## QDQM을 이용한 마스터 코드 관리

|  |  |  |
| --- | --- | --- |
| **단계** | **절차 설명** | **활용 도구** |
| **1. 레퍼런스 코드와 비교 검증** | 레퍼런스 코드(ICAO/IATA/ISO 등)와 내부 마스터코드와의 정합성 검증 | QDQM  코드 매핑표,  레퍼런스 코드셋 |
| **2. 중복성/중의성 검사** | 테이블내에서 동일 코드 중복 여부 검증  유사 의미 코드 존재 여부 (다른 코드로 동일 의미의 코드 표현) | QDQM  SQL or Python  ETL Validation |
| **3. 매핑 정합성 검증** | 업무 간에 코드 매핑 테이블 관리  예: FARE\_BASIS\_CODE ↔ RBD(예약등급)  AGENCY\_CODE ↔ SALES\_CHANNEL\_CODE | 비즈니스 로직 및 마이그레이션 플랜에서 작성  매핑 테이블 |
| **4. 시스템 간 코드 통합 여부 점검** | 시스템(ERP, PSS, RM 등) 간 코드값 불일치 여부 분석 | QDQM  Interface Validation  매핑 테이블 |
| **5. 사용현황 및 미사용 코드 점검** | 실제 사용 이력 없는 코드 식별 → 폐기 대상 여부 결정  Timestamp 기준으로 참조 테이블별 사용 이력 추적 (테이블 기준 및 코드값 기준) | QDQM Code Usage  Usage Audit, Log 분석 |
| **6. 변경이력 추적** | 코드 등록·변경·폐기에 대한 승인 및 변경이력 검토 | QDQM 적용검토(?)  Data Governance,  Master Data Workflow |
| **7. 데이터 기반 정합성 검증** | 매핑 정의서에 기준 관계 정합성 검증  사용여부 Flag 기준 무효 코드 사용 탐지  Rule 기반 정합성 및 규칙 검증 | QDQM  BI, SQL 분석 |
| **8. 예외코드 관리 프로세스 검토** | NULL, UNKNOWN, TEMP 등 비표준 코드(예외 로직) 처리 기준에 의한 사용 여부 검증  (향후 코드 Lifecycle 관리 체계 정립시 활용)  5~7 번 과정 작업에서 현 예외 코드리스트가 나타나고 클린징 작업 이후 수행 | 예외 로직 정의서  SQL or Python |
| **9. 정기 리포트/모니터링** | 정기적인 검사에 의한 이상값/오류 발생 자동 감지 및 알림 시스템  향후 운영시 마스터 코드 데이터를 관리하는 도구 적용 | QDQM  Data Quality Dashboard |

데이터 기반의 ERD 자동 생성

상위 수준 Data Architecture 정리

## 마스터코드 외부 표준(레퍼런스 코드)과의 정합성 확인

### 외부 표준 코드(레퍼런스 코드)란?

|  |  |  |
| --- | --- | --- |
| **표준 종류** | **예시** | **설명** |
| **IATA** | 공항코드(인천: ICN), 항공사코드(KE) | 항공사/공항/노선 등 국제표준 |
| **ICAO** | 공항: RKSI, 항공사: KAL | ATC 및 운항용 4자리 표준 코드 |
| **ISO** | 국가코드(KR), 통화코드(KRW), 언어코드(KO) | 국제 기구에서 제정한 코드셋 |
| **UN/LOCODE** | 인천: KRINC | 무역/물류 국제위치코드 |
| **기타** | ICH, BSP, SWIFT, GS1 등 | 정산, 금융, 물류 관련 표준 |

\*\*IATA Clearing House (ICH)\*\*는 항공 운송 산업을 위한 신속하고 안전하며 비용 효율적인 청구 및 정산 서비스를 제공합니다.

**Billing and Settlement Plan (BSP)** 데이터는 **IATA**(International Air Transport Association)에서 운영하는 **BSPlink** 시스템을 통해 접근할 수 있습니다. BSPlink는 여행사와 항공사 간의 판매, 보고, 송금을 간소화하는 웹 기반 플랫폼입니다.

