기로 인정하지 않는다. “이름”의 이음동의어에 “명”이 있더라도 “명”을 사용할 수는 없다. 이음동의어까지 인정하면 같은 개념에 두 개의 이름이 공존하게 되어 표준화의 목적이 훼손되기 때문이다.

`word.synonym-registered` 규칙은 이음동의어가 표준단어로도 등재된 경우를 경고로 검출한다. `word.synonym-cycle` 규칙은 A의 이음동의어가 B이고 B의 이음동의어가 A인 순환 관계를 검출한다.

#### 2.2.2.1 대체어가 정말 같은 뜻인지

이음동의어를 지정한다는 것은 “두 단어가 같은 뜻이다”라고 선언하는 일이다. 그 선언이 잘못되면 표준화가 원래의 뜻을 바꾸게 된다.

> ! 어색한 대체어는 폐기 판단이 틀렸다는 신호다
>
> 추천기가 제시한 이름이 통용어와 크게 달라 보이면 그 위화감을 근거로 삼는다. “데이터표준점수”가 “자료표준점수”로 바뀌어 제시되는 것이 그러한 예이다. 규칙이 과도한 것이 아니라 대체 관계가 잘못 지정된 경우가 많으므로, 해당 폐기어와 대체어를 사전에서 확인한다.

`word.synonym-meaning` 규칙은 폐기한 단어와 대체어의 영문명을 대조한다. 영문명이 물리명의 근거이므로, 영문명이 다르다는 것은 만들어질 컬럼명이 달라진다는 뜻이기도 한다.

```
· 폐기한 '파라미터'(Parameter)를 '변수'(Variable)로 대체하도록
  이음동의어에 적혀 있으나 영문명이 다릅니다.
```

다음은 이 규칙으로 드러나는 대표적인 오류 유형이다.

표 2.4. 영문명이 달라 드러나는 대체 오류

| 폐기어 | 대체어 | 문제 |
|---|---|---|
| 사전(Dictionary) | 이전(Before) | 정의는 辭典인데 대체는 事前 기준이었다 |
| 파라미터(Parameter) | 변수(Variable) | parameter는 variable이 아니다 |
| 프로젝트(Project) | 사업(Business) | project는 business가 아니다 |
| 어린이(Child) | 자녀(Children) | 어린이는 자녀가 아니다 |

⚠ 영문명이 같아도 안심할 수 없다

이 규칙은 검토 대상을 한정하는 역할만 한다. 영문명이 같으면 검출되지 않으므로, 다음과 같은 잘못된 대체는 규칙을 통과한다.

```
라이브러리(Library) → 도서관     소프트웨어 라이브러리는 도서관이 아니다
지반(Ground)      → 지상       地盤과 地上은 다른 말이다
데이터(Data)      → 자료       뜻은 같으나 통용성이 다르다
```

그러므로 검증 통과를 정합성의 근거로 삼지 않는다. 폐기어와 대체어의 쌍은 사람이 전수 검토한다. 규칙은 검토 순서를 정하는 데 사용한다.

#### 2.2.2.2 표준단어는 통용어로 정한다

폐기 대상을 정할 때 판단이 한 방향으로 치우치기 쉽다. 국어사전에 등재된 한자어와 고유어를 우선하고 업무에서 관용적으로 정착한 외래어를 폐기하는 방향이다. 데이터·노드·페이지·폰트·템플릿·카테고리·홈페이지가 이러한 판단에 걸리기 쉬운 낱말이다.

! 순화어보다 통용어

표준단어는 업무 현장에서 실제로 사용하는 말로 정한다. 사용되지 않는 말을 표준으로 삼으면 표준을 우회하는 이름이 생기고, 그 시점부터 표준은 문서상으로만 존재하게 된다.

바로잡는 시점은 이를수록 좋다. 표준용어가 아직 적을 때에는 폐기어를 구성단어로 사용하는 용어가 거의 없으므로 사전만 고치면 된다. 산출물이 축적된 뒤에는 용어와 산출물을 모두 고쳐야 하므로 되돌리는 비용이 급격히 증가한다.

전수 검토는 다음 다섯 가지로 판정한다. "뜻이 달라 서로 대체할 수 없다"가 절반 가까이를 차지하므로 이 유형을 반드시 둔다. 이 유형이 없으면 되살릴 수 없는 낱말을 되살리는 쪽으로 판정이 몰린다.

표 2.5. 폐기어 82건의 판정

| 판정 | 뜻 | 건수 |
|---|---|---|
| 유지 | 폐기가 옳고 대체어도 맞다 | 41 |

| 판정 | 뜻 | 건수 |
|---|---|---|
| 둘다살림 | 뜻이 달라 대체할 수 없다. 되살리고 대체 관계만 해제한다 | 37 |
| 되살림 | 뜻은 같으나 표준으로 삼을 대상이 뒤바뀌었다 | 3 |
| 정의수정 | 단어는 적절하나 정의가 잘못 기술되어 있다 | 1 |
| 대체어교체 | 폐기는 옳으나 대체어가 틀렸다 | 4 |

검토 결과 상당수가 확정으로 전환되고, 반대로 통용성이 낮은 것으로 판정된 낱말이 새로 폐기된다. 폐기 목록은 한 번 정하고 끝나는 것이 아니라 검토를 거쳐 양방향으로 조정된다.

⚠ 되살리면 영문약어가 겹친다

폐기했던 말과 대체어는 대개 영문약어가 같다. 한쪽이 폐기 상태였기 때문에 `word.abbr-unique` 검사에서 제외되었을 뿐이다. 두 단어를 모두 되살리면 그 시점에 위반이 된다.

폐기어와 대체어의 절반 이상이 약어를 공유하므로, 되살릴 낱말을 정한 뒤에는 약어 충돌을 함께 정리해야 한다. 뜻이 다른 쌍은 한쪽의 약어를 새로 지정한다.

```
라이브러리 LIB → SWLIB      도서관은 LIB 유지
지상       GRND → ABGRND    지반은 GRND 유지
조항       ARTCL → CLAUS    항목은 ARTCL → ITM
```

새 약어는 영문명에서 도출하되, 변경 대상 단어에 부여한다.

ℹ 약어가 영문명에서 기계적으로 나와야 하는 것은 아니다

`일자`는 영문명이 `Date`이지만 약어는 `YMD`이다. `_YMD` 접미사가 일자를 담는 컬럼의 표기로 정착해 있기 때문이다. `word.abbr-format` 규칙은 형식만 검사하므로 이러한 관용 약어를 제한하지 않는다.

다만 그러한 사정을 영문명에 괄호로 기재해서는 안 된다. 영문명은 뜻을 대조하는 기준이자 영문 용어명의 재료이므로, `Date(Year Month Day)`로 기재하면 `계약일자`가 `Contract Date(Year Month Day)`가 된다. 이러한 관행은 정의(`WORD_EXPLN`)에 기술한다.

## 2.3 분류어와 상태

분류어는 표준도메인으로 연결되는 기준이며, 상태는 해당 단어를 현재 사용할 수 있는지를 나타낸다.

### 2.3.1 분류어와 도메인분류

분류어(형식단어)는 표준용어의 맨 뒤에 오면서 데이터의 형식을 결정하는 단어이다. “일자”, “금액”, “여부”, “코드” 등이 이에 해당한다.

분류어만 `STD_DMN_CLSF_CD`(도메인분류코드)를 가진다. 이 코드가 표준도메인으로 연결되는 기준이다.

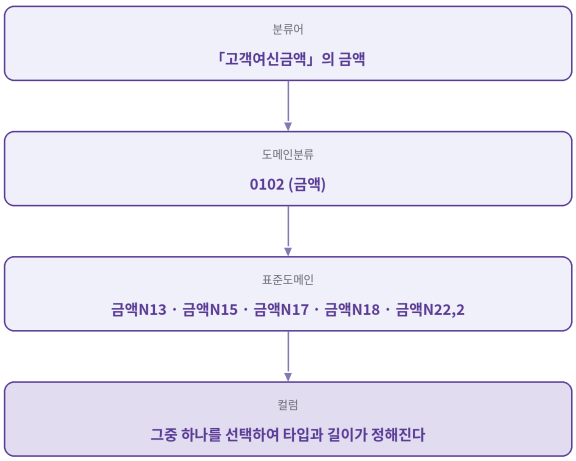


**그림 2.1.** 분류어 하나가 컬럼의 타입과 길이까지 결정한다

`word.classifier-domain` 규칙은 두 방향을 함께 검사한다. 분류어에 분류코드가 없어도 안 되고, 분류어가 아닌 단어에 분류코드가 있어도 안 된다.

### 2.3.2 상태

**표 2.6.** 상태 코드와 검사 대상 여부

| 코드 | 상태 | 검사 대상 |
|---|---|---|
| 01 | 후보 | 포함 |
| 02 | 확정 | 포함 |
| 03 | 폐기 | 제외 |

폐기 항목은 모든 검증 규칙에서 제외된다. 폐기한 단어는 더 이상 사전의 일부가 아니므로, 해당 단어가 보유한 영문약어나 이음동의어를 위반으로 집계하면 정리 작업의 효과가 결과에 드러나지 않는다.

후보(01)는 검사 대상으로 유지한다. 확정 이전에 문제를 발견하는 것이 검증의 목적이기 때문이다.

> **i 폐기 상태와 이력 관리의 구분**
>
> 폐기 상태는 해당 단어를 사용하지 말라는 현재 시점의 금지 표시이다. 이력 관리는 변경 시점과 변경자, 변경 사유를 남기는 기록이다. 전자는 현재 시점에 필요하고, 후자는 애플리케이션 구축 이후로 미룰 수 있다.

## 2.4 표준단어 검증 규칙

**표 2.7.** 표준단어 검증 규칙 8종

| 규칙 | 심각도 | 검사 내용 |
|---|---|---|
| `word.name-unique` | ERROR | 표준단어명은 유일해야 한다 |
| `word.abbr-unique` | ERROR | 영문약어는 유일해야 한다 |
| `word.abbr-format` | ERROR | 영문약어는 `[A-Z][A-Z0-9]*` 형식이어야 한다 |
| `word.classifier-domain` | ERROR | 분류어는 도메인분류코드가 있어야 하고, 그 외는 없어야 한다 |
| `word.forbidden` | ERROR | 금칙어로 등록된 말이 표준단어로도 등재되면 안 된다 |
| `word.synonym-registered` | WARNING | 이음동의어로 지정된 말이 표준단어로 등재되면 안 된다 |
| `word.synonym-cycle` | WARNING | 이음동의어가 서로를 가리키는 순환이 없어야 한다 |
| `word.synonym-meaning` | WARNING | 대체 관계로 묶인 두 단어의 영문명이 같아야 한다 |

`word.synonym-meaning` 규칙만 폐기 항목을 검사한다. 폐기어와 대체어를 대조하는 것이 이 규칙의 목적이므로, 폐기 항목을 제외하면 검사 대상이 남지 않는다.

```
from kdsm.io import MetaWorkbook
from kdsm.validate import Validator, word_rules

wb = MetaWorkbook.open("dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx")
print(Validator(word_rules()).run(wb.read_words()).to_text())
```

CHAPTER 3

# 표준용어

표준용어는 엔터티명과 속성명으로 사용하는 이름이다. 표준단어를 순서대로 조합하여 만든다.

물리 테이블은 `MT_STD_TRM`이다.

## 3.1 구조와 구성

표준용어가 어떤 구조로 만들어지고 어떤 항목을 갖는지 살펴본다. 이 구조가 자동화의 근거이다.

### 3.1.1 조합 규칙

```
주제어 + 수식어* + 분류어
```

수식어는 없거나 여럿, 주제어는 하나, 분류어는 맨 뒤에 하나이다.

> 이 구조가 자동화의 근거
>
> 한글명을 표준단어 단위로 분리할 수 있으면 나머지는 기계적으로 결정된다. 물리명은 영문약어를 밑줄로 연결하고, 영문명은 영문 전체명을 공백으로 연결하며, 표준도메인은 마지막 분류어에서 유도된다. 어려운 부분은 분리하는 작업뿐이며, 이는 `kdsm.suggest`가 수행한다(12장).

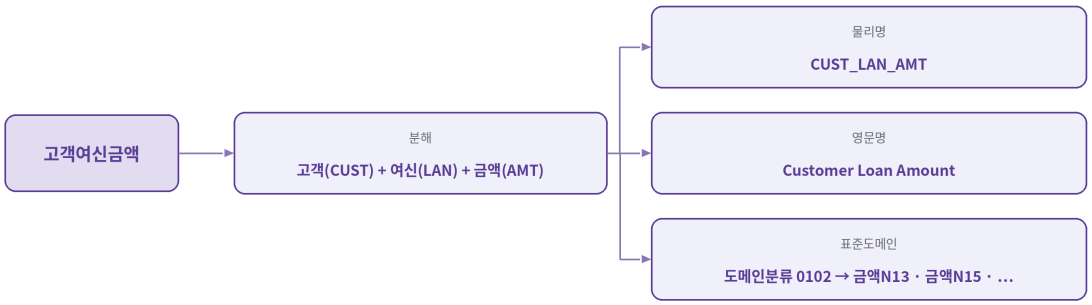


**그림 3.1.** 한글명을 단어로 분리하면 나머지가 기계적으로 결정된다

### 3.1.2 컬럼 구성

**표 3.1.** 표준용어 컬럼 구성

| 컬럼 | 한글명 | 설명 |
|---|---|---|
| `STD_TRM_NM` | 표준용어이름 | 사전의 기본키 |
| `TRM_ENG_NM` | 용어영문이름 | 영문 전체명 |
| `TRM_PHYS_NM` | 용어물리이름 | 물리명. 컬럼명이 된다 |
| `TRM_EXPLN` | 용어설명 | |
| `CPST_STD_WORD_CN` | 구성표준단어내용 | `고객 + 여신 + 금액` 형식 |
| `STD_DMN_NM` | 표준도메인이름 | 속성명만 가진다 |
| `STDCD_ID` | 표준코드아이디 | 표준코드에 해당할 때만 |
| `TRM_SE_CD` | 용어구분코드 | 01 엔터티명 / 02 속성명 / 03 양쪽 |
| `STTS_CD` | 상태코드 | |

## 3.2 용어의 쓰임

엔터티명으로 사용하는지 속성명으로 사용하는지에 따라 적용되는 규칙이 달라진다.

### 3.2.1 용어구분

**표 3.2.** 용어구분별 제약과 건수

| 코드 | 구분 | 분류어 종결 | 표준도메인 | 건수 |
|---|---|---|---|---|
| 01 | 엔터티명 | 불필요 | 불필요 | 12 |
| 02 | 속성명 | 필수 | 필수 | 90 |
| 03 | 양쪽 | 필수 | 필수 | 0 |

엔터티명은 테이블 이름이 될 뿐 컬럼이 아니므로, 분류어로 끝날 이유도 표준도메인을 가질 이유도 없다. “표준단어”, “표준용어”, “도메인그룹” 등이 엔터티명에 해당한다.

> ! 엔터티명에 제약을 걸지 않는 이유
>
> 모든 용어에 분류어 종결을 요구하면 엔터티명이 전부 위반으로 검출된다. “표준단어”는 분류어로 끝나지 않지만 테이블 이름으로는 문제가 없다. 규칙이 검출한 것을 무조건 데이터의 잘못으로 보지 않는다. 한 구분의 값이 모두 위반으로 나온다면 규칙의 적용 범위를 먼저 확인한다.

`term.ends-classifier`와 `term.domain-registered`는 용어구분이 속성명(02)이나 양쪽(03)인 것만 검사한다. 용어구분이 비어 있으면 속성명인지 알 수 없으므로 건너뛴다.

### 3.2.2 한 단어가 곧 용어인 경우

“고객”과 같이 하나의 단어가 그대로 표준용어가 되기도 한다. 이 경우에는 분류어 종결을 요구하지 않는다. 조합이 아니라 단어 자체이므로 조합 규칙을 적용할 대상이 아니다.

이 예외는 사전의 실제 구성에서 나온 것이다. `생성자`·`제안자`·`담당자`처럼 사람을 가리키는 낱말은 공공데이터 공통표준에서 가져온 것이 많고 대부분 일반어로 등재되어 있다. 이들을 분류어로 올리거나 뒤에 `이름`을 덧붙이도록 강제하면 오히려 공통표준에서 멀어진다.

다만 안내 없이 생략하지는 않는다. 해당 단어가 분류어가 아니면 INFO로 보고한다.

```
정보: 속성명 '생성자': 표준단어 하나가 그대로 용어가 되어 분류어 종결을 검사하지
      않았습니다. 쪼갤 여지가 없어 예외로 둔 것이며 위반이 아닙니다.
```

INFO는 통과·실패를 가르지 않는다. 오류도 경고도 아니므로 종료 코드가 바뀌지 않고 `--strict`에도 걸리지 않는다. 남기는 것은 “검사하지 않았다”는 사실 하나이다.

> i 검사한 것과 건너뛴 것을 구별한다
>
> 건너뛴 항목을 아무 말 없이 두면 결과만 보는 사람은 검사를 통과한 것으로 읽는다. 확인하지 않은 것을 확인한 것처럼 보고하지 않는다는 기준을 여기에도 적용한다. 진단이 길이를 재지 못한 항목을 따로 밝히는 것과 같은 이유이다(18.5.5절).
>
> 그 단어가 분류어이면 알리지 않는다. 검사할 것이 없어 넘어간 것이 아니라 실제로 분류어로 끝난 것이기 때문이다.

`금칙어`·`생성자`·`수정자`·`기본키`·`문자셋`과 같이 그 자체로 하나의 개념을 이루는 낱말이 여기에 해당한다.

## 3.3 이름이 만들어지는 방식

구성 단어를 정하면 물리명과 영문명이 따라 나온다. 사람이 옮겨 적는 것이 아니라 계산되는 값이다.

### 3.3.1 구성 단어는 표준단어명만

> ⚠ 경고
>
> `CPST_STD_WORD_CN`에는 표준단어명만 사용할 수 있다. 이음동의어는 표준단어가 아니므로 구성단어로 사용할 수 없다.

"이름"의 이음동의어에 "명"이 있더라도, 구성단어에 "명"이라고 기재하면 `term.words-registered` 규칙에 검출된다. "이름"으로 기재해야 한다.

### 3.3.2 물리명은 계산되는 값

물리명은 임의로 작성하는 것이 아니라 구성 단어의 영문약어를 밑줄로 연결한 결과이다.




그림 3.2. 물리명은 구성 단어의 영문약어로 계산한다

`term.physical-matches` 규칙이 이 계산 결과와 실제 물리명을 대조한다. 구성 단어가 사전에 없으면 기대값을 만들 수 없으므로 건너뛰고, 그 문제는 `term.words-registered`가 보고한다.

한글명에도 같은 대조가 있다. `term.name-matches` 규칙이 구성 단어명을 이어 붙인 값과 표준용어명을 대조한다. 이 규칙은 표준단어 사전을 참조하지 않는다. 비교할 두 값이 모두 용어 자신에게 있기 때문이다. 그러므로 사전을 제공하지 않아도 규칙 묶음에서 제외되지 않는다.

> ⚠ 약어를 추정하여 물리명을 짓지 않는다
>
> `고객주문일자`의 물리명을 `CUST_ORD_DT`로 짐작하기 쉬우나, 사전에 등재된 약어가 `ORDR`·`YMD`라면 실제 물리명은 `CUST_ORDR_YMD`이다. 통상적으로 쓰이는 약어와 사전에 등재된 약어는 다를 수 있다.
>
> 물리명은 추정하지 않고 사전에서 계산하여 확인한다. `term.physical-matches`가 이 차이를 검출하지만, 검출된 뒤에 고치면 그 이름을 사용한 산출물을 모두 함께 고쳐야 한다.

### 3.3.3 표준코드 참조

표준코드에 해당하는 용어만 `STDCD_ID`를 가진다. 표 3.3 의 6건이 전부이다.

표 3.3. 표준코드를 참조하는 표준용어

| 표준용어 | 물리명 | 참조 코드 |
|---|---|---|
| 단어구분코드 | `WORD_SE_CD` | `WORD_SE` |
| 상태코드 | `STTS_CD` | `STTS` |
| 용어구분코드 | `TRM_SE_CD` | `TRM_SE` |
| 엔터티구분코드 | `ENTTY_SE_CD` | `ENTTY_CLSF` |
| 속성유형코드 | `ATRB_TYPE_CD` | `ATRB_TYPE` |
| 자료발생주기코드 | `DATA_OCRN_CYCLE_CD` | `DT_OCRN_CYCLE` |

여섯 건 모두 `_CD`로 끝난다. `_YN` 계열은 하나도 없는데, 그 이유는 5.1.2절 에서 다룬다.

`term.code-registered` 규칙은 이 참조가 코드마스터에 존재하는지 검사한다. 값이 비어 있는 경우는 위반이 아니며, 값이 있으나 해당 코드가 등재되어 있지 않은 경우가 문제이다.

## 3.4 표준용어 검증 규칙

표 3.4. 표준용어 검증 규칙 8종

| 규칙 | 심각도 | 참조 | 검사 내용 |
|---|---|---|---|
| `term.name-unique` | ERROR | | 표준용어명은 유일해야 한다 |
| `term.physical-unique` | ERROR | | 물리명은 유일해야 한다 |
| `term.name-matches` | ERROR | | 용어명이 구성 표준단어명 조합과 일치해야 한다 |
| `term.words-registered` | ERROR | O | 구성 표준단어가 모두 사전에 있어야 한다 |
| `term.ends-classifier` | ERROR | O | 속성명의 마지막 구성 단어는 분류어여야 한다. 단어 하나가 그대로 용어이면 INFO로 알린다(3.2.2절) |
| `term.physical-matches` | ERROR | O | 물리명이 영문약어 조합과 일치해야 한다 |
| `term.domain-registered` | ERROR | O | 속성명의 표준도메인이 사전에 등재되어 있어야 한다 |
| `term.code-registered` | ERROR | O | 표준코드 아이디가 코드마스터에 등재되어 있어야 한다 |

참조 표시가 있는 규칙은 다른 사전을 함께 제공해야 검사할 수 있다. 제공하지 않으면 규칙 묶음에서 제외된다.

```
term_rules()                        # 3종 — 용어 자체 검사만
term_rules(words)                   # 6종
term_rules(words, domains)          # 7종
term_rules(words, domains, codes)   # 8종
```

> ! 검사하지 않은 것을 통과로 착각하지 않기
>
> 참조를 제공하지 않았을 때 규칙을 "통과"로 처리하지 않고 묶음에서 제외하는 것은 의도한 설계이다. 리포트에 나타나지 않는 것은 검사하지 않았다는 뜻이며, 문제가 없다는 뜻이 아니다.

CHAPTER 4

# 표준도메인

표준도메인은 데이터 타입과 길이를 묶어 정의한 것이다. 컬럼마다 타입을 개별적으로 정하지 않고 도메인이 결정하도록 하는 장치이다.

물리 테이블은 `MT_STD_DMN`, 그 위에 도메인그룹(`MT_STD_DMN_GROUP`)과 도메인분류(`MT_STD_DMN_CLSF`)가 있다.

## 4.1 체계와 이름

표준도메인은 그룹·분류·도메인 3단으로 이루어진다. 이름만으로 타입과 길이를 알 수 있도록 명명한다.

### 4.1.1 3단 체계

```
도메인그룹 (8)  ──1:N──>  도메인분류 (53)  ──1:N──>  표준도메인 (139)
  금액(01)                  금액(0102)                 금액N15 – NUMERIC(15)
```

표 4.1. 표준도메인 3단 체계

| 층 | 코드 체계 | 예 |
|---|---|---|
| 도메인그룹 | 2자리 | `01` 금액 |
| 도메인분류 | 4자리 = 그룹코드 + 일련번호 | `0102` 금액 |
| 표준도메인 | 코드 없음. 이름이 곧 식별자 | `금액N15` |

도메인그룹 8종은 금액, 날짜/시간, 내용, 명칭, 번호, 수량, 율, 코드이다.

### 4.1.2 도메인명 규칙

```
도메인분류명 + 데이터타입 약자 + 길이[,소수자릿수]
```

표 4.2. 데이터 타입 약자

| 타입 | 약자 | 예 |
|---|---|---|
| NUMERIC | N | `금액N15`, `가격N9,2` |
| CHAR | C | `연월일C8` |
| VARCHAR | V | `주소V200` |
| DATETIME | D | `연월일시분초D` |

> **이름만 보고 타입을 알 수 있게**
>
> `금액N15`라는 이름만으로 NUMERIC(15)임을 알 수 있다. 도메인 사전을 조회하지 않아도 되며, 컬럼 정의서에 기록된 타입이 맞는지 육안으로 대조할 수 있다.

DATETIME은 길이를 지정하지 않는다. 길이가 지정되어 있으면 이 또한 위반에 해당한다.

`domain.name-convention` 규칙이 이름 규칙을 검사한다. 기대 이름을 생성하려면 분류코드에 대응하는 분류명이 필요하므로 도메인분류 사전을 참조한다.

### 4.1.3 그룹·분류는 코드로 관리한다

`MT_STD_DMN`은 소속 그룹과 분류를 이름이 아니라 코드로 보유한다.

표 4.3. 표준도메인의 그룹·분류 코드 컬럼

| 컬럼 | 값 예 |
|---|---|
| `STD_DMN_GROUP_CD` | `01` |
| `STD_DMN_CLSF_CD` | `0102` |

> **이름에서 코드로 바꾼 이유**
>
> 이름이 아니라 코드를 보관하면 분류명을 수정하여도 이 값을 함께 수정할 필요가 없다. 이름을 보관하면 분류명을 수정할 때마다 이를 참조하는 모든 행을 함께 수정해야 한다. 다만 이름을 확인하려면 도메인분류 사전을 조회해야 한다.

분류코드는 `그룹코드 + 일련번호`이므로 두 컬럼이 서로 일치해야 한다. `domain.classification` 규칙이 등재 여부와 함께 이 정합성도 검사한다. 분류코드가 그룹코드로 시작하지 않는 조합, 예를 들어 그룹코드 `02`에 분류코드 `0102`를 기재한 경우가 위반으로 검출된다.

## 4.2 길이 정하기

길이는 컬럼의 크기를 직접 결정한다. 길이를 짧게 지정하면 데이터를 적재할 때 오류가 발생한다.

### 4.2.1 길이

`DMN_LEN`은 Null을 허용한다. DATETIME처럼 길이 개념이 없는 타입 때문이다.

> ⚠ 검증 규칙과 데이터베이스 제약을 같은 기준으로 맞춘다
>
> `domain.length-required` 규칙은 DATETIME 에 길이가 없는 상태를 정상으로 판정한다. 그런데 `DMN_LEN` 컬럼을 NOT NULL 로 정의하면 데이터베이스는 그 반대를 요구하게 된다. 검증을 통과한 정의서가 적재에서 중단되는 형태이다.
>
> 값이 없는 것이 정상인 항목은 산출물의 Null 허용을 `Y` 로 기재한다. 규칙과 제약이 서로 다른 기준을 적용하고 있지 않은지는 적재 이전에 확인한다(21장).

### 4.2.2 길이는 넉넉하게

길이 초과는 도메인 길이가 실제 값보다 짧을 때 적재 시점에 발생한다. 설명·표현형식·값범위와 같이 내용을 기술하는 컬럼에서 특히 자주 발생한다. 다음은 길이를 늘리게 되는 대표적인 컬럼이다.

표 4.4. 길이가 부족하기 쉬운 컬럼

| 컬럼 | 이전 | 이후 |
|---|---|---|
| `STRG_FRM_CN` 저장형식내용 | 내용V40 | 내용V80 |
| `EXPRS_FRM_CN` 표출형식내용 | 내용V40 | 내용V80 |
| `PRM_VL_SCP_CN` 허용값범위내용 | 내용V80 | 내용V256 |

> 💡 설명성 컬럼은 길이에 여유를 둔다
>
> 허용값범위나 표출형식과 같이 자유롭게 기술하는 항목은 길이를 여유 있게 지정해야 한다. 91자인 값 하나 때문에 80을 100으로 늘리면 여유가 9자에 불과하여 곧 다시 부족해진다. 한 단계 크게 지정하는 것이 적절하다.

이러한 문제를 사전에 확인하려면 `column.length-satisfied` 규칙을 사용한다(13.3.10절).

## 4.3 사전 사이의 관계

도메인분류가 그룹 정보를 함께 보유하는 이유와 그 관계를 양방향으로 검사하는 이유를 설명한다.

### 4.3.1 도메인분류의 비정규화

메타파일의 `MT_STD_DMN_CLSF` 시트는 소속 그룹의 코드와 이름을 함께 보유한다. 이름은 가독성을 위해 둔 값이므로, 그룹명을 변경하면 이 값이 실제와 어긋난다. `domain-group.group-name-matches` 규칙이 이를 검출한다.

같은 이름의 테이블에는 그룹명 컬럼이 없다. 그룹 코드로 연결되어 있으므로 동일한 값을 중복해서 보관하지 않기 때문이다. 따라서 데이터베이스에서 읽은 분류에는 이 값이 없으며, 값이 없는 경우 위 규칙은 검사를 건너뛴다.

### 4.3.2 부모-자식은 양방향으로

도메인그룹과 분류는 부모-자식 관계이다. 한쪽만 검사해서는 확인할 수 없는 문제가 있으므로 두 방향으로 검사한다.

표 4.5. 부모-자식 양방향 검사

| 방향 | 검출 대상 | 규칙 |
|---|---|---|
| 부모 → 자식 | 자식 없는 부모 | `has-classifications` |
| 자식 → 부모 | 부모 없는 자식 | `orphan-classification` |

폐기 항목 제외 규칙과 맞물려, 부모를 폐기하면 그 자식이 고아 항목으로 검출된다. 정리 작업이 완료되지 않았음을 나타내는 신호이다.

## 4.4 검증 규칙

### 4.4.1 표준도메인 4종

표 4.6. 표준도메인 검증 규칙 4종

| 규칙 | 참조 | 검사 내용 |
|---|---|---|
| `domain.name-unique` | | 표준도메인명은 유일해야 한다 |
| `domain.length-required` | | 길이가 필요한 타입에 길이가 지정되어야 한다 |
| `domain.name-convention` | O | 도메인명이 `분류명+타입+길이` 규칙을 따라야 한다 |
| `domain.classification` | O | 도메인분류 코드가 사전에 있고 그룹코드와 맞아야 한다 |

### 4.4.2 도메인그룹·분류 9종

표 4.7. 도메인그룹·분류 검증 규칙 9종

| 규칙 | 참조 | 검사 내용 |
|---|---|---|
| `domain-group.code-unique` | | 그룹 코드는 유일해야 한다 |
| `domain-group.name-unique` | | 그룹명은 유일해야 한다 |
| `domain-group.code-format` | | 그룹 코드는 두 자리 숫자여야 한다 |
| `domain-group.has-classifications` | O | 그룹에는 분류가 하나 이상 있어야 한다 |
| `domain-group.orphan-classification` | O | 분류의 그룹코드가 그룹에 있어야 한다 |
| `domain-group.classification-code-unique` | O | 분류 코드는 유일해야 한다 |

| 규칙 | 참조 | 검사 내용 |
| --- | --- | --- |
| `domain-group.classification-name-unique` | O | 분류명은 유일해야 한다 |
| `domain-group.classification-prefix` | O | 분류코드는 그룹코드로 시작하는 네 자리여야 한다 |
| `domain-group.group-name-matches` | O | 분류에 적힌 그룹명이 그룹 사전과 일치해야 한다 |

CHAPTER 5

# 표준코드

표준코드는 정해진 값만 가질 수 있는 컬럼의 허용값 목록이다. 마스터(`MT_STDCD_MSTR`)와 상세(`MT_STDCD_DTL`)가 부모-자식으로 짝을 이룬다.

## 5.1 무엇을 코드로 만드는가

코드로 관리할 대상과 그렇지 않은 대상을 구분하는 기준이다.

### 5.1.1 현재 6종

표 5.1. 현재 등재된 표준코드 6종

| 코드 아이디 | 표준코드명 | 타입 | 코드값 |
|---|---|---|---|
| `STTS` | 상태 | CHAR(2) | 3 |
| `WORD_SE` | 단어구분 | CHAR(2) | 3 |
| `TRM_SE` | 용어구분 | CHAR(2) | 3 |
| `ENTTY_CLSF` | 엔터티구분 | CHAR(2) | 3 |
| `ATRB_TYPE` | 속성유형 | CHAR(2) | 3 |
| `DT_OCRN_CYCLE` | 자료발생주기 | CHAR(2) | 11 |

### 5.1.2 여부 코드는 만들지 않는다

! `Y` / `N` 두 값으로 구성된 코드는 표준코드로 만들지 않는다

값이 두 개뿐인 여부 성격의 코드는 코드 체계로 관리할 실익이 없다.

이유는 세 가지이다.

- 컬럼명이 `_YN`으로 끝나 의미가 이미 드러난다.
- 도메인 `여부C1`이 CHAR(1)로 타입과 길이를 고정한다.
- 값이 두 개뿐이므로 코드표를 조회할 필요가 없다.

여부 컬럼이 여럿이라고 하여 `USE_YN` 과 같은 공용 코드를 두면, `NUL_PRM_YN`·`PRVC_YN`·`BKUP_YN`·`ENCPT_YN`이 모두 같은 코드를 참조하게 된다. 이 구조는 코드표를 하나 늘리지만 실질적인 제약은 수행하지 못한다. 어느 컬럼에 어떤 값이 와야 하는지를 코드가 구분하지 못하기 때문이다.

다른 항목으로 대체할 수 있는 코드도 두지 않는다. 키구분은 기본키 순번으로 대체되므로 별도의 코드가 필요하지 않다(10장).

## 5.2 코드값 다루기

코드를 삭제할 때 남는 참조, 코드값의 상태, 자릿수를 다룬다.

### 5.2.1 코드를 삭제할 때 정리할 항목

코드 하나를 삭제하면 다음 네 항목을 함께 정리해야 한다.

1. `MT_STDCD_MSTR`의 마스터 행
2. `MT_STDCD_DTL`의 코드값 행
3. `MT_STD_TRM`의 `STDCD_ID` 참조
4. 산출물 컬럼 정의서의 `참조 코드` 항목

> ⚠ 표준용어의 참조가 가장 늦게 드러난다
>
> 산출물의 `참조 코드`는 코드를 삭제할 때 함께 눈에 띄지만, `MT_STD_TRM`의 `STDCD_ID`는 별도의 시트에 있어 누락되기 쉽다. 이 경우 표준용어가 존재하지 않는 표준코드를 참조하는 상태가 된다.
>
> 폐기 상태로 남겨 둔 코드는 더 늦게 드러난다. 폐기 항목을 제외하는 규칙 때문에 검사 대상에서 빠지므로, 참조가 남아 있어도 위반으로 보고되지 않는다. 폐기는 문제를 해결한 상태가 아니라 문제를 가린 상태일 수 있다.

`term.code-registered`가 표준용어의 `STDCD_ID`가 코드마스터에 존재하는지 검사한다. 폐기 코드도 등재되지 않은 것으로 취급하므로, 코드를 폐기만 하고 참조를 남겨 두면 이 규칙이 검출한다.

### 5.2.2 코드값에도 상태가 있다

`MT_STDCD_DTL`은 `STTS_CD`(상태코드)로 코드값의 상태를 기록한다. 사전 메타파일의 7개 시트가 모두 동일한 상태 체계를 사용하므로, 어느 시트에서든 같은 코드값이 같은 뜻을 가진다.

> ℹ 상태가 생기면 검사 대상이 바뀐다
>
> 검증 규칙은 폐기(03) 코드값을 검사에서 제외한다. 제외하지 않으면 폐기한 코드값 때문에 중복이나 자릿수 위반이 계속 검출되어 정리 작업의 효과가 결과에 드러나지 않는다.
>
> 다만 코드값이 모두 폐기되면 `code.has-values` 규칙이 "코드값이 없다"로 검출한다. 이는 의도한 동작이다. 사용할 수 있는 값이 하나도 없는 코드는 코드로서 기능하지 못한다.

### 5.2.3 코드값 자릿수

`code.value-digits` 규칙은 타입에 따라 다르게 판정한다.

표 5.2. 타입별 코드값 자릿수 판정

| 타입 | 판정 |
|---|---|
| CHAR | 고정 길이. 자릿수와 정확히 일치해야 한다 |
| VARCHAR | 최대 길이. 초과하지 않으면 된다 |

## 5.3 표준코드 검증 규칙

표 5.3. 표준코드 검증 규칙 9종

| 규칙 | 참조 | 검사 내용 |
|---|---|---|
| `code.id-unique` | | 코드 아이디는 유일해야 한다 |
| `code.name-unique` | | 표준코드명은 유일해야 한다 |
| `code.id-format` | | 코드 아이디는 `[A-Z][A-Z0-9_]*` 형식이어야 한다 |
| `code.has-values` | O | 마스터에는 코드값이 하나 이상 있어야 한다 |
| `code.orphan-value` | O | 코드값의 코드 아이디가 마스터에 있어야 한다 |
| `code.value-unique` | O | 같은 코드 안에서 코드값은 유일해야 한다 |
| `code.sort-unique` | O | 같은 코드 안에서 정렬순서는 유일해야 한다 |
| `code.value-digits` | O | 코드값의 길이가 코드자릿수와 맞아야 한다 |
| `code.default-value` | O | 기본값은 등록된 코드값 가운데 하나여야 한다 |

코드 아이디는 컬럼 정의서의 `참조 코드`로 그대로 사용되므로 SQL 식별자 규칙을 따른다.

# II

# 2부 · 데이터 모델링


6 데이터 모델링 개요 . . . . . 53
7 논리 데이터 모델링 . . . . . 59
8 물리 데이터 모델링 . . . . . 65
9 산출물 작성 . . . . . 71
10 기본키와 외래키 . . . . . 79

CHAPTER 6

# 데이터 모델링 개요

데이터 모델링은 업무가 다루는 대상을 두 단계로 기술하는 과정이다. 먼저 무엇을 관리할 것인지를 엔터티와 속성으로 정의하고, 다음으로 그것을 어느 DBMS에 어떤 테이블과 컬럼으로 구현할 것인지를 정한다. 앞이 논리 데이터 모델링이고 뒤가 물리 데이터 모델링이다.

1부에서 정의한 네 개의 사전은 이 과정에서 사용된다. 모델링은 이름을 새로 만드는 작업이 아니라 사전에 등재된 이름을 선택하는 작업이다. 1부와 2부가 이어져 있는 지점이 여기이다.

## 6.1 두 단계로 나누는 이유

한 번에 테이블을 설계하면 업무의 구조와 구현의 사정이 한 문서에 섞인다. 「고객과 주문이 어떤 관계인가」와 「이 컬럼을 `VARCHAR(40)`으로 할 것인가」는 판단의 근거가 서로 다르다. 앞의 것은 업무가 정하고 뒤의 것은 DBMS와 운영 조건이 정한다.

두 단계로 나누면 두 가지를 얻는다.

**업무 구조를 DBMS와 무관하게 확정할 수 있다**
: 논리 모델은 어느 DBMS를 사용할지 정하지 않은 상태에서도 작성할 수 있다. DBMS를 교체하여도 논리 모델은 그대로 유지된다.

**성능을 위한 조정이 업무 구조를 훼손하지 않는다**
: 두 엔터티를 하나의 테이블로 통합하거나 하나의 엔터티를 여러 테이블로 나누는 판단은 물리 단계에서 이루어진다. 논리 모델에는 원래의 구조가 남아 있으므로, 왜 그렇게 통합하였는지를 나중에 확인할 수 있다.

! 물리 모델만 남기면 근거가 사라진다

물리 모델만 작성하면 문서는 하나로 줄어들지만, 테이블이 지금의 모습인 이유를 기록할 곳이 없어진다. 통합한 테이블과 처음부터 하나였던 테이블을 구별할 수 없고, 성능을 위한 비정규화와 설계 오류도 구별할 수 없다.

두 문서를 유지하는 비용은 대조로 감당한다. 같은 사실을 두 곳에 기재하면 어긋나므로, `kdsm validate` 가 두 문서를 항목별로 대조한다(13.3.8절).

## 6.2 두 모델의 대응 관계

논리 모델의 각 요소는 물리 모델의 요소로 대응된다. 대응은 이름이 비슷하다는 이유로 추정하지 않고, 산출물에 기재한 항목이 근거가 된다.

표 6.1. 논리 모델과 물리 모델의 대응

| 논리 (DM-DST-002) | 물리 (DM-DST-003) | 대응의 근거 |
|---|---|---|
| 엔터티 | 테이블 | 테이블 정의서의 `관련 엔터티` 항목 |
| 속성 | 컬럼 | 소속 테이블과 물리명 |
| 엔터티 영문명 | 테이블 물리명 | 같아야 한다 |
| 속성 영문명 | 컬럼 물리명 | 같아야 한다 |
| 표준도메인 | 데이터 타입 · 길이 | 도메인이 결정한다 |
| 기본키 순번 | 기본키 순번 | 같아야 한다 |
| 참조 엔터티 · 참조 속성 | 참조 테이블 · 참조 컬럼 | 외래키가 된다 |

두 모델이 항상 1:1인 것은 아니다. 엔터티 하나가 테이블 둘이 되거나 엔터티 둘이 테이블 하나가 되기도 한다(8.2절). 이 경우에도 `관련 엔터티` 항목에 대응 관계를 기재하므로 근거는 남는다.

관계선의 수도 양쪽이 일치하지 않는다. 값을 여러 개 기재하는 항목으로 맺어진 관계는 외래키를 설정할 수 없으므로, 논리 모델의 관계선 수가 물리 모델의 외래키 수보다 많다(1.4절).

## 6.3 사전이 모델링에서 하는 일

모델링의 각 항목은 사전의 어느 부분에서 값을 가져온다. 사전이 없으면 이름과 타입을 설계자가 임의로 정하게 되고, 그 시점부터 같은 개념에 서로 다른 이름이 생긴다.

표 6.2. 사전에서 값을 가져오는 항목

| 모델링 항목 | 값을 가져오는 곳 | 확인하는 규칙 |
|---|---|---|
| 엔터티명 | 표준용어 (용어구분 `01` · `03` ) | `entity.name-registered` |
| 속성명 | 표준용어 (용어구분 `02` · `03` ) | `attribute.name-registered` |
| 엔터티 영문명 · 속성 영문명 | 표준용어의 물리명 | `entity.english-name` · `attribute.english-name` |

| 모델링 항목 | 값을 가져오는 곳 | 확인하는 규칙 |
|---|---|---|
| 속성의 도메인 | 표준도메인 | `attribute.domain` |
| 컬럼의 타입 · 길이 | 표준도메인 | `column.data-type` · `column.length` |
| 용어의 참조 코드 | 표준코드 | `term.code-registered` |

즉 모델링에서 설계자가 새로 정하는 것은 이름과 타입이 아니다. 어떤 엔터티가 필요한지, 어떤 속성으로 구성되는지, 무엇이 식별자인지, 어느 엔터티를 참조하는지가 설계자의 판단이다. 이름과 타입은 그 판단의 결과로 사전에서 결정된다.

> ! 사전에 없는 이름을 먼저 사용하지 않는다
>
> 일정을 이유로 사전에 없는 이름을 모델에 먼저 기재하면, 이후에 그 이름을 사전에 등재할 때 이미 사용한 곳을 모두 찾아 수정해야 한다. 산출물이 늘어날수록 되돌리는 비용이 커진다.
>
> 사전에 없는 이름이 필요하면 그 자리에서 1부의 작업 순서를 수행하여 후보로 등록한다(1.5절). 등재 검토가 진행되는 동안에도 후보 상태로 모델링을 이어 갈 수 있다.

## 6.4 작업 순서

읽는 순서와 일하는 순서는 다르다. 실제 작업은 다음 절차를 따른다. 왼쪽 여백의 세로 띠가 산출물의 경계이다. 위쪽 구간이 DM-DST-002에, 아래쪽 구간이 DM-DST-003에 기재된다.

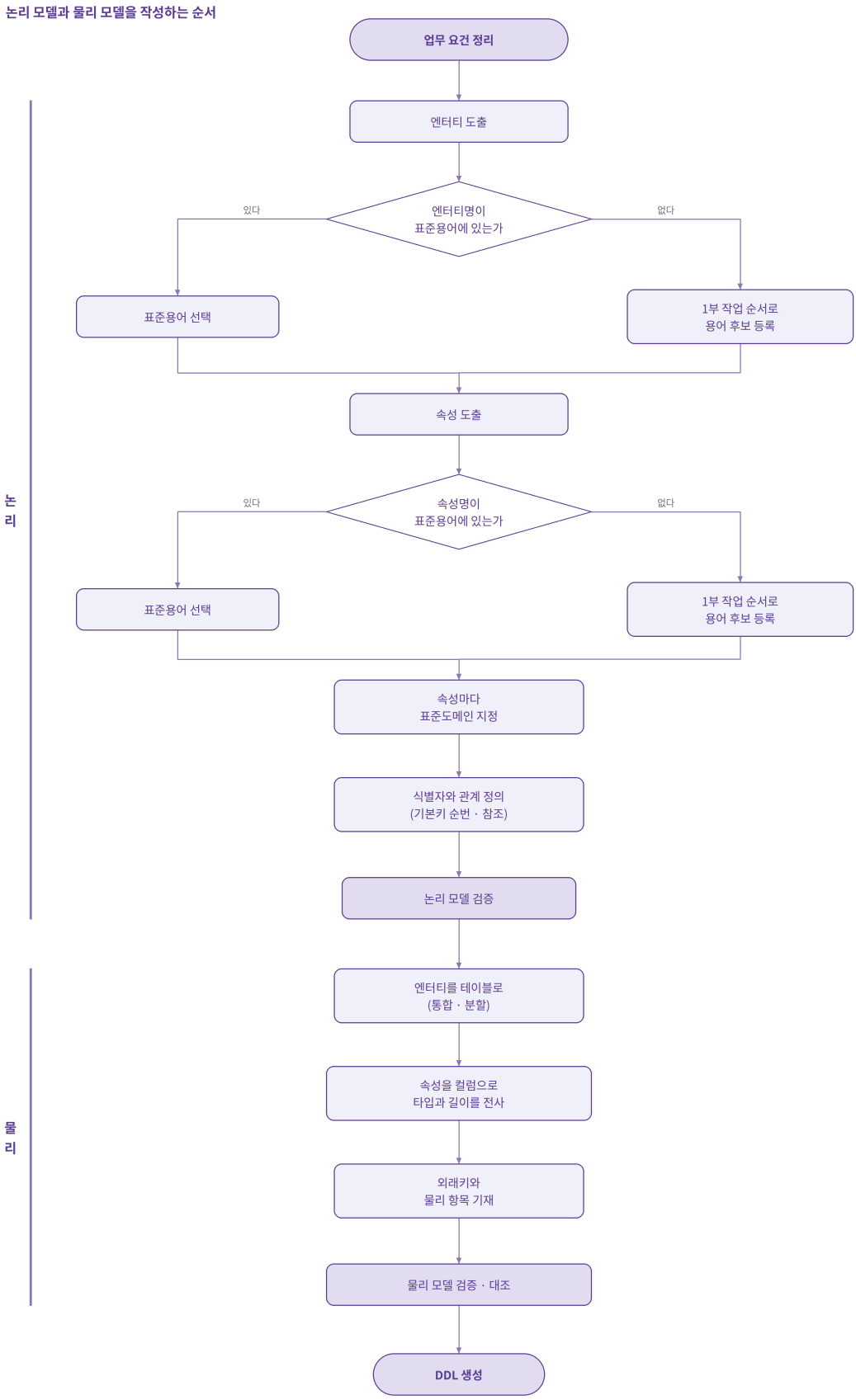


**그림 6.1.** 논리 모델과 물리 모델을 작성하는 순서

이 순서도에는 분기가 두 곳이다. 둘 다 「이 이름이 표준용어에 있는가」이며, 없으면 1부의 작업 순서로 넘어가 후보를 등록한 뒤 되돌아온다. 나머지는 한 방향으로 진행한다.

표 6.3. 모델링 단계와 산출물

| 단계 | 하는 일 | 결과가 기재되는 곳 |
|---|---|---|
| 엔터티 도출 | 업무가 관리하는 대상을 식별한다 | 엔터티 정의서 |
| 속성 도출 | 각 엔터티가 가지는 값을 식별한다 | 속성 정의서 |
| 표준도메인 지정 | 속성마다 도메인을 선택한다 | 속성 정의서 |
| 식별자와 관계 정의 | 기본키 순번과 참조를 기재한다 | 속성 정의서 |
| 논리 모델 검증 | 16개 규칙으로 검사한다 | — |
| 테이블 도출 | 엔터티를 테이블로 변환한다. 통합·분할을 결정한다 | 테이블 정의서 |
| 컬럼 도출 | 속성을 컬럼으로 변환하고 타입과 길이를 전사한다 | 컬럼 정의서 |
| 외래키와 물리 항목 | 참조와 운영 정보를 기재한다 | 테이블·컬럼 정의서 |
| 물리 모델 검증과 대조 | 물리 모델을 검사하고 논리 모델과 대조한다 | — |

검증은 마지막에 한 번만 실행하는 것이 아니다

논리 모델 검증을 통과한 뒤에 물리 모델링을 시작한다. 논리 모델에 남은 문제를 물리 모델까지 옮겨 적으면 고쳐야 할 곳이 두 배가 된다.

작성 도중에도 실행할 수 있다. 검증은 파일만 읽으므로 데이터베이스 접속이 필요하지 않다(13장).

```
kdsm validate DM-DST-002_*.xlsx --meta <사전메타>
```

## 6.5 이 부의 구성

표 6.4. 2부의 구성

| 장 | 다루는 내용 |
|---|---|
| 7장 | 엔터티와 속성을 정의하는 방법 |
| 8장 | 테이블과 컬럼으로 변환하는 방법 |
| 9장 | DM-DST 산출물 3종의 서식과 작성 규칙 |
| 10장 | 기본키와 외래키를 기재하여 식별관계까지 표현하는 방법 |

앞의 두 장이 무엇을 정하는가를 다루고, 뒤의 두 장이 그것을 어디에 어떻게 기재하는가를 다룬다. 산출물의 서식을 먼저 확인하려면 9장 부터 읽어도 무방하다.

기본키와 외래키는 별도의 장에서 설명한다. 두 항목에 나누어 기재해야 식별관계가 표현되는데, 이 부분이 산출물에서 가장 오해가 잦기 때문이다.

CHAPTER 7

# 논리 데이터 모델링

논리 데이터 모델링은 업무가 다루는 대상을 엔터티와 속성으로 정의하는 과정이다. 이 단계의 결과물은 아직 테이블과 컬럼이 아니다. 어떤 DBMS를 사용할지, 데이터 타입의 길이를 얼마로 지정할지는 다음 단계에서 결정한다. 두 단계로 나누는 이유는 6.1절 에 있다.

이 단계에서는 두 개의 시트를 작성한다.

표 7.1. 논리 모델링 산출물

| 시트 | 수록 내용 | 물리 테이블 |
|---|---|---|
| 엔터티 정의서 | 어떤 엔터티가 존재하는가 | `MT_ENTTY` |
| 속성 정의서 | 각 엔터티가 어떤 값을 가지는가 | `MT_ATRB` |

이 두 시트는 DM-DST-002 「논리 데이터 모델」 산출물에 들어 있다. 빈 템플릿은 `kdsm` 패키지 안에 들어 있다(9장).

## 7.1 이름은 표준용어에서 선택한다

엔터티명과 속성명은 새로 만드는 것이 아니라 표준용어 사전에서 선택하는 것이다. 1부에서 정의한 결과가 이 단계로 이어진다(6.3절).

표준용어에는 용어구분이 있어 어느 위치에 사용할 수 있는지가 정해져 있다.

표 7.2. 용어구분과 사용 위치

| 용어구분 | 사용 위치 |
|---|---|
| `01` 엔터티명 | 엔터티 이름에 사용한다 |

| 용어구분 | 사용 위치 |
|---|---|
| `02` 속성명 | 속성 이름에 사용한다 |
| `03` 엔터티명/속성명 | 두 위치 모두에 사용한다 |

`entity.name-registered`와 `attribute.name-registered` 규칙이 이를 검사한다. 사전에 없는 이름을 사용하면 위반으로 보고되며, 속성명으로만 등재된 용어를 엔터티 이름에 사용한 경우도 마찬가지이다.

> ! 사전에 없으면 모델링을 중단하고 사전을 먼저 정비한다
>
> 일정을 이유로 사전에 없는 이름을 먼저 사용하면, 나중에 그 이름을 사전에 등재할 때 이미 사용한 곳을 모두 찾아 수정해야 한다. 순서를 지키는 것이 결과적으로 더 빠르다(1.5절).

## 7.2 엔터티 정의서

한 행이 엔터티 하나에 해당한다.

표 7.3. 엔터티 정의서의 항목

| 항목 | 기재 내용 |
|---|---|
| 엔터티명 | 표준용어(엔터티명). 한글 |
| 영문명 | 물리 단계의 테이블명이 될 이름 |
| 엔터티 설명 | 이 엔터티가 무엇인지에 대한 설명 |
| 식별자(PK) | 한 행을 식별하는 속성명. 복합키는 `+`로 연결한다 |
| 엔터티 구분 | 기본 · 중심 · 행위 |
| 상위 엔터티 | 이 엔터티가 소속되는 상위 엔터티 |
| 개인정보 포함여부 | 속성 중 하나라도 개인정보이면 `Y` |
| 주요 관계 | `표준용어(1:N)` 형식으로 기록하는 상대 엔터티와 카디널리티 |

### 7.2.1 엔터티 구분

세 가지로 구분한다. 다른 엔터티에 의존하지 않고 독립적으로 존재하는지, 그리고 행위의 기록인지를 기준으로 판단한다.

표 7.4. 엔터티 구분

| 코드 | 구분 | 의미 |
|---|---|---|
| `01` | 기본엔터티 | 다른 엔터티 없이도 존재한다. 사전·마스터 성격을 가진다 |
| `02` | 중심엔터티 | 기본엔터티에서 파생되며 업무의 중심이 된다 |
| `03` | 행위엔터티 | 둘 이상의 엔터티 사이에서 발생한 행위를 기록한다 |

KDSM 메타 모델의 엔터티 열두 개는 모두 기본엔터티이다. 사전을 담는 모델이기 때문이다. 업무 모델을 작성하면 중심엔터티와 행위엔터티의 비중이 훨씬 커진다.

### 7.2.2 식별자 작성

식별자는 물리명이 아니라 한글 속성명으로 기록한다. 복합키는 순서에도 의미가 있으므로 정해진 순서대로 기록한다.

```
표준코드상세 컬럼    표준코드아이디 + 표준코드값내용
                    테이블물리이름 + 컬럼물리이름
```

이 순서가 인덱스의 컬럼 순서를 결정하므로 임의로 기록해서는 안 된다. 조회 조건으로 자주 사용하는 속성을 앞에 둔다.

## 7.3 속성 정의서

한 행이 속성 하나에 해당한다. 어느 엔터티에 속하는지를 첫 항목에 기록한다.

표 7.5. 속성 정의서의 항목

| 항목 | 기재 내용 |
|---|---|
| 소속 엔터티 | 엔터티 정의서에 있는 이름 |
| 속성명 | 표준용어(속성명). 한글 |
| 영문명 | 물리 단계의 컬럼명이 될 이름 |
| 속성 유형 | 기본형 · 설계형 · 추출형 |
| 도메인 | 표준도메인 이름. 타입과 길이가 이 값에서 결정된다 |
| 기본키 순번 | 식별자를 구성하면 1부터 부여하고, 그렇지 않으면 비운다 |
| Null 허용여부 | `Y` / `N` |
| 개인정보 여부 | `Y` / `N` |
| 참조 엔터티·참조 속성이름 | 다른 엔터티를 참조하는 경우에 기록한다 |
| 기본값 | 기본값이 있는 경우에 기록한다 |
| 속성 설명 | 이 속성이 무엇인지에 대한 설명 |

### 7.3.1 속성 유형

값의 출처를 기준으로 구분한다.

표 7.6. 속성 유형

| 코드 | 유형 | 의미 |
|---|---|---|
| `01` | 기본형 | 업무에서 그대로 입력되는 값 |
| `02` | 설계형 | 관리를 위해 설계자가 추가한 값. 일련번호와 상태코드가 이에 해당한다 |

| 코드 | 유형 | 의미 |
|---|---|---|
| `03` | 추출형 | 다른 값에서 계산하여 도출한 값 |

추출형을 도입하면 언제 다시 계산할 것인가라는 문제가 함께 발생한다. 원본이 변경되었는데 추출값이 갱신되지 않으면 두 값이 서로 다른 사실을 나타내게 된다. 따라서 반드시 필요한 경우에만 사용한다.

## 7.4 관계 작성

관계를 기록하는 항목은 두 개이며, 각각의 용도가 다르다.

**표 7.7.** 관계를 기록하는 두 항목

| 항목 | 수록 위치 | 용도 |
|---|---|---|
| 참조 엔터티·참조 속성이름 | 속성 정의서 | 구조를 기록한다. 검증과 ERD가 이 값을 읽는다 |
| 주요 관계 | 엔터티 정의서 | 엔터티를 설명할 때 사람이 읽는 용도이다 |

앞의 항목이 실제 근거가 된다. 어느 속성이 어느 엔터티의 어떤 속성을 참조하는지가 한 행에 모두 기록되어 있으므로, 프로그램이 이를 그대로 읽어 ERD를 생성한다(16.6.1절).

> **i 두 항목은 서로를 포괄하지 못한다**
>
> 관계선은 두 항목에서 도출되며 어느 한쪽만 읽으면 나머지를 누락하게 된다. 대부분은 참조 항목에서 도출되지만, 값을 여러 개 기록하는 항목으로 맺어진 관계는 참조를 설정할 수 없어 주요 관계에만 남는다(1.4절). 관계가 존재하지 않는 것이 아니라 해당 항목이 이를 표현하지 못하는 것이다.
>
> 그러므로 관계선을 세어 확인할 때에는 두 항목의 합계로 센다. 참조 항목만 세면 실제보다 적게 집계된다.

### 7.4.1 식별관계와 비식별관계

부모 엔터티의 키가 자식 엔터티의 키에 포함되면 식별관계이다. 이를 별도로 표시하는 항목은 없으며, 기본키 순번이 이를 나타낸다.

```
속성.엔터티이름     기본키 순번 1   →  엔터티의 키가 속성의 키에 들어간다 = 식별관계
속성.도메인이름     기본키 순번 없음 →  가리키기만 한다 = 비식별관계
```

`상위 엔터티` 항목이 있으나 이를 식별관계의 판정 근거로 사용하지 않는다. 이 항목은 소속을 나타내는 뜻으로 쓰이는 경우가 많아 기본키 구성과 일치하지 않을 수 있다. 식별관계는 기본키가 정한다(16.6.2절).

## 7.5 검증

논리 모델은 엔터티 규칙과 속성 규칙으로 검사한다. 같은 산출물에 작성한 주제영역과 배정은 주제영역 규칙과 배정 규칙으로 검사한다.

```
kdsm validate DM-DST-002_*.xlsx                # 산출물을 검사한다
kdsm validate --meta <사전메타> --logical <논리메타>   # 메타파일을 검사한다
```

산출물을 지정하는 방식을 우선한다. `--logical` 은 산출물 없이 메타파일만 있는 경우에 사용한다. 어느 쪽으로 지정하든 같은 규칙을 실행한다. 나머지 옵션은 표 13.8 에 있다.

이 인자를 지정하지 않으면 이 규칙들은 실행되지 않는다. 검사하지 않은 항목을 통과한 것으로 보고하지 않기 위해서이다(13장).

표 7.8. 논리 모델 검증 항목

| 검사 항목 | 규칙 |
|---|---|
| 이름이 표준용어인가 | `entity.name-registered` · `attribute.name-registered` |
| 영문명이 표준용어 물리명인가 | `entity.english-name` · `attribute.english-name` |
| 식별자가 실재하는 속성인가 | `entity.key-registered` |
| 상위 엔터티가 실재하는가 | `entity.parent-defined` |
| 주요 관계가 양쪽에서 일치하는가 | `entity.relation-paired` |
| 도메인이 사전에 있는가 | `attribute.domain` |
| 참조가 실재하는 곳을 가리키는가 | `attribute.reference-defined` |
| 기본키 순번이 식별자와 일치하는가 | `attribute.primary-key` |

전체 목록은 `library/docs/validation-rules.md`에 있다.

## 7.6 다음 단계

논리 모델이 완성되면 물리 모델링으로 진행한다(8장). 엔터티가 테이블이 되고 속성이 컬럼이 되며, 도메인이 데이터 타입과 길이로 변환된다.

CHAPTER 8

# 물리 데이터 모델링

물리 데이터 모델링은 논리 모델을 실제 DBMS에 적용하는 과정이다. 엔터티가 테이블이 되고 속성이 컬럼이 되며, 도메인이 데이터 타입과 길이로 변환된다. 두 모델의 대응 관계는 6.2절 에 정리되어 있다.

이 단계에서는 세 개의 시트를 작성한다.

표 8.1. 물리 모델링 산출물

| 시트 | 수록 내용 | 물리 테이블 |
|---|---|---|
| 데이터베이스 정의서 | 어디에 배치하는가 | `MT_DB` |
| 테이블 정의서 | 어떤 테이블을 만드는가 | `MT_TBL` |
| 컬럼 정의서 | 어떤 값을 담는가 | `MT_COL` |

이 세 시트는 DM-DST-003 「물리 데이터 모델」 산출물에 들어 있다. 논리 모델과 짝을 이루므로 두 산출물을 서로 대조할 수 있다(13.3.8절).

표 8.2. 짝을 이루는 시트

| 논리 모델 (DM-DST-002) | 물리 모델 (DM-DST-003) |
|---|---|
| 엔터티 정의서 | 테이블 정의서 |
| 속성 정의서 | 컬럼 정의서 |

## 8.1 논리에서 물리로

변환 규칙은 네 가지이다.

표 8.3. 논리에서 물리로

| 논리 | 물리 | 연결 근거 |
|---|---|---|
| 엔터티 | 테이블 | 테이블 정의서의 `관련 엔터티` |
| 엔터티 영문명 | 테이블 물리명 | 같아야 한다 |
| 속성 영문명 | 컬럼 물리명 | 같아야 한다 |
| 도메인 | 데이터 타입·길이 | 도메인이 결정한다 |

두 문서를 연결하는 유일한 근거는 `관련 엔터티` 항목이다. 이름이 비슷하다는 이유로 대응 관계를 추정하지 않는다. 추정으로는 규칙을 위반한 경우와 의도적으로 다르게 명명한 경우를 구별할 수 없기 때문이다.

> ! 접두어는 엔터티 영문명에 포함한다
>
> 이 저장소의 테이블은 모두 `MT_`로 시작한다. 메타 테이블(Meta Table)이라는 뜻이다.
>
> 접두어는 물리 단계에서 부여하지 않고 엔터티 영문명에 포함하는 편이 안전하다. 물리 단계에서 부여하면, 접두어를 붙일 엔터티와 붙이지 않을 엔터티가 함께 있을 때 어느 쪽에 부여할지를 규칙이 판단할 수 없다.
>
> `pairing.table-name` 규칙은 두 이름이 같은지만 확인한다. 접두어를 물리 단계에서 부여하는 환경이라면 `--name-prefix MT_` 로 지정한다.

## 8.2 엔터티와 테이블은 1:1이 아니다

변환 과정에서 엔터티 하나가 테이블 두 개가 되거나, 엔터티 두 개가 테이블 하나가 되기도 한다.

성능을 위해 두 엔터티를 하나의 테이블로 통합하기도 하고, 이력 분리나 파티셔닝을 위해 하나의 엔터티를 여러 테이블로 분할하기도 한다. 통합하는 경우에는 `관련 엔터티`에 여러 이름을 쉼표로 구분하여 기록한다.

```
관련 엔터티    표준단어, 표준용어
```

이 경우 검사 방식도 달라진다. 등재 여부를 확인하는 규칙은 이름마다 각각 검사하고, 이름을 1:1로 비교하는 규칙은 판정하지 않는다. 두 엔터티를 통합한 테이블에는 비교할 기준 이름이 존재하지 않기 때문이다(9.4.3절).

## 8.3 물리 단계에서 추가되는 항목

논리 모델에는 없고 물리 모델에만 있는 항목이다. 운영에 필요한 정보를 기록한다.

표 8.4. 물리 단계에서 추가되는 항목

| 항목 | 결정 내용 |
|---|---|
| 예상 건수 | 용량 산정과 인덱스 전략의 근거가 된다 |
| 보관 기간 | 데이터를 언제까지 보관하고 언제 삭제하는가 |

| 항목 | 결정 내용 |
|---|---|
| 발생주기 | 실시간 · 일 · 주 · 월 · 분기 · 년 등의 주기 |
| 백업 여부 | 백업 대상에 포함되는가 |
| 테이블 유형·소유자 | 테이블의 성격과 운영 주체 |

컬럼 정의서에도 논리 모델에 없던 항목이 추가된다. 암호화 여부가 이에 해당한다. 개인정보 여부는 논리 모델에서 이미 정의하고, 이를 실제로 암호화할지는 물리 모델에서 결정한다.

## 8.4 타입과 길이는 도메인이 정한다

컬럼 정의서의 데이터 타입과 길이는 설계자가 임의로 결정하지 않는다. 속성이 참조하는 표준도메인에 이미 정의된 값을 옮겨 기록한다.




그림 8.1. 타입과 길이는 속성이 참조하는 도메인이 정한다

옮겨 기록하는 과정에서 불일치가 발생할 수 있다. 이를 위해 `column.data-type`과 `column.length` 규칙이 도메인에 정의된 값과 컬럼에 기록된 값을 대조한다.

> ⚠ 길이를 여유 있게 지정하려는 경우
>
> 여유를 이유로 `VARCHAR(4000)`과 같이 지정하면 표준이 무력화된다. 길이가 도메인에서 결정되지 않으면 도메인을 정의한 이유가 사라진다.
>
> 실제로 더 긴 길이가 필요하다면 도메인을 새로 정의하여 사전에 등재하는 것이 올바른 순서이다. 그래야 다른 설계자도 같은 도메인을 사용할 수 있다.

## 8.5 외래키 작성

컬럼 정의서의 `참조 테이블`·`참조 컬럼` 항목이 외래키가 된다. DDL 생성기가 이 값을 읽어 `FOREIGN KEY` 절을 생성한다.

현재 모델에서는 열두 개의 외래키가 생성된다.

```
MT_ATRB.ENTTY_NM     → MT_ENTTY.ENTTY_NM
MT_TBL.DB_NM,SCHM_NM → MT_DB.DB_NM,SCHM_NM
MT_COL.TBL_PHYS_NM   → MT_TBL.TBL_PHYS_NM
...
```

외래키를 어떻게 기록해야 식별관계까지 표현되는지는 별도의 장에서 다룬다(10장).

> **i 모든 논리 관계가 외래키로 구현되지는 않는다**
>
> 값을 여러 개 기록하는 항목으로 맺어진 관계는 외래키를 설정할 수 없다(1.4절). 따라서 논리 모델의 관계선 수보다 물리 모델의 외래키 수가 적은 것이 정상이다. 두 수가 같아야 한다고 보면 존재하지 않는 누락을 찾게 된다.
>
> 외래키로 구현하지 못한 관계는 데이터베이스가 보장하지 못하므로 검증 규칙이 그 역할을 대신한다. 두 ERD를 나란히 비교하면 어느 관계가 여기에 해당하는지 확인할 수 있다(16.6.5절).

## 8.6 검증과 대조

물리 모델은 두 가지 관점에서 검사한다. 물리 모델 내부의 정합성과, 논리 모델과의 일치 여부이다.

```
kdsm validate DM-DST-002_*.xlsx DM-DST-003_*.xlsx --name-prefix AP_
kdsm validate --meta <사전메타> --logical <논리메타> -P <물리메타> --name-prefix MT_
```

윗줄은 배포 산출물을, 아랫줄은 KDSM 자신을 구축할 때 쓰는 메타파일을 검사한다. `-P` / `--physical`은 물리 모델 메타파일을, `--logical`은 논리 모델 메타파일을 지정한다. 물리 모델 내부의 정합성은 `--physical`만으로 검사하지만, 논리 모델과의 대조에는 둘이 모두 필요하다. 나머지 옵션은 표 13.8 에 있다.

표 8.5. 물리 모델 검증 항목

| 검사 항목 | 규칙 |
|---|---|
| 테이블에 컬럼이 존재하는가 | `table.has-columns` · `column.table-defined` |
| 기본키 표기가 컬럼과 일치하는가 | `table.primary-key` |
| 유일키 표기가 컬럼과 일치하는가 | `table.unique-key` |
| 데이터베이스가 정의되어 있는가 | `table.database-defined` |
| 관련 엔터티가 논리 모델에 등재되어 있는가 | `table.entity-registered` |
| 컬럼 물리명이 표준용어에 등재되어 있는가 | `column.term-registered` · `column.term-name` |
| 도메인이 사전에 등재되어 있는가 | `column.domain-registered` |
| 타입·길이가 도메인과 일치하는가 | `column.data-type` · `column.length` |
| 외래키가 존재하는 대상을 참조하는가 | `column.foreign-key-target` |

논리 모델과의 대조 규칙은 열여섯 개이다. 테이블 여섯 개와 컬럼 열 개이며, 이름·한글명·기본키·유일키·Null 허용·개인정보·기본값·참조·설명을 항목별로 비교한다(13.3.8절).

> **두 문서를 함께 수정하고 대조한다**
>
> 속성의 정의를 수정하면 대응하는 컬럼도 함께 수정한다. 한쪽만 수정하여도 파일을 읽는 데에는 문제가 없으므로 적재 시점까지 드러나지 않는다. Null 허용 여부가 특히 어긋나기 쉽다.
>
> 같은 사실을 두 문서에 기록하는 한 이러한 불일치는 계속 발생하므로, 수정한 뒤에는 반드시 대조를 실행한다(13.3.8절).

## 8.7 다음 단계

물리 모델이 완성되면 DDL을 생성하여 스키마를 만든다(14장). 이어서 메타파일을 적재하면 문서가 아니라 데이터베이스가 기준이 된다(15장).

CHAPTER 9

# 산출물 작성

사용자가 작성하는 산출물은 3종이다. 표준화 작업의 세 단계가 각각 한 권에 대응한다.

표 9.1. 사용자가 작성하는 산출물

| 문서번호 | 산출물 | 작성하는 작업 |
|---|---|---|
| DM-DST-001 | 데이터 표준 정의 | 데이터 표준 작업 |
| DM-DST-002 | 논리 데이터 모델 | 논리 데이터 모델링 |
| DM-DST-003 | 물리 데이터 모델 | 물리 데이터 모델링 |

이와 별도로 DM-LIB 6종이 있다. KDSM 메타테이블에 적재하는 초기 데이터를 정의한 문서이며 시스템 관리자 영역의 산출물이다. 사용자가 작성하거나 개정할 대상이 아니다. 두 계열의 차이는 다음 절에서 다룬다.

## 9.1 DM-LIB과 DM-DST

문서번호는 두 계열로 나뉜다. 구분 기준은 누가 무엇을 위하여 작성하는가이다.

표 9.2. 두 산출물 계열

| 계열 | 수록 내용 | 권수 | 작성 영역 |
|---|---|---|---|
| DM-DST | 표준·논리·물리를 각각 한 권에 담은 산출물 | 3 | 사용자 |
| DM-LIB | 대상 영역을 논리·물리로 나눈 정의서 | 6 | KDSM 시스템 관리자 |

DM-DST는 사용자가 KDSM으로 표준화를 수행하면서 작성하는 산출물이다. 영역으로 구분하지 않고 표준 전체·논리 모델 전체·물리 모델 전체를 각각 한 권에 수록한다. 빈 템플릿에서 시작하여 사람이 작성한다. 이 매뉴얼의 작성법과 명령 예시는 모두 이 계열을 대상으로 한다.

표 9.3. DM-DST 산출물

| 문서번호 | 산출물 | 시트 |
|---|---|---|
| DM-DST-001 | 데이터 표준 정의 | 단어·용어·도메인·도메인 그룹·도메인 분류·코드 마스터·코드 값 상세 |
| DM-DST-002 | 논리 데이터 모델 | 엔터티·속성 |
| DM-DST-003 | 물리 데이터 모델 | 데이터베이스·테이블·컬럼 |

빈 템플릿은 `kdsm` 패키지 안(`library/src/kdsm/templates`)에 들어 있다. 저장소를 내려받지 않고 패키지만 설치한 사람도 `kdsm export`로 서식을 얻을 수 있게 하기 위해서이다. 항목 이름이 DM-LIB과 같으므로 동일한 리더가 읽는다. 다만 `참조 속성` 항목의 이름이 DM-LIB에서는 `참조 속성이름`, DM-DST-002에서는 `참조 속성명`이다. 두 계열을 함께 다룰 때에는 이 한 항목만 표기가 다르다는 점을 확인한다.

> **i** `kdsm export`가 DM-DST를 대신 쓰는 것은 아니다
>
> `kdsm export`는 적재된 메타테이블을 읽어 DM-DST 형식의 문서를 만든다(17장). 사람의 작성을 대신하는 도구가 아니라 두 경우에 쓴다.
>
> - 산출물 없이 구축한 데이터베이스에서 산출물을 얻을 때. 표준화를 거치지 않고 만든 데이터베이스를 인수한 경우이며, 백지에서 시작하는 대신 현재 상태를 문서로 뽑아 두고 거기에서 손을 댄다
> - 산출물의 판을 데이터베이스에 맞출 때. 운영하면서 테이블이 바뀌면 문서와 데이터베이스가 어긋나는데, 어긋난 자리를 손으로 찾아 고치는 대신 다시 생성한다
>
> 어느 쪽이든 생성한 문서는 버전 `0.0.0`, 작성자 공란으로 나온다. 사람이 검토하기 전임을 나타내는 표시이다.

DM-LIB은 KDSM 메타테이블에 적재하는 초기 데이터를 정의한 문서이다. KDSM 자신의 데이터 표준·논리 데이터 모델·물리 데이터 모델을 기록한 것이며, 대상 영역을 논리와 물리로 한 번 더 구분하여 3 × 2 = 6종이다.

표 9.4. DM-LIB 산출물 구성

| 대상 영역 | 논리 데이터 모델링 | 물리 데이터 모델링 |
|---|---|---|
| 데이터 표준 | DM-LIB-001 | DM-LIB-004 |
| 논리 데이터 모델 | DM-LIB-002 | DM-LIB-005 |
| 물리 데이터 모델 | DM-LIB-003 | DM-LIB-006 |

> **i 메타데이터 시스템 자체를 모델링한 결과**
>
> DM-LIB 6종은 데이터 표준과 데이터 모델을 관리하는 시스템 자체를 모델링한 결과이다. 표준을 관리하는 시스템도 그 표준을 준수해야 한다는 것이 전제이다. 따라서 메타파일의 컬럼명도 표준 용어 물리명이어야 하며, `meta_header_rules`가 이러한 자기 참조를 검사한다.
>
> 시스템 관리자가 KDSM을 구축하거나 개정할 때에만 다루는 문서이므로, 사용자는 이 계열을 작성하지 않는다.

## 9.2 파일명 규칙

파일명은 `문서번호_{산출물명}-{대상}_V{버전}_{작성일}.xlsx` 형식이다.

```
DM-DST-001_데이터표준정의-KDSM-APP_V0.9.0_20260818.xlsx
└문서번호┘└──산출물명──┘└대상┘└버전┘└작성일┘
```

DM-LIB은 대상 자리에 모델링 단계와 대상 영역을 함께 기재한다. 한 영역을 논리와 물리로 나누어 작성하므로 어느 단계의 문서인지를 파일명에서 구별해야 하기 때문이다.

```
DM-LIB-001_KDSM논리데이터모델-데이터표준정의서_V{버전}_{작성일}.xlsx
└문서번호┘└─────모델링단계-대상영역─────────┘└버전┘└작성일┘
```

작성일은 `YYYYMMDD` 여덟 자리로 표기한다. 구분자를 사용하지 않는 것은 파일명에서 `_`가 이미 구성 요소를 구분하는 기호이므로, 날짜 안의 `-`와 역할이 중복되기 때문이다.

