**Snowflake Data Platform**

**Hands on Guide**

# **1. 프로젝트 개요**

## **1.1. 프로젝트 추진 배경**

• 향후 투입될 프로젝트에서 Snowflake를 주요 플랫폼으로 활용하게 될 것을 대비

• 사전에 Snowflake의 구조와 데이터 처리 기능을 체득하고자

Snowflake Data Platform을 직접 설계·구축하는 프로젝트를 수행

## **1.2. 프로젝트 목적**

• Snowflake를 기반으로 **Medallion Architecture**를 적용하여

데이터 수집–정제–집계까지의 전 과정과 운영·보안 체계를 실제 환경 수준으로

구현해보는 것을 목표

• 정형(**CSV**)과 반정형(**JSON**) 데이터를 외부 스토리지(**AWS S3**, **Azure Blob**)와 연동하여 Stage를 통해 Snowflake로 적재하고, **Bronze–Silver–Gold** 레이어를 따라 체계적으로

관리·가공

• **자동화 파이프라인** 구축

• 데이터 거버넌스 기능을 직접 적용하여 실무 수준의 **보안·접근제어** 환경을 구성

## **1.3. 실습 환경 구성**

**AWS** 또는 **Azure** 선택 사용 가능(선택사항)

• 모두 Snowflake와 동일한 방식으로 연동되며, 원본 데이터를 저장하는

Bronze 레이어(**Data Lake**) 역할을 수행하므로 실습 목적을 달성하는 데

기능적 차이는 없음

### 1.3.1. [**Kaggle** 계정](https://www.kaggle.com/datasets)

실습에 사용할 샘플 데이터셋을 다운로드하기 위해 필요

### 1.3.2. [**AWS** 계정 (Amazon Web Services)](https://ap-northeast-2.console.aws.amazon.com/console/home?region=ap-northeast-2)

S3 Bucket을 활용해 외부 스테이지를 구성하기 위한 계정

### 1.3.3. [**Azure** 계정 (Microsoft Azure – 학생 계정 추천)](https://portal.azure.com/#home)

Azure Blob Storage를 활용해 외부 스테이지를 구성하기 위한 계정

### 1.3.4. [**Snowflake** 계정](https://app.snowflake.com/) (Enterprise Edition)

본 실습의 핵심 플랫폼으로 실습 전체에 필요한 기능을 제공

## **1.4. 샘플 데이터**

**Kaggle**에서 제공되는 공개 데이터를 기반으로 샘플 데이터를 구성

### **1.4.1. CSV (정형 데이터)**

#### 1.4.1.1. [manufacturing\_defect\_dataset](https://www.kaggle.com/datasets/rabieelkharoua/predicting-manufacturing-defects-dataset?utm_source=chatgpt.com)

**제조 결함 데이터 세트**

#### 1.4.1.2. [hybrid\_manufacturing\_categorical](https://www.kaggle.com/datasets/ziya07/manufacturing-production-data)

**생산 제조 데이터 세트**

### **1.4.2. JSON(반정형 데이터)**

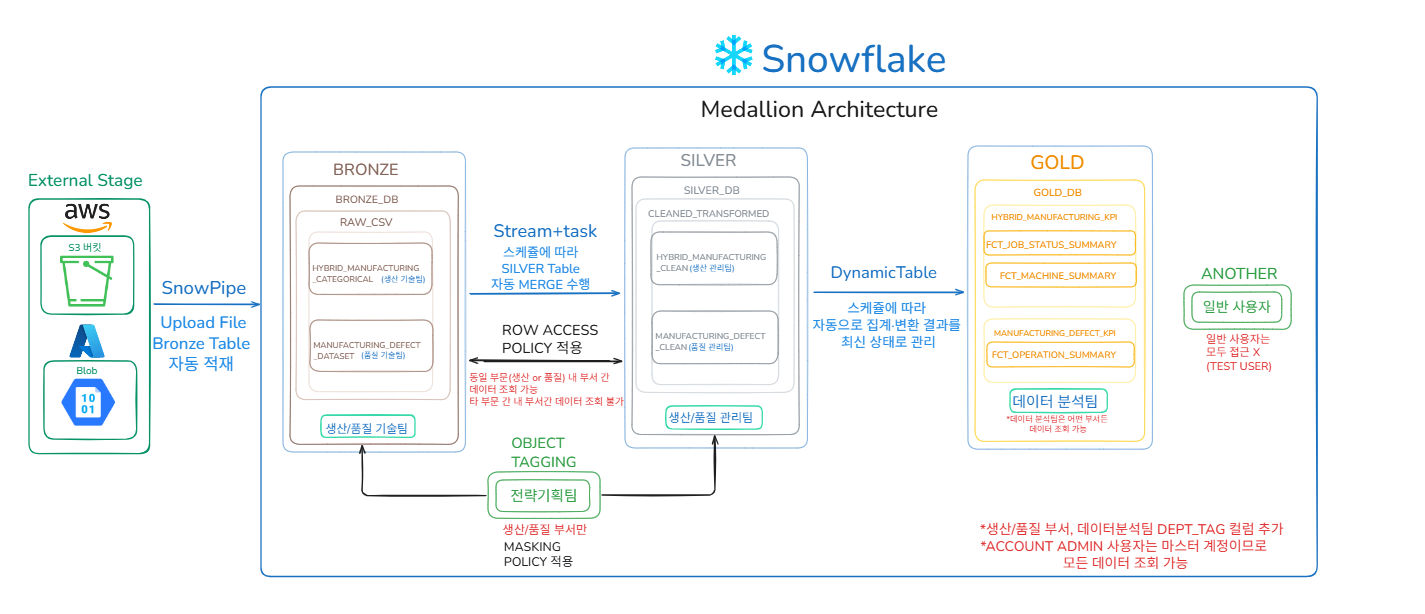
#### 1.4.2.1. [intents](https://www.kaggle.com/datasets/rizwanrizwannazir/jsonchatbot)

**챗봇 형식의 데이터 세트**

• 본 프로젝트에서 JSON 형태의 샘플 데이터는 Bronze → Silver 단계까지만 수행

• 이후 단계에서는 CSV 기반 데이터셋과 연계하여, 사용자가 자연어로 질의를 입력하면 해당 CSV 데이터를 기반으로 결과를 반환하는 챗봇 기반 데이터 질의 기능을 구현할 계획

# **2. Medallion Architecture Data Flow**



Medallion 아키텍처1) 방식으로 진행

1) 데이터 레이크(Lakehouse)에서 데이터를 점진적으로 정제하고 가공하기 위해 사용하는 계층적 데이터 구조 설계 방식

• Bronze = 원시(raw) 데이터

• Silver = 정제(clean) 데이터

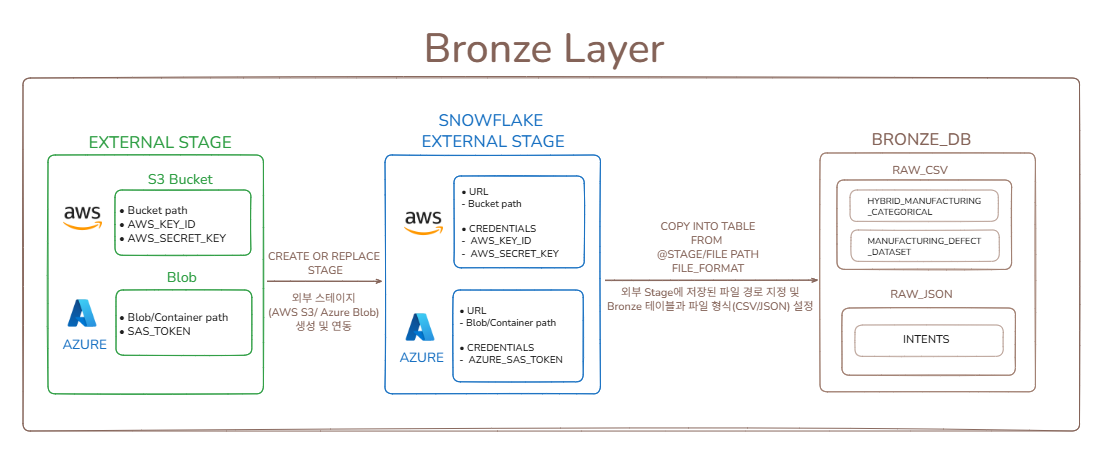
• Gold = 분석(curated) 데이터

• 실제 제조·품질 데이터 환경을 가정하여 **Snowflake 기반 Data Lake 설계**

• **데이터 파이프라인의 자동화** 운영

• 팀/부서별 역할에 따른 **보안·거버넌스** 구성

# **3. 1주차: Bronze Layer**



• Bronze Layer는 외부 스토리지(AWS S3, Azure Blob)에 저장된 원본 데이터를 Snowflake로 안전하게 가져와 **가공 없이 그대로 보관**하는 단계

• AWS S3 버킷 또는 Azure Blob 경로를 기반으로 외부 Stage를 생성하고,

Snowflake External Stage와 연동하여 데이터 접근 권한과 경로를 설정

• External Stage에 있는 파일을 지정된 Bronze 테이블에 적재

## **3.1. 실습 목표**

• **Snowflake 기본 구조 이해**: Database, Schema, Table의 관계 및 역할을 명확히 파악

• **정형/반정형 데이터 적재 실습**: CSV 및 JSON 파일을 Snowflake에 적재하고 구조 확인

• **Stage 활용법 익히기**: AWS S3, Azure Blob Storage를 이용한 파일 업로드 실습

## **3.2. 샘플 데이터 준비**

### **3.2.1. CSV (정형 데이터)**

#### 3.2.1.1. [manufacturing\_defect\_dataset](https://www.kaggle.com/datasets/rabieelkharoua/predicting-manufacturing-defects-dataset?utm_source=chatgpt.com)

**제조 결함 데이터 세트**

**데이터 구조 요약:**

ProductionVolume : 생산량

ProductionCost : 생산 비용

SupplierQuality : 공급 업체 품질

DeliveryDelay : 납품 지연

DefectRate : 불량률

QualityScore : 품질 점수

MaintenanceHours : 유지보수 시간

DowntimePercentage : 다운타임 비율

InventoryTurnover : 재고 회전율

StockoutRate: 품절률

WorkerProductivity : 노동자 생산성

SafetyIncidents : 안전 사고

EnergyConsumption : 에너지 소비량

EnergyEfficiency : 에너지 효율성

AdditiveProcessTime : 첨가 공정 시간

AdditiveMaterialCost : 첨가 재료 비용

DefectStatus : 불량 상태

#### 3.2.1.2. [hybrid\_manufacturing\_categorical](https://www.kaggle.com/datasets/ziya07/manufacturing-production-data)

**생산 제조 데이터 세트**

**데이터 구조 요약:**

Job\_ID : 작업 식별자

Machine\_ID : 해당 작업이 수행된 기계의 식별자

Operation\_Type : 공정 유형

Material\_Used : 사용된 재료의 양

Processing\_Time : 작업 처리 시간

Energy\_Consumption : 에너지 소비량

Machine\_Availability : 기계 가용성

Scheduled\_Start : 예정 시작 시간

Scheduled\_End : 예정 종료 시간

Actual\_Start : 실제 시작 시간

Actual\_End : 실제 종료 시간

Job\_Status : 작업 상태 (예: 완료, 지연 등)

Optimization\_Category : 최적화 범주 (예: 효율성 수준 등)

### **3.2.2. JSON(반정형 데이터)**

#### 3.2.2.1. [intents](https://www.kaggle.com/datasets/rizwanrizwannazir/jsonchatbot)

**챗봇 형식의 데이터 세트**

**코드 구성:**

{

"intents": [

{

"tag": "인사",

"patterns": [

"안녕",

"안녕하세요",

"누가 있나요?",

"안녕하세요",

"있나요"

],

"responses": [

"안녕하세요",

"안녕",

"안녕하세요"

]

},

{

"tag": "안녕",

"patterns": [

"잘가",

"나중에 보자",

"안녕"

],

"responses": [

"나중에 봐요",

"좋은 하루 보내세요",

"잘가요! 또 오세요"

]

},

{

"tag": "감사",

"patterns": [

"고마워",

"고마워요",

"도움이 돼서 고마워",

"도움이 돼서 고마워요"

],

"responses": [

"도와드릴 수 있어 기뻐요!",

"언제든지!",

"제 양성입니다",

"어떤 일이든 도와드릴게요!"

]

},

{

"tag": "대화 내용",

"patterns": [

"당신은 누구세요?",

"무엇을 하는 거죠?",

"당신은 누구신가요?"

],

"responses": [

"나는 조안나, 당신의 챗봇 어시스턴트입니다.",

"저는 조안나, 인공지능 챗봇입니다."

]

},

{

"tag": "이름",

"patterns": [

"당신의 이름은 무엇인가요?",

"무엇으로 부르면 되나요?",

"당신의 이름은 뭐야?"

],

"responses": [

"당신은 제 이름을 조안나(로) 불러도 돼요.",

"저는 조안나에요!",

"그냥 저를 조안나라고 불러 주세요."

]

},

{

"tag": "도움",

"patterns": [

"도와주세요.",

"도와주세요.",

"도와줄 수 있나요?",

"무엇을 도와드릴까요?",

"지원이 필요해요.",

"도움이 필요해요.",

"도와주세요."

],

"responses": [

"어떻게 도와드릴까요?",

"도와드릴 문제를 말씀해주세요.",

"네, 무엇을 지원해 드릴까요?"

]

},

{

"tag": "계정생성",

"patterns": [

"새로운 계정을 만들고 싶어요",

"어떻게 새로운 계정을 만들죠?",

"계정을 생성하고 싶어요",

"제게 계정을 만들어주실 수 있나요?",

"새로운 계정을 어떻게 만들어야 하나요?"

],

"responses": [

"당사 웹 사이트에서 손쉽게 새로운 계정을 만들 수 있습니다",

"당사 웹 사이트로 이동하여 지침에 따라 새로운 계정을 만들어보세요"

]

},

{

"tag": "불만",

"patterns": [

"불만이 있어",

"불만을 제기하고 싶어",

"서비스에 관한 불만이 있어"

],

"responses": [

"불만 내용을 제공해 주시면 도와드릴 수 있습니다",

"불만 사항을 언급해 주시고, 불편을 끼쳐드려 죄송합니다. 저희가 연락드리겠습니다."