**SWIFT**(Society for Worldwide Interbank Financial Telecommunication)는 금융 기관 간의 메시지 표준을 관리합니다.

### 정합성 확인 절차 (프로세스 기반)

|  |  |  |
| --- | --- | --- |
| **단계** | **설명** | **방법** |
| 1️⃣ 외부 표준(레퍼런스) 코드 수집 | IATA/ISO/ICAO 등 최신 버전 데이터 확보 | 공식 웹사이트 또는 유료 DB 구독 |
| 2️⃣ 마스터코드 매핑표 구축 | 내부 마스터 코드 ↔ 외부표준(레퍼런스) 코드 매핑 | **QDQM 자동 매핑 기능 적용**  **매핑표 수정 및 확정**  예: CITY\_CODE ↔ IATA\_CITY\_CODE |
| 3️⃣ 불일치 여부 탐지 | 매핑 누락  잘못된 연결  포맷 불일치 확인  1:n 및 n:1 관계 처리 | **QDQM 매핑 점검 기능 적용**  **매핑표 수정 후 재 점검**  SQL / ETL / BI Rule 활용 |
| 4️⃣ 예외값/가공 코드 관리 | 내부 특화코드(예: VIRTUAL, UNKNOWN) 처리 | 매핑표에 명시적 정의 후  **QDQM 매핑 점검 기능 적용** |
| 5️⃣ 연동 시스템 점검 | GDS, 정산, BI와의 인터페이스 검증 | 실제 연계 테스트 적용  **QDQM 연동 점검 기능 적용** |
| 6️⃣ 정기 검토 및 업데이트 | 연 1~2회 외부 표준 갱신 반영 | IATA 등 기준 변경 주기 따라 지속적인 점검 |

**QDQM에 자동 매핑 기능 적용 예정**

**SQL 기반으로 매핑 검증 방법**

**🔧 1. 정확 매칭 (Exact Match) – 기본 구조**

**✔ 방식:**

* 내부코드 = 외부코드일 경우 단순 일치 매핑

**✅ 예시 SQL**

SELECT i.INTERNAL\_CODE, e.STANDARD\_CODE

FROM INTERNAL\_MASTER i

JOIN EXTERNAL\_STANDARD e ON i.CODE = e.CODE;

**💡 활용:**

* ISO COUNTRY\_CODE, CURRENCY\_CODE, IATA 3-letter AIRPORT\_CODE 등 **일치성이 높은 표준**

**🧠 2. 유사도 기반 매핑 (Fuzzy Matching)**

**✔ 방식:**

* 코드명/설명/지역/국가 등 텍스트 기반 유사도를 계산하여 매핑

**✅ 도구/알고리즘:**

* Levenshtein Distance (edit distance)
* Jaro-Winkler, TF-IDF, Cosine Similarity (벡터 기반)
* Python 라이브러리: fuzzywuzzy, RapidFuzz, spaCy, pandas + difflib

**✅ 예시 코드 (Python + FuzzyWuzzy)**

from fuzzywuzzy import process

match = process.extractOne("Seoul Incheon", external\_airport\_list)

print(match)

**💡 활용:**

* AGENCY\_NAME, CITY\_NAME, PRODUCT\_DESC 등 텍스트 기반의 데이터에 대한 일치 여부 확인

**🧩 3. 매핑 테이블 + 규칙 기반 룰셋 활용**

**✔ 방식:**

* 내부 → 외부 매핑 테이블을 유지하고, 룰셋으로 관리
* 규칙 예시: 접두어 규칙, 포맷 기준, 코드 분리/병합 등

**✅ 예시 SQL (패턴 매칭)**

SELECT i.CODE, e.CODE

FROM INTERNAL i

JOIN EXTERNAL e

ON i.CODE LIKE CONCAT('%', e.CODE)

OR i.NAME = e.DESCRIPTION;

**💡 활용:**

* FARE\_BASIS, SSR, BRAND\_CODE 등 구조적 유사성이 있는 코드에 적용

**🔄 4. ETL 자동화 및 검증 로직 삽입**

**✔ 방식:**

* ETL 툴(Talend, Informatica, Pentaho 등)에 매핑 로직을 Flow로 구현
* 오류/누락/예외 코드 로그 수집 자동화

**✅ 구성 예시:**

[INTERNAL\_CODE] → [JOIN 매핑표] → [IF NULL → 로그 적재] → [출력 테이블]

**🤖 5. AI/ML 기반 지능형 매핑 (대규모 레거시 대상)**

**✔ 방식:**

* ML 모델이 수백~수천 쌍의 매핑 데이터를 학습하고, 새로운 매핑을 예측
* Features: 코드 유사도, 설명 유사도, 키워드, 국가명, 지역 등

**✅ 적용 도구:**

* AutoML, HuggingFace + spaCy (텍스트 임베딩)
* Vertex AI Matching Engine, AWS SageMaker