> **! `V`는 파일명에만 붙인다**
>
> 파일명에는 `V0.9.5`로, 표지와 양식 시트의 버전 항목에는 `0.9.5`로 기록한다.
>
> 파일명에서는 앞뒤 구성 요소와 구별하는 표시로서 의미가 있지만, 셀 안에서는 값 앞에 붙은 문자에 지나지 않는다. 표기가 일정하지 않으면 같은 버전을 가리키는 `0.9.5`와 `V0.9.5`가 문자열로는 다른 값이 되어, 파일명과 표지를 대조할 때 불일치로 판정된다.

> **⚠ 한글 파일명은 NFC로**
>
> macOS 파일 시스템은 한글 파일명을 자모가 분리된 NFD로 반환하는 경우가 있다. `한`이 `ㅎ`+`ㅏ`+`ㄴ` 세 글자로 저장된 형태이다.
>
> 화면에는 동일하게 보이지만 문자열로는 다른 값이므로, 검색·비교·중복 확인에서 오류가 발생한다. 육안으로는 발견할 수 없는 종류의 오류이다. 프로그램이 생성하는 파일명은 항상 NFC로 표기한다(17장).

## 9.3 양식

산출물 여섯 종의 시트 구성과 각 항목을 읽는 방식을 설명한다.

### 9.3.1 시트 구성

**표 9.5.** 모델링 단계별 시트 구성

| 단계 | 시트 |
|---|---|
| 논리 (001~003) | 표지 / 엔터티 정의서 / 속성 정의서 |
| 물리 (004~006) | 표지 / 데이터베이스 정의서 / 테이블 정의서 / 컬럼 정의서 / 별첨_작성예시 |

### 9.3.2 양식 읽는 법

양식 시트는 표지 문구, 헤더, 항목 설명, 데이터, 작성 지침이 한 시트에 혼재된 서식 문서이다. 위치가 고정되어 있지 않으므로 프로그램은 다음 순서로 데이터 영역을 탐색한다.

1. 필수 항목 표시(`*`)가 붙은 셀이 있는 행을 헤더로 인식한다
2. 그다음 한 행은 항목 설명이므로 건너뛴다
3. 첫 컬럼이 비었거나 `*`로 시작하는 행(작성 지침)을 만나면 읽기를 중단한다

> ⚠ 양식을 고칠 때 지킬 것
>
> - 헤더의 별표(`*`)를 지우지 않는다. 프로그램이 헤더를 찾지 못한다.
> - 헤더 바로 아래 연보라색 설명 행을 지우지 않는다. 첫 데이터 행이 설명으로 취급된다.
> - 데이터 중간에 빈 행을 넣지 않는다. 해당 위치에서 읽기가 종료된다.

### 9.3.3 컬럼 정의서

물리 모델링 산출물의 핵심 시트이며, 16개 항목으로 구성된다.

표 9.6. 컬럼 정의서의 16개 항목

| 항목 | 설명 |
|---|---|
| 소속 테이블 | 테이블 정의서의 테이블 물리명과 같아야 한다 |
| 컬럼 순번 | 생성 순서 |
| 컬럼 물리명 | 표준용어의 물리명 |
| 컬럼 한글명 | 표준용어명 |
| 데이터 타입 | 표준도메인의 데이터 타입 |
| 길이 | 표준도메인의 길이 |
| Null 허용 | `Y` / `N` |
| 기본키 순번 | 기본키인 경우 1부터 부여하고, 그렇지 않으면 비운다 |
| 소속 유일키 | 유일 제약에 속하는 경우 `UK<번호>-<순번>` 표기. 그렇지 않으면 비운다 |
| 기본값 | |
| 참조 테이블 | 외래키인 경우 참조 대상 테이블 |
| 참조 컬럼 | 외래키인 경우 참조 대상 컬럼 |
| 참조 코드 | 표준코드를 참조하는 경우 코드 아이디 |
| 개인정보 여부 | |

| 항목 | 설명 |
|---|---|
| 암호화 여부 | |
| 컬럼 설명 | |

기본키와 외래키를 기록하는 방법은 10장 에서 상세히 다룬다.

### 9.3.4 유일키 표기

기본키처럼 유일키도 두 층에 기록한다. 아래층이 기계가 읽는 정본이고, 위층은 사람이 읽는 선언이다.

표 9.7. 유일키의 두 층

| 층 | 시트 · 항목 | 표기 |
|---|---|---|
| 아래층 | 속성 정의서·컬럼 정의서의 「소속 유일키」 | `UK<번호>-<순번>` (예: `UK1-1`) |
| 위층 | 엔터티 정의서의 「보조 식별자(UK)」 · 테이블 정의서의 「유일키(UK)」 | `UK<번호>: 구성명 + 구성명` |

아래층 표기에서 번호가 같은 컬럼들이 하나의 UNIQUE 제약을 이루고, 순번이 제약 안의 컬럼 순서를 정한다. 대소문자는 가리지 않는다. 컬럼 순서는 제약의 의미에는 영향이 없으나 그 제약이 만드는 인덱스의 선두 컬럼 검색에는 영향이 있다.

위층은 한 줄이 제약 하나이며 셀 안 줄바꿈으로 여럿을 적는다. 구성명의 구분자는 기본키처럼 `+`와 쉼표를 모두 받는다. DDL은 아래층만 읽으며, 두 층이 어긋나면 검증 규칙이 잡는다( `attribute.unique-key` · `table.unique-key` ). 생성되는 제약 이름은 `UX_<테이블 물리명>_<번호>`이므로 표기의 번호가 DDL에 그대로 남는다(14.7.5절).

한 컬럼이 여러 유일 제약에 동시에 속하는 경우는 칸 하나에 표기 하나라 담지 못한다. 지원하지 않는다.

## 9.4 같은 사실이 여러 곳에 기록된다

산출물은 문서이므로 같은 사실을 여러 항목에 중복하여 기록하게 된다. 불일치가 발생하기 쉬운 위치는 다음과 같다.

### 9.4.1 수작업으로 옮겨 기록하는 항목

같은 사실을 여러 곳에 기록하면 불일치가 발생한다. 산출물에는 표 9.8 의 네 가지 경우가 있다.

표 9.8. 수작업으로 옮겨 기록하는 항목과 검사 규칙

| 위치 | 원본 | 검사 규칙 |
|---|---|---|
| 컬럼 물리명·한글명 | 표준용어 사전 | `column.term-registered`, `column.term-name` |
| 데이터 타입·길이 | 표준도메인 사전 | `column.data-type`, `column.length` |
| 테이블 정의서의 기본키(PK) 항목 | 컬럼 정의서의 기본키 순번 | `table.primary-key` |

| 위치 | 원본 | 검사 규칙 |
|---|---|---|
| 테이블 정의서의 유일키(UK) 항목 | 컬럼 정의서의 소속 유일키 | `table.unique-key` |

> ! 테이블 정의서의 기본키 항목은 DDL에 반영되지 않는다
>
> DDL은 컬럼 정의서의 기본키 순번만을 근거로 생성된다. 테이블 정의서의 기본키 항목이 잘못 기록되어 있어도 DDL은 정상적으로 생성되며, 문서를 읽는 사람만 잘못된 정보를 확인하게 된다. `table.primary-key` 규칙이 구성 컬럼뿐 아니라 순서까지 대조한다. 테이블 정의서의 유일키(UK) 항목도 같다. DDL은 컬럼 정의서의 소속 유일키만 읽으며 `table.unique-key`가 두 기록을 대조한다.

### 9.4.2 테이블 이름은 두 시트에 기록된다

테이블 정의서의 `테이블 물리명`과 컬럼 정의서의 `소속 테이블`이 같아야 한다.

> ⚠ 한쪽만 수정한 경우의 증상
>
> 표 9.9. 테이블 이름의 불일치 위치에 따른 증상

| 불일치 위치 | 증상 | 규칙 |
|---|---|---|
| 테이블 정의서 | 빈 테이블이 되어 DDL 생성이 중단된다 | `table.has-columns` |
| 컬럼 정의서 | 해당 컬럼이 어느 테이블에도 속하지 못하여 안내 없이 누락된다 | `column.table-defined` |

> 후자가 훨씬 위험하다. 오류 없이 컬럼이 누락된 DDL이 생성되므로 생성 결과만으로는 확인할 수 없다. 따라서 두 방향을 모두 검사한다.
>
> 테이블 이름은 메타파일의 시트명까지 포함하면 세 곳에 기록된다. 접두어를 붙이거나 낱말을 추가하는 개정에서 세 곳이 갈리기 쉬우므로, 테이블 이름을 변경할 때에는 세 곳을 함께 변경한다 (22.2.5절).

### 9.4.3 관련 엔터티

테이블 정의서의 `관련 엔터티`는 이 테이블이 어느 논리 엔터티에서 구현되었는지를 기록하는 항목이다.

> ! 외래키를 설정하지 않는다
>
> 엔터티와 테이블은 1:1이 아니다. 성능을 위하여 두 엔터티를 하나의 테이블로 통합하기도 하고, 이력 분리나 파티셔닝을 위하여 하나의 엔터티를 여러 테이블로 분할하기도 한다.
>
> 외래키를 설정하면 테이블 하나에 엔터티 하나라는 관계를 데이터베이스가 강제하게 되어 통합과 분할 자체가 불가능해진다. 따라서 참조 테이블과 참조 컬럼은 비워 둔다.

여러 엔터티를 기록할 때는 쉼표로 구분한다. 표준단어의 이음동의어·금칙어와 동일한 관례이다. 이 항목은 값을 여러 개 담으므로 이름 하나 분량으로 도메인을 정하면 길이가 부족해진다. `MT_TBL.REL_ENTTY_NM`은 `명V100`을 사용한다.

> **i 통합한 테이블은 검사 방식도 달라진다**
>
> 이 항목을 쉼표로 분리하여 사용하는 규칙은 여덟 개이다. 등재 여부를 확인하는 규칙(`pairing.table-entity`·`table.entity-registered`)은 이름마다 각각 검사하고, 컬럼과 속성을 대응시키는 규칙(`pairing.column-attribute`·`attribute-column`)은 해당 테이블이 구현한 엔터티를 차례로 조회한다.
>
> 반면 1:1을 전제로 이름을 비교하는 세 규칙(`pairing.table-name`·`table-korean-name`·`table-key`)은 통합한 테이블을 판정하지 않는다. 두 엔터티를 통합한 경우에는 기준으로 삼을 이름이 존재하지 않기 때문이다. 어느 쪽 이름을 따르더라도 위반이 되므로, 통합 자체가 적절한지는 담당자의 판단에 맡긴다.

## 9.5 산출물을 개정할 때

1. 이전 버전을 해당 디렉터리의 `backup/`으로 옮긴다
2. 새 버전 번호와 작성일로 파일을 만든다
3. 표지와 양식 시트의 문서번호·버전·작성일을 파일명과 일치시킨다 (버전 항목에는 `V`를 제외하고 기록한다)
4. 검증을 실행한다
5. DDL을 다시 생성한다

DM-DST 산출물은 메타테이블이 이미 구축되어 있다면 2·3을 수작업으로 수행하지 않아도 된다. `kdsm export`가 파일명·표지·양식 시트에 동일한 버전과 동일한 날짜를 기록한다(17장). 시스템 관리자가 다루는 DM-LIB 정의서는 수작업으로 개정한다.

> **엑셀 파일을 닫은 뒤에 작업한다**
>
> 파일이 열려 있으면 `~$파일명.xlsx` 잠금 파일이 생성되며, 저장하지 않은 변경 내용은 프로그램이 읽지 못한다. 이때 오류가 발생하지 않고 이전 내용이 그대로 읽히므로, 결과를 보고 나서야 무엇이 반영되지 않았는지 확인하게 된다.

CHAPTER 10

# 기본키와 외래키

기본키와 외래키를 어떻게 기재해야 식별관계까지 표현되는지를 다룬다. 산출물의 서식 가운데 가장 오해가 잦은 부분이다.

## 10.1 두 축으로 나누어 기재한다

기본키 순번과 참조 정보를 독립된 두 항목에 기재한다. 하나의 코드로 통합하면 표현할 수 없는 관계가 생기기 때문이다.

### 10.1.1 키구분 코드 하나로는 표현할 수 없다

컬럼마다 키구분 코드(`KEY_SE_CD`)를 하나 두는 방식을 먼저 검토할 수 있다.

표 10.1. 키구분 코드값

| 코드 | 의미 |
|---|---|
| 01 | 기본키 |
| 02 | 외래키 |
| 99 | 일반 컬럼 |

여기에 기본키와 외래키를 조합하여 기본키로 삼는다는 규칙을 더할 수도 있다. 그러나 이 방식으로는 표현할 수 없는 경우가 있다.

! 반례

`MT_TBL`의 `DB_NM`과 `SCHM_NM`은 `MT_DB`를 참조하는 외래키이다. 그런데 `MT_TBL`의 기본키는 `TBL_PHYS_NM` 하나이다. 외래키인데 기본키가 아니다.

반대로 `MT_COL`의 `TBL_PHYS_NM`은 `MT_TBL`을 참조하는 외래키이면서 동시에 `MT_COL` 기본키의 첫 컬럼이다. 기본키이면서 외래키이다.

값이 하나뿐인 코드로는 이 두 가지를 함께 표현할 수 없다.

## 10.1.2 두 축으로 나눈다

컬럼이 기본키인지 여부와 외래키인지 여부는 **서로 다른 질문**이다. 따라서 항목도 나누어 기재한다.

표 10.2. 기본키와 외래키를 담는 두 축

| 항목 | 기재 내용 |
|---|---|
| 기본키 순번 | 기본키인 경우 1부터의 순번. 복합키의 순서까지 기록한다 |
| 참조 테이블 / 참조 컬럼 | 외래키인 경우 참조 대상 |

두 항목의 조합으로 표 10.3 의 네 가지 경우가 모두 표현된다.

표 10.3. 두 축의 조합이 나타내는 관계

| 기본키 순번 | 참조 | 의미 |
|---|---|---|
| 있음 | 없음 | 일반 기본키 |
| 있음 | 있음 | **식별관계** 외래키 |
| 없음 | 있음 | **비식별관계** 외래키 |
| 없음 | 없음 | 일반 컬럼 |

## 10.1.3 생성되는 DDL

### 10.1.3.1 식별관계

`MT_COL.TBL_PHYS_NM`은 기본키 순번 1이면서 `MT_TBL`을 참조한다.

```
CREATE TABLE kdsm_meta.MT_COL (
    TBL_PHYS_NM VARCHAR(40) NOT NULL,
    COL_PHYS_NM VARCHAR(40) NOT NULL,
    ...
    CONSTRAINT PK_MT_COL PRIMARY KEY (TBL_PHYS_NM, COL_PHYS_NM),
    CONSTRAINT FK_MT_COL_1 FOREIGN KEY (TBL_PHYS_NM) REFERENCES kdsm_meta.MT_TBL (TBL_PHYS_NM)
);
```

### 10.1.3.2 비식별관계

`MT_TBL.DB_NM`·`SCHM_NM`은 기본키 순번이 없고 참조만 있다. 같은 테이블을 가리키므로 하나의 복합 외래키로 생성된다.

```
CONSTRAINT PK_MT_TBL PRIMARY KEY (TBL_PHYS_NM),
CONSTRAINT FK_MT_TBL_1 FOREIGN KEY (DB_NM, SCHM_NM) REFERENCES kdsm_meta.MT_DB (DB_NM, SCHM_NM)
```

## 10.2 외래키 기록

참조 테이블과 참조 컬럼을 기록할 때 지켜야 할 사항이다.

### 10.2.1 외래키는 대상의 기본키와 정확히 일치해야 한다

> ! 복합 기본키의 일부만 참조할 수 없다
>
> 관계형 DBMS는 외래키가 가리키는 컬럼 조합에 유일성이 보장되어야 제약 조건을 허용한다. 복합 기본키의 앞부분 일부만 참조하면 DDL 실행 단계에서 거부된다.

다음이 그러한 형태이다.

```
CONSTRAINT PK_MT_TBL PRIMARY KEY (TBL_PHYS_NM, DB_NM, SCHM_NM),   -- 3컬럼 복합키
...
CONSTRAINT FK_MT_COL_1 FOREIGN KEY (TBL_PHYS_NM) REFERENCES kdsm_meta.MT_TBL (TBL_PHYS_NM)
```

참조 대상의 기본키가 3컬럼인데 참조하는 쪽은 앞의 하나만 가리킨다. DDL 은 생성되지만 실행 단계에서 거부된다. 생성이 성공하였다는 사실이 실행 가능을 뜻하지 않으므로, 적용하기 전에 검증으로 확인한다.

`column.foreign-key-target` 규칙이 이 세 가지를 함께 검사한다.

- 참조 테이블과 참조 컬럼 가운데 한쪽만 채워진 경우 (외래키가 안내 없이 누락된다)
- 참조 대상 테이블이 정의서에 없는 경우
- 참조하는 컬럼 조합이 대상 테이블의 기본키와 다른 경우

### 10.2.2 복합키의 순서

복합 기본키는 순서가 인덱스 성능을 좌우한다. 부모 키를 앞에 두는 것이 일반적이다.

```
MT_COL: TBL_PHYS_NM(1), COL_PHYS_NM(2)
```

`table.primary-key` 규칙은 구성 컬럼뿐 아니라 순서까지 대조한다. `STDCD_ID, STDCD_VL_CN`과 `STDCD_VL_CN, STDCD_ID`는 다른 키이다.

### 10.2.3 문서 간 외래키

산출물이 여러 권으로 나뉘어 있으면 참조 대상이 다른 문서에 있을 수 있다. 물리 모델의 테이블이 논리 모델의 엔터티를 가리키는 경우가 이에 해당한다. 영역별로 분할하는 DM-LIB 계열에서는 한 계열 안에서도 발생한다(9.1절).

같은 스키마에 함께 생성되므로 유효한 참조이지만, 한 문서만 확인하면 존재하지 않는 테이블로 보이다. 이러한 경우에는 다른 문서의 컬럼 정의를 `external`로 전달한다.

```
Validator(column_rules(terms, domains, tables, external=other_columns)).run(columns)
```

CLI에서는 산출물을 여럿 나열하면 자동으로 서로를 참조한다.

```
kdsm validate DM-DST-002_....xlsx DM-DST-003_....xlsx --meta 메타파일.xlsx
```

> **i 파일 실행 순서를 지켜야 한다**
>
> 문서 간 외래키가 생성되면 DDL 파일도 순서를 지켜 실행해야 한다. `physical`을 먼저 실행하면 외래키 대상이 존재하지 않아 실패한다.

## 10.3 DDL 생성 순서

생성기는 참조 의존 관계에 따라 테이블을 위상 정렬한다. 참조 대상이 먼저 생성되고, DROP은 그 역순이다. 정렬은 SQL 파일 하나를 만드는 범위 안에서 수행하므로, 산출물 하나에 모든 테이블이 있으면 그 전체가 한 순서로 정렬된다.

그림 10.1 는 KDSM 메타테이블 12개를 주제영역별로 나누어 생성하였을 때의 결과이다. 세 파일이 만들어지고 각 파일 안에서 정렬되며, 상자의 번호가 그 파일에서의 생성 차례이다.

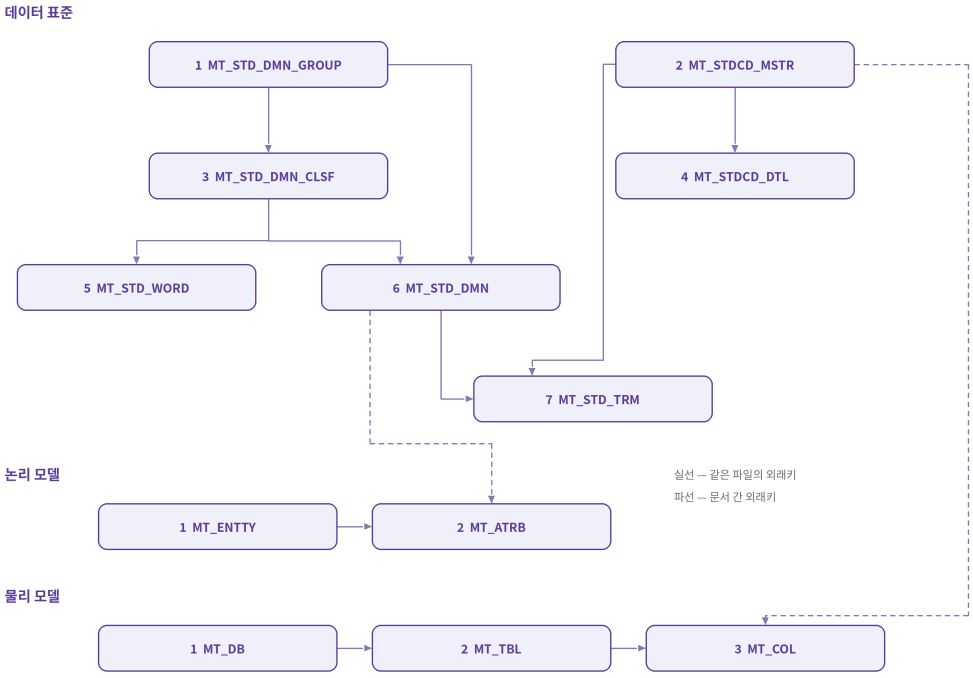


**그림 10.1.** 참조 의존 관계가 생성 차례를 정한다

화살표는 참조 의존 관계이며 가리키는 쪽이 나중에 생성된다. 번호는 그 결과이지 작성자가 정한 값이 아니다. `MT_STD_DMN` 이 6번인 것은 `MT_STD_DMN_CLSF`(3번)와 `MT_STD_DMN_GROUP`(1번)을 모두 참조하기 때문이다.

파선은 문서 간 외래키이다(10.2.3절). 다른 파일의 테이블을 가리키므로 정렬에 반영되지 않는다. 대신 파일을 실행하는 순서가 그 역할을 대신하므로, 데이터 표준을 먼저 적용한 뒤 논리 모델과 물리 모델을 적용한다.

순환 참조가 있는 경우에는 처리를 중단하지 않고 남은 테이블을 뒤에 배치한다. 이 경우 외래키를 `CREATE TABLE` 문 안에 포함할 수 없으므로 별도로 처리해야 한다.

# III

# 3부 · 라이브러리


11 kdsm 라이브러리 . . . . . 85
12 표준용어 추천 . . . . . 97
13 검증 . . . . . 109
14 DDL 생성과 대조 . . . . . 125
15 메타데이터 적재 . . . . . 137
16 ERD 자동 작성 . . . . . 149
17 산출물 내보내기 . . . . . 169
18 표준화 진단 . . . . . 179
19 커맨드라인 . . . . . 207

CHAPTER 11

# kdsm 라이브러리

`library/` 의 `kdsm` 패키지는 데이터 표준화와 데이터 모델링의 산출물을 코드로 다루기 위한 파이썬 라이브러리이다. 1부와 2부에서 사람이 판단하여 산출물에 기재한 내용을, 이 라이브러리가 읽어 검사하고 다른 형태로 변환한다.

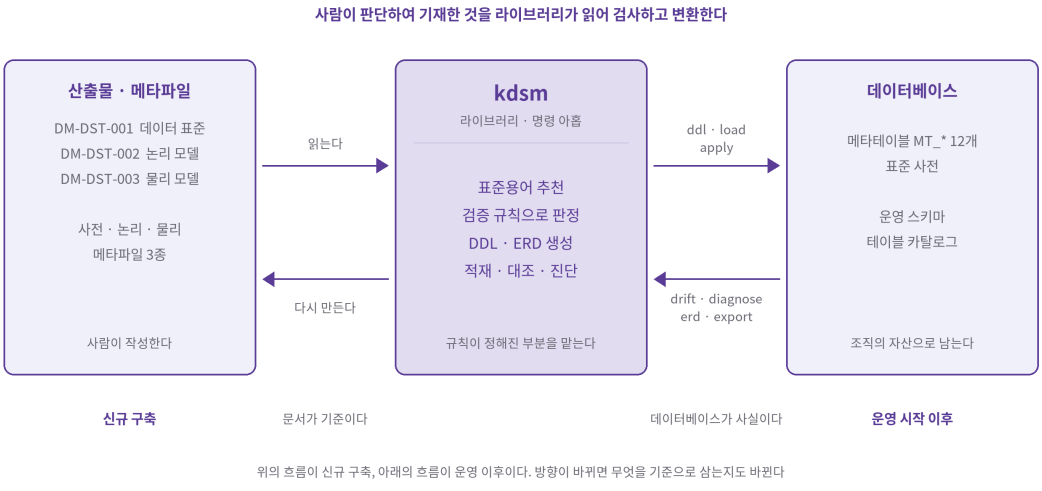


**그림 11.1.** 라이브러리가 놓인 자리와 두 방향

그림 11.1 의 위쪽 흐름이 신규 구축이고 아래쪽 흐름이 운영 이후이다. 방향이 바뀌면 무엇을 기준으로 삼는지도 함께 바뀐다(11.1.2절).

## 11.1 무엇을 위한 라이브러리인가

### 11.1.1 개발 목적

표준화 작업에는 사람의 판단이 필요한 부분과 그렇지 않은 부분이 섞여 있다. 어떤 낱말을 표준단어로 삼을 것인지, 어떤 엔터티가 필요한지는 사람이 정한다. 그러나 그 결정을 여러 문서에 옮겨 적고, 옮겨 적은 값이 서로 어긋나지 않는지 확인하고, 확인을 마친 내용을 DDL이나 ERD로 바꾸는 일은 규칙이 정해져 있다.

이 라이브러리는 규칙이 정해져 있는 부분을 담당한다. 사람의 판단을 대신하지 않는다. 판단의 결과가 일관되게 반영되었는지를 확인하고, 반영된 내용을 다른 형태로 변환한다.

표 11.1. 사람의 판단과 라이브러리의 역할

| 사람이 하는 일 | 라이브러리가 하는 일 |
|---|---|
| 표준단어·표준용어를 무엇으로 정할지 결정한다 | 이름을 낱말로 분해하여 후보를 제시한다 |
| 어떤 엔터티와 속성이 필요한지 결정한다 | 기재한 이름이 사전에 등재되어 있는지 검사한다 |
| 산출물에 값을 기재한다 | 여러 문서에 기재한 값이 서로 일치하는지 대조한다 |
| 개선 대상과 순서를 결정한다 | 표준 준수 수준을 측정하여 제시한다 |

### 11.1.2 활용 범위

다루는 대상은 산출물 파일과 데이터베이스 카탈로그이다. 두 방향으로 사용한다.

**문서에서 데이터베이스로**
: 산출물을 읽어 검증하고, DDL을 생성하여 스키마를 만들며, 메타데이터를 적재한다. 신규 구축 시점의 방향이다.

**데이터베이스에서 문서로**
: 운영 중인 데이터베이스의 카탈로그를 읽어 정의서와 대조하고, 표준 준수 수준을 진단하며, 산출물과 ERD를 다시 생성한다. 운영이 시작된 뒤의 방향이다.

한 가지를 분명히 해 둔다. 이 라이브러리는 데이터를 처리하지 않는다. 다루는 것은 데이터의 구조와 이름에 관한 정보, 즉 메타데이터이다. 업무 데이터를 조회하거나 변환하는 기능은 제공하지 않는다.

### 11.1.3 기대효과

표 11.2. 라이브러리를 사용할 때 달라지는 것

| 항목 | 사람이 수행할 때 | 라이브러리를 사용할 때 |
|---|---|---|
| 표준 준수 확인 | 산출물을 육안으로 대조한다 | 검증 규칙으로 전수 검사한다 |
| 문서 사이의 불일치 | 적재 시점에 드러난다 | 적재 이전에 검출한다 |
| DDL 작성 | 정의서를 보고 직접 작성한다 | 정의서에서 생성한다 |
| 산출물 갱신 | 데이터베이스를 보고 문서를 고친다 | 데이터베이스에서 다시 생성한다 |
| 레거시 진단 | 표본을 추출하여 확인한다 | 카탈로그 전체를 측정하여 비율로 제시한다 |

가장 큰 차이는 검사 범위이다. 육안 대조는 표본을 볼 수밖에 없으므로, 통과하였다는 사실이 나머지에 대하여 아무것도 말해 주지 않는다. 전수 검사는 검사 대상과 결과가 함께 기록되므로, 무엇을 확인하였고 무엇을 확인하지 않았는지가 남는다.

두 번째 차이는 검출 시점이다. 문서 사이의 불일치는 파일을 읽는 데에는 지장이 없으므로 적재 시점까지 드러나지 않는다. 그 시점에는 이미 스키마를 만든 뒤이므로 되돌리는 비용이 크다.

## 11.2 라이브러리와 커맨드라인

`kdsm` 은 그림 11.2 와 같이 두 계층으로 이루어진다. 아래에 기능을 제공하는 라이브러리가 있고, 그 위에 사용자가 바로 실행할 수 있는 커맨드라인이 있다.




그림 11.2. 라이브러리와 커맨드라인의 관계

표 11.3. 두 계층의 대상과 제공 범위

| 계층 | 대상 | 제공하는 것 |
|---|---|---|
| 라이브러리 (`kdsm.*`) | 애플리케이션을 개발하는 사람 | 모델·리더·규칙·생성기를 파이썬 API로 제공한다 |
| 커맨드라인 (`kdsm.cli`) | 표준화와 모델링을 수행하는 사람 | 9개 명령으로 전형적인 작업을 한 줄에 수행한다 |

### 11.2.1 커맨드라인은 라이브러리의 진입점이다

커맨드라인은 별도의 구현이 아니다. 인자를 해석하고 라이브러리의 함수를 호출한 뒤 결과를 화면에 출력하는 얇은 층이다. 따라서 두 계층의 판단 기준이 어긋나는 일은 발생하지 않는다.

```
# kdsm ddl 고객주문_V0.1.0_20260812.xlsx -o dbms/ddl/고객주문.sql --drop
from kdsm.ddl import generate_ddl_from_spec

generate_ddl_from_spec(
    "고객주문_V0.1.0_20260812.xlsx",
```

```
    output="dbms/ddl/고객주문.sql",
    include_drop=True,
)
```

위의 두 가지는 같은 작업이다. 명령을 사용하면 인자 검사와 오류 안내를 함께 얻고, 함수를 직접 호출하면 결과를 파이썬 객체로 받아 이어서 처리할 수 있다.

일반 사용자에게는 커맨드라인이 사실상 유일한 진입점이다. 파이썬을 사용하지 않아도 9개 명령으로 검증부터 진단까지 수행할 수 있다(19장).

### 11.2.2 라이브러리를 직접 사용하는 경우

다음과 같은 경우에는 커맨드라인 대신 라이브러리를 사용한다.

**애플리케이션을 개발하는 경우**
: 표준화와 모델링을 화면에서 수행하는 웹 애플리케이션이 그러하다. 검증 결과를 화면에 표시하려면 종료 코드와 화면 출력이 아니라 결과 객체가 필요하다.

**결과를 이어서 처리하는 경우**
: 검증 결과에서 특정 규칙의 위반만 추려 별도의 보고서로 만들거나, 진단 결과를 다른 시스템에 전달하는 경우이다.

**명령이 제공하지 않는 조합이 필요한 경우**
: 커맨드라인은 전형적인 작업을 대상으로 하므로 모든 조합을 옵션으로 제공하지는 않는다.

> **i 어느 쪽을 사용하든 결과는 같다**
>
> 같은 작업을 커맨드라인과 라이브러리가 다르게 판정하는 일은 없다. 규칙과 생성기가 한 곳에 있고 커맨드라인이 그것을 호출하기 때문이다.
>
> 따라서 커맨드라인으로 확인한 결과를 애플리케이션에서 재현할 수 있고, 반대로 애플리케이션에서 발견한 문제를 커맨드라인으로 재현할 수 있다.

## 11.3 설치와 구성

`kdsm` 패키지의 설치 방법과 모듈 구성을 살펴본다.

### 11.3.1 설치

Git 저장소에서 바로 설치한다. 별도의 패키지 서버를 두지 않으며 저장소의 태그가 곧 판이다.

```
uv tool install 'kdsm[db] @ git+https://<리파지토리 URL>/data-standardization-
application.git@v0.11.0#subdirectory=library'
```

자기 코드에서 `import kdsm` 한다면 독립 설치 대신 그 프로젝트의 환경에 넣는다. KDSM 자체를 고칠 때에만 저장소를 내려받아 `library/` 에서 동기화한다.

```
cd library && uv sync --all-extras      # 개발판
```

Python 3.11 이상이 필요하다. 설치 절차 전문과 접근 권한·판 올리기, uv 자체의 설치 방법과 psql·mermaid-cli 같은 외부 도구는 20장 에 있다.

**표 11.4.** 선택 의존성

| extra | 활성화되는 기능 |
|---|---|
| `db` | `kdsm load`, `--from-db`, PostgreSQL 진단 |
| `llm` | `--coin` 제안을 Claude API로 |
| `bedrock` | `--coin` 제안을 AWS Bedrock으로 |
| `databricks` | `--dbms databricks` 적재 · 메타데이블 조회 · 카탈로그 읽기 · 진단 |
| `mysql` | `--dbms mysql` 적재 · 메타데이블 조회 · 카탈로그 읽기 · 진단 |
| `pdf` | ERD의 PDF 합본(`--pdf-merge`) |

⚠ 선택 의존성은 한 번에 지정한다

`uv sync`는 기본이 **정확 동기화**이다. 요청한 것만 남기고 나머지를 제거한다. 따라서 다음과 같이 실행하면 두 번째 명령이 첫 번째 명령으로 설치한 psycopg를 제거한다.

```
uv sync --extra db       # psycopg 설치
uv sync --extra llm      # psycopg 제거되고 anthropic 설치
kdsm suggest 종신보험이름 --from-db --coin
#> 오류: psycopg가 설치되어 있지 않습니다
```

사용할 extra를 한 줄에 모두 지정하거나, 판단이 어려우면 `--all-extras`를 지정한다. extra가 추가되어도 명령을 수정할 필요가 없다.

```
uv sync --extra db --extra llm
uv sync --all-extras
```

이미 설치한 것을 유지한 채 추가하려면 `--inexact`를 지정한다. 다만 잠금 파일과 환경이 일치하지 않을 수 있으므로 권장하지 않는다. 자세한 내용은 21.1.4.2절 에 있다.

`ModuleNotFoundError: No module named 'kdsm'`이 발생하는 경우

`uv sync`는 가상환경에 편집 가능 모드로 설치한다. 스크립트를 실행할 때 해당 환경을 사용하고 있는지 확인한다.

```
uv run python your_script.py
uv run pytest
```

## 11.3.2 구성

표 11.5. kdsm 서브패키지 구성

| 서브패키지 | 역할 |
|---|---|
| `kdsm.constants` | 표준코드 상수 (`WordSection`, `Status`, `Table`) |
| `kdsm.exceptions` | 예외 계층 (`KdsmError` 하위 6종) |

| 서브패키지 | 역할 |
|---|---|
| `kdsm.models` | 도메인 모델 (`StandardWord`, `StandardTerm`, …) |
| `kdsm.io` | 메타파일·산출물 정의서 입출력 |
| `kdsm.suggest` | 표준용어 추천 · 신규 단어 제안 |
| `kdsm.validate` | 표준 준수 검증 |
| `kdsm.ddl` | DDL 생성 (PostgreSQL · MySQL · Databricks) |
| `kdsm.drift` | 정의서와 운영 데이터베이스의 차이 대조 |
| `kdsm.db` | 메타데이터 적재 · 적재본 읽기 (PostgreSQL · MySQL · Databricks) |
| `kdsm.diagnose` | 표준화 수준 진단 (PostgreSQL · MySQL · Databricks) |
| `kdsm.export` | DM-DST 산출물 자동 생성 (메타테이블 · DBMS 카탈로그) |
| `kdsm.erd` | Mermaid ERD 작성 (산출물 · 메타테이블 · DBMS 카탈로그) |
| `kdsm.seed` | 패키지와 함께 배포하는 초기 데이터의 경로 |
| `kdsm.cli` | 커맨드라인 |

앞의 넷은 이름을 짓고 옮겨 적는 일을 다루고, 뒤의 다섯은 데이터베이스를 상대하는 일을 다룬다. 3부의 장 차례도 이 순서를 따른다(22장). `kdsm.seed` 와 `kdsm.cli` 는 그 둘에 속하지 않는다. 하나는 배포에 얹는 자료의 자리를 알려 주고, 하나는 명령을 받는다.

### 11.3.3 패키지와 함께 배포하는 것

`kdsm` 은 파이썬 코드만 담고 있지 않다. 라이브러리가 제 기능을 하는 데 필요한 자료를 함께 넣는다. 저장소를 내려받지 않고 패키지만 설치하여도 메타테이블 열둘을 세우고 산출물을 만들 수 있어야 하기 때문이다.

표 11.6. 패키지에 함께 들어가는 자료

| 자리 | 담긴 것 | 쓰는 곳 |
|---|---|---|
| `templates/` | DM-DST 빈 템플릿 3종 | `kdsm export` |
| `seed/meta_table/` | 물리 정의서 DM-LIB-004·005·006 | `kdsm ddl` · `kdsm load -s` |
| `seed/meta_data/` | 메타파일 3종. 표준 사전과 KDSM 자신의 논리·물리 모델 | `kdsm load` |
| `diagnose/assets/` | Typst 서식과 스타일 | `kdsm diagnose --qmd` |

경로는 `kdsm.seed` 가 알려 준다. 파일명 뒤에 작성일이 붙어 개정할 때마다 달라지므로 이름을 직접 적지 않는다.

```
from kdsm.seed import pairs
```

```
for area, metafile, spec in pairs():
    print(area, metafile.name, spec.name)
```

돌려주는 순서가 적재 순서이다. 뒤의 것이 앞의 것을 가리키므로 순서를 바꾸면 외래키에 걸린다.

> **i 정본은 저장소에 있다**
>
> 같은 파일이 저장소의 `dbms/meta` 아래에도 있으며 그쪽이 정본이다. 패키지 안의 것은 배포용 사본이므로 정본을 개정하면 함께 갱신한다.
>
> 논리 정의서 DM-LIB-001·002·003 은 사본에 넣지 않는다. 메타테이블을 만드는 데 필요한 것은 물리 정의서이며, 논리 정의서는 그 근거를 사람이 읽는 서식으로 남긴 문서이다.

> **! 함께 배포하는 사전은 출발점이다**
>
> 패키지에 든 표준단어 사전을 그대로 써야 하는 것이 아니다. 업무에 맞지 않는 낱말을 덜어 내고 필요한 낱말을 더하는 것이 표준화 작업의 첫 단계이다(2장).

`kdsm.diagnose`의 규모가 가장 크다. 파일이 12개이므로 내부 구성을 나누어 제시한다.

표 11.7. kdsm.diagnose 모듈

| 모듈 | 역할 |
|---|---|
| `catalog` | 카탈로그 읽기와 `CatalogColumn`, `CatalogSource` 규약 |
| `dialect` | DBMS별 타입 규칙 (`POSTGRES` · `MYSQL` · `DATABRICKS`) |
| `databricks` | Unity Catalog 리더와 접속 |
| `mysql` | MySQL 카탈로그 리더와 접속 |
| `source` | 접속 값으로 카탈로그 원천을 여는 진입점 (`open_catalog` · `catalog_config`) |
| `diagnoser` | 사전과 대조하여 세 가지 진단 대상의 준수율을 산출한다 |
| `consistency` | 사전 없이 스키마 내부의 불일치를 검출한다 |
| `stats` | 주석 채움율과 이름 통계 |
| `advice` | 진단 결과에서 조치 사항을 순서대로 도출한다 |
| `run` | 위의 셋을 한 번에 실행한다 (`diagnose_catalog` · `Diagnosis`) |
| `report` | 화면·엑셀·Quarto 세 형태로 출력한다 |
| `render` | Quarto 문서를 HTML과 PDF로 조판한다 |

**i 방언 모듈이 셋인 이유**

`kdsm.ddl.dialect`와 `kdsm.diagnose.dialect`와 `kdsm.db.writer`는 서로 다른 모듈이다.

전자는 DDL을 **작성할 때**의 규칙이다. 식별자를 인용하는 방식과 타입을 표기하는 방식을 정한다. 사전의 이름을 DBMS의 표기로 옮기는 방향이다. PostgreSQL·MySQL·Databricks 세 종류를 갖추고 있다.

후자는 카탈로그를 **읽을 때**의 규칙이다. DBMS가 사용하는 타입 이름을 사전의 이름과 대응시키고, 길이를 측정할 수 있는지를 판단한다. PostgreSQL·MySQL·Databricks 세 종류를 갖추고 있다.

앞의 둘은 적용 방향이 반대이다. 하나는 사전의 이름을 DBMS의 표기로 옮기고 다른 하나는 그 반대이므로 서로 대체할 수 없다.

`kdsm.db.writer`는 값을 **넣을 때**의 규칙이다. DBMS마다 다른 UPSERT 문장을 만든다(15.3.1절). 방향은 `ddl.dialect`와 같으며 식별자를 인용하는 규칙도 그쪽 것을 그대로 쓴다. 만들 때와 넣을 때 이름을 다르게 적으면 없는 테이블을 찾게 되기 때문이다. 다루는 문장이 다를 뿐이다. 하나는 테이블을 만들고 하나는 값을 넣는다.

방언을 확장할 때에는 셋 중 어느 것인지 먼저 확인한다.

## 11.4 설계 메모

설계 의도와 구현 과정에서 확인한 사항을 정리한다.

### 11.4.1 모델과 물리 컬럼의 대응

도메인 모델의 필드는 파이썬 관례를 따르되, `alias`로 물리 컬럼명을 유지한다.

```
from kdsm.models import StandardWord

word = StandardWord.model_validate({"STD_WORD_NM": "고객", ...})
row = word.model_dump(by_alias=True)     # 다시 물리 컬럼명으로
```

이에 따라 메타파일에서 읽은 행 딕셔너리를 변환 없이 그대로 전달할 수 있고, 적재할 때도 별칭으로 덤프하면 키가 곧 컬럼명이 된다.

**⚠ 헤더가 바뀌면 별칭도 바꿔야 한다**

메타파일의 컬럼명을 개정하면 이 별칭도 함께 수정한다. 별칭이 일치하지 않으면 `extra="ignore"` 설정으로 인해 값이 안내 없이 `None`이 된다. 예외가 발생하지 않으므로 결과를 보고 나서야 인지하게 된다.

`meta_header_rules`가 이러한 불일치를 검출한다.

### 11.4.2 파일 두 종류를 다르게 읽는다

표 11.8. 파일 종류별 리더

| 파일 | 리더 | 구조 |
|---|---|---|
| 메타파일 `kdsm_modeling_meta_*.xlsx` | `MetaWorkbook` | 1행이 헤더인 단순 표 |

| 파일 | 리더 | 구조 |
|---|---|---|
| 논리 메타파일 `kdsm_logical_meta_*.xlsx` | `MetaWorkbook` | 1행이 헤더인 단순 표 |
| 물리 메타파일 `kdsm_physical_meta_*.xlsx` | `MetaWorkbook` | 1행이 헤더인 단순 표 |
| 산출물 `DM-DST-001·002·003` | `SpecWorkbook` | 표지·설명·지침이 섞인 서식 문서 |

메타파일은 시트명이 물리 테이블명과 같고 1행이 컬럼명이므로 그대로 읽을 수 있다. 산출물은 헤더 위치가 고정되어 있지 않으므로 `*` 표시로 헤더를 찾는다(9.3.2절).

```
from kdsm.io import MetaWorkbook, SpecWorkbook

wb = MetaWorkbook.open("dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx")
words = wb.read_words()
terms = wb.read_terms()

logical = MetaWorkbook.open("dbms/meta/meta_data/kdsm_logical_meta_20260828.xlsx")
entities = logical.read_entities()
attributes = logical.read_attributes()

spec = SpecWorkbook.open("DM-DST-003_물리데이터모델_....xlsx")
rows = spec.read_data("컬럼 정의서")

# 논리 모델 산출물은 엔터티·속성을 모델로 바로 돌려준다
logical_spec = SpecWorkbook.open("DM-DST-002_논리데이터모델_....xlsx")
spec_entities = logical_spec.read_entities()
```

### 11.4.3 관리용 컬럼

메타파일에는 표준화 대상이 아닌 관리용 컬럼이 추가될 수 있다. `MT_STD_WORD`의 `SRC`(출처)가 그러한 예이다.

모델은 `extra="ignore"`로 이러한 컬럼을 무시하고, 검증은 `MANAGED_COLUMNS`로 면제한다.

```
meta_header_rules(terms)                  # SRC 자동 면제
meta_header_rules(terms, set())           # 전부 검사
meta_header_rules(terms, {"SRC", "MEMO"})  # 면제 목록 직접 지정
```

### 11.4.4 엑셀을 읽을 때 주의할 점

> ⚠ 숫자로만 된 표준단어
>
> "10", "1월" 같은 값을 엑셀이 수치 셀로 저장하면 openpyxl이 `int`로 반환한다. 리더가 이를 문자열로 변환하므로 별도로 조치할 사항은 없다. 다만 사전을 직접 수정할 때 해당 셀의 서식이 텍스트인지 확인해 두면 다른 도구로 읽을 때에도 값이 그대로 유지된다.

> ⚠ 지운 셀은 빈 문자열로 남는다
>
> 엑셀에서 셀 내용을 지우면 `None`이 아니라 공백이나 빈 문자열이 남는 경우가 흔하다. 리더가 이를 `None`으로 정규화하며, `column.not-null-satisfied` 규칙도 공백만으로 이루어진 문자열을 빈 값으로 판정한다.

### 11.4.5 예외 계층

```
KdsmError
├── MetadataNotFoundError    파일·시트·항목을 찾을 수 없음
├── MetadataFormatError      시트 구성이나 컬럼 구조가 기대와 다름
├── StandardViolationError   표준 규칙을 위반한 값
├── DdlGenerationError       DDL 생성 정보 부족·모순
├── DatabaseError            연결·적재·조회 오류
└── RenderError              Quarto 조판 실패·실행 파일 없음
```

CLI 는 `KdsmError` 를 포착하여 추적 정보 없이 메시지만 출력한다. 따라서 추적 정보가 함께 출력되었다면 사용자가 조치할 사항이 아니라 라이브러리의 결함이다. 그대로 보고한다.

## 11.5 모듈이 제공하는 것

모듈별로 사용할 수 있는 범위이다. 여기에 없는 기능은 제공하지 않으므로 사람이 수행한다(표 22.4).

표 11.9. 모듈이 제공하는 범위

| 모듈 | 제공하는 범위 |
|---|---|
| `constants` · `exceptions` · `models` | 상수·예외·데이터 모델 |
| `validate` | 검증 규칙 |
| `io.excel` | 메타파일 읽기 (사전 7 · 논리 2 · 물리 3 시트). 쓰기는 제공하지 않는다 |
| `io.spec` | 산출물 읽기·쓰기 (서식 보존) · 표지 읽기 · 파일명과 표지의 대조 |
| `ddl` | DDL 생성 (PostgreSQL · MySQL · Databricks) |
| `db.connection` · `db.loader` · `db.writer` | 접속과 적재 (12개 테이블 · 세 DBMS) |
| `db.reader` | `MetaDatabase` 로 사전 조회 (12개 테이블 · 세 DBMS) |
| `db.repository` | 테이블별 저장소 (12개 테이블 · 세 DBMS). 조건 조회와 한 테이블 단위의 적재·삭제 |
| `suggest.segmenter` · `suggest.replacements` · `suggest.suggester` | 분해와 추천 |
| `suggest.coining` | 등재 초안 생성 (선택 의존성 `llm` 또는 `bedrock`) |
| `diagnose.catalog` · `diagnose.dialect` | 카탈로그 읽기와 DBMS별 타입 규칙 |
| `diagnose.source` | 접속 값으로 카탈로그 원천 열기 (세 DBMS 공통 · 18.5.1.1절) |
| `diagnose.diagnoser` · `consistency` · `stats` · `advice` | 준수율·일관성·이름 통계·제언 |
| `diagnose.run` | 진단 한 번의 조립 (18.5.1.2절) |
| `diagnose.report` | 화면·엑셀·Quarto 리포트 |
| `diagnose.databricks` · `diagnose.mysql` | 해당 DBMS 접속 (선택 의존성) |

| 모듈 | 제공하는 범위 |
|---|---|
| `drift` | 정의서와 카탈로그 대조 |
| `export` | DM-DST 산출물 3종 생성 |
| `erd` | ERD 생성 (산출물 · 메타테이블 · 카탈로그) |
| `cli` | 9개 명령 |

`db.repository` 는 메타테이블 열둘에 저장소를 하나씩 둔다. 사전 한 벌을 전부 읽을 때는 `db.reader` 의 `MetaDatabase` 를 사용하고, 조건을 지정하여 조회하거나 한 테이블만 적재·삭제할 때는 저장소를 사용한다.

진단할 수 있는 DBMS 는 PostgreSQL·MySQL·Databricks 이다(표 18.16).

CHAPTER 12

# 표준용어 추천

한글 용어명을 표준단어 단위로 분리하여 표준용어 후보를 생성한다. 표준화되지 않은 컬럼 목록을 일괄 표준화할 때 사용한다.

```
from kdsm.io import MetaWorkbook
from kdsm.suggest import TermSuggester

wb = MetaWorkbook.open("dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx")
suggester = TermSuggester(wb.read_words(), wb.read_domains(), wb.read_terms())

for c in suggester.suggest("고객여신금액"):
    print(c.physical_name, c.standard_domain, c.score)
```

## 12.1 사전을 어디서 읽는가

메타파일과 DB 테이블 양쪽에서 모두 읽을 수 있다. 두 경로가 같은 `read_*` 메서드를 제공하므로, 사전을 전달받아 사용하는 쪽은 어느 경로에서 읽었는지 알 필요가 없다.

```
kdsm suggest 고객여신금액 --meta dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx
kdsm suggest 고객여신금액 --from-db                    # KDSM_DB_* 환경변수로 접속
kdsm suggest 고객여신금액 --from-db --schema other_meta   # 다른 스키마의 사본
```

`--meta`는 파일만 읽으므로 환경변수가 필요하지 않다. `--from-db`는 `KDSM_DB_HOST`부터 `KDSM_DB_PASSWORD`까지 여섯 변수로 접속한다(20.9절). `--schema`를 생략하면 `KDSM_DB_SCHEMA`를 쓴다.

```
from kdsm.db import MetaDatabase

with MetaDatabase.from_env() as source:
    suggester = TermSuggester(
        source.read_words(), source.read_domains(), source.read_terms()
    )
```

**i 둘 다 남겨 둔 이유**

메타파일은 사람이 수정하는 원본이고, 테이블은 애플리케이션이 조회하는 사본이다. 표준을 정의하는 단계에서는 파일이 빠르다. 엑셀에서 수정한 뒤 즉시 실행해 볼 수 있기 때문이다. 반면 애플리케이션에서 추천 기능을 호출할 때는 이미 열려 있는 연결을 사용하는 것이 적절하다. 파일 경로를 배포 구성에 포함하지 않아도 되기 때문이다.

어느 한쪽만 제공하면 다른 용도에서 불편이 발생한다.

**⚠ 사본은 최신 상태가 아닐 수 있다**

테이블은 `kdsm load`로 적재한 시점의 사전이다. 메타파일을 수정하고 적재하지 않으면 `--meta`와 `--from-db`의 결과가 달라진다.

표준을 수정하는 중이라면 `--meta`를, 확정된 사전으로 작업할 때는 `--from-db`를 사용한다.

`MetaDatabase`는 연결을 전달받으면 닫지 않는다. 다른 작업과 한 트랜잭션으로 묶을 수 있어야 하기 때문이다. `from_env()`로 만들면 이 객체가 연결을 소유하고 컨텍스트를 벗어날 때 닫는다.

**! 조회할 때도 식별자를 인용하지 않는다**

DDL이 인용 없이 테이블을 생성하여 PostgreSQL이 이름을 소문자로 변환하여 저장한다(21.1.4.1절). 조회할 때만 인용하면 존재하지 않는 테이블을 조회하게 된다.

반환되는 컬럼명도 소문자이다. 모델의 별칭은 대문자이므로, 별칭을 소문자로 변환한 대응표를 만들어 원래 이름으로 복원한다. 대소문자만 대응시키므로 컬럼이 추가되어도 수정할 부분이 없다.

## 12.2 분해

한글 용어명을 표준단어 단위로 분리하는 것이 이 기능의 핵심이다. 나머지는 기계적으로 결정된다.

### 12.2.1 원리

표준용어는 `주제어 + 수식어* + 분류어` 구조이다. 따라서 분리에 성공하면 나머지는 기계적으로 결정된다.

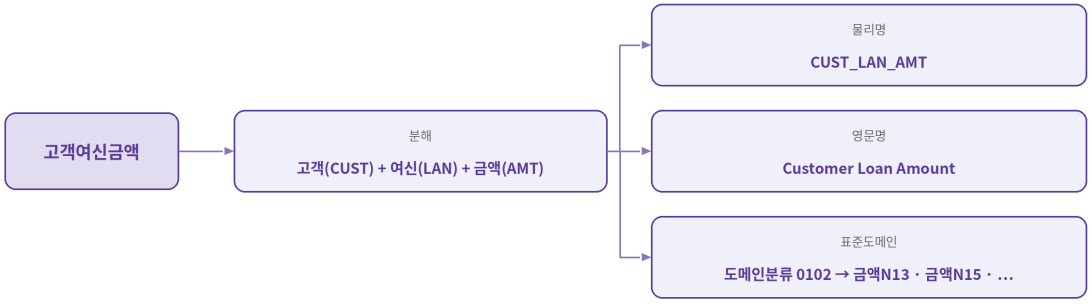


그림 12.1. 분리에 성공하면 나머지는 기계적으로 결정된다

어려운 부분은 분리하는 작업뿐이다.

### 12.2.2 분해기

사전 4,143개 단어를 트라이(trie)로 색인한다.

> 트라이 색인의 구조
>
> “고객”과 “고가”가 있으면 “고” 아래에 “객”과 “가”가 연결된다. 특정 위치에서 시작하는 단어를 찾을 때 글자를 하나씩 따라가기만 하면 되므로, 4천 건의 사전을 매번 전수 조회하지 않아도 된다. 구축에 3밀리초, 분해에 1밀리초 미만이 소요된다.

한국어는 같은 문자열이 여러 방식으로 분리될 수 있다. 가능한 분해를 모두 생성한 뒤 표 12.1 의 기준으로 점수를 부여하여 순위를 정한다.

#### 12.2.2.1 점수 기준

표 12.1. 분해 점수 기준

| 항목 | 점수 |
|---|---|
| 기준 점수 | 100 |
| 단어 하나마다 | −10 |
| 마지막이 분류어 | +40 |
| 미등록 단어 하나마다 | −30 |
| 미등록 단어의 글자 하나마다 | −15 |

```
"고객주문일자"
  110.0  고객 + 주문 + 일자    CUST_ORDR_YMD    분류어 종결
   25.0  고객 + 주문 + 일      CUST_ORDR_DAY    미등록 ['자']
```

> 분류어 가산이 핵심
>
> 단어 수만으로는 고객+주문+일자(3단어)와 고객+주문+일+자(4단어)의 차이가 10점에 그친다. 분류어 가산 40점이 있어야 표준용어가 될 수 있는 분해가 확실하게 상위에 배치된다.

⚠ 미등록 단어는 글자 수까지 반영한다

미등록 단어의 감점은 하나당 30점에 글자당 15점을 더한다. 개수만으로 감점하면 다음과 같은 역전이 발생하기 때문이다. 「데이터표준점수」를 분해하되 사전에 「표준」과 「점수」만 등재되어 있는 경우이다.

표 12.2. 미등록 단어의 감점 방식에 따른 점수

| 분해 | 개수만 셀 때 | 글자 수까지 셀 때 |
|---|---|---|
| 점수 · 미등록 `데이터표준` | 60.0 | −15.0 |
| 표준 + 점수 · 미등록 `데이터` | 50.0 | 5.0 |

개수만 셀 때에는 미등록 단어가 양쪽 모두 하나이므로 감점이 같고, 단어를 적게 인식한 쪽이 단어당 감점 10점을 덜 받아 상위에 오른다. 인식할 수 있는 단어를 미등록 구간에 넣을수록 유리해지는 구조이다.

글자 수를 함께 계산하면 미등록 구간이 길수록 감점이 커지므로 이 역전이 발생하지 않는다. 다섯 글자를 인식하지 못한 쪽은 75점을, 세 글자만 인식하지 못한 쪽은 45점을 감점받는다.

한 글자 미등록 단어도 분류어 가산 40점보다 큰 45점이 감점되도록 정한 것은, 「수료」를 분류어 「수」와 미등록 「료」로 나누는 것과 같은 억지 분해가 상위에 오르지 않도록 하기 위해서이다.

#### 12.2.2.2 미등록 단어가 섞여도 나머지를 활용한다

```
"고객XX금액"
  60.0  고객 + 금액    CUST_AMT    미등록 ['XX']
```

일부만 인식하지 못한 이름도 나머지 부분을 활용한다. 미등록 단어는 신규 표준단어 등재 후보가 된다.

```
suggester.find_unregistered_words("고객XX금액")   # ['XX']
```

#### 12.2.2.3 폐기 단어는 조합 대상에서 제외한다

폐기(03) 단어는 색인에서 제외한다. 더 이상 사용하지 않기로 결정한 단어이므로 새 용어를 구성하는 요소가 될 수 없다. 이음동의어와 금칙어도 색인에 포함하지 않는다. 표준단어가 아니기 때문이다.

다만 제외하는 것으로 끝내지 않는다. 12.3절 를 참고한다.

## 12.3 금지어

사용할 수 없는 낱말은 세 가지로 정의된다.

표 12.3. 금지어의 세 가지 구분

| 구분 | 기록 위치 | 예 |
|---|---|---|
| 이음동의어 | 확정 단어의 `ALLO_SYNM_CN` | 파라미터 → 인자 (185건) |
| 금칙어 | 확정 단어의 `PHWRD` | 갯수 → 개수 (38건) |
| 폐기된 표준단어 | `MT_STD_WORD`의 상태 03만 | 지금은 없음 |

앞의 두 가지는 대체어를 제시할 수 있고, 세 번째는 사용 금지만 안내할 수 있다. 한 낱말이 두 가지에 모두 해당하면 대체어를 제시할 수 있는 쪽으로 안내한다.

> ! 미등록 낱말과 사용 금지 낱말은 다르다
>
> 세 가지 모두 색인에서 제외되므로 분해기에서는 동일하게 미등록 단어로 처리된다. 그러나 사용자에게는 전혀 다른 정보이다. 전자는 신규 등재 여부를 판단할 후보이고, 후자는 이미 판단이 끝나 대체어까지 정해진 낱말이다.
>
> "파라미터값"을 입력한 사용자에게 필요한 안내는 "파라미터를 알 수 없다"가 아니라 "파라미터는 인자로 사용한다"이다.

따라서 금지어가 포함되면 대체어로 변경한 이름의 후보를 함께 생성한다.

```
# 계약날짜
  *  120.0  CTRT_YMD  [연월일C8]  Contract Date
      이름: 계약날짜 → 계약일자
      구성: 계약 + 일자
      · '날짜'는 이음동의어. 일자(YMD)로 쓴다
```

```
suggester.normalize("계약날짜")           # ('계약일자', (Replacement(...),))
suggester.find_banned_words("계약날짜")
suggester.suggest("계약날짜", rewrite=False)   # 바꾸지 않고 그대로 보기
```

> i 폐기할 때에는 대체어를 함께 지정한다
>
> 폐기는 사용을 금지한다는 의미이고, 대체어는 대신 사용할 낱말을 지정한다는 의미이다. 대체어 없이 폐기만 하면 사용자가 취할 수 있는 조치가 없으므로, 추천기는 「폐기되었으나 대체어를 확인할 수 없다」고 안내하게 된다.
>
> 폐기 단어를 모두 확정 단어의 이음동의어나 금칙어로 지정해 두면 이 안내가 발생하지 않는다. 전수 검토는 이 상태를 만드는 작업이다(2.2.2.2절).

### 12.3.1 인식하지 못한 구간만 치환한다

> ⚠ 이름 전체를 치환하면 정상적인 단어까지 변형된다
>
> 금지어 중에는 "명"·"량"·"비"와 같이 한 글자인 것이 있다. 이름 전체에 치환을 적용하면 정상적인 단어의 일부까지 치환된다. 사전 규모가 커질수록 잘못 변형되는 단어가 늘어난다.
>
> ```
> 설명 → 설이름      매출액 → 매출금액      비교 → 비용교
> ```
>
> 따라서 분해기가 인식하지 못한 구간에 걸친 것만 치환한다. 이 제한을 적용하면 이미 등재된 단어와 용어는 치환 대상이 되지 않는다.

구간이 겹치기만 하면 치환 대상으로 판정한다. 금지어가 미등록 단어보다 넓은 범위를 차지할 수 있기 때문이다.

```
"갯수"  →  분해기는 "수"만 알아보고 "갯"을 남긴다
           금칙어는 "갯수" 전체다
           남은 자리(0번 글자)에 걸치므로 "개수"로 바꾼다
```

### 12.3.2 대체어가 여럿이면 고르지 않는다

```
· '패스워드'는 이음동의어. 대체어가 여럿이라 골라야 함: 비밀번호(PSWD) · 암호(PW)
```

시스템이 임의로 선택하면 잘못된 이름이 인지되지 않은 채 등재된다. 사람이 선택해야 한다.

> **i** 후보가 여럿이면 사전을 먼저 확인한다
>
> 한 금칙어에 대체어가 둘 이상 제시되면 사전의 입력 오류일 가능성이 있다. 금칙어 항목은 값이 한 행씩 밀려 입력되기 쉬운 자리이며, 이 경우 뜻이 전혀 다른 낱말이 대체어로 함께 제시된다. 「메세지」의 대체어로 「메시지」와 「과장」이 함께 나오는 것이 그러한 형태이다.
>
> 뜻이 통하지 않는 후보가 섞여 있으면 사전의 해당 행을 확인한다. 시스템이 둘 중 하나를 임의로 선택하면 이 오류가 드러나지 않은 채 잘못된 이름이 등재된다. 후보를 좁히지 않고 모두 제시하는 이유가 여기에 있다.
>
> 대체어가 실제로 둘인 경우도 있으므로 후보가 여럿이라는 사실 자체가 오류를 뜻하지는 않는다. 뜻이 통하는지로 판별한다.

## 12.4 추천기

### 12.4.1 도메인 추론

분류어의 도메인분류에 속하는 표준도메인 중에서 고른다.

표 12.4. 길이 힌트에 따른 도메인 선택

| 상황 | 선택 |
|---|---|
| 힌트 없음 | 그 분류에서 가장 짧은 것 |
| 힌트 있음 | 담을 수 있는 것 중 가장 작은 것 |
| 어느 것도 못 담음 | 가장 큰 것 |

> **담을 수 있는 것 중 가장 작은 도메인**
>
> 더 큰 도메인이 필요하면 사용자가 후보 목록에서 선택하면 된다. 후보가 여럿이면 근거에 모두 나열한다.
>
> ```
> · 도메인 후보: 금액N13 · 금액N15 · 금액N17 · 금액N18 · 금액N22,2
> ```

```
suggester.infer_domain(classifier, length_hint=14)   # 금액N15
```

### 12.4.2 기존 용어와의 관계

표 12.5. 기존 용어와의 관계 판정

| 상황 | 판정 |
|---|---|
| 다른 용어가 같은 물리명을 사용한다 | 충돌. 점수 −30 |
| 같은 이름의 용어가 같은 물리명을 사용한다 | 이미 등재됨. 충돌 아님 |
| 폐기한 용어의 물리명 | 다시 사용할 수 있음 |

### 12.4.3 미등록 단어의 등재 초안

추천기는 미등록 단어를 찾는 데까지 관여하고, 영문명·영문약어·정의·단어구분은 사람이 작성한다. `--coin` 옵션을 지정하면 그 초안을 LLM으로부터 제공받는다.

```
kdsm suggest 메타데이터마트이름 --meta ... --coin
```

#### 12.4.3.1 LLM 제공자

Claude API에 직접 연결하거나 AWS Bedrock을 경유한다.

표 12.6. LLM 제공자

| 제공자 | 설치 | 자격 증명 |
|---|---|---|
| `anthropic` (기본) | `uv sync --extra llm` | `KDSM_LLM_API_KEY` |
| `bedrock` | `uv sync --extra bedrock` | `AWS_ACCESS_KEY_ID` · `AWS_SECRET_ACCESS_KEY` |

```
export KDSM_LLM_API_KEY=sk-ant-...        # anthropic. ANTHROPIC_API_KEY 도 읽는다

export AWS_ACCESS_KEY_ID=AKIA...          # bedrock
read -rs -p "시크릿 키: " AWS_SECRET_ACCESS_KEY && export AWS_SECRET_ACCESS_KEY
export AWS_REGION=ap-northeast-2

kdsm suggest 메타데이터마트이름 --meta ... --coin --llm-provider bedrock
```

API 키와 AWS 키도 인자로 받지 않는다. 비밀번호와 같은 이유이며, 값을 노출하지 않고 지정하는 방법은 20.10절 에 있다. 위에서 시크릿 키만 `read -rs`로 받은 것은 액세스 키 ID는 비밀이 아니고 시크릿 키가 비밀이기 때문이다.

`KDSM_LLM_PROVIDER=bedrock`을 셸 설정에 기록해 두면 옵션을 매번 지정하지 않아도 된다. 인자를 지정하면 인자가 우선한다.

자격 증명이 없으면 `--coin`을 지정하여도 무엇이 없는지 표준 오류로 알린 뒤 초안 없이 진행한다. 중단하지 않는 이유는 초안이 추천의 부가 기능이기 때문이다. 사전에 있는 단어로 만든 후보는 자격 증명 없이도 그대로 제시된다.

```
오류: 표준단어 제안을 만들지 못했습니다: LLM 제안에는 API 키가 필요합니다. KDSM_LLM_API_KEY를 설정하세요.
```

프롬프트와 도구 정의는 두 제공자가 동일하다. Bedrock의 Claude도 도구 사용을 지원하고 메시지 형식이 같기 때문이다. 차이가 있는 것은 클라이언트를 생성하는 방법과 모델 이름뿐이므로 그 두 가지만 분리한다.

⚠ Bedrock은 모델 이름이 다르다

`claude-sonnet-5` 같은 Claude API 이름을 그대로 지정하면 연결되지 않는다. 기본값은 `global.anthropic.claude-sonnet-4-6`으로 설정되어 있다.

앞의 `global.`은 교차 리전 추론 프로파일 접두어이다. 최근 모델은 기본 모델 ID만으로는 호출되지 않고 이 접두어가 필요하다. 리전을 고정해야 하면 `us.`·`eu.`·`apac.`·`jp.`로 변경한다. 규제에 따라 데이터가 특정 지역을 벗어날 수 없는 경우가 이에 해당한다. 다만 지역 지정 엔드포인트는 요금이 10% 높다.

계정에서 사용할 수 있는 모델은 리전마다 다르므로 `KDSM_LLM_MODEL`로 재정의할 수 있다.

```
export KDSM_LLM_MODEL=apac.anthropic.claude-opus-4-6-v1
```

접두어를 누락하여 호출이 실패하면 그 사실을 한 줄로 다시 안내한다. Bedrock이 반환하는 원문은 길어서 무엇을 수정해야 하는지가 드러나지 않기 때문이다.

자격 증명은 액세스 키 한 쌍 외에 베어러 토큰(`AWS_BEARER_TOKEN_BEDROCK`)이나 `AWS_PROFILE`로도 지정할 수 있다. 키를 지정하지 않으면 SDK가 AWS의 기본 자격 증명 체인을 따른다. 이 체인을 별도로 구현하지 않았다.

```
- 신규 표준단어 제안 (LLM · 확정 아님)
  * 데이터마트 (조각을 합침)
    Data Mart  DMART  02 도메인어  상태 01 후보
    정의: (data mart) 특정 주제 영역의 자료를 이용 목적에 맞게 모아 둔 저장소
    근거: '마트'는 단독으로 쓰이지 않는다. 사전에 '메타데이터'가 한 단어로 등재되어 있다.
    → MT_STD_WORD 에 상태 01(후보)로 직접 추가한 뒤 확정하세요
```

! 제공받은 초안을 그대로 사용하지 않는다

LLM은 확정 표준단어 4,143개의 영문약어를 알지 못한다. 그래서 약어 후보를 세 개 제공받아 코드가 사용되지 않은 것을 선택한다. 사전 전체를 프롬프트에 포함하여 전송하는 방식보다 비용이 적고 결과가 확실하다.

제공받은 초안은 기존 규칙으로 다시 검사한다. 사용자에게 제시되는 것은 그 검사를 통과한 결과뿐이다.

```
× '창고'은(는) 이미 표준단어로 등재되어 있습니다
× 영문약어 'CUST'가 이미 쓰이고 있습니다
× 정의가 '(한자 또는 영어) 뜻풀이' 형식이 아닙니다
```

형식만 어긋난 항목은 차단하지 않고 코드에서 보정한다. 영문명의 대소문자가 이에 해당한다.

```
· 영문명 표기를 규칙에 맞춰 'DATA MART' → 'Data Mart'
```

### 12.4.3.2 자동으로 채우는 항목

표 12.7. LLM이 채우는 항목과 규칙

| 항목 | 규칙 |
|---|---|
| `WORD_ENG_NM` | 낱말마다 첫 글자만 대문자 |
| `WORD_ENG_ABBR_NM` | 기존 약어와 중복되지 않는 것을 코드가 선택한다 |
| `WORD_EXPLN` | `(한자 또는 영어) 뜻풀이`. 한자어는 한자, 외래어는 원어 |
| `WORD_SE_CD` | 일반어(01) 또는 도메인어(02). 애매하면 01 |
| `ALLO_SYNM_CN` · `PHWRD` | 떠오르는 이음동의어·금칙어 |
| `STD_DMN_CLSF_CD` | 비운다. 분류어는 이 기능에서 제안하지 않는다 |
| `STTS_CD` | 언제나 `01` 후보 |
| `SRC` | `LLM`. 이후 자동 생성된 단어만 별도로 조회할 수 있다 |

> ⚠ 조각을 그대로 등재할지부터 묻는다
>
> "마트"는 단독으로 사용되지 않는다. 실무에서는 "데이터마트"가 하나의 단위이며, 사전에 이미 "메타데이터"가 한 단어로 등재된 선례가 있다.
>
> 그래서 조각만 전달하지 않고 원래 용어명과 분해 결과를 함께 전달한다. 인접한 조각과 결합해야 하는 경우에는 결합한 형태를 반환하도록 하기 위해서이다. 조각을 기계적으로 채우는 방식보다 이 방식이 유용하다.
>
> 이미 같은 뜻의 단어가 있으면 새로 만들지 않도록 지시한다. 이음동의어를 늘리는 것이 이 사전에서 가장 피해야 할 사항이기 때문이다.

#### 12.4.3.3 제안 단어로 만들어지는 용어까지

단어만 제안하고 종료하면 그 단어로 어떤 용어가 구성되는지를 사용자가 직접 판단해야 한다. 제안 단어를 사전에 등재하였다고 가정하고 추천기를 한 번 더 실행하여 그 결과까지 목록에 포함한다.

```
  기호  * 그대로 등재 가능   + LLM이 제안한 단어를 써야 함 ([ ] 안이 그 단어)   (없음) 그 밖의 문제

# 메타데이터마트이름
  +  110.0  META_DMART_NM  [명V5]  Meta Data Mart Name
       구성: 메타 + [데이터마트] + 이름
       · 먼저 표준단어 '데이터마트'를 등재해야 합니다
  *   60.0  MTDT_NM       [명V5]  Metadata Name
       구성: 메타데이터 + 이름
       · 표준단어가 아닌 조각: 마트
```

제안 단어를 사용한 후보가 상위에 표시된다. 미등록 단어가 없어져 감점이 적용되지 않기 때문이다. 이는 의도한 결과이다. "이 단어를 등재하면 이 용어가 된다"는 것이 현 시점에서 제시할 수 있는 가장 적절한 선택이기 때문이다.

! 사전에 있는 단어로 만든 후보와 구분한다

`*`가 아니라 `+`로 표시한다. 분해와 도메인에는 문제가 없으나 구성 단어가 아직 사전에 없다. 그대로 등재하면 `term.words-registered` 위반이다.

두 표시를 구분하지 않으면 즉시 등재할 수 있는 후보와 단어를 먼저 등재해야 하는 후보가 섞여 보이다. 구성 줄에서도 제안 단어만 대괄호로 표시한다.

표 12.8. 후보 목록의 표시

| 표시 | 뜻 |
|---|---|
| `*` | 사전에 있는 단어로만 구성된 후보이다. 그대로 등재해도 된다 |
| `+` | LLM이 제안한 단어를 사용한다. 그 단어를 먼저 등재해야 한다 |
| 없음 | 미등록 단어가 남았거나 물리명이 충돌한다 |
| `[단어]` | 아직 사전에 없는 제안 단어 |

`TermCandidate.provisional_words`에 그 단어가 담긴다. 점수와 무관하게 별도로 제시되어야 하는 정보이므로 이유 문구가 아니라 필드로 두었다.

```
from kdsm.suggest.coining import assemble

assemble(suggester, "메타데이터마트이름", drafts)   # 제안 단어를 쓴 후보만
```

i 화면에만 출력하는 이유

시트에 기록하는 작업은 사람의 판단이 선행되어야 한다. 무엇을 등재할지 결정하지 않은 상태에서 행이 늘어나면, 이후에 그 행이 사람이 결정한 것인지 시스템이 추가한 것인지 구분할 수 없다.

`validate`·`load`·`ddl`에는 이 기능을 연결하지 않았다. 검증은 사실 확인이고 제안은 새로운 생성이다. 두 가지를 섞으면 CI를 실행할 때마다 결과가 달라진다.

### 12.4.4 후보가 없는 경우

표준단어를 하나도 찾지 못하면 후보를 생성하지 않는다. 미등록 단어만으로 구성된 결과를 후보로 제시하는 것은 도움이 되지 않는다.

대신 후보를 생성하지 못한 이유를 안내한다. 안내 없이 빈 결과만 반환하면 사전에 없는 낱말인지 사용이 금지된 낱말인지 확인할 수 없다.

```
# 패스워드
  후보를 만들지 못했습니다.
    · '패스워드'는 이음동의어. 대체어가 여럿이라 골라야 함: 비밀번호(PSWD) · 암호(PW)
```

```
suggester.explain("패스워드")
```

## 12.5 쓰는 법

결과를 읽는 방법, 여러 이름을 일괄 처리하는 방법, 커맨드라인 사용법을 설명한다.

### 12.5.1 결과 읽기

```
# 고객여신금액
  *  110.0  CUST_LAN_AMT  [금액N13]  Customer Loan Amount
      구성: 고객 + 여신 + 금액
      · 도메인 후보: 금액N13 · 금액N15 · 금액N17 · 금액N18 · 금액N22,2
```

앞의 `*`는 그대로 등재해도 되는 후보라는 표시이다. 물리명 충돌이 없고 표준도메인이 정해진 경우이다.

```
candidate.ok              # 충돌 없고 도메인 있음
candidate.reasons         # 점수 근거와 주의 사항
candidate.rewritten_from  # 금지어를 바꿔 만든 후보면 원래 이름
candidate.to_row()        # 메타파일 MT_STD_TRM 행으로
```

이름이 변경된 후보는 `이름:` 줄로 표시한다. 이름이 달라졌다는 사실은 반드시 확인되어야 하므로 점수나 근거에 섞지 않고 별도로 표시한다.

### 12.5.2 일괄 처리

```
result = suggester.suggest_many(["고객여신금액", "계약해지일자", "상품가격"])
```

```
고객여신금액 → CUST_LAN_AMT [금액N13]
계약해지일자 → CTRT_CNCLTN_YMD [연월일C8]
상품가격     → GDS_PRC [가격N9,2]
```

### 12.5.3 커맨드라인

```
kdsm suggest 고객여신금액 상품가격 --meta dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx -l 3
kdsm suggest 고객여신금액 --from-db --coin      # 적재된 사전에서, 미등록 단어는 LLM 초안까지
```

용어명을 여러 개 지정할 수 있다. 후보를 하나도 생성하지 못하면 종료 코드는 1이다.