]

}

]

}

## **3.3. Snowflake 데이터 구조 구성 (Database · Schema · Table)**

### 3.3.1. **데이터베이스 및 스키마 생성**

**Layer**별로 공간을 분리해 데이터 혼선과 권한 충돌을 방지

• 데이터베이스:

**BRONZE\_DB**

외부에서 들어오는 **원본(raw) 데이터**를 그대로 보관하는 전용 데이터베이스

• 스키마:

**RAW\_CSV (정형 CSV 원본용)**

**RAW\_JSON (반정형 JSON 원본용)**

=> 데이터베이스 안에서 주제·프로젝트별 구분 필요

## **3.4. 타입별 원본 테이블 생성(CSV / JSON)**

\*주의사항: 테이블 생성 시 스키마 RAW\_CSV와 RAW\_JSON 구분하여 테이블 생성

### **3.4.1. RAW\_CSV**

#### 3.4.1.1. **HYBRID\_MANUFACTURING\_CATEGORICAL**

CREATE OR REPLACE TABLE BRONZE\_DB.RAW\_CSV.HYBRID\_MANUFACTURING\_CATEGORICAL (

Job\_ID VARCHAR(10),

Machine\_ID VARCHAR(10),

Operation\_Type VARCHAR(100),

Material\_Used NUMBER(10,2),

Processing\_Time NUMBER(10,0),

Energy\_Consumption NUMBER(10,2),

Machine\_Availability NUMBER(10,0),

Scheduled\_Start TIMESTAMP\_NTZ,

Scheduled\_End TIMESTAMP\_NTZ,

Actual\_Start TIMESTAMP\_NTZ,

Actual\_End TIMESTAMP\_NTZ,

Job\_Status VARCHAR(100),

Optimization\_Category VARCHAR(100)

);

#### 3.4.1.2. **MANUFACTURING\_DEFECT\_DATASET**

CREATE OR REPLACE TABLE BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET (

PRODUCTION\_ID STRING,

PRODUCTION\_VOLUME NUMBER(10,0),

PRODUCTION\_COST FLOAT,

SUPPLIER\_QUALITY FLOAT,

DELIVERY\_DELAY NUMBER(3,0),

DEFECT\_RATE FLOAT,

QUALITY\_SCORE FLOAT,

MAINTENANCE\_HOURS NUMBER(3,0),

DOWNTIME\_PERCENTAGE FLOAT,

INVENTORY\_TURNOVER FLOAT,

STOCKOUT\_RATE FLOAT,

WORKER\_PRODUCTIVITY FLOAT,

SAFETY\_INCIDENTS NUMBER(3,0),

ENERGY\_CONSUMPTION FLOAT,

ENERGY\_EFFICIENCY FLOAT,

ADDITIVE\_PROCESS\_TIME FLOAT,

ADDITIVE\_MATERIAL\_COST FLOAT,

DEFECT\_STATUS BOOLEAN

);

### **3.4.2. RAW\_JSON**

#### 3.4.2.1. **INTENTS**

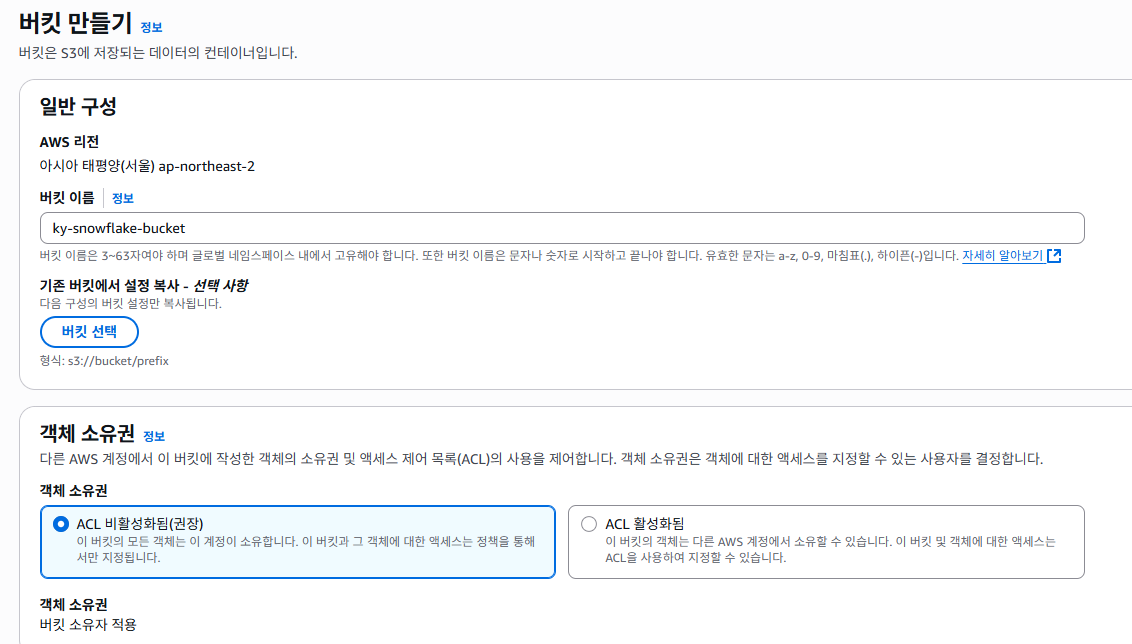
CREATE OR REPLACE TABLE BRONZE\_DB.RAW\_JSON.INTENTS (DATA VARIANT);

## 3.5. **외부 스테이지 연동 및 파일 적재**

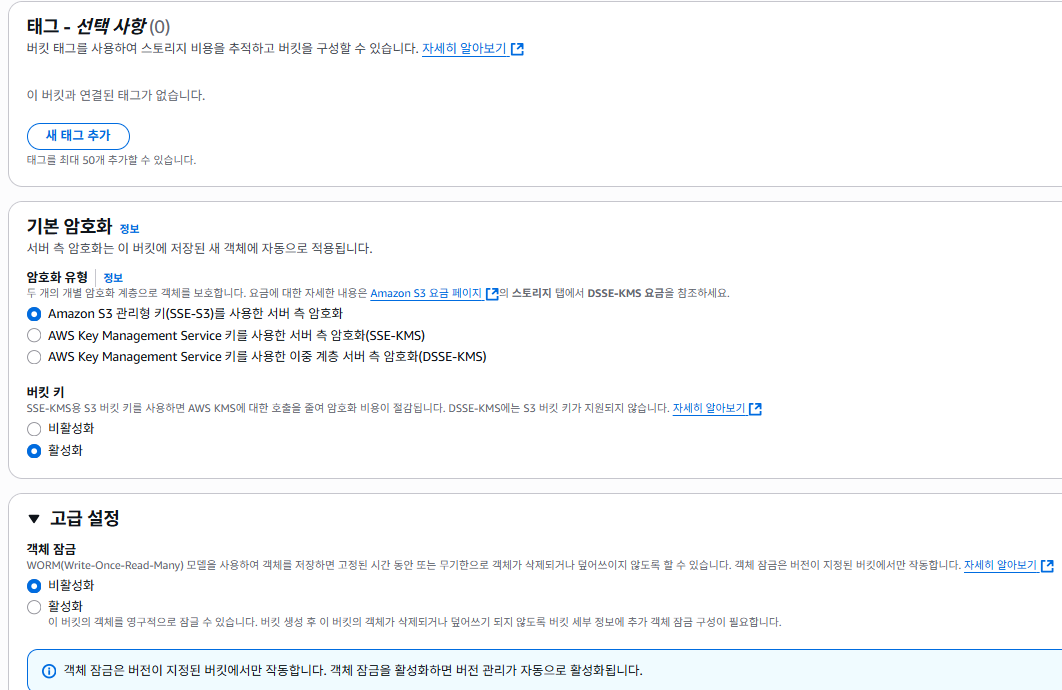
### 3.5.1. AWS S3 연동 및 Snowflake 파일 업로드

[AWS 콘솔 로그인](https://ap-northeast-2.console.aws.amazon.com/console/home?region=ap-northeast-2)

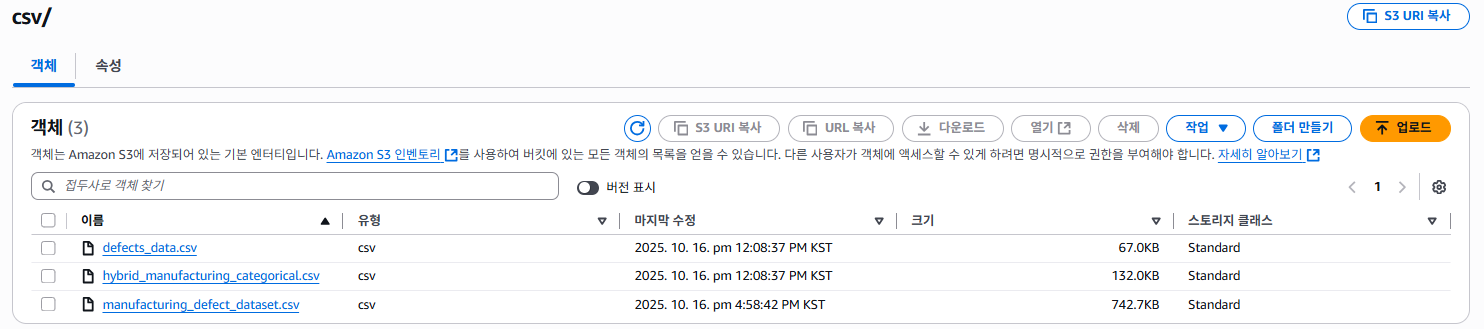
#### 3.5.1.1. **S3 버킷 생성**

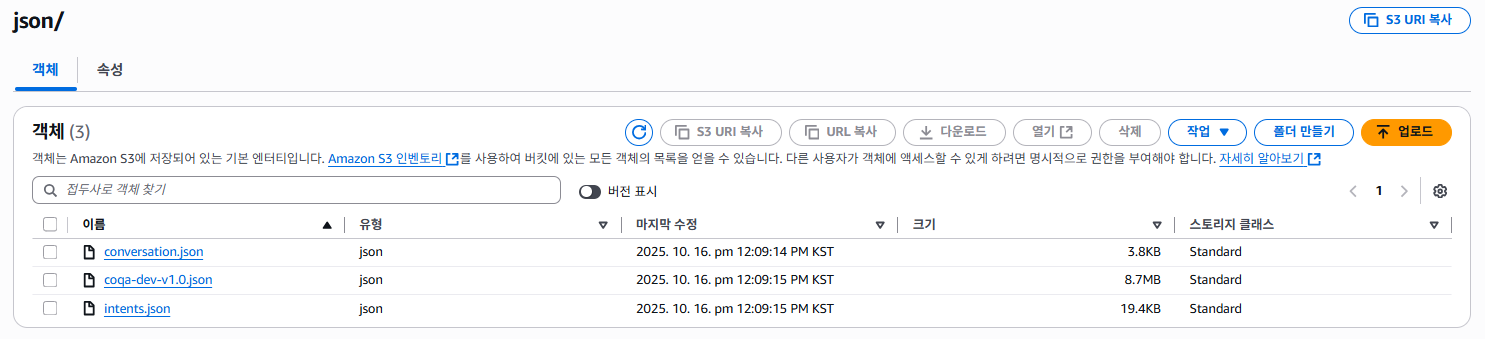






#### 3.5.1.2. **S3 버킷 폴더 생성 및 파일 업로드**

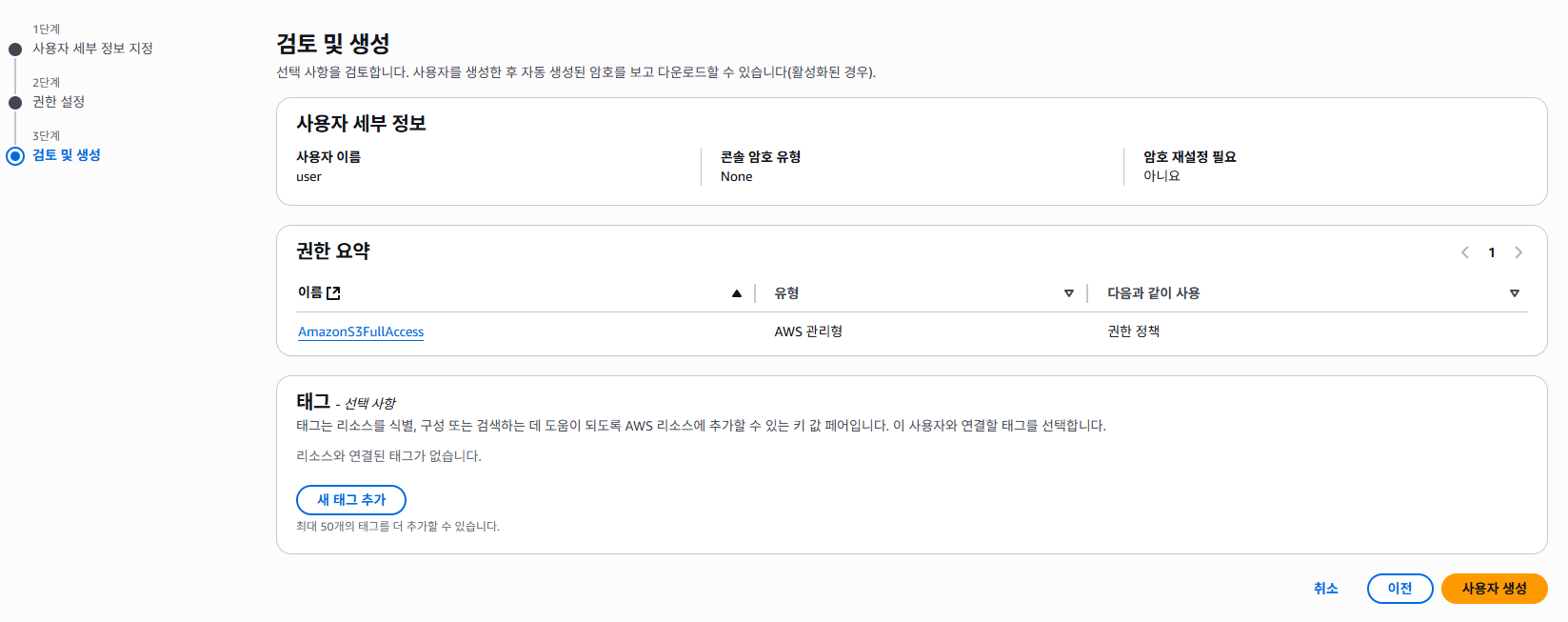
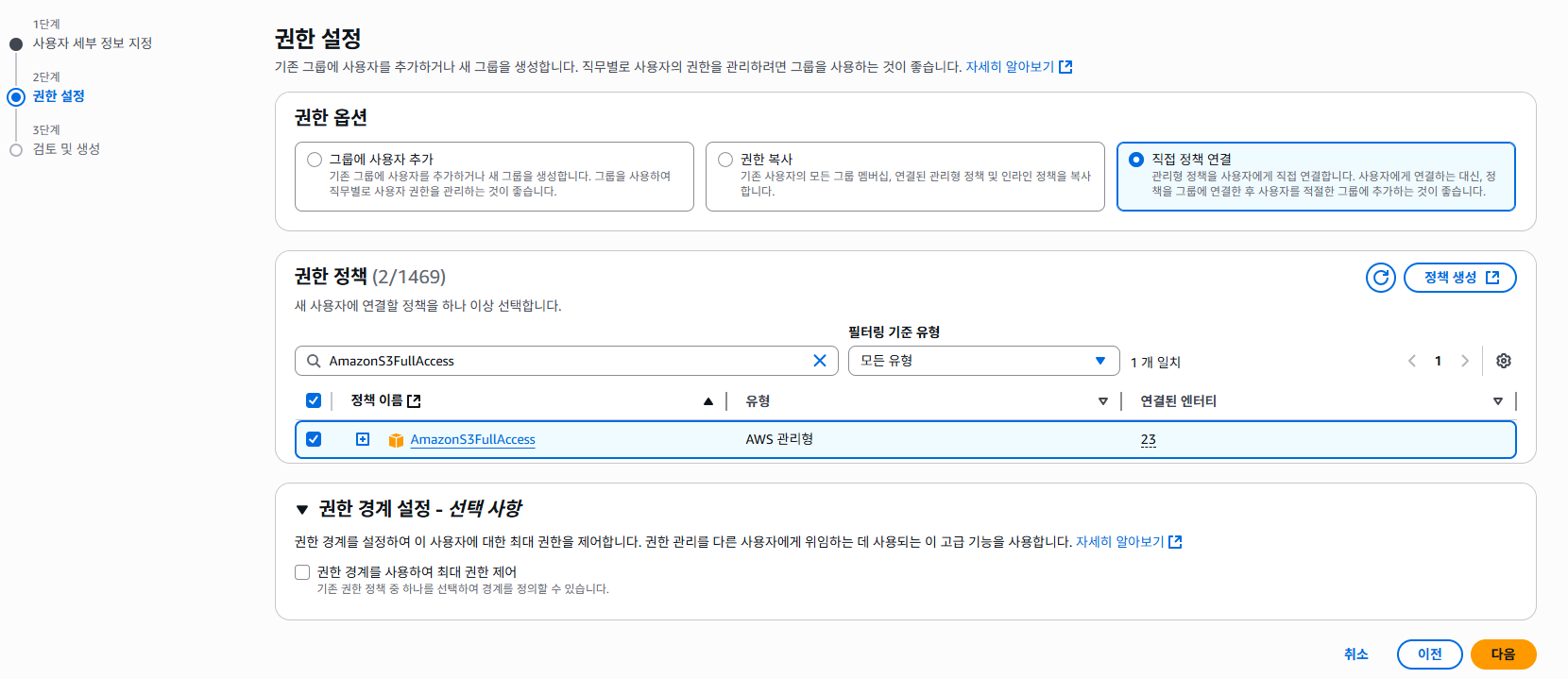
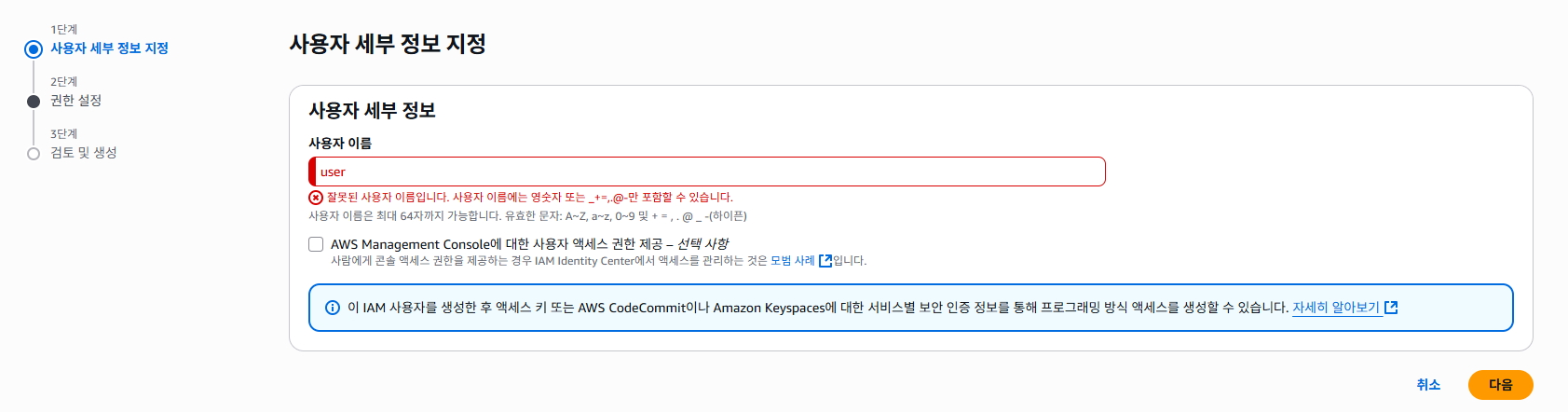