**💡 추천 상황:**

* 대규모 매핑 (> 10,000건 이상), 수작업 불가능, 반복성이 높을 때

**✅ 매핑 자동화 시스템 구성 예시**

text

복사편집

[ 내부 코드 테이블 ]

↓

[ 규칙 기반 정합 매핑 ] ← [표준 코드 테이블]

↓

[ 유사도 필터 (90%↑)] ← [AI 기반 예측 보조]

↓

[ 수동 검토 영역 (예외)]

↓

[ 통합 매핑 테이블 생성 ]

## 마스터코드 중의성(Ambiguity) 검사 방법

|  |  |  |
| --- | --- | --- |
| **검사 항목** | **설명** | **예시** |
| **1. 코드값의 다중 의미 검사** | 하나의 코드가 시스템/업무/문서마다 다른 의미로 쓰이는지 확인 | "Y"가 FARE\_BASIS, GENDER, ANSWER 등 다르게 해석 |
| **2. 명칭(Name) 중복 검사** | 서로 다른 코드값이 동일하거나 유사한 이름을 가지는지 점검 | "ECONOMY\_BASIC"과 "ECO\_BASIC" 등 혼동 가능 |
| **3. 매핑 테이블 정합성 검사** | 코드 간 매핑이 1:1 또는 N:1/1:N으로 되어 있는지 검사 | RBD ↔ FARE\_BASIS,  PRODUCT ↔ BRAND 매핑 불일치 |
| **4. 시스템 간 정의 불일치 탐지** | 동일 코드가 PSS와 ERP 등에서 다르게 정의돼 있는지 비교 | "OTA"가 한쪽은 B2C, 다른 쪽은 GDS로 분류됨 |
| **5. 다국어/라벨 변환 오류 검사** | 동일 코드가 언어마다 다르게 라벨링되어 사용자 혼란 유발 여부 확인 | "Standard" vs "기본" vs "스탠다드" |
| **6. 코드 파생 규칙 중복 여부 확인** | 코드 생성 시 규칙적 파생이 중복되지 않는지 검사 | "KE01"과 "KE001" 등 |
| **7. 예외값 포함 여부 점검** | UNKNOWN, TEMP, NULL 등이 코드로 쓰이는 경우, 정의/범위 검토 | "XXX"가 공항코드? 미정? 예외값? |
| **8. 시각화/출력 시 중복 해석 가능성** | 사용자 화면, 리포트, 대시보드에서 혼동 유발 여부 | "B"가 Brand인지 Booking인지 명확하지 않음 |

### 마스터코드 중의성(Ambiguity) 검사 자동화 방법

#### 🔍 1. 중의성의 정의 (2가지 패턴)

|  |  |  |
| --- | --- | --- |
| **유형** | **설명** | **예시** |
| 🔁 **동일 코드 ↔ 여러 의미** | 코드값은 같지만 의미가 다름 | EC001 → 경제석, 교육과정 |
| 🔀 **동일 의미 ↔ 여러 코드** | 의미는 같지만 코드가 다름 | BASIC\_ECONOMY, ECO\_BASIC |

#### 🧭 2. 자동화 검사 방법 SQL

✅ A. 코드값 + 코드명 다중매핑 탐지 (1:N)

SELECT CODE, COUNT(DISTINCT NAME) AS NAME\_VARIANTS

FROM MASTER\_CODE

GROUP BY CODE

HAVING COUNT(DISTINCT NAME) > 1;

* ✔ 동일 코드값에 다양한 명칭이 존재하면 중의성 의심

Python 코드

import pandas as pd

# 예시 데이터프레임 생성

data = {

'CODE': ['EC001', 'EC001', 'EC001', 'BC001', 'BC001', 'FR001'],

'NAME': ['ECONOMY', 'ECONOMY BASIC', 'ECO BASIC', 'BUSINESS', 'BIZ CLASS', 'FIRST']

}

df = pd.DataFrame(data)

# 중의성 의심 코드 추출: CODE별 NAME의 고유값 개수 > 1

ambiguous\_codes = df.groupby('CODE')['NAME'].nunique().reset\_index()

ambiguous\_codes = ambiguous\_codes[ambiguous\_codes['NAME'] > 1]

ambiguous\_codes.rename(columns={'NAME': 'NAME\_VARIANTS'}, inplace=True)

# 결과 확인

print("✅ 중의성 의심 CODE 목록 (1:N 다중 매핑)")

print(ambiguous\_codes)