표 12.9. kdsm suggest 옵션

| 옵션 | 설명 |
|---|---|
| `<용어명>` | 추천받을 한글 용어명. 여러 개를 나열할 수 있다 |
| `--meta` | 사전을 읽을 메타파일. `--from-db`와 함께 쓸 수 없다 |
| `--from-db` | 적재된 `kdsm_meta` 테이블에서 사전을 읽는다 |
| `--schema` | 사전이 있는 스키마명. `--from-db`일 때만 쓴다 |
| `-l`, `--limit` | 이름마다 제시할 후보 수 (기본 3) |
| `--coin` | 미등록 단어의 등재 초안을 LLM에게 받는다(12.4.3절) |
| `--llm-provider` | 초안을 받을 곳. `anthropic` 또는 `bedrock` |

사전을 어디서 읽을지는 `--meta`와 `--from-db` 둘 중 하나로 반드시 지정한다. 기본값을 두지 않는 이유는 어느 사전으로 추천했는지가 결과의 뜻을 바꾸기 때문이다. 메타파일은 작성 중인 사전이고 데이터베이스는 적재를 마친 사전이라, 둘의 내용이 다를 수 있다.

> ⚠ 추천은 초안일 뿐
>
> 도메인 선택과 이음동의어 판단은 사람이 수행해야 한다. 추천기는 사전에 등재된 단어를 조합할 뿐이며, 그 조합이 업무상 적절한 이름인지는 판단하지 않는다.

CHAPTER 13

# 검증

검증 대상은 배포 산출물 3종이다. 데이터 표준 정의(DM-DST-001), 논리 데이터 모델(DM-DST-002), 물리 데이터 모델(DM-DST-003)이다. 목적은 하나이다. 해당 산출물을 메타 테이블에 적재할 수 있는지를 적재 이전에 판정한다.




**그림 13.1.** 검증 규칙이 놓인 자리

그림 13.1 는 검증 규칙이 어느 대조에 놓여 있는지를 나타낸다. 산출물 안에서만 판정하는 것, 두 산출물 사이를 대조하는 것, 사전과 대조하는 것 셋이다.

**표 13.1.** 검증 대상과 목적

| 산출물 | 검사 대상 | 무엇을 묻는가 |
|---|---|---|
| DM-DST-001 | 표준단어·표준용어·표준도메인·도메인그룹·분류·표준코드 | 사전 테이블에 적재할 대상이 무결한가. 기존 등재분과 중복되지 않는가 |
| DM-DST-002 | 엔터티·속성 | 논리 모델 테이블에 적재할 대상이 무결한가 |
| DM-DST-003 | 데이터베이스·테이블·컬럼 | 물리 모델 테이블에 적재할 대상이 무결한가. 논리 모델과 상충하지 않는가 |

비교 기준은 사전이다. 사전은 기본적으로 데이터베이스의 `kdsm_meta` 스키마에서 조회한다. 검증의 목적이 현재 적재 가능 여부의 판정이므로 현재 등재된 내용과 대조하는 것이 타당하기 때문이다. 데이터베이스를 아직 구축하지 않았거나 사전을 수정하는 중이어서 파일이 최신인 경우에는 `--meta`로 메타파일을 지정한다(13.2절).

다만 적재 시점의 중복 여부는 메타테이블만 판정할 수 있다. 따라서 메타파일을 사전으로 지정하면 해당 검사는 수행하지 않는다(13.3.1.2절).

검증 규칙이 사전 다섯, 메타파일 헤더, 논리 모델, 물리 모델, 두 정의서 사이의 대조, 그리고 신규 등재분과 기존 사전의 대조를 대상으로 정의되어 있다. 규칙의 전체 목록과 수는 `library/docs/validation-rules.md` 에 있다.

**표 13.2.** 검증 규칙 묶음과 규칙 수

| 대상 | 규칙 묶음 | 수 |
|---|---|---|
| 표준단어 | `word_rules()` | 8 |
| 표준용어 | `term_rules(words, domains, codes)` | 8 |
| 표준도메인 | `domain_rules(classifications)` | 4 |
| 도메인그룹·분류 | `domain_group_rules(classifications)` | 9 |
| 표준코드 | `code_rules(details)` | 9 |
| 메타파일 헤더 | `meta_header_rules(terms, exempt, tables)` | 4 |
| 논리 모델 엔터티 | `entity_rules(terms, details, prefix=)` | 7 |
| 논리 모델 속성 | `attribute_rules(entities, terms, domains, details)` | 10 |
| 물리 모델 컬럼 | `column_rules(terms, domains, tables, external, data)` | 9 |
| 물리 모델 테이블 | `table_rules(columns, databases, entities)` | 5 |
| 물리 모델 데이터베이스 | `database_rules(tables)` | 2 |
| 엔터티 ↔ 테이블 | `table_pairing_rules(entities, attributes)` | 6 |
| 속성 ↔ 컬럼 | `column_pairing_rules(attributes, tables)` | 10 |
| 신규 등재분 ↔ 메타테이블 | `already_registered_rules(reference, model, fields)` | 1 |
| 신규 등재분 읽기 | 규칙 묶음이 아니라 읽기 단계 | 1 |

마지막 행의 `standard.unreadable`은 규칙 묶음이 아니라 정의서를 읽는 단계에서 나온다. 헤더가 다르거나 필수 항목이 비어 모델로 옮기지 못한 행을 알린다. 그 행은 이후 규칙에서 검사되지 않으므로, 알리지 않으면 검사한 것으로 오해하게 된다.

규칙별 상세는 `library/docs/validation-rules.md`에 있다.

## 13.1 설계 원칙

### 13.1.1 규칙은 목록 단위로 동작한다

규칙 하나는 대상 목록 전체를 입력받아 위반 목록을 반환하는 호출 가능 객체이다. 유일성이나 중복과 같은 교차 검사가 많으므로 항목 단위가 아니라 목록 단위로 받는다.

```
report = Validator(word_rules()).add(my_rule).run(words)
```

`Rule` 프로토콜만 만족하면 사내 규칙을 추가할 수 있다.

### 13.1.2 심각도는 생성 시점에 지정한다

무엇을 오류로 판정할지는 조직마다 다르다.

```
from kdsm.validate.report import Severity
from kdsm.validate.rules import UniqueAbbreviationRule

rules = [r for r in word_rules() if r.name != "word.abbr-unique"]
rules.append(UniqueAbbreviationRule(severity=Severity.WARNING))
```

### 13.1.3 INFO는 판정이 아니라 기록이다

심각도는 셋이다. ERROR는 반드시 고쳐야 할 위반, WARNING은 검토가 필요한 것, INFO는 판정하지 않았다는 기록이다.

INFO는 종료 코드에 영향을 주지 않으며 `--strict`에도 해당하지 않는다. 사용하는 경우는 둘이다.

첫째, 규칙이 특정 항목을 검사할 수 없어 생략하였을 때 그 사실만 기록한다.

```
정보: 속성명 '생성자': 표준단어 하나가 그대로 용어가 되어 분류어 종결을 검사하지
      않았습니다. 쪼갤 여지가 없어 예외로 둔 것이며 위반이 아닙니다.
```

둘째, 산출물의 내용이 아니라 그 문서를 다루는 방식에 관한 지적이다. 파일명과 표지의 대조가 여기에 해당한다(13.3.11절). 내용은 표준을 지키고 있으므로 적재를 막을 이유가 없으나, 알리지 않으면 어긋난 채로 배포된다.

생략한 항목을 안내 없이 두면 결과만 확인하는 사람은 검사를 통과한 것으로 해석한다. 이 기준은 매뉴얼 전반에 반복하여 적용한다. 참조 사전이 없으면 규칙을 묶음에서 제외하는 것도(13.1.5절), 길이를 재지 못한 항목을 따로 밝히는 것도(18.5.5절) 같은 이유이다.

### 13.1.4 폐기 항목은 검사하지 않는다

모든 규칙이 상태 폐기(03) 항목을 제외한다. 필요하면 포함할 수 있다.

```
UniqueAbbreviationRule(include_discarded=True)
```

### 13.1.5 참조 사전이 없으면 규칙이 빠진다

> **! 중요**
>
> 다른 사전을 참조해야 검사할 수 있는 규칙은 참조를 지정하지 않으면 묶음에서 제외된다. 검사하지 않은 항목을 통과한 것으로 오인하지 않도록 하기 위해서이다.

```
term_rules()                          # 3종
term_rules(words)                     # 6종
term_rules(words, domains)            # 7종
term_rules(words, domains, codes)     # 8종
```

### 13.1.6 사전에서 끊긴 지점은 앞선 규칙이 보고한다

컬럼 규칙 다섯 종은 컬럼 → 표준용어 → 표준도메인의 두 단계를 거친다. 어느 단계에서 연결이 끊겼는지에 따라 보고하는 규칙이 다르며, 후행 규칙은 같은 문제를 중복하여 보고하지 않고 건너뛴다.

컬럼 하나가 미등재일 때 위반이 다섯 건씩 중복하여 보고되지 않도록 한 것이다.

## 13.2 사전의 출처

검사 대상과 비교 기준을 구분한다. 대상은 산출물이고, 사전은 이를 판정하는 기준이다. 사전은 이미 검토를 거쳐 적재된 것이므로 검사 대상에 포함하지 않는다.

사전의 출처는 둘이다.

표 13.3. 사전의 두 출처

| 출처 | 지정 방법 | 사용 시점 |
|---|---|---|
| 데이터베이스 `kdsm_meta` | 기본 | 운영 중인 사전과 대조할 때. 검증의 목적이 적재 가능 여부의 판정이므로 이쪽을 기본으로 한다 |
| 메타파일 | `--meta` | 데이터베이스를 아직 구축하지 않았거나, 사전을 수정하는 중이어서 파일이 최신일 때 |

```
kdsm validate DM-DST-003_*.xlsx                    # 사전은 DB에서
kdsm validate DM-DST-003_*.xlsx --meta kdsm_meta.xlsx  # 사전은 파일에서
kdsm validate DM-DST-003_*.xlsx --host db.internal --database aiplatform
```

접속 정보는 `--host` · `--port` · `--database` · `--schema` · `--user`로 덮어쓸 수 있다. 비밀번호만은 인수로 받지 않는다. 명령줄에 적으면 `ps`와 셸 기록에 남기 때문이다. `KDSM_DB_PASSWORD`를 쓴다(20.10절).

```
from kdsm.db import MetaDatabase
from kdsm.io import MetaWorkbook

with MetaDatabase.from_env() as dictionary:   # 1안
    ...

dictionary = MetaWorkbook.open("kdsm_modeling_meta_20260828.xlsx")   # 2안
```

두 원천은 동일한 `read_*` 메서드를 갖추고 있다. 따라서 원천을 선택한 이후의 코드는 어느 쪽을 전달받았는지 알 필요가 없다.

> **i 사전 없이 검사할 수도 있다**
>
> `--no-dict`를 지정하면 산출물 내부의 무결성만 검사한다. 사전을 참조해야 하는 규칙은 묶음에서 제외된다(13.1.5절). 아직 사전을 구축하지 않은 환경에서 정의서의 구조만 확인할 때 사용한다.
>
> 상태·단어 구분·용어 구분은 사전 없이도 판별한다. 이 셋은 KDSM이 자체적으로 정의한 것이므로 프로젝트가 변경하지 않기 때문이다. 이것까지 사전에서 조회하도록 하면 `--no-dict`를 지정하였을 때 모든 행을 읽지 못하여, 수행하려던 검사가 하나도 실행되지 않는다.

> **i 사전 자체를 검사할 수도 있다**
>
> 산출물 없이 `--meta`만 지정하면 그 메타파일 자체를, `--from-db`를 지정하면 데이터베이스에 적재된 사전 자체를 검사한다. 이때 사전이 대상이자 기준을 겸하므로 적재 충돌 검사(13.3.1.2절)는 수행하지 않는다.
>
> ```
> kdsm validate --meta dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx   # 메타파일 자체
> kdsm validate --from-db                                                     # 적재된 사전 자체
> ```
>
> 두 경로는 사전 다섯 벌에 같은 규칙 묶음을 적용하며, 그 구성의 정본은 `validate_dictionary()`다. 기준선 차감(13.3.1.1절)도 같은 함수로 기준선을 산출하므로, 사전만 검사한 결과와 산출물 검증이 본 기존 위반이 어긋나지 않는다. `--from-db`는 산출물·`--meta`·`--no-dict`와 함께 지정할 수 없다.

## 13.3 무엇을 검사하는가

산출물 3종을 각각 어떤 규칙으로 검사하는지, 그리고 정의와 실제 데이터를 대조하는 규칙을 설명한다.

### 13.3.1 데이터 표준 정의 검증

DM-DST-001은 사전에 신규 등재할 대상을 수록한다. 검사는 두 계층으로 구성한다.

표 13.4. 검증의 두 계층

| 계층 | 판정 내용 | 비교 기준 |
|---|---|---|
| 표준 준수 | KDSM 규칙의 준수 여부 | 사전. 메타파일과 메타테이블 모두 가능 |
| 적재 가능성 | 현재 적재 가능 여부 | 메타테이블 |

두 계층을 구분하는 이유는 판정 근거가 다르기 때문이다. 「영문약어는 유일해야 한다」는 적재 이전의 표준 규칙이므로 메타파일로도 판정할 수 있다. 반면 「기본키가 중복되는가」는 적재 대상 테이블만 판정할 수 있다.

```
from kdsm.db import MetaDatabase
from kdsm.io import SpecWorkbook
from kdsm.io.standard import StandardCodec
from kdsm.validate.standard import validate_standard

with MetaDatabase.from_env() as tables:
    codec = StandardCodec.of(tables)
    found = SpecWorkbook.open("DM-DST-001_....xlsx").read_standard(codec)
    report = validate_standard(found, tables, loaded=tables)
```

두 인자에 동일한 객체를 전달한 것은 이 경우 비교 기준과 적재 대상이 같기 때문이다. `--meta`로 메타파일을 사전으로 지정하였다면 `loaded`는 전달하지 않는다.

#### 13.3.1.1 결합 검사와 기준선 차감

유일성 규칙은 하나의 목록 내에서만 중복을 검사한다. 따라서 신규 표준단어의 영문약어가 사전에 등재된 단어와 중복되는지 확인하려면 두 집합을 하나의 목록으로 결합하여 검사해야 한다.

그러나 결합하여 검사하면 기존 사전 내부의 기존 위반까지 함께 보고된다. 4천 행 규모의 사전에서 수백 건이 보고되면 이번 산출물이 유발한 위반을 식별하기 어렵다. 그래서 기존 사전만으로 검사한 결과를 기준선으로 삼고, 결합하여 검사한 결과에서 기준선을 차감한다. 차감 후 남는 것이 이번 산출물이 유발한 위반이다.

```
from kdsm.validate.report import added

fresh = added(_internal(reference).issues, _internal(merged).issues)
```

이 방식의 이점은 규칙이 추가되어도 수정할 곳이 없다는 점이다. 규칙마다 신규 항목에 기인한 위반인지를 개별적으로 판단하도록 구현하면, 규칙을 추가할 때마다 그 판단 로직을 함께 작성해야 한다.

#### 13.3.1.2 기존 등재분과 중복되는 경우

기준선 대조만으로는 판별하지 못하는 경우가 있다. 이름이 동일할 때, 그것이 동일한 항목을 중복 기재한 것인지 동일한 이름으로 다른 의미를 정의하려는 것인지이다. 유일성 규칙은 두 경우를 모두 「중복 등재」로만 보고한다.

`standard.already-registered`가 두 경우를 구분한다. 이 규칙의 비교 대상은 항상 메타테이블이다.

> ! 이 검사의 비교 대상은 메타테이블이다
>
> 이 규칙이 판정하는 것은 적재 시점의 기본키 중복 여부이며, 이는 적재 대상 테이블만 판정할 수 있다.
>
> 메타파일을 기준으로 삼으면 두 가지 오판이 발생한다. 첫째, 파일에는 존재하지만 테이블에는 존재하지 않는 표준을 이미 등재된 것으로 간주하여 적재를 차단한다. 둘째, 테이블에만 존재하는 항목과의 중복을 검출하지 못하여 적재 시점에 기본키 제약 위반이 발생한다. 메타파일은 담당자가 직접 수정하는 원본이므로 테이블보다 최신인 경우가 일반적이며, 두 원천의 불일치는 드물지 않다.
>
> 따라서 `--meta`로 메타파일을 사전으로 지정하면 이 검사를 수행하지 않는다. 대신 수행하지 않았다는 사실을 INFO로 기록한다(13.1.3절).
>
> ```
> [INFO] standard.already-registered — 1건
>   적재 충돌: 메타테이블을 조회하지 않아 적재 가능 여부는 검사하지 않았습니다.
>             기존 등재분과의 중복은 적재 대상 데이터베이스만 판정할 수 있습니다.
> ```
>
> `kdsm apply`는 메타파일을 인수로 받지 않는다. 테이블에 적재하는 작업이므로 비교 기준도 적재 대상이어야 하기 때문이다(15.5절).

표 13.5. 중복의 세 가지 유형

| 구분 | 심각도 | 보고 내용 |
|---|---|---|
| 이름과 내용이 모두 동일하다 | WARNING | 동일한 내용이 이미 존재하므로 기본키 중복으로 적재할 수 없다 |

| 구분 | 심각도 | 보고 내용 |
|---|---|---|
| 이름은 동일하나 내용이 상이하다 | ERROR | 상이한 항목의 테이블 값과 정의서 값을 함께 제시한다 |
| 이름은 신규이나 약어가 중복된다 | ERROR | `word.abbr-unique`가 검출한다. 기준선 대조의 영역이다 |

```
[WARNING] standard.already-registered — 1건
  가격: '가격'은(는) 동일한 내용이 이미 메타테이블에 존재합니다. 기본키(Primary Key)
        중복으로 인해 적재할 수 없습니다.

[ERROR] standard.already-registered — 1건
  가격: '가격'은(는) 이미 메타테이블에 존재하며 내용이 상이합니다:
        WORD_ENG_ABBR_NM 메타테이블 'PRC' ≠ 정의서 'GAGYEOK'
```

내용까지 동일한 경우를 오류가 아닌 경고로 판정하는 이유는, 폐기한 항목을 재등재하려는 의도일 수 있기 때문이다. 담당자가 판단할 여지를 남긴다.

> **i** 비교에서 제외하는 항목이 둘 있다
>
> 상태는 비교하지 않는다. 신규 등재분은 대개 후보(01)이고 기존 등재분은 확정(02)이므로, 상태를 비교 항목에 포함하면 모든 항목이 상이한 것으로 보고된다. 상태의 차이는 의미의 차이가 아니라 검토 진행 단계의 차이이다.
>
> 정의서에 해당 항목이 없는 값도 비교하지 않는다. 도메인분류의 소속 그룹명이 이에 해당한다. 메타파일은 이 값을 함께 기재하지만 정의서에는 해당 항목이 없어 항상 빈 값으로 읽힌다. 이를 비교 항목에 포함하면 도메인분류를 재등재할 때마다 실재하지 않는 차이가 보고된다. 기재할 항목이 없어 비어 있는 것을 삭제한 것으로 판정하는 셈이다.

#### 13.3.1.3 정의서를 읽지 못한 경우

읽지 못한 행은 검사와 적재 모두 수행하지 않는다. 이를 별도로 보고하지 않으면 「오류 0건」이라는 요약이 출력되어, 검사하지 않았다는 사실이 통과로 해석된다. 그래서 `standard.unreadable`이라는 이름으로 동일한 리포트에 오류로 수록한다.

```
[ERROR] standard.unreadable — 1건
  표준용어 정의서 머리글: '영문명' 칸이 없습니다. 이 칸의 값은 읽지 않았습니다.
```

머리글을 먼저 확인하는 이유는, 머리글 한 항목이 서식과 다르면 해당 항목을 사용하는 모든 행이 동일한 원인으로 실패하기 때문이다. 시트의 행 수만큼 같은 문장이 반복되면 다른 행의 실제 문제를 식별하기 어렵다. 머리글을 한 번 보고한 뒤에는 그 원인만으로 실패한 행은 다시 보고하지 않는다.

따라서 위반이 대량으로 보고되면 머리글 보고가 있는지부터 확인한다. 머리글 하나를 고치면 그 아래의 위반이 함께 사라지는 경우가 많다.

### 13.3.2 논리 데이터 모델 검증

DM-DST-002는 엔터티와 속성을 수록한다. 사전과 대조하는 규칙까지 모두 수행한다.

```
from kdsm.io.standard import MODEL_CODE_LABELS, StandardCodec

codec = StandardCodec.of(dictionary, MODEL_CODE_LABELS)
```

```
book = SpecWorkbook.open("DM-DST-002_....xlsx")

Validator(entity_rules(terms, details, prefix="AP_")).run(book.read_entities(codec))
Validator(attribute_rules(entities, terms, domains, details)).run(
    book.read_attributes(codec)
)
```

⚠ 코덱을 지정하지 않으면 코드 검사가 전부 실패한다

같은 사실을 정의서와 메타테이블이 서로 다른 표기로 기재한다. 정의서는 사람이 읽는 문서이므로 `속성 유형`에 `기본형`이라고 적고, 메타테이블은 코드값 `01`을 저장한다.

표 13.6. 같은 사실의 두 표기

| 위치 | 기재 |
|---|---|
| DM-DST-002 「속성」 정의서 | `기본형` |
| `MT_ATRB.ATRB_TYPE_CD` | `01` |
| `MT_STDCD_DTL` | 코드값 `01` ↔ 코드값명 `기본형` |

검증 규칙은 메타테이블의 표기를 기준으로 판정한다. `attribute.type-code`는 「속성 유형이 `MT_STDCD_DTL`에 등재된 코드값인가」를 확인하므로, 정의서에 적힌 `기본형`을 그대로 넘기면 코드값 목록에 없는 값이 되어 모든 속성이 위반으로 보고된다. 정의서가 서식대로 올바르게 작성되었더라도 그러하다.

코덱은 그 두 표기를 잇는 대응표이다. `read_attributes(codec)`처럼 읽는 시점에 넘기면 `기본형`을 `01`로 바꾸어 담으므로 규칙이 정상적으로 판정한다. 코드명으로 기재하는 항목은 셋이며, 대응하는 규칙은 둘이다.

표 13.7. 코덱이 변환하는 항목

| 항목 | 표준코드 | 해당 규칙 |
|---|---|---|
| 엔터티 구분 | `ENTTY_CLSF` | `entity.classification-code` |
| 속성 유형 | `ATRB_TYPE` | `attribute.type-code` |
| 발생주기 | `DT_OCRN_CYCLE` | 없음 |

`kdsm validate`는 코덱을 자동으로 구성하여 넘긴다. 따라서 이 주의는 위의 예시처럼 라이브러리를 직접 호출하는 경우에만 해당한다.

### 13.3.3 산출물 3종을 함께 지정하는 경우

DM-DST-001에 기재한 내용이 이어지는 논리·물리 모델 검증의 기준이 된다. 신규 표준용어는 아직 사전에 등재되지 않았으므로, 정의서를 결합하지 않으면 해당 용어를 사용한 컬럼이 모두 미등재로 보고된다.

```
kdsm validate DM-DST-001_*.xlsx DM-DST-002_*.xlsx DM-DST-003_*.xlsx \
  --meta dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx --name-prefix AP_
```

3종을 함께 지정한다는 것은 이 셋이 하나의 집합을 이룬다는 의미이다. 따라서 001을 먼저 읽어 사전에 추가하고, 추가된 사전을 기준으로 002와 003을 검사한다. 산출물의 종류는 시트 구성으로 판별하므로 파일명과 지정 순서에는 제약이 없다.

### 13.3.4 메타파일 검증

```
from kdsm.io import MetaWorkbook
from kdsm.validate import Validator, meta_header_rules, validate_dictionary

wb = MetaWorkbook.open("dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx")
report = validate_dictionary(wb)                          # 사전 다섯 벌
Validator(meta_header_rules(wb.read_terms())).run(wb.read_headers())   # 헤더
```

사전 다섯 벌의 규칙 구성은 `validate_dictionary()`가 한 곳에 담고 있다(13.2절). 메타파일 고유의 검사는 헤더뿐이며, 헤더는 적재될 테이블의 열 이름이므로 `meta_header_rules`로 따로 검사한다.

### 13.3.5 논리 모델 검증

논리 데이터 모델 메타파일(`MT_ENTTY`·`MT_ATRB` 시트)은 별도 파일이다. 사전 메타파일에는 이 시트가 없으므로 따로 연다.

```
from kdsm.validate import attribute_rules, entity_rules

logical = MetaWorkbook.open("dbms/meta/meta_data/kdsm_logical_meta_20260828.xlsx")
entities = logical.read_entities()
attributes = logical.read_attributes()

Validator(entity_rules(terms, wb.read_code_details(), prefix="MT_")).run(entities)
Validator(attribute_rules(entities, terms, domains, wb.read_code_details())).run(attributes)
```

두 층위를 구분하여 적용하는 것이 이 묶음의 핵심이다. 엔터티명은 표준용어의 엔터티명(용어구분 01), 속성명은 속성명(02)이어야 한다. 겸용(03)은 양쪽 모두에 사용할 수 있다.

논리 메타파일도 KDSM 메타파일이므로 `kdsm validate --logical`은 이 파일의 헤더에도 `meta_header_rules`를 적용한다. 헤더의 컬럼 물리명은 한 글자만 달라도 육안으로 발견하기 어렵고, 파일을 읽는 데에는 문제가 없어 적재 시점에야 드러난다. `PRVC_INCL_YN`을 `PRVC_ENCL_YN`으로 기록하는 것과 같은 형태이다. 헤더 검사는 반드시 거친다.

> **i 같은 사실을 두 곳에 기록하면 불일치가 발생한다**
>
> 기본키는 `MT_ENTTY.PK`에 이름을 이어 붙여 기록하고, `MT_ATRB.PK_SN`에 순번으로 다시 기록한다. 한쪽만 수정하면 두 값이 어긋나는데, 적재하면 양쪽 모두 반영되므로 이후에 어느 쪽이 정확한지 판별할 수 없다. `attribute.primary-key`는 순번으로 기본키를 구성하여 표기와 대조한다. 유일키도 같은 구조이다. `MT_ENTTY.UK`의 선언과 `MT_ATRB.UK_CN`의 표기가 짝이며, `attribute.unique-key`가 표기로 유일키를 구성하여 선언과 대조한다(9.3.4절).
>
> `PK` 표기는 `+`와 쉼표를 모두 허용한다. 두 표기 모두 의미가 분명하므로 어느 쪽으로 기재하여도 검사를 통과한다. 다만 한 산출물 안에서는 한 표기로 통일하는 편이 읽기 쉽다.

> 관계는 양쪽에 짝으로 기록된다
>
> `entity.relation-paired`는 `주요 관계`가 양쪽 엔터티에 서로 반대 방향으로 기록되어 있는지 확인한다. 부모가 `자식(1:N)`이면 자식은 `부모(N:1)`여야 한다. 양쪽 모두 `1:N`이면 어느 쪽이 부모인지 판별할 수 없다.
>
> 관계 표기는 참고 정보이며 적재를 중단시키지 않으므로 심각도는 경고이다.

> ⚠ 코드값 자리에 코드명을 기록하는 오류
>
> `ATRB_TYPE_CD`와 `ENTTY_SE_CD`에는 코드값(`01`)이 들어가야 하는데 코드명(`기본형`)을 기록하는 사례가 잦다. 적재하면 그 값이 그대로 반영되어 코드 조인이 성립하지 않는다. `attribute.type-code` 규칙은 기록된 값이 코드명과 일치하면 어느 코드값을 의도한 것인지 함께 제시한다.
>
> 이 때문에 `Entity.classification`은 열거형이 아니라 문자열로 받는다. 열거형으로 받으면 잘못된 코드가 있을 때 파일을 읽는 시점에 예외가 발생하여 검증이 첫 행에서 중단된다. 검증은 문제를 모아서 제시하는 작업이므로 판정은 규칙에 위임한다.

### 13.3.6 산출물 검증

```
from kdsm.io import SpecWorkbook
from kdsm.models import Column, Table
from kdsm.validate import column_rules, table_rules

spec = SpecWorkbook.open("DM-DST-003_물리데이터모델_....xlsx")
tables = [Table.model_validate(r) for r in spec.read_data("테이블 정의서")]
columns = [Column.model_validate(r) for r in spec.read_data("컬럼 정의서")]

Validator(column_rules(terms, domains, tables)).run(columns)
Validator(table_rules(columns)).run(tables)
```

### 13.3.7 물리 모델 검증

물리 데이터 모델 메타파일(`MT_DB`·`MT_TBL`·`MT_COL` 시트)도 별도 파일이다. 논리 메타파일이 DM-DST-002를 옮겨 담은 것이듯, 이 파일은 DM-DST-003을 옮겨 담은 것이다.

```
physical = MetaWorkbook.open("dbms/meta/meta_data/kdsm_physical_meta_20260828.xlsx")
databases = physical.read_databases()
meta_tables = physical.read_tables()
meta_columns = physical.read_columns()

Validator(database_rules(meta_tables)).run(databases)
Validator(table_rules(meta_columns, databases, entities)).run(meta_tables)
Validator(column_rules(terms, domains, meta_tables)).run(meta_columns)
```

> 정의서용 규칙을 그대로 사용한다
>
> `column_rules`·`table_rules`는 본래 산출물 정의서를 대상으로 작성한 것이지만 메타파일에도 그대로 사용한다. 메타파일이 정의서를 옮겨 담은 것이므로 검사할 내용이 동일하기 때문이다. 리더만 `SpecWorkbook`에서 `MetaWorkbook`으로 교체하면 된다.
>
> 같은 검사를 두 곳에 중복하여 작성하면 한쪽만 수정하는 문제가 발생한다.

메타파일에만 해당하는 규칙이 네 개이다. 데이터베이스 시트에 대응하는 검사는 산출물 쪽에 없고(`database.unique`·`database.has-tables`·`table.database-defined`), 관계엔터티이름은 논리 메타파일을 참조해야 판정할 수 있다(`table.entity-registered`). 산출물만 검사하면 이 네 규칙은 실행되지 않는다.

> ⚠ 모델에 포함하지 않은 컬럼
>
> `MT_TBL`의 `EXPC_NOCS`·`KPNG_PRD`·`DATA_OCRN_CYCLE_CD`·`BKUP_YN`은 `Table` 모델에 없다. DDL 생성과 검증에 사용되지 않는 운영 정보이므로 모델에 포함하지 않았다. 시트에는 그대로 남아 있으며 `meta-header` 규칙이 그 이름을 검사한다.

### 13.3.8 논리 모델과 물리 모델 대조

논리 모델 산출물(DM-DST-002)과 물리 모델 산출물(DM-DST-003)은 짝을 이룬다. 엔터티가 테이블에, 속성이 컬럼에 대응한다.

같은 사실을 두 문서에 각각 기록하므로 한쪽만 수정하는 일이 발생한다. Null 허용 여부가 대표적이다. 속성과 컬럼 중 어느 한쪽에서만 필수 여부를 바꾸면 두 문서가 갈리고, 파일을 읽는 데에는 문제가 없으므로 적재 시점까지 드러나지 않는다.

대조는 메타파일을 작성하기 전에 산출물끼리 수행한다. 메타파일은 산출물을 옮겨 적은 것이므로, 메타파일끼리 대조하면 이미 옮겨진 뒤의 결과를 보게 되어 어느 산출물이 원인인지 판별할 수 없다.

설명도 대조한다. 논리 모델에서 물리 모델을 생성할 때(`kdsm.derive`) 속성 설명을 컬럼 설명으로 그대로 옮기므로, 두 설명이 서로 다르면 어느 한쪽을 직접 수정한 것이다. `pairing.column-explanation`이 이를 오류로 보고하며, 컬럼 설명을 비워 둔 경우도 포함한다.

두 묶음은 `validate_pairing`이 하나의 구성으로 실행한다. `kdsm validate`와 산출물 적재가 이 함수를 함께 사용하므로, 검증 시점과 적재 시점의 대조 내용이 같다. 적재는 논리 모델 산출물과 물리 모델 산출물을 함께 지정한 경우에만 대조한다(15.5절). 한쪽만 지정하면 대조할 상대가 없으므로 실행하지 않는다.

```
from kdsm.io import SpecWorkbook
from kdsm.models import Column, Table
from kdsm.validate import column_pairing_rules, table_pairing_rules

logical = SpecWorkbook.open("DM-DST-002_논리데이터모델_....xlsx")
physical = SpecWorkbook.open("DM-DST-003_물리데이터모델_....xlsx")

entities = logical.read_entities()
attributes = logical.read_attributes()
tables = [Table.model_validate(r) for r in physical.read_data("테이블 정의서")]
columns = [Column.model_validate(r) for r in physical.read_data("컬럼 정의서")]

Validator(table_pairing_rules(entities, attributes)).run(tables)
Validator(column_pairing_rules(attributes, tables)).run(columns)
```

> ! 두 문서를 연결하는 기준은 `관련 엔터티`다
>
> 테이블 정의서의 `관련 엔터티` 항목이 두 문서를 연결하는 유일한 기준이다. 이름 규칙으로 유추하지 않는다. 유추하면 규칙을 위반한 경우와 의도적으로 다르게 명명한 경우를 구분할 수 없기 때문이다.
>
> 이 항목이 비어 있거나 존재하지 않는 이름을 가리키면 해당 테이블은 이후의 대조를 전혀 수행할 수 없다. `pairing.table-entity`가 이를 먼저 보고한다.

### 13.3.9 접두어는 어디에 담는가

KDSM은 메타데이터 테이블(Meta Table)에 `MT_`를 붙이다. 이 접두어를 어디에 담느냐에 따라 두 규칙의 형태가 결정된다.

접두어는 엔터티 영문명에 담는다. 그러면 물리 테이블명은 엔터티 영문명과 동일하기만 하면 된다.

```
표준용어 물리명   STD_WORD          ← 사전. 영문약어 조합
엔터티 영문명     MT_STD_WORD       ← 접두어 + 물리명
테이블 물리명     MT_STD_WORD       ← 엔터티 영문명 그대로
```

> ! 접두어를 사전에 담지 않는 이유
>
> `MT_`를 표준용어 물리명에 담으면 "물리명은 영문약어를 이어 붙인 것"이라는 규칙이 성립하지 않는다. `MT`는 표준단어가 아니므로 `term.physical-matches` 위반으로 검출된다.
>
> 물리 단계에서 붙이지 않는 이유는 이와 다르다. 메타 테이블이 아닌 엔터티가 포함되면 어느 것에 접두어를 붙이고 어느 것에 붙이지 않을지를 규칙이 판단할 수 없다. 논리 모델링 단계에서 정해 두면 그 판단이 문서에 기록된다.

따라서 `entity.english-name`이 접두어를 인식한다. 접두어를 제외한 나머지를 사전과 대조하므로 `MT_STANDARD_WORD` 같은 오기는 그대로 검출된다.

```
entity_rules(terms, details, prefix="MT_")     # 영문명 = MT_ + 물리명
table_pairing_rules(entities, attributes)      # 테이블명 = 영문명
```

접두어를 물리 단계에서 붙이는 조직에서는 반대로 지정한다.

```
entity_rules(terms, details)                           # 영문명 = 물리명
table_pairing_rules(entities, attributes, prefix="TB_")  # 테이블명 = TB_ + 영문명
```

커맨드라인은 `--name-prefix`로 받는다. 기본은 `MT_`이다.

> ⚠ 정의서는 코드값이 아니라 코드명을 기록한다
>
> 정의서는 사람이 읽는 문서이므로 `엔터티 구분`에 `기본`, `속성 유형`에 `기본형`과 같이 코드명을 기록한다. 메타파일은 코드값 `01`을 기록한다.
>
> `SpecWorkbook.read_entities(codec)`에 코덱을 전달하면 읽는 동안 코드값으로 되돌리므로 `entity.classification-code`를 그대로 적용할 수 있다. 전달하지 않으면 기록된 그대로 두는데, 그 상태에서 이 규칙을 적용하면 전부 위반으로 판정된다.
>
> 표기가 둘인 코드도 받는다. `ENTTY_CLSF`의 `01`을 사전은 `기본엔터티`, 정의서는 `기본`이라고 기록한다. 어느 쪽으로 기록되어 있든 같은 값으로 읽는다.

### 13.3.10 정의와 데이터를 대조하는 규칙

`column.not-null-satisfied`와 `column.length-satisfied`는 성격이 다르다. 나머지 규칙이 정의끼리 대조하는 데 비해, 이 두 규칙은 산출물의 정의와 메타파일의 실제 값을 대조한다.

```
from kdsm.io.excel import read_sheet_rows

meta = "dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx"
```

```
data = {s: read_sheet_rows(meta, s) for s in MetaWorkbook.open(meta).sheet_names()}

Validator(column_rules(terms, domains, tables, data=data)).run(columns)
```

> ! 적재하기 전에 실행해야 의미가 있다
>
> DBMS 가 `NotNullViolation` 이나 `StringDataRightTruncation` 으로 거부한 뒤에 확인하는 것보다, 트랜잭션을 시작하기 전에 확인하는 것이 적절하다. 이 두 유형은 적재를 중단시키는 원인 가운데 가장 빈번하다(21장).

두 규칙 모두 시트에 존재하지 않는 컬럼은 검사에서 제외한다. `CRT_DT`·`CREATR` 같은 감사 컬럼은 적재기가 채우므로 시트가 비어 있는 것이 정상이다.

`length-satisfied`는 문자 타입(VARCHAR, CHAR)만 검사한다. NUMERIC의 길이는 글자 수가 아니라 정밀도이다. 길이는 글자 수로 계산하는데, PostgreSQL의 `character varying(n)`도 바이트가 아니라 글자 수를 계산하므로 한글 한 자가 1이다.

### 13.3.11 파일명과 표지의 대조

문서번호·버전·작성일은 파일명과 표지 두 곳에 기록된다. 같은 사실을 두 곳에 기록하므로 어긋날 수 있다. 개정하면서 파일명만 변경하고 표지를 그대로 두는 경우가 가장 빈번하다.

```
DM-DST-001_데이터표준정의-KDSM-APP_V0.9.0_20260818.xlsx
└문서번호┘                          └버전┘ └작성일┘
```

`kdsm validate`는 산출물을 열 때 이 셋을 대조하고 어긋난 항목을 INFO로 기록한다.

```
[INFO] spec.filename-consistent — 2건
  고객주문_V0.9.4_20260810.xlsx: 버전이 다릅니다. 파일명 'V0.9.4' · 표지 'V0.9.3'
  고객주문_V0.9.4_20260810.xlsx: 작성일이 다릅니다. 파일명 '20260810' · 표지 '2026-08-04'
```

> i 이 항목은 검증 규칙에 포함하지 않는다
>
> 검증 규칙은 산출물의 내용이 표준을 지키는지 판정한다. 이 대조는 내용이 아니라 파일을 다루는 방식을 확인하는 것이며, 어긋나 있어도 적재에는 지장이 없다. 그러므로 심각도를 INFO로 두고 종료 코드에도 반영하지 않는다.
>
> 파일명이 규칙 형식이 아니면 대조하지 않고 그 사실만 기록한다. 형식을 모르는 이름을 해석하면 잘못된 위치를 비교하게 되기 때문이다. 표지를 읽지 못한 경우에도 검증을 중단하지 않는다.

문서명은 대조하지 않는다. 파일명은 `템플릿`을 제외하고 작성하는데 표지의 문서명에는 남아 있는 경우가 있어, 대조하면 수정할 필요가 없는 문서가 계속 보고된다. 버전의 `v`와 작성일의 붙임표는 표기 규칙의 차이이므로 동일한 형태로 변환한 뒤 대조한다(9.2절).

## 13.4 결과 보기

리포트를 읽는 방법과 커맨드라인에서 실행하는 방법을 설명한다.

### 13.4.1 리포트 읽기

```
검사 699건 · 오류 1건 · 경고 0건

[ERROR] column.length-satisfied — 1건
  MT_STD_DMN.PRM_VL_SCP_CN: VARCHAR(256) 컬럼인데 길이를 넘는 값이 1건 있습니다. 최대 300자: 가
격N9,2
```

같은 규칙에 위반된 항목을 모아 제시한다. 4천 행 규모의 사전에서는 위반이 수백 건 발생할 수 있는데, 규칙별로 묶으면 어느 부분을 수정해야 하는지를 먼저 파악할 수 있다.

```
report.ok          # ERROR가 하나도 없으면 참
report.errors      # ERROR만
report.warnings    # WARNING만
report.summary()   # {"error": 0, "warning": 0, "info": 0}
report.to_rows()   # 엑셀 기록용
```

### 13.4.2 커맨드라인

검사할 산출물을 위치 인수로 지정한다. 산출물의 종류는 시트 구성으로 판별하므로 옵션으로 구분하지 않는다.

```
# 배포 산출물 한 벌. 사전은 DB에서 읽는다
kdsm validate app/deliverable/DM-DST-00{1,2,3}_*.xlsx --name-prefix AP_

# 사전을 파일에서 읽는다
kdsm validate app/deliverable/DM-DST-00{1,2,3}_*.xlsx \
  --meta dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx --name-prefix AP_

# KDSM 자신을 구축할 때. 메타파일 자체를 검사한다
kdsm validate --meta dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx \
  --logical dbms/meta/meta_data/kdsm_logical_meta_20260828.xlsx \
  -P dbms/meta/meta_data/kdsm_physical_meta_20260828.xlsx \
  dbms/meta/meta_table/DM-LIB-00{4,5,6}_*.xlsx
```

논리 모델의 원천은 세 가지이다. 산출물(DM-DST-002)이 첫째이고, `--logical`로 지정한 메타파일이 둘째, `--logical-from-db`로 조회하는 메타테이블이 셋째이다. 산출물이 지정되어 있으면 그것을 우선한다. 물리 모델을 함께 수정하는 중이라면 작업 중인 산출물이 최신이기 때문이다.

물리 정의서를 여럿 주면 주제영역별로 나누어 적은 것으로 보고 한 벌로 모아 대조한다. 짝을 지정하지 않아도 된다. 이름이 문서를 통틀어 유일하므로 차례가 섞여도 짝을 잘못 연결하지 않는다.

표 13.8. kdsm validate 옵션

| 옵션 | 설명 |
|---|---|
| `<산출물>` | 검사할 DM-DST 산출물. 복수 지정할 수 있다. 종류는 시트로 판별한다 |
| `--meta` | 사전을 읽을 메타파일. 생략하면 DB의 `kdsm_meta`에서 읽는다 |
| `--no-dict` | 사전 없이 산출물 안에서의 무결성만 검사한다 |
| `--host` `--port` `--database` `--schema` `--user` | 사전을 읽을 접속 정보. 생략하면 환경변수 |
| `--env-prefix` | 접속 환경변수의 접두어 (기본 `KDSM_DB_`) |

| 옵션 | 설명 |
|---|---|
| `--logical` | 논리 모델 메타파일. 논리 산출물을 주지 않았을 때의 갈음이다 |
| `--logical-from-db` | 논리 모델을 DB의 `MT_ENTTY`·`MT_ATRB`에서 읽는다 |
| `-P`, `--physical` | 물리 모델 메타파일. 데이터베이스·테이블 규칙을 실행한다 |
| `--name-prefix` | 메타데이터 테이블의 물리명 접두어 (기본 `MT_`) |
| `--strict` | 경고도 실패로 처리한다 |
| `-q`, `--quiet` | 한 줄 요약만 출력한다 |

산출물을 하나도 주지 않고 `--meta`만 주면 그 메타파일 자체를 검사한다. 둘 다 지정하지 않으면 사전을 조회하기 전에 중단하고 필요한 인수를 안내한다. 인수 검사가 접속보다 먼저 수행되므로, 인수를 빠뜨렸을 때 드라이버 미설치와 같은 다른 원인을 안내받지 않는다.

ERROR가 있으면 종료 코드는 1이다. 지정하지 않은 원천에 해당하는 규칙은 실행하지 않는다. 검사하지 않은 항목을 통과한 것으로 보고하지 않기 위해서이다.

CHAPTER 14

# DDL 생성과 대조

DDL은 직접 작성하지 않고 물리 데이터 모델링 산출물에서 생성한다. 사용자는 물리 모델을 DM-DST-003 형식 산출물에 작성하고, 그 파일에서 DDL을 만든다(9.1절).

```
# 포워드 엔지니어링 — 산출물에서 DDL을 만든다
kdsm ddl 고객주문_V0.1.0_20260812.xlsx -o dbms/ddl/고객주문.sql --drop
kdsm ddl 고객_....xlsx 주문_....xlsx -o dbms/ddl        # 주제영역별로 나눈 경우

# 리버스 엔지니어링 — 운영 데이터베이스와 정의서를 대조한다
kdsm drift -s 고객주문_V0.1.0_20260812.xlsx
```

```
from kdsm.ddl import generate_ddl_from_spec

generate_ddl_from_spec(
    "고객주문_V0.1.0_20260812.xlsx",
    output="dbms/ddl/고객주문.sql",
    include_drop=True,
)
```

## 14.1 두 가지 방향

DDL을 다루는 일은 두 방향으로 나뉜다. 두 방향은 우열 관계가 아니라 사용하는 시점이 다르다.

표 14.1. DDL을 다루는 두 방향

| 방향 | 하는 일 | 명령 | 사용 시점 |
|---|---|---|---|
| 포워드 엔지니어링 | 산출물에서 DDL을 만든다 | `kdsm ddl` | 신규 테이블 생성 |
| 리버스 엔지니어링 | 운영 데이터베이스와 정의서를 대조한다 | `kdsm drift` | 데이터 모델 유지보수 |

적재 이전에는 데이터베이스가 없으므로 산출물이 유일한 원천이다. 운영이 시작되면 데이터베이스가 사실이 되고, 정의서는 그 사실을 따라가야 하는 문서가 된다. 이때 필요한 것이 대조다.

## 14.2 산출물 형식은 시트 구성으로 판별한다

`ddl`과 `drift`는 파일명으로 형식을 가리지 않는다. 사용자가 파일 이름을 자기 방식대로 짓기 때문이다. 파일명이 `고객주문_V0.1.0_20260812.xlsx`이더라도 `테이블 정의서`·`컬럼 정의서` 시트가 있으면 DM-DST-003 형식으로 읽는다. 파일명에 문서번호를 포함하도록 요구하지 않는다.

두 시트가 모두 있지 않으면 판별 기준과 실제 시트 구성을 함께 제시하고 중단한다.

```
오류: 물리 데이터 모델링 산출물이 아닙니다: 고객주문.xlsx
      DM-DST-003 형식은 '테이블 정의서'·'컬럼 정의서' 시트가 있어야 합니다.
```

## 14.3 파일 하나가 SQL 하나가 된다

산출물을 여러 개 지정하면 데이터 모델링을 주제영역별로 나눈 것으로 판단한다. 이때 파일 수만큼 SQL 파일을 만들고, 각 파일의 이름은 주제영역 이름을 따른다.

```
kdsm ddl 데이터표준_V0.1.0_20260812.xlsx \
         모델관리_V0.1.0_20260812.xlsx -o dbms/ddl
```

```
DDL 2벌을 만들었습니다.
  데이터표준 — 테이블 7개 · dbms/ddl/데이터표준.sql
  모델관리 — 테이블 5개 · dbms/ddl/모델관리.sql
```

주제영역의 이름은 파일명에서 버전과 날짜를 제외한 부분이다. `고객주문_V0.1.0_20260812.xlsx`는 `고객주문`이 된다. 명명 규칙과 일치하지 않는 파일명은 전체를 그대로 사용한다.

여러 벌을 하나의 SQL 로 합치지 않는다. 주제영역별로 나누어 작성한 의도가 배포 단위에도 유지되어야 하기 때문이다. 주제영역 하나만 다시 적용하는 상황에서 합쳐진 SQL 은 사용할 수 없다.

> **i** 주제영역 이름이 겹치면 중단한다
>
> 이름이 같으면 저장할 파일명도 같아져 뒤의 것이 앞의 것을 덮어쓴다. 안내 없이 덮어쓰면 무엇이 손실되었는지 확인할 수 없으므로 생성 이전에 안내한다.
>
> ```
> 오류: 주제영역 이름이 겹칩니다: 고객
>       저장할 파일명이 같아집니다. 파일명을 다르게 지으세요.
> ```

> **!** 주제영역 사이를 넘는 외래키
>
> 참조 대상이 그 파일에 없더라도 외래키 제약문은 그대로 생성한다. 참조 대상이 있는 파일을 먼저 적용하면 되므로, 생성 단계에서 제거하면 제약 자체가 사라진다.
>
> 적용 순서는 참조 관계가 정한다. `MT_TBL`이 `MT_ENTTY`를 참조하므로 논리 모델을 담은 파일이 물리 모델을 담은 파일보다 앞서야 한다.
>
> ```
> psql -h localhost -U kdsm -d kdsm \
>   -f dbms/ddl/데이터표준.sql \
>   -f dbms/ddl/모델관리.sql
> ```

## 14.4 출력 경로의 규칙

`-o`의 뜻은 산출물의 개수에 따라 달라진다.

표 14.2. 산출물 개수에 따른 출력 규칙

| 산출물 | `-o`를 준 경우 | `-o`를 생략한 경우 |
|---|---|---|
| 하나 | 그 경로에 SQL을 저장한다 | 표준 출력으로 내보낸다 |
| 여럿 | 그 디렉터리에 `<주제영역>.sql`로 저장한다 | `<저장소>/document/out/ddl`에 저장한다 |

산출물이 하나일 때만 표준 출력으로 내보내는 이유는, 여러 벌을 한 줄기로 이어 붙이면 어느 주제영역의 것인지 가릴 수 없기 때문이다.

```
kdsm ddl 고객주문_V0.1.0_20260812.xlsx ¦ psql -d kdsm
kdsm ddl 고객주문_V0.1.0_20260812.xlsx -t MT_DB --no-comments   # 한 테이블만 확인할 때
```

경로를 지정하면 요약만 표준 오류로 출력한다. 요약이 표준 출력에 섞이면 파이프로 전달되는 SQL이 손상되기 때문이다.

## 14.5 커맨드라인 옵션

표 14.3. kdsm ddl 옵션

| 옵션 | 설명 |
|---|---|
| `<산출물>` | 물리 데이터 모델링 산출물. 여럿이면 주제영역별로 나눈 것으로 본다 |
| `--seed` | 산출물 대신 패키지에 든 물리 정의서 셋을 읽는다(14.5.1절) |
| `-o`, `--output` | 산출물이 하나면 저장 경로, 여럿이면 저장 디렉터리(표 14.2) |
| `--drop` | `CREATE` 앞에 `DROP TABLE IF EXISTS`를 넣는다. PostgreSQL에는 `CASCADE`가 붙는다 |
| `--no-comments` | `COMMENT ON` 문을 만들지 않는다 |
| `--comment-style` | 주석에 넣을 것. `name` 또는 `full`(기본). 표 14.7 참고 |
| `--no-foreign-keys` | `FOREIGN KEY` 제약을 만들지 않는다 |
| `--no-schema` | `CREATE SCHEMA` 문을 만들지 않는다 |
| `--schema` | 스키마명을 덮어쓴다. 생략하면 정의서의 스키마를 쓴다 |
| `--dbms` | DDL을 만들 DBMS. `postgresql`(기본)·`mysql`·`databricks`. 14.7.3절 참고 |
| `--catalog` | `--dbms databricks`일 때 이름 앞에 붙일 카탈로그 |
| `--no-varchar` | `--dbms databricks`일 때 `VARCHAR(n)`도 `CHAR(n)`도 `STRING`으로 만든다 |
| `-t`, `--table` | 만들 테이블 물리명. 여러 번 지정할 수 있다. 생략하면 전부 만든다 |

### 14.5.1 패키지에 든 정의서로 만들기

`--seed`를 지정하면 산출물을 인수로 받지 않고 `kdsm` 패키지에 들어 있는 물리 정의서 셋(DM-LIB-004·005·006)을 읽는다. KDSM 메타테이블 12개의 DDL이 나온다. 저장소를 내려받지 않고 패키지만 설치한 사람이 자기 시스템을 세우는 자리이다(11.3.3절).

```
kdsm ddl --seed -o ddl/
#> DDL 3벌을 만들었습니다. 앞에서 뒤로 적용하십시오.
#>    ddl/kdsm_meta_std.sql — 데이터 표준 7테이블
#>    ddl/kdsm_meta_logical.sql — 논리 모델 2테이블
#>    ddl/kdsm_meta_physical.sql — 물리 모델 3테이블
```

파일 이름은 정의서 이름에서 뽑지 않고 정해 둔 것을 쓴다. 정의서 이름에는 판과 작성일이 붙어 있어 개정하면 SQL 이름까지 바뀌는데, 그러면 적용 스크립트를 매번 고쳐야 한다. 저장소의 `dbms/ddl`에 있는 이름과 같게 맞추었다.

적용 순서는 표준 사전 → 논리 → 물리이다. 뒤의 것이 앞의 것을 가리키므로 순서를 바꾸면 외래키에 걸린다.

> **i 산출물과 `--seed`는 함께 줄 수 없다**
>
> 어느 쪽을 읽을지 프로그램이 임의로 정하면 사용자가 준 것이 조용히 무시된다. 둘을 함께 주거나 둘 다 주지 않으면 무엇을 지정해야 하는지 알린다.
>
> ```
> 오류: 산출물과 --seed 는 함께 줄 수 없습니다. 어느 쪽을 읽을지 정하십시오.
> ```
>
> `--seed`를 받으면서 위치 인수가 필수에서 선택으로 바뀌었다. 따라서 아무것도 주지 않은 경우는 argparse의 사용법 오류(종료 코드 2)가 아니라 명령 자체의 오류(종료 코드 1)로 끝난다. `kdsm load`도 같다.
>
> ```
> 오류: 산출물을 지정하거나 --seed 를 주십시오.
>       kdsm ddl --help 로 사용법을 볼 수 있습니다.
> ```

> **만든 SQL을 적용하는 것은 이 명령의 몫이 아니다**
>
> `kdsm ddl`은 파일을 만들 뿐이며 데이터베이스에 실행하지 않는다. 만드는 일과 적용하는 일은 필요한 권한도 되돌리는 방법도 다르기 때문이다. 적용은 `psql`·`mysql`이나 `dbms/apply_sql.py`가 맡는다(15.6절).

> **⚠ `--drop`은 데이터를 삭제한다**
>
> `DROP TABLE IF EXISTS`가 앞에 붙는다. 운영 데이터베이스에 이미 데이터가 있다면 지정하지 않는다. 이 경우에는 필요한 `ALTER` 문을 별도로 작성한다.
>
> PostgreSQL에는 `CASCADE`가 함께 붙으므로 참조하는 객체까지 사라진다. MySQL과 Databricks에는 붙지 않는다. MySQL은 `CASCADE`를 문법으로 받되 아무 일도 하지 않고, Databricks의 `DROP TABLE`에는 그 절이 없다.
>
> 지우고 다시 만드는 것을 전제로 쓰는 자리는 개발 환경의 재구축뿐이다.

`-t`는 정의서 전체가 아니라 일부만 확인할 때 쓴다. 정의서에 없는 이름을 주면 만들기 전에 중단한다.

```
오류: 정의서에 없는 테이블입니다: MT_GHOST
```

## 14.6 라이브러리 인자

표 14.4. generate_ddl_from_spec 인자

| 인자 | 기본값 | 설명 |
|---|---|---|
| `output` | 없음 | 저장 경로. 상위 디렉터리는 자동 생성 |
| `include_drop` | `False` | `DROP TABLE IF EXISTS` 포함. PostgreSQL에는 `CASCADE`가 붙는다 |
| `include_comments` | `True` | `COMMENT ON` 포함 |
| `include_foreign_keys` | `True` | `FOREIGN KEY` 제약 포함 |
| `include_schema` | `True` | `CREATE SCHEMA IF NOT EXISTS` 포함 |
| `schema` | 정의서의 스키마 | 스키마명 덮어쓰기 |
| `tables` | 전체 | 생성할 테이블 물리명 목록 |
| `comment_style` | `full` | 주석에 무엇을 넣을지. 표 14.7 참고 |
| `dialect` | PostgreSQL | DBMS 방언 |
| `catalog` | 없음 | 이름 앞에 붙일 카탈로그. Databricks 방언에서만 쓴다 |

산출물 여러 벌을 한 번에 처리하려면 `generate_ddl_from_specs`를 사용한다. 위의 인자를 그대로 받고 `output_dir`만 추가된다.

```
from kdsm.ddl import generate_ddl_from_specs

made = generate_ddl_from_specs(
    ["데이터표준_V0.1.0_20260812.xlsx", "모델관리_V0.1.0_20260812.xlsx"],
    output_dir="dbms/ddl",
    include_drop=True,
)
```

`include_drop`은 `--drop`과, `include_comments`는 `--no-comments`의 반대와 같다. 커맨드라인이 부정형 이름을 쓰는 것은 기본값이 참이라 끄는 쪽만 지정하면 되기 때문이다.

## 14.7 생성기의 판단

### 14.7.1 식별자를 인용하지 않는다

> ! 중요
>
> 표준 물리명이 모두 대문자·숫자·밑줄로 이루어져 있으므로 PostgreSQL이 소문자로 변환하여도 충돌이 없다. 인용하면 조회할 때마다 따옴표를 붙여야 하므로 사용이 번거로워진다. 예약어이거나 허용 문자를 벗어나는 경우에만 인용한다.

```
CREATE TABLE kdsm_meta.MT_STD_WORD (...)   -- 카탈로그에는 mt_std_word로 저장된다
```

따라서 카탈로그에 저장된 이름은 소문자이다. 조회하거나 적재할 때 대문자로 인용하면 서로 다른 이름이 되므로 이 점에 유의한다(21.1.4.1절).

### 14.7.2 타입 매핑

`DATETIME`은 PostgreSQL 타입이 아니므로 `TIMESTAMP`로 변환하고 길이를 제거한다.

표 14.5. 논리 타입과 PostgreSQL 타입 매핑

| 논리 타입 | PostgreSQL |
|---|---|
| `DATETIME` | `TIMESTAMP` |
| `NUMBER` | `NUMERIC` |
| `STRING` | `VARCHAR` |

`TIMESTAMP`, `DATE`, `TIME`, `TEXT`, `BOOLEAN`, `SERIAL`, `BIGSERIAL`은 길이를 붙이지 않는다.

### 14.7.3 어느 DBMS의 DDL을 만드는가

`--dbms`로 고른다. 정의서는 그대로 두고 표기만 그 DBMS의 것으로 바꾼다. 하나의 물리 데이터 모델로 세 DBMS의 DDL을 만들 수 있다.

표 14.6. DBMS별 DDL 표기

| 항목 | PostgreSQL | MySQL | Databricks |
|---|---|---|---|
| 이름 | `스키마.테이블` | `스키마.테이블` | `카탈로그.스키마.테이블` |
| 인용 부호 | 큰따옴표 | backtick | backtick |
| 문자열 | `VARCHAR(40)` | `VARCHAR(40)`. 길이가 없으면 `TEXT` | `VARCHAR(40)`. 길이가 없으면 `STRING` |
| 고정 길이 문자열 | `CHAR(2)` | `CHAR(2)` | `CHAR(2)` |
| 수치 | `NUMERIC(9,2)` | `DECIMAL(9,2)` | `DECIMAL(9,2)` |
| 일시 | `TIMESTAMP` | `DATETIME` | `TIMESTAMP` |
| 주석 | `COMMENT ON` 문 | 정의 안에 기재 | `COMMENT ON` 문 |
| 삭제 | `DROP TABLE ... CASCADE` | `DROP TABLE ...` | `DROP TABLE ...` |

카탈로그는 `--catalog`로 지정한다. Databricks에서만 사용한다. 생략하면 두 단으로 적으며, 이때에는 접속 설정의 기본 카탈로그에 만들어진다. 실행하는 곳에 따라 대상이 달라지므로 배포용 SQL에는 지정하는 편이 낫다.

#### 14.7.3.1 MySQL은 주석을 정의 안에 기재한다

MySQL에는 `COMMENT ON` 문장이 없다. 컬럼은 정의 끝에, 테이블은 닫는 괄호 뒤에 기재한다.

```
CREATE TABLE sales.ORDR (
    ORDR_ID VARCHAR(20) NOT NULL COMMENT '주문아이디',
    CUST_ID VARCHAR(20) COMMENT '고객아이디',
    CONSTRAINT PK_ORDR PRIMARY KEY (ORDR_ID)
) COMMENT='주문 - 주문 내역';
```

`--no-comments`를 지정하면 이 부분도 함께 생성되지 않는다. 주석의 내용과 `--comment-style`의 동작은 DBMS와 무관하게 동일하다(14.7.6절).

> **i** MySQL에서 확인할 세 가지
>
> `VARCHAR`는 길이 없이 사용할 수 없다. 길이를 지정하지 않은 문자열 도메인은 `TEXT`로 생성한다. `TEXT`는 기본키와 인덱스에 그대로 사용할 수 없으므로, 키가 되는 컬럼의 도메인에는 길이를 지정해 둔다.
>
> `TIMESTAMP`는 `DATETIME`으로 변환한다. MySQL의 `TIMESTAMP`는 1970년부터 2038년까지만 저장하며 시간대에 따라 값이 변환된다. `DATETIME`은 기재한 값을 그대로 보관한다.
>
> `CREATE SCHEMA` 문에 `DEFAULT CHARACTER SET utf8mb4`를 함께 기재한다. MySQL의 기본 문자셋은 서버 설정을 따르므로, 지정하지 않으면 한글이 서버마다 다르게 저장될 수 있다. `utf8`은 3바이트까지만 저장하는 옛 이름이므로 사용하지 않는다.

> **!** `VARCHAR(n)`과 `CHAR(n)`의 길이는 강제된다
>
> Databricks의 문자열 타입은 셋이다. `STRING`은 길이 제한이 없고, `VARCHAR(n)`은 n 자를 넘는 값을 거부하며, `CHAR(n)`은 거기에 더해 남는 자리를 공백으로 채워 언제나 n 자로 저장한다.
>
> ```
> DELTA_EXCEED_CHAR_VARCHAR_LIMIT: Exceeds char/varchar type length limitation
> ```
>
> 그러므로 길이를 가진 표준도메인은 정의서에 기재한 타입을 그대로 지켜 `VARCHAR(n)` 또는 `CHAR(n)`으로 만든다. 표준도메인이 정한 길이를 데이터베이스가 지키게 하는 것이 표준화의 목적이기 때문이다. 길이가 없는 도메인만 `STRING`이 된다.
>
> `CHAR`를 `VARCHAR`로 대체하지 않는다. 고정 길이와 최대 길이는 다른 제약이다. 표준코드가 `CHAR(2)`이므로, `VARCHAR(2)`로 만들면 `1`과 `01`이 모두 저장되어 DBMS마다 다른 값이 남는다.
>
> `VARCHAR`와 `CHAR`는 테이블 스키마에서만 사용할 수 있다. `CAST`나 함수, 뷰의 표현식을 거치면 `STRING`이 된다. 컬럼의 타입으로 선언하는 이 용도에는 해당하지 않는다.
>
> 워크스페이스에 `spark.sql.legacy.charVarcharAsString`이 `true`로 설정되어 있으면 길이 검사와 공백 채움이 모두 동작하지 않는다. 이 설정을 사용하는 환경에서는 `--no-varchar`로 `STRING`을 생성하는 편이 낫다. 지켜지지 않는 제한을 정의서에만 남겨 두면 지켜지는 것으로 오해하게 된다.
>
> `--no-varchar`는 `VARCHAR`와 `CHAR`를 함께 `STRING`으로 만든다. 옵션 이름에 `varchar`만 들어 있으나 `CHAR`도 대상이다. 길이 검사를 끈 환경에서는 `CHAR`의 공백 채움도 동작하지 않으므로 둘을 갈라 둘 이유가 없다.

> **i 기본키·유일키·외래키는 선언될 뿐 강제되지 않는다**
>
> Databricks의 기본키·유일키·외래키 제약은 선언적 정보이며 데이터베이스가 위반을 막지 않는다. 생성문에 포함하는 것은 모델을 문서와 데이터베이스 양쪽에 남기기 위해서이다(18.5.4.6절).
>
> 기본키와 외래키는 Databricks Runtime 13.3 LTS 이상, 유일키(UNIQUE)는 18.2 이상에서 선언할 수 있다. 그 이전 판은 문장 자체를 거부한다. 외래키는 `--no-foreign-keys`로 제외할 수 있다.
>
> 기본키를 구성하는 컬럼은 `NOT NULL`이어야 한다. 정의서의 `Null 허용`이 `Y`인 컬럼에 기본키 순번을 부여하면 생성 시점에 거부되므로, 생성 전에 검증한다(13장).

### 14.7.4 복합 외래키를 묶는다

같은 테이블을 가리키는 컬럼이 여럿이면 하나로 묶는다.

```
CONSTRAINT FK_MT_TBL_1 FOREIGN KEY (DB_NM, SCHM_NM) REFERENCES kdsm_meta.MT_DB (DB_NM, SCHM_NM)
```

컬럼 순서는 컬럼 순번을 따른다. 검증 규칙 `column.foreign-key-target`도 같은 방식으로 묶으므로, 검사를 통과한 정의가 생성 단계에서 문제를 일으키지 않는다.

### 14.7.5 유일 제약은 표기의 번호로 묶는다

컬럼 정의서의 「소속 유일키」 표기(`UK1-1`)를 읽어 `UNIQUE` 제약을 만든다. 번호가 같은 컬럼이 한 제약으로 묶이고 순번이 컬럼 순서를 정한다(9.3.4절). 배치는 기본키 뒤, 외래키 앞이다.

```
CONSTRAINT PK_AP_USER PRIMARY KEY (OGNZ_ID, USER_ID),
CONSTRAINT UX_AP_USER_1 UNIQUE (EML_ADDR),
CONSTRAINT FK_AP_USER_1 FOREIGN KEY (OGNZ_ID) REFERENCES kdsm_app.AP_OGNZ (OGNZ_ID)
```

제약 이름은 `UX_<테이블 물리명>_<번호>`이다. 표기의 번호가 이름에 그대로 남아 산출물과 DDL을 나란히 볼 수 있다. 표기를 읽지 못하면 어느 컬럼이 왜 문제인지 밝히며 생성을 중단한다. `UNIQUE` 문법은 세 DBMS가 같다.

### 14.7.6 주석에 무엇을 넣을지 고른다

주석의 재료는 정의서의 한글명과 설명 둘이다. 둘을 모두 넣을지 한글명만 넣을지는 `comment_style`이 정한다. 테이블 주석과 컬럼 주석에 함께 적용되며, 한 스키마 안에서 주석 표기가 갈리지 않도록 둘을 따로 지정하지 않는다.

표 14.7. 주석 방식

| 값 | 넣는 것 | 결과 |
|---|---|---|
| `full` (기본) | 한글명 + `-` + 설명 | `데이터베이스이름 - 정보 시스템에서 DB를 식별하기 위하여 사용하는 물리 정보명(영문명)` |
| `name` | 한글명 | `데이터베이스이름` |

```
kdsm ddl 고객주문_V0.1.0_20260812.xlsx --comment-style name
```

기본값은 `full` 이다. 주석은 카탈로그만 보고 작업하는 사람이 의지할 유일한 설명이므로, 지정하지 않았다면 더 많이 제공하는 쪽을 택한다. 설명이 길어 카탈로그 조회 결과가 한 줄에 담기지 않는 것이 부담이라면 `name` 을 지정한다.

```
-- full
COMMENT ON COLUMN kdsm_meta.MT_DB.DB_NM IS '데이터베이스이름 - 정보 시스템에서 DB를 식별하기 위하여 사용하는 물리 정보명(영문명)';

-- name
COMMENT ON COLUMN kdsm_meta.MT_DB.DB_NM IS '데이터베이스이름';
```

`full`에서 한글명과 설명이 같으면 한 번만 기록한다. `name`을 선택하여도 한글명이 비어 있으면 설명을 기록한다. 주석을 짧게 유지하려는 것이지 비워 두려는 것이 아니기 때문이다.

> **i 주석 방식을 변경하면 대조 결과가 달라지지 않는다**
>
> `kdsm drift`는 주석을 대조 대상으로 삼지 않는다(14.9.1절). 따라서 이미 생성한 테이블과 다른 방식을 선택하여도 차이로 보고되지 않는다. 기존 테이블의 주석을 변경하려면 `COMMENT ON` 문만 다시 실행한다.

## 14.8 생성하기 전에 검증한다

> **힌트**
>
> 생성기는 정의서의 내용을 그대로 옮길 뿐이다. 잘못된 정의는 잘못된 DDL로 이어진다. 특히 `column.foreign-key-target`은 DDL이 생성되더라도 실행 단계에서 거부되는 경우를 검출한다.

```
kdsm validate 고객주문_V0.1.0_20260812.xlsx --meta 메타파일.xlsx
kdsm ddl 고객주문_V0.1.0_20260812.xlsx -o dbms/ddl/고객주문.sql --drop
```

## 14.9 정의서와 데이터베이스 대조

DDL을 만들어 적용한 순간에는 정의서와 데이터베이스가 일치한다. 그러나 운영이 시작되면 급한 수정이 데이터베이스에만 반영되고 정의서는 그대로 남는 일이 생긴다. 시간이 지날수록 둘의 거리는 벌어지는데, 벌어졌다는 사실 자체를 아무도 인지하지 못하는 것이 문제다. `kdsm drift`는 그 거리를 측정하여 제시한다.

```
kdsm drift -s 고객주문_V0.1.0_20260812.xlsx
kdsm drift -s 데이터표준_....xlsx -s 모델관리_....xlsx --schema sales
```

### 14.9.1 네 가지 대조 대상

표 14.8. 네 가지 대조 대상

| 대조 대상 | 대조 항목 |
|---|---|
| 테이블 | 한쪽에만 있는 테이블 |
| 컬럼 | 양쪽에 다 있는 테이블에서 한쪽에만 있는 컬럼 |

| 대조 대상 | 대조 항목 |
|---|---|
| 타입 | 같은 이름의 컬럼이 가진 데이터 타입과 길이 |
| 제약 | NOT NULL·기본키·외래키 |

대조 대상으로 먼저 묶고 그 안에서 구분별로 묶어 출력한다. 동일한 구분의 차이는 대개 동일한 원인에서 발생하므로, 한자리에 모으면 원인을 한 번에 파악할 수 있다.

```
정의서와 데이터베이스 대조
==========================================

대조 대상  테이블 11개 · 컬럼 143개
찾은 차이  4건

[테이블] 1건
  정의서에만 있음 1건
    kdsm_meta.MT_STDCD_DTL — 정의서에 있으나 데이터베이스에 없다.
      정의서: 표준코드상세
      데이터베이스: (없음)

[타입] 2건
  서로 다름 2건
    kdsm_meta.MT_STD_TRM.STD_TRM_NM — 데이터 타입이 다르다.
      정의서: VARCHAR(100)
      데이터베이스: INTEGER
    kdsm_meta.MT_STD_WORD.STD_WORD_NM — 길이가 다르다.
      정의서: VARCHAR(40)
      데이터베이스: VARCHAR(50)

[제약] 1건
  정의서에만 있음 1건
    kdsm_meta.MT_COL — 기본키 구성이 다르다.
      정의서: PRIMARY KEY (TBL_PHYS_NM, COL_PHYS_NM)
      데이터베이스: (없음)
```

### 14.9.2 어느 쪽이 옳은지는 판정하지 않는다

차이를 발견하면 구분만 제시한다. 정의서에만 있음, 데이터베이스에만 있음, 서로 다름 셋이다. 어느 쪽을 기준으로 삼을지는 시점이 정한다. 적재 이전이면 정의서가 계획이므로 정의서가 기준이고, 운영 중이면 데이터베이스가 사실이므로 데이터베이스가 기준이다. 프로그램은 그 시점을 알 수 없으므로 판정하지 않는다.

같은 이유로 차이가 있어도 정상 종료한다. 차이가 있다는 사실 자체는 잘못이 아니기 때문이다. 어긋남을 막으려면 `--strict`를 지정한다. 이때는 차이가 하나라도 있으면 종료 코드 1로 끝나므로 CI에 연결할 수 있다.

### 14.9.3 제약은 이름이 아니라 구성 컬럼으로 대조한다

정의서에는 제약 이름을 기재하는 항목이 없고, DDL을 생성할 때 `PK_<테이블>` 형식으로 지어 붙인다. 따라서 이름을 대조하면 사람이 직접 생성한 제약이 모두 다르다고 판정된다. 무엇을 키로 삼았는가는 이름이 아니라 구성 컬럼이 정한다.

기본키는 컬럼의 순서까지 본다. 복합 기본키의 컬럼 순서가 인덱스의 쓰임을 바꾸기 때문이다. 같은 세 컬럼이라도 순서가 다르면 다른 인덱스다.

> **i 대소문자는 접어서 대조한다**
>
> DBMS는 인용하지 않은 식별자를 접어서 저장한다(14.7.1절). 정의서에 `MT_TBL`이라 기재하고 인용 없이 생성하면 PostgreSQL에는 `mt_tbl`로 남는다. 접지 않고 대조하면 같은 테이블이 서로 다른 것으로 판정된다.
>
> 접는 것은 대조를 위한 처리이므로 화면에 출력할 때는 원래 기재된 표기로 되돌린다.

### 14.9.4 대조하지 않는 것

세 가지는 대조 대상에서 제외한다. 확인할 수 없는 것을 확인한 것처럼 보고하지 않기 위해서다.

**소수점 이하 자릿수**
: 컬럼 정의서에 해당 항목이 없어 정의서 쪽 값을 알 수 없다. 없는 값을 0으로 놓고 대조하면 실제로는 일치하는 것이 어긋난 것으로 판정된다.

**고정 폭 수치 타입의 길이**
: `information_schema`는 `INTEGER`의 `numeric_precision`을 32로 제시한다. 이것은 사람이 정한 자릿수가 아니라 저장 폭이 32비트라는 뜻이다. `SMALLINT`·`INTEGER`·`BIGINT`·`SERIAL`·`BIGSERIAL`·`REAL`·`DOUBLE`·`FLOAT`는 이름만 대조한다.

**정의서가 언급하지 않은 스키마**
: 정의서가 다루는 스키마 안의 테이블만 본다. 그 밖의 스키마까지 훑으면 대조가 아니라 진단이 되고, 그 일은 `kdsm diagnose`가 담당한다(18장).

길이를 측정할 수 없는 타입은 따로 모아 밝힌다. Databricks의 `STRING`처럼 카탈로그가 길이를 제공하지 않는 경우다. 측정하지 못한 것을 일치한다고 보고하면 확인하지 않은 것을 확인한 것처럼 제시하는 셈이다. Databricks에서는 정의서의 `VARCHAR`·`CHAR`와 데이터베이스의 `STRING`을 같은 타입으로 보고 길이를 측정할 수 없는 항목으로 분류한다. `--no-varchar`로 생성한 테이블이 이에 해당한다.

### 14.9.5 커맨드라인 옵션

**표 14.9.** kdsm drift 옵션

| 옵션 | 설명 |
|---|---|
| `-s`, `--spec` | 물리 데이터 모델링 산출물. 여러 번 지정할 수 있다. 유일한 필수 인자이다 |
| `--schema` | 정의서의 스키마를 이 이름으로 바꿔 대조하고, 카탈로그도 이 스키마를 읽는다 |
| `--dbms` | 대조할 DBMS. `postgresql`(기본)·`mysql`·`databricks` |
| `--catalog` | `--dbms databricks`일 때 기본 카탈로그 |
| `--env-prefix` | 대조할 데이터베이스의 접속 정보 환경변수 접두어 (기본 `KDSM_DB_`) |
| `--host` `--port` `--database` `--user` | 환경변수의 접속 정보를 덮어쓴다 |
| `--strict` | 차이가 하나라도 있으면 종료 코드 1로 끝낸다 |
| `-q`, `--quiet` | 한 줄 요약만 출력한다 |

접속 정보는 `KDSM_DB_HOST`부터 `KDSM_DB_PASSWORD`까지 여섯 환경변수에서 읽는다(20.9절). `--host`·`--port`·`--database`·`--user`로 덮어쓸 수 있으나 비밀번호는 인자가 없으므로 `KDSM_DB_PASSWORD`를 쓴다(20.10절). `kdsm ddl`은 파일만 읽으므로 환경변수가 필요하지 않다.