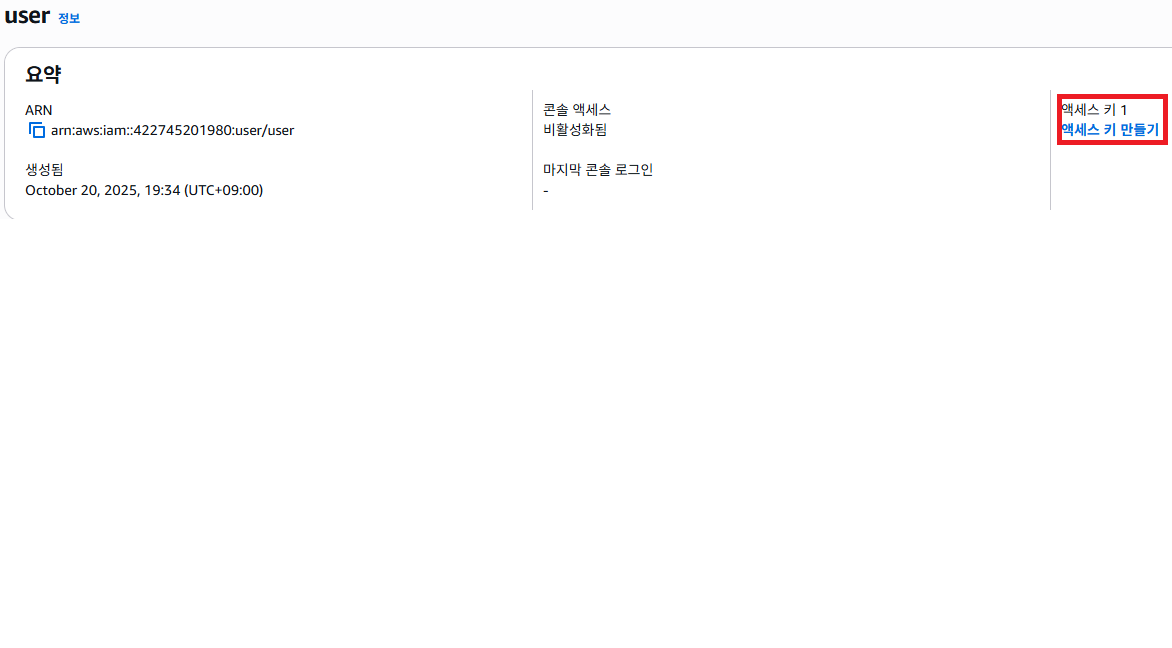
#### 3.5.1.3. **IAM 사용자 선택 또는 생성**

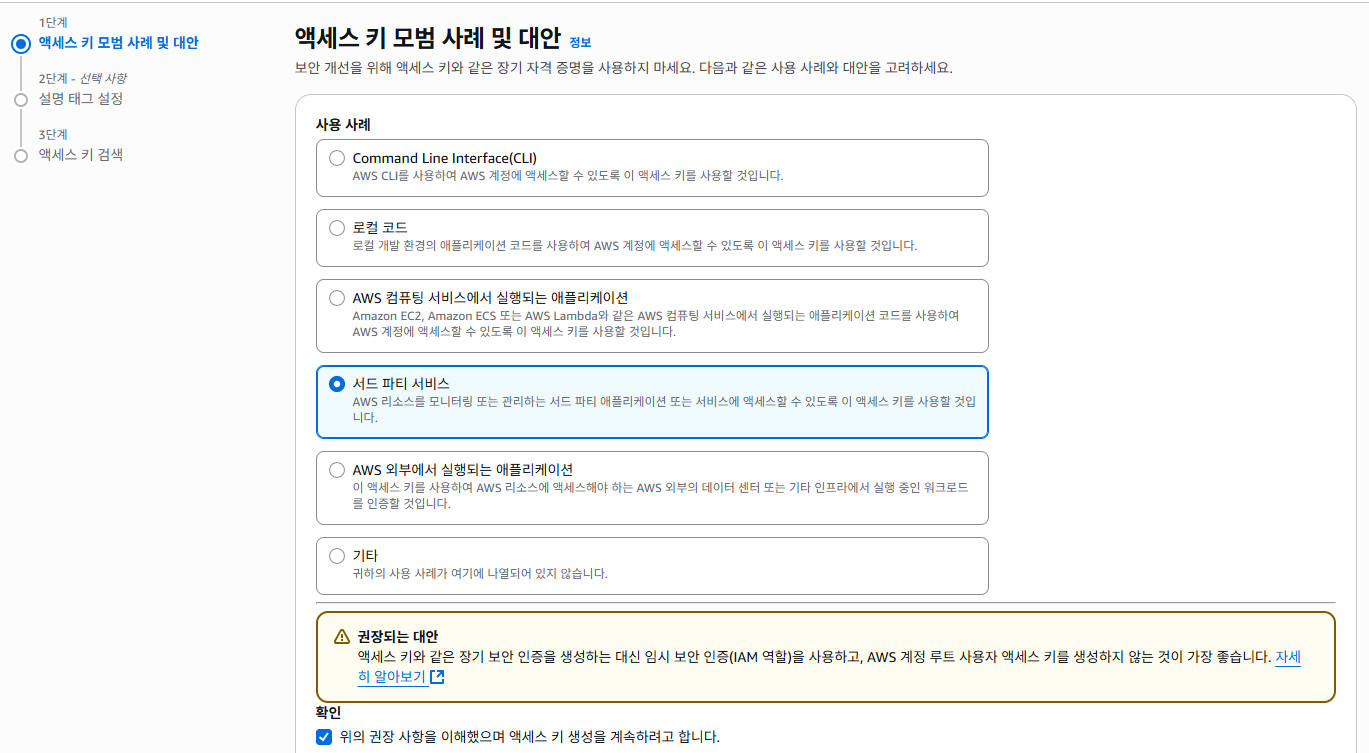
❶ 상단 검색창에 IAM 입력 → **IAM** 선택

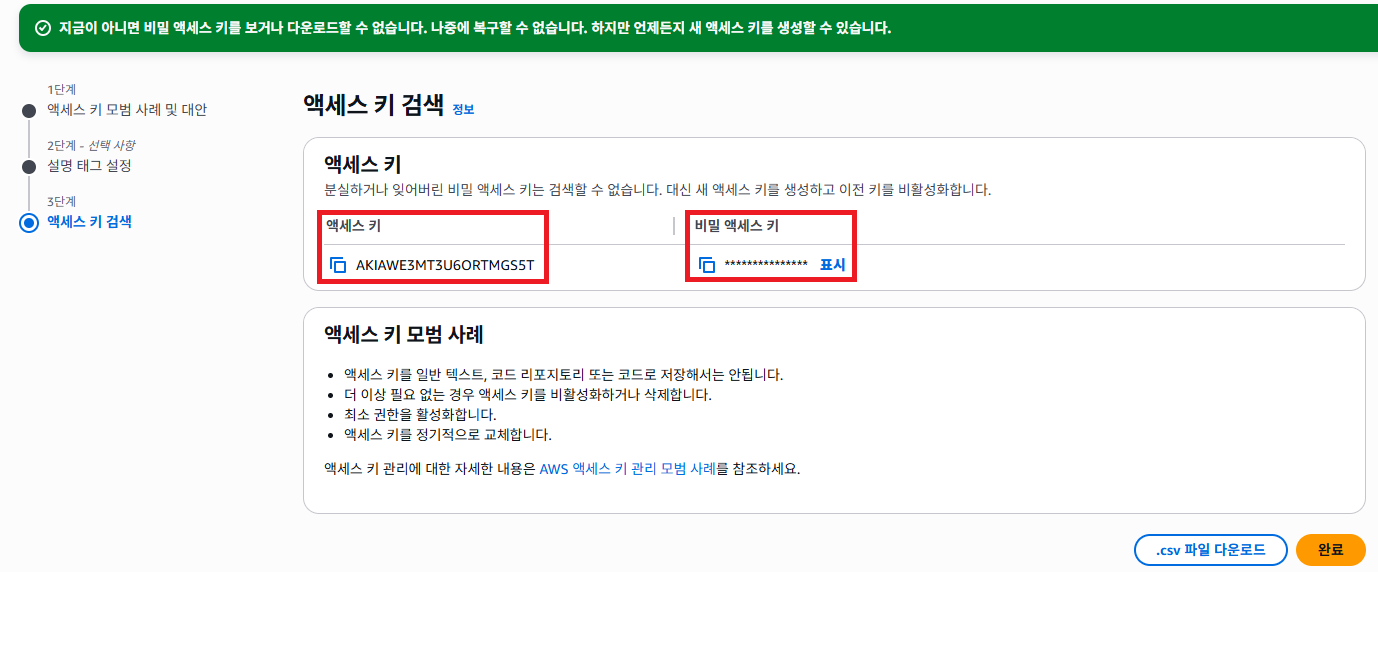
❷ IAM 사용자 생성



#### 3.5.1.4. **Access Key & Secret Access Key 생성**







#### 3.5.1.5. **AWS S3 연동**

\***스테이지 생성 및 S3 연동**

CREATE OR REPLACE STAGE 스테이지 이름

URL = 's3://S3 버킷 경로/'

CREDENTIALS = (

AWS\_KEY\_ID = 'Access Key'

AWS\_SECRET\_KEY = ‘Secret Access Key'

);

**\*생성된 스테이지 조회**

LIST @스테이지 이름;

#### 3.5.1.6. **Snowflake 테이블에 적재**

COPY INTO 테이블 이름

FROM @스테이지 이름/샘플 데이터.파일 형식

-- CSV

FILE\_FORMAT = (TYPE = CSV, FIELD\_OPTIONALLY\_ENCLOSED\_BY='"',SKIP\_HEADER=1);

-- JSON

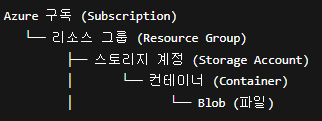
FILE\_FORMAT = (TYPE='JSON', STRIP\_OUTER\_ARRAY=TRUE);

\*테이블 적재 시, 대상 데이터베이스와 스키마가 적절한 위치에 존재하는지 확인

-- 조회

SELECT \* FROM 데이터베이스.스키마.테이블 이름 LIMIT 30;

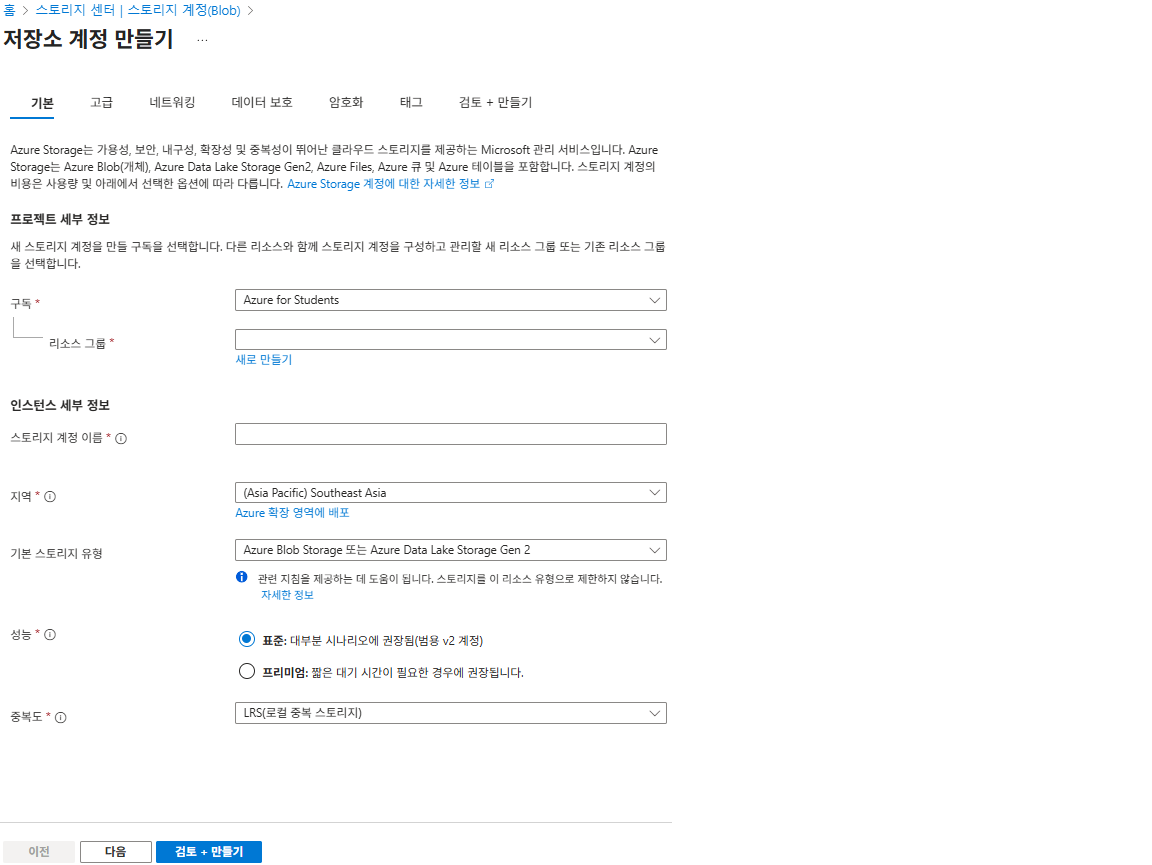
### 3.5.2. **Azure Blob Storage 연동 및 Snowflake 파일 업로드**