# 상세 보기: 어떤 NAME들이 연결되어 있는지

detailed\_view = df[df['CODE'].isin(ambiguous\_codes['CODE'])].sort\_values(by='CODE')

print("\n🔍 상세 매핑 내역:")

print(detailed\_view)

✅ 중의성 의심 CODE 목록 (1:N 다중 매핑) 결과

CODE NAME\_VARIANTS

0 BC001 2

1 EC001 3

🔍 상세 매핑 내역:

CODE NAME

0 EC001 ECONOMY

1 EC001 ECONOMY BASIC

2 EC001 ECO BASIC

3 BC001 BUSINESS

4 BC001 BIZ CLASS

✅ B. 코드명 + 의미 기준 중복코드 탐지 (N:1)

SELECT NAME, COUNT(DISTINCT CODE) AS CODE\_VARIANTS

FROM MASTER\_CODE

GROUP BY NAME

HAVING COUNT(DISTINCT CODE) > 1;

* ✔ 한 의미에 여러 코드가 존재하는 경우 → 병합 검토

✅ C. 텍스트 유사도 기반 의미 유사 중복 감지 (AI 기반)

from fuzzywuzzy import process

process.extract("ECONOMY BASIC", list\_of\_code\_names)

* ✔ 자연어 유사도 점수로 ECO\_BASIC, ECONOMY\_BASIC, EC\_BASIC 유사군 탐지

✅ D. 의미 사전 (Ontology) 매핑 비교

* 내부 코드명 ↔ 외부표준(예: IATA SSR, ISO, UN/LOCODE) 의미맵 비교

SELECT a.CODE, b.STANDARD\_MEANING

FROM MASTER\_CODE a

LEFT JOIN EXTERNAL\_MEANING\_MAP b ON a.CODE = b.CODE

WHERE a.MEANING != b.STANDARD\_MEANING;

## 코드 관계 검증 (매핑 정합성) 방법

### 대표적인 코드 관계 예시

|  |  |  |
| --- | --- | --- |
| **코드 A** | **코드 B** | **설명** |
| FARE\_BASIS\_CODE ↔ RBD | 운임과 예약등급 간 매핑 |  |
| AGENCY\_CODE ↔ SALES\_CHANNEL\_CODE | 여행사 → 판매 채널 유형 |  |
| PRODUCT\_CODE ↔ BRAND\_CODE | 상품 코드 → 브랜드 |  |
| AIRPORT\_CODE ↔ CITY\_CODE | 공항 → 도시 |  |
| AIRCRAFT\_TYPE\_CODE ↔ ROUTE\_CODE | 기종별 노선 제한 관계 |  |
| COST\_CENTER ↔ ORG\_UNIT | 회계 센터 ↔ 조직 |  |

### 코드 관계 검증 절차

|  |  |  |
| --- | --- | --- |
| **단계** | **설명** | **방법** |
| 1️⃣ 매핑 테이블 식별 | 연계 관계가 필요한 코드 간 연결 정보 정의 | 수작업 or  시스템 인터페이스 기준 |
| 2️⃣ 기준 테이블 정의 | 참조 기준이 되는 테이블 또는 표준 테이블 지정 | 예: MASTER\_RBD vs MASTER\_FARE |
| 3️⃣ 매핑 누락/다중 매핑 탐지 | 1:1이 아닌 1:N, N:1 오류 식별 | **QDQM 매핑 점검 기능 적용**  SQL, ETL Rule 또는 BI |
| 4️⃣ 유효성 체크 | 양방향 정합성, 사용 여부, 예외 처리 정의 | **QDQM 매핑 점검 기능 적용**  SQL JOIN 또는 Full Outer Join |
| 5️⃣ 결과 시각화 및 리뷰 | 매핑 충돌 현황 보고서 작성 | Excel or BI 대시보드 |

## 시스템 간 코드 통합 여부 점검 방법

### 🔍 1. 점검 목적

|  |  |
| --- | --- |
| 목적 | 설명 |
| 🔄 코드 이중관리 방지 | 동일 항목이 여러 시스템에서 중복 정의되어 불일치 발생 방지 |
| 🧭 데이터 연계 품질 향상 | 예약 → 발권 → 정산 등 시스템 간 매핑 정확도 향상 |
| 🛡 통제 체계 유지 | 변경 시 전체 시스템 반영 여부 추적 가능 |
| 📊 분석 정합성 확보 | BI/정산 리포트 기반 정확성 보장 |