`--dbms`를 지정하면 그 DBMS의 접두어를 읽는다. MySQL은 `MYSQL_*`, Databricks는 `DATABRICKS_*`이다. 진단과 같은 카탈로그 리더를 사용하므로 접속 방법과 대상 표기가 같다(18.5.4.3절).

```
kdsm drift -s 고객주문_*.xlsx --dbms databricks --catalog dev_landing
```

타입 이름은 방언이 표준 표기로 맞춘 뒤에 대조한다. 같은 `VARCHAR(40)`을 Databricks는 `STRING`이라고 부르므로, 이름을 그대로 대조하면 모든 컬럼이 다르다고 나온다. 다만 Databricks는 문자열의 길이를 알려 주지 않으므로 길이는 대조하지 못한다(18.5.5절).

`-s`를 여러 번 지정하면 주제영역별로 나눈 산출물을 한데 모아 대조한다. DDL과 달리 결과를 파일별로 나누지 않는다. 데이터베이스는 하나이고 대조는 그 하나를 상대로 하는 일이기 때문이다.

`--schema`를 생략하면 정의서의 `스키마` 항목에 기재된 이름을 그대로 읽는다. 그 항목이 비어 있으면 어디를 읽을지 알 수 없으므로 중단한다.

```
오류: 대조할 스키마를 알 수 없습니다. 정의서의 `스키마` 칸이 비어 있습니다. --schema 로 알려 주세요.
```

비밀번호는 인자로 받지 않는다. 명령줄에 기재한 값은 `ps` 출력과 셸 기록에 남기 때문이다. `<접두어>PASSWORD` 환경변수를 사용한다.

### 14.9.6 라이브러리로 사용하기

```
from kdsm.db import Connection
from kdsm.ddl import read_physical_spec
from kdsm.diagnose import read_catalog, read_constraints
from kdsm.drift import compare, to_text

tables, columns = read_physical_spec("고객주문_V0.1.0_20260812.xlsx")
with Connection.from_env() as connection:
    report = compare(
        tables,
        columns,
        read_catalog(connection, ["kdsm_meta"]),
        read_constraints(connection, ["kdsm_meta"]),
    )
print(to_text(report))
```

`DriftReport.ok`는 차이가 하나도 없을 때 참이다. `of(Axis.TYPE)`으로 축별 차이만 추릴 수 있고, `summary()`는 축별 건수를 제공한다.

> 검증과 대조는 보는 곳이 다르다
>
> `kdsm validate`는 문서와 문서를 대조한다. 논리 정의서와 물리 정의서가 어긋났는지, 물리명이 표준용어에 등재되어 있는지를 본다(13장).
>
> `kdsm drift`는 문서와 데이터베이스를 대조한다. 검증을 모두 통과한 정의서라도 데이터베이스가 그것을 따르고 있는지는 별개의 문제다. 두 명령은 대체 관계가 아니라 보완 관계다.

CHAPTER 15

# 메타데이터 적재

메타 테이블에 데이터를 적재하는 목적은 두 가지이다. 같은 테이블을 대상으로 하지만 시점과 방법이 다르기 때문에 명령어로 구분한다.

**표 15.1.** 적재의 두 가지 목적

| 목적 | 명령어 | 적재 대상 | 수행 시점 |
|---|---|---|---|
| 메타데이터 파일의 적재 | `kdsm load` | `dbms/meta`의 메타파일 3종 | KDSM 구축 초기 단계 |
| 산출물 파일의 적재 | `kdsm apply` | 프로젝트가 작성한 DM-DST 산출물 3종 | 해당 산출물의 검증 완료 후 |

전자는 사전 자체를 구축하는 작업이고, 후자는 구축된 사전에 프로젝트의 표준과 모델을 추가하는 작업이다. 후자는 기존 등재분과의 충돌 여부를 사전에 확인해야 한다는 점도 다르다(15.5절).

`dbms/rebuild.sh`는 전자만 수행한다. 산출물 적재는 이 스크립트의 대상이 아니다(15.6절).

## 15.1 메타데이터 파일의 적재

메타파일의 시트를 `kdsm_meta` 스키마의 동일한 이름을 가진 테이블에 적재한다.

메타파일은 세 개이다. 표준 사전 7시트, 논리 모델 2시트, 물리 모델 3시트가 각각 다른 파일에 있다. `load_metafile()`은 파일에 실제로 존재하는 시트만 선별하여 적재하므로, 세 파일을 차례로 전달하면 된다.

```
kdsm load 사전메타파일.xlsx  -s DM-DST-003_물리데이터모델_....xlsx
kdsm load 논리메타파일.xlsx  -s DM-DST-003_물리데이터모델_....xlsx
kdsm load 물리메타파일.xlsx  -s DM-DST-003_물리데이터모델_....xlsx
```

적재 대상은 `--dbms`로 고른다. PostgreSQL이면 `--host`·`--port`·`--database`·`--user`로 접속 정보를 지정할 수 있고, 생략하면 `KDSM_DB_*` 환경변수를 참조한다. MySQL과 Databricks는 이 네 인수를 사용하지 않고 `MYSQL_*`·`DATABRICKS_*` 환경변수만 읽는다. `--schema`와 `--catalog`는 세 DBMS 모두에 적용된다. 비밀번호와 토큰은 인수로 받지 않는다(20.10절).

> ! 파일이 세 개이면 트랜잭션도 세 개이다
>
> `load_metafile()` 한 번의 호출이 하나의 트랜잭션이다. 세 파일을 차례로 전달하면 트랜잭션도 세 개로 나뉜다. 두 번째에서 실패하면 첫 번째는 이미 반영된 상태이다. Databricks는 트랜잭션 자체가 없으므로 한 파일 안에서도 앞의 것이 남는다(15.3.1절).
>
> 세 파일을 한꺼번에 되돌려야 한다면 `dbms/rebuild.sh`와 같이 검증을 먼저 일괄 수행하고, 세 파일이 모두 통과한 뒤에 적재를 시작해야 한다.

## 15.2 사용 방법

접속 정보를 준비하고 적재를 실행하는 절차를 설명한다.

### 15.2.1 준비

```
uv sync --extra db      # psycopg 설치. LLM 기능도 쓴다면 --extra llm 을 함께 지정한다

export KDSM_DB_HOST=localhost
export KDSM_DB_PORT=5432
export KDSM_DB_NAME=kdsm          # 데이터베이스 이름. NAME 이지 DATABASE 가 아니다
export KDSM_DB_SCHEMA=kdsm_meta
export KDSM_DB_USER=kdsm
read -rs -p "비밀번호: " KDSM_DB_PASSWORD && export KDSM_DB_PASSWORD
```

여섯 변수를 모두 지정한다. 지정하지 않은 항목은 기본값을 쓰는데, 그 기본값은 개발용 로컬 서버를 가리키므로 운영 환경에서는 대개 맞지 않는다. `KDSM_DB_` 접두어는 `--env-prefix`로 바꿀 수 있다(20.9.4절).

> i 비밀번호는 환경변수로만 받는다
>
> `--password` 같은 인자는 두지 않았다. 명령줄에 적은 값은 `ps` 출력과 셸 기록에 남기 때문이다. 위에서 `export` 대신 `read -rs`를 쓴 것도 같은 이유이다. 배경과 다른 방법은 20.10절 에 있다.
>
> `ConnectionConfig.dsn()`은 비밀번호를 의도적으로 제외한다. 이 값은 로그나 예외 메시지에 기록되기 쉬운데, 그 안에 비밀번호가 포함되면 되돌릴 수 없다. `repr()`도 비밀번호를 마스킹한다.

### 15.2.2 적재

```
from kdsm.db import load_metafile

report = load_metafile(
    "dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx",
    actor="choonghyunryu",
    spec="DM-DST-003_물리데이터모델_....xlsx",
)
print(report.to_text())
```

```
검사 4448건 · 오류 0건 · 경고 0건
위반 없음

적재 완료 — 7개 시트 · 4527행
  MT_STD_DMN_GROUP: 8행 읽어 8행 반영
  MT_STDCD_MSTR: 6행 읽어 6행 반영
  MT_STD_DMN_CLSF: 53행 읽어 53행 반영
  MT_STDCD_DTL: 26행 읽어 26행 반영
  MT_STD_WORD: 4193행 읽어 4193행 반영
  MT_STD_DMN: 139행 읽어 139행 반영
  MT_STD_TRM: 102행 읽어 102행 반영
```

### 15.2.3 인자

표 15.2. load_metafile 인자

| 인자 | 기본값 | 설명 |
|---|---|---|
| `actor` | 접속 사용자명 | `CREATR`·`MDFR`에 기록할 작업자명 |
| `sheets` | 파일에 있는 것 전부 | 적재할 시트명 목록 |
| `validate` | `True` | 적재 전 검증. 오류가 있으면 적재하지 않음 |
| `dry_run` | `False` | 읽기·검증·변환까지만 하고 쓰지 않음 |
| `schema` | `kdsm_meta` | 대상 스키마명 |
| `spec` | 없음 | 산출물 경로. NOT NULL·길이 제약까지 확인 |
| `dbms` | `postgresql` | 넣을 DBMS. `mysql`·`databricks`도 받는다 |
| `catalog` | 없음 | 대상 카탈로그. Databricks에서만 쓴다 |

처음에는 `dry_run=True`로

```
report = load_metafile(path, dry_run=True, spec=spec)
if not report.ok:
    print(report.validation.to_text())
```

## 15.3 내부 동작

행을 하나씩 적재하지 않는다. 어떤 순서로 무엇을 수행하는지 살펴본다.

### 15.3.1 적재 방식

DBMS마다 문장이 다르다. 결과는 같다. 기본키가 이미 있으면 갱신하고 없으면 넣는다.

표 15.3. DBMS별 적재 문장

| DBMS | 문장 |
|---|---|
| PostgreSQL | 임시 테이블에 `COPY` 후 `INSERT ... ON CONFLICT DO UPDATE` |

| DBMS | 문장 |
|---|---|
| MySQL | `INSERT ... ON DUPLICATE KEY UPDATE` |
| Databricks | `MERGE INTO ... USING (VALUES ...)` |

PostgreSQL은 행을 하나씩 INSERT하지 않는다. 임시 테이블을 생성하여 COPY로 일괄 적재한 뒤, 한 번의 `INSERT ... SELECT ... ON CONFLICT DO UPDATE`로 본 테이블에 반영한다.

```
CREATE TEMP TABLE stg_mt_std_word (LIKE kdsm_meta.mt_std_word INCLUDING DEFAULTS) ON COMMIT DROP;
COPY stg_mt_std_word (...) FROM STDIN;
INSERT INTO kdsm_meta.mt_std_word (...) SELECT ... FROM stg_mt_std_word
ON CONFLICT (std_word_nm) DO UPDATE SET ..., mdfcn_dt = ..., mdfr = 'hryu';
```

> 일괄 요청으로 I/O 비용을 낮춘다
>
> 데이터베이스와의 네트워크 라운드 트립(Round-trip) 횟수를 행 수가 아닌 처리 테이블 수 수준으로 고정한다. 이에 따라 4,193행 규모의 마스터 데이터도 단 3번의 일괄 요청(Upsert)만으로 적재를 완료하여 I/O 성능을 극대화한다.

PostgreSQL에서는 전체가 한 트랜잭션이다. 중간에 실패하면 아무것도 반영되지 않는다.

MySQL도 동일하다. 임시 테이블을 생성하지 않으므로 스키마에 `CREATE` 권한이 필요하지 않다. 갱신 대상을 `new.컬럼`으로 지정하며, 이 표기는 MySQL 8.0.19 이상에서 사용할 수 있다.

Databricks는 값을 임시 테이블이 아니라 `VALUES` 절에 담아 병합한다. 문장이 길어지지 않도록 500행마다 나누어 전송한다. Databricks에는 트랜잭션이 없으므로 앞 문단의 「전체가 한 트랜잭션」은 이 DBMS에 해당하지 않는다. 그 결과는 아래에서 다룬다.

> ! 적재에는 읽기 권한도 필요하다
>
> 기본키가 이미 있는지 확인해야 넣을 것인지 갱신할 것인지 결정할 수 있다. 세 DBMS가 모두 그러하며, 쓰기 권한만 부여하면 적재가 중단된다.
>
> ```
> 오류: 'MT_STD_DMN_GROUP' 적재에 실패했습니다: [INSUFFICIENT_PERMISSIONS] ...
>       적재에는 읽기 권한도 필요합니다.
> ```
>
> Databricks에서는 스키마 단위로 부여한다. 테이블이 추가될 때마다 다시 부여하지 않아도 된다.
>
> ```
> GRANT USE CATALOG ON CATALOG dev_landing TO `<주체>`;
> GRANT USE SCHEMA ON SCHEMA dev_landing.kdsm_meta TO `<주체>`;
> GRANT SELECT, MODIFY ON SCHEMA dev_landing.kdsm_meta TO `<주체>`;
> ```
>
> 테이블을 생성한 계정과 적재하는 계정이 다르면 이 상황이 발생한다. Unity Catalog에서는 생성한 주체가 소유자가 되며, 다른 주체에게는 권한이 자동으로 부여되지 않는다. SQL 편집기에서 사람이 테이블을 만들고 서비스 프린시펄로 적재하는 구성이 이에 해당한다.

> ! Databricks는 중간에 실패하면 앞의 것이 남는다
>
> Databricks에는 여러 문장을 하나로 묶는 트랜잭션이 없다. 문장 하나하나가 그 자체로 완결되므로 되돌릴 방법이 없다. 시트가 일곱이고 행이 많으면 문장도 여럿이 되며, 그 가운데 하나가 실패하면 앞에서 반영된 것은 그대로 남는다.
>
> 같은 파일을 다시 적재하면 기본키가 같은 행은 갱신되므로 결과는 같아진다. 그러므로 실패한 지점을 수정한 뒤 처음부터 다시 적재한다. 반쯤 반영된 상태를 사람이 손으로 정리하지 않는다.
>
> 적재 전에 `--dry-run`으로 검증을 통과시켜 두면 이 상황 자체가 드물어진다. 검증에서 걸러지는 오류는 데이터베이스에 닿기 전에 멈춘다.

### 15.3.2 적재 순서

외래키가 설정되어 있으므로 참조 대상 테이블을 먼저 적재한다. 열두 테이블이 하나의 순서를 이룬다.

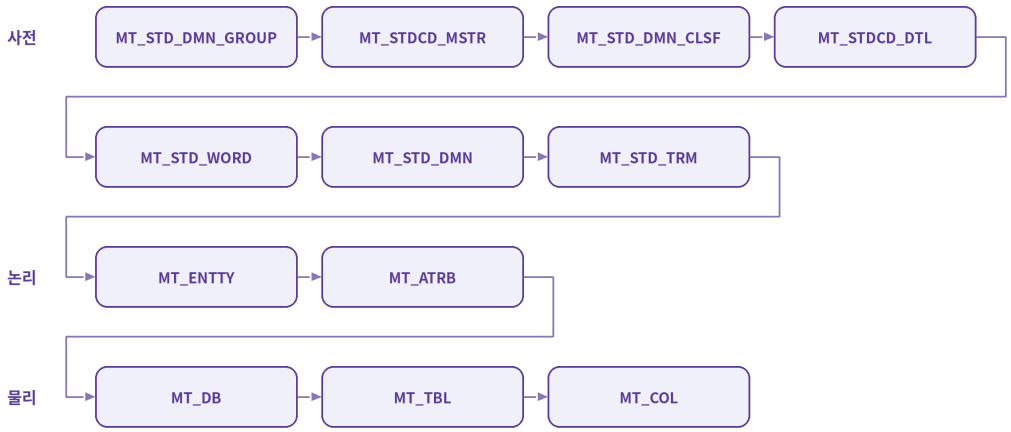


그림 15.1. 외래키를 위반하지 않는 적재 순서

영역 사이에도 순서가 있다. `MT_ATRB.DMN_NM`이 `MT_STD_DMN`을, `MT_TBL.REL_ENTTY_NM`이 `MT_ENTTY`를 가리키기 때문이다. 파일이 세 개로 분리되어 있어도 순서는 하나이다.

`sheets=`로 일부만 선택하여도 이 순서는 유지한다. 지정한 순서를 그대로 따르면 외래키 제약을 위반하게 된다.

> i 시트를 지정하지 않으면 파일에 있는 것만 선택한다
>
> `sheets=`를 생략하면 열두 시트를 모두 요구하는 것이 아니라, 그 파일에 실제로 존재하는 시트만 선택한다. 메타파일이 세 개로 분리되어 있으므로 전부를 요구하면 어느 파일도 온전히 적재할 수 없기 때문이다.
>
> 적재할 수 있는 시트가 하나도 없으면 파일에 어떤 시트가 있었는지 함께 안내하고 중단한다.
>
> ```
> 오류: 적재할 수 있는 시트가 없습니다. 파일의 시트: 표지, 메모.
>       가능한 시트: MT_STD_DMN_GROUP, MT_STDCD_MSTR, ...
> ```

### 15.3.3 감사 컬럼

시트에는 없고 테이블에만 있는 네 컬럼은 적재기가 표 15.4 과 같이 채운다.

표 15.4. 감사 컬럼의 신규·갱신 동작

| 컬럼 | 신규 | 갱신 |
|---|---|---|
| `CRT_DT` 생성일시 | 적재 시각 | 변경하지 않음 |
| `CREATR` 생성자 | 작업자명 | 변경하지 않음 |
| `MDFCN_DT` 수정일시 | `NULL` | 적재 시각 |
| `MDFR` 수정자 | `NULL` | 작업자명 |

> ! 생성 정보를 덮어쓰지 않는다
>
> 갱신할 때 `CRT_DT`·`CREATR`를 그대로 두는 것이 핵심이다. 누가 언제 최초로 적재하였는지가 남아 있어야 이력이 의미를 갖는다.

### 15.3.4 UPSERT의 한계

기본키가 이미 있으면 갱신하므로 같은 파일을 여러 번 적재해도 안전하다.

> ⚠ 시트에서 삭제한 행은 DB에 남는다
>
> UPSERT는 삽입과 갱신만 수행하며 삭제하지 않는다. 시트와 DB를 정확히 일치시키려면 테이블을 비우고 다시 적재해야 한다. 다만 외래키 제약이 있으므로 자식 테이블부터 삭제하고 부모 테이블부터 적재하는 순서를 지켜야 한다.

## 15.4 그 밖에

다른 작업과 트랜잭션을 함께 묶는 방법과 커맨드라인 사용법을 설명한다.

### 15.4.1 연결을 재사용할 때

```
from kdsm.db import Connection, ConnectionConfig

config = ConnectionConfig(host="db.internal", database="kdsm", user="hryu", password=...)
load_metafile(path, config=config)

with Connection(config) as conn:          # 다른 작업과 한 트랜잭션으로 묶을 때
    load_metafile(path, conn)
```

`with` 블록을 정상적으로 빠져나가면 커밋하고, 예외가 발생하면 롤백한다.

### 15.4.2 커맨드라인

```
kdsm load dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx \
  --actor choonghyunryu -s DM-DST-003_물리데이터모델_....xlsx
```

```
kdsm load dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx \
  --dbms databricks --catalog dev_landing
```

접속 정보는 대상 DBMS의 접두어에서 읽는다. PostgreSQL은 `KDSM_DB_*`, MySQL은 `MYSQL_*`, Databricks는 `DATABRICKS_*`이다(20.9절).

**표 15.5.** kdsm load 옵션

| 옵션 | 설명 |
|---|---|
| `<메타파일>` | 적재할 메타파일 경로 |
| `--seed` | 메타파일 대신 패키지에 든 초기 데이터 세 벌을 적재 순서대로 넣는다 (15.4.3절) |
| `--actor` | 작업자명 |
| `--sheet` | 적재할 시트명. 여러 번 지정할 수 있다 |
| `-s`, `--spec` | 산출물 경로 |
| `--dbms` | 넣을 DBMS. `postgresql`(기본)·`mysql`·`databricks` |
| `--catalog` | `--dbms databricks`일 때 대상 카탈로그 |
| `--schema` | 대상 스키마명. 세 DBMS 모두에 적용된다 |
| `--host` `--port` `--database` `--user` | PostgreSQL일 때의 접속 정보. 생략하면 환경변수 |
| `--env-prefix` | 접속 환경변수의 접두어 (기본 `KDSM_DB_`) |
| `--no-validate` | 적재 전 검증을 건너뛴다 |
| `-n`, `--dry-run` | 읽기·검증까지만 |

검증에 실패하여 적재하지 못한 경우에도 종료 코드는 1이다. 0은 데이터가 반영되었다는 의미여야 하기 때문이다.

### 15.4.3 패키지에 든 초기 데이터 적재하기

`--seed`를 지정하면 메타파일을 인수로 받지 않고 `kdsm` 패키지에 들어 있는 세 벌을 적재 순서대로 넣는다(11.3.3절). 정의서도 영역마다 짝지어 함께 읽으므로 `-s`를 주지 않는다.

```
kdsm load --seed --actor choonghyunryu
#> == standard — kdsm_modeling_meta_20260828.xlsx
#> == logical  — kdsm_logical_meta_20260828.xlsx
#> == physical — kdsm_physical_meta_20260828.xlsx
```

순서는 표준 사전 → 논리 → 물리이다. 뒤의 것이 앞의 것을 가리키므로 순서를 바꾸면 외래키에 걸린다.

! 세 벌이므로 트랜잭션도 셋이다

한 벌이 한 트랜잭션이다. 두 번째에서 실패하면 첫 번째는 이미 반영된 상태로 남으며, 어느 영역이 들어가지 못했는지 알린다.

```
오류: logical · physical 를 넣지 못했습니다. 앞의 것은 이미 들어갔습니다.
```

다시 실행해도 안전하다. 기본키가 있으면 갱신하므로 앞서 들어간 것이 겹쳐 쌓이지 않는다.

i 함께 배포하는 사전은 출발점이다

패키지에 든 표준단어 사전을 그대로 써야 하는 것이 아니다. 적재한 뒤 업무에 맞지 않는 낱말을 덜어 내고 필요한 낱말을 더한다(2장).

## 15.5 산출물 파일의 적재

프로젝트가 작성한 DM-DST 산출물 3종을 이미 구축된 메타 테이블에 적재한다. 메타데이터 파일의 적재로 생성된 레코드에 추가하는 작업이다.

```
kdsm apply app/deliverable/DM-DST-00{1,2,3}_*.xlsx --actor choonghyunryu --name-prefix AP_
```

산출물의 종류는 시트 구성으로 판별한다. 따라서 파일명과 지정 순서에는 제약이 없다.

### 15.5.1 비교 대상은 적재 대상 테이블이다

이 명령은 메타파일을 참조 사전으로 받지 않는다. 수행하는 작업이 테이블에 대한 적재이므로, 적재 가능 여부는 적재 대상 테이블만 판정할 수 있기 때문이다.

메타파일을 기준으로 삼으면 두 가지 오판이 발생한다. 첫째, 파일에는 존재하지만 테이블에는 존재하지 않는 표준을 이미 등재된 것으로 간주하여 적재를 차단한다. 둘째, 테이블에만 존재하는 항목과의 중복을 검출하지 못하여 적재 시점에 기본키 제약 위반이 발생한다. 메타파일은 담당자가 직접 수정하는 원본이므로 테이블보다 최신인 경우가 일반적이며, 두 원천의 불일치는 드물지 않다.

따라서 하나의 연결로 조회·검증·적재를 모두 수행한다.

```
from kdsm.db import apply_specs

report = apply_specs(
    ["app/deliverable/DM-DST-001_....xlsx", "app/deliverable/DM-DST-002_....xlsx",
     "app/deliverable/DM-DST-003_....xlsx"],
    actor="hryu",
    name_prefix="AP_",
)
```

`--dry-run`을 지정하여도 테이블은 조회한다. 적재를 수행하지 않을 뿐이며, 적재 가능 여부를 판정하려면 적재 대상을 확인해야 하기 때문이다.

> ! 검증을 통과한 산출물도 여기서 다시 검사한다
>
> `kdsm validate`를 통과하였더라도 적재 직전에 다시 검증한다. 검증 시점과 적재 시점 사이에 사전이 변경되었을 수 있기 때문이다. 그 사이에 다른 프로젝트가 동일한 영문약어를 등재하였다면 검증 시점에는 통과하지만 적재 시점에는 중복이 발생한다.
>
> 검증에서 오류가 발생하면 어떤 항목도 적재하지 않는다. 중복 내역은 `standard.already-registered`가 상이한 항목까지 함께 보고한다(13.3.1.2절).
>
> 논리 모델 산출물과 물리 모델 산출물을 함께 지정하면 두 산출물의 대조도 적재 이전에 실행한다(13.3.8절). `kdsm validate`와 같은 구성이다. 한쪽만 지정하면 이미 적재된 모델과 대조하지 않는다. 적재된 모델에는 다른 주제영역의 엔터티도 포함되어 있어, 대조하면 대응하는 테이블이 없다는 위반이 다수 보고되기 때문이다.

> i 3종을 하나의 트랜잭션으로 처리한다
>
> 메타데이터 파일의 적재는 파일 하나가 하나의 트랜잭션이지만, 산출물 적재는 3종을 묶어 하나의 트랜잭션으로 처리한다. 산출물 3종이 하나의 집합을 이루기 때문이다. 표준만 적재되고 모델이 누락된 상태로 남으면 그 사전은 어느 관점으로도 해석할 수 없다.
>
> Databricks는 예외이다. 트랜잭션이 없으므로 3종을 묶어도 중간에 실패하면 앞의 것이 남는다. 이 경우에는 원인을 수정한 뒤 3종을 처음부터 다시 적재한다(15.3.1절).

**표 15.6.** kdsm apply 옵션

| 옵션 | 설명 |
|---|---|
| `<산출물>` | 적재할 DM-DST 산출물. 복수 지정할 수 있다 |
| `--actor` | 작업자명 |
| `--dbms` | 넣을 DBMS. `postgresql`(기본)·`mysql`·`databricks` |
| `--catalog` | `--dbms databricks`일 때 대상 카탈로그 |
| `--schema` | 대상 스키마명. 세 DBMS 모두에 적용된다 |
| `--host` `--port` `--database` `--user` | PostgreSQL일 때의 접속 정보. 생략하면 환경변수 |
| `--env-prefix` | 접속 환경변수의 접두어 (기본 `KDSM_DB_`) |
| `--name-prefix` | 엔터티 영문명에 덧붙는 접두어. 기본은 없음 |
| `--no-validate` | 넣기 전 검증을 건너뛴다 |
| `-n`, `--dry-run` | 읽기·검증까지만 |

## 15.6 일괄 적재하기

산출물을 여러 차례 수정한 뒤에는 DDL 세 개를 다시 생성하고 메타파일 세 개를 다시 적재해야 한다. `dbms/rebuild.sh`가 그 순서를 담고 있다.

```
./dbms/rebuild.sh              # 대조 → 검증 → DDL 생성 → 적용 → 적재
./dbms/rebuild.sh --drop       # 테이블을 지우고 다시 만든다 (데이터 삭제)
./dbms/rebuild.sh --prune      # 정의서에 없는 테이블도 지운다
./dbms/rebuild.sh --dry-run    # 검증과 DDL 생성까지만
./dbms/rebuild.sh --dbms mysql # PostgreSQL 이 아닌 곳에 구축한다
./dbms/rebuild.sh --skip-apply # 스키마 적용을 건너뛴다
```

`--dbms`를 지정하면 그 DBMS에 스키마를 적용하고, `kdsm ddl`과 `kdsm load`에도 같은 값을 전달한다.

> ! `--drop` 은 앱이 등재한 사전을 되돌릴 수 없게 지운다
>
> 운영이 시작된 뒤에는 이 옵션을 쓰지 않는다. 잃는 것이 둘이다.
>
> 첫째는 신규 사전이다. 다시 세운 뒤 적재되는 것은 패키지에 든 초기 사전뿐이며, 웹 애플리케이션에서 사용자가 후보(`01`)로 올리고 관리자가 확정(`02`)한 항목은 되돌아오지 않는다. 그 값은 메타파일에도 산출물에도 사본이 없다.
>
> 둘째는 애플리케이션 계정의 권한이다. 테이블이 새로 만들어지므로 `app/dbms/init_app_postgresql.sql` 이 부여한 `INSERT` 와 `UPDATE` 가 사라진다. 그 스크립트를 다시 돌려야 앱이 사전에 쓴다.
>
> 구조를 바꿔야 하면 `ALTER` 로 옮긴다. 그래도 다시 세워야 한다면 `kdsm export` 로 현재 사전을 산출물로 내보낸 뒤에 시작한다(17장).

가장 앞의 0단계는 데이터베이스를 건드리지 않는다. `kdsm` 패키지가 함께 배포하는 정의서와 메타파일의 사본이 저장소의 정본과 같은지 대조한다(11.3.3절). 어긋나도 멈추지 않고 알리기만 한다. 사본이 어긋난 것과 데이터베이스를 다시 세우는 것은 서로 다른 일이기 때문이다.

```
python3 dbms/check-seed.py          # 대조만. 어긋나면 0이 아닌 값으로 끝난다
python3 dbms/check-seed.py --sync   # 정본을 사본에 맞추고 옛 판을 지운다
```

배포 전에 막아야 하는 자리에서는 이 명령을 직접 부른다. 종료 코드가 있으므로 CI에 그대로 등록할 수 있다. 세 가지를 찾는다. 사본에 없는 파일, 이름은 같으나 내용이 다른 파일, 그리고 사본에만 남은 옛 판이다. 파일명 뒤에 판과 작성일이 붙으므로 개정하면 이름 자체가 달라지고, 그때 옛 이름이 사본에 남는다.

표 15.7. DBMS별 스키마 적용 수단

| 대상 | 스키마를 적용하는 수단 |
|---|---|
| PostgreSQL | `psql --file` |
| MySQL | `mysql < 파일` |
| Databricks | `dbms/apply_sql.py` |

앞의 둘은 클라이언트를 설치해 두어야 한다. Databricks는 커넥터가 이미 있으므로 추가로 설치할 것이 없다.

**i 테이블을 만들 권한이 없을 때**

스키마를 만드는 권한과 테이블을 만드는 권한은 다르다. 스키마 생성까지는 통과하고 첫 `CREATE TABLE`에서 중단되는 경우가 이에 해당한다.

```
PERMISSION_DENIED: User does not have CREATE TABLE on Schema '...'
```

권한을 부여받기 어려운 조직이라면 `--skip-apply`로 3단계를 건너뛴다. 검증과 DDL 생성까지 수행한 뒤, 생성된 SQL을 권한이 있는 사람이 적용하고 나서 다시 실행하면 적재부터 이어진다.

```
./dbms/rebuild.sh --dbms databricks --skip-apply
```

`--drop`과 함께 지정할 때에는 주의한다. 삭제문이 먼저 실행되고 생성문에서 중단되면 테이블이 사라진 상태로 남는다. 권한을 확인한 뒤에 `--drop`을 사용한다.

**i PostgreSQL 이외에는 두 단계를 건너뛴다**

3-2 「정의서에 없는 테이블」과 5 「적재 결과」는 `psql`로 카탈로그를 조회한다. 대상마다 조회문을 다시 작성해야 하는데, 같은 확인을 `kdsm drift`와 `kdsm diagnose`가 DBMS를 가리지 않고 수행한다.

```
kdsm drift -s DM-LIB-006_*.xlsx --dbms mysql       # 정의서와 카탈로그 대조
kdsm diagnose --dbms mysql --schema kdsm_meta      # 적재 결과 확인
```

Databricks는 SQL 파일을 그대로 실행하는 명령이 없다. `dbms/apply_sql.py`가 그 자리를 대신하여 파일을 문장 단위로 나누어 전송한다. 접속 정보는 `kdsm`과 같은 `DATABRICKS_*`를 사용하므로 별도로 설정할 것이 없다.

검증이 가장 먼저 실행된다. 잘못된 정의로 DDL을 생성하면 그 오류가 스키마에 그대로 반영되기 때문이다. `kdsm validate`가 ERROR에 대해 1을 반환하고 `set -e`에 의해 그 지점에서 중단된다.

이 스크립트는 bash 로 작성되어 있다. Windows 에서는 Git Bash 나 WSL 에서 실행하거나, 아래의 각 단계를 직접 실행한다(20.3절).

**i 이 스크립트는 메타데이터 파일만 적재한다**

KDSM 구축 초기 단계의 일괄 작업이다. 프로젝트가 작성한 DM-DST 산출물의 적재는 포함하지 않는다. 수행 시점이 다르기 때문이다. 스키마 재구축은 저장소를 수정할 때마다 수행하지만, 산출물 적재는 해당 프로젝트가 검증을 완료한 시점에 한 번 수행한다.

두 작업을 하나의 스크립트에 포함하면 스키마만 재구축하려는 경우에도 다른 프로젝트의 산출물까지 다시 적재하게 된다.

접속 대상은 환경변수 하나로 지정한다. 스크립트 안에서 `--host` 등으로 따로 지정하지 않는다. 클라이언트와 `kdsm load`가 서로 다른 경로로 접속 정보를 받으면 스키마와 데이터가 다른 데이터베이스에 생성될 수 있다. PostgreSQL은 `KDSM_DB_*`, MySQL은 `MYSQL_*`, Databricks는 `DATABRICKS_*`를 사용한다.

⚠ `CREATE SCHEMA`는 DDL에 넣지 않는다

스크립트는 DDL을 `--no-schema`로 생성한다. `CREATE SCHEMA`는 데이터베이스에 대한 CREATE 권한을 요구하는데 운영 계정에는 이 권한이 없는 경우가 대부분이다. PostgreSQL은 `IF NOT EXISTS`를 붙여도 권한을 먼저 확인하므로, 스키마가 이미 존재하여도 오류가 발생한다.

```
psql:<파일>.sql:1: ERROR:  permission denied for database <데이터베이스>
```

스키마 생성은 한 번만 수행하는 작업이며 권한 층위도 다르다. 스크립트는 스키마의 존재를 먼저 확인하고, 없는 경우에만 생성을 시도한다.

! 이름을 바꾸면 옛 테이블이 남는다

DDL 은 현재 이름으로 새 테이블을 생성할 뿐 이전 이름의 테이블을 삭제하지 못한다. 테이블 물리명을 변경하고 DDL 을 다시 적용하면 이전 이름의 테이블이 데이터를 포함한 채로 남는다.

```
    정의서에 없는 테이블이 있습니다
      mt_std_cd_dtl (26행)
      mt_std_cd_mstr (6행)
      그대로 둡니다. 지우려면 --prune 을 주세요.
```

무엇이 남았는지는 방금 생성한 DDL 로 판단한다. DDL 에 없는 테이블이 곧 잔존 테이블이다. 이전 이름을 스크립트에 직접 기재하지 않으므로 테이블명을 변경하여도 스크립트를 수정할 필요가 없다.

기본 동작은 안내에 그친다. 담당자가 의도적으로 남겨 둔 테이블일 수 있으므로, 삭제하려면 `--prune`을 명시해야 한다.

파일이 세 개이므로 트랜잭션도 세 개이다. 두 번째에서 실패하면 첫 번째는 이미 반영된 상태이다. 스크립트가 검증을 일괄하여 먼저 수행하는 이유가 여기에 있다. 세 파일이 모두 통과한 뒤에 적재를 시작한다.

CHAPTER 16

# ERD 자동 작성

메타테이블이나 산출물을 읽어 Mermaid ERD를 작성한다. 논리 ERD는 엔터티와 속성에서, 물리 ERD는 테이블과 컬럼에서 도출된다.

```
# 포워드 엔지니어링 — 작성한 산출물에서
kdsm erd -s 고객_V0.1.0_20260812.xlsx          # 파일 하나에서
kdsm erd -s 고객_....xlsx -s 주문_....xlsx      # 주제영역별로 분할한 여러 파일에서

# 리버스 엔지니어링 — 적재된 메타테이블에서
kdsm erd                                      # 논리·물리 모두 document/out/erd 에
kdsm erd -m physical                          # 물리만
kdsm erd --meta 논리메타.xlsx --meta 물리메타.xlsx   # 데이터베이스 없이 메타파일에서
kdsm erd --model-schema kdsm_app              # 적재된 모델 중 kdsm_app 의 것만

# 리버스 엔지니어링 — DBMS 카탈로그에서
kdsm erd --from-catalog --schema shop         # 접속한 데이터베이스의 스키마에서
kdsm erd --from-catalog --dbms mysql --schema shop

# 형식
kdsm erd -f md,png                            # 마크다운과 PNG로 출력
kdsm erd -f all -m logical                    # 논리만, 모든 형식으로 출력
```

작성 대상은 `--model`, 출력 형식은 `--format`, 입력 원천은 `--spec`·`--meta`·`--from-catalog`로 지정한다.

## 16.1 세 가지 원천

ERD는 세 가지 원천에서 작성한다. 셋은 우열 관계가 아니라 사용하는 시점과 목적이 다르다.

표 16.1. ERD 작성의 세 가지 원천

| 방향 | 원천 | 옵션 | 사용 시점 |
|---|---|---|---|
| 포워드 엔지니어링 | 작성한 산출물 | `-s` | 신규 데이터 모델 개발 |

| 방향 | 원천 | 옵션 | 사용 시점 |
|---|---|---|---|
| 리버스 엔지니어링 | 적재된 메타테이블 | (기본) · `--meta` | 데이터 모델 유지보수 |
| 리버스 엔지니어링 | DBMS 카탈로그 | `--from-catalog` | 기존 데이터베이스 구조 파악 |

앞의 두 가지는 KDSM 산출물이 있는 경우이다. 세 번째는 산출물이 없거나 데이터베이스가 산출물보다 앞서 있는 경우에 사용한다. 표준화를 거치지 않은 기존 시스템을 인수받았을 때가 그 예이다.

### 16.1.1 포워드 엔지니어링: 산출물에서 작성한다

KDSM은 데이터 모델링 산출물로 DM-DST-002(논리)와 DM-DST-003(물리) 두 종의 템플릿을 제공한다(9.1절). 사용자는 이 템플릿에 데이터 모델을 작성하고, 작성한 파일에서 ERD와 DDL을 생성한다.

```
kdsm erd -s 고객_V0.1.0_20260812.xlsx -s 고객_물리_V0.1.0_20260812.xlsx
```

신규 데이터 모델을 개발하는 단계에서는 적재된 메타테이블이 존재하지 않는다. DBMS에 테이블을 생성하기 전에 ERD를 확인하는 경우가 오히려 더 많다. 모델링과 ERD 확인이 서로를 보완하며 진행되므로, 이 단계에서는 산출물이 유일한 원천이다.

#### 16.1.1.1 산출물 형식은 시트 구성으로 판별한다

논리 모델과 물리 모델 모두 `-s` 옵션 하나로 지정한다. 파일을 열어 시트 구성으로 형식을 판별하기 때문이다.

표 16.2. 산출물 형식을 가리는 기준

| 시트가 있으면 | 그 파일은 |
|---|---|
| 엔터티 정의서 | 논리 데이터 모델 (DM-DST-002 형식) |
| 테이블 정의서 · 컬럼 정의서 | 물리 데이터 모델 (DM-DST-003 형식) |

파일명으로 판별하지 않는 이유는 사용자가 파일명을 자유롭게 지정하기 때문이다. 파일명이 `고객주문_V0.1.0_20260812.xlsx`이더라도 내용이 DM-DST-002 형식이면 논리 모델로 판별한다. 파일명에 문서번호를 포함하도록 요구하지 않는다.

두 형식 중 어느 쪽에도 해당하지 않으면 판별 기준과 실제 시트 구성을 함께 제시하고 중단한다.

```
오류: 표준·논리·물리 어느 쪽 산출물인지 가릴 수 없습니다: 알수없음.xlsx
      표준은 '표준단어 정의서', 논리는 '엔터티 정의서', 물리는 '테이블 정의서'·'컬럼 정의서' 시트가
있어야 합니다.
      존재하는 시트: 표지
```

#### 16.1.1.2 파일 하나가 주제영역 하나에 대응한다

파일을 여러 개 지정하면 데이터 모델링을 주제영역별로 분할한 것으로 판단한다. 이때 파일별 ERD를 각각 작성하고, 전체를 통합한 ERD를 추가로 작성한다.

```
kdsm erd -s 데이터표준_V0.1.0_20260812.xlsx \
         -s 데이터표준_물리_V0.1.0_20260812.xlsx \
```

```
        -s 모델관리_V0.1.0_20260812.xlsx \
        -s 모델관리_물리_V0.1.0_20260812.xlsx
```

```
ERD 6장을 그렸습니다.
  논리-데이터표준 (mmd)  상자 7개 · 간선 8개
  논리-모델관리 (mmd)  상자 5개 · 간선 3개
  논리-전체 (mmd)  상자 12개 · 간선 13개
  물리-데이터표준_물리 (mmd)  상자 7개 · 간선 7개
  물리-모델관리_물리 (mmd)  상자 5개 · 간선 3개
  물리-전체 (mmd)  상자 12개 · 간선 12개
```

주제영역의 이름은 파일명에서 버전과 날짜를 제외한 부분이다. `고객주문_V0.1.0_20260812.xlsx`는 `고객주문`이 된다. 명명 규칙과 일치하지 않는 파일명은 전체를 그대로 사용한다.

표지의 문서명을 사용하지 않는 이유는 해당 항목이 비어 있을 수 있기 때문이다. 파일명은 항상 존재한다.

파일이 하나인 경우에는 통합 ERD를 별도로 작성하지 않는다. 파일별 ERD와 내용이 동일하기 때문이다.

> **i 영역 구분은 사용하지 않는다**
>
> 영역 구분은 적재된 메타테이블에서 읽는 경우에만 사용한다(16.3절). 메타테이블에는 파일이라는 경계가 없어 분할 기준을 별도로 정의해야 하지만, 산출물에서 읽는 경우에는 파일이 그 경계에 해당한다.

> **i 지정한 산출물에 해당하는 ERD만 작성한다**
>
> 물리 산출물만 지정하면 물리 ERD만 작성된다. 논리 ERD를 작성할 엔터티와 속성이 없기 때문이다.
>
> `--model logical`과 같이 산출물을 지정하지 않은 모델을 요청하면, ERD를 작성하지 않고 필요한 입력을 안내한 뒤 종료한다. 안내 없이 생략하면 ERD가 작성되지 않은 원인을 확인할 수 없다.
>
> ```
> 오류: logical 모델을 그릴 산출물이 없습니다. --spec 으로 주세요.
> ```

### 16.1.2 리버스 엔지니어링: 데이터베이스에서 복원한다

운영이 시작되면 데이터베이스가 기준이 된다. 이 시점의 ERD는 문서가 아니라 적재된 메타테이블에서 복원해야 한다. 문서 갱신이 지연되더라도 ERD는 현재 상태를 반영한다.

```
kdsm erd                                  # 데이터베이스의 kdsm_meta 에서
kdsm erd --dbms databricks --catalog dev_landing --schema kdsm_meta
kdsm erd --meta dbms/meta/meta_data/kdsm_logical_meta_*.xlsx \
         --meta dbms/meta/meta_data/kdsm_physical_meta_*.xlsx
```

`--meta`는 데이터베이스 대신 메타파일에서 읽는다. 적재 이전에 메타파일이 올바르게 작성되었는지 확인할 때 사용한다.

메타테이블을 읽을 곳은 `--dbms`로 지정한다. PostgreSQL·MySQL·Databricks 세 곳을 모두 읽으며, 생략하면 PostgreSQL이다. Databricks는 이름이 세 단이므로 `--catalog`가 필요하다.

인자 없이 실행하면 `KDSM_DB_HOST`부터 `KDSM_DB_PASSWORD`까지 여섯 환경변수로 접속한다(20.9절). `--dbms`를 지정하면 그 DBMS의 접두어를 읽는다. `-s`나 `--meta`를 지정하면 파일만 읽으므로 접속 정보가 필요하지 않다. 세 가지 방법 중 무엇을 사용하는지에 따라 환경변수의 필요 여부가 결정된다.

#### 16.1.2.1 적재된 모델을 스키마로 가린다

메타테이블에는 여러 스키마의 모델 정의가 함께 적재될 수 있다. KDSM 자신이 그 예이다. 표준 사전의 모델(`kdsm_meta`)과 웹 애플리케이션의 모델(`kdsm_app`)을 모두 적재하면 ERD에는 두 모델이 한 도면에 섞여 나타난다. `--model-schema`는 그중 지정한 스키마에 속하는 모델만 작성한다.

```
kdsm erd --model-schema kdsm_meta       # 표준 사전의 모델만
kdsm erd --model-schema kdsm_app        # 웹 애플리케이션의 모델만
kdsm erd --model-database aiplatform --model-schema kdsm_app   # aiplatform 데이터베이스의 kdsm_app
만
```

스키마의 이름은 데이터베이스 안에서만 고유하다. 같은 이름의 스키마가 두 데이터베이스에 있으면 `--model-schema`만으로는 두 모델이 한 도면에 섞인다. `--model-database`를 함께 지정하면 그 데이터베이스의 것만 작성한다. 이 옵션도 메타테이블에 접속할 데이터베이스를 가리키는 `--database`와 다르다.

> ! `--schema`와는 다른 옵션이다
>
> `--schema`는 메타테이블이 존재하는 스키마를 가리키고, `--model-schema`는 적재된 모델 중 어느 스키마의 것을 작성할지를 지정한다. 이름이 유사하여 혼동하기 쉬우므로 도움말에도 이 차이를 기재해 두었다.

물리 모델은 테이블 정의서의 `스키마` 항목으로 직접 판별한다. 논리 모델에는 스키마 항목이 없다. 엔터티는 물리 이전의 개념이기 때문이다. 그래서 해당 스키마의 테이블이 `관련 엔터티`로 가리키는 엔터티로 범위를 좁힌다. 두 모델이 같은 근거를 사용하므로 논리와 물리가 같은 범위를 나타낸다. 엔터티 영문명의 접두어(`MT_`·`AP_`)로 판별하지 않는 이유는, 이름 규칙에 의존하면 규칙이 지켜지지 않았을 때 잘못된 결과를 안내 없이 내기 때문이다.

물리 정의가 없는 엔터티는 어느 스키마에도 속하지 않으므로 도면에서 제외된다. 아직 물리로 구현되지 않았다는 사실의 반영이다. 제외된 엔터티는 `ErdReport.schemaless`에 기록되므로 안내 없이 누락되지 않는다.

엔터티가 하나도 없는 도면은 작성하지 않는다. 스키마로 판별하면 영역 전체가 비는 경우가 많다. `--model-schema kdsm_app`을 지정하면 KDSM 표준 사전의 세 영역이 모두 비고, 웹 애플리케이션 영역 한 장이 모델마다 작성되어 모두 두 장이 된다(16.3절). 전체 도면은 범위를 판별한 뒤에도 엔터티가 남은 영역이 둘 이상일 때만 작성한다. 남은 영역이 하나이면 전체 도면이 그 영역 도면과 같은 내용이기 때문이다.

> 두 방향의 결과가 다른 경우
>
> 동일한 시점이라면 두 ERD는 일치해야 한다. 일치하지 않는다면 산출물과 데이터베이스가 어긋난 것이다. 어느 쪽을 기준으로 삼을지는 시점에 따라 결정된다. 적재 이전이면 산출물이 기준이고, 운영 중이면 데이터베이스가 기준이다.
>
> 두 ERD를 육안으로 비교하는 것보다 `kdsm validate`로 대조하는 것이 정확하다. 어느 항목이 어긋났는지까지 확인할 수 있다(13.3.8절).

## 16.2 작성 대상과 출력 형식

표 16.3. 작성 대상과 출력 형식

| 옵션 | 지정할 수 있는 값 | 기본값 |
|---|---|---|
| `-m`, `--model` | `logical` `physical` `all` | `all` |
| `-f`, `--format` | `mmd` `md` `png` `svg` `pdf` `all` | `mmd,md` |

쉼표로 연결하여 지정하거나 옵션을 여러 번 지정한다. 두 방식의 결과는 동일하다.

```
kdsm erd -f md,png
kdsm erd -f md -f png
```

> **i** 기본값의 근거
>
> `--model`의 기본값이 `all`인 이유는 한 모델만으로는 확인할 수 없는 사항이 있기 때문이다. 논리 모델에만 존재하는 관계는 두 ERD를 대조해야 드러난다(16.6.5절). 매번 두 번 실행해야 한다면 이 대조를 수행하지 않게 된다.
>
> `--format`의 기본값에 이미지 형식이 없는 이유는 이미지 생성에 mermaid-cli가 필요하기 때문이다. 기본 동작이 외부 도구를 요구하면 해당 도구가 설치되지 않은 환경에서는 아무 결과도 얻을 수 없다.

## 16.3 영역 설정

이 절은 적재된 메타테이블에서 읽는 경우에 해당한다. 산출물에서 읽는 경우에는 파일이 경계이므로 별도의 설정이 필요하지 않다(16.1.1.2절).

12개 엔터티를 한 장에 작성하면 관계선이 복잡하게 얽힌다. 그래서 영역별로 한 장씩 작성하고, 전체를 담은 한 장을 추가로 작성한다. 기본 설정에서는 8장이 생성된다.

```
논리-데이터 표준      논리-논리 데이터 모델   논리-물리 데이터 모델   논리-전체
물리-데이터 표준      물리-논리 데이터 모델   물리-물리 데이터 모델   물리-전체
```

엔터티·테이블과 영역의 대응은 주제영역구성(`MT_SBJAR_CPST`)이 담는다. 논리 정의서의 `주제영역구성 정의서` 시트에 작성하여 적재한 것이며, 주제영역 화면과 검증 규칙도 같은 것을 참조한다. 명명 규칙으로 유추하지 않는다.

표 16.4. 주제영역구성 정의서의 작성 예

| 소속 주제영역 | 구성 순서 | 구성대상 | 구성대상 이름 |
|---|---|---|---|
| 데이터 표준 | 1 | 엔터티 | 표준단어 |
| 데이터 표준 | 2 | 엔터티 | 표준용어 |
| 논리 데이터 모델 | 1 | 엔터티 | 주제영역 |

구성대상이 `테이블`인 행은 물리 ERD의 분할에 사용한다. 테이블이 한 행도 적혀 있지 않은 영역은 테이블의 `관련 엔터티`가 가리키는 엔터티의 영역을 따른다.

배정이 한 줄도 없으면 `document/areas.toml`로 대체한다. 주제영역 시트가 없는 이전 판 메타파일로 작성하는 경우이며, 빈 영역 구분을 그대로 사용하면 도면이 한 장도 생성되지 않는다. `--areas`로 파일을 지정하면 배정이 있더라도 그 파일이 우선한다. 지정한 것을 원천이 덮으면 옵션을 지정한 의미가 없어지기 때문이다.

```
# document/areas.toml — 배정이 비어 있을 때의 대비책
[[area]]
name = "데이터 표준"
logical = "DM-LIB-001"
physical = "DM-LIB-004"
entities = ["표준단어", "표준용어", "표준도메인", ...]
```

위 예시는 KDSM 자신의 설정이므로 문서번호가 DM-LIB이다. 사업에서는 그 사업의 영역 이름과 산출물 문서번호를 기재한다(9.1절).

> ⚠ 엔터티를 추가하면 배정도 함께 작성한다
>
> 배정이 없는 엔터티는 영역별 ERD에 포함되지 않는다. 전체 ERD에는 나타나므로 영역별 ERD만 확인하면 누락을 발견하기 어렵다. 검증 규칙 `composition.entity-assigned`가 그 누락을 이름을 들어 보고한다.

> 💡 `-f png`만 지정해도 된다
>
> `.mmd`를 함께 지정하지 않아도 이미지가 생성된다. mermaid-cli는 소스 파일을 입력으로 받으므로 원본이 필요한데, 이 경우 원본은 임시 디렉터리에 생성하고 이미지만 남긴다. 저장소에 이미지만 보관하려는 경우에 사용한다.

### 16.3.1 리버스 엔지니어링: DBMS 카탈로그에서 작성한다

KDSM 산출물이 없는 데이터베이스에서는 카탈로그가 유일한 근거이다. 표준화를 거치지 않은 기존 시스템을 인수받아 구조를 파악해야 하는 경우가 이에 해당한다. `--from-catalog`를 지정하면 `information_schema`를 읽어 ERD를 작성한다.

```
kdsm erd --from-catalog --schema shop           # PostgreSQL (기본)
kdsm erd --from-catalog --dbms mysql --schema shop
kdsm erd --from-catalog --dbms databricks --schema main.sales
kdsm erd --from-catalog --schema shop --schema hr -f png
```

접속 정보는 `KDSM_DB_*` 환경변수에서 읽고 `--host`·`--port`·`--database`·`--user`로 덮어쓴다. 비밀번호는 인수로 받지 않는다(20.9절).

DBMS가 PostgreSQL이 아니면 그 DBMS의 접두어를 사용한다. MySQL은 `MYSQL_*`, Databricks는 `DATABRICKS_*`이다. Databricks에서는 `--host`(서버 호스트명)와 `--catalog`만 인자로 받는다. `--port`·`--database`·`--user`는 접속 방식이 달라 사용하지 않는다.

```
# Databricks — OAuth M2M (서비스 프린시펄)
export DATABRICKS_HOSTNAME='adb-1234.azuredatabricks.net'
export DATABRICKS_CLIENT_ID='xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
export DATABRICKS_CLIENT_SECRET='dose...'
```

```
export DATABRICKS_WAREHOUSE_ID='abc123'      # 또는 DATABRICKS_HTTP_PATH

kdsm erd --from-catalog --dbms databricks --schema main.sales -f png
```

인증 방식과 변수 목록은 18.5.4.1절, 웨어하우스를 가리키는 방법은 18.5.4.2절 에 있다. 진단과 ERD가 같은 접속 경로를 쓰므로 구성이 동일하다.

표 16.5. 읽을 수 있는 DBMS

| DBMS | `--dbms` | 범위(`--schema`)가 가리키는 것 | 필요한 선택 의존성 |
|---|---|---|---|
| PostgreSQL | `postgresql` (기본) | 스키마 이름 | `db` |
| MySQL | `mysql` | 데이터베이스 이름 | `mysql` |
| Databricks | `databricks` | `카탈로그.스키마` | `databricks` |

`--dbms`와 필요한 선택 의존성은 메타테이블을 읽을 때에도 같다. `--from-catalog` 없이 실행하면 같은 `--dbms`가 메타테이블이 있는 곳을 가리킨다(16.1.2절).

다만 Databricks의 범위 지정 방식은 다르다. 카탈로그를 읽을 때에는 `--schema`에 `카탈로그.스키마`를 점으로 이어 지정하지만, 메타테이블을 읽을 때에는 `--catalog`와 `--schema`를 나누어 지정한다. 메타테이블은 이름의 세 단을 각각 받아 조립하므로, 여기에 점으로 이은 이름을 주면 네 단이 되어 없는 테이블을 찾게 된다.

MySQL에는 데이터베이스 아래에 스키마를 두는 층이 없다. `information_schema`가 데이터베이스 이름을 스키마 자리에 그대로 돌려주므로, `--schema`에 데이터베이스 이름을 지정한다.

#### 16.3.1.1 물리 모델만 작성한다

카탈로그에는 엔터티와 속성이라는 층이 없다. 그래서 `--model`을 보지 않고 물리 ERD만 작성한다. 논리 ERD가 필요하다면 산출물이나 메타테이블이 있어야 한다.

#### 16.3.1.2 한글명은 주석에서 가져온다

표준을 따르지 않은 데이터베이스에서 한글명을 담을 자리는 주석뿐이다. 테이블과 컬럼의 주석을 읽어 물리명 옆에 병기하고, 주석이 없으면 물리명만 기재한다. 없는 것을 지어내지 않는다.

```
"CUST 고객" {
    VARCHAR(20) CUST_ID PK "고객번호"
    VARCHAR(100) CUST_NM
}
```

주석을 `한글명 - 설명` 형식으로 작성한 경우에는 앞의 한글명만 가져온다. 이때 가르는 기준은 앞뒤에 공백이 있는 붙임표뿐이며, `고객-법인 구분`처럼 붙여 쓴 붙임표는 이름의 일부로 본다.

주석이 부정확한 환경에서는 `--no-names`로 병기를 중단한다. 잘못된 한글명은 없는 것만 못하다.

#### 16.3.1.3 스키마마다 한 장이 된다

카탈로그가 실제로 가진 경계는 스키마뿐이므로 주제영역 설정(`--areas`)을 받지 않는다. 스키마를 여럿 지정하면 스키마마다 한 장씩 작성하고, 전체를 담은 도면을 한 장 더 작성한다. 스키마가 하나이면 전체 도면은 같은 내용이므로 작성하지 않는다.

#### 16.3.1.4 관계선은 선언된 외래키만 나타난다

> ⚠ 외래키를 설정하지 않은 관계는 나타나지 않는다
>
> 카탈로그에서 관계를 알 수 있는 근거는 외래키 제약뿐이다. 실제로는 참조하고 있으나 제약을 설정하지 않았다면 그 관계는 도면에 나타나지 않는다. 애플리케이션에서만 참조 무결성을 관리하는 데이터베이스에서는 도면이 비어 보일 수 있다.
>
> Databricks Unity Catalog의 기본키·외래키는 강제되지 않는 정보성 제약이다. 데이터베이스가 위반을 막아 주지 않으므로, 읽어 온 관계는 「선언된 것」이지 「지켜지고 있는 것」이 아니다. 선언할 수 있게 된 시점과 확인 방법은 18.5.4.6절 에 있다.

식별관계와 선택 여부는 다음 기준으로 판정한다. 산출물에서 작성할 때와 같은 기준이다.

표 16.6. 카탈로그에서 판정하는 관계의 성격

| 표기 | 판정 기준 |
|---|---|
| 식별관계 (실선) | 외래키의 구성 컬럼이 모두 기본키에 포함된다 |
| 필수 (`\|\|`) | 외래키의 구성 컬럼 중 하나라도 NOT NULL이다 |

조회 범위 밖의 테이블을 가리키는 외래키는 관계선을 작성하지 않는다. 상자가 없는 쪽으로 선을 그으면 Mermaid가 빈 상자를 만들어 내어 도면이 사실과 달라지기 때문이다. 대상 테이블이 다른 스키마에 있다면 그 스키마도 `--schema`에 함께 지정한다.

### 16.3.2 메타테이블 대신 카탈로그를 읽으면 잃는 것

KDSM 산출물이 있다면 카탈로그가 아니라 산출물이나 메타테이블을 읽는 것이 낫다. 카탈로그에는 다음 정보가 없다.

```
표준단어 ─1:N─> 표준용어      논리에는 있는데 외래키는 없다
```

표준용어의 구성단어(`CPST_STD_WORD_CN`)는 여러 값을 `+`로 연결한 단일 항목이므로 외래키를 설정할 수 없다. 관계가 존재하지 않는 것이 아니라 항목의 구조가 이를 표현하지 못하는 것이다. 메타테이블은 이 관계를 `주요 관계` 항목에 기재해 두고 있다.

메타테이블은 이 밖에도 카탈로그가 보유하지 않은 정보를 가지고 있다. `주요 관계`에 카디널리티가 기재되어 있고, 엔터티명이 한글이므로 사람이 읽기 쉬운 ERD가 작성된다.

> ℹ 표준화 수준까지 확인하려면 진단을 사용한다
>
> `--from-catalog`는 구조를 그려 보여 줄 뿐 판정하지 않는다. 이름이 표준을 따르는지, 도메인이 일치하는지까지 확인하려면 `kdsm diagnose`를 사용한다(18장). 두 명령이 같은 카탈로그 리더를 공유하므로 접속 방법과 `--dbms`·`--schema` 사용법이 동일하다.

## 16.4 전체 옵션

표 16.7. kdsm erd 옵션

| 옵션 | 뜻 |
|---|---|
| `-o`, `--output` | ERD를 저장할 디렉터리. 기본값은 `<저장소>/document/out/erd` |
| `--areas` | 영역 구분 TOML. 지정하면 적재된 배정보다 우선한다. 생략하면 배정을 사용하고, 배정이 비어 있으면 `<저장소>/document/areas.toml` |
| `-m`, `--model` | 작성 대상. 표 16.3 참고 |
| `--no-attributes` | 엔터티 내부를 생략하고 관계선만 작성한다 |
| `--no-combined` | 전체를 담은 통합 ERD를 작성하지 않는다 |
| `--model-schema` | 적재된 모델 중 지정한 스키마의 것만 작성한다. 16.1.2.1절 참고 |
| `--model-database` | 적재된 모델 중 지정한 데이터베이스의 것만 작성한다. 16.1.2.1절 참고 |
| `-f`, `--format` | 출력 형식. 표 16.3 참고 |
| `--width` `--scale` `--background` `--no-pdf-fit` `--pdf-merge` `--mmdc` `--chrome` `--puppeteer-config` | 이미지 생성 관련. 표 16.10 참고 |
| `-s`, `--spec` | 데이터 모델링 산출물에서 읽는다. 여러 번 지정할 수 있다 |
| `--meta` | 데이터베이스 대신 메타파일에서 읽는다. 여러 번 지정할 수 있다 |
| `--from-catalog` | DBMS 카탈로그에서 읽는다. 16.3.1절 참고 |
| `--dbms` | 읽을 DBMS. 메타테이블과 카탈로그 어느 쪽에도 적용된다. 표 16.5 참고 |
| `--catalog` | `--dbms databricks` 일 때 카탈로그 |
| `--no-names` | `--from-catalog` 일 때 주석에서 한글명을 가져오지 않는다 |
| `--host` `--port` `--database` `--user` | `--from-catalog` 일 때의 접속 정보. 생략하면 환경변수 |
| `--env-prefix` | 접속 정보의 환경변수 접두어 |
| `--schema` | 대상 스키마명. `--from-catalog` 일 때는 여러 번 지정할 수 있다 |
| `-q`, `--quiet` | 한 줄 요약만 출력한다 |

## 16.5 ERD 파일 출력 형식

출력 형식은 두 종류로 구분된다. Mermaid 소스 파일과 렌더링된 이미지 파일이다. 소스 파일에는 ERD의 구조와 문법이 기록되고, 이미지 파일은 그 소스를 mermaid-cli가 렌더링한 결과이다.

표 16.8. ERD 파일 출력 형식

| 구분 | 확장자 | 용도 | 필요한 도구 |
|---|---|---|---|
| Mermaid 소스 | `.mmd` | mermaid-cli 등의 도구에 그대로 입력한다 | 없음 |

| 구분 | 확장자 | 용도 | 필요한 도구 |
|---|---|---|---|
| Mermaid 소스 | `.md` | 사람이 읽는다. GitHub와 VS Code가 추가 설치 없이 렌더링한다 | 없음 |
| 렌더링 이미지 | `.png` | 문서에 삽입하고 인쇄한다. 가장 널리 쓰이는 형식이다 | mermaid-cli |
| 렌더링 이미지 | `.svg` | 확대해도 선명하다. 웹 문서와 편집 도구에 적합하다 | mermaid-cli |
| 렌더링 이미지 | `.pdf` | 벡터 형식이며 인쇄와 배포에 적합하다 | mermaid-cli |

기본값은 소스 파일 두 형식(`mmd,md`)이다. 이미지 형식을 기본에 넣지 않은 것은 mermaid-cli가 필요하기 때문이다(16.2절).

## 16.6 Mermaid 소스 파일

소스 파일은 `erDiagram` 문법으로 기록된 텍스트이다. 이 절에서는 파일의 형식을 먼저 보고, 이어서 그 안에 무엇이 어떤 근거로 기록되는지를 다룬다. 관계선이 어디에서 도출되는지, 식별관계를 무엇으로 판정하는지, 표기를 어떻게 읽는지가 그 내용이다.

`.mmd`와 `.md`는 내용이 같다. `.md`는 `.mmd`를 펜스 블록으로 감싼 것이다. 두 파일을 각각 생성하지 않는 이유는 그 경우 내용이 달라질 수 있고, 그때 어느 쪽이 정확한지 판단할 근거가 없기 때문이다.

````
```mermaid
---
title: 논리 데이터 모델 (논리 모델)
---
erDiagram
    "엔터티" ||--o{ "속성" : "엔터티이름"
```
````

> **i** `.mmd`만으로는 확인하기 어렵다
>
> 많은 도구가 `.mmd`를 그대로 렌더링하지 않는다. GitHub는 마크다운 안의 펜스 블록만 렌더링하고, VS Code의 기본 Mermaid 지원도 `.md` 파일에 한정된다. `.mmd`를 미리 보려면 별도의 확장을 설치해야 한다.
>
> 그러므로 `.md`를 함께 생성한다. 표준 확장자라고 하여 확인이 편리한 것은 아니므로, 결과를 확인할 환경에서 열리는 형식으로 함께 두는 편이 낫다.

한 장만 빠르게 확인하려면 mermaid.live에 붙여 넣는 방법이 가장 간편하다.

```
pbcopy < "document/out/erd/논리-전체.mmd"
```

소스 파일은 텍스트이므로 `git diff`로 변경 내용을 확인할 수 있다. 이미지 파일로는 모델의 변경을 확인하기 어렵다. 저장소에서 이력을 관리할 대상은 소스 파일이고, 이미지는 배포 시점에 생성하는 편이 적절하다.

## 16.6.1 관계선의 두 가지 출처

### 16.6.1.1 주 출처: 속성의 참조 엔터티

속성 정의서의 `참조 엔터티`·`참조 속성이름` 항목이다. 어느 속성이 어느 엔터티의 어떤 속성을 참조하는지가 한 행에 기재되어 있으므로, 관계선의 양끝과 레이블을 함께 도출할 수 있다.

동일한 두 엔터티 사이에 참조가 여러 개이면 한 행으로 통합한다.

```
"데이터베이스" ¦¦..o{ "테이블" : "데이터베이스이름 + 스키마이름"
```

관계선을 두 개 그리는 것보다 한 관계선에 두 이름을 표기하는 편이 읽기 쉽다.

### 16.6.1.2 보조 출처: 엔터티의 주요 관계

`표준용어(1:N)`와 같이 상대 엔터티와 카디널리티만 기재된 개략적인 기록이다. 참조로 이미 표현된 관계선은 제외하고, 속성으로 표현할 수 없는 관계만 채택한다. 참조 쪽이 레이블과 식별관계까지 담고 있어 더 정확하기 때문이다.

채택한 관계선은 리포트에 기재한다.

```
속성으로는 표현할 수 없어 주요 관계에서 살린 간선:
  표준단어 → 표준용어  (표준단어.주요 관계)
```

> ! 두 출처를 모두 읽어야 한다
>
> 두 항목은 용도가 다르므로 서로를 포괄하지 못한다. `주요 관계`는 엔터티를 설명하기 위한 항목이고, `참조 엔터티`는 구조를 기록하는 항목이다.
>
> 참조 항목만 읽으면 값을 여러 개 기록하는 항목으로 맺어진 관계가 빠지고(1.4절), 주요 관계만 읽으면 참조 항목에만 기재된 관계가 빠진다. 어느 한쪽만 읽어 그린 ERD 는 실제 구조보다 관계선이 적다.

## 16.6.2 식별관계는 기본키가 정한다

부모 엔터티의 키가 자식 엔터티의 키에 포함되면 식별관계이며, Mermaid에서 실선으로 표현된다. 따라서 참조하는 속성에 `기본키 순번`이 있는지로 판정한다.

`상위 엔터티` 항목은 판정에 사용하지 않는다. 이 항목이 실제로는 소속의 의미로 사용되고 있기 때문이다. 두 값이 일치하지 않으면 리포트로 보고한다.

```
상위 엔터티와 기본키가 어긋납니다:
  표준도메인분류: 상위 엔터티는 표준도메인그룹인데 기본키(표준도메인분류코드)에
                  표준도메인그룹의 키가 없습니다.
  테이블: 상위 엔터티는 데이터베이스인데 기본키(테이블물리이름)에
          데이터베이스의 키가 없습니다.
```

어느 쪽을 기준으로 삼을지는 사람이 판단할 사항이다. 기본키를 복합키로 변경하여 식별관계로 만들 수도 있고, `상위 엔터티` 항목을 비워 비식별관계임을 명시할 수도 있다. 불일치가 있으면 `kdsm erd`는 ERD를 작성한 뒤 종료 코드 1을 반환한다. ERD만 확인해서는 이 불일치를 발견할 수 없기 때문이다.

## 16.6.3 표기법

**표 16.9.** ERD 표기

| 표기 | 뜻 |
|---|---|
| `\|\|--o{` | 실선. 식별관계. 부모 하나에 자식 여럿 |
| `\|\|..o{` | 점선. 비식별관계 |
| `\|o..o{` | 부모 쪽이 없을 수 있다. 참조하는 속성이 Null을 허용하는 경우 |
| `PK` / `FK` / `PK, FK` | 기본키 / 참조 / 둘 다 |

### 16.6.4 작성 결과 확인

> **i** 영역을 넘는 관계선은 영역별 ERD에 포함되지 않는다
>
> 양끝이 모두 해당 영역에 있는 관계선만 남긴다. 한쪽만 걸친 관계선을 그리면 엔터티가 없는 위치로 선이 이어져 Mermaid가 빈 엔터티를 생성한다.
>
> 따라서 영역별 ERD의 관계선을 모두 합해도 전체 ERD보다 적다. 현재는 8 + 1 + 2 = 11이고 전체는 13이다. 누락된 두 개는 속성이 표준도메인을 참조하는 관계선과 컬럼이 표준코드마스터를 참조하는 관계선으로, 논리 데이터 모델 영역이 데이터 표준 영역을 참조하는 부분이다.

다음은 논리 데이터 모델 영역이다. 엔터티가 두 개이므로 속성까지 표시해도 읽을 수 있다.

논리 데이터 모델 (논리 모델)

| 엔터티 | | |
|---|---|---|
| 명V40 | 엔터티이름 | PK |
| 명V40 | 엔터티영문이름 | |
| 내용V1000 | 엔터티설명 | |
| 명V200 | 기본키 | |
| 코드C2 | 엔터티구분코드 | |
| 명V40 | 상위엔터티이름 | |
| 여부C1 | 개인정보포함여부 | |
| 내용V80 | 주요관계내용 | |
| 연월일시분초D | 생성일시 | |
| 명V20 | 생성자 | |
| 연월일시분초D | 수정일시 | |
| 명V20 | 수정자 | |




| 속성 | | |
|---|---|---|
| 명V40 | 엔터티이름 | PK,FK |
| 명V40 | 속성이름 | PK |
| 명V80 | 속성영문이름 | |
| 코드C2 | 속성유형코드 | |
| 명V40 | 도메인이름 | FK |
| 일련번호N2 | 기본키일련번호 | |
| 여부C1 | 널허용여부 | |
| 여부C1 | 개인정보여부 | |
| 명V40 | 참조엔터티이름 | |
| 명V40 | 참조속성이름 | |
| 내용V256 | 속성기본값내용 | |
| 내용V1000 | 속성설명 | |
| 연월일시분초D | 생성일시 | |
| 명V20 | 생성자 | |
| 연월일시분초D | 수정일시 | |
| 명V20 | 수정자 | |

그림 16 ㅣ ERD)

데이터 표준 영역은 엔터티가 일곱 개이므로 관계선만 표시한다.

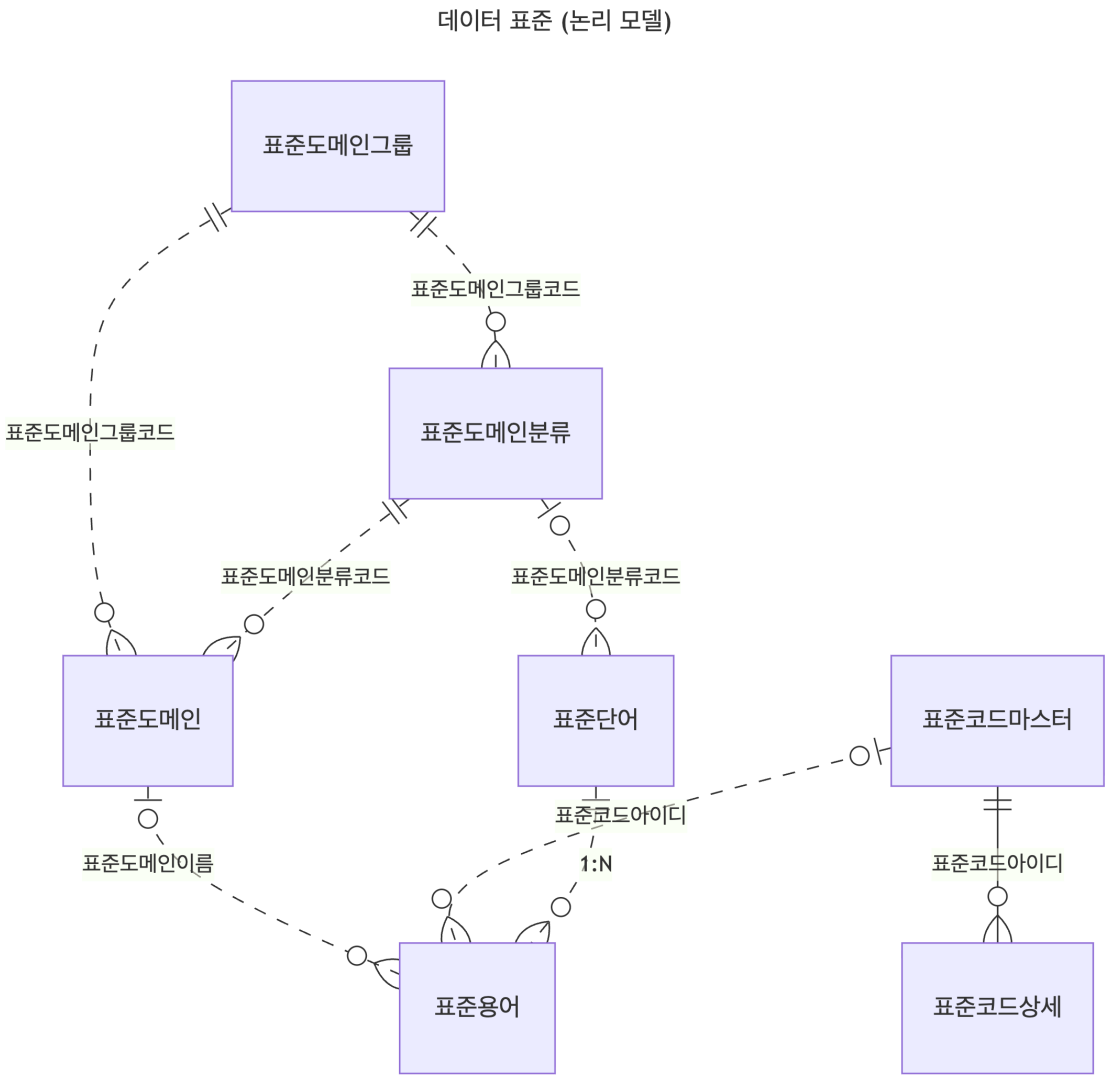


그림 16.2. 데이터 표준 영역 (관계선만)

그림 16.2 의 원본은 다음과 같다.

```
erDiagram
    "표준도메인분류" |o..o{ "표준단어" : "표준도메인분류코드"
    "표준도메인" |o..o{ "표준용어" : "표준도메인이름"
    "표준코드마스터" |o..o{ "표준용어" : "표준코드아이디"
    "표준도메인그룹" ||..o{ "표준도메인" : "표준도메인그룹코드"
    "표준도메인분류" ||..o{ "표준도메인" : "표준도메인분류코드"
    "표준도메인그룹" ||..o{ "표준도메인분류" : "표준도메인그룹코드"
    "표준코드마스터" ||--o{ "표준코드상세" : "표준코드아이디"
    "표준단어" ||..o{ "표준용어" : "1:N"
```

마지막 행이 주요 관계에서 채택한 관계선이다. 레이블이 속성명이 아니라 `1:N`인 것은 이를 뒷받침하는 속성이 없다는 뜻이다.