\*리소스 그룹은 스토리지 계정을 생성하면서 같이 생성 가능

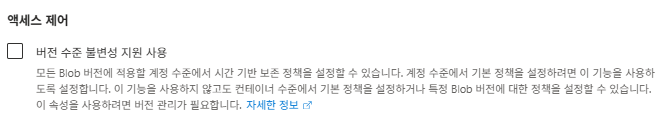
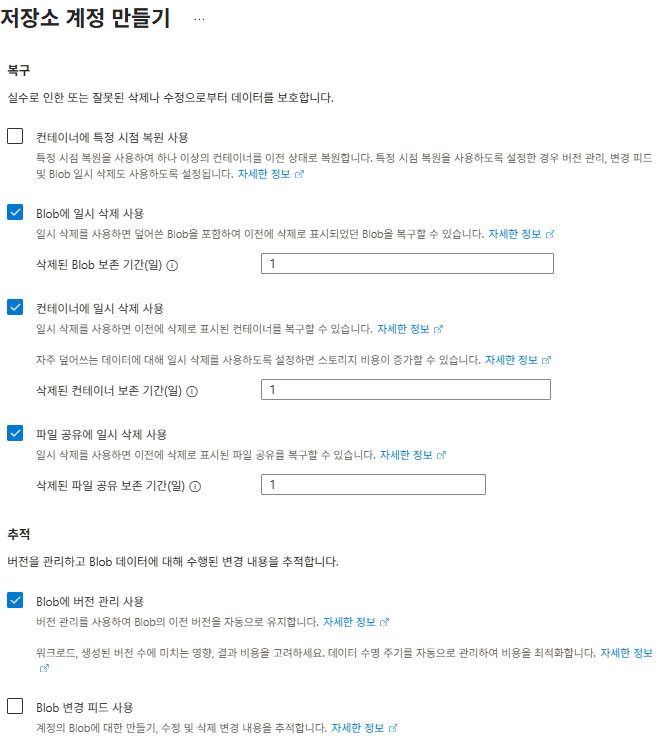
#### 3.5.2.1. [Microsoftsoft Azure 로그인](https://portal.azure.com/#home)

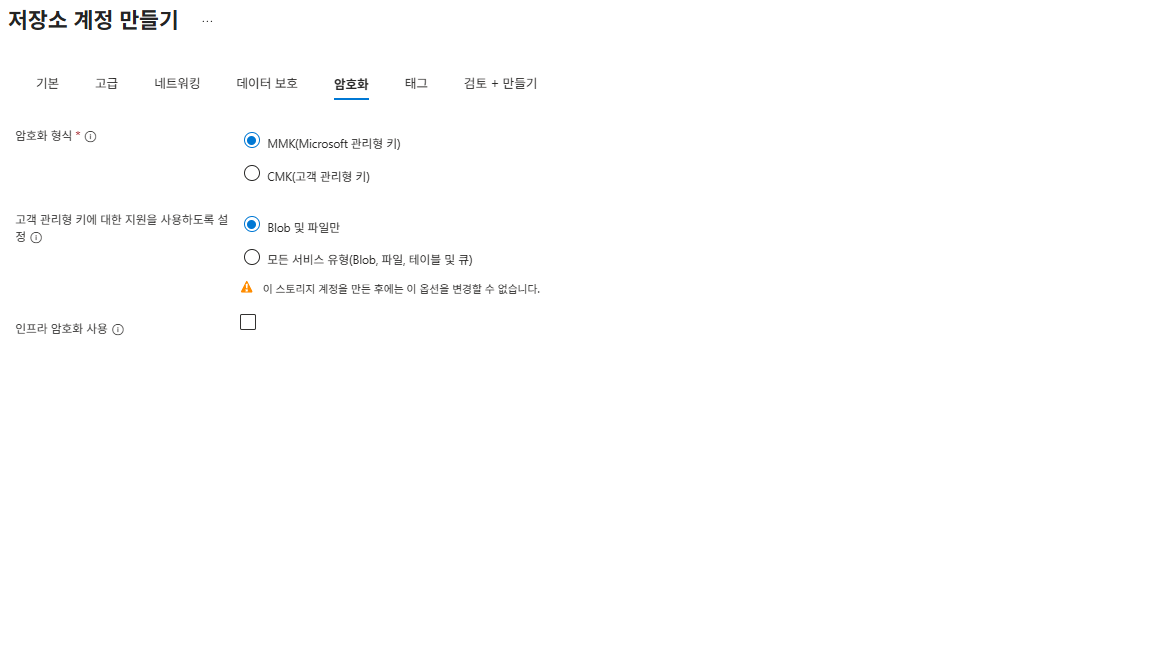
#### 3.5.2.2. **스토리지 계정 만들기**





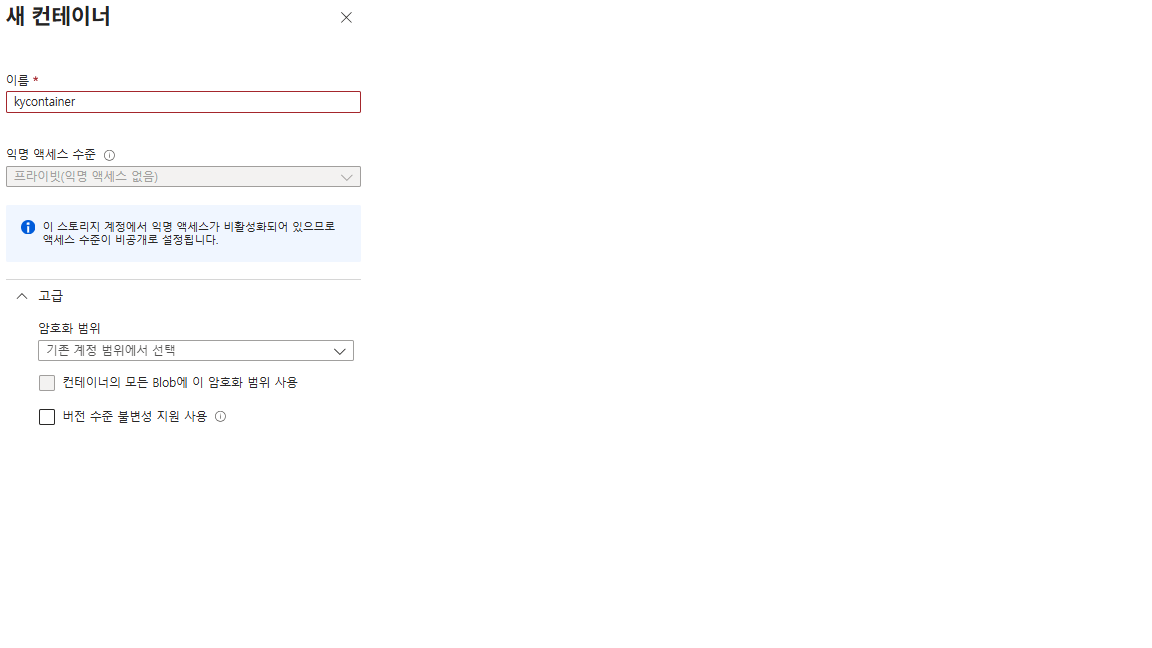
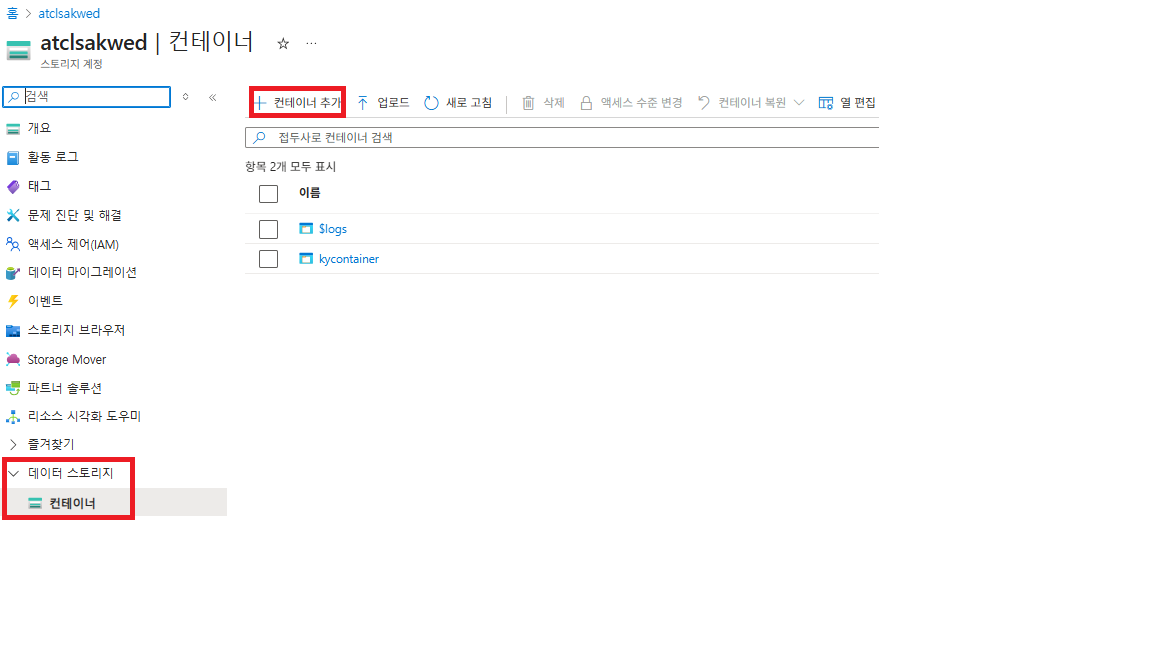


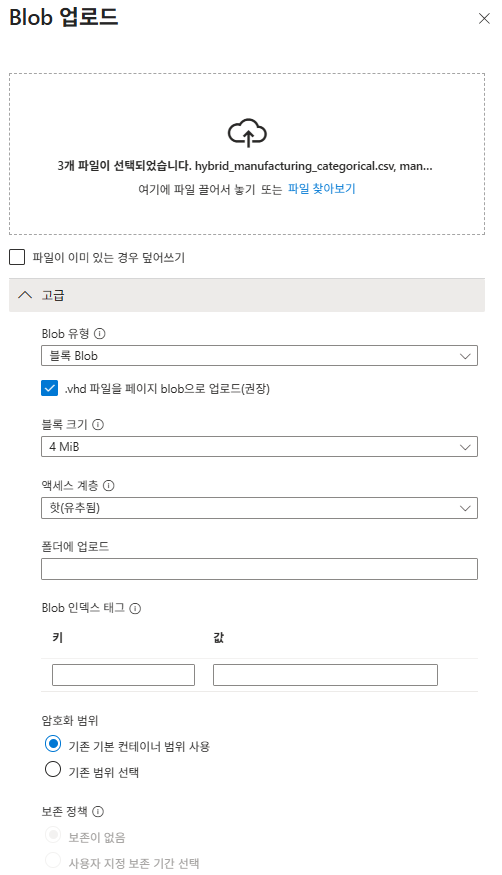
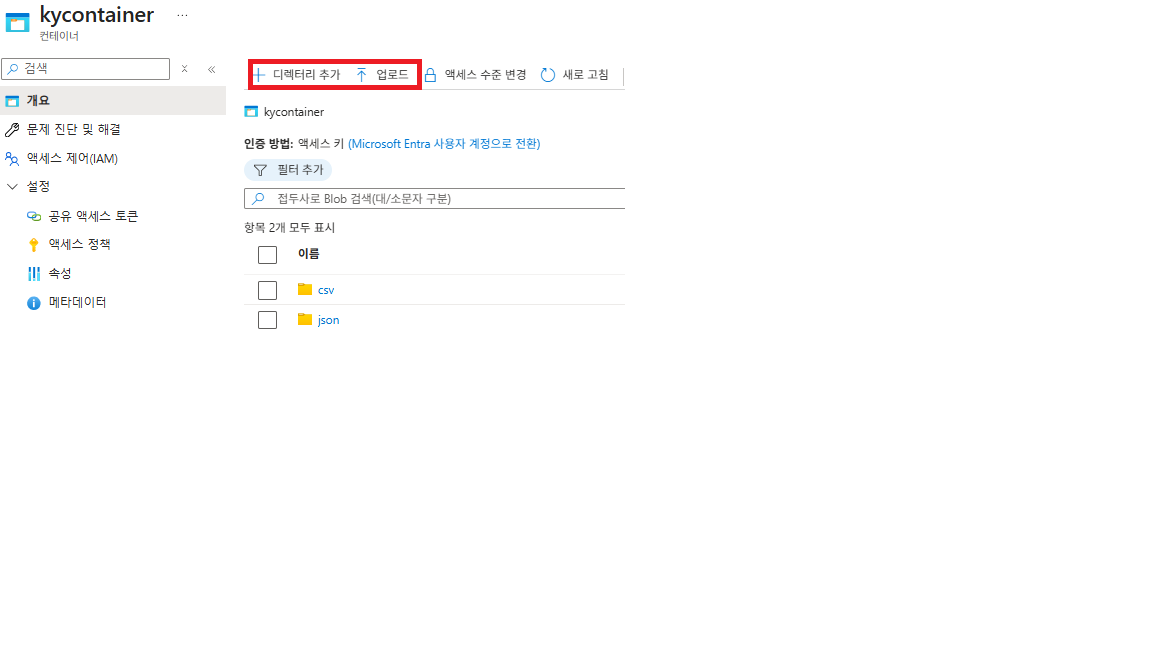




\*태그는 선택사항

#### 3.5.2.3. **컨테이너 생성 및 파일 업로드**





#### 3.5.2.4. **SAS Token 생성**



\*시작 및 만료 날짜/시간은 사용자 임의로 설정

#### 3.5.2.5. **Azure Blob Storage 연동**

**\*스테이지 생성 및 Azure Blob Storage 연동**

CREATE OR REPLACE STAGE 스테이지 이름

URL='azure://컨테이너 URL/'

CREDENTIALS=(AZURE\_SAS\_TOKEN='?SAS 토큰');

**\*컨테이너 URL은 속성에서 확인**



\*SAS 토큰 연동 시 URL 입력 구문에 ‘?’를 포함해야 Snowflake에서 정상 인식됨

\* 생성된 스테이지 조회

LIST @ 스테이지 이름;

#### 3.5.2.6. **Snowflake 테이블에 적재**

COPY INTO 테이블 이름

FROM @스테이지 이름/샘플 데이터.파일 형식

-- CSV

FILE\_FORMAT = (TYPE = CSV, FIELD\_OPTIONALLY\_ENCLOSED\_BY='"', SKIP\_HEADER=1);

-- JSON

FILE\_FORMAT = (TYPE='JSON', STRIP\_OUTER\_ARRAY=TRUE);