### ✅ 2. 점검 방식 요약

|  |  |  |
| --- | --- | --- |
| **방식** | **설명** | **적용 예시** |
| 📋 **직접 비교 방식** | 동일 코드 항목을 시스템별 테이블에서 직접 비교 | PSS vs ERP vs BI |
| 🔗 **중앙 통합 매핑표 기준 점검** | 통합관리 테이블과 매핑여부 확인 | MASTER\_CODE\_MAP 기준 |
| 🧠 **자동 탐지(비일치, 누락)** | JOIN, NOT IN, EXCEPT, FULL OUTER JOIN | SQL or Python |

### 🧾 3. SQL 기반 점검 예시

**✔ [A] 코드 일치 여부 확인**

SELECT A.CODE AS PSS\_CODE, B.CODE AS ERP\_CODE

FROM PSS\_CODE\_TABLE A

FULL OUTER JOIN ERP\_CODE\_TABLE B ON A.CODE = B.CODE

WHERE A.CODE IS NULL OR B.CODE IS NULL;

**✔ [B] 매핑 누락 탐지 (기준 테이블 존재)**

SELECT A.CODE

FROM PSS\_CODE\_TABLE A

LEFT JOIN MASTER\_CODE\_MAP M ON A.CODE = M.PSS\_CODE

WHERE M.PSS\_CODE IS NULL;

### 🧠 4. 파이썬(Pandas) 예제 프로그램:

시스템 간 통합 여부 자동 점검

import pandas as pd

# 예시: PSS와 ERP에서 사용 중인 코드 리스트

pss\_codes = pd.DataFrame({'CODE': ['ECONOMY', 'BUSINESS', 'FIRST', 'PREMIUM']})

erp\_codes = pd.DataFrame({'CODE': ['ECONOMY', 'BUSINESS', 'FIRST']})

# 공통코드

common = pd.merge(pss\_codes, erp\_codes, on='CODE', how='inner')

# 누락코드 (PSS에는 있지만 ERP에는 없는 것)

missing\_in\_erp = pss\_codes[~pss\_codes['CODE'].isin(erp\_codes['CODE'])]

# 누락코드 (ERP에는 있지만 PSS에는 없는 것)

missing\_in\_pss = erp\_codes[~erp\_codes['CODE'].isin(pss\_codes['CODE'])]

print("✅ 공통 코드:")

print(common)

print("\n❌ ERP에 없는 코드:")

print(missing\_in\_erp)

print("\n❌ PSS에 없는 코드:")

print(missing\_in\_pss)

### 📦 5. 결과 리포트 예시

|  |  |  |  |
| --- | --- | --- | --- |
| **코드** | **PSS 존재** | **ERP 존재** | **상태** |
| ECONOMY | ✅ | ✅ | 일치 |
| FIRST | ✅ | ✅ | 일치 |
| PREMIUM | ✅ | ❌ | 누락(ERP) |
| FLEX | ❌ | ✅ | 누락(PSS) |

## 사용현황 및 미사용 코드 점검 방법

### 주요 점검 목적

|  |  |
| --- | --- |
| **목적** | **설명** |
| 🔍 **미사용 코드 식별** | 시스템에 존재하지만 거래/이벤트에 사용되지 않은 코드 탐지 |
| 🗃 **폐기 대상 분리** | 퇴역/만료/계약해지된 코드 구분 |
| ⛔ **오류 방지** | 잘못 사용된 코드(예: TEST, OLD)로 인한 리포팅 오류 제거 |
| 📊 **실사용 통계 기반 분석** | 어떤 코드가 어떤 부서, 업무에서 얼마나 자주 쓰이는지 파악 |

### 2. 점검 방식 (2-Table 비교)

|  |  |
| --- | --- |
| **비교 대상** | **설명** |
| 🎯 **기준 테이블** (MASTER\_CODE) | 시스템에 등록된 코드 목록 |
| 📊 **사용 테이블** (TRANSACTION\_LOG, BOOKING, SALES, 등) | 실제 사용 내역 포함된 테이블 |