#### 16.6.4.1 문서에 게재할 때에는 이미지로 변환한다

> 렌더러에 따라 결과가 다르다
>
> Mermaid 텍스트를 문서 도구에 그대로 넣으면 도구가 내장한 Mermaid 버전으로 렌더링된다. 버전이 다르면 관계 기호의 모양과 위치가 달라지므로, 문서에 게재할 ERD는 mermaid-cli 로 이미지를 생성하여 그 이미지를 넣는 편이 결과가 일정하다.
>
> 인쇄를 함께 고려한다면 `--scale 3`으로 지정한다(표 16.10). 파일이 지나치게 커지면 이 값을 낮춘다.

### 16.6.5 물리 ERD와의 대조

물리 ERD는 테이블과 컬럼으로 동일한 구조를 표현한다. 엔터티 안의 `type` 위치에 도메인명 대신 데이터 타입이 표기된다.

```
"MT_ENTTY" ¦¦--o{ "MT_ATRB" : "ENTTY_NM"
"MT_DB" ¦¦..o{ "MT_TBL" : "DB_NM + SCHM_NM"
```

이 12개의 관계선은 `dbms/ddl/*.sql`의 `FOREIGN KEY` 12개와 정확히 일치한다. 부모 테이블과 컬럼도 모두 동일하다. 두 ERD를 대조하면 논리 모델에만 존재하는 관계선이 드러난다. 현재는 표준단어와 표준용어 사이의 관계 하나이다.

> ERD를 검증에 활용하기
>
> 논리 13개, 물리 12개라는 수치 자체가 검사 기준이 된다. 논리 모델에 관계를 추가했는데 물리 모델의 수치가 그대로라면 물리 모델에 반영하지 않은 것이고, 반대의 경우라면 문서 없이 외래키를 설정한 것이다. `kdsm validate`의 대조 규칙(13.3.8절)은 컬럼 단위로 판정하고, ERD는 전체 구조를 한 화면에서 제시한다.

## 16.7 렌더링 이미지 파일

`--format`에 `png`·`svg`·`pdf`를 지정하면 이미지도 함께 생성한다. Node.js와 mermaid-cli 설치 방법은 20.6절 에 있다.

```
npm install -g @mermaid-js/mermaid-cli          # 한 번만
kdsm erd -f mmd,md,png                          # 소스 두 형식과 PNG
kdsm erd -f png --scale 3                       # 더 선명하게
kdsm erd -f png --background transparent        # 배경 없이
kdsm erd -f pdf                                 # ERD 한 장을 PDF 한 페이지로
kdsm erd -f pdf --pdf-merge                     # 전체를 PDF 한 파일로 병합
```

렌더링은 mermaid-cli(`mmdc`)가 수행한다. 글꼴을 적용하여 문자를 배치하는 작업이므로 브라우저 엔진이 필요하며, 파이썬만으로는 처리할 수 없다.

표 16.10. 이미지 생성 옵션

| 옵션 | 기본값 | 뜻 |
|---|---|---|
| `--scale` | 2 | 픽셀 배율. 선명도를 결정한다 |
| `--width` | 1600 | 가로 픽셀. ERD가 이보다 넓은 경우에만 의미가 있다 |

| 옵션 | 기본값 | 뜻 |
|---|---|---|
| `--background` | `white` | 배경색. `transparent`도 지정할 수 있다 |
| `--no-pdf-fit` | | PDF를 인쇄 규격에 맞춘다. 기본은 도면 크기에 맞춘 한 페이지 |
| `--pdf-merge` | | 생성한 PDF를 한 파일로 병합한다. 파일명을 생략하면 `ERD.pdf` |
| `--mmdc` | | mermaid-cli 경로 |
| `--chrome` | | 렌더링에 사용할 브라우저. 값 없이 지정하면 자동으로 찾는다 |
| `--puppeteer-config` | | puppeteer 설정 JSON. `--chrome`보다 우선한다 |

**! 해상도는 `--width`가 아니라 `--scale`이 결정한다**

`--width`를 늘리면 이미지가 커질 것으로 예상하기 쉬우나 그렇지 않다. Mermaid는 다이어그램을 자연 크기까지만 그리고 남는 폭은 비워 둔다. mermaid-cli는 그 자연 크기만큼만 잘라 출력하므로, 폭을 두 배로 지정해도 결과는 동일하다.

실제로 픽셀 수를 늘리는 것은 `--scale`이다. 브라우저의 화면 배율을 높여 동일한 다이어그램을 더 높은 밀도로 출력한다. 선명도가 부족하면 이 값을 높이다.

```
kdsm erd -f png --scale 3      # 픽셀이 2배 대비 2.25배
```

파일 크기는 대체로 배율의 제곱에 비례하여 증가한다. `--width`는 다이어그램이 화면보다 넓어지는 경우에만 조정한다.

**! PDF는 도면 크기에 맞춘 한 페이지로 생성한다**

mermaid-cli는 기본적으로 브라우저의 인쇄 설정으로 PDF를 생성한다. A4 크기의 페이지에 도면을 배치하므로 도면이 페이지보다 크면 여러 장으로 분할되고, 경계에 걸친 엔터티는 두 장에 걸쳐 잘린다. 첫 장이 비어 나오는 것도 같은 원인이다.

그래서 `--pdfFit`을 기본으로 지정한다. 페이지 크기를 도면 크기에 맞추고 출력 범위를 한 장으로 고정하므로, 엔터티가 분할되지 않고 도면 전체가 한 페이지에 들어간다.

```
kdsm erd -f pdf                # 도면 크기에 맞춘 한 페이지 (기본)
kdsm erd -f pdf --no-pdf-fit   # 인쇄 규격에 맞춘 여러 페이지
```

한 페이지가 지나치게 커지는 경우에는 영역별 ERD를 사용한다. 영역마다 한 장씩 생성되므로 각 PDF가 한 페이지이며 엔터티도 분할되지 않는다(16.3절). 산출물에서 읽는 경우에는 파일 하나가 곧 한 장이다(16.1.1.2절).

! 여러 도면을 PDF 한 파일로 병합한다

PDF는 이미지 형식과 달리 여러 도면을 페이지 단위로 한 파일에 담을 수 있다. `--pdf-merge`를 지정하면 생성한 PDF를 한 파일로 병합한다. 도면 하나가 한 페이지가 되며, 페이지 순서는 생성 순서를 따른다. 논리 모델이 앞에, 물리 모델이 뒤에 온다.

```
kdsm erd -f pdf --pdf-merge                 # ERD.pdf 로 병합한다
kdsm erd -f pdf --pdf-merge 데이터모델.pdf   # 이름을 지정한다
```

```
논리-데이터 표준.pdf   →  1쪽
물리-데이터 표준.pdf   →  2쪽
                          ERD.pdf
```

페이지 크기는 원본을 그대로 따른다. 도면마다 크기가 다르고 각 PDF가 이미 자기 도면에 맞춘 한 페이지이므로, 여기에서 규격을 통일하면 앞에서 맞춘 크기가 다시 어긋난다.

개별 PDF도 그대로 남긴다. 배포용으로 합본만 필요한 경우에는 개별 파일을 삭제한다.

병합에는 pypdf가 필요하다. `uv sync --extra pdf`로 설치한다. 이 기능을 사용하지 않으면 설치하지 않아도 된다.

⚠ 브라우저를 찾지 못하는 경우

```
Error: Could not find chrome-headless-shell (ver. 150.0.7871.24)
```

mermaid-cli는 puppeteer로 브라우저를 실행하여 렌더링한다. 이 메시지는 해당 브라우저가 없을 때 나타나며, 원인은 여러 가지이다. Homebrew로 설치하여 브라우저가 함께 설치되지 않았거나, npm으로 설치했으나 설치 스크립트가 차단되어 내려받지 못했거나, 캐시 경로가 일치하지 않는 경우이다.

다음 세 가지 중 하나로 해결한다.

```
kdsm erd -f png --chrome                          # 1) 설치된 크롬을 사용한다

npx puppeteer browsers install chrome-headless-shell   # 2) 브라우저를 내려받는다

brew uninstall mermaid-cli                        # 3) npm으로 다시 설치한다
npm install -g @mermaid-js/mermaid-cli
```

내려받기가 차단된 망에서는 1번을 사용한다. 브라우저를 추가로 내려받지 않으므로 가장 빠르다. 다만 puppeteer 가 검증한 조합이 아니므로 브라우저의 버전 차이가 크면 렌더링이 실패할 수 있다. 이 경우 2번으로 전환한다.

3번을 수행했는데도 같은 메시지가 나타난다면 이전 버전이 `PATH`에 남아 있는지 확인한다. 오류 메시지에 사용 중인 mermaid-cli 경로가 함께 표시된다.

```
which -a mmdc
```

`--chrome`을 값 없이 지정하면 `Google Chrome` → `Chromium` → `Microsoft Edge` 순으로 탐색하고, 리눅스에서는 PATH에서 `google-chrome`·`chromium`·`chromium-browser`를 탐색한다. 경로를 직접 지정할 수도 있다.

```
kdsm erd -f png --chrome '/Applications/Google Chrome.app/Contents/MacOS/Google Chrome'
```

mermaid-cli는 브라우저 경로를 인자로 받지 않고 puppeteer 설정 파일로만 받는다. 따라서 `--chrome`을 지정하면 해당 경로만 담은 설정 파일을 임시로 생성하여 전달한다. 설정을 직접 작성하려면 `--puppeteer-config`를 지정하며, 이 옵션이 우선한다.

ℹ mmdc를 탐색하는 순서

`--mmdc` → `KDSM_MMDC_BIN` → `PATH` 순이다. `rebuild.sh`가 kdsm을 탐색하는 방식과 같으며, 어떤 도구로 렌더링했는지 분명해야 하기 때문이다.

`npx`로 실행 시점에 내려받는 방식은 사용하지 않는다. 명령 한 줄이 수백 메가바이트를 내려받는 것은 예측하기 어려운 동작이며, 네트워크가 차단된 환경에서는 원인을 파악하기 어려운 실패로 이어진다. 도구를 찾지 못하면 다음과 같이 안내한다.

```
mermaid-cli(mmdc)를 찾을 수 없습니다. 이미지 파일은 이 도구로 생성합니다.
  npm install -g @mermaid-js/mermaid-cli
다른 경로에 설치했다면 --mmdc 또는 KDSM_MMDC_BIN 으로 지정하세요.
```

> ⚠ 이미지의 한글이 사각형으로 표시되면 글꼴 문제이다
>
> mermaid-cli는 헤드리스 크로미움으로 그리며, 크로미움은 시스템 글꼴을 사용한다. macOS에서는 문제가 없지만 CJK 글꼴이 없는 컨테이너에서 CI로 실행하면 한글이 빈 사각형으로 출력된다. 그러한 환경에서는 `fonts-noto-cjk`와 같은 글꼴 패키지를 함께 설치해야 한다.
>
> `.mmd`와 `.md`는 텍스트이므로 이 문제가 발생하지 않는다. 이미지는 최종 단계에서 한 번 생성하고, 저장소에서 관리하며 확인할 대상은 소스 파일로 두는 것이 적절하다.

한 장만 직접 렌더링할 수도 있다.

```
mmdc -i "document/out/erd/논리-전체.mmd" -o 논리-전체.svg
```

> ⚠ `PK`라는 이름의 컬럼은 그대로 표기할 수 없다
>
> Mermaid는 엔터티 내부를 `type name key` 순서로 해석하며, `key` 위치에 `PK`·`FK`·`UK`를 받는다. 따라서 컬럼 이름이 이 중 하나이면 이름이 아니라 키 표시로 해석하고 중단한다.
>
> ```
> Parse error: Expecting 'ATTRIBUTE_WORD', got 'ATTRIBUTE_KEY'
> ```
>
> 기본키 컬럼 목록을 담는 컬럼에 `PK` 라는 물리명을 부여하면 여기에 해당한다. 정상적인 컬럼이지만 이름이 Mermaid 의 예약어와 일치한다. Mermaid 문법에는 이 낱말을 따옴표로 감싸는 방법이 없으므로 다른 표기를 사용한다.
>
> ```
> VARCHAR(200) PK_ "PK"
> ```
>
> 밑줄을 붙여 표기하고 원래 이름은 주석 위치에 기재한다. 안내 없이 변경하면 ERD를 보는 사람이 컬럼명을 잘못 인식하게 되므로 리포트에도 기록한다.
>
> ```
> Mermaid 문법과 겹쳐 피해서 적은 이름:
>   MT_ENTTY.PK: ... PK_ 로 적고 진짜 이름은 주석에 두었습니다.
>   MT_TBL.PK: ...
> ```
>
> 이것은 데이터의 오류가 아니라 렌더러의 제약이므로 종료 코드를 1로 변경하지 않는다. 논리 ERD는 속성명이 한글이므로 예약어와 겹치지 않는다.
>
> 표기가 바뀌는 것을 피하려면 `PK`·`FK`·`UK` 를 컬럼 물리명으로 사용하지 않는다. 두 글자 약어는 뜻이 좁아 표준용어로도 적합하지 않으므로, 물리명을 정하는 단계에서 걸러 두는 편이 낫다.

> ⚠ 엔터티 내부의 한글은 Mermaid 버전에 영향을 받는다
>
> Mermaid 문서는 엔터티명·관계·속성 모두 유니코드를 지원한다고 기술하고 있다. 다만 속성의 `type`·`name` 위치에 대한 유니코드 지원은 비교적 최근에 추가되었으므로, 오래된 버전에서는 엔터티 내부가 정상적으로 표시되지 않을 수 있다.
>
> 이 경우 `--no-attributes`로 관계선만 작성하거나 `-m physical`을 사용한다. 물리 ERD는 이름이 모두 영문이므로 버전의 영향을 받지 않는다.

CHAPTER 17

# 산출물 내보내기

적재된 메타테이블을 읽어 배포용 산출물을 생성한다. 그 방법이 리버스 엔지니어링이다. 정방향이 정의서 → 메타파일 → 데이터베이스의 순서라면 이 기능은 그 역방향으로 거슬러 올라가 문서를 만든다.

DM-DST를 사람이 작성하는 일을 대신하는 것이 아니다(9.1절). 산출물 없이 구축한 데이터베이스에서 산출물을 얻을 때, 그리고 이미 있는 산출물의 판을 데이터베이스에 맞출 때 쓴다.

생성하는 산출물은 DM-DST 세 종이다. `standard`가 데이터 표준 정의(DM-DST-001), `logical`이 논리 데이터 모델(DM-DST-002), `physical`이 물리 데이터 모델(DM-DST-003)이다. `--target`으로 지정한다.

```
kdsm export                                  # DB에서 읽어 document/out 에 세 종 모두
kdsm export --target standard                # 데이터 표준 정의 산출물만
kdsm export --target logical                 # 논리 데이터 모델 산출물만
kdsm export --target physical                # 물리 데이터 모델 산출물만
kdsm export --target all                     # 세 종 모두. 생략한 것과 같다
kdsm export --meta 논리메타.xlsx --meta 물리메타.xlsx   # DB 없이 메타파일에서
kdsm export -o /tmp/out --version 1.0.0      # 다른 곳에, 판을 올려서
```

메타테이블을 읽을 곳은 `--dbms`로 지정한다. PostgreSQL·MySQL·Databricks 세 곳을 모두 읽으며, 생략하면 PostgreSQL이다. 적재는 세 곳에 모두 수행되므로(15장) 읽는 쪽도 같은 범위를 지원한다.

```
kdsm export --dbms databricks --catalog dev_landing --schema kdsm_meta
kdsm export --dbms mysql --schema kdsm_meta
```

Databricks는 이름이 세 단이므로 `--catalog`가 필요하다. 생략하면 `DATABRICKS_CATALOG`를 사용하며, 그마저 없으면 두 단으로 조회하여 접속 설정의 기본 카탈로그를 대상으로 한다. 접속 정보의 환경변수는 `kdsm load`와 동일하다(20.9절).

`--target`을 지정하지 않으면 3종을 모두 생성한다. 영역마다 문서가 1종이고 3종이 하나의 산출물 묶음을 구성하므로, 지정하지 않은 경우에는 묶음 전체를 생성하려는 것으로 판단한다.

DM-LIB 산출물은 생성하지 않는다. 그 6종은 KDSM 메타테이블의 초기 데이터를 정의한 시스템 관리자 영역의 문서이며, 사용자에게 배포하는 대상이 아니기 때문이다(9.1절).

처음 사용할 때에는 기존 산출물과 대조해 볼 것을 권한다. 메타테이블과 산출물이 일치하는 상태에서 실행하면 같은 내용이 생성되므로, 결과가 다르다면 둘 중 한쪽이 이미 어긋나 있는 것이다.

실질적인 효용은 그다음부터 나타난다. 운영 중에 테이블이 변경되면 문서와 데이터베이스의 내용이 달라지는데, 수작업으로 맞추는 대신 다시 생성한다.

> ! 기존 경로에 쓰지 않는다
>
> 기본 출력 경로는 `document/out` 이다. 내보낸 문서로 원본을 덮어쓰면 내보내기가 잘못되었을 때 복구할 수 없다. 먼저 두 벌을 비교하고, 교체할 의도가 분명해진 뒤에 옮긴다.

## 17.1 메타테이블에 없는 두 가지

이 작업이 까다로운 이유는 문서에는 있으나 테이블에는 없는 정보가 있기 때문이다. 두 가지를 각각 다른 방식으로 처리한다.

산출물마다 문서가 한 권이므로 영역 설정은 받지 않는다. 어느 내용을 어느 권에 수록할지는 문서번호가 이미 정하고 있다.

### 17.1.1 하나. 버전과 작성자

내보낸 문서는 버전 `0.0.0`, 작성자 공란으로 생성된다. 사람이 작성한 버전의 번호와 이름을 승계하지 않는다.

승계하면 사람이 검토하여 올린 `0.9.5`와 자동으로 생성된 `0.9.5`가 같은 이름으로 혼용된다. 작성자도 마찬가지이다. 이름을 그대로 옮기면 그 사람이 이 버전을 작성하였다는 뜻이 되는데, 실제로는 자동으로 생성된 것이며 검토한 사람은 아직 없다. `0.0.0`과 공란은 아직 사람이 검토하지 않았음을 나타내는 표시이다.

검토를 마치고 버전을 올릴 때 `--version`으로 버전을, `--author`로 그 버전을 책임질 사람을 지정한다.

```
kdsm export --version 1.0.0 --author 유충현
```

`--author`는 표지와 양식 시트의 작성자 항목에 함께 기재된다.

> 작성자를 기재하는 것의 의미
>
> `--author`를 지정하는 순간 그 문서는 "자동 생성된 것"에서 "검토를 거쳐 배포하는 것"으로 성격이 바뀐다. 생성 결과를 그대로 확인하는 단계가 아니라, 내용을 검토하고 버전을 올릴 때 함께 지정하는 것이 적절하다.
>
> `--author ''`와 같이 빈 문자열을 지정하면 공란과 동일하게 처리한다. 기본값으로 복귀하지 않는다.

> **i** 파일 안에는 `V`를 붙이지 않는다
>
> 표지와 양식 시트 모두 `0.0.0`으로 기재한다. `V`는 파일명에만 붙이다.
>
> 파일명에서는 `_V0.0.0_20260811_`과 같이 다른 구성 요소와 구별하는 표시로서 의미가 있지만, 셀 안에서는 값 앞에 붙은 문자일 뿐이다. 표기가 일정하지 않으면 `0.0.0`과 `V0.0.0`을 비교할 때 불일치가 발생한다.

#### 17.1.1.1 파일명도 함께 변경된다

파일명 규칙은 9.2절 을 따른다. 버전과 날짜를 교체하고 이름에서 `템플릿`을 제외한다. 문서번호는 그대로 유지한다.

```
DM-DST-001_데이터표준정의산출물템플릿_V0.9.3_20260810.xlsx
                                  ↓       ↓      ↓
DM-DST-001_데이터표준정의산출물_______V0.0.0_20260812.xlsx
```

템플릿은 작성해야 할 빈 양식이고 내보낸 문서는 내용이 채워진 산출물이다. 이름에 `템플릿`이 남아 있으면 무엇을 배포해야 하는지 혼동하게 된다.

날짜는 생성한 날이며, 표지와 양식 시트의 작성일과 같은 값에서 결정된다. `--written-at`을 지정하면 세 위치가 함께 변경된다. 세 위치에서 날짜를 각각 생성하면 언젠가 불일치가 발생하는데, 그때는 어느 값이 정확한지 판단할 수 없다.

macOS의 한글 파일명은 NFC로 쓴다. 이유는 9.2절 에 적었다.

이 형식이 아닌 파일명은 수정하지 않고 그대로 사용한다. 규칙을 알 수 없는 이름을 임의로 변경하는 것보다, 그대로 두어 사람이 확인하도록 하는 것이 적절하다.

### 17.1.2 둘. 서식과 작성 지침

산출물은 표지·헤더·설명 행·데이터·작성 지침이 한 시트에 함께 있는 서식 문서이다. 빈 파일에서 생성하면 이 네 가지를 모두 잃게 된다.

그러므로 패키지에 들어 있는 템플릿을 복사하고 데이터 영역만 교체한다. 행 수가 달라지면 아래의 지침 블록을 이동하고, 새로 생성된 행에는 첫 데이터 행의 서식을 적용한다.

> **i** 서식은 내용에 맞추어 조정된다
>
> 데이터 행의 높이는 자동 줄바꿈으로 접히는 줄 수에 맞추고, 열 너비는 가장 긴 값에 맞추어 넓힌다. 좁히지는 않는다. 템플릿이 정한 너비는 가독성을 고려한 값이므로 데이터가 짧다는 이유로 축소하면 서식이 훼손된다.
>
> 너비는 60에서 제한한다. 설명 항목과 같이 한 문장을 넘는 열을 내용에 모두 맞추면 그 열 하나 때문에 표 전체를 가로로 확장해야 한다. 초과하는 내용은 자동 줄바꿈으로 처리한다.
>
> 줄 수는 글자 폭으로 추정하므로 실제로는 한 줄이 더 늘어날 수 있다. 생성한 파일을 열었을 때 일부 행이 좁아 보이면 해당 행의 높이만 조정한다.
>
> 데이터가 끝난 위치의 빈 행은 한 행만 남기고 제거한다. 남긴 한 행은 표와 작성 지침 사이의 간격이므로 제거하면 지침이 표에 붙는다.

> **i** `<br>`는 셀 안의 줄바꿈으로 바꾼다
>
> 메타파일의 설명 항목에는 `<br>`로 줄을 구분한 곳이 있다. 마크다운으로 변환하여 읽기 위하여 사용한 표기인데, 셀 안에서는 문자 그대로 표시된다. 산출물에 기록할 때 줄바꿈 문자로 변환한다.
>
> 데이터 항목에는 자동 줄바꿈을 설정한다. 설정하지 않으면 긴 내용이 오른쪽 빈 셀까지 표시되어 옆 항목의 값처럼 보이고, 셀 안의 줄바꿈도 나타나지 않는다. 머리글과 항목 설명 행은 템플릿이 정한 대로 유지한다.

## 17.2 표기를 옮기는 두 가지

테이블과 문서가 동일한 사실을 서로 다르게 기재하는 경우가 있다.

### 17.2.1 코드값 → 코드명

메타테이블은 코드값(`01`)을 기재하고, 정의서는 사람이 읽는 이름(`기본형`)을 기재한다. `MT_STDCD_DTL`을 조회하여 복원한다.

표 17.1. 코드값을 옮기는 항목

| 정의서 항목 | 표준코드 | 예 |
|---|---|---|
| 엔터티 구분 | `ENTTY_CLSF` | `01` → `기본` |
| 속성 유형 | `ATRB_TYPE` | `01` → `기본형` |
| 발생주기 | `DT_OCRN_CYCLE` | `01` → `실시간` |
| 상태 | `STTS` | `02` → `확정` |
| 단어 구분 | `WORD_SE` | `01` → `일반어` |
| 용어 구분 | `TRM_SE` | `03` → `엔터티명/속성명` |

`ENTTY_CLSF`만 코드명을 그대로 사용하지 않는다. 코드명은 `기본엔터티`인데 엔터티 정의서에는 `기본`이라고만 기재한다. 바로 옆 항목이 이미 엔터티를 가리키고 있어 반복하지 않는 것이다. 이러한 예외는 `CODE_LABELS`에 정의해 두었다.

표준코드가 아닌 대응도 세 가지 있다. 데이터 표준 산출물에만 나타난다.

표 17.2. 이름을 다른 테이블에서 조회하는 항목

| 정의서 항목 | 읽는 대상 | 예 |
|---|---|---|
| 도메인 분류명 · 도메인 분류 | `MT_STD_DMN_CLSF` | `0101` → `가격` |
| 도메인 그룹 · 도메인 그룹 이름 | `MT_STD_DMN_GROUP` | `01` → `금액` |
| 표준 코드명 | `MT_STDCD_MSTR` | `STTS` → `상태` |

도메인 분류 시트의 `도메인 그룹 이름`도 그룹명이 아니라 그룹 코드에서 도출한다. 메타파일은 분류 시트에 그룹명을 함께 기재하지만 `MT_STD_DMN_CLSF` 테이블에는 해당 컬럼이 없다. 코드는 양쪽에 모두 있으므로 코드에서 도출하면 메타파일에서 읽든 데이터베이스에서 읽든 동일한 결과가 산출된다.

표에서 찾지 못한 코드는 그대로 둔다. 삭제하거나 비우면 내보낸 문서만 확인하는 사람에게는 값이 처음부터 없었던 것처럼 보인다. 코드가 그대로 수록되어 있으면 적어도 어느 값이 변환되지 않았는지 확인할 수 있다.

### 17.2.2 복합키 구분자

동일한 복합키를 논리 정의서는 `엔터티이름 + 속성이름`으로, 물리 정의서는 `DB_NM, SCHM_NM`으로 기재한다. 규칙은 두 표기를 모두 허용하지만 관례가 층위마다 다르다.

내보낼 때 그 관례에 맞추어 표기한다. 그렇게 하지 않으면 사람이 작성한 문서와 내보낸 문서가 의미는 같으나 표기가 달라, 변경 내용을 확인할 때 이 차이가 함께 나타난다.

## 17.3 데이터 표준 정의 산출물

`--target standard`는 데이터 표준 일곱 메타테이블을 읽어 DM-DST-001의 일곱 시트를 채운다.

**표 17.3.** DM-DST-001의 시트와 원천

| 시트 | 읽는 메타테이블 |
|---|---|
| 표준단어 정의서 | `MT_STD_WORD` |
| 표준용어 정의서 | `MT_STD_TRM` |
| 표준도메인 정의서 | `MT_STD_DMN` |
| 표준도메인 정의서_도메인 그룹 | `MT_STD_DMN_GROUP` |
| 표준도메인 정의서_도메인 분류 | `MT_STD_DMN_CLSF` |
| 표준코드 정의서_코드 마스터 | `MT_STDCD_MSTR` |
| 표준코드 정의서_코드값 상세 | `MT_STDCD_DTL` |

영역 설정을 받지 않는다. 데이터 표준은 문서가 한 권이므로 분할할 필요가 없기 때문이다.

상태로 선별하지 않는다. 폐기한 항목도 그대로 수록한다. 산출물이 메타데이터의 사본이라는 관계를 유지하기 위해서이다. 확정된 항목만 모은 표가 필요하다면 그것은 내보내기가 아니라 별도로 추출할 사항이다.

> **i 도메인 그룹 코드는 수록되지 않는다**
>
> 도메인 그룹 시트에는 코드를 기재하는 항목이 없다. 사람이 읽는 문서에서 그룹을 지칭하는 것은 이름이기 때문이다. 코드는 메타테이블에만 남는다.
>
> 내보낸 문서를 다시 메타테이블로 옮길 때에는 그룹 코드를 별도로 확보해야 한다는 뜻이기도 한다.

## 17.4 논리 데이터 모델 산출물

`--target logical`은 `MT_ENTTY`·`MT_ATRB`를 읽어 DM-DST-002의 두 시트를 채운다.

표 17.4. DM-DST-002의 시트와 원천

| 시트 | 읽는 메타테이블 |
|---|---|
| 엔터티 정의서 | `MT_ENTTY` |
| 속성 정의서 | `MT_ATRB` |

영역 설정을 받지 않는다. DM-LIB 계열은 영역마다 한 권씩 분할되지만 이 산출물은 논리 모델 전체를 한 권에 수록하기 때문이다. 그러므로 영역 설정에 없는 엔터티도 수록된다.

코드값과 복합키 표기를 옮기는 방식은 DM-LIB 논리 정의서와 같다. 항목 이름이 하나 다르다. 동일한 값을 DM-LIB는 `참조 속성이름`, DM-DST-002는 `참조 속성명`으로 기재한다. 그 한 항목만 교체한다.

## 17.5 물리 데이터 모델 산출물

`--target physical`은 `MT_DB`·`MT_TBL`·`MT_COL`을 읽어 DM-DST-003의 세 시트를 채운다.

표 17.5. DM-DST-003의 시트와 원천

| 시트 | 읽는 메타테이블 |
|---|---|
| 데이터베이스 정의서 | `MT_DB` |
| 테이블 정의서 | `MT_TBL` |
| 컬럼 정의서 | `MT_COL` |

항목 이름은 DM-LIB 물리 정의서와 같다. 영역 설정을 받지 않는다는 점만 다르다. 데이터베이스도 선별하지 않고 모두 수록한다. DM-LIB는 해당 영역의 테이블이 실제로 사용하는 데이터베이스만 선별하여 수록하지만, 이 산출물은 물리 모델 전체를 한 권에 수록하기 때문이다.

## 17.6 카탈로그에서 내보내기

`--from-catalog`를 지정하면 메타테이블이 아니라 DBMS 카탈로그를 읽는다. 표준화를 거치지 않은 데이터베이스의 현재 구조를 DM-DST-003 서식에 담아, 표준화의 출발점을 만드는 용도이다.

```
uv run kdsm export --from-catalog --dbms databricks --catalog dev_landing
```

내보내는 대상은 물리 데이터 모델뿐이다. 카탈로그에는 엔터티와 속성이라는 층이 없고 표준단어와 표준용어도 없다. 없는 것을 추정하여 채우면 사람이 정한 값과 프로그램이 생성한 값이 한 문서에 섞이므로, 확인할 수 있는 것만 수록한다.

카탈로그 전체를 한 권에 수록한다. 스키마는 데이터베이스 정의서의 행으로 구분된다. `--schema`로 대상을 하나로 좁힐 수 있다. 대상을 지정하는 표기는 진단과 동일하다(18.5.4.3절).

### 17.6.1 한글명을 채우는 순서

표 17.6. 한글명의 출처

| 순서 | 출처 | 비고 |
|---|---|---|
| 1 | 카탈로그의 주석 | `한글명 - 설명` 형식이면 앞부분만 사용한다 |
| 2 | 표준용어 사전 | 물리명이 일치하는 표준용어의 한글명 |
| 3 | 공란 | 양쪽 모두에서 찾지 못한 경우 |

주석을 앞세우는 것은 내보낸 산출물이 현재의 모습을 담아야 하기 때문이다. 주석은 그 데이터베이스가 실제로 사용하는 이름이고, 사전은 그렇게 명명하였어야 할 이름이다. 둘이 어긋나는 부분은 `kdsm diagnose`가 별도로 확인한다(18장).

물리명에서 한글명을 생성하지 않는다. 생성하면 사람이 명명한 것과 프로그램이 생성한 것을 구별할 수 없게 되고, 검토해야 할 항목이 검토를 마친 항목처럼 보인다. 공란은 채워야 할 자리가 남아 있음을 그대로 드러낸다.

사전은 `--meta`로 메타파일에서 읽거나, 생략하면 `--dict-env-prefix`가 가리키는 데이터베이스에서 읽는다. 사전이 아직 없으면 `--no-dict`를 지정하여 주석만 사용한다.

> **i** 수록되지 않는 항목
>
> 카탈로그가 알 수 없는 항목은 공란으로 남는다. 관련 엔터티, 표준도메인, 개인정보 여부, 예상 건수와 보관 기간이 이에 해당한다. 표준코드도 없으므로 코드값을 코드명으로 변환하지 않는다.
>
> 컬럼 순번은 카탈로그의 자리 번호를 그대로 옮기지 않고 테이블마다 1부터 다시 부여한다. 정의서의 순번은 테이블 안에서의 위치를 나타내므로 1부터 빈틈없이 이어져야 하는데, 카탈로그의 자리 번호는 그 조건을 보장하지 않는다. 컬럼을 삭제한 자리가 번호에 남는 경우가 있다. 읽어 온 순서는 그대로 유지하므로 컬럼의 배열 자체는 변경되지 않는다.
>
> `DBMS 종류·버전` 칸에는 종류만 기재한다. 카탈로그가 알려 주는 것은 종류뿐이며, 버전을 추정하여 기재하면 사람이 확인한 값과 구별되지 않는다.

## 17.7 옵션

표 17.7. kdsm export 옵션

| 옵션 | 설명 |
|---|---|
| `--target` | 내보낼 산출물: `standard`·`logical`·`physical`·`all`. 쉼표로 여러 개를 지정할 수 있다 (기본 `all`) |
| `-o`, `--output` | 산출물을 저장할 디렉터리 (기본 `<저장소>/document/out`) |
| `--template-dir` | 템플릿 디렉터리. 생략하면 패키지에 든 템플릿을 사용한다 |
| `--meta` | DB 대신 메타파일에서 읽는다. 여러 번 지정할 수 있다 |
| `--env-prefix` | 접속 정보의 환경변수 접두어 |
| `--schema` | 대상 스키마명 |
| `--from-catalog` | 메타테이블이 아니라 DBMS 카탈로그에서 읽는다. 물리 모델만 내보낸다 |

| 옵션 | 설명 |
|---|---|
| `--dbms` | 읽을 DBMS: `postgresql`(기본)·`mysql`·`databricks` |
| `--catalog` | `--dbms databricks`일 때 카탈로그. 생략하면 `DATABRICKS_CATALOG` |
| `--host`, `--port`, `--database`, `--user` | `--from-catalog` 일 때의 접속 정보. 생략하면 `<접두어>` 환경변수 |
| `--no-dict` | `--from-catalog` 일 때 한글명을 주석에서만 가져온다. 사전을 조회하지 않는다 |
| `--dict-env-prefix` | `--from-catalog` 일 때 표준 사전이 있는 데이터베이스의 환경변수 접두어 |
| `--dict-schema` | `--from-catalog` 일 때 사전이 있는 스키마 |
| `--version` | 파일명과 표지의 버전. 기본 `V0.0.0` |
| `--author` | 표지와 양식 시트의 작성자. 기본 공란 |
| `--written-at` | 작성일. 파일명의 날짜도 따른다. 기본 오늘 |
| `-q`, `--quiet` | 한 줄 요약만 출력한다 |

템플릿은 문서번호로 찾는다. 파일명에 버전과 날짜가 붙어 있어 정확한 이름을 인자로 받기 어렵기 때문이다.

> **i 기본 경로는 저장소 루트 기준이다**
>
> 경로 기본값은 파서에 고정하지 않고 실행 시점에 결정한다. `document`와 `dbms`가 함께 있는 디렉터리를 상위로 이동하며 탐색하여 그 위치를 루트로 사용한다.
>
> 하위 디렉터리에서 실행하여도 같은 위치를 참조한다. 상대 경로를 그대로 사용하면 실행 위치에 따라 참조 대상이 달라진다.
>
> ```
> cd library && uv run kdsm export     # 어디서 불러도 같은 곳을 본다
> ```
>
> 저장소 밖에서 실행하면 현재 디렉터리를 루트로 간주한다. 이 경우에는 `--template-dir`로 위치를 지정한다.

메타테이블에서 읽으므로 접속 정보가 필요하다. 필요한 환경변수는 `--dbms`가 결정한다. PostgreSQL은 `KDSM_DB_HOST`부터 `KDSM_DB_PASSWORD`까지 여섯 변수, MySQL은 `MYSQL_*`, Databricks는 `DATABRICKS_*`이다(20.9절). `--meta`로 메타파일을 지정하면 접속하지 않으므로 환경변수도 필요하지 않다. `--env-prefix`는 PostgreSQL의 접두어를 바꾸며, 비밀번호와 토큰은 인자로 받지 않는다(20.10절).

## 17.8 내보낸 결과를 확인하는 방법

내보낸 문서를 기존 문서와 대조하는 것이 가장 확실하다. 현재 시점에서는 두 문서가 같아야 하며, 다르다면 그 차이가 곧 데이터베이스에서 변경된 내용이다.

```
kdsm export -o /tmp/out
kdsm validate /tmp/out/DM-DST-00{1,2,3}_*.xlsx \
  --meta dbms/meta/meta_data/kdsm_modeling_meta_*.xlsx
```

내보낸 산출물에도 검증 규칙을 그대로 적용한다. 데이터베이스에서 도출한 것이므로 통과할 것으로 예상되지만, 통과 여부를 확인하는 것과 통과할 것으로 간주하는 것은 다르다.

DM-DST-002와 DM-DST-003을 모두 위치 인수로 지정한다. 항목 이름이 DM-LIB 정의서와 같으므로 동일한 리더가 읽는다.

CHAPTER 18

# 표준화 진단

이미 구축된 데이터베이스가 표준을 어느 정도 준수하는지 측정한다. 테이블 카탈로그를 읽어 표준단어·표준용어·표준도메인 사전과 대조한다. PostgreSQL·MySQL·Databricks Unity Catalog를 진단할 수 있다(18.5.2절). 사전이 없는 스키마도 진단한다. 이때는 스키마 내부에서 서로 어긋난 부분을 확인한다(18.3절).

```
kdsm diagnose --schema sales --schema hr
```

! 검증과 목적이 다르다

`kdsm validate`는 사전과 산출물이 서로 어긋나지 않는지 확인한다. 결과는 통과 또는 실패이다. 진단은 이미 구축된 데이터베이스가 표준을 어느 정도 준수하는지 측정한다. 결과는 비율이다.

전자는 "이대로 구축해도 되는가"를 확인하고, 후자는 "이미 구축한 것을 어디부터 개선할 것인가"를 확인한다. 그러므로 진단은 기본적으로 정상 종료한다. 현황을 측정하는 작업이지 합격 여부를 판정하는 작업이 아니기 때문이다.

## 18.1 무엇을 측정하는가

표준 준수 수준을 세 가지 진단 대상으로 구분하여 측정하고, 개선 순서를 정하는 근거를 함께 제시한다.

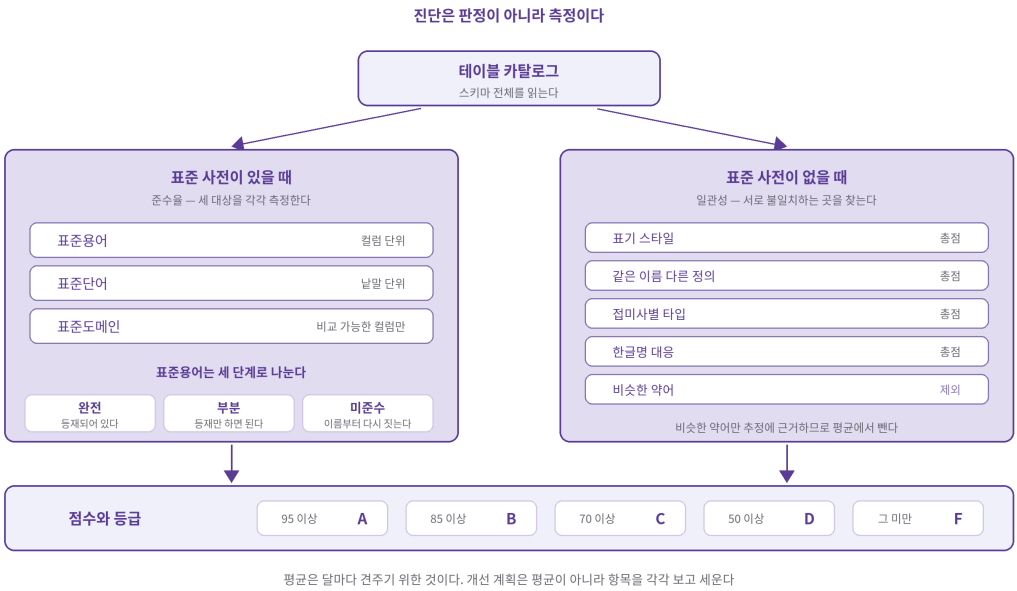


그림 18.1. 진단의 구조

그림 18.1 와 같이 사전이 있으면 준수율을, 없으면 일관성을 산출한다. 어느 쪽이든 카탈로그를 읽는 것에서 시작하여 점수와 등급으로 마무리된다.

## 18.1.1 진단 대상

진단 대상마다 측정 항목과 집계 단위가 다르다.

표 18.1. 세 가지 진단 대상

| 진단 대상 | 측정 항목 | 집계 단위 |
|---|---|---|
| 표준용어 | 컬럼 물리명이 사전에 등재된 용어인가 | 컬럼 |
| 표준단어 | 물리명을 밑줄로 분리한 낱말이 등재된 영문약어인가 | 낱말 |
| 표준도메인 | 용어가 정의한 타입·길이가 실제 컬럼과 일치하는가 | 용어가 등재된 컬럼 |

### 18.1.1.1 부분 준수를 따로 세는 이유

표준용어는 세 단계로 구분한다.

```
완전   CUST_NM       표준용어로 등재되어 있다
부분   CUST_ORDR_NM  등재는 없지만 고객·주문·이름 약어 조합이다
미준수 CUST_FOO_NM   FOO 가 표준단어가 아니다
```

> ! 개선 난이도가 전혀 다르다
>
> 부분 준수는 **사전에 등재만 하면 되는** 상태이다. 이름은 이미 표준에 따라 부여되어 있다. 미준수는 이름부터 다시 부여해야 하고, 그러면 이미 생성한 테이블의 컬럼명과 애플리케이션 코드까지 함께 변경된다.
>
> 두 가지를 하나로 묶어 집계하면 "미준수 40%"라는 수치만 남아 어디부터 조치해야 할지 판단할 수 없다.

부분 준수까지 인정한 비율도 함께 제시한다. 현재 상태와 등재를 마쳤을 때의 상태를 나란히 비교하기 위해서이다.

#### 18.1.1.2 도메인은 비교할 수 있을 때만 집계한다

용어가 등재되지 않은 컬럼은 준수해야 할 도메인도 없다. 이때 도메인 준수를 0으로 집계하면 표준용어 미준수가 두 번 반영되는 결과가 된다. 그러므로 분모에서 제외한다.

```
   표준도메인 100.0%   비교 가능한 컬럼 154개
```

### 18.1.2 함께 제시하는 항목

세 가지 진단 대상 외에 두 가지를 함께 집계한다. 개선 순서를 정하는 근거가 되기 때문이다.

#### 18.1.2.1 미등록 약어

표준단어가 아닌 약어를 **사용 빈도가 높은 순서로** 집계한다.

```
### 표준단어가 아닌 약어 12종
  사용 빈도가 높은 것부터 등재하면 준수율이 가장 빨리 상승합니다.
  ORD            87회
  YMD            41회
```

87개 위치에서 사용된 약어 하나를 등재하면 87개 컬럼이 한 번에 부분 준수가 된다. 한 곳에서만 사용된 약어를 먼저 수정하는 것보다 훨씬 효율적이다.

#### 18.1.2.2 같은 이름 다른 정의

같은 컬럼명이 테이블마다 다른 타입으로 정의된 경우이다.

```
   CUST_NM              VARCHAR(100)(3) · VARCHAR(40)(12)
```

> ⚠ 이것이 가장 강한 신호다
>
> 사전에 없는 이름이라도 한 곳에서만 사용했다면 수정 대상은 하나이다. 그러나 같은 이름을 서로 다른 타입으로 열다섯 곳에 정의했다면, 그 이름의 의미에 대하여 아무도 합의하지 않았다는 뜻이다. 표준화의 목적이 바로 이러한 상황을 방지하는 것이다.

### 18.1.3 점수와 등급

세 가지 진단 대상의 평균을 종합 점수로 산출한다.

표 18.2. 점수 구간과 등급

| 점수 | 등급 |
|---|---|
| 95 이상 | A |
| 85 이상 | B |
| 70 이상 | C |
| 50 이상 | D |
| 그 미만 | F |

> **i 평균이 늘 옳지는 않다**
>
> 진단 대상마다 집계 단위가 달라 평균에 큰 의미는 없다. 다만 지난달과 이번 달을 비교하려면 하나의 수치가 필요하다. 개선 계획을 수립할 때에는 평균이 아니라 세 가지를 각각 확인한다.

## 18.2 리포트

동일한 결과를 세 가지 형태로 제공한다. 계산은 진단기가 이미 수행하였으며 출력 형식만 다르다.

### 18.2.1 세 형태

```
kdsm diagnose --schema sales                    # 화면
kdsm diagnose --schema sales -o 진단.xlsx        # 엑셀까지
kdsm diagnose --schema sales --qmd 진단.qmd      # Quarto 까지
kdsm diagnose --schema sales --qmd 진단.qmd --render        # HTML·PDF 까지
kdsm diagnose --schema sales --qmd 진단.qmd --render pdf    # PDF 만
kdsm diagnose --schema sales -q                 # 한 줄 요약
```

엑셀 리포트는 내용을 시트별로 구분하여 수록한다. 개선 작업을 담당자별로 분담할 때 사용한다.

표 18.3. 엑셀 리포트의 시트

| 시트 | 수록 내용 |
|---|---|
| 요약 | 표준용어·표준단어·표준도메인 수치를 한 장에 정리 |
| 진단 대상 · 진단 대상 컬럼 | 이번에 읽은 테이블과 컬럼 |
| 컬럼별 진단 · 미등록 약어 · 같은 이름 다른 정의 · 미사용 표준용어 | 표준 사전 대조 |
| 일관성 점수 · 일관성 지적 · 표기 스타일 | 사전 없이 확인한 일관성 |
| 컬럼명 통계 · 주석 통계 · 주석 누락 | 이름 통계 |
| 진단 제언 | 다음 단계의 조치 사항 |

시트 구성은 항상 동일하다. 사전의 유무와 관계없이 열네 장이 생성된다. 사전이 없을 때에는 준수율 네 시트에 `측정 불가 (표준 사전 없음)`이라고 한 줄 기재한다.

따라서 시트가 비어 있지 않은데 `측정 불가`만 있다면 진단이 완료되지 않은 것이 아니라 사전을 지정하지 않고 실행한 것이다. `--dict-schema`를 지정하여 다시 실행한다.

### 18.2.2 진단 대상

요약 다음에 위치하는 시트이다. "컬럼 596개"라는 요약만으로는 그 596개가 어느 테이블에서 도출되었는지 확인할 수 없다. 진단 범위가 올바른지 확인하려면 목록이 필요하며, 목록이 없으면 수치를 신뢰하기 어렵다.

테이블별로 컬럼 수, 설명이 있는 컬럼 수, 완전·부분·미준수 컬럼 수, 용어 준수율을 함께 제시한다. 준수율이 낮은 테이블부터 정렬한다. 테이블은 개선 작업을 분담하는 단위이기도 하므로, 이 시트를 그대로 업무 분장표로 활용할 수 있다.

준수와 관련된 네 항목이 모두 `-`로 표시된다면 표준 사전을 읽지 못한 것이다. 준수 여부는 사전과 대조해야 산출되는 값이므로 사전 없이는 집계할 수 없다. 이때에는 화면에 경고가 함께 출력된다. 사전이 다른 데이터베이스에 있으면 `--dict-env-prefix`를, 같은 데이터베이스의 다른 스키마에 있으면 `--dict-schema`를 확인한다(18.5.7절).

### 18.2.3 진단 대상 컬럼

읽어 들인 컬럼을 빠짐없이 나열한다. 카탈로그에서 확인한 사실만 수록하고 판정은 포함하지 않는다.

표 18.4. 진단 대상 컬럼의 항목

| 항목 | 기재 내용 |
|---|---|
| 스키마 · 테이블 · 순번 · 컬럼 | 카탈로그에서 확인한 위치 |
| 데이터 타입 · 길이 · 소수 자릿수 · NULL 허용 | 실제 정의 |
| 낱말 | 컬럼명을 밑줄과 대소문자 경계로 분리한 결과 |
| 설명 | 컬럼 주석 |

> **i 사실과 판정을 구분한다**
>
> `컬럼별 진단` 시트에도 컬럼이 한 행씩 수록된다. 그 시트는 사실을 사전과 대조한 결과이고, 이 시트는 대조하기 전의 원본이다.
>
> 두 가지를 한 시트에 수록하면 판정 근거를 확인할 때 원본을 확인할 수 없다. "이 컬럼이 왜 미준수인가"라는 질문에 답하려면 무엇을 읽었는지부터 제시할 수 있어야 한다.

낱말 항목은 이름 통계와 접미사 진단이 해당 컬럼명을 어떻게 분리하였는지 제시한다. `orderAmt`가 `ORDER · AMT`로 분리된 것을 육안으로 확인할 수 있어야 통계 수치를 검토할 수 있다.

테이블 내부에서는 컬럼 순서를 유지한다. DDL과 나란히 놓고 대조하는 경우가 많기 때문이다. 설명이 없는 컬럼은 붉은색으로 표시한다.

### 18.2.4 시트의 공통 구조

모든 시트가 동일한 배치를 사용한다. 1행은 제목, 2행은 한 줄 설명, 4행은 머리글이며 5행부터 자료이다. 배치가 일정하면 어느 시트를 열어도 확인할 위치를 즉시 파악할 수 있다. 머리글은 고정되어 있고 자동 필터가 설정되어 있어 별도의 조작 없이 자료를 선별할 수 있다.

조치가 필요한 행은 연한 붉은색으로 표시한다. 문제가 있는 컬럼, 설명이 누락된 테이블, 이름을 다시 부여해야 하는 제언이 이에 해당한다.

열 너비는 해당 열에서 가장 긴 값에 맞춘다. 한글은 두 자리로 계산한다. 엑셀의 너비 단위는 기본 글꼴의 `0` 한 글자 너비인데 한글은 그 두 배가량이므로, 글자 수로만 계산하면 한글 항목이 항상 잘린다.

지나치게 긴 값은 너비 70에서 제한하고 자동 줄바꿈으로 처리한다. 긴 설명 하나 때문에 표가 가로로 계속 늘어나면 인쇄와 화면 확인이 모두 어렵다. 제목과 설명 두 행은 너비 계산에서 제외한다. 표 전체를 가로지르는 문장이므로 열 하나의 너비로 반영하면 첫 열만 과도하게 넓어진다.

### 18.2.5 진단 제언

마지막 시트이다. 리포트를 받은 사람이 이어서 확인하려는 것은 "무엇부터 착수할 것인가"이며, 그 답을 순서로 제시한다.

표 18.5. 진단 제언의 항목

| 항목 | 기재 내용 |
|---|---|
| 순위 | 해당하는 컬럼이 많은 순서 |
| 구분 | 무엇에 관한 제언인지 |
| 조치 사항 | 무엇을 하면 되는지 한 문장 |
| 판단 근거 | 진단에서 산출된 수치로 뒷받침한 근거 |
| 영향 컬럼 | 해당 조치로 정리되는 컬럼 수 |
| 난이도 | 쉬움 · 보통 · 어려움 |
| 근거 시트 | 수치를 확인할 시트 |

순서는 해당하는 컬럼 수를 기준으로 정한다. 동일한 노력을 투입할 때 더 많이 정리되는 항목을 앞에 배치한다. 난이도를 별도로 기재한 것은 해당하는 컬럼이 많더라도 이름을 다시 부여해야 하는 작업이라면 뒤로 미룰 수 있어야 하기 때문이다.

- 쉬움 — 사전에 등재하거나 주석을 작성하면 되는 작업. 기존 코드에 영향이 없다.
- 보통 — 타입이나 길이를 조정하는 작업. DDL을 수정해야 하지만 이름은 유지된다.
- 어려움 — 이름을 다시 부여하는 작업. 해당 컬럼을 참조하는 모든 코드가 영향을 받는다.

> ! 수정까지 수행하지는 않는다
>
> 컬럼명을 변경하는 SQL은 생성하지 않는다. 이름 변경은 해당 컬럼을 참조하는 모든 코드에 영향을 주므로 사람이 결정할 사항이다. 제언은 가장 적은 노력으로 가장 큰 개선을 얻을 수 있는 순서를 제시하는 데 그친다.
>
> 지적 사항이 하나도 없으면 제언도 제시하지 않는다. 무리하게 생성하면 실제로 필요한 제언이 있을 때 그 내용이 묻히기 때문이다.

### 18.2.6 HTML 과 PDF 로 조판한다

`--qmd`가 만든 문서에는 표제부와 조판 설정이 이미 들어 있고, 같은 디렉터리에 `assets/report.scss`와 `assets/report.typ`가 함께 놓인다. 그래서 해당 디렉터리에서 `quarto render`를 실행하면 즉시 조판된다. `--render`는 그 명령을 대신 실행할 뿐이다.

```
kdsm diagnose --schema sales --qmd 진단.qmd --render          # html,pdf
kdsm diagnose --schema sales --qmd 진단.qmd --render html     # 하나만
```

```
진단.qmd            원본
진단.html           단일 파일. 그림과 스타일을 안에 담는다
진단.pdf            표지 + 목차 + 본문. A4 조판
assets/             조판 자산 (report.scss · report.typ · typst-show.typ)
```

표 18.6. 조판할 수 있는 형식

| 형식 | Quarto 형식 | 특징 |
|---|---|---|
| `html` | `html` | `embed-resources`로 파일 하나에 담는다. 메일에 그대로 첨부할 수 있다 |
| `pdf` | `typst` | A4 조판. LaTeX 배포판이 필요하지 않다 |

PDF 를 Typst 로 만드는 이유는 LaTeX 배포판을 요구하지 않기 때문이다. Quarto 하나만 설치하면 한글 조판까지 끝난다. Quarto 1.4 부터 Typst 가 함께 배포된다.

#### 18.2.6.1 PDF 에는 표지가 붙는다

배포하는 보고서이므로 첫 쪽을 표지로 둔다. 위아래에 보라 띠를 두르고 그 사이에 다음 넷을 배치한다.

표 18.7. 표지에 들어가는 것

| 자리 | 내용 | 출처 |
|---|---|---|
| 제목 | KDSM 데이터 표준화 진단 보고서 | 표제부 `title` |
| 부제 | 대상 스키마: `<스키마명>` | 표제부 `subtitle` |
| 발행 정보 | 생성일자 | 표제부 `date` |
| 하단 | 본 보고서는 디플래닉스의 KDSM을 이용해서 자동 생성되었습니다 | 고정 문구 |

표지는 쪽수에 넣지 않는다. 목차가 1쪽이고 본문이 그 다음이다. 표지 다음에 목차를 한 쪽으로 두고 본문을 새 쪽에서 시작한다.

#### 18.2.6.2 장은 새 쪽에서 시작한다

1수준 제목마다 쪽을 넘긴다. 장이 앞 장의 끝에 이어 붙으면 어디에서 갈리는지 보이지 않는다.

쪽을 넘기되 이미 쪽의 맨 위에 있으면 넘기지 않는다(`pagebreak(weak: true)`). 조건 없이 넘기면 목차 다음의 첫 장 앞에 빈 쪽이 한 장 생긴다. 목차를 낸 뒤 이미 쪽을 넘겨 두었기 때문이다.

#### 18.2.6.3 머리말과 꼬리말

표지를 제외한 모든 쪽에 삽입한다.

표 18.8. 머리말과 꼬리말

| 자리 | 왼쪽 | 오른쪽 |
|---|---|---|
| 머리말 | KDSM 데이터 표준화 진단 보고서 | 그 쪽이 속한 장의 이름 |
| 꼬리말 | KDSM | 쪽 번호 |

장 이름은 쪽 번호로 찾는다. 머리말은 쪽의 맨 위에서 조판되므로, 「지금 위치보다 앞선 제목」으로 찾으면 그 쪽에서 시작하는 장을 아직 지나지 않은 것으로 보아 앞 장의 이름이 나온다.

쪽 번호는 꼬리말이 직접 찍는다. Quarto 의 `page-numbering` 을 함께 지정하지 않는 것은 같은 값을 두 곳에서 정하지 않기 위해서이다.

#### 18.2.6.4 개요를 맨 앞에 둔다

본문의 첫 장은 개요이다. 보고서를 받은 사람이 수치를 읽기 전에 알아야 할 것을 먼저 놓는다.

표 18.9. 개요가 밝히는 것

| 항목 | 밝히는 것 |
|---|---|
| 진단 범위 | 어느 스키마의 테이블 몇 개, 컬럼 몇 개를 읽었는가 |
| 두 방향 | 표준 사전 대조와 스키마 내부 일관성의 차이 |
| 판정과 사실 | 준수율은 판정이고 이름 통계는 집계라는 구분 |
| 구성 | 어느 절에 무엇이 있는가 |

판정과 사실을 가르는 항목이 이 장의 목적이다. 이름 통계를 성적표로 읽으면 「낱말이 몇 종이니 나쁘다」와 같은 결론에 이르는데, 그 수치는 좋고 나쁨을 가리는 값이 아니라 표준을 새로 세울 때 기준을 정하는 근거이다.

표준 사전 없이 진단한 경우에는 개요에 그 사실을 함께 밝힌다. 뒤의 수치가 모두 스키마 내부에서 측정한 것이며 밖의 표준과 대조한 결과가 아니라는 점을 앞에서 알려야 한다.

> **i 조판 자산은 리포트 옆에 함께 생성된다**
>
> `--qmd` 로 생성한 문서 옆에 조판 자산이 함께 생성되며, 리포트는 그 자산을 상대 경로로 참조한다. 따라서 리포트를 다른 위치로 옮기거나 메일로 보낼 때에는 자산을 함께 옮긴다. 문서만 옮기면 조판이 적용되지 않는다.
>
> 자산은 생성할 때마다 덮어쓴다. 라이브러리를 갱신한 뒤 조판이 이전 모양으로 나온다면 옛 자산이 남은 것이므로 리포트를 다시 생성한다.

> ⚠ Quarto 와 한글 폰트가 필요하다
>
> `--render`는 Quarto 를 실행한다. 설치되어 있지 않으면 무엇을 설치해야 하는지 안내하고 중단한다. `--qmd`만 지정하면 지금까지처럼 문서만 생성하므로 Quarto 가 없어도 된다.
>
> ```
> 오류: Quarto 를 찾지 못했습니다. HTML·PDF 조판에 필요합니다.
>       brew install --cask quarto   (또는 https://quarto.org 에서 내려받기)
>       설치 경로가 PATH 에 없으면 --quarto 나 KDSM_QUARTO_BIN 로 알려 주십시오.
> ```
>
> 한글 폰트도 하나는 있어야 한다. 조판 자산은 Pretendard·Noto Sans KR·Apple SD Gothic Neo·맑은 고딕을 차례로 찾는다. 특정 폰트를 요구하지 않는 것은, 해당 폰트가 없는 환경에서 한글 전체가 표시되지 않기 때문이다.

## 18.3 사전이 없는 스키마

여기까지는 표준 사전이 이미 구축되어 있는 경우를 전제로 설명하는 것이다. 그런데 진단이 가장 필요한 대상은 대개 사전 없이 구축된 레거시이다. 비교할 기준이 없는 상태에서 무엇을 측정할 것인가 하는 문제가 발생한다.

사전이 있으면 그 사전을 함께 참조한다. 추정은 사전으로 판단할 수 없을 때 사용하는 방법이며 사전을 배제하고 사용하는 방법이 아니다. 이에 따라 두 항목의 판정이 달라진다. 접미사별 타입은 표준 도메인을 확인하고(18.3.2.1절), 비슷한 약어는 표준단어 등재 여부를 확인한다(18.3.2.2절).

### 18.3.1 스키마 내부에서 기준을 도출한다

외부에서 "무엇이 옳은가"에 대한 기준을 가져올 수 없으면 내부에서 도출한다. 전제는 하나이다. 여러 위치에서 같은 대상을 다르게 지칭하고 있다면 그중 하나는 잘못된 것이다. 무엇이 옳은지는 알 수 없더라도 서로 어긋나 있다는 사실은 확인할 수 있다.

길이를 측정하는 것이 아니라 수평 여부를 확인하는 것에 가깝다. 수평계는 대상의 길이를 제시하지 않는다. 다만 기울어져 있는지는 제시한다.

### 18.3.2 다섯 가지 구분

표 18.10. 사전 없이 확인하는 다섯 가지 구분

| 구분 | 확인 내용 | 예 |
|---|---|---|
| 표기 스타일 | 한 스키마에 표기법이 혼용되어 있는가 | `CUST_NM` 과 `custNm` |
| 같은 이름 다른 정의 | 같은 컬럼명의 타입이 서로 다른가 | `CUST_NM` 이 `VARCHAR(40)` 과 `VARCHAR(100)` |
| 접미사별 타입 | 같은 접미사의 타입이 서로 다른가 | `_YMD` 가 `DATE` 와 `CHAR(8)` |
| 비슷한 약어 | 같은 낱말을 다르게 축약하였는가 | `CUSTOMER` · `CUST` · `CUS` |
| 한글명 대응 | 주석의 한글명과 컬럼명이 일대일인가 | 같은 `고객번호` 에 컬럼명이 셋 |

접미사를 확인하는 이유는 사전이 없더라도 접미사가 사실상 분류어의 역할을 수행하기 때문이다. `_YMD`로 끝나는 컬럼을 정의한 사람은 그 컬럼이 일자임을 인지하고 있었다. 다만 타입을 합의하지 않았을 뿐이다.

#### 18.3.2.1 사전이 있으면 접미사 판정이 달라진다

표준도메인이 있으면 접미사별 타입이 서로 다른 것이 정상일 수 있다.

```
MBR_ID  VARCHAR(20)   회원식별자 → 번호V20
DOC_ID  VARCHAR(40)   문서식별자 → 식별자V40
```

둘 다 `_ID`로 끝나지만 타입이 다르다. 그런데 각각 등재된 표준용어의 도메인을 정확히 준수하고 있으므로 어느 쪽도 잘못되지 않았다. 접미사는 분류어일 뿐이고, 실제 타입을 정하는 것은 도메인이다.

그러므로 사전이 있을 때에는 도메인을 준수한 컬럼을 접미사 판정에서 제외한다. 남은 컬럼끼리만 비교한다.

> ! 표준화할수록 점수가 내려가면 안 된다
>
> 제외하지 않으면 표준을 준수한 쪽을 지적하는 결과가 된다. 사전을 갖추고 도메인을 정의할수록 접미사별 지적이 증가하고 일관성 점수가 하락하게 되어, 점수가 표준화의 진척과 반대로 움직인다.
>
> 제외한 컬럼이 있으면 지적 내용에 함께 기재한다.
>
> ```
> `_ID`로 끝나는 컬럼 2개가 2가지 타입으로 나뉩니다 (표준도메인을 따르는 2개는 뺐습니다)
> ```

도메인을 위반한 컬럼은 제외하지 않는다. 근거 있는 차이가 아니기 때문이다. 다만 그러한 컬럼은 이름이 표준을 준수하고 있으므로 미준수로 집계되지 않아 드러나기 어렵다. 그러므로 `표준도메인 위반`이라는 항목으로 진단 제언에 별도로 수록한다. 이름을 변경하지 않고 정의만 수정하면 되는 작업이므로 효용이 크다.

유사 약어는 한쪽이 다른 쪽의 앞부분일 때만 묶는다. `CUST`는 `CUSTOMER`의 앞부분이고 `CUS`는 `CUST`의 앞부분이므로 세 개가 하나의 그룹이 된다. 편집거리와 같은 기준을 사용하면 `AMT`와 `AMD`처럼 관련이 없는 약어까지 묶여 지적 항목이 지나치게 늘어난다.

#### 18.3.2.2 사전이 있으면 약어를 사전에 물어본다

앞부분이 일치한다는 사실은 추정의 근거일 뿐이다. 사전으로 판단할 수 있는 사항을 추정으로 판단해서는 안 된다.

```
STD     표준       표준단어로 등재
STDCD   표준코드   표준단어로 등재
```

`STDCD`는 `STD`로 시작하지만 사전에 두 약어가 서로 다른 낱말로 등재되어 있다. 이러한 경우는 묶지 않는다.

한쪽만 등재된 경우는 그대로 지적한다. `CUSTOMER`가 등재어인데 `CUST`를 50회 사용했다면 그것이야말로 지적해야 할 사항이기 때문이다. 이때는 어느 쪽이 표준인지도 함께 기록한다.

```
[비슷한 약어] CUST · CUSTOMER
    같은 말을 2가지로 줄여 쓴 것으로 보입니다.
    이 가운데 `CUSTOMER`가 표준단어로 등재되어 있습니다
    · CUST 50회
    · CUSTOMER 3회 (표준단어)
```

> **i 폐기된 단어는 등재어가 아니다**
>
> 상태가 폐기(03)인 표준단어는 등재어로 간주하지 않는다. 이미 사용하지 않기로 결정한 낱말을 근거로 지적을 면제하면, 폐기 처리를 수행할수록 문제가 은폐된다.

사전이 없으면 앞에서 설명한 방식으로 추정한다. 이때에는 어느 약어가 올바른 낱말인지 확인할 수 없으므로 앞부분이 일치하는지만을 근거로 판단한다. 그러므로 이 항목은 사전의 유무와 관계없이 일관성 총점에 반영하지 않는다.

#### 18.3.2.3 주석에서 한글명만 분리한다

한글명 대응은 컬럼 주석을 비교한다. 그런데 `한글명 - 설명` 형식으로 주석을 작성하는 관행이 일반적이다.

```
ENTTY_NM   엔터티이름 - 속성이 속한 엔터티명 1개
ENTTY_NM   엔터티이름 - 표준용어정의서에 등록된 용어를 사용하여 한글로 정의한 엔터티의 이름 1개
```

한글명은 둘 다 `엔터티이름`으로 동일하다. 차이가 있는 부분은 설명뿐이다. 주석 전체를 한글명으로 간주하면 "한 컬럼명에 한글명이 두 가지"라는 지적이 발생하지만, 실제로는 한글명이 같고 설명만 다른 경우이다.

그러므로 붙임표 왼쪽만 분리하여 비교한다. 붙임표가 없는 주석은 전체를 한글명으로 간주한다.

> **⚠ 붙임표 앞뒤의 공백을 확인한다**
>
> 분리 기준은 앞뒤에 공백이 있는 붙임표뿐이다.
>
> 표 18.11. 붙임표를 분리하는 기준
>
| 주석 | 한글명 |
|---|---|
| `엔터티이름 - 속성이 속한 엔터티명` | `엔터티이름` |
| `고객-법인 구분` | `고객-법인 구분` |
| `e-mail주소` | `e-mail주소` |
>
> 공백 없이 사용한 붙임표까지 분리하면 `고객-법인 구분`이 `고객`이 되어 존재하지 않는 이름이 생성된다. 붙임표 대신 en dash와 em dash를 사용한 경우도 함께 확인한다.

`진단 대상 컬럼` 시트에는 주석을 분리하지 않고 그대로 수록한다. 그 시트는 카탈로그에서 읽은 사실을 제시하는 자리이기 때문이다.

> **! 무엇을 표준으로 삼을지는 정해 주지 않는다**
>
> `CUSTOMER`·`CUST`·`CUS` 중 무엇이 옳은지는 이 진단으로 판단할 수 없다. 사용 빈도가 가장 높은 것을 앞에 배치할 뿐이다. 다수가 곧 정답은 아니다. 표준으로 채택할 대상은 업무를 아는 사람이 결정한다. 진단은 현재 세 가지로 나뉘어 있다는 사실을 제시하는 데 그친다.

### 18.3.3 일관성 점수는 어떻게 나오는가

총점에 반영하는 항목의 단순 평균이다. 항목마다 “해당 문제에 해당하지 않는 컬럼의 비율”을 산출하고 그 값을 평균한다.

표 18.12. 일관성 점수의 항목

| 항목 | 산출 방식 | 분모 | 총점 반영 |
|---|---|---|---|
| 표기 스타일 | 사용 빈도가 가장 높은 표기를 따르는 컬럼의 비율 | 전체 컬럼 | O |
| 같은 이름 다른 정의 | 타입이 서로 다른 컬럼명에 해당하지 않는 컬럼의 비율 | 전체 컬럼 | O |
| 접미사별 타입 | 접미사별 타입이 서로 다르지 않은 컬럼의 비율 (표준도메인이 정한 차이는 제외) | 밑줄로 나뉜 컬럼 | O |
| 한글명 대응 | 한글명과 컬럼명이 일대일로 대응하는 컬럼의 비율 | 주석에서 한글명을 확인할 수 있는 컬럼 | O |
| 비슷한 약어 | 앞부분이 일치하는 약어에 해당하지 않는 컬럼의 비율 | 전체 컬럼 | — |

가중치를 부여하지 않은 것은 무엇이 더 중요한지 외부에서 정할 근거가 없기 때문이다. 그 기준은 스키마마다 다르다.

### 18.3.4 분모는 항목마다 다르다

분모를 전체 컬럼으로 통일하지 않는다. 두 항목은 판정할 수 없는 컬럼을 가지고 있기 때문이다. 주석이 없는 컬럼은 한글명을 확인할 수 없고, 밑줄로 나뉘지 않은 이름은 마지막 토막이 곧 이름 전체이므로 접미사라 부를 수 없다.

전체 컬럼을 분모로 사용하면 이 컬럼들이 문제 없는 쪽으로 집계된다. 주석이 하나도 없는 스키마에서 한글명 대응이 100%가 되는 것이 그 예이다. 판정할 대상이 없으므로 어긋난 사례도 없기 때문이다. 주석을 작성하지 않을수록 점수가 상승하게 되므로, 각 항목이 판정한 범위만을 분모로 사용한다.

분모의 실제 개수는 항목마다 함께 제시한다. 「한글명이 있는 컬럼 1,240개 중」과 같이 산출 기준에 포함된다.

> **i 분모가 0이면 측정 불가로 표기한다**
>
> 주석이 하나도 없으면 한글명 대응을, 밑줄로 나뉜 이름이 하나도 없으면 접미사별 타입을 측정할 수 없다. 이때 그 항목은 점수를 산출하지 않고 「측정 불가」로 표기하며 총점 평균에서도 제외한다. 항목 자체는 표에서 삭제하지 않는다.
>
> 0%로 표기하면 준수하지 않은 것으로, 100%로 표기하면 준수한 것으로 해석된다. 둘 다 사실이 아니다. 표준 사전이 없을 때 준수율을 「측정 불가」로 표기하는 것과 같은 원칙이다. 항목을 삭제하지 않는 것은 동일한 대상을 두 번 진단하여 비교할 때 표의 구조가 달라지면 무엇이 변경된 것인지 확인할 수 없기 때문이다.

`비슷한 약어`만 총점에서 제외한다. 이 항목만 추정에 근거하기 때문이다. `CUST`와 `CUSTOMER`가 실제로 같은 낱말인지는 사람이 확인해야 판단할 수 있다. 추정을 총점에 반영하면 점수가 하락하였을 때 실제로 악화된 것인지 추정이 잘못된 것인지 구분할 수 없다. 점수는 산출하되 평균에서만 제외한다.

등급 기준은 준수율과 같다. 95점 이상 A, 85점 이상 B, 70점 이상 C, 50점 이상 D, 그 아래가 F이다.

항목별 점수와 등급 기준은 엑셀의 `일관성 점수` 시트에, Quarto 리포트에는 같은 이름의 표로 나온다.

### 18.3.5 지적 건수는 항목별로 구분하여 확인한다

"지적 87건"이라는 수치 하나만으로는 무엇이 87건인지 확인할 수 없다. 그러므로 구분별 건수, 해당하는 컬럼 수, 전체에서 차지하는 비중, 그중 규모가 가장 큰 건을 함께 제시한다. 엑셀에서는 `일관성 점수` 시트의 「지적 건수 내역」 블록이 이 다섯 열을 갖는다.

```
구분           지적 건수  영향 컬럼  비중(%)  가장 큰 건
접미사별 타입       23건      189개     26.4  _ID – `_ID`로 끝나는 컬럼 89개가 …
비슷한 약어         31건       74개     35.6  USER · USE · USED – 같은 말을 …
```

이 내용을 확인하면 어디부터 조치할지를 즉시 판단할 수 있다. 개별 지적 사항은 `일관성 지적` 시트에 수록되어 있다.

### 18.3.6 실행 방법

일관성 진단은 항상 함께 수행한다. 사전이 있는 경우에도 동일하다. 사전과 대조하여 100%가 산출되더라도 표기가 혼용되어 있을 수 있기 때문이다.

사전이 전혀 없는 스키마라면 `--no-dict`를 지정한다. 이미 구축한 표준 사전이 있다면 지정하지 않는다. 지정하면 준수율을 측정하지 않으므로, 사전이 있는데도 일관성 점수만 얻게 된다.

```
kdsm diagnose --host db.internal --database erp --schema legacy --no-dict
```

이때 세 가지 진단 대상의 항목에는 0%가 아니라 `측정 불가`라고 기록한다. 사전 없이 산출한 0%는 "표준을 준수하지 않았다"가 아니라 "비교 기준이 없었다"는 뜻인데, 수치로 기록하면 두 경우가 동일하게 보인다.

> ! 두 진단은 서로를 대체하지 않는다
>
> 표준 사전 대조와 사전 없는 진단은 항상 함께 제시된다. 사전이 없다고 하여 준수율 항목을 제거하지 않고, 사전이 있다고 하여 일관성과 통계를 제외하지 않는다.
>
> 리포트의 구성이 실행할 때마다 달라지면 같은 스키마를 두 번 진단하여 비교할 수 없다. 지난달 리포트에 있던 절이 이번 달에 없을 때, 그것이 개선되어 사라진 것인지 측정하지 못하여 누락된 것인지 확인할 방법이 없기 때문이다.

`--min-score`는 사전이 있으면 종합 점수를, 없으면 일관성 점수를 기준으로 판정한다. 엑셀과 Quarto 리포트도 사전의 유무와 관계없이 동일하게 생성된다.

```
kdsm diagnose --schema legacy --no-dict -o 진단.xlsx --qmd 진단.qmd
```

## 18.4 이름 통계

일관성 진단이 "어디가 어긋났는가"를 지적한다면, 통계는 "현재 어떤 상태인가"를 집계한다. 두 가지는 쓰임이 다르다. 지적은 수정할 대상을 제시하고, 통계는 표준을 수립할 때 무엇을 기준으로 삼을지 제시한다.

### 18.4.1 여섯 가지 구분

표 18.13. 이름 통계 여섯 가지 구분

| 구분 | 확인 내용 | 집계 단위 |
|---|---|---|
| 설명(Comment) 채움율 | 테이블과 컬럼에 주석이 작성되어 있는가 | 테이블·컬럼 |
| 낱말 길이 순위 | 현재 사용하는 약어가 몇 글자인가 | 낱말 |
| 전체 낱말 길이 순위 | 컬럼명 전체가 몇 글자인가 | 컬럼 |
| 낱말 개수 순위 | 컬럼명 하나가 낱말 몇 개로 구성되는가 | 컬럼 |
| 접미사 순위 | 컬럼명이 무엇으로 끝나는가 | 컬럼 |
| 최댓값 | 가장 긴 이름·낱말과 낱말이 가장 많은 이름 | 컬럼·낱말 |

낱말 길이와 전체 낱말 길이는 집계 단위가 다르다. 전자는 낱말을 단위로 집계하고, 후자는 컬럼을 단위로 집계한다. `CUST_NM` 한 컬럼은 낱말 길이 순위에 `4자`와 `2자`로 두 번 들어가지만, 전체 낱말 길이 순위에는 `7자`로 한 번만 들어간다.

낱말은 밑줄과 대소문자 경계를 기준으로 분리한다. `orderAmt`는 `ORDER`와 `AMT` 두 개로 집계한다. 밑줄로만 분리하면 `ORDERAMT` 하나가 되어 `ORDER`와 다른 낱말로 집계된다.