\*테이블 적재 시, 대상 데이터베이스와 스키마가 적절한 위치에 존재하는지 확인

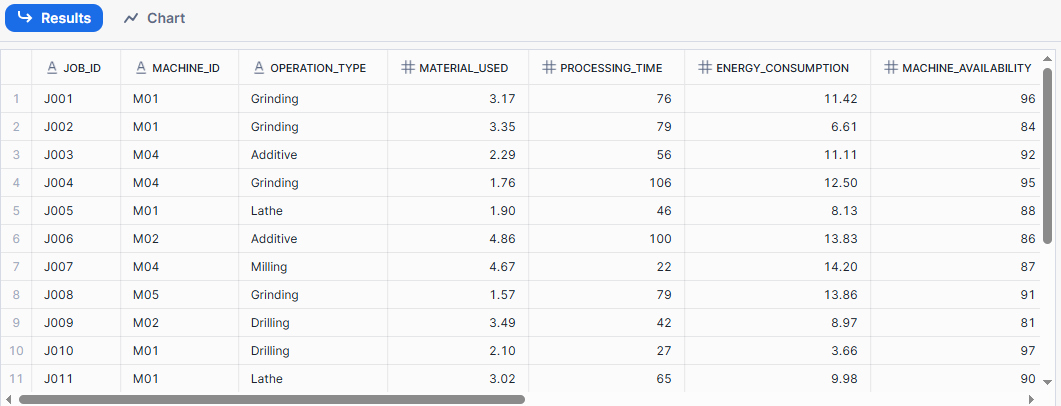
-- 조회

SELECT \* FROM 데이터베이스.스키마.테이블 이름 LIMIT 30;

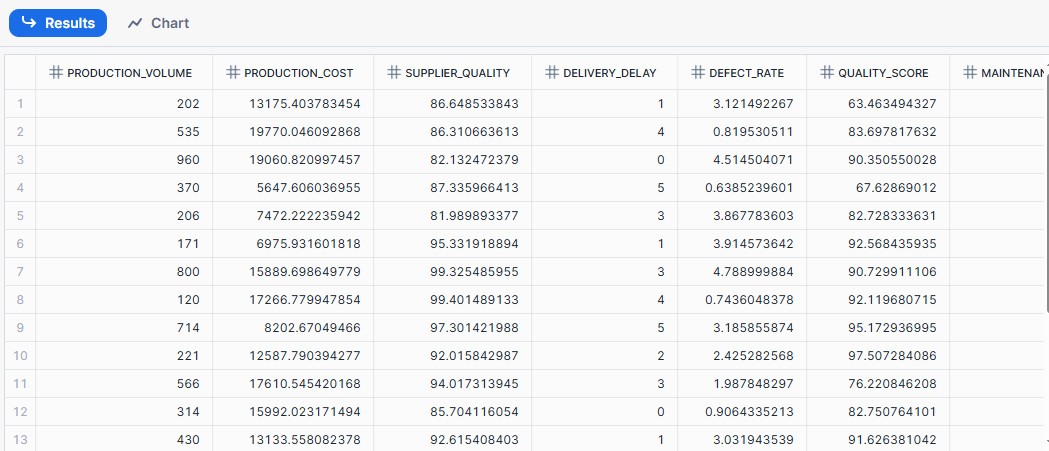
## 3.6. 결과

### 3.6.1. CSV

-- **HYBRID\_MANUFACTURING\_CATEGORICAL**



-- **MANUFACTURING\_DEFECT\_DATASET**



### 3.6.2. JSON

-- **INTENTS**



# 4. 2주차: Silver → Gold

• Silver Layer에서는 **컬럼명 표준화**와 **품질 규칙**을 적용

기존 Bronze 데이터뿐만 아니라 추후 유입될 데이터까지 고려한 유연하고

일관된 데이터 구조 설계

• Gold Layer에서는 Silver 테이블을 바탕으로 필요한 컬럼만 **요약·집계**하여

**주제별 요약 테이블**을 생성하고, 이를 시각화 및 대시보드로 구현

## 4.1. Silver 데이터베이스, 스키마, 테이블 생성

### 4.1.1. **데이터베이스&스키마 생성 및 사용**

-- 데이터베이스 및 스키마 생성

CREATE DATABASE SILVER\_DB;

CREATE SCHEMA CLEANED\_TRANSFORMED;

-- 데이터베이스.스키마 사용

USE SILVER\_DB.CLEANED\_TRANSFORMED;

### 4.1.2. **테이블 생성 및 조회**

• 정형 데이터(**CSV**)는 BRONZE 단계의 기존 테이블을 활용하고,

반정형 데이터(**JSON**)는 BRONZE 단계의 Stage를 재사용하여 VARIANT 타입으로

적재· FLATTEN 함수 사용으로 조회

#### 4.1.2.1. **MANUFACTURING\_DEFECT\_CLEAN**

-- MANUFACTURING\_DEFECT\_CLEAN

CREATE OR REPLACE TABLE SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN (

PRODUCTION\_ID STRING NOT NULL, -- 생산 식별자

-- 생산량을 등급으로 나누어 골드 레이어에서 집계하기 편하게 OUTPUT\_LEVEL컬럼 추가

OUTPUT\_LEVEL STRING, -- 생산량 등급

PRODUCTION\_VOLUME NUMBER(10,0), -- 생산량

PRODUCTION\_COST\_USD FLOAT, -- 생산 비용

SUPPLIER\_QUALITY\_PERCENT FLOAT, -- 공급 업체 품질

DELIVERY\_DELAY\_DAYS NUMBER(3,0), -- 납품 지연

DEFECT\_RATE\_PERCENT FLOAT, -- 불량률

QUALITY\_SCORE\_PERCENT FLOAT, -- 품질 점수

MAINTENANCE\_HOURS NUMBER(3,0), -- 유지보수 시간

DOWNTIME\_PERCENT FLOAT, -- 가동 중지 시간 비율

INVENTORY\_TURNOVER\_RATIO FLOAT, -- 재고 회전률

STOCKOUT\_RATE\_PERCENT FLOAT, -- 품절률

WORKER\_PRODUCTIVITY\_PERCENT FLOAT, -- 노동자 생산성

SAFETY\_INCIDENT\_COUNT NUMBER(3,0), -- 안전 사고

ENERGY\_CONSUMPTION\_KWH FLOAT, -- 에너지 소비량

ENERGY\_EFFICIENCY\_RATIO FLOAT, -- 에너지 효율성

ADDITIVE\_PROCESS\_HOURS FLOAT, -- 첨가 공정 시간

ADDITIVE\_MATERIAL\_COST\_USD FLOAT, -- 첨가 재료 비용

DEFECT\_STATUS STRING -- 불량 상태

)

AS

WITH base AS (

SELECT

PRODUCTION\_VOLUME,

PRODUCTION\_COST,

SUPPLIER\_QUALITY,

DELIVERY\_DELAY,

DEFECT\_RATE,

QUALITY\_SCORE,

MAINTENANCE\_HOURS,

DOWNTIME\_PERCENTAGE,

INVENTORY\_TURNOVER,

STOCKOUT\_RATE,

WORKER\_PRODUCTIVITY,

SAFETY\_INCIDENTS,

ENERGY\_CONSUMPTION,

ENERGY\_EFFICIENCY,

ADDITIVE\_PROCESS\_TIME,

ADDITIVE\_MATERIAL\_COST,

DEFECT\_STATUS

FROM BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET

WHERE PRODUCTION\_VOLUME IS NOT NULL

)

SELECT

CONCAT('P', LPAD(ROW\_NUMBER() OVER (ORDER BY PRODUCTION\_VOLUME DESC), 4, '0')) AS PRODUCTION\_ID,

CASE

WHEN PRODUCTION\_VOLUME BETWEEN 100 AND 324 THEN 'LOW'

WHEN PRODUCTION\_VOLUME BETWEEN 325 AND 549 THEN 'MID\_LOW'

WHEN PRODUCTION\_VOLUME BETWEEN 550 AND 774 THEN 'MID\_HIGH'

WHEN PRODUCTION\_VOLUME BETWEEN 775 AND 999 THEN 'HIGH'

ELSE 'UNKNOWN'

END AS OUTPUT\_LEVEL,

PRODUCTION\_VOLUME,

PRODUCTION\_COST,

SUPPLIER\_QUALITY,

DELIVERY\_DELAY,

DEFECT\_RATE,

QUALITY\_SCORE,

MAINTENANCE\_HOURS,

DOWNTIME\_PERCENTAGE,

INVENTORY\_TURNOVER,

STOCKOUT\_RATE,

WORKER\_PRODUCTIVITY,

SAFETY\_INCIDENTS,

ENERGY\_CONSUMPTION,

ENERGY\_EFFICIENCY,

ADDITIVE\_PROCESS\_TIME,

ADDITIVE\_MATERIAL\_COST,

CASE

WHEN DEFECT\_STATUS IN (FALSE, 0) THEN '낮음'

WHEN DEFECT\_STATUS IN (TRUE, 1) THEN '높음'

ELSE 'Undefined'

END AS DEFECT\_STATUS

FROM base;

-- 데이터 수로 적재 확인

SELECT COUNT(\*) FROM BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET;

SELECT COUNT(\*) FROM SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN;

표준화 및 품질 규칙 정의:

• 샘플 데이터를 수집한 문서의 명확한 단위·범위 정보를 반영해 기존 컬럼명에

단위 추가

#### 4.1.2.2. **HYBRID\_MANUFACTURING\_CLEAN**

-- HYBRID\_MANUFACTURING\_CLEAN

CREATE OR REPLACE TABLE SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN (

Job\_ID VARCHAR(10) NOT NULL, -- 작업 식별자

Machine\_ID VARCHAR(10) NOT NULL, -- 해당 작업이 수행된 기계의 식별자

Operation\_Type VARCHAR(100), -- 공정 유형

Material\_Used NUMBER(10,2), -- 사용된 재료의 양

Processing\_Time NUMBER(10,0), -- 작업 처리 시간

Energy\_Consumption NUMBER(10,2), -- 에너지 소비량

Machine\_Availability NUMBER(10,0), -- 기계 가용성

-- 데이터 타입이 명확한 시간 컬럼명 표준화

Scheduled\_Start\_TS TIMESTAMP\_NTZ, -- 예정 시작 시간

Scheduled\_End\_TS TIMESTAMP\_NTZ, -- 예정 종료 시간

Actual\_Start\_TS TIMESTAMP\_NTZ, -- 실제 시작 시간

Actual\_End\_TS TIMESTAMP\_NTZ, -- 실제 종료 시간

Job\_Status VARCHAR(100), -- 작업 상태

Optimization\_Category VARCHAR(100) -- 최적화 범주

)

AS

WITH base AS (

SELECT

Job\_ID,

Machine\_ID,

Operation\_Type,

Material\_Used,

Processing\_Time,

Energy\_Consumption,

Machine\_Availability,

Scheduled\_Start,

Scheduled\_End,

Actual\_Start,

Actual\_End,

Job\_Status,

Optimization\_Category

FROM BRONZE\_DB.RAW\_CSV.HYBRID\_MANUFACTURING\_CATEGORICAL

WHERE Job\_ID IS NOT NULL

AND Machine\_ID IS NOT NULL

)

SELECT

Job\_ID,

Machine\_ID,

Operation\_Type,

Material\_Used,

Processing\_Time,

Energy\_Consumption,

Machine\_Availability,

Scheduled\_Start AS Scheduled\_Start\_TS,

Scheduled\_End AS Scheduled\_End\_TS,

Actual\_Start AS Actual\_Start\_TS,

Actual\_End AS Actual\_End\_TS,

Job\_Status,

Optimization\_Category

FROM base;

-- 데이터 수로 적재 확인

SELECT COUNT(\*) FROM BRONZE\_DB.RAW\_CSV.HYBRID\_MANUFACTURING\_CATEGORICAL;

SELECT COUNT(\*) FROM SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN;

표준화 및 품질 규칙 정의:

• 샘플 데이터를 수집한 문서에 단위·범위 정보가 명확하지 않아, 브론즈 테이블의

컬럼명을 기준으로 동일하게 적용(시간 관련 컬럼 제외)

#### 4.1.2.3. **INTENTS**

-- 기존에 진행했던 BRONZE 단계 참고

COPY INTO BRONZE\_DB.RAW\_JSON.INTENTS

-- SILVER 단계에서 필요한 부분

FROM @MYSTAGE/json/intents.json

FILE\_FORMAT = (TYPE='JSON' STRIP\_OUTER\_ARRAY=TRUE);

-- SILVER 단계에서는 별도 Stage를 두지 않고, BRONZE 단계에서 생성된 Stage를 재사용

COPY INTO SILVER\_DB.CLEANED\_TRANSFORMED.INTENTS

FROM @BRONZE\_DB.RAW\_JSON.MYSTAGE/json/intents.json

FILE\_FORMAT = (TYPE='JSON' STRIP\_OUTER\_ARRAY=TRUE);

-- JSON 데이터를 적재한 반정형 테이블은 조회할 때 FLATTEN 함수 사용해서 배열을 펼쳐 조회

SELECT

VALUE:tag::STRING AS TAG,

VALUE:patterns::ARRAY AS PATTERNS,

VALUE:responses::ARRAY AS RESPONSES

FROM intents,

LATERAL FLATTEN(INPUT => DATA:intents);

INTENTS 테이블 조회 결과



\* 현재 INTENTS 테이블은 위 단계까지만 진행

• 본 프로젝트의 주요 목표는 Snowflake 환경에 대한 이해 및 데이터 정제·적재

과정의 학습이므로 프로젝트 이후 단계에서 추가 진행할 예정

## 4.2. **Gold 데이터베이스, 스키마, 테이블 생성**

### 4.2.1. **데이터베이스&스키마 생성**

-- 데이터베이스 및 스키마 생성

DATABASE GOLD\_DB;

-- 각 테이블마다 스키마 별도 생성

CREATE SCHEMA GOLD\_DB.HYBRID\_MANUFACTURING\_KPI;

CREATE SCHEMA GOLD\_DB.MANUFACTURING\_DEFECT\_KPI;

### 4.2.2. **테이블 생성 및 조회**

#### 4.2.2.1. **FCT\_OPERATION\_SUMMARY1)**

-- MANUFACTURING\_DEFECT\_KPI 스키마 사용

USE GOLD\_DB.MANUFACTURING\_DEFECT\_KPI;

-- GOLD\_DB.MANUFACTURING\_DEFECT\_KPI.FCT\_OPERATION\_SUMMARY 테이블 생성

CREATE OR REPLACE TABLE GOLD\_DB.MANUFACTURING\_DEFECT\_KPI.FCT\_OPERATION\_SUMMARY AS

SELECT

OUTPUT\_LEVEL, -- 생산량 등급

-- 생산량 등급별 평균 KPI

ROUND(AVG(PRODUCTION\_COST\_USD), 2) AS PRODUCTION\_COST\_USD\_AVG, -- 생산 비용

ROUND(AVG(MAINTENANCE\_HOURS), 2) AS MAINTENANCE\_HOURS\_AVG, -- 유지 보수 시간

ROUND(AVG(ENERGY\_CONSUMPTION\_KWH), 2) AS ENERGY\_CONSUMPTION\_KWH\_AVG, -- 에너지 소비량

ROUND(AVG(ADDITIVE\_PROCESS\_HOURS), 2) AS ADDITIVE\_PROCESS\_HOURS\_AVG, -- 첨가 공정 시간

ROUND(AVG(INVENTORY\_TURNOVER\_RATIO), 2) AS INVENTORY\_TURNOVER\_RATIO\_AVG, -- 재고 회전률

ROUND(AVG(STOCKOUT\_RATE\_PERCENT), 2) AS STOCKOUT\_RATE\_PERCENT\_AVG, -- 품절률

ROUND(AVG(ADDITIVE\_MATERIAL\_COST\_USD), 2) AS ADDITIVE\_MATERIAL\_COST\_USD\_AVG, -- 첨가 재료비

-- 불량 상태 비율 (범주형 -> 비율로 변환)