### 점검 항목 및 방법

|  |  |  |
| --- | --- | --- |
| **항목** | **점검 방법** | **예시** |
| **1. 최근 사용이력 확인** | 거래 로그, 예약 내역, 정산내역에서 참조 여부 조회 | SQL로 6~12개월 내 미사용 탐지 |
| **2. 시스템별 참조 여부** | 각 시스템(PSS, ERP, 정산 등)에서 사용 여부 조회 | API, DWH 쿼리 |
| **3. Active Flag 확인** | 코드 정의에 Active 여부 컬럼이 있는지 점검 | ACTIVE\_FLAG = 'N' 필터 |
| **4. 등록 이후 미사용 코드 식별** | 신규 등록 코드 중 미사용 상태 계속 유지되는 코드 탐지 | 등록일 기준 + 사용 이력 없음 |
| **5. 유사코드 존재 여부** | 동일 목적의 다른 코드와 중복 가능성 점검 | 이름/속성 유사 분석 |
| **6. 사용처 문서화 여부** | 코드별 시스템/업무 사용처 기록되어 있는가 | 코드 정의서/데이터카탈로그 |

### 3. SQL 기반 점검 예시

**✔ [A] 미사용 코드 식별 (최근 12개월 기준)**

SELECT m.CODE

FROM MASTER\_CODE m

LEFT JOIN USAGE\_LOG u ON m.CODE = u.CODE

AND u.USE\_DATE >= CURRENT\_DATE - INTERVAL '12 MONTH'

WHERE u.CODE IS NULL;

**✔ [B] 사용 횟수 통계**

SELECT m.CODE, COUNT(u.CODE) AS USAGE\_COUNT

FROM MASTER\_CODE m

LEFT JOIN USAGE\_LOG u ON m.CODE = u.CODE

GROUP BY m.CODE

ORDER BY USAGE\_COUNT ASC;

### 🧠 4. Python 예제 프로그램 (Pandas 기반)

import pandas as pd

from datetime import datetime, timedelta

# 예시: 등록된 마스터코드

master\_df = pd.DataFrame({'CODE': ['ECO', 'BUS', 'FIR', 'PRE']})

# 예시: 사용 이력

usage\_df = pd.DataFrame({

'CODE': ['ECO', 'ECO', 'BUS'],

'USE\_DATE': ['2023-12-01', '2024-03-01', '2024-02-20']

})

usage\_df['USE\_DATE'] = pd.to\_datetime(usage\_df['USE\_DATE'])

# 최근 12개월 기준 필터

cutoff = datetime.today() - timedelta(days=365)

recent\_usage = usage\_df[usage\_df['USE\_DATE'] >= cutoff]

# 사용 여부 판단

used\_codes = recent\_usage['CODE'].unique()

master\_df['USED\_IN\_12M'] = master\_df['CODE'].isin(used\_codes)

# 미사용 코드 추출

unused = master\_df[master\_df['USED\_IN\_12M'] == False]

print("✅ 사용 현황:")

print(master\_df)

print("\n❌ 미사용 코드:")

print(unused)

### 📊 5. 보고서 양식 예시

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CODE** | **최근 사용일** | **사용 건수** | **12개월 이내 사용 여부** | **폐기 후보** |
| ECO | 2024-03-01 | 25 | ✅ 사용됨 | ❌ |
| BUS | 2024-02-20 | 5 | ✅ 사용됨 | ❌ |
| FIR | - | 0 | ❌ 미사용 | ✅ |
| PRE | - | 0 | ❌ 미사용 | ✅ |

## 6. 변경이력 추적 및 승인체계 확인

### 🔍 1. 목적 요약

|  |  |
| --- | --- |
| **항목** | **설명** |
| 🧾 변경 감사 추적 | 변경 전후 값을 저장하여 추후 감사/추적 가능 |
| 🔒 승인/반려 이력 관리 | 코드 변경이 누가 승인했는지 이력 관리 |
| ⛔ 무단 변경 방지 | 직접 수정이 아닌 요청-승인 기반 구조 |
| 📜 정책 연계 | 데이터 거버넌스 정책 기반 승인 제어 가능 |

### 🏗 2. 구성 방식 개요

**▶️ 일반 구조**

[코드 변경 요청서]

↓

[요청 테이블 저장] ← [요청자]

↓

[승인 대기] ← [승인자 검토]

↓

[승인 시 적용] → [마스터코드 테이블 반영]

↓

[변경 이력 저장]

### ✅ 3. 테이블 구성 예시

**🔹 마스터코드 테이블 (예: MASTER\_CODE)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | CODE | NAME | ACTIVE | LAST\_UPDATED |

**🔹 변경 요청 테이블 (예: CODE\_CHANGE\_REQUEST)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| REQUEST\_ID | CODE | FIELD | OLD\_VALUE | NEW\_VALUE | REQUESTOR | REQUEST\_DATE | STATUS | APPROVER | APPROVE\_DATE |