### 18.4.2 최댓값

순위표 앞에 세 가지 최댓값을 제시한다. 최대 전체 낱말 길이, 최대 낱말 길이, 최대 낱말 개수이다.

```
최대 전체 낱말 길이   19자  1건   data_item_unit_clss
최대 낱말 길이         8자  1건   HOMEPAGE
최대 낱말 개수         4개  1건   data_item_unit_clss
```

> ! 순위표만으로는 확인할 수 없다
>
> 순위는 건수가 많은 항목부터 나열한다. 그런데 가장 긴 이름은 대개 하나뿐이므로 `--top`의 범위에서 제외된다. 정작 조치가 필요한 항목이 표에서 누락되는 결과가 된다.
>
> 그러므로 최댓값은 `--top`과 무관하게 별도로 집계한다. `--top 3`으로 축소하여도 최댓값 항목은 그대로 출력된다.

값만 제시하지 않고 해당하는 이름을 함께 제시한다. "필요 이상으로 긴 이름이 있는가"에 답하려면 이름을 확인해야 조치 여부를 결정할 수 있기 때문이다. 같은 값을 가진 이름이 여럿이면 건수도 함께 제시한다.

무엇이 "필요 이상"인지는 제시하지 않는다. 업무에 따라 다르기 때문이다. 다만 물리명 길이 한도는 확정된 사실이므로 함께 기재한다. PostgreSQL은 식별자를 63바이트에서 절단한다.

### 18.4.3 활용 방법

낱말 길이 순위는 영문약어 규칙을 정할 때 참고한다. 현재 사용하는 낱말이 대부분 세 글자인데 약어를 다섯 글자로 정하면 기존 이름을 거의 전부 변경해야 한다. 표준을 준수하기 어렵게 만드는 대표적인 사례이다.

전체 낱말 길이는 낱말 길이의 합에 낱말을 연결하는 문자의 수를 더한 값이며, 곧 컬럼명 전체의 길이이다.

```
CUST_NM              4 + 2 + 밑줄 1        =  7자
orderAmt             5 + 3 + 이음 문자 0   =  8자
data_item_unit_clss  16 + 밑줄 3           = 19자
```

이 값으로 물리명 길이 한도를 초과할 위험을 확인한다. PostgreSQL은 식별자를 63바이트에서 절단한다. 낱말을 서너 개 연결하면 그 한도에 쉽게 도달한다.

> **i 이름이 길어지면 약어가 분화된다**
>
> 한도에 도달하면 낱말을 더 축약하여 사용하게 된다. `CUSTOMER`를 `CUST`로, 다시 `CUS`로 축약하는 현상이 이렇게 발생한다. 전체 낱말 길이 순위의 상위 값이 크다면 약어를 더 짧게 다시 정할 시점이다.
>
> `비슷한 약어` 지적이 많으면서 이 순위의 값도 크다면 두 현상은 같은 원인에서 비롯된 문제이다.

낱말 개수 순위는 표준용어의 `주제어 + 수식어* + 분류어` 구조와 대조한다. 낱말이 하나뿐인 컬럼이 많다면 분류어를 부여하지 않은 것이고, 다섯 개를 넘는 컬럼이 많다면 수식어를 과도하게 사용한 것이다.

접미사 순위는 표준 분류어의 우선 후보이다. 이미 사용하고 있는 낱말이므로 변경해야 할 이름이 가장 적다. 낱말이 하나뿐인 컬럼은 제외하고 집계한다. `ID`와 같이 낱말이 하나인 컬럼의 끝을 접미사로 간주하면 모든 컬럼이 자기 자신으로 끝나게 되어 아무런 정보를 제공하지 못한다.

설명 채움율은 값이 높을수록 좋다고 판단할 수 있는 유일한 지표이다. 주석이 없으면 해당 컬럼의 의미를 카탈로그에서 확인할 수 없다. 한글명 대응 진단도 주석이 있어야 수행할 수 있다.

채움율은 스키마 전체로만 산출하지 않고 테이블별로도 산출한다. 90%라는 수치에는 "모든 테이블이 조금씩 비어 있다"와 "테이블 다섯 개가 전부 비어 있다"가 모두 포함될 수 있는데, 두 경우는 필요한 조치가 전혀 다르다. 엑셀의 `주석 통계` 시트가 채움율이 낮은 테이블부터 나열하고, `주석 누락` 시트가 채워야 할 대상을 그대로 제시한다.

낱말 재사용 횟수도 함께 제시한다. 낱말 하나가 평균 몇 번 사용되었는지를 나타내는 값이다. 1에 가까울수록 낱말을 재사용하지 않고 매번 새로 만들었다는 뜻이며, 표준화의 효과가 가장 큰 상태이기도 한다.

> **i 통계에는 등급이 없다**
>
> 낱말이 세 개인 컬럼이 두 개인 컬럼보다 우수하다고 판단할 근거가 없다. 그러므로 통계에는 점수를 부여하지 않는다.
>
> 설명 채움율은 높을수록 좋은 지표이지만 이 값도 일관성 점수에는 포함하지 않았다. 일관성 점수는 불일치를 측정하는 것이고, 주석이 없는 것은 불일치가 아니라 미작성이다. 두 가지를 하나의 수치에 포함하면 다음 달에 점수가 하락했을 때 어느 쪽이 악화되었는지 알 수 없다.

순위표에 수록할 행 수는 `--top`으로 지정한다. 기본값은 10이다.

```
kdsm diagnose --schema legacy --no-dict --top 20
```

## 18.5 사용 방법

접속 대상, 사전의 위치, 진단할 스키마, 판정 기준을 지정하는 옵션이다.

### 18.5.1 접속 대상

기본값은 `KDSM_DB_*` 환경변수이다.

```
export KDSM_DB_HOST=db.internal
export KDSM_DB_PORT=5433
export KDSM_DB_NAME=erp                    # 데이터베이스 이름. 인자는 --database 이다
export KDSM_DB_USER=analyst
read -rs -p "비밀번호: " KDSM_DB_PASSWORD && export KDSM_DB_PASSWORD

kdsm diagnose --schema sales --schema billing
```

진단 대상이 평소 사용하는 데이터베이스와 다르면 인자로 덮어쓴다.

```
kdsm diagnose \
  --host db.internal --port 5433 --database erp --user analyst \
  --schema sales --schema billing
```

지정한 항목만 덮어쓰고 나머지는 환경변수의 값을 그대로 사용한다. 비밀번호는 인자가 없으므로 `KDSM_DB_PASSWORD`가 유일한 통로이다. 값을 노출하지 않고 지정하는 방법은 20.10절 에 있다.

대상을 자주 바꾼다면 인자를 매번 나열하는 대신 접두어를 나누는 편이 낫다(18.5.7절).

#### 18.5.1.1 파이썬에서 접속을 여는 방법

커맨드라인이 아니라 파이썬 코드에서 카탈로그를 읽는 경우에는 `kdsm.diagnose` 의 두 함수를 사용한다. 접속 정보를 환경변수가 아닌 곳에 보관하는 애플리케이션이 이에 해당한다.

```
from kdsm.diagnose import open_catalog

with open_catalog("postgresql", address="db.internal", database="erp",
                  schema="sales", user="analyst", password=secret) as source:
    columns = source.read_columns(["sales"])
    keys = source.read_constraints(["sales"])
```

`open_catalog` 는 접속을 열어 카탈로그 원천을 제공하고 블록을 벗어나면 연결을 닫는다. 예외로 벗어나는 경우에도 닫으므로 요청마다 연결을 여는 곳에서 연결이 누적되지 않는다. 설정 객체만 필요하면 `catalog_config` 를 사용한다. 두 함수의 인자는 동일하다.

접속 값은 DBMS에 매이지 않는 7종의 이름으로 지정한다. 해당 DBMS가 사용하지 않는 값은 제외된다.

표 18.14. 접속 값의 이름과 DBMS별 대응

| 지정하는 이름 | PostgreSQL | MySQL | Databricks |
|---|---|---|---|
| `address` | `host` | `host` | `server_hostname` |
| `port` | `port` | `port` | 제외 |
| `database` | `database` | `database` | `catalog` |

| 지정하는 이름 | PostgreSQL | MySQL | Databricks |
|---|---|---|---|
| `schema` | `schema_name` | 제외 | `schema_name` |
| `user` | `user` | `user` | `client_id` |
| `password` | `password` | `password` | `client_secret` 또는 `access_token` |
| `path` | 제외 | 제외 | `http_path` |

Databricks의 비밀은 `user` 의 유무에 따라 놓이는 자리가 달라진다. `user` 를 함께 지정하면 서비스 프린시펄의 클라이언트 시크릿으로, 지정하지 않으면 개인용 액세스 토큰으로 사용한다(18.5.4.1절).

> ⚠ 방언의 필드 이름을 지정하면 오류이다
>
> `host` 나 `server_hostname` 과 같이 해당 DBMS의 설정 객체가 사용하는 이름을 지정하면 오류로 중단한다.
>
> 안내 없이 무시하면 주소를 지정하였는데도 기본값인 `localhost` 에 접속을 시도하며, 그 실패는 접속되지 않는다는 형태로만 나타난다. 지정한 값이 전달되지 않았다는 사실은 드러나지 않는다.

`env_prefix` 를 지정하면 그 접두어의 환경변수를 기본값으로 삼고 인자로 지정한 값이 우선한다. 커맨드라인이 이 방식을 사용한다. 지정하지 않으면 환경변수를 읽지 않는다. 접속 정보를 별도로 보관하여 전달하는 쪽에서 서버의 환경변수가 함께 반영되면 의도하지 않은 데이터베이스에 접속하게 된다.

포트는 문자열로 지정할 수 있다. 표준도메인의 번호 계열에 숫자형이 없어 접속 정보를 표에 기재하면 문자형이 되기 때문이다. 숫자로 해석할 수 없는 값은 접속을 시도하기 전에 오류로 중단한다.

> ℹ DBMS별 분기는 이 함수에만 있다
>
> `--dbms` 를 해석하는 커맨드라인과 웹 애플리케이션의 ERD 화면이 모두 이 함수를 호출한다. 접속 값의 이름을 DBMS의 필드로 대응시키는 표가 두 곳에 있으면, 한쪽을 수정하지 않았을 때 의도하지 않은 서버에 접속하게 된다.

#### 18.5.1.2 파이썬에서 진단을 실행하는 방법

진단 한 번은 세 가지로 이루어진다. 사전과 대조한 준수율, 사전 없이 스키마 안에서만 본 일관성, 그리고 이름 통계이다. 셋을 한 번에 얻으려면 `diagnose_catalog` 를 사용한다.

```
from kdsm.diagnose import diagnose_catalog, open_catalog, to_text

with open_catalog("postgresql", address="db.internal", database="erp",
                  schema="sales", user="analyst", password=secret) as source:
    columns = source.read_columns(["sales"])

found = diagnose_catalog(columns, words, terms, domains)
print(to_text(found.report, found.consistency, found.stats))
```

돌려주는 `Diagnosis` 에 셋이 함께 담긴다. 리포트를 만드는 함수들이 셋을 함께 받으므로, 따로 들고 다니면 차례를 바꾸어 전달하는 오류를 형 검사가 검출하지 못한다.

표 18.15. Diagnosis 가 담는 것

| 항목 | 담긴 것 |
|---|---|
| `report` | 사전과 대조한 준수율 |
| `consistency` | 사전 없이 본 일관성 |
| `stats` | 이름 통계와 설명 채움율 |
| `score` | 한 숫자로 말해야 할 때의 값 |
| `measure` | 그 값이 무엇을 잰 것인가. `종합` 또는 `일관성` |
| `advice` | 조치 사항. 영향 컬럼이 많은 것부터 |

`score` 는 사전이 있으면 세 준수율의 평균이고, 없으면 일관성 점수이다. 사전 없이 잰 준수율 0%는 「표준을 준수하지 않았다」가 아니라 「측정하지 못했다」이므로 그 값을 기준값으로 사용하지 않는다. 무엇을 잰 값인지는 `measure` 가 알려 준다.

사전을 지정하지 않으면 사전 없는 진단이 된다. 커맨드라인의 `--no-dict` 와 같다.

```
found = diagnose_catalog(columns)
assert not found.report.has_dictionary
```

접속은 이 함수가 다루지 않는다. 카탈로그를 여는 일은 `open_catalog` 가 맡고 있고, 대상을 정하는 방법은 호출하는 쪽마다 다르다. 커맨드라인은 명령줄 인자와 환경변수로, 웹 애플리케이션은 등록된 접속 대상으로 정한다.

> **i 조립을 라이브러리가 갖는 이유**
>
> `kdsm diagnose` 명령과 웹 애플리케이션의 진단 화면이 모두 이 함수를 호출한다. 같은 데이터베이스를 진단하고 서로 다른 점수를 산출하면 어느 쪽이 옳은지 판단할 방법이 없다.
>
> 세 가지를 엮는 순서와 파생값을 산출하는 규칙을 한 곳에 두고 양쪽이 그것을 호출한다.

## 18.5.2 어느 DBMS를 진단하는가

`--dbms` 로 지정한다. 기본값은 `postgresql` 이며 `mysql` 과 `databricks` 를 함께 지원한다.

```
kdsm diagnose --dbms mysql --schema shop
kdsm diagnose --dbms databricks --schema main.sales
```

DBMS 가 달라도 사용법은 같다. 다른 것은 타입 이름의 표기와 길이를 측정할 수 있는지 여부이며, 그 차이는 진단 결과의 해석에만 영향을 준다.

표 18.16. 지금 진단할 수 있는 DBMS

| DBMS | 카탈로그 리더 | 접속 |
|---|---|---|
| `postgresql` | `information_schema` + `col_description()` | `psycopg` |
| `mysql` | `information_schema` | `pymysql` |
| `databricks` | `<카탈로그>.information_schema` | `databricks-sql-connector` |

같은 카탈로그 리더를 `kdsm erd --from-catalog`와 `kdsm export --from-catalog`도 사용한다. 진단은 이름과 도메인을 사전과 대조하고, ERD는 구조를 도면으로 작성하며, 내보내기는 구조를 정의서 서식에 담는다. 접속 방법과 `--dbms`·`--schema` 사용법은 세 명령이 동일하다(16.3.1절 · 17.6절).

#### 18.5.2.1 접두어는 DBMS마다 다르다

DBMS가 다르면 읽는 환경변수 자체가 다르다. PostgreSQL은 `KDSM_DB_*`, MySQL은 `MYSQL_*`, Databricks는 `DATABRICKS_*`를 사용한다. `--dbms`를 지정하면 해당 DBMS의 접두어를 기본으로 삼는다.

`--env-prefix`로 다른 접두어를 지정하면 경고한다. 접두어를 공유하려는 의도는 대개 「같은 서버를 다른 이름으로 가리킨다」인데, DBMS가 다르면 호스트도 포트도 다르기 때문이다.

```
경고: mysql 접속에 --env-prefix 로 지정한 ECODI_DB_ 를 사용합니다. 기본값은 MYSQL_ 입니다.
      접두어를 공유하면 호스트와 포트가 다른 DBMS의 값으로 채워집니다.
      대상이 다른 서버라면 MYSQL_* 를 사용하거나 --host·--port 로 지정하십시오.
```

> ⚠ 접두어를 공유하면 포트가 어긋난다
>
> PostgreSQL 설정을 그대로 MySQL에 사용하면 `<접두어>PORT`의 5432를 읽어 PostgreSQL 서버에 접속을 시도한다. MySQL 드라이버가 보낸 인사말에 서버가 응답하지 않으므로 handshake 도중 연결이 끊어진다.
>
> ```
> 오류: MySQL에 연결하지 못했습니다: ecodi@prd-rds.example.com:5432
>       원인: (2013, 'Lost connection to MySQL server during query')
>       포트 5432는 PostgreSQL의 기본 포트입니다. MySQL의 기본 포트는 3306입니다.
> ```
>
> 원인이 포트에 있다는 것을 알기 어려운 실패이므로, 오류 메시지가 포트 번호를 보고 짚어 준다.

경고일 뿐 실행을 막지는 않는다. 두 DBMS를 같은 서버에 두고 접두어를 통일하여 관리하는 경우가 있을 수 있고, 그 판단은 사용자의 몫이다.

### 18.5.3 MySQL

선택 기능이다. `uv sync --extra mysql`로 드라이버를 설치해야 사용할 수 있다.

```
export MYSQL_HOST=db.internal
export MYSQL_DATABASE=shop
export MYSQL_USER=reader
export MYSQL_PASSWORD='...'
```

```
kdsm diagnose --dbms mysql --schema shop -o 진단.xlsx
```

MySQL에는 데이터베이스 아래에 스키마를 두는 층이 없다. `information_schema.COLUMNS.TABLE_SCHEMA`가 데이터베이스 이름을 그대로 돌려주므로, `--schema`에 데이터베이스 이름을 지정한다.

> ⚠ 비밀번호는 인수로 받지 않는다
>
> `--host`·`--port`·`--user`는 인수로 지정할 수 있으나 비밀번호는 `MYSQL_PASSWORD`가 유일한 통로이다. 명령줄에 기재한 값은 같은 서버에 접속한 다른 사용자의 `ps` 출력과 셸 기록에 남는다(20.10절).
>
> 지정하지 않으면 빈 비밀번호로 접속을 시도하고 서버가 거부한다. 이 경우 오류 메시지가 변수 이름을 함께 안내한다.
>
> ```
> 오류: MySQL에 연결하지 못했습니다: ecodi@localhost:3306
>       원인: (1045, "Access denied for user 'ecodi'@'localhost' (using password: NO)")
>       비밀번호가 설정되어 있지 않습니다. MYSQL_PASSWORD 환경변수로 지정하십시오.
> ```

`information_schema`·`mysql`·`performance_schema`·`sys`는 MySQL이 스스로 만들어 두는 것이므로 지정하더라도 대상에서 제외한다.

주석은 `COLUMN_COMMENT`와 `TABLE_COMMENT`에 담겨 있다. PostgreSQL처럼 별도의 함수를 호출하지 않아도 되지만, 주석이 없으면 `NULL`이 아니라 빈 문자열로 반환한다. 빈 값은 주석이 없는 것으로 판정한다.

## 18.5.4 Databricks Unity Catalog

선택 기능이다. `uv sync --extra databricks`로 커넥터와 SDK를 설치해야 사용할 수 있다.

### 18.5.4.1 인증 방식 두 가지

표 18.17. Databricks 인증 방식

| 방식 | 자격 증명 | 사용하는 곳 |
|---|---|---|
| OAuth M2M | 서비스 프린시펄의 클라이언트 아이디와 시크릿 | 무인 실행. 배치와 CI |
| 액세스 토큰(PAT) | 개인용 토큰 | 사람이 직접 실행할 때 |

무인 실행에는 OAuth M2M을 권장한다. 개인용 토큰은 사람에게 매인 자격 증명이라 담당자가 바뀌면 끊기고, 만료도 사람이 챙겨야 한다.

```
# OAuth M2M — 서비스 프린시펄
export DATABRICKS_HOSTNAME='adb-1234.azuredatabricks.net'
export DATABRICKS_HTTP_PATH='/sql/1.0/warehouses/abc123'
export DATABRICKS_CLIENT_ID='xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
export DATABRICKS_CLIENT_SECRET='dose...'
export DATABRICKS_CATALOG=main
```

```
# 액세스 토큰
export DATABRICKS_HOSTNAME='adb-1234.azuredatabricks.net'
export DATABRICKS_HTTP_PATH='/sql/1.0/warehouses/abc123'
export DATABRICKS_TOKEN='dapi...'
```

`DATABRICKS_CLIENT_ID`와 `DATABRICKS_CLIENT_SECRET`이 둘 다 있으면 OAuth M2M으로 접속하고, 아니면 토큰으로 접속한다. 둘 다 설정되어 있으면 OAuth를 선택한다. 무인 실행에서 더 안전한 쪽이며, 토큰을 지우지 않은 채 OAuth로 옮겨 가는 중일 수 있기 때문이다.

호스트명은 `DATABRICKS_HOSTNAME` · `DATABRICKS_SERVER_HOSTNAME` · `DATABRICKS_HOST` 순으로 확인한다. Databricks 문서와 SDK 예제는 `SERVER_HOSTNAME` 을 사용하고, 다른 DBMS 와 접두어 체계를 맞추면 `HOST` 가 되므로 세 이름을 모두 받아들인다. 어느 이름으로 지정하여도 동작한다.

> ! 접속은 되는데 권한이 없다고 나올 때
>
> 자격 증명이 맞아도 SQL 웨어하우스를 사용할 권한이 별도로 필요하다. 인증과 권한은 다른 층위이다.
>
> ```
> 오류: Databricks에 접속하지 못했습니다: adb-1234.azuredatabricks.net (OAuth M2M)
>       원인: PERMISSION_DENIED: <서비스 프린시펄> is not authorized to use this SQL Endpoint.
>       인증은 통과하였으나 SQL 웨어하우스를 사용할 권한이 없습니다.
>       워크스페이스의 SQL → SQL Warehouses → 해당 웨어하우스 → Permissions 에서 OAuth M2M 주체에
> CAN USE 를 부여해야 합니다.
>       권한 부여는 워크스페이스 관리자만 수행할 수 있습니다.
> ```
>
> 워크스페이스 관리자에게 해당 서비스 프린시펄의 `CAN USE` 부여를 요청한다. 오류 문구에 주체의 식별자가 실려 있으므로 그대로 전달하면 된다. 클라이언트 아이디와 시크릿은 확인할 필요가 없다. 그 값이 틀렸다면 `invalid_client` 로 먼저 막힌다.

#### 18.5.4.2 웨어하우스를 가리키는 두 가지 방법

커넥터는 어느 SQL 웨어하우스에 접속할 것인지를 알아야 한다. 인증과 무관하며 두 인증 방식 모두에 필요하다. 운영 조직이 제공하는 형태가 경로인 경우와 식별자인 경우가 있으므로 둘 다 지원한다.

표 18.18. 웨어하우스를 가리키는 방법

| 변수 | 주는 값 |
|---|---|
| `DATABRICKS_HTTP_PATH` | `/sql/1.0/warehouses/<식별자>` |
| `DATABRICKS_WAREHOUSE_ID` | `<식별자>` |

식별자만 있으면 `/sql/1.0/warehouses/<식별자>`로 경로를 만든다. 값은 워크스페이스의 SQL → SQL Warehouses → 웨어하우스 선택 → Connection details 탭에 있다.

둘 다 없으면 접속을 시도하기 전에 어디에서 찾는지 함께 알린다.

```
오류: Databricks 접속 정보가 모자랍니다: DATABRICKS_HTTP_PATH
      어느 SQL 웨어하우스에 접속할지 가리키는 값입니다. 둘 가운데 하나로 알려 주십시오.
        DATABRICKS_HTTP_PATH=/sql/1.0/warehouses/<식별자>
        DATABRICKS_WAREHOUSE_ID=<식별자>
      워크스페이스의 SQL → SQL Warehouses → Connection details 에 있습니다.
```

#### 18.5.4.3 대상을 가리키는 표기

Unity Catalog는 `카탈로그.스키마.테이블` 세 단계이므로, `--schema` 에 주는 값이 스키마를 뜻하는지 카탈로그를 뜻하는지 갈린다. 점의 유무로 구분한다.

표 18.19. `--schema` 에 주는 값

| 주는 값 | 대상 |
|---|---|
| `finlife` | 기본 카탈로그 안의 `finlife` 스키마 |

| 주는 값 | 대상 |
|---|---|
| `dev_landing.finlife` | `dev_landing` 카탈로그의 `finlife` 스키마 |
| `dev_landing.` | `dev_landing` 카탈로그 전체 |

기본 카탈로그는 `--catalog` 또는 `DATABRICKS_CATALOG` 로 지정한다. 카탈로그 이름만 적으며 점은 붙이지 않는다. 점은 `--schema` 의 구분자이다.

이름 하나를 스키마로 해석하는 것은 기본 카탈로그가 지정되어 있을 때이다. 지정되어 있지 않으면 이름 하나를 카탈로그로 본다. 두 경우를 오가지 않으려면 `--catalog` 를 함께 지정하는 편이 낫다.

`--schema` 를 생략하면 기본 카탈로그의 스키마를 모두 진단한다. 이때 `information_schema` 는 제외한다. 카탈로그 안에 스키마가 여럿인 것이 보통이므로, 하나를 골라야 한다면 `--schema` 로 지정한다.

> 대상은 맞는데 컬럼이 없다고 나올 때
>
> 카탈로그 이름을 `--schema` 로 주면 그 이름의 스키마를 찾는다. 대개 그런 스키마가 없으므로 결과가 비어 나온다.
>
> ```
> 오류: 진단할 컬럼이 없습니다. 대상을 확인하세요: dev_landing
> ```
>
> 카탈로그는 `--catalog` 로 지정하고 `--schema` 는 생략한다. 특정 스키마만 보려면 `--schema dev_landing.finlife` 처럼 두 단계를 모두 적는다.

#### 18.5.4.4 필요한 권한

권한은 두 층으로 나뉜다. 접속할 권한과 읽을 권한이 다르다.

표 18.20. Databricks 진단에 필요한 권한

| 대상 | 필요한 권한 | 없을 때 |
|---|---|---|
| SQL 웨어하우스 | `CAN USE` | `PERMISSION_DENIED` 로 접속에 실패한다 |
| 카탈로그 | `USE CATALOG` | 그 카탈로그의 행이 조회되지 않는다 |
| 스키마 | `USE SCHEMA` | 그 스키마의 행이 조회되지 않는다 |
| 테이블 | `SELECT` | 그 테이블의 행이 조회되지 않는다 |

앞의 하나와 뒤의 셋은 실패하는 방식이 다르다. 웨어하우스 권한이 없으면 오류로 중단되므로 즉시 알 수 있다. 나머지 셋은 오류가 발생하지 않는다.

! 권한이 없는 대상은 조용히 빠진다

Unity Catalog의 `information_schema`는 요청한 주체가 볼 권한을 가진 관계만 반환한다. 권한이 없는 테이블은 오류가 아니라 목록에서 제외된다.

그러므로 권한이 일부만 부여된 상태에서 진단하면 준수율이 실제보다 높거나 낮게 산출된다. 문제가 있는 스키마가 조회 대상에서 빠졌는데 결과는 정상으로 보인다.

진단을 시작하기 전에 대상 규모를 먼저 확인한다. 화면 리포트의 컬럼 수와 아래 조회 결과가 일치하지 않으면 권한이 부족한 것이다.

```
SELECT count(*)
  FROM dev_landing.information_schema.columns
 WHERE table_schema <> 'information_schema'
```

정기 점검으로 활용하려면 서비스 프린시펄에 카탈로그 단위로 `USE CATALOG`와 `SELECT`를 부여한다. 스키마가 추가될 때마다 권한을 다시 부여하지 않아도 된다.

#### 18.5.4.5 진단하는 테이블 유형

`MANAGED`와 `EXTERNAL` 두 유형을 진단한다. 저장소를 Unity Catalog가 관리하는지 여부가 다를 뿐, 둘 다 실체가 있는 테이블이며 컬럼명은 동일하게 표준화 대상이다.

`VIEW`는 제외한다. 뷰의 컬럼명은 대개 원본 테이블에서 도출된 것이므로 동일한 문제를 두 번 집계하게 된다. 각 카탈로그의 `information_schema`도 제외한다. 사용자가 정의한 대상이 아니기 때문이다.

대상 규모가 예상과 다르면 유형별로 확인한다.

```
SELECT table_type, count(*)
  FROM dev_landing.information_schema.tables
 WHERE table_schema <> 'information_schema'
 GROUP BY table_type
```

#### 18.5.4.6 기본키와 외래키는 선언일 뿐이다

Unity Catalog의 기본키·외래키 제약은 선언적 정보이며 강제되지 않는다. 값의 중복이나 참조 무결성을 데이터베이스가 검사하지 않는다는 뜻이다. 질의 최적화에만 사용된다.

이 제약은 Databricks Runtime 13.3 LTS 이상에서 선언할 수 있고 15.2 이상에서 정식 기능이 되었다. 그 이전에 만들어진 테이블에는 제약이 선언되어 있지 않은 경우가 많다.

kdsm은 선언된 제약을 그대로 읽는다. 선언이 없으면 다음과 같이 나타난다.

- ERD에 관계선이 그려지지 않는다(16.3.1절)
- 내보낸 정의서의 기본키 순번과 참조 항목이 공란으로 남는다(17.6절)

제약이 없다고 하여 관계가 없는 것은 아니다. 선언하지 않았을 뿐이다. 이 경우 관계는 사람이 파악하여 정의서에 기재한다. 반대로 제약이 선언되어 있더라도 데이터가 그것을 지킨다는 보장은 없으므로, 실제 값의 정합성은 별도로 확인한다.

### 18.5.5 문자열 길이는 타입에 따라 다르다

문자열 타입이 둘이다. `STRING`은 길이 제한이 없고, `VARCHAR(n)`은 n 자를 넘는 값을 거부한다. 진단이 길이를 확인할 수 있는지도 이에 따라 갈린다.

표 18.21. Databricks에서 길이를 확인할 수 있는 경우

| 컬럼의 타입 | 길이 확인 | 근거 |
|---|---|---|
| `VARCHAR(40)` | 확인한다 | `FULL_DATA_TYPE`에 적힌 표기에서 읽는다 |
| `STRING` | 확인하지 못한다 | 길이라는 개념이 없다 |

길이를 `CHARACTER_MAXIMUM_LENGTH`에서 읽지 않는다. 그 칸은 문서에 기술된 대로 항상 `NULL`이다. `VARCHAR(40)`으로 만든 컬럼도 이 칸은 비어 있으므로, 컬럼 정의에 적힌 그대로를 돌려주는 `FULL_DATA_TYPE`에서 타입 이름과 길이를 분리하여 읽는다.

> ⚠ STRING 컬럼은 길이를 확인할 수 없다
>
> `STRING`으로 만들어진 컬럼은 `명V40`에 대하여 "40자가 맞는가"를 확인할 방법이 없다. 논리 타입만 대조하고 길이는 측정하지 못하였음을 명시한다.
>
> ```
> 표준도메인 100.0%  비교 가능한 컬럼 2개 · 길이를 측정하지 못한 컬럼 1개
> ```
>
> "40자가 아니므로 위반"이라고 판정하면 측정할 수 없는 항목을 오류로 처리하는 것이고, "일치한다"고만 기재하면 확인하지 않은 항목을 확인한 것으로 처리하는 것이다. 어느 쪽도 사실이 아니므로 측정하지 못하였음을 그대로 기재한다. 이 경우 준수율은 길이를 제외한 값으로 해석한다.
>
> 길이를 확인할 수 있게 하려면 테이블을 `VARCHAR(n)`으로 생성한다(14.7.3절). `DECIMAL(13,2)`와 같이 자릿수가 있는 타입은 처음부터 그대로 대조한다.

사전은 수정하지 않아도 된다. `PHYS_DATA_TYPE_CN`에 `VARCHAR(40)`이 기재되어 있더라도 Databricks를 진단할 때에는 그 값을 사용하지 않고 논리 타입(`DATA_TYPE_CN`)과 길이로부터 대상 DBMS의 표기를 생성한다. 하나의 사전으로 여러 DBMS를 진단할 수 있다.

### 18.5.6 Databricks의 타입 이름

카탈로그가 알려 주는 타입 이름은 DBMS마다 다르다. 사전과 대조하려면 표준 표기로 맞추어야 한다. 원문은 그대로 보존하고 맞춘 이름을 따로 보관하므로, 판정 근거를 나중에 확인할 수 있다.

표 18.22. Databricks 타입과 표준 표기

| Databricks | 표준 표기 | 비고 |
|---|---|---|
| `STRING` | `STRING` | 바꾸지 않는다. 길이라는 개념이 없다 |
| `VARCHAR(n)` | `VARCHAR` | 길이를 떼어 내어 함께 대조한다 |
| `INT` | `INTEGER` | |
| `LONG` | `BIGINT` | |
| `SHORT` | `SMALLINT` | |
| `BYTE` | `TINYINT` | |
| `DECIMAL` · `DEC` | `NUMERIC` | 전체 자릿수와 소수 자릿수를 그대로 대조한다 |
| `TIMESTAMP_NTZ` | `TIMESTAMP` | 시간대 없는 타입 |

`STRING`을 `VARCHAR`로 바꾸지 않는 것은 의도적이다. `STRING`에는 길이라는 개념이 없으므로 `VARCHAR`로 옮겨 적으면 길이가 있는 타입처럼 보인다. 이름을 그대로 두고 길이를 측정하지 못하였음을 밝히는 편이 사실에 가깝다. 반면 `VARCHAR(n)`은 길이가 실제로 강제되므로 그대로 두고 길이까지 대조한다(18.5.5절).

식별자는 큰따옴표가 아니라 backtick으로 감싼다. Unity Catalog의 인용 부호이다.

### 18.5.7 데이터베이스가 둘일 때

진단 대상과 표준 사전이 서로 다른 서버에 있는 경우가 많다. 레거시를 진단하면서 사전은 새로 구축한 데이터베이스에 두는 경우가 오히려 일반적이다. 이때에는 호스트와 사용자, 비밀번호가 모두 다르므로 접속 정보가 두 벌 필요하다.

`--host`와 같은 인자로는 한 벌만 지정할 수 있고, 비밀번호는 인자로 받지 않는다. 그러므로 환경변수 접두어를 구분하여 사용한다.

```
# 진단 대상
export ECODI_DB_HOST=prd-extdata-common-rds....rds.amazonaws.com
export ECODI_DB_PORT=5432
export ECODI_DB_NAME=ecodi
export ECODI_DB_SCHEMA=ecodi_meta
export ECODI_DB_USER=d_db_dev
export ECODI_DB_PASSWORD='...'

# 표준 사전
export KDSM_DB_HOST=dev-internal-...-aiplatform-rds....rds.amazonaws.com
export KDSM_DB_PORT=5432
export KDSM_DB_NAME=aiplatform
export KDSM_DB_SCHEMA=kdsm_meta
export KDSM_DB_USER=kdsm
export KDSM_DB_PASSWORD='...'
```

```
kdsm diagnose --env-prefix ECODI_DB_ -o 진단.xlsx --qmd 진단.qmd
```

`--env-prefix`는 진단 대상을, `--dict-env-prefix`는 사전이 있는 데이터베이스를 지정한다. 둘 다 생략하면 `KDSM_DB_`가 적용된다. 위 예에서 사전 쪽을 생략한 것은 기본값이 이미 `KDSM_DB_`이기 때문이다.

표 18.23. 접속 정보 두 벌

| 인자 | 읽는 환경변수 | 기본 |
|---|---|---|
| `--env-prefix` | 진단 대상의 접속 정보 | `KDSM_DB_` |
| `--dict-env-prefix` | 표준 사전의 접속 정보 | `KDSM_DB_` |

두 접두어가 같은 데이터베이스를 가리키더라도 연결은 각각 생성한다. 진단 대상이 Databricks 이거나 MySQL 인 경우가 있어 두 연결의 방식 자체가 다를 수 있고, 같은 PostgreSQL 이어도 대상과 사전이 다른 서버인 경우가 잦기 때문이다. 대상 연결을 사전 조회에 물려 쓰면 엉뚱한 곳에서 메타테이블을 찾게 된다.

> 접두어 하나로 진단 대상을 교체한다
>
> 여러 스키마를 진단한다면 대상마다 접두어를 정의해 두는 것이 편리하다. 명령이 간결해지고, 각 데이터베이스의 비밀번호가 셸 설정 파일 한 곳에 모이다.
>
> ```
> kdsm diagnose --env-prefix ECODI_DB_ -o ecodi_진단.xlsx
> kdsm diagnose --env-prefix CRM_DB_   -o crm_진단.xlsx
> ```

> ⚠ 비밀번호는 인자로 받지 않는다
>
> `--password` 옵션은 의도적으로 제공하지 않는다. 명령줄에 기재하면 같은 서버에 접속한 다른 사용자의 `ps` 출력에 그대로 표시되고, 셸 기록 파일에도 남는다. `KDSM_DB_PASSWORD`나 `PGPASSWORD`를 사용한다.

### 18.5.8 사전은 어디서 읽는가

기본값은 데이터베이스의 `kdsm_meta` 테이블이다. 진단 대상과 사전이 같은 연결에 있는 경우에 해당한다. 메타파일과 대조하려면 `--meta`를 지정한다.

```
kdsm diagnose --schema sales --meta dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx
```

사전은 `--dict-env-prefix`가 가리키는 데이터베이스의 `<접두어>SCHEMA`에서 읽는다. 기본값은 `KDSM_DB_SCHEMA`이며, 이 값도 없으면 `kdsm_meta`를 사용한다. 같은 데이터베이스의 다른 스키마라면 `--dict-schema`만 지정하면 된다.

```
kdsm diagnose --schema sales --dict-schema our_meta
```

> ⚠ 사전은 진단 대상 스키마에서 찾지 않는다
>
> 진단 대상 스키마는 이 명령에서 진단할 위치를 가리킨다. 그 값을 사전의 위치로도 사용하면 레거시 스키마에서 `MT_STD_WORD`를 탐색하다가 항상 실패한다. 그러므로 사전의 위치는 진단 대상과 별도로 지정한다.

레거시 스키마를 진단할 때 `--no-dict`를 먼저 사용하는 경우가 많은데, 이미 구축한 표준 사전이 있다면 지정하지 않는 것이 적절하다. 준수율까지 함께 산출된다. `--no-dict`는 비교할 사전이 실제로 없을 때만 사용한다.

사전을 읽지 못하거나 읽은 결과가 비어 있으면 오류로 중단하지 않고 경고를 출력한 뒤 일관성 진단만 수행한다. 안내 없이 진행하면 리포트의 준수 항목이 모두 `-`로 표시되는데, 리포트를 받은 사람은 그것이 진단 결과인지 오류인지 확인할 수 없다.

### 18.5.9 CI 연계

기준선을 유지하려면 `--min-score`를 지정한다. 이 경우에만 종료 코드가 1이 될 수 있다.

```
kdsm diagnose --schema sales --min-score 85 -q
```

```
컬럼 154개 · 용어 100.0% · 단어 100.0% · 도메인 100.0% · 종합 100.0(A) · 일관성 100.0(A) · 설명 100.0%
```

> 기준선은 지금 점수에서 시작한다
>
> 처음부터 90점을 기준으로 설정하면 매번 실패하여 결과를 확인하지 않게 된다. 현재 점수를 기준선으로 설정하고, 개선할 때마다 단계적으로 상향하는 것이 적절하다. 목적은 기준 통과가 아니라 상태가 악화되지 않도록 하는 것이다.

### 18.5.10 대상 스키마

`--schema`로 지정한다. 여러 번 지정할 수 있으며, 생략하면 `KDSM_DB_SCHEMA`를 사용한다.

`pg_catalog`·`information_schema`와 같은 시스템 스키마는 항상 제외한다. 사용자가 정의한 대상이 아니기 때문이다. 뷰도 진단하지 않는다. 뷰의 컬럼명은 대개 원본 테이블에서 도출된 것이므로 같은 문제를 두 번 집계하게 된다.

Databricks에서는 `MANAGED`와 `EXTERNAL` 두 유형을 모두 진단한다. 저장소를 Unity Catalog가 관리하는지 여부가 다를 뿐, 둘 다 실체가 있는 테이블이며 컬럼명은 똑같이 표준화 대상이다. 적재 영역은 외부 저장소를 가리키는 `EXTERNAL`로 구성하는 경우가 많다.

### 18.5.11 전체 옵션

**표 18.24.** kdsm diagnose 옵션

| 옵션 | 설명 |
|---|---|
| `--dbms` | 진단 대상 DBMS. `postgresql`(기본)·`mysql`·`databricks`(18.5.2절) |
| `--catalog` | Databricks 기본 카탈로그. 생략하면 `<접두어>CATALOG` |
| `--schema` | 진단할 스키마. 여러 번 지정할 수 있다 |
| `--env-prefix` | 진단 대상의 접속 정보 환경변수 접두어 (기본 `KDSM_DB_`) |
| `--host` `--port` `--database` `--user` | 환경변수의 접속 정보를 덮어쓴다 |
| `--meta` | 사전을 메타파일에서 읽는다. 생략하면 데이터베이스에서 읽는다 |
| `--no-dict` | 사전 없이 일관성만 진단한다 |
| `--dict-env-prefix` | 사전이 있는 데이터베이스의 환경변수 접두어 |
| `--dict-schema` | 사전이 있는 스키마. 생략하면 `<접두어>SCHEMA` 또는 `kdsm_meta` |
| `--top` | 이름 통계 순위표에 수록할 행 수 (기본 10) |
| `-o`, `--output` | 엑셀 리포트를 저장할 경로 |
| `--qmd` | Quarto 리포트를 저장할 경로 |
| `--render` | `--qmd` 문서를 조판한다. `html`·`pdf`·`all`. 생략하면 둘 다 (18.2.6절) |

| 옵션 | 설명 |
| --- | --- |
| `--quarto` | Quarto 실행 파일 경로. 생략하면 `KDSM_QUARTO_BIN` 과 `PATH` |
| `--min-score` | 이 점수에 미달하면 종료 코드 1로 종료한다 |
| `-q`, `--quiet` | 한 줄 요약만 출력한다 |

비밀번호는 인자로 받지 않는다. 명령줄에 기재한 값은 `ps` 출력과 셸 기록에 남기 때문이다. `<접두어>PASSWORD` 환경변수를 사용한다.

CHAPTER 19

# 커맨드라인

```
kdsm ddl <산출물>... [-o 출력]                # DDL 생성
kdsm drift -s <산출물>... [--schema 스키마]   # 정의서와 운영 DB 대조
kdsm validate <산출물>... [--meta 메타파일]   # 표준 준수 검증
kdsm load <메타파일> [-s 산출물]              # 메타데이터 파일 적재
kdsm apply <산출물>...                        # 산출물 파일 적재
kdsm suggest <용어명>... (--meta <메타파일> ¦ --from-db)   # 표준용어 추천
kdsm diagnose [--schema <스키마>]...          # 표준화 수준 진단
kdsm erd [-s 산출물]... [-m 모델] [-f 형식]    # ERD 자동 작성
kdsm export [--target standard¦logical¦physical¦all] [--from-catalog]  # 산출물 자동 생성
```

## 19.1 공통 규약

아홉 명령이 공통으로 지키는 종료 코드와 오류 표시 방식이다.

### 19.1.1 종료 코드

표 19.1. 종료 코드

| 코드 | 뜻 |
|---|---|
| 0 | 정상 |
| 1 | 처리 오류 또는 검증 실패 |
| 2 | 사용법 오류 (argparse) |

> **i** 0은 정상 처리를 의미해야 한다
>
> `validate`는 ERROR가 있으면 1, `load`는 검증에 실패하여 적재하지 못한 경우에도 1, `suggest`는 후보를 하나도 생성하지 못하면 1이다. 스크립트에서 그대로 활용할 수 있다.
>
> `export`는 산출물을 하나도 생성하지 못하면 1이다. `erd`는 상위 엔터티나 기본키가 어긋나 관계를 그리지 못한 경우에 1이다.
>
> `diagnose`와 `drift`는 다르다. 측정하고 대조하는 작업이지 합격 여부를 판정하는 작업이 아니므로 기본값은 항상 0이다. `diagnose`는 `--min-score`, `drift`는 `--strict`를 지정한 경우에만 1이 될 수 있다.

### 19.1.2 오류 표시

`KdsmError` 계열은 추적 정보 없이 메시지만 출력한다.

```
오류: 정의서에 없는 테이블입니다: MT_GHOST
```

파일을 찾지 못한 경우나 정의가 모순된 경우는 사용자가 조치할 사항이므로 스택 추적 없이 메시지만 출력한다. 스택 추적이 함께 출력되었다면 조치 대상이 아니라 라이브러리의 결함이다.

## 19.2 환경변수

데이터베이스에 접속하는 명령은 `KDSM_DB_HOST`부터 `KDSM_DB_PASSWORD`까지 여섯 환경변수를 읽는다. 접두어는 `--env-prefix`로 바꾼다. 파일만 읽는 명령에는 환경변수가 필요하지 않다.

`--dbms`를 받는 명령은 일곱이다. 그 가운데 `ddl`은 파일만 만들므로 접속하지 않는다. 나머지 여섯이 대상 DBMS 의 접속 정보를 따로 읽는다. `load`·`apply`는 넣을 곳을, `diagnose`·`drift`·`erd`·`export`는 읽을 곳을 가리킨다. 뒤의 넷은 카탈로그를 읽든 메타테이블을 읽든 이 옵션을 따른다. 대상이 PostgreSQL 이면 `KDSM_DB_*`, MySQL 이면 `MYSQL_*`, Databricks 이면 `DATABRICKS_*`이다. 사전은 언제나 제 연결로 읽으므로 두 벌이 필요할 수 있다(18.5.7절).

**표 19.2.** 명령별 환경변수

| 필요한 것 | 명령 |
|---|---|
| 없음 | `ddl` · `validate --meta` · `validate --no-dict` · `suggest --meta` · `erd -s` · `erd --meta` · `export --meta` |
| `KDSM_DB_*` | `load` · `apply` · `drift` · `diagnose` · `erd` · `export` · `validate` (기본) · `suggest --from-db` |
| `MYSQL_*` | `--dbms mysql` ( `load` · `apply` · `diagnose` · `drift` · `erd` · `export` ) |
| `DATABRICKS_*` | `--dbms databricks` ( `load` · `apply` · `diagnose` · `drift` · `erd` · `export` ) |
| LLM 자격 증명 | `suggest --coin` |

비밀번호와 토큰은 어느 명령도 인자로 받지 않는다. 명령줄에 적은 값은 `ps` 출력과 셸 기록에 남기 때문이다. 값을 노출하지 않고 지정하는 방법은 20.10절 에, 변수 이름과 기본값은 참고 자료에 있다.

## 19.3 명령

아홉 하위 명령의 인자와 사용 예이다.

### 19.3.1 ddl

```
kdsm ddl 고객주문_V0.1.0_20260812.xlsx -o dbms/ddl/고객주문.sql --drop
kdsm ddl 고객_....xlsx 주문_....xlsx -o dbms/ddl     # 주제영역별로 나눈 경우
kdsm ddl 고객주문_....xlsx --dbms databricks --catalog dev_landing
```

> ! 중요
>
> `ddl`은 메타파일이 아니라 사용자가 작성한 물리 데이터 모델링 산출물(DM-DST-003 형식)을 받는다. 테이블과 컬럼의 물리 정의가 해당 산출물에 기재되어 있기 때문이다. 형식은 파일명이 아니라 시트 구성으로 판별하므로 파일 이름은 자유롭게 지어도 된다.

표 19.3. kdsm ddl 옵션

| 옵션 | 설명 |
|---|---|
| `<산출물>` | 물리 데이터 모델링 산출물. 여럿 지정할 수 있다 |
| `--seed` | 산출물 대신 패키지에 든 물리 정의서 셋을 읽는다(14.5.1절) |
| `-o`, `--output` | 산출물이 하나면 저장 경로, 여럿이면 저장 디렉터리 |
| `--drop` | `DROP TABLE IF EXISTS`를 포함한다 |
| `--no-comments` | `COMMENT ON`을 생성하지 않는다 |
| `--comment-style` | 주석에 넣을 것. `name`(한글명) 또는 `full`(한글명 - 설명, 기본) |
| `--no-foreign-keys` | `FOREIGN KEY` 제약을 생성하지 않는다 |
| `--no-schema` | `CREATE SCHEMA`를 생성하지 않는다 |
| `--schema` | 스키마명을 덮어쓴다 |
| `--dbms` | DDL을 만들 DBMS. `postgresql`(기본)·`mysql`·`databricks` |
| `--catalog` | `--dbms databricks`일 때 이름 앞에 붙일 카탈로그 |
| `--no-varchar` | Databricks의 `VARCHAR(n)`과 `CHAR(n)`을 `STRING`으로 만든다 |
| `-t`, `--table` | 생성할 테이블 물리명. 여러 번 지정할 수 있다 |

산출물을 여러 개 지정하면 주제영역별로 분할한 것으로 판단하여 파일 수만큼 SQL을 만든다. 이때 `-o`는 디렉터리이며, 생략하면 `<저장소>/document/out/ddl`에 저장한다. 판단 근거는 14.3절 에 있다.

산출물이 하나이고 출력 경로를 지정하지 않으면 SQL이 표준 출력으로 전달되므로 다른 명령에 연결할 수 있다.

```
kdsm ddl 고객주문_V0.1.0_20260812.xlsx ¦ psql -d kdsm
```

경로를 지정하면 요약만 표준 오류로 출력한다. 표준 출력에 함께 출력하면 파이프가 오염되기 때문이다.

### 19.3.2 drift

```
kdsm drift -s 고객주문_V0.1.0_20260812.xlsx
kdsm drift -s 데이터표준_....xlsx -s 모델관리_....xlsx --schema sales --strict
kdsm drift -s 고객주문_....xlsx --dbms databricks --catalog dev_landing
```

물리 데이터 모델링 산출물과 운영 데이터베이스의 카탈로그를 읽어 테이블·컬럼·타입·제약 네 가지를 대조한다. 접속 정보는 대상 DBMS의 접두어에서 읽는다. PostgreSQL은 `KDSM_DB_*`이다.

표 19.4. kdsm drift 옵션

| 옵션 | 설명 |
|---|---|
| `-s`, `--spec` | 물리 데이터 모델링 산출물 경로. 여러 번 지정할 수 있다 |
| `--schema` | 정의서의 스키마를 이 이름으로 바꿔 대조하고 카탈로그도 이곳을 읽는다 |
| `--dbms` | 대조할 DBMS. `postgresql`(기본)·`mysql`·`databricks` |
| `--catalog` | `--dbms databricks`일 때 기본 카탈로그 |
| `--env-prefix` | 접속 정보의 환경변수 접두어 (기본 `KDSM_DB_`) |
| `--host` `--port` `--database` `--user` | 환경변수를 덮어쓴다 |
| `--strict` | 차이가 하나라도 있으면 종료 코드 1로 끝낸다 |
| `-q`, `--quiet` | 한 줄 요약만 출력한다 |

차이가 있어도 기본은 정상 종료다. 어느 쪽이 옳은지는 시점이 정하는 문제이므로 프로그램이 판정하지 않는다. 판단 근거는 14.9절 에 있다.

### 19.3.3 validate

```
kdsm validate app/deliverable/DM-DST-00{1,2,3}_*.xlsx --name-prefix AP_

# KDSM 자신을 구축할 때. 시스템 관리자 영역이다
kdsm validate --meta dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx \
  --logical dbms/meta/meta_data/kdsm_logical_meta_20260828.xlsx \
  -P dbms/meta/meta_data/kdsm_physical_meta_20260828.xlsx \
  dbms/meta/meta_table/DM-LIB-00{4,5,6}_*.xlsx
```

검사 대상은 위치 인수로 준다. 종류는 시트 구성으로 판별하므로 이름과 순서는 무관하다. 산출물을 여럿 주면 서로를 참조하여 문서 간에 걸친 외래키도 판정한다.

표 19.5. kdsm validate 옵션

| 옵션 | 설명 |
|---|---|
| `<산출물>` | 검사할 DM-DST 산출물. 여럿 줄 수 있다 |

| 옵션 | 설명 |
|---|---|
| `--meta` | 사전을 읽을 메타파일. 생략하면 DB의 `kdsm_meta`에서 읽는다 |
| `--no-dict` | 사전 없이 산출물 안에서의 무결성만 검사한다 |
| `--from-db` | 산출물 없이 DB에 적재된 사전 자체를 검사한다 |
| `--host` `--port` `--database` `--schema` `--user` | 사전을 읽을 접속 정보 |
| `--env-prefix` | 접속 환경변수의 접두어 (기본 `KDSM_DB_`) |
| `--logical` | 논리 모델 메타파일 경로 (`MT_ENTTY`·`MT_ATRB` 시트) |
| `--logical-from-db` | 논리 모델을 DB의 `MT_ENTTY`·`MT_ATRB`에서 읽는다 |
| `-P`, `--physical` | 물리 모델 메타파일 경로 (`MT_DB`·`MT_TBL`·`MT_COL` 시트) |
| `--name-prefix` | 엔터티 영문명에 덧붙는 접두어 (기본 `MT_`) |
| `--strict` | 경고도 실패로 판정한다 |
| `-q`, `--quiet` | 한 줄 요약만 출력한다 |

사전은 기본적으로 DB에서 읽는다. 산출물을 하나도 주지 않고 `--meta`만 주면 그 메타파일 자체를, `--from-db`를 주면 DB에 적재된 사전 자체를 검사한다(13.2절). `--from-db`는 사전이 대상이자 기준이므로 산출물·`--meta`·`--no-dict`와 함께 지정할 수 없다.

논리 산출물과 물리 산출물을 함께 위치 인수로 지정하면 두 문서를 대조한다. 동일한 사실을 두 문서에 각각 기재하는 과정에서 불일치가 발생하므로, 메타파일을 생성하기 전에 이를 검출하기 위한 것이다(13.3.8절). `--logical`은 산출물이 아니라 논리 모델 메타파일을 받는 옵션이다.

`-P`는 물리 모델 메타파일을 받는다. 정의서용 규칙을 그대로 적용하고, `--logical`을 함께 지정하면 관계엔터티이름이 논리 모델에 존재하는지까지 확인한다(13.3.7절).

`--name-prefix`는 엔터티 영문명이 `접두어 + 표준용어 물리명` 형식인지 확인할 때 사용한다. 접두어를 사용하지 않는 경우에는 빈 문자열을 지정한다(13.3.9절).

산출물을 열 때 파일명과 표지의 문서번호·버전·작성일을 함께 대조한다. 어긋난 항목은 INFO로 기록하며 종료 코드에는 반영하지 않는다(13.3.11절).

```
검사 699건 · 오류 0건 · 경고 0건
```

## 19.3.4 load

```
kdsm load 사전메타파일.xlsx \
  --actor choonghyunryu -s DM-DST-003_물리데이터모델_....xlsx
```

표 19.6. kdsm load 옵션

| 옵션 | 설명 |
|---|---|
| `<메타파일>` | 적재할 메타파일 |

| 옵션 | 설명 |
|---|---|
| `--seed` | 메타파일 대신 패키지에 든 초기 데이터 세 벌을 넣는다(15.4.3절) |
| `--actor` | `CREATR`·`MDFR`에 기록할 작업자명 |
| `--sheet` | 적재할 시트명. 여러 번 지정할 수 있다 |
| `-s`, `--spec` | 산출물 경로. NOT NULL·길이까지 확인 |
| `--dbms` | 넣을 DBMS. `postgresql`(기본)·`mysql`·`databricks` |
| `--catalog` | `--dbms databricks`일 때 대상 카탈로그 |
| `--schema` | 대상 스키마명. 세 DBMS 모두에 적용된다 |
| `--host` `--port` `--database` `--user` | PostgreSQL일 때의 접속 정보 |
| `--env-prefix` | 접속 환경변수의 접두어 (기본 `KDSM_DB_`) |
| `--no-validate` | 적재 전 검증을 생략한다 |
| `-n`, `--dry-run` | 읽기와 검증까지만 수행하고 적재하지 않는다 |

접속 정보는 생략하면 `--dbms`가 가리키는 DBMS의 환경변수에서 읽는다. PostgreSQL은 `KDSM_DB_*`, MySQL은 `MYSQL_*`, Databricks는 `DATABRICKS_*`이다. 비밀번호와 토큰은 인수로 받지 않는다.

KDSM을 세우는 초기 단계의 명령이다. 프로젝트가 작성한 산출물을 넣는 것은 `apply`가 맡는다(15장).

### 19.3.5 apply

```
kdsm apply app/deliverable/DM-DST-00{1,2,3}_*.xlsx --actor choonghyunryu --name-prefix AP_
```

표 19.7. kdsm apply 옵션

| 옵션 | 설명 |
|---|---|
| `<산출물>` | 넣을 DM-DST 산출물. 여럿 지정할 수 있다 |
| `--actor` | `CREATR`·`MDFR`에 기록할 작업자명 |
| `--dbms` | 넣을 DBMS. `postgresql`(기본)·`mysql`·`databricks` |
| `--catalog` | `--dbms databricks`일 때 대상 카탈로그 |
| `--schema` | 대상 스키마명. 세 DBMS 모두에 적용된다 |
| `--host` `--port` `--database` `--user` | PostgreSQL일 때의 접속 정보 |
| `--env-prefix` | 접속 환경변수의 접두어 (기본 `KDSM_DB_`) |
| `--name-prefix` | 엔터티 영문명에 덧붙는 접두어. 기본은 없음 |
| `--no-validate` | 넣기 전 검증을 생략한다 |

| 옵션 | 설명 |
|---|---|
| `-n`, `--dry-run` | 읽기와 검증까지만 수행하고 넣지 않는다 |

PostgreSQL과 MySQL에서는 3종이 한 트랜잭션이다. Databricks에는 트랜잭션이 없으므로 중간에 실패하면 앞의 것이 남는다(15.3.1절). 적재 직전에 다시 검증하며, 위반이 있으면 아무것도 적재하지 않는다. 메타파일은 사전으로 받지 않는다. 적재해도 되는지는 적재 대상 테이블만 판정할 수 있기 때문이다(15.5절).

### 19.3.6 suggest

```
kdsm suggest 고객여신금액 상품가격 --meta dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx
kdsm suggest 고객여신금액 상품가격 --from-db
```

`--meta`와 `--from-db` 중 하나를 반드시 지정해야 한다. 두 옵션을 함께 지정하면 거부한다. 사전의 출처를 알 수 없는 상태로 결과를 확인하는 일이 없어야 하기 때문이다(12.1절).

표 19.8. kdsm suggest 옵션

| 옵션 | 설명 |
|---|---|
| `<용어명>` | 후보를 받을 한글 용어명. 여럿 지정할 수 있다 |
| `--meta` | 메타파일 경로 |
| `--from-db` | 메타파일 대신 DB의 `kdsm_meta` 테이블에서 읽는다 |
| `--schema` | `--from-db` 일 때 대상 스키마명 |
| `-l`, `--limit` | 용어당 후보 수 (기본 3) |
| `--coin` | 미등록 단어의 표준단어 등재 초안을 LLM으로 제안 (화면 출력만) |
| `--llm-provider` | `anthropic`(기본) 또는 `bedrock`. 생략하면 `KDSM_LLM_PROVIDER` |

금지어(폐기 단어·이음동의어·금칙어)가 포함되어 있으면 대체어로 변경한 후보를 함께 제시한다. 이름이 변경된 후보는 `이름:` 행에 원래 이름을 함께 기재한다(12.3절).

`--coin`은 선택 기능이다. `uv sync --extra llm`으로 패키지를 설치하고 `KDSM_LLM_API_KEY`를 설정해야 사용할 수 있다. `--from-db`와 함께 사용한다면 `uv sync --extra db --extra llm`과 같이 한 번에 지정한다. 이 기능을 사용하지 않으면 추가 의존성이 필요하지 않으며, 제안을 생성하지 못하더라도 종료 코드는 변경되지 않는다(12.4.3절).

AWS Bedrock을 사용하려면 `uv sync --extra bedrock`으로 설치하고 `AWS_ACCESS_KEY_ID`·`AWS_SECRET_ACCESS_KEY`를 설정한다. 모델 이름 체계가 다르므로 12.4.3.1절 를 함께 참고한다.

```
kdsm suggest 메타데이터마트이름 --meta ... --coin --llm-provider bedrock
```

```
# 계약날짜
  *  120.0  CTRT_YMD  [연월일C8]  Contract Date
     이름: 계약날짜 → 계약일자
```

```
구성: 계약 + 일자
· '날짜'는 이음동의어. 일자(YMD)로 쓴다
```

### 19.3.7 diagnose

```
kdsm diagnose --schema sales --schema hr -o 진단.xlsx --qmd 진단.qmd
```

표 19.9. kdsm diagnose 옵션

| 옵션 | 설명 |
|---|---|
| `--dbms` | 진단 대상 DBMS. `postgresql`(기본)·`mysql`·`databricks` 중에서 |
| `--render` | `--qmd` 문서를 HTML·PDF 로 조판한다. Quarto 가 있어야 한다 |
| `--catalog` | Databricks 기본 카탈로그. 생략하면 `<접두어>CATALOG` |
| `--env-prefix` | 진단 대상 접속 정보의 환경변수 접두어. 생략하면 `KDSM_DB_` |
| `--host` | 접속할 서버. 생략하면 `<접두어>HOST` |
| `--port` | 포트. 생략하면 `<접두어>PORT` |
| `--database` | 데이터베이스명. 생략하면 `<접두어>NAME` |
| `--user` | 접속 사용자. 생략하면 `<접두어>USER` |
| `--schema` | 진단할 스키마. 여러 번 지정할 수 있다. 생략하면 `<접두어>SCHEMA` |
| `--meta` | 사전을 메타파일에서 읽는다. 생략하면 DB에서 읽는다 |
| `--dict-env-prefix` | 사전이 있는 DB의 환경변수 접두어. 생략하면 `KDSM_DB_` |
| `--dict-schema` | 사전이 있는 스키마. 생략하면 사전 쪽 `<접두어>SCHEMA`, 그것도 없으면 `kdsm_meta` |
| `--no-dict` | 사전 없이 일관성만 진단한다 |
| `--top` | 이름 통계 순위표에 수록할 행 수 (기본 10) |
| `-o`, `--output` | 엑셀 리포트 저장 경로 |
| `--qmd` | Quarto 리포트 저장 경로 |
| `--quarto` | Quarto 실행 파일 경로. 생략하면 `KDSM_QUARTO_BIN`과 `PATH` |
| `-q`, `--quiet` | 한 줄 요약만 출력한다 |
| `--min-score` | 점수가 이 값보다 낮으면 실패로 판정한다 |

```
컬럼 154개 · 용어 100.0% · 단어 100.0% · 도메인 100.0% · 종합 100.0(A) · 일관성 100.0(A) · 설명 100.0%
```

접속 정보는 지정한 항목만 환경변수를 덮어쓴다. 비밀번호는 인자로 받지 않는다. 명령줄에 기재하면 `ps` 출력과 셸 기록에 남기 때문이다. `<접두어>PASSWORD`를 사용한다.

진단 대상과 표준 사전이 서로 다른 서버에 있으면 환경변수 접두어를 구분하여 사용한다. 비밀번호도 각각 별도로 지정한다(18.5.7절).

```
kdsm diagnose --env-prefix ECODI_DB_ -o 진단.xlsx
```

사전 없이 구축된 스키마는 `--no-dict`를 지정한다. 준수율 대신 일관성만 산출한다(18.3절).

```
kdsm diagnose --host db.internal --database erp --schema legacy --no-dict -o 진단.xlsx
kdsm diagnose --schema legacy --qmd 진단.qmd --render        # HTML·PDF 까지
```

리포트 경로는 표준 오류로 출력한다. 표준 출력에는 진단 결과만 전달해야 파이프가 오염되지 않는다(18장).

### 19.3.8 erd

```
kdsm erd -s 고객_V0.1.0_20260812.xlsx -s 주문_V0.1.0_20260812.xlsx   # 산출물에서
kdsm erd -f png                      # 적재된 메타테이블에서, 그림으로
kdsm erd --from-catalog --schema shop # DBMS 카탈로그에서
```

`-s`로 데이터 모델링 산출물을 지정하면 해당 파일에서, 지정하지 않으면 적재된 메타테이블에서 ERD를 작성한다. `--from-catalog`를 지정하면 DBMS 카탈로그에서 읽어 물리 ERD만 작성한다(16.3.1절). 논리 모델인지 물리 모델인지는 파일명이 아니라 시트 구성으로 판별한다. 파일이 여러 개이면 주제영역별로 분할한 것으로 판단하여 파일마다 한 장씩 작성하고, 전체를 통합한 한 장을 추가로 작성한다. 판단 근거는 16장 에 있다.

**표 19.10.** kdsm erd 주요 옵션

| 옵션 | 설명 |
|---|---|
| `-s`, `--spec` | 데이터 모델링 산출물 경로. 여러 번 지정할 수 있다 |
| `--meta` | 데이터베이스 대신 메타파일에서 읽는다. 여러 번 지정할 수 있다 |
| `-m`, `--model` | `logical`·`physical`·`all` (기본 `all`) |
| `-f`, `--format` | `mmd`·`md`·`png`·`svg`·`pdf`·`all` (기본 `mmd,md`) |
| `-o`, `--output` | ERD를 저장할 디렉터리 |
| `--areas` | 영역 구분 TOML. 메타테이블에서 읽을 때만 사용한다 |
| `--no-pdf-fit` | PDF를 인쇄 규격에 맞춘다. 기본은 도면 크기에 맞춘 한 페이지 |
| `--pdf-merge` | 생성한 PDF를 한 파일로 병합한다. 파일명을 생략하면 `ERD.pdf` |
| `--from-catalog` | DBMS 카탈로그에서 읽는다. 물리 ERD만 작성한다 |
| `--dbms` | 읽을 DBMS. `postgresql`(기본)·`mysql`·`databricks` 중에서 |
| `--catalog` | `--dbms databricks`일 때 카탈로그 |
| `--schema` | 대상 스키마명. `--from-catalog` 일 때는 여러 번 지정할 수 있다 |
| `--no-names` | `--from-catalog` 일 때 주석에서 한글명을 가져오지 않는다 |

나머지 옵션은 표 16.7 에 있다.

### 19.3.9 export

```
kdsm export -o document/out          # DM-DST 산출물 셋
kdsm export --target standard        # 데이터 표준 정의 산출물(DM-DST-001)만
kdsm export --dbms databricks --catalog dev_landing --schema kdsm_meta
kdsm export --from-catalog --dbms databricks --catalog dev_landing
```

적재된 메타테이블을 읽어 DM-DST 산출물을 내보낸다. 메타테이블은 `--dbms`가 가리키는 곳에서 읽으며 PostgreSQL·MySQL·Databricks를 모두 지원한다. `--from-catalog`를 지정하면 메타테이블이 아니라 DBMS 카탈로그에서 읽으며, 이때에는 물리 데이터 모델만 내보낸다(17.6절).

**표 19.11.** kdsm export 주요 옵션

| 옵션 | 설명 |
|---|---|
| `--target` | 내보낼 산출물. `standard` · `logical` · `physical` · `all` (기본 `all`) |
| `-o`, `--output` | 산출물을 저장할 디렉터리 |
| `--template-dir` | 템플릿 디렉터리 |
| `--meta` | DB 대신 메타파일에서 읽는다. 여러 번 지정할 수 있다 |
| `--from-catalog` | DBMS 카탈로그에서 읽는다. 물리 모델만 내보낸다 |
| `--dbms` | 읽을 DBMS (기본 `postgresql`) |
| `--catalog` | `--dbms databricks`일 때 카탈로그 |

나머지 옵션은 표 17.7 에 있다. 생성 방식의 판단 근거는 17장 에 있다.

생성된 파일은 버전 `0.0.0`, 작성자 공란, 생성 당일 날짜로 기록된다. 사람이 작성한 버전의 번호와 이름을 승계하지 않는다는 뜻이며, 파일명의 버전과 날짜도 함께 변경된다. 이름에서 `템플릿`은 제외한다.

```
DM-DST-001_데이터표준정의산출물_V0.0.0_20260812.xlsx
```

## 19.4 CI 연계

검증만 연계하는 것이 가장 간단하다. 산출물과 메타파일을 한 번에 전달한다.

```
#!/usr/bin/env bash
set -euo pipefail

kdsm validate \
  DM-DST-00{1,2,3}_*.xlsx \
  --meta     dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx \
  --logical  dbms/meta/meta_data/kdsm_logical_meta_20260828.xlsx \
  --physical dbms/meta/meta_data/kdsm_physical_meta_20260828.xlsx \
  -q

kdsm diagnose --schema kdsm_meta --min-score 95 -q
```

검증이 실패하면 `set -e`에 의하여 그 지점에서 중단된다.

DDL 생성과 적재까지 함께 수행하려면 `dbms/rebuild.sh`를 사용한다. 위 검증을 가장 앞에 포함하고 있다(15.6절).

```
./dbms/rebuild.sh --dry-run    # CI에서는 여기까지만
```

# IV

# 4부 · 실무

20 환경 준비 . . . . . 221
21 자주 발생하는 문제와 예방 . . . . . 247
22 작업 흐름 . . . . . 255

CHAPTER 20

# 환경 준비

「환경 준비」에는 두 가지 뜻이 담긴다. 이 장은 둘을 모두 다룬다.

**KDSM 라이브러리(Python 패키지) 환경을 준비한다**
: 명령을 실행할 도구를 설치하고 접속 정보를 지정한다. 도표의 1·2 단계이며 1절부터 7절까지가 이에 해당한다.

**KDSM 시스템을 준비한다**
: 사용자의 데이터베이스에 KDSM이 쓸 사용자와 스키마를 만들고, 메타테이블 12개를 세우고, 표준 사전을 그 안에 넣는다. 도표의 3·4·5 단계이며 8절이 이에 해당한다(20.11절).

앞의 것은 장비마다 한 번, 뒤의 것은 데이터베이스마다 한 번이다. 앞의 것만 마쳐도 파일을 읽는 작업은 모두 수행할 수 있다. 뒤의 것까지 마쳐야 데이터베이스를 읽고 쓰는 기능이 동작한다.

macOS·Linux·Windows 를 모두 다룬다. 운영체제에 따라 달라지는 부분은 20.3절 에 모아 두었고, 각 도구의 설치 명령은 절마다 운영체제별로 나누어 기술한다.

도구를 전부 설치할 필요는 없다. 필수는 uv 와 `kdsm` 패키지뿐이며, 나머지는 수행하려는 작업에 따라 추가된다. 파일만 읽는 작업은 이 둘로 수행할 수 있다. 데이터베이스에서 읽는 작업에는 선택 의존성 `kdsm[db]` 가 추가된다.

## 20.1 처음 한 번의 순서

두 가지를 합하면 다섯 단계이다. 차례가 정해져 있으므로 먼저 전체를 훑는다.

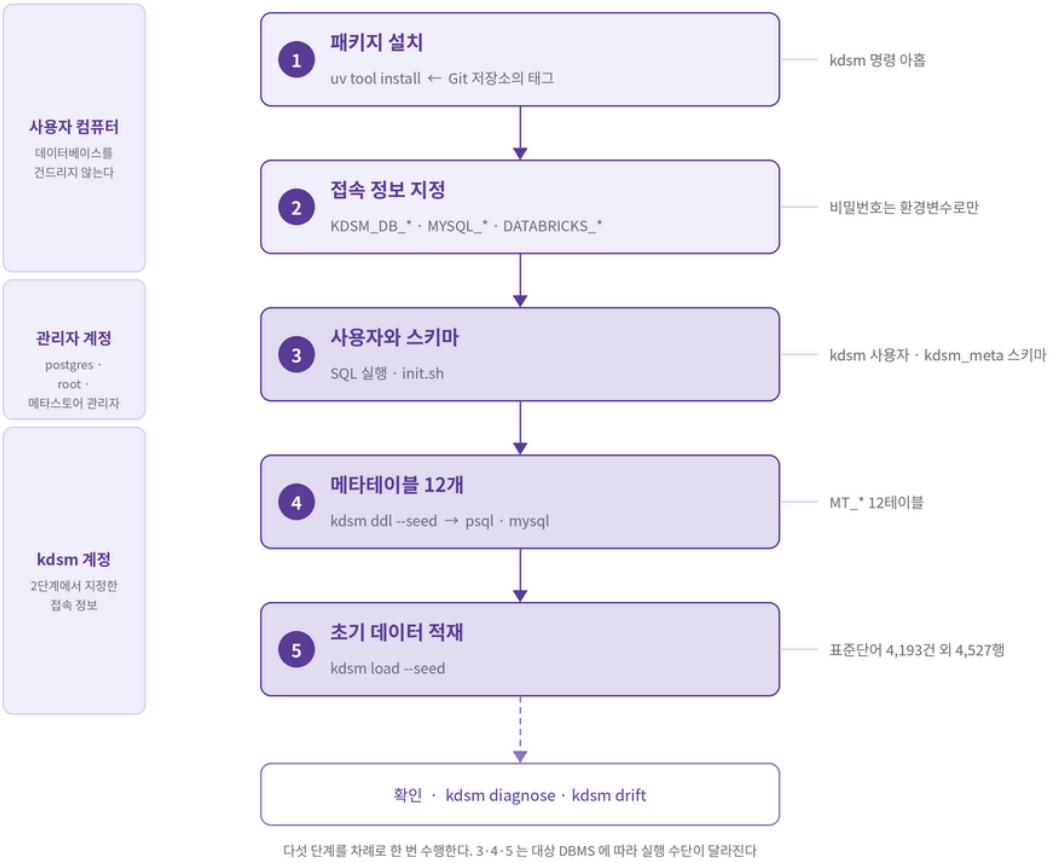


그림 20.1. 환경 준비 순서

표 20.1. 환경 준비의 다섯 단계

| 단계 | 하는 일 | 실행할 것 | 근거 |
|---|---|---|---|
| 1 | 패키지를 설치한다 | Git 저장소에서 `uv tool install` | 20.4.2절 |
| 2 | 접속 정보를 지정한다 | `KDSM_DB_*` · `MYSQL_*` · `DATABRICKS_*` | 20.9절 |
| 3 | 사용자와 스키마를 만든다 | SQL 을 실행한다. 저장소가 있으면 `dbms/script/init.sh` | 20.11.1절 |
| 4 | 메타테이블 12개를 만든다 | `kdsm ddl --seed` 로 SQL을 만들어 적용한다 | 14.5.1절 |
| 5 | 초기 데이터를 적재한다 | `kdsm load --seed` | 15.4.3절 |

1단계와 2단계는 데이터베이스를 건드리지 않는다. 이 둘까지 마치면 파일만 읽는 작업은 모두 수행할 수 있다. 산출물 검증과 DDL 생성, 산출물에서 그리는 ERD가 그러하다(표 20.2).

3단계부터가 데이터베이스에 손을 대는 자리이다. 다섯 단계는 모두 차례로 한 번 수행하며, 이후의 일상 작업에서는 다시 수행하지 않는다. 3·4·5 를 여러 번 되풀이하는 것이 아니다. 대상 DBMS 를 무

엇으로 정하는지에 따라 실행 수단이 갈릴 뿐이다. 3단계의 SQL, 4단계에서 SQL을 적용하는 클라이언트, 5단계의 `--dbms` 가 그것이다.

> ! 3단계와 4단계의 계정이 다르다
>
> 사용자와 스키마를 만드는 권한과 테이블을 만드는 권한은 층위가 다르다. 3단계는 그 권한을 가진 관리자로 접속한다. PostgreSQL은 보통 `postgres`, MySQL은 `root`이며, Databricks는 카탈로그를 만들 수 있는 메타스토어 관리자이다.
>
> 4단계부터는 3단계에서 만든 `kdsm` 계정으로 수행한다. 한 계정으로 묶으면 일상 작업에 쓰는 계정에 필요 이상의 권한을 주게 된다.

> i 저장소를 내려받지 않아도 3·4·5를 수행할 수 있다
>
> 4단계와 5단계가 읽는 정의서와 메타파일은 `kdsm` 패키지에 함께 들어 있다. `--seed`가 그것을 가리킨다(11.3.3절). 따라서 패키지를 설치하기만 하면 그 두 단계를 수행할 수 있다.
>
> 패키지 밖에 있는 것이 둘이다. 3단계에서 사용자와 스키마를 만드는 SQL, 그리고 4단계에서 SQL을 적용하는 수단이다. 앞의 것은 대상 DBMS마다 문장이 다르고, 뒤의 것은 `kdsm ddl`이 파일을 만들 뿐 실행하지 않기 때문이다. 저장소를 내려받았다면 `dbms/script/init.sh`가 앞의 것을 맡고, 그러지 않았다면 20.11.1절 의 SQL을 관리자에게 넘긴다. 뒤의 것은 `psql`·`mysql` 클라이언트가 맡는다.

> 다시 실행해도 안전하다
>
> 3단계는 이미 있는 사용자의 비밀번호를 덮어쓰지 않는다. 5단계는 기본키가 있으면 갱신하므로(UPSERT) 같은 파일을 여러 번 넣어도 겹쳐 쌓이지 않는다(15.3.1절).
>
> 중간에서 멈추었을 때에는 멈춘 단계부터 다시 시작한다. 5단계는 세 벌을 차례로 넣으므로 어느 영역이 들어가지 못했는지 알린다.