ROUND(

100 \* SUM(CASE WHEN DEFECT\_STATUS = '높음' THEN 1 ELSE 0 END)

/ NULLIF(COUNT(\*), 0),

2

) AS DEFECT\_HIGH\_PERCENT,

ROUND(

100 \* SUM(CASE WHEN DEFECT\_STATUS = '낮음' THEN 1 ELSE 0 END)

/ NULLIF(COUNT(\*), 0),

2

) AS DEFECT\_LOW\_PERCENT

FROM SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN

GROUP BY OUTPUT\_LEVEL

ORDER BY

CASE

WHEN OUTPUT\_LEVEL = 'LOW' THEN 1

WHEN OUTPUT\_LEVEL = 'MID\_LOW' THEN 2

WHEN OUTPUT\_LEVEL = 'MID\_HIGH' THEN 3

WHEN OUTPUT\_LEVEL = 'HIGH' THEN 4

ELSE 'UNKNOWN'

END;

-- FCT\_OPERATION\_SUMMARY 테이블 조회

SELECT \* FROM GOLD\_DB.MANUFACTURING\_DEFECT\_KPI.FCT\_OPERATION\_SUMMARY;

FCT\_OPERATION\_SUMMARY 테이블 조회 결과:



• 컬럼 수: 10

1) FCT\_OPERATION\_SUMMARY: 생산 지표 집계 테이블

• **생산량 등급별**로 생산 비용, 유지보수 시간, 에너지 소비량, 불량 상태 비율 등을 집계

#### 4.2.2.2. **FCT\_JOB\_STATUS\_SUMMARY1)**

-- HYBRID\_MANUFACTURING\_KPI 스키마 사용

USE GOLD\_DB.HYBRID\_MANUFACTURING\_KPI;

-- GOLD\_DB.HYBRID\_MANUFACTURING\_KPI.FCT\_JOB\_STATUS\_SUMMARY 테이블 생성

CREATE OR REPLACE TABLE GOLD\_DB.HYBRID\_MANUFACTURING\_KPI.FCT\_JOB\_STATUS\_SUMMARY AS

WITH

-- Optimization\_Category 비율 계산

opt\_ratio AS (

SELECT

Job\_Status,

Optimization\_Category,

COUNT(\*) AS cnt,

ROUND(

COUNT(\*) \* 100.0 / NULLIF(SUM(COUNT(\*)) OVER (PARTITION BY Job\_Status), 0),

2

) AS category\_ratio\_percent

FROM SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN

GROUP BY Job\_Status, Optimization\_Category

),

-- Job\_Status별 전체 건수 계산

status\_total AS (

SELECT

Job\_Status,

COUNT(\*) AS total\_count

FROM SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN

GROUP BY Job\_Status

)

-- 최종 테이블 생성

SELECT

h.Job\_Status, -- 작업 상태

s.total\_count, -- 작업 상태 Count

o.Optimization\_Category, -- 최적화 범주

o.category\_ratio\_percent, -- 최적화 범주 비율

-- 지연 시간, 처리 시간, 에너지 소비량, 기계 가용성 평균

ROUND(AVG(DATEDIFF(minute, h.Scheduled\_End\_TS, h.Actual\_End\_TS)), 2) AS avg\_delay\_min,

ROUND(AVG(h.Processing\_Time), 2) AS processing\_time\_avg,

ROUND(AVG(h.Energy\_Consumption), 2) AS energy\_consumption\_avg,

ROUND(AVG(h.Machine\_Availability), 2) AS machine\_availability\_avg

FROM SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN h

LEFT JOIN opt\_ratio o

ON h.Job\_Status = o.Job\_Status

AND h.Optimization\_Category = o.Optimization\_Category

LEFT JOIN status\_total s

ON h.Job\_Status = s.Job\_Status

GROUP BY

h.Job\_Status,

s.total\_count,

o.Optimization\_Category,

o.category\_ratio\_percent

ORDER BY

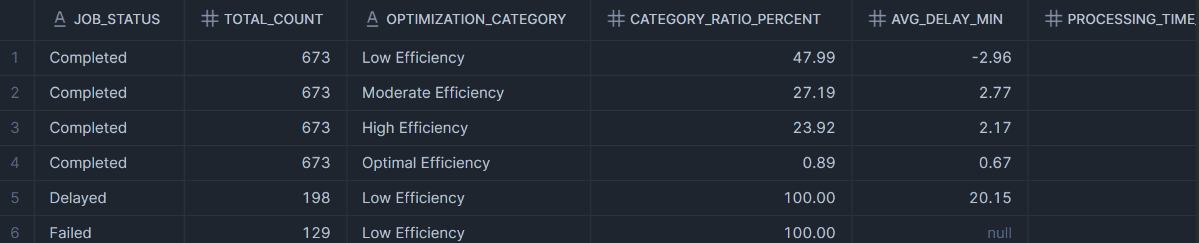
h.Job\_Status,

o.category\_ratio\_percent DESC;

-- FCT\_JOB\_STATUS\_SUMMARY 테이블 조회

SELECT \* FROM GOLD\_DB.HYBRID\_MANUFACTURING\_KPI.FCT\_JOB\_STATUS\_SUMMARY;

FCT\_JOB\_STATUS\_SUMMARY 테이블 조회 결과:



• 컬럼 수: 8

1) FCT\_JOB\_STATUS\_SUMMARY: 작업 상태별 주요 지표 집계

• **Job\_Status(실패·지연·완료)에 따라** 최적화 범주(Category) 수·비율, 평균 지연 시간,

평균 처리 시간, 평균 에너지 소비량, 평균 기계 가용성 등을 집계

#### 4.2.2.3. **FCT\_MACHINE\_SUMMARY1)**

-- GOLD\_DB.HYBRID\_MANUFACTURING\_KPI.FCT\_MACHINE\_SUMMARY 테이블 생성

CREATE OR REPLACE TABLE GOLD\_DB.HYBRID\_MANUFACTURING\_KPI.FCT\_MACHINE\_SUMMARY AS

WITH operation\_ratio AS (

-- 각 Machine\_ID × Operation\_Type별 비율(%) 계산

SELECT

Machine\_ID,

Operation\_Type,

COUNT(\*) AS operation\_count,

ROUND(

COUNT(\*) \* 100.0 / NULLIF(SUM(COUNT(\*)) OVER (PARTITION BY Machine\_ID), 0),

2

) AS operation\_ratio\_percent

FROM SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN

GROUP BY Machine\_ID, Operation\_Type

)

-- 기계별 주요 KPI + 공정 비율 결합

SELECT

h.Machine\_ID, -- 기계 식별자

o.Operation\_Type, -- 공정 유형

o.operation\_ratio\_percent, -- 기계별 공정 유형 비율

ROUND(AVG(h.Machine\_Availability), 2) AS Machine\_Availability\_avg, -- 기계 가용성

ROUND(AVG(h.Energy\_Consumption), 2) AS Energy\_Consumption\_avg, -- 에너지 소비량

ROUND(AVG(h.Material\_Used), 2) AS Material\_Used\_avg, -- 사용된 재료의 양

FROM SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN h

LEFT JOIN operation\_ratio o

ON h.Machine\_ID = o.Machine\_ID

GROUP BY

h.Machine\_ID,

o.Operation\_Type,

o.operation\_ratio\_percent

ORDER BY

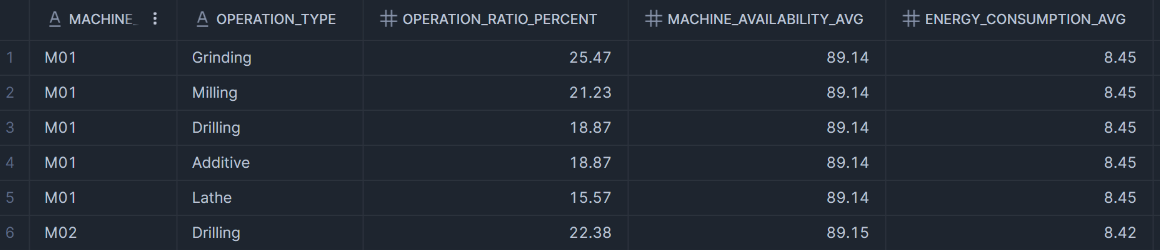
h.Machine\_ID ASC,

o.operation\_ratio\_percent DESC;

-- FCT\_MACHINE\_SUMMARY 테이블 조회

SELECT \* FROM GOLD\_DB.HYBRID\_MANUFACTURING\_KPI.FCT\_MACHINE\_SUMMARY;

FCT\_MACHINE\_SUMMARY 테이블 조회 결과:



• 컬럼 수: 6

1) FCT\_MACHINE\_SUMMARY: 기계별 공정 유형 주요 지표 집계

• **기계별 공정 유형** 비율, 평균 기계 가용성, 평균 에너지 소비량,

평균 사용 재료량을 집계·요약

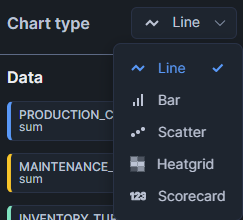
## 4.3. **Snowsight 차트 → 쿼리 결과 시각화**

### 4.3.1. Chart



• 시각화를 원하는 **골드 레이어 요약·집계 테이블을 조회**한 후, **결과창(Results)**에서 **Chart** 기능을 선택하여 시각화 수행

### 4.3.2. Chart type



• **Chart type**: **시각화 유형**을 선택하여 데이터의 특성에 맞는 형태로 표현

• **Line**: 시간 또는 순서에 따른 **변화 추세(Trend)**를 시각적으로 표현할 때 사용

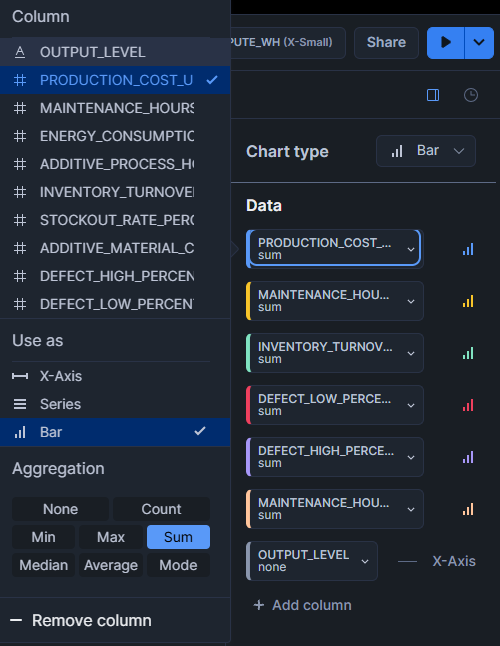
• **Bar**: 범주(category)별 **비교나 구성 비율**을 직관적으로 보여줄 때 사용

• **Scatter**: 두 변수 간의 **상관관계나 분포 패턴**을 확인할 때 사용

• **Heatgrid**: 값의 크기나 밀도를 **색상으로 구분**해 데이터 패턴을 빠르게 파악할 때 사용

• **Scorecard**: **핵심 지표(KPI)**의 단일 값을 강조하여 한눈에 보여줄 때 사용

### 4.3.3. Data



**Add Column** 기능을 통해 시각화에 사용할 **컬럼을 추가하고**, 해당 컬럼의 **차트 유형 및 집계 방식(Aggregation)**을 설정:

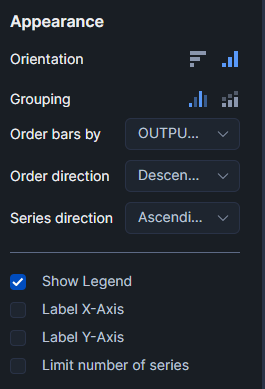
• **X-Axis**: **차트의 가로축**으로 사용할 컬럼을 지정

• **Series**: **X축 기준**으로 비교하거나 시각화 값(측정값)을 지정

• **Bar**: **차트의 유형(Bar Chart)**을 의미하며, 막대 그래프로 데이터를 표현

• **Aggregation**: 시각화 데이터의 **집계** 방식

### 4.3.4. Appearance



• **Orientation**: **차트 방향**(가로/세로)을 설정

• **Grouping**: 데이터를 **그룹화**하여 항목 간 비교 구조 지정

• **Order bars by**: **막대 정렬 기준** 컬럼 선택

• **Order direction**: **정렬 방향**(오름차순/내림차순) 지정

• **Series direction**: 시리즈(데이터 항목) 표시 **순서** 설정

• **Show Legend**: **범례(legend)** 표시 여부 선택

• **Label X-Axis/Y-Axis**: **축 이름** 표시 여부 설정

• **Limit number of series**: 표시할 시리즈(데이터 그룹) **개수 제한**

• **Label X-Axis**: X축(가로축)에 축 **제목(Label)**을 표시할지 여부를 설정

• **Label Y-Axis**: Y축(세로축)에 측정값 **이름(Label)**을 표시할지 여부를 설정

• **Limit number of series**: 차트에 표시할 데이터 시리즈의 개수를 제한하여 **복잡도**를

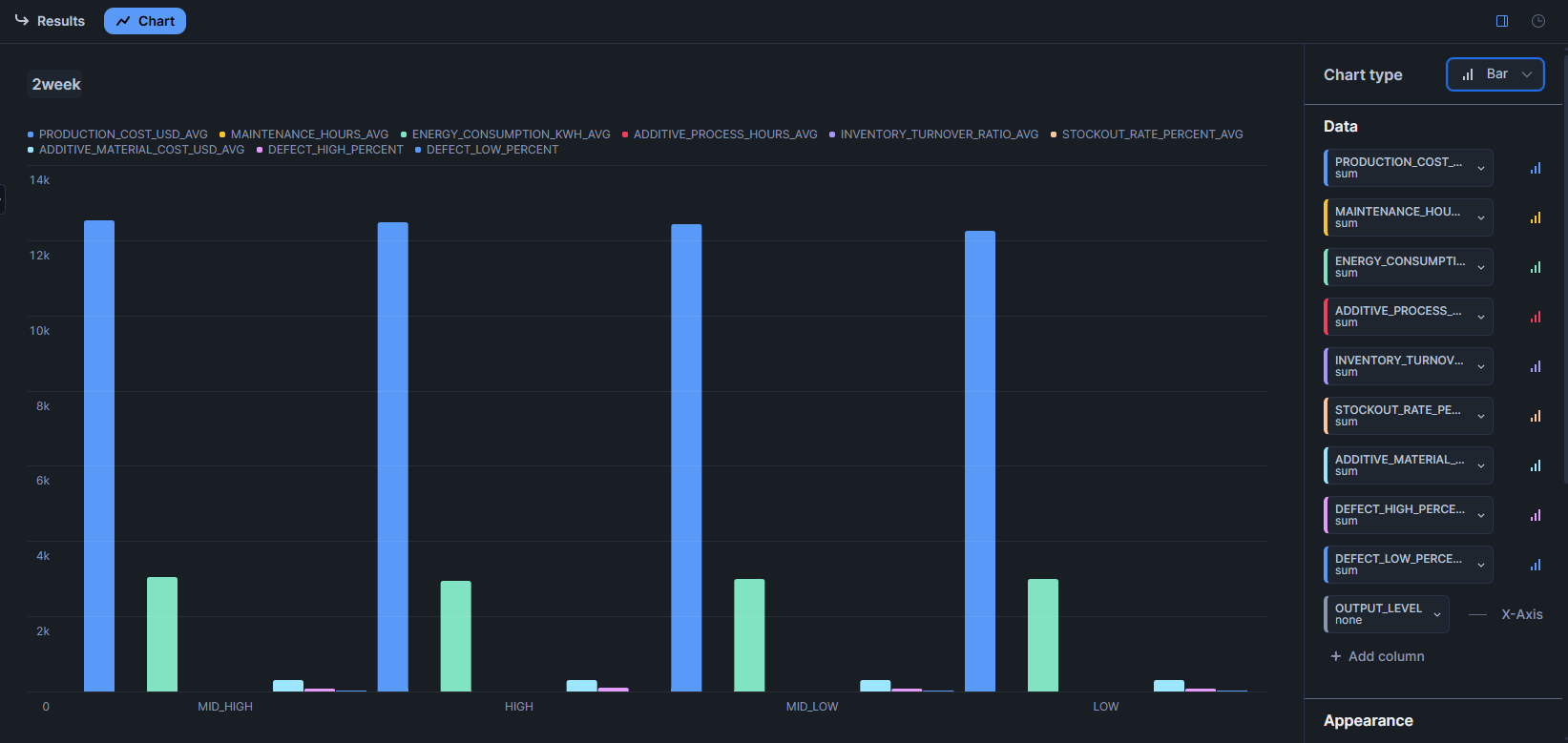
줄이고 가독성을 높임

### 4.3.5. 쿼리 결과 시각화

##### 4.3.5.1. **FCT\_OPERATION\_SUMMARY**

-- FCT\_OPERATION\_SUMMARY 테이블 조회

SELECT \* FROM GOLD\_DB.MANUFACTURING\_DEFECT\_KPI.FCT\_OPERATION\_SUMMARY;