### 🧾 4. SQL 예시: 변경 이력 자동 저장 트리거

CREATE TABLE CODE\_AUDIT\_LOG (

LOG\_ID SERIAL PRIMARY KEY,

CODE VARCHAR(20),

FIELD\_CHANGED VARCHAR(50),

OLD\_VALUE TEXT,

NEW\_VALUE TEXT,

MODIFIED\_BY VARCHAR(50),

MODIFIED\_AT TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE OR REPLACE FUNCTION log\_code\_changes()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.NAME <> OLD.NAME THEN

INSERT INTO CODE\_AUDIT\_LOG (CODE, FIELD\_CHANGED, OLD\_VALUE, NEW\_VALUE, MODIFIED\_BY)

VALUES (OLD.CODE, 'NAME', OLD.NAME, NEW.NAME, current\_user);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_code\_update

AFTER UPDATE ON MASTER\_CODE

FOR EACH ROW

EXECUTE FUNCTION log\_code\_changes();

### 🧠 5. Python 예제: 승인 흐름 반영

import pandas as pd

from datetime import datetime

# 변경 요청 등록

change\_requests = pd.DataFrame({

'REQUEST\_ID': [1],

'CODE': ['ECO001'],

'FIELD': ['NAME'],

'OLD\_VALUE': ['ECONOMY'],

'NEW\_VALUE': ['ECONOMY BASIC'],

'REQUESTOR': ['jane.kim'],

'REQUEST\_DATE': [datetime.now()],

'STATUS': ['PENDING'],

'APPROVER': [None],

'APPROVE\_DATE': [None]

})

# 승인 처리 함수

def approve\_request(df, request\_id, approver):

idx = df[df['REQUEST\_ID'] == request\_id].index[0]

df.at[idx, 'STATUS'] = 'APPROVED'

df.at[idx, 'APPROVER'] = approver

df.at[idx, 'APPROVE\_DATE'] = datetime.now()

print(f"✅ 요청 {request\_id} 승인 완료.")

# 승인 실행 예시

approve\_request(change\_requests, 1, 'manager.kang')

print(change\_requests)

## 데이터 기반 정합성 검증 방법

### 🔍 1. 목적 및 필요성

|  |  |
| --- | --- |
| **항목** | **설명** |
| 🎯 코드 사용의 실제 유효성 검증 | 등록된 코드가 운영 데이터에서 정확히 사용되고 있는지 확인 |
| 🔎 데이터와 로직 간 일치 여부 점검 | 마스터코드 ↔ 거래데이터 간 매핑, 룰 기반 연결 검증 |
| ⚙ 자동화 규칙 기반 탐지 | 단순히 등록된 코드가 존재하는지 뿐 아니라 ‘정상적인 방식으로 사용되는지 판단 |

### ✅ 2. 주요 검증 유형

|  |  |  |
| --- | --- | --- |
| **검증 항목** | **설명** | **예시** |
| 🔗 **관계 정합성** | 예약코드 ↔ 여정 ↔ 운임코드 연결 | FARE\_BASIS가 여정과 연결돼야 함 |
| ⛔ **무효 코드 사용 탐지** | 비활성(폐기) 코드가 사용되고 있는 경우 | ACTIVE = 'N' 코드 사용 |
| 📐 **조건 기반 정합성** | 특정 조건 하에 다른 코드의 조합이 맞는지 확인 | RBD Y는 항상 FARE Y로 연결 |
| 🧾 **규칙 위반 탐지** | 업무 규칙 위배 데이터 존재 여부 | SSR 'WCHR'가 항공기 타입 B737에서 사용 불가 등 |

### 🧾 3. SQL 기반 예제

**✔ [A] 비활성 코드 사용 탐지**

SELECT t.CODE, COUNT(\*)

FROM TRANSACTION\_LOG t

JOIN MASTER\_CODE m ON t.CODE = m.CODE

WHERE m.ACTIVE\_FLAG = 'N'

GROUP BY t.CODE;

**✔ [B] 매핑 누락 (관계 불일치)**

SELECT f.FARE\_BASIS

FROM BOOKING b

LEFT JOIN FARE\_MASTER f ON b.FARE\_BASIS = f.FARE\_BASIS

WHERE f.FARE\_BASIS IS NULL;

**✔ [C] 조건 기반 정합성 (CASE 기반 Rule 검증)**

SELECT \*

FROM BOOKING

WHERE RBD = 'Y' AND FARE\_BASIS NOT LIKE 'Y%';