DBMS별 명령 전문은 22.4.1절 에 있다. 저장소를 내려받아 세우는 경우에는 3단계 이후를 `dbms/rebuild.sh` 하나로 대신할 수 있다(15.6절). 5단계까지 마친 뒤의 일상 작업 순서는 22장 에 있다.

## 20.2 작업별 필요 도구

`kdsm` 은 uv 로 설치한 패키지를 뜻하고, 대괄호 안은 선택 의존성이다(20.4.2.5절). uv 자체는 모든 줄에 필요하므로 적지 않았다.

표 20.2. 작업별 필요 도구

| 수행하려는 작업 | 필요한 것 |
|---|---|
| 산출물 검증 `kdsm validate --meta` | `kdsm` |
| 산출물 검증 `kdsm validate` (사전을 DB에서) | `kdsm[db]` |
| DDL 생성 `kdsm ddl` | `kdsm` |
| ERD 생성 `kdsm erd -s` · `kdsm erd --meta` | `kdsm` |
| ERD 생성 `kdsm erd` (메타테이블에서) | `kdsm[<대상 DBMS>]` |

| 수행하려는 작업 | 필요한 것 |
|---|---|
| 산출물 자동 생성 `kdsm export --meta` | `kdsm` |
| 산출물 자동 생성 `kdsm export` (메타테이블에서) | `kdsm[<대상 DBMS>]` |
| DB 적재 `kdsm load` | `kdsm[<대상 DBMS>]` + 그 클라이언트 |
| 산출물 적재 `kdsm apply` | `kdsm[<대상 DBMS>]` |
| 표준화 진단 `kdsm diagnose` | `kdsm[db]` |
| 정의서와 DB 대조 `kdsm drift` | `kdsm[<대상 DBMS>]` |
| Databricks 적재·조회·진단 `--dbms databricks` | `kdsm[databricks]` + OAuth M2M 또는 토큰 |
| 위에서 사전을 DB에서 읽는 경우 | 위와 동일 + `db` |
| MySQL 적재·조회·진단 `--dbms mysql` | `kdsm[mysql]` |
| 카탈로그 ERD·산출물 `--from-catalog` | `kdsm[<대상 DBMS>]` |
| 표준단어 제안 `--coin` | `kdsm[llm]` 또는 `kdsm[bedrock]` + API 키 |
| ERD 이미지 생성 `kdsm erd -f png` | `kdsm` + Node.js + mermaid-cli + 브라우저 |
| ERD PDF 합본 `--pdf-merge` | 위와 동일 + `pdf` |
| 진단 보고서 조판 `--render` | `kdsm[db]` + Quarto + 한글 폰트 |
| 저장소의 `rebuild.sh` · `refresh-erd.sh` | 위에 더해 저장소를 내려받는다 |
| 이 매뉴얼 빌드 | Quarto + 저장소 |

**i 설치 여부 확인**

현재 설치되어 있는 도구는 다음 명령으로 확인한다. 설치되어 있지 않으면 `command not found`가 출력된다.

```
uv --version          # 필수
psql --version        # DB 적재
node --version        # ERD 그림
mmdc --version        # ERD 그림
quarto --version      # 매뉴얼 빌드, 진단 보고서 조판
```

PowerShell 에서도 같은 명령을 사용한다. 설치되어 있지 않으면 `command not found` 대신 `용어가 cmdlet ... 인식되지 않습니다`가 출력된다.

## 20.3 운영체제별 차이

`kdsm` 라이브러리는 macOS·Linux·Windows 에서 동일하게 동작한다. 파이썬으로 작성되어 있고 경로와 파일 입출력은 표준 라이브러리가 처리하므로, 명령과 옵션은 세 환경이 같다.

차이가 있는 곳은 세 가지이다.

표 20.3. 운영체제별로 달라지는 것

| 구분 | macOS · Linux | Windows |
|---|---|---|
| 설치 도구 | Homebrew · apt · dnf | winget · Scoop · 설치 파일 |
| 셸 | bash · zsh | PowerShell (권장) · cmd |
| 환경변수 지정 | `export NAME=값` | `$env:NAME = '값'` |

**! 셸 스크립트 2종은 bash 가 필요하다**

`dbms/rebuild.sh` 와 `document/manual/refresh-erd.sh` 는 bash 스크립트이다. Windows 의 PowerShell 이나 cmd 에서는 실행되지 않는다. 다음 중 하나를 사용한다.

- Git for Windows 에 포함된 Git Bash
- WSL(Windows Subsystem for Linux)

두 스크립트는 여러 명령을 순서대로 실행하는 편의 도구이므로, 없어도 각 명령을 직접 실행하면 같은 결과를 얻는다. `rebuild.sh` 가 수행하는 절차는 15.6절 에 단계별로 기술되어 있다.

**⚠ Windows 콘솔의 한글 출력**

`kdsm` 의 출력에는 한글과 `·` `–` 같은 기호가 포함된다. 콘솔의 코드 페이지가 UTF-8 이 아니면 이러한 문자에서 `UnicodeEncodeError` 가 발생할 수 있다.

Windows Terminal 과 PowerShell 7 이상을 사용하면 대개 문제가 없다. 그래도 오류가 발생한다면 파이썬을 UTF-8 모드로 실행한다.

```
$env:PYTHONUTF8 = '1'
chcp 65001                    # cmd 를 사용하는 경우
```

엑셀 리포트와 Quarto 리포트는 파일로 저장하므로 콘솔의 코드 페이지와 무관하다.

## 20.4 1. uv와 kdsm 설치 (필수)

`kdsm`은 파이썬으로 작성한 패키지이며 `kdsm` 이라는 이름의 명령을 함께 제공한다. 배포한 패키지를 설치하면 그 명령을 바로 쓸 수 있다. 파이썬과 가상환경은 uv가 관리한다.

### 20.4.1 uv 설치

macOS · Linux

```
curl -LsSf https://astral.sh/uv/install.sh ¦ sh   # 공식 설치 스크립트
wget -qO- https://astral.sh/uv/install.sh ¦ sh    # curl 이 없는 경우

brew install uv                                   # macOS. Homebrew
sudo port install uv                              # macOS. MacPorts
```

Linux 배포판의 패키지 관리자에는 등록되어 있지 않은 경우가 많으므로 설치 스크립트를 사용한다. 스크립트는 `~/.local/bin` 에 설치하므로 관리자 권한이 필요하지 않다.

Windows

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"

winget install --id=astral-sh.uv -e              # 또는 winget
scoop install main/uv                            # 또는 Scoop
```

설치 스크립트를 사용하면 `%USERPROFILE%\.local\bin` 에 설치된다. 설치 후 `PATH` 가 적용되도록 터미널을 다시 연다.

파이썬은 세 환경 모두 별도로 설치하지 않아도 된다. uv 가 필요한 버전을 자동으로 내려받는다.

> ! 파이썬 3.11 이상이어야 한다
>
> `StrEnum` 을 사용하기 때문이다. 패키지의 `requires-python` 이 `>=3.11` 로 명시되어 있어, uv가 조건에 맞는 버전을 선택하여 사용한다.

## 20.4.2 kdsm 설치

Git 저장소에서 바로 설치한다. 별도의 패키지 서버를 두지 않고, 저장소의 태그가 곧 판이 된다. (설치는 디플래닉스에 문의 요망)

```
uv tool install 'kdsm[db] @ git+https://<리파지토리 URL>/data-standardization-
application.git@v0.11.0#subdirectory=library'
```

한 줄이 길어 보이지만 조각은 넷뿐이다.

표 20.4. 설치 주소의 구성

| 조각 | 뜻 |
|---|---|
| `kdsm[db]` | 패키지 이름과 선택 의존성(20.4.2.5절) |
| `git+https://...` | 저장소 주소 |
| `@v0.11.0` | 설치할 판. 태그·브랜치·커밋을 지정한다 |
| `#subdirectory=library` | 패키지가 저장소 뿌리가 아니라 `library/` 에 있다 |

`#subdirectory=library` 를 빠뜨리면 저장소 뿌리에 `pyproject.toml` 이 없다는 오류가 난다. 이 저장소는 라이브러리와 산출물과 문서를 한자리에 두므로 패키지가 하위 디렉터리에 있다.

### 20.4.2.1 설치할 판을 고른다

`@` 뒤에 무엇을 적는지로 갈린다.

표 20.5. 설치할 판을 지정하는 방법

| 적는 것 | 예 | 쓰는 때 |
|---|---|---|
| 태그 | `@v0.11.0` | 평상시. 판이 고정되어 다시 설치해도 같은 것이 온다 |
| 브랜치 | `@main` | 아직 태그를 달지 않은 최신을 볼 때 |
| 커밋 | `@a1b2c3d` | 특정 시점을 정확히 짚어야 할 때 |

> ! 태그를 지정한다
>
> `@main` 은 가리키는 대상이 계속 바뀐다. 어제 설치한 사람과 오늘 설치한 사람이 서로 다른 코드를 쓰게 되고, 문제가 생겼을 때 무엇을 쓰고 있었는지 되짚을 수 없다.
>
> 평상시에는 태그를 지정한다. 무엇이 설치되어 있는지는 아래 명령으로 확인한다.
>
> ```
> uv tool list                    # 독립 설치한 경우
> uv pip show kdsm                # 프로젝트 환경에 넣은 경우
> ```

#### 20.4.2.2 명령만 쓸 것인가, 코드에서 부를 것인가

`kdsm` 명령만 쓴다면 독립 설치가 단순하다. 전용 환경을 따로 만들어 명령을 `PATH` 에 올리므로 다른 파이썬 환경을 건드리지 않는다.

```
uv tool install 'kdsm[db] @ git+https://<리파지토리 URL>/data-standardization-
application.git@v0.11.0#subdirectory=library'
kdsm --help
```

자기 코드에서 `import kdsm` 한다면 그 프로젝트의 환경에 넣는다.

```
uv pip install 'kdsm[db] @ git+https://<리파지토리 URL>/data-standardization-
application.git@v0.11.0#subdirectory=library'
python -c "import kdsm; print(kdsm.__version__)"
```

주소가 길므로 프로젝트에서는 `pyproject.toml` 에 적어 두는 편이 낫다. 판을 올릴 때 그 한 줄만 고치면 된다.

```
[project]
dependencies = [
    "kdsm[db] @ git+https://<리파지토리 URL>/data-standardization-
application.git@v0.11.0#subdirectory=library",
]
```

#### 20.4.2.3 접근 권한

저장소가 비공개이면 인증이 필요하다. Git 이 이미 인증되어 있으면 위의 명령이 그대로 동작한다. 그렇지 않다면 둘 중 하나를 갖춘다.

```
gh auth login                   # GitHub CLI 로 자격을 저장한다
git config --global credential.helper store
```

SSH 키를 쓰는 조직이라면 주소를 `git+ssh://git@github.com/...` 로 바꾼다.

> ⚠ 토큰을 주소에 적지 않는다
>
> `https://<토큰>@github.com/...` 형태로 적으면 그 값이 셸 기록과 `pyproject.toml` 에 남는다. `ps` 출력에도 나타난다(20.10.1절). 자격 증명은 Git 이 관리하게 두고 주소에는 넣지 않는다.

#### 20.4.2.4 판을 올린다

새 태그가 붙으면 주소의 판만 바꾸어 다시 설치한다. `--force` 는 이미 있는 독립 설치를 다시 만든다는 뜻이다.

```
uv tool install --force 'kdsm[db] @ git+https://<리파지토리 URL>/data-standardization-application.git@v0.9.1#subdirectory=library'
```

같은 태그를 다시 설치하면 캐시에 있는 것을 쓴다. 태그를 옮겨 달았거나 브랜치를 지정한 경우에는 캐시를 지운다.

```
uv cache clean kdsm
```

#### 20.4.2.5 선택 의존성

패키지 이름 뒤의 대괄호에 적는다. 수행하려는 작업에 필요한 것만 고른다(표 20.2).

표 20.6. 선택 의존성

| extra | 활성화되는 기능 |
|---|---|
| `db` | `kdsm load`, `--from-db`, PostgreSQL 진단 |
| `llm` | Claude API를 이용한 `--coin` 제안 |
| `bedrock` | AWS Bedrock을 이용한 `--coin` 제안 |
| `databricks` | `--dbms databricks` 적재·메타테이블 조회·카탈로그 읽기·진단 |
| `mysql` | `--dbms mysql` 적재·메타테이블 조회·카탈로그 읽기·진단 |
| `pdf` | `kdsm erd --pdf-merge` PDF 합본 |

여럿이 필요하면 쉼표로 잇는다. 하나로 묶는 이름은 따로 두지 않았으므로 필요한 것을 열거한다.

```
kdsm[db,mysql,databricks]                  # 세 DBMS 를 모두 다룰 때
kdsm[db,mysql,databricks,llm,bedrock,pdf]  # 전부
```

> ⚠ 주소 전체를 따옴표로 감싼다
>
> `zsh` 와 `bash` 가 대괄호를 파일명 패턴으로 해석하고, `#` 뒤를 주석으로 본다. 따옴표 없이 실행하면 `#subdirectory=library` 가 함께 사라져 엉뚱한 오류가 난다.
>
> ```
> uv tool install kdsm[db] @ git+https://...#subdirectory=library    # 잘못
> uv tool install 'kdsm[db] @ git+https://...#subdirectory=library'  # 이렇게 적는다
> ```

### 20.4.3 저장소에서 개발판을 쓰는 경우

KDSM 자체를 고치거나 배포 전의 판을 시험할 때이다. 저장소를 내려받아 `library/` 에서 동기화한다. 이 경우에만 `uv sync` 를 사용한다.

```
cd library
uv sync --all-extras
uv run kdsm --help
```

> **`uv run`을 앞에 붙이는 습관**
>
> `uv sync`는 `library/.venv`에 설치한다. 해당 환경을 활성화하지 않으면 `kdsm`이 `PATH`에 등록되지 않는다. `uv run kdsm ...`으로 실행하면 uv가 해당 환경에서 실행한다.
>
> `ModuleNotFoundError: No module named 'kdsm'`이 발생하면 대부분 이 문제에 해당한다.

> **`uv sync`는 요청한 것만 남긴다**
>
> `uv sync`는 기본이 정확 동기화이다. 두 번에 나누어 실행하면 먼저 설치한 항목이 제거된다. 사용할 extra를 한 줄에 모두 지정하거나 `--all-extras`를 지정한다. 자세한 내용은 11장 에 있다.
>
> 설치한 패키지에는 이 문제가 없다. `uv pip install`과 `pip install`은 이미 설치한 것을 지우지 않는다.

## 20.5 2. PostgreSQL 클라이언트 (적재할 때)

`dbms/rebuild.sh`가 `psql`을 호출한다. 서버는 필요하지 않으며 클라이언트만 있으면 된다. `kdsm load`는 psycopg 로 직접 접속하므로 `psql` 이 없어도 동작한다.

macOS

```
brew install libpq                      # 클라이언트만
brew link --force libpq                 # PATH에 잡아 준다
```

Linux

```
sudo apt install postgresql-client      # Debian·Ubuntu
sudo dnf install postgresql             # Fedora·RHEL 계열
sudo zypper install postgresql          # openSUSE
```

Windows

```
winget install --id PostgreSQL.PostgreSQL.17 -e
scoop install postgresql                # 또는 Scoop
```

winget 으로 설치하면 서버까지 함께 설치된다. 클라이언트만 필요한 경우에는 설치 마법사에서 `Command Line Tools` 만 선택한다. 설치 경로는 기본값이 `C:\Program Files\PostgreSQL\17\bin` 이며, 이 경로가 `PATH` 에 등록되어야 `psql` 을 실행할 수 있다.

```
$env:PATH += ';C:\Program Files\PostgreSQL\17\bin'
```

> **libpq는 기본으로 PATH에 등록되지 않는다 (macOS)**
>
> Homebrew에서 `libpq`는 keg-only 패키지이다. 설치하더라도 `psql: command not found`가 그대로 출력된다. `brew link --force libpq`를 실행하거나 셸 설정에 경로를 직접 추가한다.
>
> ```
> echo 'export PATH="/opt/homebrew/opt/libpq/bin:$PATH"' >> ~/.zshrc
> ```

서버까지 필요한 경우 macOS 는 `brew install postgresql@17`, Debian·Ubuntu 는 `sudo apt install postgresql` 을 사용한다. 이미 사용 중인 DB 서버가 있다면 클라이언트만으로 충분하다.

접속 정보는 환경변수로 지정한다. 목록은 참고 자료의 환경변수 절에 있다.

```
export KDSM_DB_HOST=localhost KDSM_DB_PORT=5432 KDSM_DB_NAME=...
export KDSM_DB_USER=... KDSM_DB_PASSWORD=...
export KDSM_DB_SCHEMA=kdsm_meta
```

PowerShell 에서는 변수마다 한 줄씩 지정한다.

```
$env:KDSM_DB_HOST = 'localhost'
$env:KDSM_DB_PORT = '5432'
$env:KDSM_DB_NAME = '...'
$env:KDSM_DB_USER = '...'
$env:KDSM_DB_SCHEMA = 'kdsm_meta'
```

> ! 비밀번호는 인자로 주지 않는다
>
> 어느 명령도 비밀번호를 명령줄 인자로 받지 않는다. 명령줄에 기재하면 같은 서버에 접속한 다른 사용자의 `ps` 출력과 셸 기록에 남기 때문이다. `KDSM_DB_PASSWORD` 와 `DATABRICKS_TOKEN`, AWS 키가 모두 이에 해당한다.

확인은 다음과 같이 수행한다.

```
psql --version
psql -h "$KDSM_DB_HOST" -U "$KDSM_DB_USER" -d "$KDSM_DB_NAME" -c '\dt kdsm_meta.*'
```

스키마를 생성할 권한이 없으면 `rebuild.sh` 가 3단계에서 중단하고 관리자에게 요청할 문장을 제시한다 (15.6절).

## 20.6 3. Node.js와 mermaid-cli (ERD를 그림으로)

`kdsm erd` 는 mermaid 텍스트까지는 자체적으로 생성한다. `.mmd` 와 `.md` 만 필요한 경우에는 이 절을 건너뛰어도 된다.

PNG·SVG·PDF로 생성할 때만 mermaid-cli가 필요하다. 이미지 생성은 브라우저 엔진이 글꼴을 적용하여 문자를 배치하는 과정을 거쳐야 하므로 파이썬만으로는 수행할 수 없다.

macOS

```
brew install node
npm install -g @mermaid-js/mermaid-cli
```

Linux

```
sudo apt install nodejs npm                # Debian·Ubuntu
sudo dnf install nodejs npm                # Fedora·RHEL 계열
npm install -g @mermaid-js/mermaid-cli
```

배포판이 제공하는 Node.js 가 오래된 경우가 있다. `node --version` 으로 확인하고 조건에 미치지 못하면 nodejs.org의 배포본이나 `nvm` 을 사용한다. 전역 설치에 관리자 권한이 필요하면 `npm config set prefix ~/.npm-global` 로 설치 위치를 홈 디렉터리로 바꾸고 그 경로를 `PATH` 에 추가한다.

Windows

```
winget install --id OpenJS.NodeJS.LTS -e     # 또는 nodejs.org 의 설치 파일
npm install -g @mermaid-js/mermaid-cli
```

Node.js는 18.19 이상 또는 20 이상이어야 한다. mermaid-cli 11 계열이 요구하는 조건이다.

> ⚠ Homebrew로 mermaid-cli를 설치하지 않는다
>
> `brew install mermaid-cli` 는 브라우저 없이 본체만 설치하므로 렌더링 단계에서 브라우저를 찾지 못한다. mermaid-cli 저장소도 brew 설치를 지원하지 않는다고 밝히고 있다. 이미 이 방식으로 설치하였다면 제거한 뒤 npm 으로 다시 설치한다.
>
> ```
> brew uninstall mermaid-cli
> npm install -g @mermaid-js/mermaid-cli
> which -a mmdc          # 옛 것이 PATH에 남아 있지 않은지
> ```

### 20.6.1 브라우저

mermaid-cli는 puppeteer로 브라우저를 실행하여 이미지를 생성한다. npm으로 설치하면 대부분 브라우저까지 함께 내려받지만, 설치 스크립트가 차단되어 있거나 캐시 경로가 일치하지 않으면 내려받지 못한다. 이 경우 다음과 같이 중단된다.

```
Error: Could not find chrome-headless-shell (ver. 150.0.7871.24)
```

해결 방법은 두 가지이다.

```
npx puppeteer browsers install chrome-headless-shell   # 브라우저를 내려받는다
kdsm erd -f png --chrome                               # 또는 이미 깔린 크롬을 빌려 쓴다
```

`--chrome`을 값 없이 지정하면 `Google Chrome` → `Chromium` → `Microsoft Edge` 순으로 탐색한다. 이 방법이 더 빠르며, 내려받기가 차단된 망에서도 사용할 수 있다. 다만 puppeteer가 보장하는 조합이 아니므로 버전 차이가 크면 문제가 발생할 수 있다.

Windows 에는 Microsoft Edge 가 기본으로 설치되어 있으므로 `--chrome` 이 대개 동작한다. 컨테이너나 최소 설치 상태의 Linux 서버에는 브라우저가 없으므로 `npx puppeteer browsers install` 로 내려받는다.

자세한 내용은 16.7절 에 있다.

> ℹ 다른 경로에 설치한 경우
>
> `mmdc`를 찾는 순서는 `--mmdc` → `KDSM_MMDC_BIN` → `PATH`이다.
>
> ```
> export KDSM_MMDC_BIN=/opt/tools/mmdc
> ```
>
> ```
> $env:KDSM_MMDC_BIN = 'C:\tools\mmdc.cmd'
> ```
>
> Windows 에서 npm 전역 설치는 `mmdc.cmd` 를 생성한다. 경로를 직접 지정할 때에는 확장자까지 기재한다.

## 20.7 4. Quarto (매뉴얼을 빌드할 때)

이 매뉴얼 자체를 HTML·PDF로 빌드할 때만 필요하다. `kdsm diagnose --qmd`가 생성한 리포트를 조판할 때도 사용한다.

macOS

```
brew install --cask quarto
```

Linux

quarto.org에서 배포판에 맞는 파일을 내려받아 설치한다. Debian· Ubuntu 는 `.deb`, Fedora·RHEL 계열은 `.rpm`, 그 밖에는 `.tar.gz` 를 사용한다.

```
sudo dpkg -i quarto-<버전>-linux-amd64.deb        # Debian·Ubuntu
sudo rpm -i quarto-<버전>-linux-amd64.rpm         # Fedora·RHEL 계열
```

Windows

```
winget install --id Posit.Quarto -e           # 또는 quarto.org 의 .msi
```

설치 후 터미널을 다시 열어야 `PATH` 가 적용된다.

```
cd document/manual
quarto render                 # HTML + PDF
quarto preview                # 고치면서 확인할 때
```

PDF 는 Typst 로 조판한다. Quarto 가 Typst 를 내장하고 있고 인쇄용 폰트도 `assets/fonts/print` 에 동봉되어 있으므로 별도로 설치할 항목이 없다.

> **i 매뉴얼 빌드에 mermaid-cli 는 필요하지 않다**
>
> 16장 의 ERD 두 장은 이미지로 저장되어 있다. 모델을 수정하여 이미지를 다시 생성할 때에만 mermaid-cli 가 필요하다(16.6.4.1절).
>
> ```
> ./document/manual/refresh-erd.sh
> ```

## 20.8 5. 선택: LLM과 Databricks

### 20.8.1 표준단어 제안 (`--coin`)

사전에 없는 낱말이 발견되었을 때 LLM으로부터 등재 초안을 받는 기능이다. 이 기능이 없어도 나머지 기능은 모두 사용할 수 있다.

```
# extra 에 llm 을 더해 다시 설치한다 (@sec-setup-extras)
uv tool install --force 'kdsm[db,llm] @ git+https://<리파지토리 URL>/data-standardization-
application.git@v0.11.0#subdirectory=library'
export KDSM_LLM_API_KEY=sk-ant-...        # 또는 ANTHROPIC_API_KEY
```

AWS Bedrock을 경유하려면 제공자를 변경하고 AWS 자격 증명을 기존과 같이 설정한다.

```
export KDSM_LLM_PROVIDER=bedrock
```

### 20.8.2 Databricks 진단

```
export DATABRICKS_HOSTNAME=adb-.....net
export DATABRICKS_HTTP_PATH=/sql/1.0/warehouses/...
export DATABRICKS_TOKEN=dapi...
```

호스트명은 `DATABRICKS_HOSTNAME` · `DATABRICKS_SERVER_HOSTNAME` · `DATABRICKS_HOST` 세 이름을 차례로 본다. 어느 이름으로 지정하여도 동작하나, 오류 문구가 가리키는 이름은 `DATABRICKS_HOSTNAME`이다. HTTP 경로를 모를 때는 `DATABRICKS_WAREHOUSE_ID`를 지정하면 그것으로 경로를 만들어 쓴다.

두 가지 모두 변수 이름과 기본값은 참고 자료의 환경변수 절에 정리되어 있다.

## 20.9 6. 환경변수

데이터베이스 접속 정보와 API 키는 인자가 아니라 환경변수로 받는다. 어느 명령이 무엇을 필요로 하는지, 그 값을 어디에 어떻게 두어야 하는지를 정리한다. 변수 이름과 기본값의 전체 목록은 참고 자료의 환경변수 절에 있다.

### 20.9.1 명령별로 필요한 것

표 20.7. 명령별 환경변수

| 명령 | 필요한 환경변수 | 없으면 |
|---|---|---|
| `suggest --meta` | 없음 | — |
| `suggest --from-db` | `KDSM_DB_*` 여섯 개 | 접속에 실패한다 |
| `suggest --coin` | `KDSM_LLM_API_KEY` 또는 AWS 자격 증명 | 초안 없이 진행한다 |
| `ddl` | 없음 | — |
| `validate --meta` | 없음 | — |
| `validate` (기본) | `KDSM_DB_*` 여섯 개 | 접속에 실패한다 |
| `load` · `apply` | 대상 DBMS 의 접속 정보 | 접속에 실패한다 |
| `drift` | 대상 DBMS 의 접속 정보 | 접속에 실패한다 |
| `erd -s` · `erd --meta` | 없음 | — |
| `erd` (기본) | `KDSM_DB_*` 여섯 개 | 접속에 실패한다 |
| `erd -f png` | `KDSM_MMDC_BIN` (선택) | `PATH`에서 찾는다 |
| `export --meta` | 없음 | — |
| `export` (기본) | `KDSM_DB_*` 여섯 개 | 접속에 실패한다 |
| `export --from-catalog` | 대상 DBMS 의 접속 정보 | 접속에 실패한다 |
| `erd --from-catalog` | 대상 DBMS 의 접속 정보 | 접속에 실패한다 |
| `diagnose` | `KDSM_DB_*` 여섯 개 | 접속에 실패한다 |
| `diagnose --dbms mysql` | `MYSQL_*` | 접속에 실패한다 |

| 명령 | 필요한 환경변수 | 없으면 |
|---|---|---|
| `diagnose --dbms databricks` | `DATABRICKS_*` 세 개 | 무엇이 빠졌는지 알린다 |
| `diagnose --render` | `KDSM_QUARTO_BIN` (선택) | `PATH`에서 찾는다 |

파일만 읽는 명령은 환경변수가 필요하지 않다. 산출물과 메타파일로 수행할 수 있는 작업과 데이터베이스가 필요한 작업을 이 표에서 구분한다. 처음 도입하는 단계라면 접속 정보 없이 검증과 DDL 생성부터 수행할 수 있다.

### 20.9.2 값을 두는 곳

네 가지 방법이 있다. 값이 살아 있는 범위가 서로 다르므로 상황에 맞게 고른다.

**한 번만 쓰는 경우**

그 명령에만 적용되고 셸에는 남지 않는다.

```
KDSM_DB_HOST=db.example.com kdsm diagnose --schema sales
```

PowerShell 에는 이 문법이 없다. 다음 항목처럼 지정한 뒤 사용을 마치면 제거한다.

```
$env:KDSM_DB_HOST = 'db.example.com'
kdsm diagnose --schema sales
Remove-Item Env:KDSM_DB_HOST
```

**이 터미널에서 계속 쓰는 경우**

터미널을 닫으면 사라진다.

```
export KDSM_DB_HOST=db.example.com
```

```
$env:KDSM_DB_HOST = 'db.example.com'
```

**이 장비에서 늘 쓰는 경우**

셸 설정 파일(`~/.zshrc`·`~/.bashrc`)에 기록한다. PowerShell 은 `$PROFILE` 이 가리키는 파일에 같은 줄을 기록하거나, 사용자 환경변수로 등록한다.

```
[Environment]::SetEnvironmentVariable('KDSM_DB_HOST', 'db.example.com', 'User')
```

비밀번호는 어느 방법으로도 이렇게 두지 않는다(20.10절).

**프로젝트마다 다른 경우**

저장소에 `.env`를 두고 셸에 읽어 들인다. `direnv`를 사용하면 디렉터리에 들어갈 때 자동으로 적용된다.

```
set -a && . ./.env && set +a
```

PowerShell 에는 대응하는 내장 기능이 없으므로 한 줄씩 읽어 들인다.

```
Get-Content .env ¦ Where-Object { $_ -match '^\s*[^#].*=' } ¦ ForEach-Object {
    $name, $value = $_ -split '=', 2
    Set-Item -Path "Env:$($name.Trim())" -Value $value.Trim()
}
```

`.env`는 반드시 `.gitignore`에 등록한다. 값을 담은 파일이 저장소에 올라가면 지운 뒤에도 이력에 남는다.

### 20.9.3 인자가 환경변수를 이긴다

같은 값을 두 곳에서 지정하면 인자가 우선한다.

```
인자(--host 등)  >  환경변수  >  기본값
```

이 순서를 둔 이유는 평소 설정을 그대로 두고 한 번만 다른 곳을 보게 하기 위해서이다. 접속 정보를 셸 설정에 기록해 두고, 다른 서버를 한 번 살펴볼 때만 `--host`를 지정하면 된다.

비밀번호에는 이 순서가 적용되지 않는다. 인자를 제공하지 않으므로 환경변수가 유일한 지정 방법이다.

### 20.9.4 접두어를 바꿔 여러 데이터베이스를 오간다

`KDSM_DB_`는 접두어일 뿐이며 `--env-prefix`로 교체할 수 있다. 접두어를 바꾸면 여섯 변수를 한 벌로 갈아 끼우는 셈이 된다.

```
export ECODI_DB_HOST=ecodi.example.com
export ECODI_DB_NAME=ecodi
export ECODI_DB_USER=reader
export ECODI_DB_PASSWORD='...'

kdsm diagnose --env-prefix ECODI_DB_ --schema sales
```

진단 대상은 `ECODI_DB_*`에서, 표준 사전은 기본값인 `KDSM_DB_*`에서 읽는다. 사전 쪽을 따로 지정하려면 `--dict-env-prefix`를 쓴다(18.5.7절).

접두어 끝의 밑줄은 빠뜨려도 된다. `--env-prefix ECODI_DB`라고 지정하면 밑줄만 붙여 준다. 그 이상은 손대지 않는다. 접두어를 프로그램이 지어내면 어느 변수를 읽었는지 확인할 수 없게 되기 때문이다.

### 20.9.5 지금 무엇이 지정되어 있는지 확인한다

```
env ¦ grep -E '^(KDSM_¦DATABRICKS_¦AWS_¦ANTHROPIC_)' ¦ sed 's/=.*/=<가려짐>/'
```

```
Get-ChildItem Env: ¦
  Where-Object Name -match '^(KDSM_¦DATABRICKS_¦AWS_¦ANTHROPIC_)' ¦
  ForEach-Object { "$($_.Name)=<가려짐>" }
```

값을 가리고 이름만 출력한다. 값을 그대로 출력하면 그 화면을 공유하거나 기록으로 남길 때 비밀이 함께 나간다.

접속이 되는지는 psql로 확인하는 것이 빠르다.

```
psql -h "$KDSM_DB_HOST" -p "$KDSM_DB_PORT" -U "$KDSM_DB_USER" -d "$KDSM_DB_NAME" -c 'SELECT 1'
```

> ⚠ 이름이 비슷하여 혼동하기 쉬운 두 가지
>
> 데이터베이스 이름은 `KDSM_DB_NAME`이다. `KDSM_DB_DATABASE`가 아니다. 인자는 `--database`인데 변수는 `NAME`이라 서로 다르다.
>
> `KDSM_DB_PORT`에 숫자가 아닌 값을 지정하면 접속을 시도하기 전에 중단하고 그 값을 함께 알린다. 안내 없이 기본값 5432를 적용하면 의도하지 않은 서버에 접속하고도 그 사실을 확인할 수 없기 때문이다.
>
> ```
> 오류: KDSM_DB_PORT가 숫자가 아닙니다: '5432 '
> ```

## 20.10 7. 비밀을 노출하지 않고 지정한다

비밀번호와 토큰은 인자로 받지 않는다. `KDSM_DB_PASSWORD`, `DATABRICKS_TOKEN`, AWS 키가 모두 이에 해당한다. 명령줄로 지정할 수 없으므로 환경변수를 준비해야 하며, 그 이유는 다음과 같다.

### 20.10.1 명령줄 인자가 새는 두 경로

명령줄에 적은 값은 두 곳에 그대로 남는다.

**`ps` 출력**

프로세스의 명령줄은 같은 장비에 접속한 모든 사용자가 볼 수 있다. 실행하는 순간 다른 사람의 화면에 나타나며, 명령이 끝나기를 기다릴 필요도 없다.

```
$ ps -ef | grep kdsm
kdsm  4821  ...  kdsm load meta.xlsx --password 실제비밀번호
```

공용 개발 서버나 배스천 호스트에서는 이 한 줄로 충분히 유출된다.

**셸 기록**

`~/.zsh_history`나 `~/.bash_history`에 평문으로 남는다. PowerShell은 `PSReadLine`이 `%APPDATA%\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.txt`에 기록한다. 장비를 반납하거나 계정을 넘길 때 함께 넘어가고, 백업에도 실린다. 지웠다고 생각한 뒤에도 백업본에는 남아 있다.

환경변수는 이 두 경로를 모두 피한다. 프로세스의 환경은 같은 사용자이거나 관리자만 볼 수 있고, 셸 기록에도 남지 않는다. 완전한 격리는 아니지만 위험의 크기가 다르다.

### 20.10.2 값을 화면과 기록에 남기지 않고 넣는 방법

`export KDSM_DB_PASSWORD=...`라고 입력하는 순간 그 줄이 셸 기록에 남는다. 환경변수를 쓰기로 해 놓고 기록에 남기면 얻는 것이 없다. 세 가지 방법이 있다.

**물어보게 한다**

화면에도 기록에도 남지 않는다. 가장 간단하고 확실하다.

```
read -rs -p "DB 비밀번호: " KDSM_DB_PASSWORD && export KDSM_DB_PASSWORD
```

`-s`가 입력을 화면에 표시하지 않고, `read`는 셸 내장 명령이라 값이 명령줄 인자로 노출되지 않는다. PowerShell 에서는 `Read-Host -AsSecureString` 으로 입력받는다.

```
$secure = Read-Host 'DB 비밀번호' -AsSecureString
$env:KDSM_DB_PASSWORD = [System.Net.NetworkCredential]::new('', $secure).Password
```

`Read-Host` 는 입력을 기록에 남기지 않는다. 환경변수로 옮기는 시점에는 평문이 되지만, 이는 프로그램에 전달하기 위한 것이며 화면과 기록에는 나타나지 않는다.

**권한을 제한한 파일에서 읽는다**

자주 접속하는 경우에 쓴다. 본인만 읽을 수 있도록 권한을 지정한다.

```
umask 077 && printf 'KDSM_DB_PASSWORD=%s\n' "$PW" > ~/.kdsm.env
chmod 600 ~/.kdsm.env
set -a && . ~/.kdsm.env && set +a
```

Windows 에는 `chmod` 가 없으므로 파일의 상속 권한을 끊고 본인에게만 부여한다.

```
icacls "$HOME\.kdsm.env" /inheritance:r /grant:r "$($env:USERNAME):(R)"
```

저장소 안이 아니라 홈 디렉터리에 두는 것이 안전하다. 저장소 안에 두면 실수로 함께 올릴 수 있다.

**비밀 저장소에서 꺼낸다**

조직에 비밀 관리 도구가 있으면 그것을 쓴다. 값을 파일로 남기지 않는다는 점이 다르다.

```
export KDSM_DB_PASSWORD="$(aws secretsmanager get-secret-value \
  --secret-id kdsm/db --query SecretString --output text)"
```

**! 셸 기록에 남기지 않는 임시 방편**

한 줄 앞에 공백을 두면 기록에 남지 않는 셸이 있다. bash는 `HISTCONTROL`에 `ignorespace`가 포함된 경우에만, zsh는 `HIST_IGNORE_SPACE` 옵션을 켠 경우에만 동작한다.

```
export HISTCONTROL=ignorespace          # bash
setopt HIST_IGNORE_SPACE                # zsh
 export KDSM_DB_PASSWORD='...'          # 맨 앞의 공백에 주의
```

PowerShell 에는 이에 해당하는 설정이 없다. `Set-PSReadLineOption -HistorySaveStyle SaveNothing` 으로 기록 자체를 끌 수는 있으나 그 세션의 모든 기록이 사라진다.

설정을 확인하지 않고 기대하면 그대로 기록된다. 확실한 방법은 앞의 `read -rs` 와 `Read-Host -AsSecureString` 이다.

## 20.10.3 psql과 rebuild.sh

`psql`은 `KDSM_DB_PASSWORD`를 보지 않는다. `PGPASSWORD` 또는 암호 파일에서 읽는다. `dbms/rebuild.sh`는 이 차이를 알고 있어 `KDSM_DB_PASSWORD`를 `PGPASSWORD`로 옮겨 준다. 값이 없으면 접속을 시도하기 전에 중단한다.

```
export PGPASSWORD="$KDSM_DB_PASSWORD"     # psql을 직접 부를 때만 필요하다
```

```
$env:PGPASSWORD = $env:KDSM_DB_PASSWORD
```

암호 파일을 쓰면 여러 서버의 비밀번호를 한 파일에 둘 수 있다. 파일의 위치와 권한 요건이 운영체제마다 다르다.

표 20.8. 운영체제별 PostgreSQL 암호 파일

| 운영체제 | 파일 | 권한 |
|---|---|---|
| macOS · Linux | `~/.pgpass` | `600` 이 아니면 안내 없이 무시된다 |
| Windows | `%APPDATA%\postgresql\pgpass.conf` | 권한을 검사하지 않는다 |

두 경우 모두 `PGPASSFILE` 환경변수로 다른 위치를 지정할 수 있다. 셸 기록에 남기지 않으려면 편집기로 작성한다.

```
umask 077 && "$EDITOR" ~/.pgpass          # 호스트:포트:DB:사용자:비밀번호 한 줄
chmod 600 ~/.pgpass
```

```
New-Item -ItemType Directory -Force "$env:APPDATA\postgresql" | Out-Null
notepad "$env:APPDATA\postgresql\pgpass.conf"
```

macOS·Linux 에서 권한이 `600`이 아니면 PostgreSQL이 해당 파일을 안내 없이 무시한다. 경고를 출력하지 않으므로, 비밀번호를 기재하였는데도 계속 입력을 요구한다면 권한부터 확인한다.

### 20.10.4 CI에서 다루는 방법

CI에서는 저장소 설정의 비밀 저장소에 등록하고 실행 단계에서 환경변수로 주입한다. 워크플로 파일에 값을 직접 적지 않는다. 그 파일은 저장소에 그대로 남기 때문이다.

```
- run: kdsm diagnose --schema sales --min-score 85 -q
  env:
    KDSM_DB_HOST: ${{ secrets.KDSM_DB_HOST }}
    KDSM_DB_PASSWORD: ${{ secrets.KDSM_DB_PASSWORD }}
```

접속이 필요 없는 검사만 CI에 거는 방법도 있다. 검증과 DDL 생성은 파일만 읽으므로 비밀을 다루지 않아도 된다(표 20.7). 비밀을 CI에 두는 부담이 크다면 여기까지만 연계하는 편이 낫다.

> ⚠ 값이 이미 노출되었다면
>
> `ps`나 기록, 저장소 이력에 비밀이 한 번이라도 노출되었다면 그 값은 회수할 수 없는 것으로 본다. 기록을 지우는 것보다 값을 교체하는 것이 확실하다. 지우는 작업은 백업본과 다른 사람이 받아 간 사본까지 미치지 못한다.

## 20.11 8. KDSM 시스템 환경 준비

여기까지가 도구를 설치하고 접속 정보를 지정하는 일이었다. 이제 데이터베이스 쪽을 준비한다. 도표의 3·4·5 단계이다.

표 20.9. 8절을 마치면 생기는 것

| 무엇을 | 어디에 | 몇 |
|---|---|---|
| 사용자 | 대상 DBMS | 하나 (`kdsm`) |
| 스키마 | 그 DBMS | 하나 (`kdsm_meta`) |
| 테이블 | 그 스키마 | 12 (`MT_*`) |
| 행 | 그 테이블 | 4,527 |

이 세 단계를 마치면 `kdsm_meta` 스키마가 KDSM의 사전이 된다. `--from-db`로 사전을 읽는 검증과 추천, `kdsm export`, `kdsm diagnose`, `kdsm erd`가 그때부터 동작한다. 그 이전에는 사전을 파일로 지정해야 한다(표 20.2).

읽는 정의서와 메타파일은 `kdsm` 패키지에 함께 들어 있다(11.3.3절). 그래서 저장소를 내려받지 않아도 이 절을 수행할 수 있다. 저장소를 내려받았다면 3단계 이후를 `dbms/rebuild.sh` 하나로 대신할 수 있으며, 그 경우의 DBMS별 명령 전문은 22.4.1절 에 있다.

### 20.11.1 8.1 사용자와 스키마를 만든다

KDSM이 쓸 계정과 스키마를 만들고 권한을 준다. `dbms/script/init.sh`가 세 DBMS를 모두 처리한다.

```
export PGPASSWORD=...              # 접속할 관리자의 비밀번호
export KDSM_INIT_PASSWORD=...      # 새로 만들 kdsm 사용자의 비밀번호
./dbms/script/init.sh --db ... --schema kdsm_meta --user kdsm
```

대상마다 만드는 것이 조금 다르다.

표 20.10. DBMS별로 만드는 것

| 대상 | 만드는 것 | 비고 |
|---|---|---|
| PostgreSQL | 역할·데이터베이스·스키마 | 스키마 소유자를 `kdsm`으로 둔다 |
| MySQL | 사용자·데이터베이스 | 데이터베이스가 곧 스키마이다 |
| Databricks | 카탈로그·스키마·권한 | 사용자는 계정 콘솔이나 SCIM으로 미리 만든다 |

> ! 이 단계만 관리자로 접속한다
>
> 사용자와 데이터베이스를 만드는 권한과 테이블을 만드는 권한은 층위가 다르다. 이 단계는 PostgreSQL의 `postgres`, MySQL의 `root`, Databricks의 메타스토어 관리자처럼 그 권한을 가진 계정으로 접속한다.
>
> 8.2 이후는 여기서 만든 `kdsm` 계정으로 수행한다. 한 계정으로 묶으면 일상 작업에 쓰는 계정에 필요 이상의 권한을 주게 된다.

> 무엇을 보낼지 먼저 확인한다
>
> `--dry-run`을 지정하면 데이터베이스에 접속하지 않고 보낼 문장만 보여 준다.
>
> ```
> ./dbms/script/init.sh --dbms databricks --catalog dev_landing \
>   --principal kdsm@example.com --dry-run
> ```

비밀번호는 인수로 받지 않는다. 명령줄에 적은 값은 `ps` 출력과 셸 기록에 남기 때문이다(20.10.1절).

#### 20.11.1.1 저장소가 없을 때

`dbms/script/init.sh`는 저장소의 스크립트이며 패키지에 들어 있지 않다. 패키지만 설치하였다면 아래 문장을 관리자에게 넘긴다. 스크립트가 만드는 것과 같다.

PostgreSQL

```
CREATE ROLE kdsm LOGIN PASSWORD '...';
CREATE DATABASE ... OWNER kdsm ENCODING 'UTF8';
GRANT CONNECT, CREATE ON DATABASE ... TO kdsm;
-- ... 에 접속한 뒤
CREATE SCHEMA IF NOT EXISTS kdsm_meta AUTHORIZATION kdsm;
GRANT USAGE, CREATE ON SCHEMA kdsm_meta TO kdsm;
```

MySQL

```
CREATE DATABASE IF NOT EXISTS kdsm_meta DEFAULT CHARACTER SET utf8mb4;
CREATE USER IF NOT EXISTS 'kdsm'@'%' IDENTIFIED BY '...';
GRANT ALL PRIVILEGES ON kdsm_meta.* TO 'kdsm'@'%';
```

Databricks

```
CREATE CATALOG IF NOT EXISTS dev_landing;
CREATE SCHEMA IF NOT EXISTS dev_landing.kdsm_meta;
GRANT USE CATALOG ON CATALOG dev_landing TO `kdsm@example.com`;
GRANT USE SCHEMA, CREATE TABLE, MODIFY, SELECT
  ON SCHEMA dev_landing.kdsm_meta TO `kdsm@example.com`;
```

MySQL에는 데이터베이스 안의 스키마가 없다. 데이터베이스가 곧 스키마이므로 다른 DBMS의 `kdsm_meta` 스키마에 해당하는 것이 `kdsm_meta` 데이터베이스이다. Databricks에는 사용자를 만드는 SQL이 없으므로, 계정 콘솔이나 SCIM으로 등록해 둔 주체에 권한을 준다.

### 20.11.2 8.2 메타테이블 12개를 만든다

메타테이블의 구조는 물리 데이터 모델 정의서가 정한다. 그 정의서에서 DDL을 만들고 데이터베이스에 적용한다.

```
kdsm ddl --seed --no-schema -o ddl/
#> DDL 3벌을 만들었습니다. 앞에서 뒤로 적용하십시오.
```

`--seed`가 패키지에 든 정의서 셋을 가리킨다(14.5.1절). `--no-schema`를 지정하는 것은 스키마를 8.1에서 이미 만들었기 때문이다. PostgreSQL은 `CREATE SCHEMA IF NOT EXISTS`도 데이터베이스에 대한 `CREATE` 권한을 먼저 확인하므로, 스키마가 있어도 운영 계정에서는 막힌다.

만든 SQL을 적용하는 것은 이 명령의 일이 아니다. 만드는 일과 적용하는 일은 필요한 권한도 되돌리는 방법도 다르다. `kdsm`에는 SQL을 실행하는 명령이 없으며 앞으로도 두지 않는다. 적용은 DBMS 클라이언트가 이미 하는 일이고, 라이브러리가 그것을 떠안으면 드라이버와 트랜잭션, 중단 지점의 복구까지 함께 책임지게 된다.

적용 순서는 표준 사전 → 논리 → 물리이다. 뒤의 것이 앞의 것을 가리키므로 순서를 바꾸면 외래키에 걸린다. 파일 이름의 `std`·`logical`·`physical`이 그 차례이다.

**표 20.11.** DBMS별 SQL 적용 수단

| 대상 | 적용 수단 | 중간에서 실패하면 |
|---|---|---|
| PostgreSQL | `psql --file` | `--single-transaction`을 주면 처음 상태로 돌아간다 |
| MySQL | `mysql < 파일` | 그때까지 만든 테이블이 남는다 |
| Databricks | `dbms/apply_sql.py` | 몇 번째 문장인지 알리며 앞의 것은 남는다 |

### 20.11.2.1 PostgreSQL

```
export PGPASSWORD="$KDSM_DB_PASSWORD"
psql -h "$KDSM_DB_HOST" -p "$KDSM_DB_PORT" -U kdsm -d ... \
     -v ON_ERROR_STOP=1 --single-transaction \
     -f ddl/kdsm_meta_std.sql \
     -f ddl/kdsm_meta_logical.sql \
     -f ddl/kdsm_meta_physical.sql
```

`ON_ERROR_STOP=1`을 지정하지 않으면 문장 하나가 실패하여도 나머지를 계속 보내고 종료 코드 0으로 끝난다. 절반만 만들어진 스키마를 성공으로 보게 된다.

`--single-transaction`은 세 파일을 한 트랜잭션으로 묶는다. PostgreSQL은 DDL도 트랜잭션에 들어가므로 중간에서 실패하면 처음 상태로 돌아간다. 다시 실행하기 전에 지울 것이 남지 않는다.

`psql`은 `KDSM_DB_PASSWORD`를 보지 않는다. `PGPASSWORD` 또는 `~/.pgpass`에서 읽는다(20.10.3절).

### 20.11.2.2 MySQL

```
export MYSQL_PWD="$KDSM_DB_PASSWORD"
for sql in ddl/kdsm_meta_std.sql ddl/kdsm_meta_logical.sql ddl/kdsm_meta_physical.sql; do
  mysql -h "$MYSQL_HOST" -P "${MYSQL_PORT:-3306}" -u kdsm kdsm_meta < "$sql" || break
done
```

데이터베이스가 스키마를 겸하므로 접속할 때 데이터베이스 이름으로 `kdsm_meta`를 준다(표 20.10).

`mysql`은 기본적으로 첫 오류에서 멈춘다. `-f`(`--force`)를 지정하면 오류를 건너뛰고 계속 진행하므로 이 자리에서는 지정하지 않는다. 위의 `|| break`는 앞 파일이 실패했을 때 다음 파일을 보내지 않기 위한 것이다.

MySQL의 DDL은 트랜잭션에 들어가지 않는다. 중간에서 실패하면 그때까지 만들어진 테이블이 남으므로, 남은 것을 확인하고 실패한 파일부터 이어 간다.

#### 20.11.2.3 Databricks

SQL 파일을 그대로 실행하는 명령줄 도구가 없다. 저장소의 `dbms/apply_sql.py`가 그 자리를 메운다. 파일을 문장 단위로 나누어 차례로 보내며, 접속 정보는 `kdsm`과 같은 환경변수에서 읽는다.

```
python3 dbms/apply_sql.py --dbms databricks -n ddl/*.sql    # 문장 수만 센다
python3 dbms/apply_sql.py --dbms databricks ddl/*.sql
```

문장을 나눌 때 작은따옴표 안의 쌍반점은 구분자로 보지 않는다. 한글 주석에 쌍반점이 들어가는 일이 있는데, 그 자리에서 문장이 잘리면 문법 오류가 난다.

이 스크립트는 저장소에 있고 패키지에는 들어 있지 않다. 패키지만 설치한 경우에는 Databricks SQL 편집기나 노트북에 파일 내용을 붙여 실행한다.

Databricks에도 DDL 트랜잭션이 없다. 실패하면 몇 번째 문장에서 멈추는지와 그 문장의 앞부분을 알리며, 앞서 만들어진 테이블은 남는다.

#### 20.11.2.4 적용되었는지 확인한다

테이블이 12개인지를 본다. 수가 모자라면 적용이 중간에서 멈춘 것이다.

```
psql -d ... -c '\dt kdsm_meta.*'                    # PostgreSQL
mysql -u kdsm -e 'SHOW TABLES IN kdsm_meta;'        # MySQL
```

Databricks는 SQL 편집기에서 `SHOW TABLES IN dev_landing.kdsm_meta;`를 실행한다. 보다 자세한 확인은 8.4에서 수행한다(20.11.4절).

> **i 테이블을 만들 권한이 없을 때**
>
> 스키마 생성까지는 통과하고 첫 `CREATE TABLE`에서 중단되는 경우이다.
>
> ```
> PERMISSION_DENIED: User does not have CREATE TABLE on Schema '...'
> ```
>
> 만든 SQL을 권한이 있는 사람에게 넘기고, 적용을 마친 뒤 8.3부터 이어 간다. DDL을 파일로 만들어 두는 구조가 이 상황을 다룰 수 있게 해 준다.

### 20.11.3 8.3 초기 데이터를 적재한다

빈 테이블에 표준 사전과 KDSM 자신의 논리·물리 모델을 넣는다.

```
kdsm load --seed --actor 작업자 -n      # 먼저 쓰지 않고 확인
kdsm load --seed --actor 작업자
```

세 벌을 적재 순서대로 넣으며, 정의서도 영역마다 짝지어 함께 읽는다(15.4.3절). 정의서를 함께 읽는 것은 트랜잭션을 시작하기 전에 NOT NULL과 길이까지 확인하기 위함이다. 데이터베이스가 거절하고 나서 아는 것보다 낫다.

표 20.12. 영역별 적재 대상

| 영역 | 시트 | 메타테이블 | 행 |
|---|---|---|---|
| 데이터 표준 | 7 | `MT_STD_*` · `MT_STDCD_*` 7 | 4,527 |

| 영역 | 시트 | 메타테이블 | 행 |
|---|---|---|---|
| 논리 모델 | 2 | `MT_ENTTY` · `MT_ATRB` | 166 |
| 물리 모델 | 3 | `MT_DB` · `MT_TBL` · `MT_COL` | 167 |

`--actor`에 지정한 이름이 `CREATR`·`MDFR`에 기록된다. 생략하면 접속 사용자명을 쓴다. 누가 넣은 값인지 남기는 자리이므로 사람 이름으로 지정하는 편이 낫다.

다시 실행해도 안전하다. 기본키가 있으면 갱신하므로 같은 파일을 여러 번 넣어도 겹쳐 쌓이지 않는다(15.3.1절).

### 20.11.4 8.4 준비되었는지 확인한다

세 가지로 확인한다. 순서대로 수행하면 어디가 어긋났는지 좁혀 갈 수 있다.

```
# 1. 사전이 들어갔는가 — 사전을 DB에서 읽어 용어를 추천해 본다
kdsm suggest 고객주문일자 --from-db

# 2. 정의서와 실제 테이블이 같은가
kdsm drift -s <물리 정의서> --schema kdsm_meta

# 3. 적재 결과를 수치로 확인한다
kdsm diagnose --schema kdsm_meta -q
```

1이 후보를 내놓으면 사전이 들어간 것이다. 2가 「차이 없음」이면 만든 테이블이 정의서와 일치한다. 3의 준수율은 KDSM 자신의 메타테이블을 대상으로 하므로 높게 나온다. 낮게 나오면 대상 스키마를 잘못 지정한 것이다.

> **i 여기까지 마치면 사전을 파일로 지정하지 않아도 된다**
>
> 이후의 명령에서 `--meta`를 생략하고 `--from-db`를 지정하면 적재해 둔 사전을 읽는다. 파일과 데이터베이스가 갈라질 여지가 없어지므로, 사전을 적재한 뒤에는 이쪽을 쓰는 편이 낫다(12장).

> **! 함께 배포하는 사전은 출발점이다**
>
> 패키지에 든 표준단어 사전을 그대로 써야 하는 것이 아니다. 적재를 마친 뒤 업무에 맞지 않는 낱말을 덜어 내고 필요한 낱말을 더한다. 그 작업이 표준화의 첫 단계이다(2장).

## 20.12 처음부터 끝까지

새 장비에서 처음 시작하는 경우의 최소 절차이다. 두 갈래로 나누어 적는다.

### 20.12.1 배포한 패키지를 설치한 경우

일반 사용자의 길이다. 저장소를 내려받지 않는다.

macOS · Linux · Windows

```
# 1. 설치. 주소가 길어 변수에 담는다
KDSM_PKG="kdsm[db] @ git+https://<리파지토리 URL>/data-standardization-
application.git@v0.11.0#subdirectory=library"
uv tool install "$KDSM_PKG"
```

```
kdsm --help                              # 확인

export KDSM_DB_HOST=... KDSM_DB_NAME=... \
       KDSM_DB_SCHEMA=kdsm_meta KDSM_DB_USER=kdsm    # 2. 접속 정보
read -rs -p "비밀번호: " KDSM_DB_PASSWORD && export KDSM_DB_PASSWORD

# 3. 사용자와 스키마는 관리자에게 요청한다 (@sec-setup-account-sql)

kdsm ddl --seed --no-schema -o ddl/      # 4. 메타테이블 DDL
psql -d ... -f ddl/kdsm_meta_std.sql \
                  -f ddl/kdsm_meta_logical.sql \
                  -f ddl/kdsm_meta_physical.sql

kdsm load --seed --actor 작업자 -n        # 5. 먼저 쓰지 않고 확인
kdsm load --seed --actor 작업자

kdsm diagnose --schema kdsm_meta -q      # 확인
```

Windows 에서는 환경변수 지정만 다르다. `$env:NAME = '값'` 을 사용한다(표 20.3). `psql` 자리에는 설치한 클라이언트를 쓰며, MySQL은 `mysql < 파일`, Databricks는 SQL 편집기나 `dbms/apply_sql.py`를 사용한다.

### 20.12.2 저장소를 내려받은 경우

KDSM 자체를 고치거나 배포 전의 판을 시험할 때이다. 1·2 단계를 운영체제별로 나누어 적고, 3·4·5는 `dbms/rebuild.sh` 하나로 줄였다.

macOS · Linux

```
# 1. uv
curl -LsSf https://astral.sh/uv/install.sh ¦ sh

# 2. 라이브러리
cd library && uv sync --all-extras

# 3. 여기까지만으로 되는 것 — 검증
uv run kdsm validate --meta ../dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx \
  --logical ../dbms/meta/meta_data/kdsm_logical_meta_20260828.xlsx \
  --physical ../dbms/meta/meta_data/kdsm_physical_meta_20260828.xlsx \
  --name-prefix MT_ -q

# 4. ERD (텍스트) — 산출물에서 그리므로 접속하지 않는다
uv run kdsm erd -s ../app/deliverable/DM-DST-003_*.xlsx

# 5. DB까지 하려면 — psql 설치 후 접속 정보를 넣고
export KDSM_DB_HOST=... KDSM_DB_NAME=... KDSM_DB_USER=... KDSM_DB_PASSWORD=...
cd .. && ./dbms/rebuild.sh --dry-run     # 먼저 쓰지 않고 확인
./dbms/rebuild.sh
```

Windows (PowerShell)

```
# 1. uv
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 ¦ iex"

# 2. 라이브러리
cd library; uv sync --all-extras

# 3. 여기까지만으로 되는 것 — 검증
uv run kdsm validate --meta ..\dbms\meta\meta_data\kdsm_modeling_meta_20260828.xlsx `
  --logical ..\dbms\meta\meta_data\kdsm_logical_meta_20260828.xlsx `
  --physical ..\dbms\meta\meta_data\kdsm_physical_meta_20260828.xlsx `
```

```
  --name-prefix MT_ -q

# 4. ERD (텍스트) — 산출물에서 그리므로 접속하지 않는다
uv run kdsm erd -s ..\app\deliverable\DM-DST-003_*.xlsx

# 5. DB까지 하려면 — psql 설치 후 접속 정보를 넣고
$env:KDSM_DB_HOST = '...'
$env:KDSM_DB_NAME = '...'
$env:KDSM_DB_USER = '...'
$env:KDSM_DB_PASSWORD = Read-Host 'DB 비밀번호'
```

PowerShell 은 줄을 이을 때 역따옴표(`)를 사용한다. 5번의 `rebuild.sh` 는 bash 스크립트이므로 Git Bash 나 WSL 에서 실행하거나, 15.6절 의 단계를 각각 실행한다.

3번이 통과하면 파이썬 환경 구성은 완료된 것이다. 전체 작업 흐름은 22장 에 있다.

## 20.13 흔한 실패

표 20.13. 흔한 실패와 조치

| 증상 | 대상 | 원인 | 조치 |
|---|---|---|---|
| `kdsm: command not found` | 전체 | 설치 위치가 `PATH` 에 없음 | `uv tool install` 이면 `~/.local/bin` 을 `PATH` 에 넣는다. 저장소의 개발판이면 `uv run kdsm ...` 으로 실행한다 |
| `ModuleNotFoundError: kdsm` | 전체 | 다른 파이썬 환경에서 실행함 | 설치한 환경에서 실행한다 (20.4.2절) |
| `오류: psycopg가 설치되어 있지 않습니다` | 전체 | `db` extra 없이 설치 | extra 를 더해 `uv tool install --force` 로 다시 설치한다. 개발판이면 `uv sync --all-extras` |
| `permission denied for database` | 전체 | 스키마 CREATE 권한 없음 | 관리자에게 `CREATE SCHEMA ... AUTHORIZATION` 요청 |
| `mermaid-cli(mmdc)를 찾을 수 없다` | 전체 | 설치되지 않았거나 PATH에 없음 | `npm install -g @mermaid-js/mermaid-cli` |
| `Could not find chrome-headless-shell` | 전체 | 브라우저가 없음 | `npx puppeteer browsers install chrome-headless-shell` 또는 `--chrome` |
| `[WARN] pyarrow is not installed` | 전체 | Databricks 커넥터 4.0 부터 pyarrow 를 기본으로 깔지 않는다 | 무시한다. 대량 결과를 받을 때 쓰는 Arrow 경로와 cloud fetch 가 비활성화되는 것이며, DDL 적용·적재·카탈로그 조회는 그 경로를 쓰지 않는다 |
| `TABLE_OR_VIEW_ALREADY_EXISTS` | 전체 | 이미 세운 스키마에 다시 적용함 | 다시 세우려면 `--drop`, 있는 것을 두고 적재만 하려면 `--skip-apply` |
| `psql: command not found` | macOS | libpq가 keg-only | `brew link --force libpq` |

| 증상 | 대상 | 원인 | 조치 |
|---|---|---|---|
| `DDL_ARGS[@]: unbound variable` | macOS | 기본 bash 3.2 | 저장소를 최신 버전으로 갱신한다 |
| `psql` 실행 파일을 찾지 못함 | Windows | 설치 경로가 `PATH` 에 없음 | `C:\Program Files\PostgreSQL\<버전>\bin` 을 `PATH` 에 추가한다 |
| `uv` · `quarto` 를 인식하지 못함 | Windows | 설치 후 `PATH` 미적용 | 터미널을 다시 연다 |
| `.sh 파일을 실행할 수 없음` | Windows | bash 스크립트 | Git Bash 또는 WSL 에서 실행한다(20.3절) |
| `UnicodeEncodeError` | Windows | 콘솔 코드 페이지가 UTF-8 이 아님 | `$env:PYTHONUTF8 = '1'` (20.3절) |
| `이 시스템에서 스크립트를 실행할 수 없습니다` | Windows | PowerShell 실행 정책 | `powershell -ExecutionPolicy ByPass -c ...` 형태로 실행한다 |
| `npm ERR! EACCES` | Linux | 전역 설치 권한 없음 | `npm config set prefix ~/.npm-global` 후 `PATH` 에 추가한다 |
| Node.js 버전이 낮음 | Linux | 배포판 패키지가 오래됨 | nodejs.org 배포본이나 `nvm` 을 사용한다 |

원인별 예방 방법은 21장 에 있다.

> 도구가 없을 때 수행할 수 있는 범위
>
> 기능 하나가 빠졌다고 하여 전체가 중단되지는 않는다. `--extra`를 설치하지 않으면 해당 기능만 설치되어 있지 않다고 안내하고 중단되며, mermaid-cli가 없으면 이미지 생성만 수행할 수 없다. 텍스트 ERD와 검증은 그대로 수행된다. 운영체제도 마찬가지이다. Windows 에서 bash 스크립트를 실행할 수 없더라도 각 명령은 그대로 사용할 수 있다.
>
> 따라서 처음에는 uv만 설치하고 3번까지 수행해 보는 방법을 권장한다. 그 단계까지 정상적으로 수행되면 나머지는 필요할 때 하나씩 추가하면 된다.

CHAPTER 21

# 자주 발생하는 문제와 예방

표준화 작업에서 반복하여 발생하는 문제를 원인별로 정리한 것이다. 증상이 아니라 원인으로 묶었다. 같은 원인에서 나오는 문제는 증상이 서로 달라 보여도 예방 방법이 하나이기 때문이다.

이미 오류가 발생하였다면 21.2절 의 진단표에서 증상을 찾는 편이 빠르다. 이 장의 앞부분은 오류가 발생하기 전에 읽는 부분이다.

## 21.1 원인별 유형

### 21.1.1 유형 1 · 같은 사실을 두 곳에 기재하면 어긋난다

가장 빈번한 유형이다. 사람이 옮겨 기재하는 지점마다 불일치가 발생할 여지가 생긴다. 옮겨 기재할 수밖에 없는 지점을 먼저 파악하고, 각각에 검증 규칙을 두는 것이 예방 방법이다. 그 목록은 9.4.1절 에 있다.

#### 21.1.1.1 컬럼명이 표준용어 사전과 다른 경우

사전을 수정하면서 산출물을 함께 수정하지 않으면 발생한다. 한 글자 차이인 경우가 많아 육안으로는 발견하기 어렵다.

표 21.1. 산출물과 사전이 어긋나는 형태

| 산출물 | 사전 |
|---|---|
| `DT_OCRN_CYCLE_CD` | `DATA_OCRN_CYCLE_CD` |
| `PKUP_YN` | `BKUP_YN` |
| `DT_TYPE 자료타입` | `DATA_TYPE_CN 자료유형내용` |

`column.term-registered`와 `column.term-name`이 검출한다. 사전을 수정하였다면 산출물을 함께 수정하고 검증을 실행한다.

#### 21.1.1.2 테이블 이름이 문서마다 다른 경우

한 테이블의 이름이 테이블 정의서·컬럼 정의서·메타파일 시트명 세 곳에 각각 기재된다. 접두어를 붙이거나 낱말을 추가하는 개정이 있으면 세 곳이 갈린다. `DMN_GROUP` · `MT_DMN_GROUP` · `MT_STD_DMN_GROUP`과 같은 형태이다.

> ⚠ 경고
>
> 컬럼 정의서의 이름이 잘못되면 해당 컬럼들이 어느 테이블에도 포함되지 못하여 **안내 없이 누락된다**. 오류 없이 컬럼이 빠진 DDL이 생성되므로 DDL 생성 결과만으로는 확인할 수 없다.

`table.has-columns`와 `column.table-defined`가 양방향으로 검출한다. 테이블 이름을 변경할 때에는 22.2.5절 의 목록으로 함께 변경할 곳을 먼저 확인한다.

#### 21.1.1.3 기본키가 두 시트에서 다른 경우

테이블 정의서의 기본키 항목과 컬럼 정의서의 기본키 순번이 서로 다른 구성을 가리키는 경우이다. 구성만이 아니라 순서가 다른 경우도 포함한다.

표 21.2. 두 시트에서 어긋나는 형태

| 테이블 | 테이블 정의서 | 컬럼 정의서 순번 |
|---|---|---|
| `MT_TBL` | `TBL_PHYS_NM` | `TBL_PHYS_NM, DB_NM, SCHM_NM` |
| `MT_COL` | `TBL_PHYS_NM, COL_PHYS_NM` | `COL_PHYS_NM, TBL_PHYS_NM` |

DDL은 컬럼 정의서만 참조하므로 생성 결과에는 영향이 없다. 그러므로 DDL이 정상적으로 생성되었다는 사실은 두 시트가 일치한다는 근거가 되지 않는다. 문서를 읽는 사람만 오해하게 되는 형태이다. `table.primary-key`가 구성과 순서를 모두 대조한다.

#### 21.1.1.4 코드를 폐기하였으나 참조가 남는 경우

표준코드를 폐기할 때 산출물의 참조 코드는 삭제하면서 표준용어의 `STDCD_ID`를 함께 정리하지 않으면 발생한다. `term.code-registered`가 검출한다.

폐기 상태의 코드는 다른 검사에서 제외되므로 문제가 더 늦게 드러난다. 코드를 폐기할 때에는 폐기 처리와 참조 정리를 한 번에 수행한다.

### 21.1.2 유형 2 · 검증은 통과하지만 데이터베이스가 거부한다

검증 규칙과 데이터베이스 제약은 판단하는 층위가 다르다. 규칙은 정의서 안의 정합성을 보고, 제약은 실제 값을 본다. 두 기준이 어긋나면 검증을 통과한 정의서가 적재에서 중단된다.

> ! 산출물을 함께 지정하면 트랜잭션 이전에 검출된다
>
> `column.not-null-satisfied`와 `column.length-satisfied`가 이 유형을 검출한다. 두 규칙은 산출물의 정의와 실제 데이터를 대조하므로 산출물을 함께 지정해야 동작한다.
>
> ```
> load_metafile(path, spec="고객주문_V0.1.0_20260812.xlsx")
> ```
>
> 지정하지 않으면 적재 시점에 데이터베이스가 거부한 뒤에야 확인하게 된다.

#### 21.1.2.1 Null 허용 여부가 어긋나는 경우

```
psycopg.errors.NotNullViolation: null value in column "dmn_len" violates not-null constraint
DETAIL: Failing row contains (연월일시분초D, 02, 0207, DATETIME, null, ...)
```

정의상 값이 없어도 되는 항목에 NOT NULL을 지정한 경우이다. `DATETIME` 도메인의 길이 항목이 대표적이다. `domain.length-required`는 `DATETIME`에 길이가 없는 상태를 정상으로 판정하고 오히려 길이가 지정되어 있으면 위반으로 검출하는데, 테이블은 NOT NULL을 요구하는 상태가 된다.

값이 없는 것이 정상인 항목은 산출물의 Null 허용을 `Y`로 기재한다. 용어구분이 엔터티명인 표준용어의 도메인 항목도 같은 경우이다.

#### 21.1.2.2 도메인 길이가 실제 값보다 짧은 경우

```
psycopg.errors.StringDataRightTruncation: value too long for type character varying(40)
CONTEXT: COPY stg_mt_std_dmn, line 79, column exprs_frm_cn: "M99999999_S99999999_..."
```

설명·표현형식·값범위와 같이 내용을 기술하는 컬럼에서 자주 발생한다. 작성 시점의 값에 맞추어 길이를 정하면 이후에 더 긴 값이 들어오면서 적재가 중단된다.

표 21.3. 길이가 부족하기 쉬운 컬럼의 예

| 컬럼 | 이전 | 이후 |
|---|---|---|
| `STRG_FRM_CN` | 내용V40 | 내용V80 |
| `EXPRS_FRM_CN` | 내용V40 | 내용V80 |
| `PRM_VL_SCP_CN` | 내용V80 | 내용V256 |

설명성 컬럼은 현재 값보다 여유를 두어 도메인을 선택한다. 길이를 늘리는 개정은 이미 적재한 데이터가 있는 상태에서 수행해야 하므로 처음에 넉넉히 잡는 편이 비용이 적다.

### 21.1.3 유형 3 · 서식으로 표현할 수 없는 관계가 있다

#### 21.1.3.1 기본키이면서 외래키인 컬럼

하나의 코드 항목으로 기본키·외래키·일반을 구분하는 방식으로는 식별관계를 표현할 수 없다. 기본키이면서 동시에 외래키인 컬럼이 어느 쪽으로도 기재되지 못하기 때문이다.

그러므로 기본키 순번과 참조 테이블·컬럼을 각각 별도 항목에 기재한다(10장). 두 항목이 모두 채워진 컬럼이 식별관계를 이룬다.

#### 21.1.3.2 외래키가 대상 기본키의 앞 일부만 참조하는 경우

```
CONSTRAINT PK_MT_TBL PRIMARY KEY (TBL_PHYS_NM, DB_NM, SCHM_NM),
CONSTRAINT FK_MT_COL_1 FOREIGN KEY (TBL_PHYS_NM) REFERENCES kdsm_meta.MT_TBL (TBL_PHYS_NM)
```

참조 대상의 기본키가 복합키인데 외래키가 그중 일부만 가리키면 DDL은 생성되지만 실행 단계에서 거부된다. 생성이 성공하였다는 사실이 실행 가능을 뜻하지 않는 형태이므로, DDL을 적용하기 전에 `column.foreign-key-target`으로 확인한다.

## 21.1.4 유형 4 · 도구의 동작 방식을 모르면 막힌다

### 21.1.4.1 식별자는 소문자로 접혀 저장된다

```
psycopg.errors.UndefinedTable: relation "kdsm_meta.MT_STD_DMN_GROUP" does not exist
```

DDL 생성기는 식별자를 인용하지 않는다(14.7.1절). PostgreSQL은 인용하지 않은 식별자를 **소문자로 접어서** 저장하므로, `CREATE TABLE MT_STD_WORD`로 생성한 테이블이 카탈로그에는 `mt_std_word`로 등록된다.

> ! 대문자로 조회하려면 인용해야 하고, 인용하면 다른 이름이 된다
>
> 인용 부호는 이름을 글자 그대로 사용하라는 지시이다. 따라서 `"MT_STD_WORD"`는 접혀 저장된 `mt_std_word`와 서로 다른 이름이다. 조회할 때에는 인용하지 않거나 소문자로 기재한다.
>
> ```
> SELECT * FROM kdsm_meta.mt_std_word;     -- 조회된다
> SELECT * FROM kdsm_meta.MT_STD_WORD;     -- 조회된다 (접혀서 같은 이름이 된다)
> SELECT * FROM kdsm_meta."MT_STD_WORD";   -- 조회되지 않는다
> ```

카탈로그를 직접 조회하는 도구를 사용할 때에는 이 점을 먼저 확인한다. 표기가 대문자로 보이는 것은 정의서의 기재이고, 데이터베이스에 저장된 이름은 소문자이다. 진단과 대조는 양쪽을 같은 규칙으로 접어 판정하므로 이 차이를 사용자가 조정할 필요가 없다(14.9.3절).

### 21.1.4.2 선택 의존성은 한 번에 지정한다

`uv sync`는 기본적으로 정확 동기화를 수행한다. 요청한 항목만 남기고 나머지를 제거하므로, 필요한 기능을 나누어 설치하면 뒤에 실행한 것이 앞에 설치한 것을 제거한다.

```
$ uv sync --extra db
$ uv sync --extra llm
$ kdsm suggest 종신보험이름 --from-db --coin
#> 오류: psycopg가 설치되어 있지 않습니다
```

두 번째 명령이 llm만 필요하다는 의미가 되어 psycopg를 제거한 것이다. 한 번에 지정하면 발생하지 않는다.

```
uv sync --extra db --extra llm
uv sync --all-extras           # extra 가 늘어도 다시 고치지 않아도 된다
```

선택 의존성은 6종이다. 어떤 기능이 활성화되는지 판단하기 어려운 경우에는 `--all-extras`를 사용한다.

표 21.4. 선택 의존성 6종

| extra | 활성화되는 기능 | 없을 때 출력되는 메시지 |
|---|---|---|
| `db` | DB에서 사전 읽기·적재, PostgreSQL 진단 | `psycopg가 설치되어 있지 않습니다` |
| `llm` | `--coin` 제안 (Claude API) | `LLM 제안에는 추가 패키지가 필요하다` |
| `bedrock` | `--coin` 제안 (AWS Bedrock) | `Bedrock을 쓰려면 uv sync --extra bedrock` |
| `databricks` | Databricks 적재·조회·진단 (`--dbms databricks`) | `Databricks를 진단하려면 uv sync --extra databricks` |

| extra | 활성화되는 기능 | 없을 때 출력되는 메시지 |
|---|---|---|
| `mysql` | MySQL 적재·조회·진단 ( `--dbms mysql` ) | `MySQL을 조회하려면 uv sync --extra mysql` |
| `pdf` | ERD PDF 합본 ( `--pdf-merge` ) | `pypdf가 설치되어 있지 않습니다` |

> **i** `llm`과 `bedrock`은 별개이다
>
> 둘 다 `anthropic` 패키지를 사용하지만 `bedrock`은 AWS 서명에 필요한 패키지가 추가로 필요하다. Bedrock만 사용한다면 `--extra bedrock` 하나로 충분하며, `--extra llm`을 함께 지정할 필요는 없다.

> **⚠** `--inexact`는 권장하지 않는다
>
> 이미 설치한 항목 위에 추가할 수는 있으나, 환경이 잠금 파일과 일치하지 않는 상태로 남는다. 다음 담당자가 같은 명령을 실행하더라도 다른 환경이 구성된다.

#### 21.1.4.3 `CREATE SCHEMA`는 스키마가 있어도 막힌다

DDL 첫 줄의 `CREATE SCHEMA IF NOT EXISTS`는 스키마가 이미 존재하면 그대로 통과할 것으로 보이지만, PostgreSQL은 존재 여부보다 권한을 먼저 확인한다. 운영 계정에 데이터베이스 CREATE 권한이 없으면 스키마가 존재하더라도 실행이 거부된다.