• FCT\_OPERATION\_SUMMARY 테이블의 전체 컬럼 Bar 시각화

• 생산량 등급에 따라 달라지지 않고 **일정**함

##### 4.3.5.2. **FCT\_JOB\_STATUS\_SUMMARY**

-- FCT\_JOB\_STATUS\_SUMMARY 테이블 조회

SELECT \* FROM GOLD\_DB.HYBRID\_MANUFACTURING\_KPI.FCT\_JOB\_STATUS\_SUMMARY;



• FCT\_JOB\_STATUS\_SUMMARY 테이블의 작업 상태별 지연 및 비율 Bar 시각화

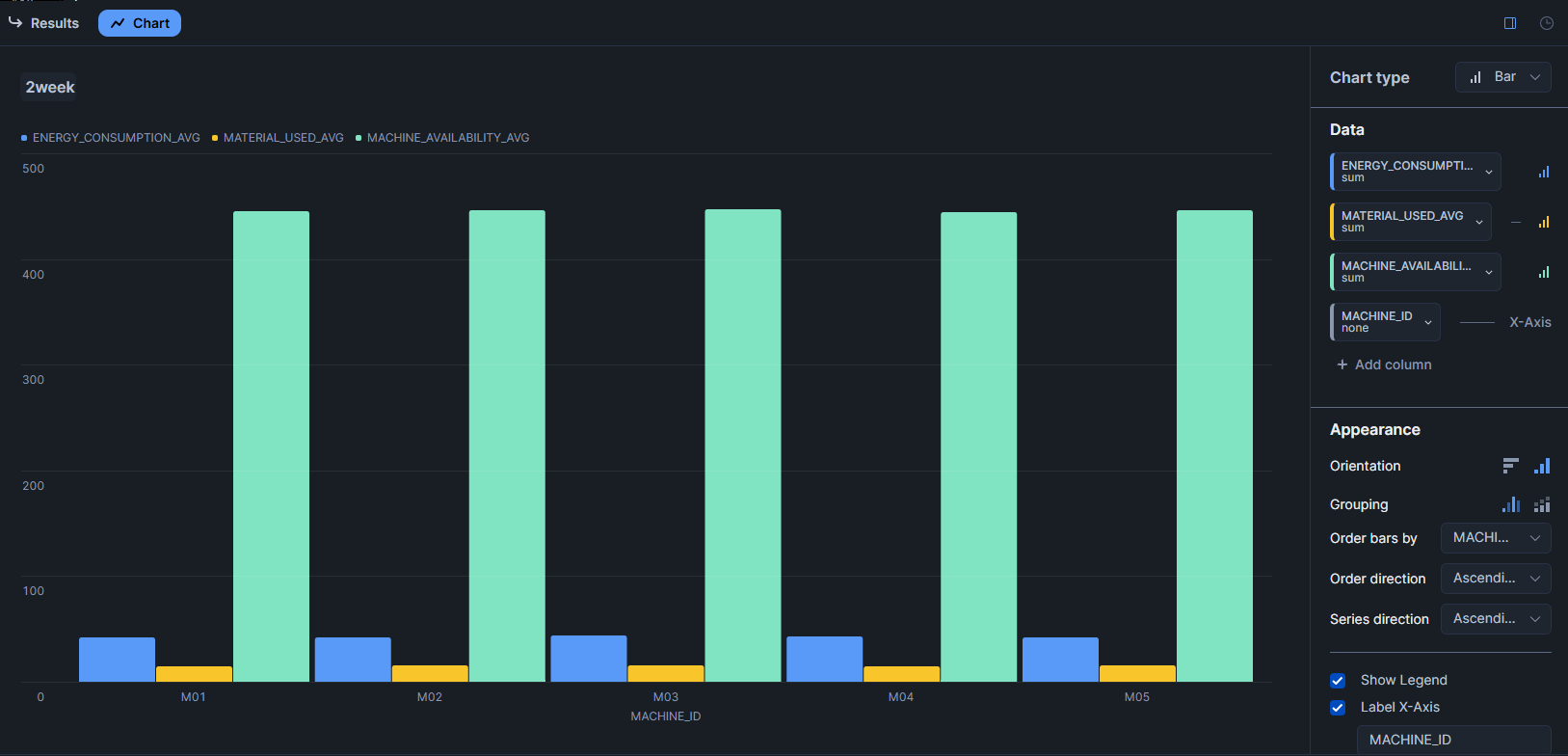
• Job\_Status가 ‘Delayed’ 또는 ‘Failed’인 경우, 해당 작업의 Optimization\_Category는

모두 ‘**Low Efficiency**’로 분류

##### 4.3.5.3. **FCT\_MACHINE\_SUMMARY**

-- FCT\_MACHINE\_SUMMARY 테이블 조회

SELECT \* FROM GOLD\_DB.HYBRID\_MANUFACTURING\_KPI.FCT\_MACHINE\_SUMMARY;



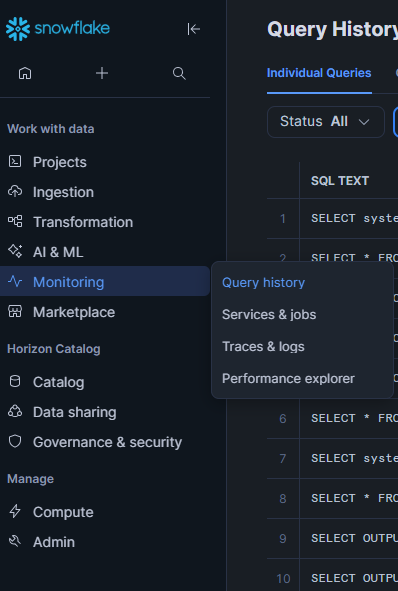
• FCT\_MACHINE\_SUMMARY 테이블의 기계별 평균 에너지 소비량, 사용된 재료 사용량,

기계 가용성 Bar 시각화

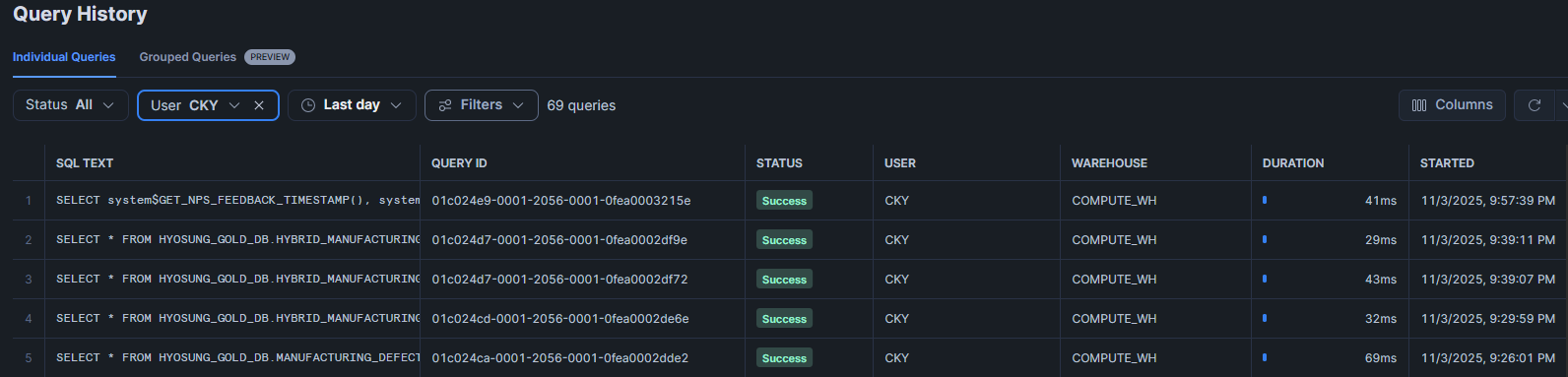
• MACHINE\_ID별 평균 에너지 소비량, 재료 사용량, 기계 가용성에서 유의미한 차이가 나타나지 않고 **전반적으로 균일**한 경향

# 5. **3주차: Snowsight 쿼리/모니터링**

## 5.1. **쿼리 이력(Query History)로 실행 시간·스캔량 분석**

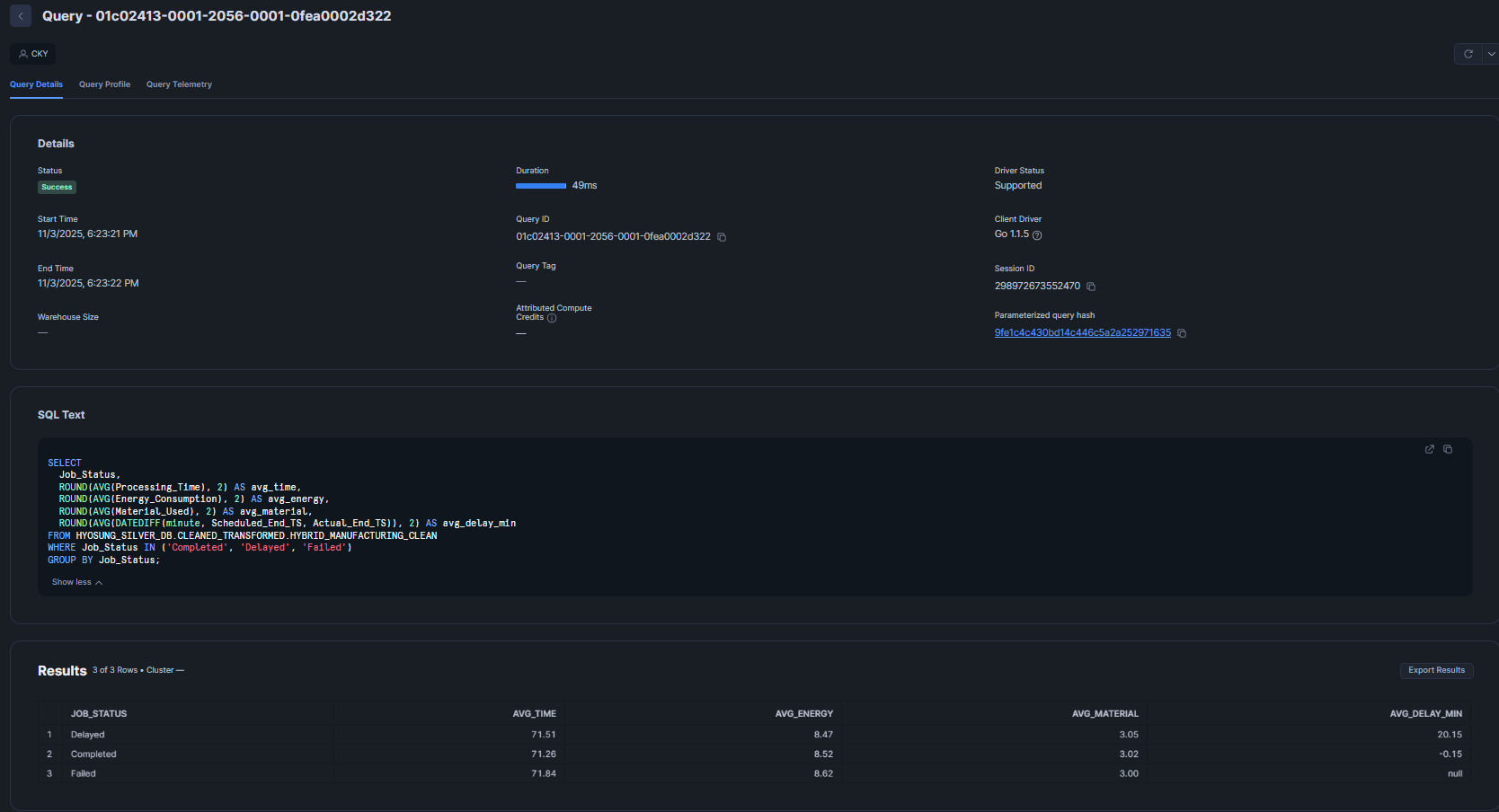


• 좌측 메뉴의 ‘**Monitoring**’ 탭에서 ‘**Query history**’를 선택하여 쿼리 수행 내역 조회



• 사용자는 수행한 모든 쿼리 내역을 확인할 수 있으며, 각 작업을 클릭하면

해당 쿼리의 **세부 실행 정보(Query Profile)**로 이동

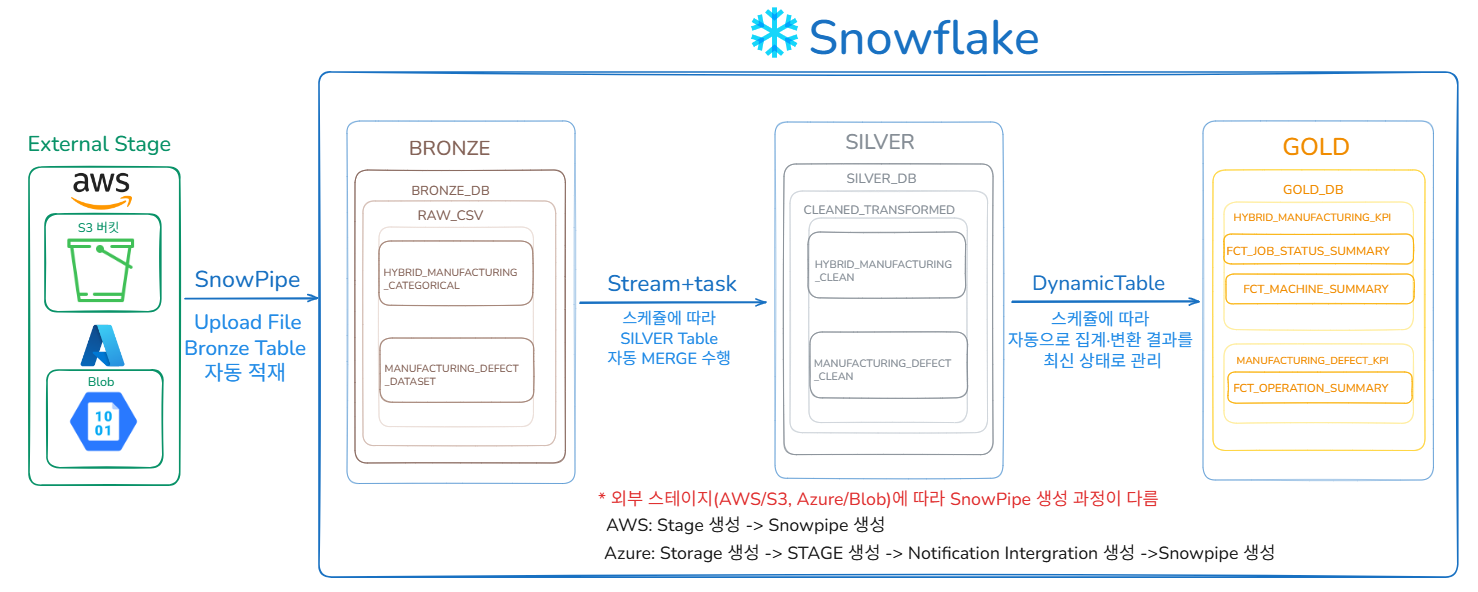


• 사용자는 각 쿼리 작업의 성공·실패·지연 여부를 확인할 수 있으며,

실행 시각·종료 시각·소요 시간 등의 **상세 정보**를 조회

• **SQL Text**를 통해 실행된 쿼리와 결과를 재실행 없이 조회 가능

# 6. **4주차: 자동화 파이프라인 구성**



• **Snowpipe · Stream+Task · Dynamic Table**을 통한 자동화 파이프라인 구성

• BRONZE(**Snowpipe**) -> SILVER(**Stream+Task**) -> GOLD(**Dynamic Table**)까지

전체 데이터 흐름을 자동화

## 6.1. **Snowpipe**

• **Bronze 단계 진행**

• 외부 스토리지(AWS S3 / Azure Blob)에 파일이 업로드되면 Snowpipe가 이를 감지하여 **자동으로 Bronze 테이블에 적재**