**✔ [D] 유효 기간 정합성 (날짜 기반)**

SELECT \*

FROM PROMOTION\_CODE p

WHERE CURRENT\_DATE < p.EFFECTIVE\_FROM

OR CURRENT\_DATE > p.EXPIRE\_DATE;

### 🧠 4. Python + Pandas 예제 프로그램

import pandas as pd

# 예시 데이터

booking = pd.DataFrame({

'CODE': ['YOW', 'BIZ01', 'FLEX99', 'YOW'],

'ACTIVE\_FLAG': ['Y', 'Y', 'N', 'N']

})

# 비활성 코드 사용 탐지

invalid\_use = booking[booking['ACTIVE\_FLAG'] == 'N']

print("❌ 비활성 코드 사용 내역:")

print(invalid\_use)

# 조건 기반 정합성 (예: 'Y'는 항상 Y로 시작해야 함)

booking['VALID\_COMBO'] = booking['CODE'].apply(lambda x: x.startswith('Y'))

print("\n📐 조건 기반 검증 결과:")

print(booking)

## 예외코드 관리 프로세스 검토 방법

### 🔍 1. 예외코드 정의

|  |  |  |
| --- | --- | --- |
| 유형 | 예시 | 설명 |
| TEMP001 | TEMP001, TEST999 | 테스트 목적 또는 일시 등록된 코드 |
| DUMMY | DUMMY123 | 값 없음 처리를 위해 임시 부여 |
| UNKNOWN | UNKNOWN, NULL\_CODE | 알 수 없는 상태 처리용 코드 |
| 폐기/과거 코드 | OLD\_XXX | 과거 사용되다 폐기된 코드가 남아있는 경우 |

### 🧾 2. 예외코드 관리 검토 항목

|  |  |
| --- | --- |
| 항목 | 검토 내용 |
| ✅ 예외코드 목록 정의 여부 | 예외코드가 미리 정의돼 있고 문서화되어 있는가 |
| ✅ 사용범위 제한 여부 | 특정 시스템 또는 업무영역에만 제한적으로 사용되는가 |
| ✅ 자동차단/경고 처리 | 사용 시 경고, 로깅, 차단 처리가 존재하는가 |
| ✅ 정기 점검 대상 포함 여부 | 클렌징 점검 시 포함되어 검토되는가 |
| ✅ 대체 코드 존재 여부 | 예외코드가 제거될 경우의 대체 방안이 정의되어 있는가 |

### ✅ 3. SQL 기반 예외코드 사용 점검

SELECT CODE, COUNT(\*) AS USAGE\_COUNT

FROM TRANSACTION\_LOG

WHERE CODE IN ('TEMP001', 'UNKNOWN', 'DUMMY001')

GROUP BY CODE;

### ✅ 4. Python 예외코드 탐지 예제

import pandas as pd

# 예시 데이터

usage\_data = pd.DataFrame({

'CODE': ['ECO', 'BUS', 'TEMP001', 'DUMMY123', 'ECO', 'UNKNOWN'],

'USE\_DATE': pd.to\_datetime(['2024-01-01', '2024-01-05', '2024-01-10', '2024-01-12', '2024-02-01', '2024-02-10'])

})

# 예외코드 리스트 정의

exception\_codes = ['TEMP001', 'DUMMY123', 'UNKNOWN']

# 사용 내역 중 예외코드 필터링

exception\_usage = usage\_data[usage\_data['CODE'].isin(exception\_codes)]

print("❗ 예외코드 사용 내역:")

print(exception\_usage)

### 📊 5. 보고서 양식 예시

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 예외코드 | 사용 건수 | 최근 사용일 | 사용 시스템 | 조치 필요 여부 |
| TEMP001 | 13 | 2024-02-10 | 예약시스템 | ✅ 제거 또는 차단 |
| DUMMY123 | 5 | 2024-01-12 | 정산모듈 | ✅ 대체코드 검토 |
| UNKNOWN | 7 | 2024-02-10 | BI 보고서 | ✅ 명확한 코드 매핑 필요 |

### 🧩 6. 예외코드 관리 자동화 절차

예외코드 마스터 목록 등록 (EXCEPTION\_CODE\_LIST)

운영데이터에서 정기적(일간/주간)으로 사용 여부 확인

사용시 로그 생성 → 담당자 알림

지정된 유예기간 초과 시 폐기 자동화 또는 보고서 알림