```
psql:<파일>.sql:1: ERROR:  permission denied for database <데이터베이스>
```

스키마 생성은 한 번만 수행하며 권한 수준도 테이블과 다르다. 운영 데이터베이스에 적용할 DDL은 `kdsm ddl --no-schema`로 스키마 생성문을 제외하고, 스키마는 관리자가 별도로 생성한다(표 14.3).

#### 21.1.4.4 macOS 기본 bash에서 빈 배열은 오류를 일으킨다

`set -u` 상태에서 원소가 없는 배열을 전개하면 bash 4.4 미만에서 `unbound variable` 오류로 중단된다. macOS 기본 bash는 3.2이므로 스크립트를 직접 작성할 때 주의한다.

```
DDL_ARGS=()
[ "$DROP" -eq 1 ] && DDL_ARGS+=(--drop)
kdsm ddl "$SPEC" -o "$OUT" "${DDL_ARGS[@]}"   # --drop 없으면 여기서 중단된다
```

옵션의 수가 적다면 분기하여 호출하는 방식이 명확하다. 이 장에 나오는 옵션의 뜻은 각 명령을 다루는 장에 있다. `kdsm ddl`은 표 14.3, 나머지는 19장 를 참고한다.

#### 21.1.4.5 Databricks는 웨어하우스와 권한을 따로 확인한다

접속에 필요한 것이 셋이다. 자격 증명, 웨어하우스를 가리키는 값, 그리고 그 웨어하우스를 사용할 권한이다. 셋은 서로 독립적이며 실패하는 방식이 다르므로 오류 문구로 구분한다.

표 21.5. Databricks 접속 실패의 구분

| 오류 문구 | 원인 |
|---|---|
| `invalid_client` | 클라이언트 아이디 또는 시크릿이 틀렸다 |
| `ENDPOINT_NOT_FOUND` | 웨어하우스 식별자가 틀렸거나 다른 워크스페이스의 것이다 |

| 오류 문구 | 원인 |
|---|---|
| `PERMISSION_DENIED` | 자격 증명은 맞으나 웨어하우스 사용 권한이 없다 |

`ENDPOINT_NOT_FOUND`는 값이 멀쩡해 보이는 경우가 많아 찾기 어렵다. 워크스페이스의 SQL → SQL Warehouses → Connection details 에서 Server hostname 과 HTTP path 를 함께 확인한다. 둘이 같은 화면에 있으므로 짝이 어긋나는 것을 그 자리에서 확인할 수 있다.

카탈로그와 테이블의 읽기 권한은 오류를 발생시키지 않고 대상에서 빠진다. 진단 결과의 컬럼 수가 예상과 다르면 권한을 먼저 확인한다(18.5.4.4절).

#### 21.1.4.6 pyarrow 경고는 조치하지 않아도 된다

`databricks-sql-connector`가 불러들여질 때 출력하는 안내이다. 4.0.0부터 `pyarrow`를 기본 의존성에서 제외하였기 때문이며, kdsm의 동작에는 영향이 없다.

```
[WARN] pyarrow is not installed by default since databricks-sql-connector 4.0.0 ...
```

kdsm은 커서에서 결과를 그대로 읽으며 Arrow 계열 기능을 사용하지 않는다. 안내에 나오는 cloud fetch 는 결과가 매우 클 때의 최적화인데, kdsm이 실행하는 것은 `information_schema` 조회이므로 그 규모에 해당하지 않는다.

#### 21.1.4.7 엑셀을 열어 둔 채 실행하지 않는다

파일이 열려 있으면 `~$파일명.xlsx` 잠금 파일이 생성되며, 저장하지 않은 변경 사항은 프로그램이 읽지 못한다. 읽는 시점과 저장 시점이 어긋나면 이전 내용을 읽고도 오류가 발생하지 않으므로, 결과를 보고 나서야 무엇이 반영되지 않았는지 확인하게 된다. 작업 전에 엑셀을 닫는다.

#### 21.1.4.8 일괄 치환은 문자열 리터럴까지 바꾼다

> ⚠ 경고
>
> `sed`로 컬럼명을 일괄 변경하면 코드의 문자열 리터럴과 주석까지 함께 변경된다. `STTS`를 `STTS_CD`로 바꾸면 `STTS.01`과 같이 바뀌면 안 되는 값도 함께 바뀐다. 치환 후에는 반드시 diff를 확인한다.

### 21.1.5 유형 5 · 정리가 절반만 되면 흔적이 남는다

이름 하나를 변경하거나 코드 하나를 폐기할 때 함께 정리해야 하는 곳을 일부만 처리한 경우이다. 남은 흔적은 오류를 일으키지 않고 조용히 남아 있다가 나중에 드러난다.

표 21.6. 정리가 절반만 되었을 때 남는 것

| 상황 | 남는 것 |
|---|---|
| 표준코드 폐기 | 코드상세가 고아 행으로 남음 |
| 표준코드 제거 | 표준용어의 `STDCD_ID` 참조가 남음 |
| 테이블 이름 변경 | 테이블 정의서만 변경하고 컬럼 정의서는 그대로 |
| 도메인 그룹·분류를 코드로 변경 | 표준용어의 도메인 참조는 그대로 |
| 도메인 그룹명 개명 | 엔터티 정의서의 이름 항목만 이전 이름 |
| 테이블 물리명 개명 | 메타파일 시트명이 이전 이름으로 남음 |

| 상황 | 남는 것 |
|---|---|
| 컬럼 물리명 개명 | 그 컬럼을 가리키는 `참조 컬럼` 항목은 그대로 |
| 테이블 이름 변경 후 DDL 재실행 | 이전 이름의 테이블이 데이터를 포함한 채 데이터베이스에 잔존 |

> 수정할 곳의 목록을 먼저 작성한다
>
> 이름 하나를 변경할 때 함께 변경해야 하는 항목은 22.2.5절 에 정리되어 있다. 변경을 시작하기 전에 이 목록으로 대상 지점을 모두 확인한다.
>
> 데이터베이스에 남은 이전 이름의 테이블은 `./dbms/rebuild.sh --prune`으로 정리한다.

#### 21.1.5.1 시트명은 헤더 검사만으로 확인되지 않는다

메타파일에서는 시트명이 곧 적재 대상 테이블의 이름이다. 리더와 적재기 모두 이를 전제로 동작한다. 그런데 시트명이 이전 이름으로 남아 있어도 헤더가 모두 일치하면 `meta-header` 규칙 3종은 통과한다. 파일을 읽는 데에도 문제가 없어 적재 시점에야 드러난다.

`meta-header.sheet-registered`가 시트명이 물리 정의서의 테이블 물리명과 일치하는지 확인한다. 테이블 물리명을 변경하였다면 이 규칙이 리포트에 포함되었는지 확인한다.

> 통과한 규칙의 수는 안전을 보장하지 않는다
>
> 검사하지 않은 것은 통과한 것이 아니다. 새 파일이나 새 계층을 추가할 때에는 「이 파일에 대하여 무엇을 검사하고 있는가」를 먼저 확인한다. 규칙이 여럿 통과하였다는 사실은 검사 범위 밖의 항목에 대하여 아무것도 말해 주지 않는다.
>
> 리포트에서 제외된 규칙을 확인하는 방법은 13.1.5절 에 있다.

## 21.2 빠른 진단표

오류 메시지가 발생하였을 때에는 표 21.7 에서 확인할 항목을 찾는다.

표 21.7. 증상별 진단표

| 증상 | 볼 곳 |
|---|---|
| `relation ... does not exist` | 식별자 대소문자. 21.1.4.1절 |
| `NotNullViolation` | 산출물의 Null 허용. `column.not-null-satisfied` |
| `StringDataRightTruncation` | 도메인 길이. `column.length-satisfied` |
| `ForeignKeyViolation` | 적재 순서. 15.3.2절 |
| DDL 실행 시 FK 거부 | 대상 기본키와 불일치. `column.foreign-key-target` |
| `'X' 테이블에 정의된 컬럼이 없다` | 테이블 이름 불일치. 9.4.2절 |
| 컬럼이 DDL에서 빠짐 | 소속 테이블 이름 불일치 |
| 값이 안내 없이 `None`으로 처리됨 | 모델 별칭과 헤더 불일치. 11.4.1절 |

| 증상 | 볼 곳 |
|---|---|
| `permission denied for database` | `CREATE SCHEMA` 권한. 15.6절 |
| `unbound variable` (셸) | bash 3.2의 빈 배열 |
| `적재할 수 없는 시트이다` | 로더가 인식하지 못하는 시트. 15.3.2절 |
| `'X' 시트가 없다` | 시트명이 테이블 물리명과 다름. `meta-header.sheet-registered` |
| 이전 이름의 테이블이 DB에 남음 | 개명 후 정리 누락. `./dbms/rebuild.sh --prune` |
| 규칙이 리포트에 없음 | 참조 사전 미제공. 13.1.5절 |
| `psycopg가 설치되어 있지 않습니다` | 선택 의존성이 제거됨. 21.1.4.2절 |
| `LLM 제안에는 추가 패키지가 필요하다` | 같은 원인. 21.1.4.2절 |
| `Bedrock을 쓰려면 uv sync --extra bedrock` | 같은 원인. 21.1.4.2절 |
| `Databricks를 진단하려면 uv sync --extra databricks` | 같은 원인. 21.1.4.2절 |
| `ENDPOINT_NOT_FOUND` (Databricks) | 웨어하우스 식별자와 호스트명. 21.1.4.5절 |
| `PERMISSION_DENIED` (Databricks) | 웨어하우스 `CAN USE` 권한. 21.1.4.5절 |
| Databricks 진단의 컬럼 수가 예상보다 적음 | 카탈로그·테이블 읽기 권한. 18.5.4.4절 |
| 진단의 준수 항목이 모두 `-` | 사전을 읽지 못함. 18.5.7절 |
| `모르는 DBMS이다` | `--dbms`는 postgresql·mysql·databricks. 18.5.2절 |
| Bedrock `on-demand throughput isn't supported` | 추론 프로파일 접두어. 12.4.3.1절 |
| `--from-db`와 `--meta` 결과가 다름 | 적재 지연. `kdsm load`를 다시 실행 |
| `ModuleNotFoundError: No module named 'kdsm'` | 다른 인터프리터의 잔재. `uv run`으로 실행 |

CHAPTER 22

# 작업 흐름

이 장은 명령을 어떤 차례로 실행하는지 보인다. 개별 옵션의 뜻은 각 명령을 다루는 장에 있다. `validate`는 표 13.8, `ddl`과 `drift`는 표 14.3 와 표 14.9, `load`는 표 15.5, `apply`는 표 15.6, `suggest`는 표 12.9, `diagnose`는 표 18.24, `export`는 표 17.7, `erd`는 표 16.7 를 참고한다. 아홉 명령을 한자리에 모은 요약은 19장 에 있다.

## 22.1 처음부터 끝까지

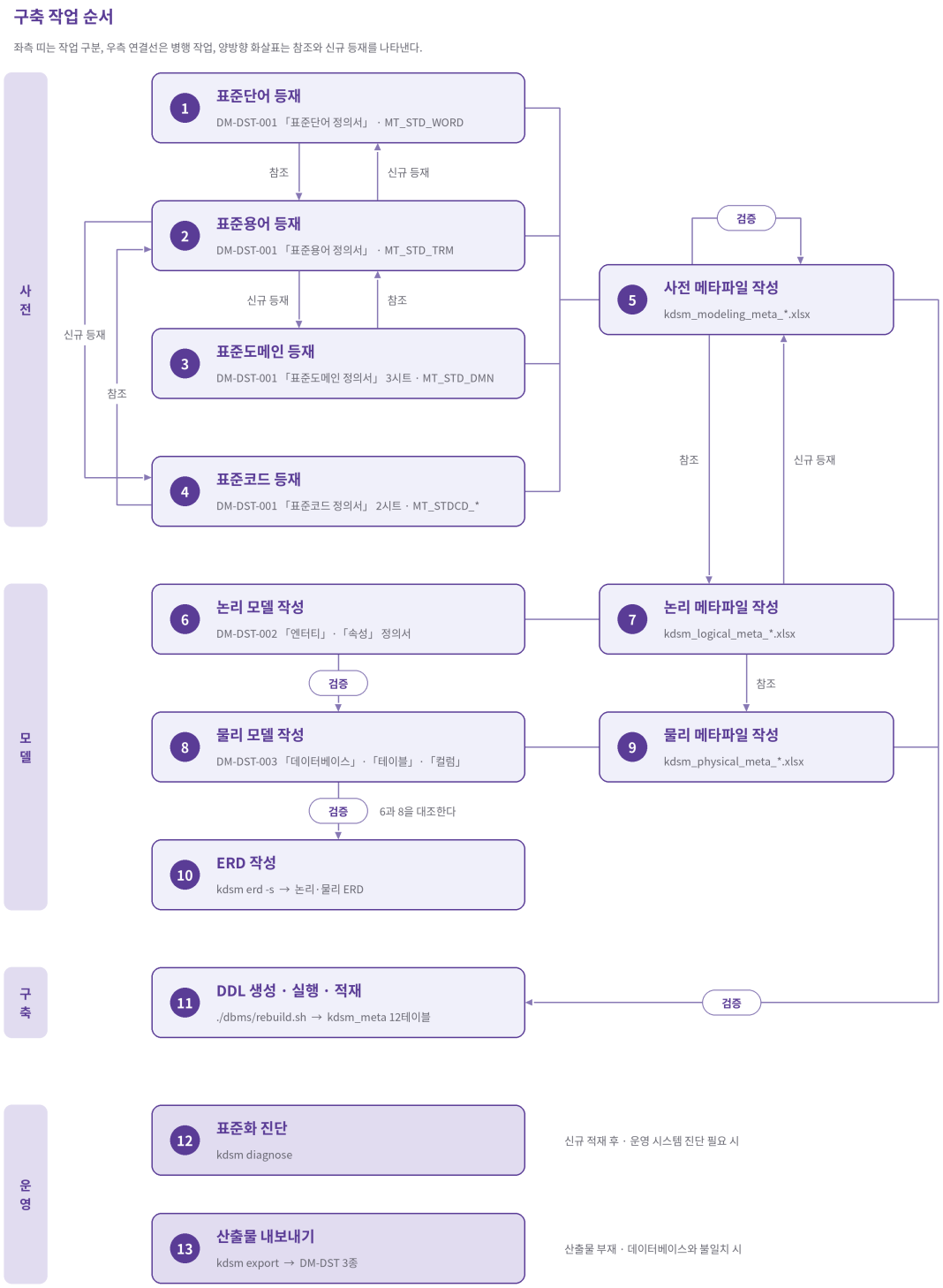


그림 22.1. 구축 작업 순서

각 단계 사이의 검증이 다음 단계로 문제가 이월되는 것을 방지한다. 6과 8은 동일한 내용을 두 번 기록하므로 서로 대조한다(13.3.8절). 10의 ERD는 작성한 산출물에서 생성하는 포워드 엔지니어링이다(16.1.1절). 11은 `dbms/rebuild.sh` 하나로 통합되어 있다(15.6절).

메타파일 작성(5·7·9)은 해당 작업 과정에 넣지 않고 오른쪽에 나란히 두었다. 산출물과 메타파일은 동일한 내용을 사람이 읽는 서식과 프로그램이 읽는 서식으로 각각 기록한 것이므로, 선행 작업의 완료 이후가 아니라 해당 작업과 동시에 수행하기 때문이다. 화살표가 아니라 연결선으로 이은 것도 같은 이유이다.

양방향으로 연결한 지점이 4개이다. 1과 2, 2와 3, 2와 4, 그리고 5와 7이다. 선행 항목을 참조하다가 미등재 항목을 발견하면 그 시점에 등재하고 원래 작업으로 복귀한다는 의미이다. 화살표는 참조되는 쪽에서 참조하는 쪽을 향하므로 참조와 신규 등재의 방향이 서로 반대가 된다. 2에서 4로 향하는 방향만 3을 건너뛰므로 좌측으로 우회한다. 3과 4 사이에 화살표를 두지 않은 것도 같은 이유이다. 표준코드는 표준도메인의 후속 단계가 아니라 표준용어에서 파생되는 단계이다.

7에서 9로 향하는 화살표는 논리 메타파일을 참조하여 물리 메타파일을 작성한다는 의미이다. 5·7·9는 모두 11의 적재로 연결되며, 11로 진입하는 경로는 이것뿐이다. 데이터베이스에 적재하는 대상은 메타파일이며 산출물이나 ERD가 아니기 때문이다. 마지막 검증도 이 경로에 있다.

5의 순환 화살표는 사전 메타파일을 작성한 직후에 검증하고 수정하여 다시 작성한다는 의미이다. 4에서 6으로 직접 연결하지 않은 것은 사전의 완성을 기다린 뒤에 모델링을 시작하는 것이 아니기 때문이다. 사전과 모델은 5와 7 사이의 양방향 연결로 이어진다.

> **i 실제로 작업이 시작되는 시점은 「6. 논리 모델 작성」이다**
>
> 번호는 1에서 시작하지만 아무런 기준 없이 표준단어부터 등재하는 경우는 드물다. 무엇을 등재할 것인지는 작성할 모델이 결정하기 때문이다. 논리 모델을 작성하다가 필요한 용어를 확인하고, 그 용어를 정의하기 위하여 단어와 도메인과 코드를 조회하며, 미등재 항목은 그 시점에 등재한다. 양방향 화살표가 이 반복을 나타낸 것이다.
>
> 따라서 1부터 4까지는 6과 별개로 수행하는 준비 작업이 아니라 6이 요구하는 작업이다. 사전을 먼저 완성한 뒤에 모델링을 시작하면 사용되지 않을 단어까지 등재하게 된다.

산출물과 메타파일이 각각 3종인 것은 대상 영역이 3개이기 때문이다. 데이터 표준·논리 모델·물리 모델이다. 산출물 1종이 메타파일 1종에 대응하며, 이를 적재하면 복수의 메타테이블이 된다.

**표 22.1.** 영역별 산출물과 메타파일

| 영역 | 산출물 | 메타파일 | 메타테이블 |
|---|---|---|---|
| 데이터 표준 | DM-DST-001 | `kdsm_modeling_meta` | `MT_STD_*` 7 |
| 논리 모델 | DM-DST-002 | `kdsm_logical_meta` | `MT_ENTTY` · `MT_ATRB` |
| 물리 모델 | DM-DST-003 | `kdsm_physical_meta` | `MT_DB` · `MT_TBL` · `MT_COL` |

KDSM 시스템 관리자가 메타테이블의 초기 데이터를 구축하는 경우에는 이 자리에 DM-LIB 6종이 해당한다. 영역을 논리와 물리로 한 번 더 구분한 문서이며, 두 계열의 차이는 9.1절 에 있다.

1부터 11까지는 신규 구축 작업이고 12와 13은 구축 완료 이후의 작업이다. 이 둘은 화살표로 연결하지 않고 수행 시점만 오른쪽에 기술하였다. 선행 단계를 완료하였다고 연속하여 수행하는 작업이 아니라 필요가 발생하였을 때 수행하는 작업이기 때문이다. 운영이 시작되면 방향이 한 번 더 바뀐다. 문

서가 기준이던 관계가 데이터베이스가 기준인 관계로 바뀌고, 그때는 문서를 수작업으로 맞추는 대신 13의 `kdsm export`로 다시 생성한다(17장). 검증은 통과와 실패를 판정하고 진단은 비율을 산출한다(18장). 기존 시스템을 인수한 경우에는 12부터 시작해도 된다. 무엇이 얼마나 어긋나 있는지 파악해야 1단계의 사전을 무엇으로 채울지 결정할 수 있기 때문이다.

## 22.2 자주 하는 일

표준용어를 등재하고 산출물을 수정하는, 일상적으로 반복되는 작업이다.

### 22.2.1 새 표준용어를 등재할 때

```
kdsm suggest 고객여신금액 --meta dbms/meta/meta_data/kdsm_modeling_meta_20260828.xlsx
kdsm suggest 고객여신금액 --from-db --coin       # 적재본에서 읽고 신규 단어까지 제안
```

1. 후보 중 `*`로 표시된 항목을 선택한다
2. `이름:` 줄이 있으면 이름이 변경된 후보이다. 금지어를 대체어로 변경한 것이므로 그대로 수용한다
3. `+` 표시는 사전에 없는 단어를 사용한 후보이다. 해당 단어를 먼저 등재할지 결정한다
4. 도메인 후보가 여럿이면 길이를 검토하여 선택한다
5. 미등록 단어가 남았으면 표준단어를 먼저 등재할지 판단한다
6. `MT_STD_TRM` 시트에 행을 추가한다
7. 용어구분(01/02/03)을 정한다
8. 검증을 실행한다

> **i 신규 표준단어를 먼저 등재할 때**
>
> `--coin`이 생성한 초안은 화면에만 출력된다. `MT_STD_WORD` 시트에 직접 옮겨 적고 상태를 후보(`01`)로 지정한다. 이후 검증을 실행하여 통과하면 확정(`02`)으로 변경하고, 그때 비로소 해당 단어를 사용하는 표준용어를 등재한다.

> **용어구분은 비워 두지 않는다**
>
> 비어 있으면 속성명 여부를 판단할 수 없어 `ends-classifier`와 `domain-registered` 검사를 건너뛴다. 검사를 수행하지 않은 상태로 통과하게 된다.

### 22.2.2 산출물을 수정할 때

```
# 1. 이전 버전 보관
mv DM-DST-003_물리데이터모델_*.xlsx backup/

# 2. 새 버전으로 저장하고 표지 시트의 버전·작성일을 맞춘다

# 3. 검증
kdsm validate DM-DST-003_물리데이터모델_....xlsx --meta 사전메타파일.xlsx

# 4. DDL 재생성
kdsm ddl DM-DST-003_물리데이터모델_....xlsx -o ddl/schema.sql --drop
```

### 22.2.3 산출물을 데이터베이스에 반영할 때

프로젝트가 작성한 DM-DST 3종을 메타테이블에 추가한다. 사전 자체를 구축하는 `kdsm load`와 달리, 이미 구축된 사전에 프로젝트의 표준과 모델을 더하는 작업이다(15.5절).

```
# 1. 세 산출물을 함께 검증한다
kdsm validate app/deliverable/DM-DST-00{1,2,3}_*.xlsx --meta 사전메타파일.xlsx

# 2. 반영한다. 기존 등재분과 충돌하면 그 자리에서 멈춘다
kdsm apply app/deliverable/DM-DST-00{1,2,3}_*.xlsx --actor 작성자 --name-prefix AP_
```

검증을 먼저 수행하는 것은 반영이 여러 테이블에 걸쳐 이루어지기 때문이다. 반영 도중에 중단되면 어디까지 반영되었는지를 사람이 확인해야 한다.

### 22.2.4 이미 있는 데이터베이스를 문서로 옮길 때

표준화를 거치지 않은 데이터베이스가 이미 운영되고 있는 경우이다. 문서가 없으므로 검증할 대상 자체가 없다. 데이터베이스의 현재 구조를 정의서 서식에 담아 출발점을 만든다.

```
# 1. 현재 구조를 물리 데이터 모델 정의서로 옮긴다
kdsm export --from-catalog --dbms databricks --catalog dev_landing \
  --meta 사전메타파일.xlsx -o document/out

# 2. 표준을 어느 정도 준수하는지 측정한다
kdsm diagnose --dbms databricks --catalog dev_landing \
  --meta 사전메타파일.xlsx -o 진단.xlsx

# 3. 구조를 도면으로 확인한다
kdsm erd --from-catalog --dbms databricks --catalog dev_landing -f png
```

세 명령이 같은 카탈로그 리더를 사용하므로 접속 정보와 대상 표기가 동일하다. 다만 `export`는 스키마를 하나만 지정한다. 카탈로그 전체를 한 권에 담기 때문이다. 1이 만든 정의서에는 한글명이 비어 있는 행이 남는다. 2의 진단 결과가 그 행을 어떤 순서로 채워야 하는지를 알려 준다.

내보낸 정의서를 그대로 산출물로 삼지 않는다. 버전 `0.0.0`, 작성자 공란으로 기록되는 것도 이 때문이다. 사람이 검토하여 한글명과 관련 엔터티를 채우고, 표준에 어긋나는 이름을 정리한 뒤에 판을 올린다.

### 22.2.5 이름 변경 시 함께 변경할 항목

#### 22.2.5.1 표준단어의 영문약어를 변경하는 경우

- ☐ 그 단어를 구성단어로 사용하는 모든 표준용어의 물리명
- ☐ 그 표준용어를 사용하는 모든 컬럼 정의서의 컬럼 물리명
- ☐ 이미 생성한 테이블의 컬럼명 ( `ALTER TABLE ... RENAME COLUMN` )
- ☐ 메타파일 시트의 헤더 (그 용어가 메타 컬럼인 경우)
- ☐ `kdsm.models`의 `alias`

#### 22.2.5.2 표준용어의 물리명을 변경하는 경우

- ☐ 컬럼 정의서의 컬럼 물리명
- ☐ 메타파일 시트 헤더
- ☐ `kdsm.models`의 `alias`
- ☐ 참조하는 컬럼의 참조 컬럼 항목

#### 22.2.5.3 표준코드를 폐기하는 경우

- ☐ `MT_STDCD_MSTR`의 마스터 행

- ☐ `MT_STDCD_DTL`의 코드값 행
- ☐ `MT_STD_TRM`의 `STDCD_ID`
- ☐ 컬럼 정의서의 참조 코드 항목

#### 22.2.5.4 테이블 이름을 변경하는 경우

- ☐ 테이블 정의서의 테이블 물리명
- ☐ 컬럼 정의서의 소속 테이블 (전 행)
- ☐ 컬럼 정의서의 참조 테이블 (그 테이블을 가리키는 모든 행)
- ☐ 메타파일 시트명
- ☐ `kdsm.constants.Table`

#### 22.2.5.5 도메인 길이를 변경하는 경우

- ☐ `MT_STD_DMN`의 도메인명 (`내용V40` → `내용V80`)
- ☐ `MT_STD_DMN`의 `DMN_LEN`·`PHYS_DATA_TYPE_CN`
- ☐ 그 도메인을 사용하는 표준용어의 `STD_DMN_NM`
- ☐ 컬럼 정의서의 길이
- ☐ 이미 생성한 테이블 (`ALTER TABLE ... ALTER COLUMN ... TYPE`)

## 22.3 점검

정기적으로 실행해야 할 작업과, 일상적인 검사에서 제외되는 항목을 확인하는 방법이다.

### 22.3.1 정기 점검

```
kdsm validate DM-DST-00{1,2,3}_*.xlsx --meta 사전메타파일.xlsx -q
```

한 줄 요약이 `오류 0건`이면 정상이다. CI에 등록해 두면 산출물을 수정할 때마다 자동으로 실행된다 (19장).

검증은 문서와 문서를 대조하는 일이다. 운영 데이터베이스가 그 문서를 따르고 있는지는 별개의 문제이므로, 적재 이후에는 대조도 함께 실행한다.

```
kdsm drift -s DM-DST-003_물리데이터모델_....xlsx -q
```

한 줄 요약이 `차이 없음`이면 정의서와 데이터베이스가 일치한다. 차이가 있으면 어느 쪽을 고칠지 판단해야 한다. 판단 기준은 14.9.2절 에 있다.

### 22.3.2 표준화 수준 진단

산출물이 아니라 실제로 운영 중인 데이터베이스를 대상으로 한다. 매월 한 번 정도 실행하여 전월과 비교하면 표준화 수준이 개선되고 있는지 확인할 수 있다.

```
kdsm diagnose --schema kdsm_meta -o 진단.xlsx --qmd 진단.qmd
```

레거시 스키마를 진단할 때에는 접두어로 대상을 변경한다. 표준 사전은 `kdsm_meta`에서 그대로 읽는다 (18.5.7절).

```
kdsm diagnose --env-prefix ECODI_DB_ --schema ecodi_meta -o ecodi_진단.xlsx
kdsm diagnose --dbms databricks --schema main.sales -o dbx_진단.xlsx
```

CI에 등록하려면 `--min-score`를 지정한다. 이 경우에만 종료 코드가 1이 될 수 있다.

```
kdsm diagnose --schema kdsm_meta --min-score 95 -q
```

> 기준선은 지금 점수에서 시작한다
>
> 처음부터 90점을 기준으로 설정하면 매번 실패하여 결과를 확인하지 않게 된다. 현재 점수를 기준선으로 설정하고 개선할 때마다 단계적으로 상향하는 것이 적절하다. 목적은 기준 통과가 아니라 상태가 악화되지 않도록 하는 것이다.

### 22.3.3 폐기 항목까지 확인할 때

정기적으로 폐기 항목의 잔재를 확인한다. 폐기 상태가 문제를 가리는 경우가 있다.

```
from kdsm.validate.rules import UniqueAbbreviationRule, code_rules

UniqueAbbreviationRule(include_discarded=True)(words)
[r for rule in code_rules(details) for r in rule(masters)]   # 폐기 마스터도
```

> 노트
>
> 폐기 상태의 코드는 폐기 항목 제외 규칙 때문에 일반 검사에서 빠진다. 상세가 없거나 참조가 남아 있어도 위반으로 보고되지 않으므로, 정기 점검에서는 폐기 항목까지 포함하여 실행한다(5.2.1절).

## 22.4 환경 구축과 남은 일

처음 환경을 구축하는 절차와 아직 구현하지 않은 항목이다.

### 22.4.1 처음 환경을 구축할 때

설치해야 할 프로그램은 20장 에 있다. 여기에서는 설치가 완료된 상태를 전제로 데이터베이스를 구축하는 절차만 다룬다. 이 절은 KDSM 시스템 관리자의 작업이므로 DM-LIB 산출물을 적재한다(9.1절).

절차는 네 단계이며, PostgreSQL·MySQL·Databricks 가 모두 같다.

표 22.2. 환경 구축의 네 단계

| 단계 | 하는 일 | 수단 |
|---|---|---|
| 1 | 라이브러리를 설치한다 | `uv sync` |
| 2 | 사용자와 스키마를 만들고 권한을 준다 | `dbms/script/init.sh` |
| 3 | 메타테이블 12개를 만든다 | `dbms/rebuild.sh` 가 `kdsm ddl` 로 |
| 4 | 메타파일 3종을 적재한다 | `dbms/rebuild.sh` 가 `kdsm load` 로 |

대상마다 다른 것은 2단계와 접속 정보의 환경변수 이름뿐이다. 3단계와 4단계는 `dbms/rebuild.sh` 가 `--dbms` 로 갈라 준다. 이 스크립트는 DDL을 만들기 전에 산출물을 먼저 검증하므로, 어긋난 정의가 스키마에 굳는 일을 막는다.

> ! 2단계는 관리자 계정으로 실행한다
>
> 사용자와 데이터베이스를 만드는 작업이므로 그 권한을 가진 계정으로 접속해야 한다. PostgreSQL은 보통 `postgres`, MySQL은 `root`이며, Databricks는 카탈로그를 만들 수 있는 메타스토어 관리자이다.
>
> 3단계 이후는 2단계에서 만든 `kdsm` 계정으로 수행한다. 권한 층위가 다르므로 두 단계를 한 계정으로 묶지 않는다.

> ⚠ 비밀번호는 인자로 주지 않는다
>
> `init.sh`와 `rebuild.sh`는 비밀번호와 토큰을 인자로 받지 않는다. 명령줄에 적은 값은 `ps` 출력과 셸 기록에 남기 때문이다(20.10.1절). 아래의 예에서 비밀번호를 환경변수로 지정한 것은 그 때문이다. 값을 화면과 기록에 남기지 않고 넣는 방법은 20.10.2절 에 있다.

#### 22.4.1.1 PostgreSQL

```
# 1. 라이브러리 (쓸 기능만 골라도 되고, 판단이 어려우면 --all-extras)
cd library && uv sync --all-extras && cd ..

# 2. 사용자와 스키마 — 관리자로 접속한다
export PGPASSWORD=...              # 접속할 관리자(postgres)의 비밀번호
export KDSM_INIT_PASSWORD=...      # 새로 만들 kdsm 사용자의 비밀번호
./dbms/script/init.sh --db aiplatform --schema kdsm_meta --user kdsm

# 3. 접속 정보 — 여기부터는 kdsm 계정이다
unset PGPASSWORD KDSM_INIT_PASSWORD
export KDSM_DB_HOST=localhost KDSM_DB_PORT=5432 KDSM_DB_NAME=aiplatform
export KDSM_DB_SCHEMA=kdsm_meta KDSM_DB_USER=kdsm KDSM_DB_PASSWORD=...

# 4. 테이블 생성과 메타파일 적재
./dbms/rebuild.sh
```

#### 22.4.1.2 MySQL

```
cd library && uv sync --extra mysql && cd ..

export MYSQL_HOST=localhost MYSQL_PORT=3306
export MYSQL_PWD=...               # 접속할 관리자(root)의 비밀번호
export KDSM_INIT_PASSWORD=...      # 새로 만들 kdsm 사용자의 비밀번호
./dbms/script/init.sh --dbms mysql --schema kdsm_meta --user kdsm --host '%'

unset MYSQL_PWD KDSM_INIT_PASSWORD
export MYSQL_USER=kdsm MYSQL_PASSWORD=... KDSM_DB_SCHEMA=kdsm_meta
./dbms/rebuild.sh --dbms mysql
```

MySQL에는 데이터베이스 안의 스키마가 없다. 데이터베이스가 곧 스키마이므로, 다른 DBMS의 `kdsm_meta` 스키마에 해당하는 것이 여기에서는 `kdsm_meta` 데이터베이스이다. `--schema`와 `KDSM_DB_SCHEMA`를 그대로 사용하며, 라이브러리도 그 값을 데이터베이스명으로 해석한다.

`--host`는 그 사용자가 어디에서 접속할 수 있는지를 정한다. MySQL의 계정은 이름과 접속 위치의 쌍이므로 `kdsm@localhost`와 `kdsm@%`는 서로 다른 계정이다.

#### 22.4.1.3 Databricks

```
cd library && uv sync --extra databricks && cd ..

export DATABRICKS_HOSTNAME=adb-1234.azuredatabricks.net
export DATABRICKS_HTTP_PATH=/sql/1.0/warehouses/abc123
export DATABRICKS_CLIENT_ID=... DATABRICKS_CLIENT_SECRET=...
export DATABRICKS_CATALOG=dev_landing KDSM_DB_SCHEMA=kdsm_meta

# 2. 카탈로그·스키마·권한 — 메타스토어 관리자로 접속한다
./dbms/script/init.sh --dbms databricks \
  --catalog dev_landing --schema kdsm_meta --principal kdsm@example.com

# 3-4. 테이블 생성과 메타파일 적재
./dbms/rebuild.sh --dbms databricks
```

Databricks에는 사용자를 만드는 SQL이 없다. 계정 콘솔이나 SCIM으로 등록해 둔 사용자·서비스 프린시펄·그룹에 권한을 줄 뿐이다. 그래서 `--user` 대신 `--principal`을 지정하며, 그 값은 이미 존재하는 주체의 이름이어야 한다.

이름이 세 단이므로 카탈로그를 함께 지정한다. 지정하지 않으면 웨어하우스의 기본 카탈로그에 들어가므로, 의도한 곳인지 확인하기 전에는 지정해 두는 편이 안전하다.

#### 22.4.1.4 초기화 스크립트가 하는 일

`dbms/script/init.sh`는 대상에 맞는 SQL 스크립트를 골라 값을 넘긴다. 값을 넘기는 방법이 클라이언트마다 다르므로, 스크립트도 대상마다 따로 두었다.

표 22.3. 초기화 스크립트 3종

| 대상 | 스크립트 | 인수를 받는 방법 |
|---|---|---|
| PostgreSQL | `init_postgresql.sql` | `psql -v` 로 넘긴 값을 `:변수` 로 읽는다 |
| MySQL | `init_mysql.sql` | 앞서 지정한 사용자 변수(`@변수`)를 읽는다 |
| Databricks | `init_databricks.sql` | 앞에 붙인 `DECLARE VARIABLE` 을 읽는다 |

세 스크립트 모두 이름을 문장에 이어 붙이므로 인용 규칙을 각 DBMS의 함수에 맡긴다. PostgreSQL은 `format()`의 `%I`와 `%L`을, MySQL은 `QUOTE()`와 백틱 이중화를, Databricks는 백틱 이중화를 사용한다. 인용을 손으로 적으면 따옴표가 든 값에서 문장이 깨진다.

세 스크립트 모두 여러 번 실행해도 결과가 같다. 사용자가 이미 있으면 비밀번호를 덮어쓰지 않는다. 두 번 실행한 것만으로 운영 중인 접속이 끊기는 일을 막기 위한 것이다.

> **i** MySQL만 권한 부여가 두 번에 나뉜다
>
> MySQL이 `PREPARE`로 만들 수 있는 문장의 목록에 `GRANT`는 역할을 주는 형태만 들어 있고, 권한을 주는 형태는 빠져 있다. 그래서 스키마명을 변수로 넘길 방법이 없다.
>
> `init_mysql.sql`은 실행할 `GRANT` 문장을 결과로 한 줄 내놓는 데까지만 하고, `init.sh`가 그것을 그대로 되돌려 보낸다. 스크립트를 손으로 실행할 때에는 출력된 문장을 복사하여 한 번 더 실행한다.

> **무엇을 보낼지 먼저 확인한다**
>
> `--dry-run`을 지정하면 데이터베이스에 접속하지 않고 보낼 문장만 보여 준다.
>
> ```
> ./dbms/script/init.sh --dbms databricks --catalog dev_landing \
>   --principal kdsm@example.com --dry-run
> ```

### 22.4.2 진단 대상을 추가할 때

표준 사전과 별개로, 진단 대상 데이터베이스마다 접속 정보를 별도로 관리한다. 대상별로 접두어를 정의해 두면 명령이 간결해지고 비밀번호가 셸 설정 한 곳에 모이다.

```
# 레거시 PostgreSQL
export ECODI_DB_HOST=... ECODI_DB_NAME=ecodi ECODI_DB_SCHEMA=ecodi_meta
export ECODI_DB_USER=... ECODI_DB_PASSWORD=...

# Databricks Unity Catalog
export DATABRICKS_SERVER_HOSTNAME=adb-1234.azuredatabricks.net
export DATABRICKS_HTTP_PATH=/sql/1.0/warehouses/abc123
export DATABRICKS_TOKEN='dapi...'
export DATABRICKS_CATALOG=main

# LLM 제안을 Bedrock으로 받는다면
export KDSM_LLM_PROVIDER=bedrock
export AWS_ACCESS_KEY_ID=... AWS_SECRET_ACCESS_KEY=... AWS_REGION=ap-northeast-2
```

전체 목록은 참고 자료의 환경변수 절에 있다.

### 22.4.3 지금 자동화되지 않는 것

다음 항목은 프로그램이 처리하지 않으므로 사람이 수행한다. 자동화 여부를 미리 알고 있어야 작업 계획에서 누락되지 않는다.

**표 22.4.** 사람이 수행하는 작업

| 항목 | 사람이 수행할 일 |
|---|---|
| 산출물 표지의 문서명과 작성자 | 파일명과의 대조는 `kdsm validate`가 문서번호·버전·작성일 세 가지만 수행한다. 문서명과 작성자는 대조 대상이 아니므로 사람이 확인한다. 대조 결과는 INFO로 보고하여 검증을 통과시키므로, 고칠지 여부도 사람이 정한다(9.2절) |
| 한 엔터티를 여러 테이블로 나눈 경우 | 대조 규칙이 마지막 테이블만 확인하므로 나머지 테이블은 직접 확인한다(9.4.3절) |
| 추천 결과의 등재 | `kdsm suggest`의 제안을 시트에 직접 기재한다. 무엇을 등재할지는 사람의 판단이 우선하기 때문이다(12.4.3절) |
| DM-LIB 6종 | 데이터베이스를 읽어 산출물 엑셀을 생성하는 것은 DM-DST 3종만 지원한다. DM-LIB 은 시스템 관리자가 작성하는 초기 데이터이므로 생성 대상에 넣지 않는다(9.1절) |

### `table.entity-registered` 는 논리 모델을 함께 지정해야 동작한다

테이블 정의서의 `관련 엔터티`를 쉼표로 분리하여 엔터티 사전과 대조하는 규칙이다. 엔터티와 테이블이 1:1이 아닌 경우에는 외래키를 설정할 수 없으므로, 데이터베이스가 제약하지 못하는 부분을 이 규칙이 대신 확인한다.

```
kdsm validate --meta <사전메타> --logical <논리메타> -P <물리메타>
```

논리 모델을 지정하지 않으면 이 규칙은 리포트에서 제외된다. 검사하지 않은 항목을 통과한 것으로 보고하지 않기 위해서이다(13장). 리포트에 이 규칙이 보이지 않으면 논리 모델을 지정하지 않은 것이다.

# 참고 자료

## 저장소 문서

표 0.1. 저장소 문서

| 문서 | 내용 |
|---|---|
| `README.md` | 저장소 전체 안내 |
| `library/README.md` | `kdsm` 패키지 설치와 사용법 |
| `library/docs/validation-rules.md` | 검증 규칙 레퍼런스 |
| `library/src/kdsm/templates/` | DM-DST 산출물 3종의 빈 템플릿. 패키지와 함께 배포한다 |
| `document/kdsm_메타데이터_관계도.svg` | 엔터티 관계도 (`.png` 사본을 매뉴얼에 게재) |
| `document/areas.toml` | ERD 영역 구분의 대비책. 정본은 주제영역 배정이다 |
| `dbms/rebuild.sh` | 검증 → DDL 생성 → 적용 → 적재 스크립트 |

다음은 이 저장소의 코드와 문서를 수정하는 사람을 위한 것이다.

표 0.2. 저장소 관리 문서

| 문서 | 내용 |
|---|---|
| `document/writing-style.md` | 문장 작성 규약 |
| `document/check-writing.py` | 문장 작성 규약 위반 검사 |
| `document/check-crossref.py` | 교차참조 검사 |
| `document/check-numbers.py` | 문서에 적힌 수와 실물 대조 |

| 문서 | 내용 |
|---|---|
| `document/engineering-notes.md` | 라이브러리 구현과 문서 조판의 기록 |
| `document/plan/roadmap.md` | 남은 작업과 구현하지 않기로 한 항목 |

## 산출물

사용자가 작성하는 산출물은 DM-DST-001·002·003 이다(9장). 빈 템플릿이 `kdsm` 패키지 안에 있다.

**표 0.3.** DM-DST 템플릿과 버전

| 문서번호 | 템플릿 | 버전 |
|---|---|---|
| DM-DST-001 | 데이터표준정의산출물템플릿 | V0.9.4 |
| DM-DST-002 | 논리데이터모델산출물템플릿 | V0.9.10 |
| DM-DST-003 | 물리데이터모델산출물템플릿 | V0.9.4 |

아래 표는 KDSM 메타테이블의 초기 데이터를 정의한 시스템 관리자 영역의 산출물이다. 사용자가 작성하는 대상이 아니다.

**표 0.4.** DM-LIB 문서번호와 버전

| 문서번호 | 이름 | 버전 |
|---|---|---|
| DM-LIB-001 | 논리데이터모델 - 데이터표준정의서 | V0.9.11 |
| DM-LIB-002 | 논리데이터모델 - 논리데이터모델정의서 | V0.9.11 |
| DM-LIB-003 | 논리데이터모델 - 물리데이터모델정의서 | V0.9.10 |
| DM-LIB-004 | 물리데이터모델 - 데이터표준정의서 | V0.9.7 |
| DM-LIB-005 | 물리데이터모델 - 논리데이터모델정의서 | V0.9.10 |
| DM-LIB-006 | 물리데이터모델 - 물리데이터모델정의서 | V0.9.4 |

논리 정의서와 물리 정의서가 영역별로 짝을 이룬다. 001↔004는 데이터 표준, 002↔005는 논리 모델, 003↔006은 물리 모델이다. 같은 사실을 두 문서에 기록하므로 서로 대조한다(13.3.8절).

## 메타파일

**표 0.5.** 메타파일 3종

| 파일 | 시트 | 작성 근거 |
|---|---|---|
| `dbms/meta/meta_data/kdsm_modeling_meta_*.xlsx` | `MT_STD_*` 7 | 수작업 등재 |
| `dbms/meta/meta_data/kdsm_logical_meta_*.xlsx` | `MT_ENTTY` · `MT_ATRB` | DM-DST-002 |

| 파일 | 시트 | 작성 근거 |
|---|---|---|
| `dbms/meta/meta_data/kdsm_physical_meta_*.xlsx` | `MT_DB` · `MT_TBL` · `MT_COL` | DM-DST-003 |

## 외부 자료

행정안전부, 「공공데이터 공통표준도메인」, 2025-11-01. 표준도메인 사전 구축의 기준 자료이다. 공공데이터포털에서 내려받는다. https://www.data.go.kr/data/15156442/fileData.do

행정안전부, 「공공데이터 공통표준단어」, 2025-11-01. https://www.data.go.kr/data/15156439/fileData.do

행정안전부, 「공공데이터 공통표준용어」, 2025-11-01. https://www.data.go.kr/data/15156379/fileData.do

세 자료는 저장소에 두지 않는다. 개정될 때마다 원본을 다시 내려받아 사전과 대조한다. 공통표준에서 가져온 낱말이 사전에 다수 포함되어 있으므로, 개정 내용을 확인하지 않으면 공통표준과 이 저장소의 사전이 갈린다(3.2.2절).

# 물리 테이블 12개

## 데이터 표준 영역

표 0.6. 데이터 표준 영역 테이블

| 테이블 | 엔터티 | 식별자 |
|---|---|---|
| `MT_STD_WORD` | 표준단어 | 표준단어명 |
| `MT_STD_TRM` | 표준용어 | 표준용어명 |
| `MT_STD_DMN` | 표준도메인 | 표준도메인명 |
| `MT_STD_DMN_GROUP` | 표준도메인그룹 | 표준도메인그룹코드 |
| `MT_STD_DMN_CLSF` | 표준도메인분류 | 표준도메인분류코드 |
| `MT_STDCD_MSTR` | 표준코드마스터 | 표준코드아이디 |
| `MT_STDCD_DTL` | 표준코드상세 | 표준코드아이디 + 표준코드값내용 |

## 논리 데이터 모델 영역

표 0.7. 논리 데이터 모델 영역 테이블

| 테이블 | 엔터티 | 식별자 |
|---|---|---|
| `MT_ENTTY` | 엔터티 | 엔터티명 |
| `MT_ATRB` | 속성 | 엔터티명 + 속성명 |

## 물리 데이터 모델 영역

표 0.8. 물리 데이터 모델 영역 테이블

| 테이블 | 엔터티 | 식별자 |
|---|---|---|
| MT_DB | 데이터베이스 | 데이터베이스명 + 스키마명 |
| MT_TBL | 테이블 | 테이블물리명 |
| MT_COL | 컬럼 | 테이블물리명 + 컬럼물리명 |

# 표준코드 값

## STTS 상태

표 0.9. STTS 상태 코드값

| 값 | 이름 |
|---|---|
| 01 | 후보 |
| 02 | 확정 |
| 03 | 폐기 |

## WORD_SE 단어구분

표 0.10. WORD_SE 단어구분 코드값

| 값 | 이름 |
|---|---|
| 01 | 일반어 |
| 02 | 도메인어 |
| 03 | 분류어 |

## TRM_SE 용어구분

표 0.11. TRM_SE 용어구분 코드값

| 값 | 이름 |
|---|---|
| 01 | 엔터티명 |
| 02 | 속성명 |
| 03 | 엔터티명·속성명 |

### ENTTY_CLSF 엔터티구분

표 0.12. ENTTY_CLSF 엔터티구분 코드값

| 값 | 이름 |
|---|---|
| 01 | 기본엔터티 |
| 02 | 중심엔터티 |
| 03 | 행위엔터티 |

### STD_DMN_GROUP_CD 도메인그룹

표 0.13. STD_DMN_GROUP_CD 도메인그룹 코드값

| 값 | 이름 |
|---|---|
| 01 | 금액 |
| 02 | 날짜/시간 |
| 03 | 내용 |
| 04 | 명칭 |
| 05 | 번호 |
| 06 | 수량 |
| 07 | 율 |
| 08 | 코드 |

도메인분류(`STD_DMN_CLSF_CD`)는 그룹 코드 두 자리에 분류 두 자리를 이어 붙인 네 자리이다. 금액 그룹의 `0102` 가 그러하다(4장).

## 환경변수

이 절은 변수의 이름과 기본값을 모은 것이다. 어느 명령이 무엇을 필요로 하는지와 값을 어디에 두어야 하는지는 20.9절 에 있다.

### PostgreSQL 접속

표 0.14. 접속 정보 환경변수

| 변수 | 기본값 |
|---|---|
| `KDSM_DB_HOST` | `localhost` |
| `KDSM_DB_PORT` | `5432` |
| `KDSM_DB_NAME` | `kdsm` |
| `KDSM_DB_SCHEMA` | `kdsm_meta` |

| 변수 | 기본값 |
|---|---|
| `KDSM_DB_USER` | `kdsm` |
| `KDSM_DB_PASSWORD` | 없음 |

접두어 `KDSM_DB_`는 변경할 수 있다. `kdsm diagnose --env-prefix ECODI_DB_`와 같이 지정하면 `ECODI_DB_HOST`부터 `ECODI_DB_PASSWORD`까지를 읽는다. 진단 대상과 표준 사전이 서로 다른 서버에 있을 때 사용한다. 사전 쪽은 `--dict-env-prefix`가 지정하며 기본값은 `KDSM_DB_`이다(18.5.7절).

비밀번호는 어느 명령에서도 인자로 받지 않는다. 명령줄에 기재하면 같은 서버에 접속한 다른 사용자의 `ps` 출력과 셸 기록에 남는다. 그 배경과 값을 노출하지 않고 지정하는 방법은 20.10절 에 있다.

## MySQL 접속 (`--dbms mysql`)

**표 0.15.** MySQL 접속 환경변수

| 변수 | 기본값 |
|---|---|
| `MYSQL_HOST` | `localhost` |
| `MYSQL_PORT` | `3306` |
| `MYSQL_DATABASE` | 없음. `MYSQL_NAME`도 참조한다 |
| `MYSQL_USER` | 없음 |
| `MYSQL_PASSWORD` | 없음 |

MySQL에는 데이터베이스 아래에 스키마를 두는 층이 없으므로 `SCHEMA` 변수를 두지 않는다. 대상은 `--schema`에 데이터베이스 이름으로 지정한다.

접두어는 `--env-prefix`로 변경할 수 있으나, 다른 DBMS의 접두어를 지정하면 경고한다. 호스트와 포트가 그 DBMS의 값으로 채워지기 때문이다(18.5.2.1절).

## Databricks 접속 (`--dbms databricks`)

**표 0.16.** Databricks 접속 환경변수

| 변수 | 기본값 |
|---|---|
| `DATABRICKS_HOSTNAME` | 없음. `DATABRICKS_SERVER_HOSTNAME`·`DATABRICKS_HOST`도 참조한다 |
| `DATABRICKS_HTTP_PATH` | 없음. `/sql/1.0/warehouses/<식별자>` |
| `DATABRICKS_WAREHOUSE_ID` | 없음. HTTP 경로를 모를 때 이것으로 만든다 |
| `DATABRICKS_CLIENT_ID` | 없음. OAuth M2M의 서비스 프린시펄 |
| `DATABRICKS_CLIENT_SECRET` | 없음. OAuth M2M의 시크릿 |
| `DATABRICKS_TOKEN` | 없음. `DATABRICKS_ACCESS_TOKEN`도 참조한다 |
| `DATABRICKS_CATALOG` | 없음. 대상을 스키마로만 지정한 경우 앞에 붙인다 |

| 변수 | 기본값 |
|---|---|
| `DATABRICKS_SCHEMA` | 없음. 대상을 지정하지 않은 경우 사용한다 |

인증 방식은 설정한 변수가 결정한다. `CLIENT_ID`와 `CLIENT_SECRET`이 둘 다 있으면 OAuth M2M이고, 아니면 `TOKEN`을 사용한다(18.5.4.1절).

`HTTP_PATH`를 모르면 `WAREHOUSE_ID`로 대신한다. 식별자로 `/sql/1.0/warehouses/<식별자>`를 만든다(18.5.4.2절).

이 접두어도 `--env-prefix`로 변경한다. 스킴(`https://`)과 끝 슬래시는 지정하더라도 제거한 뒤 읽는다.

## LLM 제안 (`--coin`)

표 0.17. LLM 제안 환경변수

| 변수 | 기본값 |
|---|---|
| `KDSM_LLM_PROVIDER` | `anthropic`, `bedrock`을 지정할 수 있다 |
| `KDSM_LLM_MODEL` | 제공자별 기본값 (아래) |
| `KDSM_LLM_API_KEY` | 없음. `ANTHROPIC_API_KEY`도 참조한다 |

표 0.18. 제공자별 기본 모델

| 제공자 | 모델 기본값 |
|---|---|
| `anthropic` | `claude-sonnet-5` |
| `bedrock` | `global.anthropic.claude-sonnet-4-6` |

Bedrock은 모델 이름 체계가 다르다. 앞의 `global.`은 교차 리전 추론 프로파일 접두어이며, 리전을 고정해야 하는 경우에는 `us.`·`eu.`·`apac.`·`jp.`로 변경한다(12.4.3.1절).

## 도구 경로

표 0.19. 도구 경로 환경변수

| 변수 | 쓰임 |
|---|---|
| `KDSM_MMDC_BIN` | mermaid-cli 실행 파일의 경로. 지정하지 않으면 `PATH`에서 찾는다 |
| `KDSM_QUARTO_BIN` | Quarto 실행 파일의 경로. 지정하지 않으면 `PATH`에서 찾는다 |

둘 다 명령줄 인수가 앞선다. `--mmdc` → `KDSM_MMDC_BIN` → `PATH`(16.7절), `--quarto` → `KDSM_QUARTO_BIN` → `PATH`(18.2.6절) 순이다.

## AWS 자격 증명 (`--llm-provider bedrock`)

표 0.20. AWS 자격 증명 환경변수

| 변수 | 쓰임 |
|---|---|
| `AWS_ACCESS_KEY_ID` | 액세스 키 ID |
| `AWS_SECRET_ACCESS_KEY` | 시크릿 액세스 키 |
| `AWS_SESSION_TOKEN` | 임시 자격 증명을 사용할 때만 |
| `AWS_REGION` | 리전. `AWS_DEFAULT_REGION`도 참조한다 |
| `AWS_PROFILE` | 프로파일로 접속할 때 |
| `AWS_BEARER_TOKEN_BEDROCK` | 베어러 토큰으로 접속할 때 |

키 한 쌍이 없어도 베어러 토큰이나 프로파일이 있으면 접속한다. 아무것도 지정하지 않으면 SDK가 AWS의 기본 자격 증명 체인을 따른다.

# 용어 사전

이 매뉴얼이 특정한 뜻으로 사용하는 낱말을 모았다. 일반적인 뜻과 다르게 쓰는 것, 이 방법론이 정의한 것, 본문에서 정의 없이 반복 사용하는 것이 대상이다.

검증 규칙의 이름은 `library/docs/validation-rules.md`, 물리 테이블과 환경변수는 참고 자료에 있다.

## 데이터 표준

**표준단어**
: 이름의 최소 단위. 한글명에 영문명과 영문약어를 연결하여 등재한다. 표준용어는 표준단어의 조합으로만 만든다(2장).

**표준용어**
: 표준단어를 순서대로 조합하여 만든 이름. 엔터티명과 속성명으로 사용하며 컬럼명이 되는 단위이다(3장).

**표준도메인**
: 데이터 타입과 길이를 묶어 정의한 것. 컬럼마다 타입을 개별적으로 정하지 않고 도메인이 결정한다(4장).

**표준코드**
: 값이 정해진 목록에서만 선택되는 항목의 허용값 정의. 코드마스터가 코드의 성격을 정의하고 코드상세가 개별 값을 담는다(5장).

**수식어 · 주제어 · 분류어**
: 표준용어를 이루는 세 자리. 수식어는 없거나 여럿, 주제어는 하나, 분류어는 맨 뒤에 하나이다(3장).

**분류어(형식단어)**
: 표준용어의 맨 뒤에 오면서 값의 형식과 타입을 결정하는 단어. 「일자」「금액」「여부」가 이에 해당한다(2.3.1절).

**단어구분**
: 표준단어의 역할 구분. 일반어는 업무나 일상에서 의미를 전달하는 단어이고, 도메인어는 특정 업무 영역의 전문 용어이며, 분류어는 형식을 결정하는 단어이다. 판단이 명확하지 않으면 일반어로 분류한다(2.1.2절).

**용어구분**
: 표준용어를 엔터티명·속성명·양쪽 가운데 어느 위치에 사용할 수 있는지 지정한 구분. 이 값에 따라 분류어 종결과 표준도메인의 필수 여부가 달라진다(3.2.1절).

**영문약어**
: 표준단어에 부여하는 영문 약어. 물리명의 재료이므로 대문자와 숫자로만 이루어져야 하고 사전 전체에서 유일해야 한다. 영문명에서 기계적으로 도출되지 않는 관용 약어도 허용한다(2장).

**구성단어**
: 표준용어를 이루는 표준단어를 `+`로 연결하여 한 항목에 기록한 값. 값이 하나가 아니므로 외래키로 보장되지 않으며 검증 규칙이 분리하여 대조한다(1.4절).

**물리명**
: 구성 단어의 영문약어를 밑줄로 연결하여 계산되는 이름. 작성자가 임의로 기재하는 값이 아니라 사전에서 도출되는 값이며 그대로 컬럼명이 된다(3장).

**도메인그룹 · 도메인분류**
: 표준도메인 위에 놓인 두 층. 도메인그룹은 두 자리 코드이고, 도메인분류는 그룹 코드에 일련번호를 이어 붙인 네 자리 코드이다. 표준도메인은 코드 없이 이름이 곧 식별자이다(4.1.2절).

**이음동의어**
: 같은 뜻이지만 표준이 아닌 낱말. 사실상 금칙어로 취급한다(2.2.2절).

**금칙어**
: 표준단어로 등재해서는 안 되는 낱말. 쉼표로 구분하여 기록한다(2.2.2절).

**금지어**
: 사용해서는 안 되는 낱말 전체. 폐기된 표준단어·이음동의어·금칙어 세 경로로 수집된다(12.3절).

**폐기어와 대체어**
: 폐기한 표준단어와 그 대신 사용하도록 지정한 표준단어의 쌍. 두 낱말이 같은 뜻이라는 선언이므로 전수 검토가 필요하다(2.2.2.1절).

**통용어**
: 업무 현장에서 실제로 사용하는 말. 표준단어는 국어사전 중심의 순화어가 아니라 통용어로 정한다(2.2.2.2절).

**상태**
: 사전 항목이 현재 사용 가능한지를 나타내는 값. 후보와 확정은 검증 대상이고 폐기는 제외된다. 폐기는 현재 시점의 사용 금지 표시이며 이력 관리와는 구분된다(2.3.2절).

**후보 등록**
: 새로 정의한 단어·도메인·코드를 즉시 등재하지 않고 후보 상태로 먼저 올리는 절차. 검토가 진행되는 동안에도 후보 상태로 작업을 이어 갈 수 있다(1.5절).

**잠정 단어**
: 아직 사전에 없으나 표준용어 후보가 구성 요소로 사용한 단어. 먼저 등재해야 해당 용어를 사용할 수 있다(12.4.3절).

**고아 행**
: 상위 항목이 폐기되거나 제거된 뒤에도 남아 있는 하위 행. 표준코드를 폐기하면서 코드상세를 정리하지 않은 경우가 이에 해당한다(5.2.1절).

# 데이터 모델링

**논리 데이터 모델링**
업무가 다루는 대상을 엔터티와 속성으로 정의하는 과정. 어느 DBMS를 사용할지 정하지 않은 상태에서도 수행할 수 있다(7장).

**물리 데이터 모델링**
논리 모델을 실제 DBMS에 적용하는 과정. 엔터티가 테이블이 되고 속성이 컬럼이 되며, 도메인이 데이터 타입과 길이로 변환된다(8장).

**엔터티 구분**
엔터티를 독립적 존재 여부와 행위 기록 여부로 나눈 구분. 기본엔터티는 다른 엔터티 없이 존재하고, 중심엔터티는 기본엔터티에서 파생되며, 행위엔터티는 둘 이상의 엔터티 사이에서 발생한 행위를 기록한다(7.2.1절).

**속성 유형**
값의 출처를 기준으로 속성을 나눈 구분. 기본형은 업무에서 그대로 입력되는 값, 설계형은 관리를 위해 설계자가 추가한 값, 추출형은 다른 값에서 계산하여 도출한 값이다(7.3.1절).

**식별관계**
부모의 기본키가 자식의 기본키 일부가 되는 관계. 자식이 부모 없이 존재할 수 없다(16.6.2절).

**비식별관계**
부모의 기본키가 자식의 일반 속성이 되는 관계(16.6.2절).

**기본키 순번**
기본키를 구성할 때 1부터 부여하는 순번. 복합키의 순서까지 기록하며, 참조 항목과 조합되어 식별관계와 비식별관계를 구별한다(10장).

**주요 관계**
엔터티 정의서의 항목으로, 상대 엔터티와 카디널리티를 기록한다. 사람이 읽는 용도이며 검증과 ERD는 속성 정의서의 참조 항목을 읽는다. 다만 참조로 표현할 수 없는 관계는 이 항목에만 남는다(7.4절).

**상위 엔터티**
엔터티가 소속되는 상위 엔터티를 기록하는 항목. 소속을 나타내는 뜻으로 쓰이는 경우가 많아 식별관계의 판정 근거로 사용하지 않는다(7장).

**관련 엔터티**
테이블 정의서의 항목으로, 그 테이블이 어느 논리 엔터티에서 구현되었는지를 기록한다. 논리 모델과 물리 모델을 연결하는 유일한 근거이며 외래키로 설정하지 않는다(9.4.3절).

**통합과 분할**
물리 단계에서 두 엔터티를 하나의 테이블로 합치거나 하나의 엔터티를 여러 테이블로 나누는 판단. 통합한 테이블에는 1:1로 비교할 기준 이름이 없으므로 일부 규칙이 판정을 수행하지 않는다(8.2절).

**짝 대조(pairing)**
짝을 이루는 논리 산출물과 물리 산출물을 항목별로 비교하여 같은 사실이 어긋나지 않았는지 확인하는 작업. 이름·한글명·기본키·Null 허용·개인정보·기본값·참조를 비교한다(13.3.8절).

**문서 간 외래키**
참조 대상이 다른 산출물에 있는 외래키. 같은 스키마에 함께 생성되므로 유효하지만, 한 문서만 읽으면 존재하지 않는 테이블로 보인다(10.2.3절).

**주제영역**
: 데이터 모델을 나누어 작성하는 단위. 산출물 파일 하나가 주제영역 하나에 대응하며, 이름은 파일명에서 버전과 날짜를 제외한 부분으로 정해진다(14.3절).

**논리 타입**
: 사전이 정의한 DBMS 중립적인 데이터 타입. `STRING`·`NUMBER`·`DATETIME` 등이며 DDL 생성에서 대상 DBMS의 타입으로 변환된다(표 14.5).

## 산출물과 서식

**DM-DST**
: 사용자가 작성하는 산출물 3종. 데이터 표준 정의(001), 논리 데이터 모델(002), 물리 데이터 모델(003)이다(9.1절).

**DM-LIB**
: KDSM 메타테이블의 초기 데이터를 정의한 산출물 6종. 시스템 관리자가 작성하며 사용자의 작성 대상이 아니다(9.1절).

**메타파일**
: 메타테이블에 적재할 값을 시트별로 담은 엑셀 파일. 시트명이 곧 테이블 물리명이다. 산출물과 달리 표지와 작성 지침이 없다(11.4.2절).

**양식 시트**
: 표지 문구·헤더·항목 설명·데이터·작성 지침이 한 시트에 섞여 있는 산출물의 기재 시트. 각 요소의 위치가 고정되어 있지 않으므로 프로그램이 정해진 순서로 데이터 영역을 탐색한다(9.3.2절).

**필수 항목 표시**
: 헤더 셀에 붙이는 별표. 프로그램이 어느 행을 헤더로 볼 것인지 판정하는 근거이므로 지우면 헤더를 찾지 못한다(9.3.2절).

**항목 설명 행**
: 헤더 바로 아래의 한 행으로, 각 항목의 기재 방법을 적은 부분이다. 프로그램이 이 행을 건너뛰므로 삭제하면 첫 데이터 행이 설명으로 취급된다(9.3.2절).

**작성 지침**
: 데이터 아래에 오는 안내 문구 영역. 프로그램은 이 영역을 만나면 읽기를 중단한다(9.3.2절).

**데이터 영역**
: 양식 시트에서 프로그램이 실제로 읽어 들이는 범위. 헤더 다음 행부터 작성 지침이나 빈 행을 만나기 전까지이므로, 데이터 중간에 빈 행을 넣으면 그 자리에서 종료된다(9.3.2절).

**표지**
: 산출물의 첫 시트로 문서번호·버전·작성일·작성자를 기록한다. 개정할 때 파일명과 일치시켜야 한다(9장).

**문서번호**
: 산출물을 식별하는 번호. `DM-DST-001` 형식이며 앞의 계열 표시로 작성 주체와 용도가 갈린다(9.2절).

**버전 표기**
: 파일명에는 `V0.9.5`로, 표지와 양식 시트에는 `0.9.5`로 기록하는 규칙. 표기가 어긋나면 같은 버전이 문자열로는 다른 값이 된다(9.2절).

**NFC · NFD**
: 한글을 완성된 글자 하나로 저장하는 표기와 자모를 분리하여 저장하는 표기. 화면에는 같아 보이지만 문자열로는 다른 값이므로 검색·비교·중복 확인에서 오류가 발생한다(9.2절).

**엑셀 잠금 파일**
엑셀에서 파일을 열어 두었을 때 함께 생성되는 `~$` 로 시작하는 파일. 열려 있는 동안에는 저장하지 않은 변경 내용을 프로그램이 읽지 못하며, 오류 없이 이전 내용이 그대로 읽힌다(21장).

**관리용 컬럼**
메타파일에 있으나 표준화 대상이 아닌 컬럼. 모델은 무시하고 헤더 검증은 면제한다(11.4.3절).

**감사 컬럼**
`CRT_DT` · `CREATR` · `MDFCN_DT` · `MDFR`. 시트에는 없고 테이블에만 있으며 적재기가 값을 기록한다(15.3.3절).

## 추천과 검증

**분해**
한글 용어명을 사전의 표준단어 단위로 나누는 기능. 가능한 분해를 모두 생성한 뒤 점수로 순위를 정한다(12.2.2절).

**미등록 단어**
용어명을 분해하였을 때 표준단어로 인식되지 않고 남은 문자열. 신규 표준단어 등재 후보가 되며 금지어와는 구분된다(12.2.2절).

**추천기**
분해 결과에 도메인 추론과 기존 용어 충돌 판정을 더하여 표준용어 후보를 생성하는 기능. 조합이 업무상 적절한지는 판단하지 않는다(12장).

**등재 초안**
미등록 단어를 신규 표준단어로 등재하기 위한 영문명·영문약어·정의·단어구분의 초안. 확정이 아니라 검토 대상이다(12.4.3절).

**심각도**
검증 결과의 세 등급. ERROR는 반드시 고쳐야 할 위반, WARNING은 검토가 필요한 사항, INFO는 검사하지 못하여 판정하지 않았다는 기록이다(13.1.3절).

**규칙 묶음**
검사 대상별로 규칙을 모아 반환하는 단위. 참조할 사전을 지정하지 않으면 해당 규칙이 묶음에서 빠지므로 실행된 규칙 수가 달라진다(13장).

**규칙 제외**
검사에 필요한 참조 자료를 지정하지 않았을 때 해당 규칙을 리포트에 포함하지 않는 처리. 검사하지 않은 항목을 통과한 것으로 보고하지 않기 위한 것이므로, 통과한 규칙의 수만으로 안전을 판단해서는 안 된다(13.1.5절).

**기준선**
기존 사전만으로 검사한 결과. 신규 등재분과 결합하여 검사한 결과에서 이를 차감하면 이번 산출물이 유발한 위반만 남는다(13.3.1.1절).

**코드값과 코드명**
같은 사실을 메타테이블은 코드값으로, 정의서는 사람이 읽는 코드명으로 기재한다. 코덱이 읽고 쓸 때 두 표기를 서로 변환한다(13.3.2절).

**사전(표준 사전)**
검증과 진단에서 대상을 판정하는 비교 기준. 메타테이블과 메타파일 두 곳에서 읽을 수 있으며, 검사 대상인 산출물과는 구분된다(13.2절).

## 진단과 대조

**진단**
: 이미 구축된 데이터베이스가 표준을 어느 정도 준수하는지 측정하는 작업. 결과는 통과·실패가 아니라 비율이다(18장).

**대조(drift)**
: 정의서와 운영 데이터베이스의 카탈로그를 맞추어 차이를 찾는 작업. 어느 쪽이 옳은지는 판정하지 않는다(14.9절).

**카탈로그**
: DBMS가 보유한 테이블·컬럼·타입·제약 정보. 대조·진단·ERD 작성의 데이터베이스 쪽 원천이다(18장).

**준수율**
: 카탈로그의 컬럼을 사전과 대조하여 표준을 따르는 비율로 산출한 값. 진단 대상마다 집계 단위가 다르다(표 18.1).

**진단 대상**
: 준수율을 산출하는 세 항목. 표준용어·표준단어·표준도메인이며 각각 컬럼·낱말·용어가 등재된 컬럼을 집계 단위로 삼는다(표 18.1).

**부분 준수**
: 컬럼명이 표준용어로 등재되지는 않았으나 모든 구성 요소가 표준단어 영문약어인 상태. 사전에 등재하면 완전 준수가 된다(18.1.1.1절).

**낱말**
: 진단에서 컬럼 물리명을 밑줄과 대소문자 경계로 분리한 단위(18.4절).

**일관성 점수**
: 사전 없이 스키마 내부의 어긋남만으로 산출하는 점수. 다섯 구분 가운데 추정에 근거하는 「비슷한 약어」를 제외한 네 항목의 평균이다(18.3.3절).

**접미사별 타입**
: 컬럼명의 끝에 오는 약어가 같은 컬럼들이 서로 다른 데이터 타입으로 정의되어 있는지 확인하는 항목. 사전이 없을 때 접미사가 사실상 분류어의 역할을 수행한다는 점에 근거한다(18.3.2.1절).

**미등록 약어**
: 컬럼명을 분리한 낱말 가운데 표준단어의 영문약어가 아닌 것. 사용 빈도가 높은 순으로 집계하여 개선 순서의 근거로 삼는다(18.3절).

**같은 이름 다른 정의**
: 동일한 컬럼명이 테이블마다 서로 다른 타입으로 정의되어 있는 상태. 그 이름의 의미에 합의가 없었음을 나타낸다(18.3절).

**비슷한 약어**
: 같은 낱말을 다르게 축약한 것으로 보이는 약어의 묶음. 한쪽이 다른 쪽의 앞부분일 때만 묶으며, 추정에 근거하므로 일관성 총점에는 반영하지 않는다(18.3.2.2절).

**표기 스타일**
: 한 스키마 안에서 컬럼명의 표기법이 혼용되어 있는지 확인하는 항목. 사용 빈도가 가장 높은 표기를 따르는 컬럼의 비율로 점수를 산출한다(18.3절).

**한글명 대응**
: 컬럼 주석에서 뽑은 한글명과 컬럼명이 일대일로 대응하는지 확인하는 항목(18.3.2.3절).

**설명 채움율**
: 테이블과 컬럼에 주석이 작성되어 있는 비율. 이름 통계 가운데 값이 높을수록 좋다고 판단할 수 있는 유일한 지표이다(18.4절).

**종합 점수와 등급**
: 세 진단 대상의 준수율을 평균한 값과 그 값에 따라 부여하는 A부터 F까지의 구간. 시점 사이의 비교를 위한 단일 수치이며, 개선 계획은 세 값을 각각 확인하여 수립한다(표 18.2).

**진단 제언**
: 진단 결과에서 도출한 조치 사항을 해당 컬럼 수가 많은 순으로 제시한 목록(18.2.5절).

**측정 불가**
: 사전이 없어 준수율을 산출할 수 없는 상태. 0%로 기재하지 않는다. 표준을 준수하지 않은 것과 비교 기준이 없었던 것은 서로 다른 사실이기 때문이다(18.3절).

## 생성과 적재

**포워드 엔지니어링**
: 산출물에서 DDL과 ERD를 만드는 방향의 작업(14.1절).

**리버스 엔지니어링**
: 운영 데이터베이스나 메타테이블에서 문서와 ERD를 거슬러 복원하거나 대조하는 방향의 작업. 포워드 엔지니어링과 우열이 아니라 사용 시점이 다르다(14.1절).

**산출물 내보내기**
: 적재된 메타테이블이나 메타파일을 읽어 DM-DST 산출물을 생성하는 작업. 생성물은 버전 `0.0.0`, 작성자 공란으로 기록된다. 사람이 아직 검토하지 않았다는 표시이다(17장).

**영역**
: 메타테이블에서 ERD를 작성할 때 도면을 몇 장으로 나눌지 정하는 편집상의 구분. 메타테이블에는 이 정보가 없으므로 설정 파일에서 읽는다(16.3절).

**위상 정렬**
: 참조 의존 관계에 따라 테이블 생성 순서를 정하는 방식. 참조 대상이 먼저 생성되고 DROP은 그 역순이다(10.3절).

**적재 순서**
: 외래키 제약을 위반하지 않도록 정한 메타테이블의 적재 차례. 참조 대상 테이블이 먼저 채워져야 한다(15.3.2절).

**UPSERT**
: 없으면 넣고 있으면 갱신하는 적재 방식. PostgreSQL의 `INSERT ... ON CONFLICT DO UPDATE` 이다(15.3.1절).

**식별자 접기**
: DBMS가 인용하지 않은 식별자를 소문자로 변환하여 저장하는 동작. 인용 부호를 붙이면 글자 그대로 사용하라는 지시가 되므로 접힌 이름과 다른 이름이 된다(21.1.4.1절).

**방언(dialect)**
: DBMS마다 다른 타입 표기와 식별자 규칙을 담은 모듈. DDL을 작성할 때의 방언과 카탈로그를 읽을 때의 방언은 적용 방향이 반대이므로 서로 대체할 수 없다(11장).

**잔존 테이블**
: 테이블 이름을 변경한 뒤 DDL을 다시 실행하였을 때 이전 이름의 테이블이 데이터를 포함한 채 남아 있는 것. 오류를 일으키지 않으므로 별도로 정리해야 한다(15.6절).

## 환경과 작업 흐름

**선택 의존성(extra)**
: 필요한 기능이 있을 때에만 설치하는 추가 패키지 묶음. 설치하지 않으면 해당 기능만 사용할 수 없다고 안내하고 중단되며 나머지 기능은 그대로 수행된다(21.1.4.2절).

**정확 동기화**
: 요청한 항목만 남기고 나머지를 제거하는 동기화 방식. 필요한 기능을 두 번에 나누어 설치하면 뒤의 실행이 앞서 설치한 것을 제거한다(21.1.4.2절).

**환경변수 접두어**
: 접속 정보 환경변수의 이름 앞부분. 접두어를 교체하면 여섯 변수를 한 벌로 바꾸는 셈이 되므로, 대상 데이터베이스마다 접두어를 달리 정해 두고 오간다(20.9.4절).

**신규 구축**
: 사전 등재에서 데이터베이스 적재에 이르는 최초 구축 작업. 이후의 대조와 진단은 필요가 발생하였을 때 수행하는 작업이므로 이 범위에 들지 않는다(22장).

**운영 전환**
: 운영이 시작되면서 기준이 문서에서 데이터베이스로 바뀌는 국면. 이후에는 문서를 수작업으로 맞추지 않고 산출물 내보내기로 다시 생성한다(22장).

**빠른 진단표**
: 오류 메시지나 증상을 왼쪽에 두고 확인할 곳을 오른쪽에 대응시킨 표. 이미 오류가 발생한 뒤에는 원인별 설명보다 이 표에서 증상을 찾는 편이 빠르다(21.2절).