• Snowpipe의 자동 적재 대상이 명확해지도록, 각 Bronze 테이블과 **1:1로 매핑**되는 Stage를 생성

### 6.1.1. **AWS S3**

#### 6.1.1.1. **스테이지 생성**

CREATE OR REPLACE STAGE 스테이지 이름

URL = 's3://S3 버킷 경로/폴더 경로/'

CREDENTIALS = (

AWS\_KEY\_ID = 'Access Key'

AWS\_SECRET\_KEY = ‘Secret Access Key'

);

• 각 테이블마다 **1:1로 매핑되는 전용 스테이지** 생성하는 방식 권장

=> 테이블별 데이터 흐름이 분리되어 관리가 쉽고, 적재 오류나 모니터링도 명확함

#### 6.1.1.2. **Snowpipe 생성**

-- MANUFACTURING\_DEFECT\_DATASET

CREATE OR REPLACE PIPE PIPE NAME

AUTO\_INGEST = TRUE

AS

-- CSV 컬럼 개수 ≠ 테이블 컬럼 개수 라면 무조건 컬럼 리스트 명시

COPY INTO BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET(

PRODUCTION\_ID,

PRODUCTION\_VOLUME,

PRODUCTION\_COST,

SUPPLIER\_QUALITY,

DELIVERY\_DELAY,

DEFECT\_RATE,

QUALITY\_SCORE,

MAINTENANCE\_HOURS,

DOWNTIME\_PERCENTAGE,

INVENTORY\_TURNOVER,

STOCKOUT\_RATE,

WORKER\_PRODUCTIVITY,

SAFETY\_INCIDENTS,

ENERGY\_CONSUMPTION,

ENERGY\_EFFICIENCY,

ADDITIVE\_PROCESS\_TIME,

ADDITIVE\_MATERIAL\_COST,

DEFECT\_STATUS

)

FROM @STAGE NAME

FILE\_FORMAT = (

TYPE = CSV,

FIELD\_OPTIONALLY\_ENCLOSED\_BY = '"',

SKIP\_HEADER = 1

)

-- 파일명, 확장자 대소문자 무시(원래 구분함)

PATTERN='(?i)^.\*\\.csv$';

• Snowpipe를 자동으로 실행하려면 **AUTO\_INGEST = TRUE 옵션을 반드시 활성화**

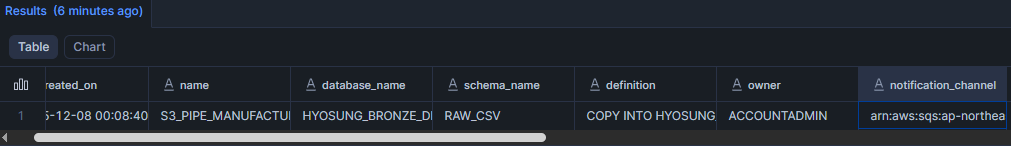
• PATTERN='(?i)^.\*\\.csv$' 옵션은 **파일 이름**과 상관없이, **대소문자 구분** 없이

모든 csv 파일만을 자동으로 식별해 적재하도록 설정

-- PIPE 구성 정보 조회

DESC PIPE NAME;

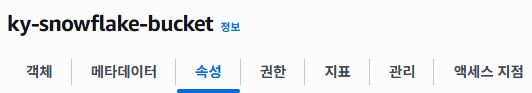
• **notification\_channel** 속성값 확인



#### 6.1.1.3. **AWS 이벤트 알림 생성**

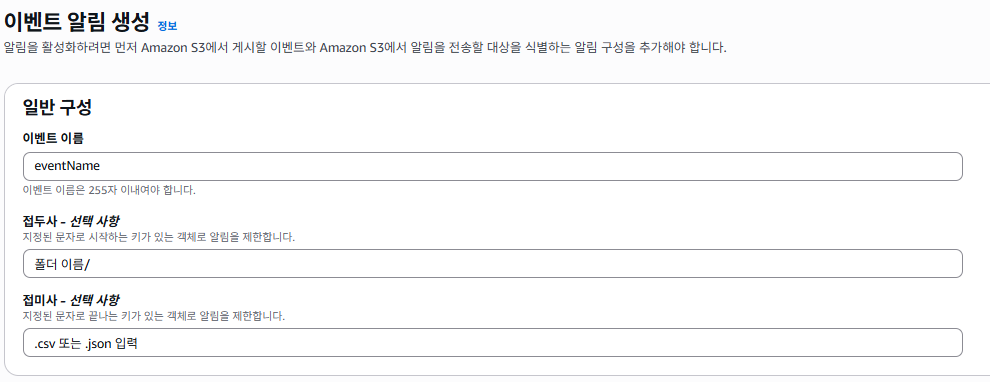
AWS 버킷에 파일이 업로드되면 Snowpipe가 이를 자동으로 감지할 수 있도록,

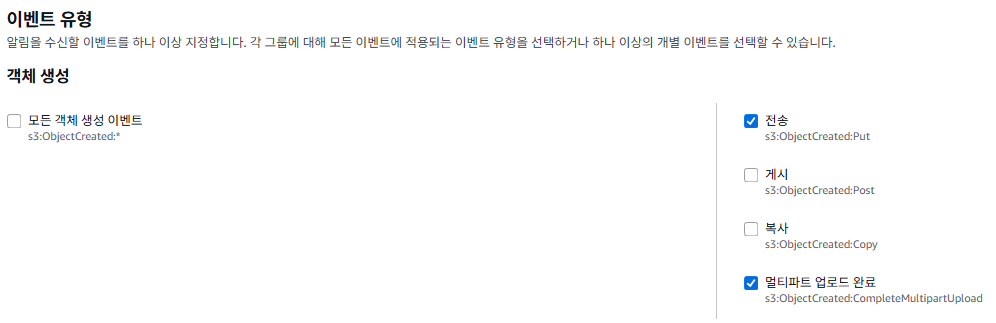
AWS 측에서 **이벤트 알림(Event Notification)**을 생성해 Snowpipe로 전달되도록 설정

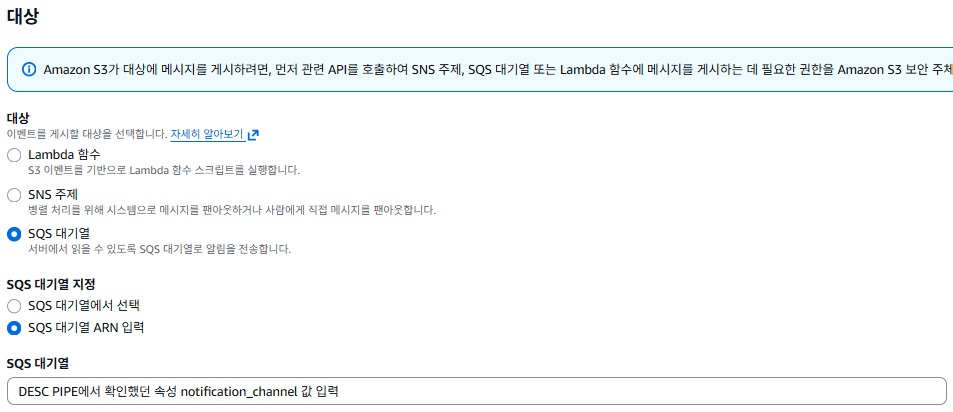




• Bucket 속성 -> 이벤트 알림 생성







• 위의 항목들을 제외한 그 외의 설정은 default 값으로 구성

• 파일을 지정된 버킷의 해당 폴더에 업로드한 후, Snowpipe가 해당 파일을 감지하여

지정 테이블로 COPY INTO 작업을 수행하는지 확인

### 6.1.2. **Azure Blob**

#### 6.1.2.1. **STORAGE INTEGRATION 생성**

CREATE OR REPLACE STORAGE INTEGRATION STORAGE NAME

TYPE = EXTERNAL\_STAGE

STORAGE\_PROVIDER = 'AZURE'

ENABLED = TRUE

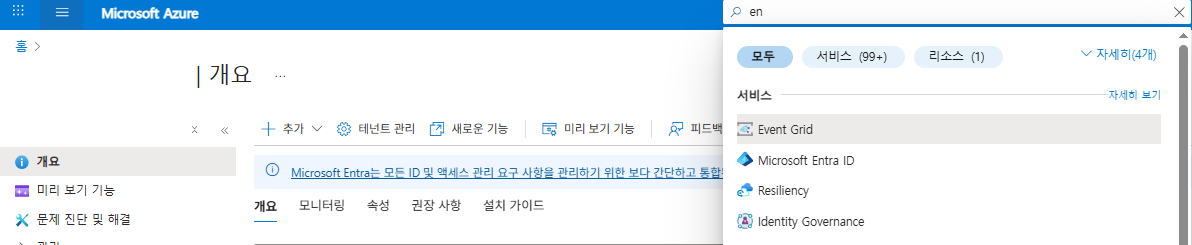
AZURE\_TENANT\_ID = ''

STORAGE\_ALLOWED\_LOCATIONS = (

'azure://blob path/container path/'

);

• AZURE\_TENANT\_ID 값은 Azure Portal에서 ‘Entra ID’를 검색해 테넌트 ID를 확인한 뒤 해당 값을 입력





#### 6.1.2.2. **STAGE 생성**

CREATE OR REPLACE STAGE STAGE\_NAME

URL='azure://blob path/container path/'

STORAGE\_INTEGRATION = STORAGE INTEGRATION NAME;

**\* 기존에 브론즈 테이블에서 실습했던 STAGE처럼 생성하면 안 됨(SAS 토큰X)**

#### 6.1.2.3. **NOTIFICATION INTEGRATION 생성**

CREATE OR REPLACE NOTIFICATION INTEGRATION AZURE\_NOTIFICATION

ENABLED = TRUE

TYPE = QUEUE

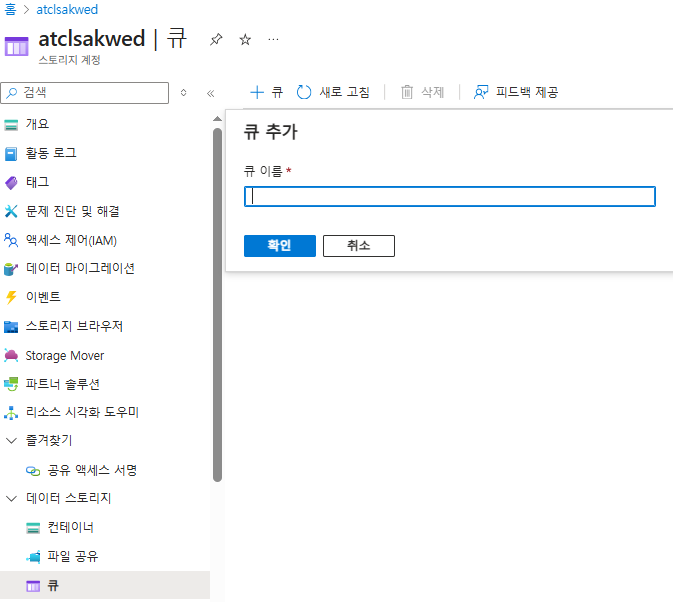
NOTIFICATION\_PROVIDER = AZURE\_STORAGE\_QUEUE

AZURE\_STORAGE\_QUEUE\_PRIMARY\_URI = 'QUEUE 생성 후 PAHT'

AZURE\_TENANT\_ID = '';

• Azure Portal에서 **스토리지 계정 → 데이터 스토리지 → 큐** 순으로 이동해

큐를 생성한 후, 생성된 큐의 URL을 복사해 **AZURE\_STORAGE\_QUEUE\_PRIMARY\_URI** 파라미터에 입력





#### 6.1.2.4. **이벤트 구독**

Azure Portal에서 **Event Grid 시스템 → 토픽 생성**을 완료한 뒤,

**스토리지 계정 → 이벤트 구독(Event Subscription)**을 설정해야 함

• 이 과정을 통해 Azure 스토리지에 파일이 업로드될 때 이벤트 알림이

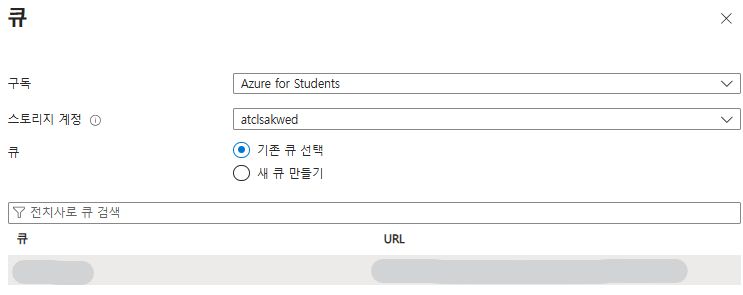
Snowflake로 전달되며, **Snowpipe가 이를 감지해 COPY INTO 자동 적재를 수행**



• 위의 항목들을 제외한 그 외의 설정은 default 값으로 구성



• **엔드포인트** 구성은 기존 큐 선택



• 위의 항목들을 제외한 그 외의 설정은 default 값으로 구성

#### 6.1.2.5. **액세스 제어(IAM)**

Snowpipe를 Azure Storage와 연동하려면 **Notification Integration** 및 **Storage Integration**을 구성해야 하며, 이 과정에서 **Azure 권한 허용(Consent)**과

**서비스 프린시펄(Role Assignment)1)** 설정이 필수적으로 요구

-- INTEGRATION 구성 정보 조회

DESC INTEGRATION INTEGRATION NAME;

-- NOTIFICATION INTEGRATION 구성 정보 조회

DESC NOTIFICATION INTEGRATION NOTIFICATION INTEGRATION NAME;

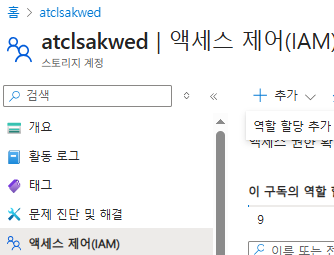


1) **서비스 프린시펄(Role Assignment)**: Azure에서 애플리케이션이나 서비스가 리소스에 접근할 수 있도록 만들어진 ‘서비스 전용 계정’

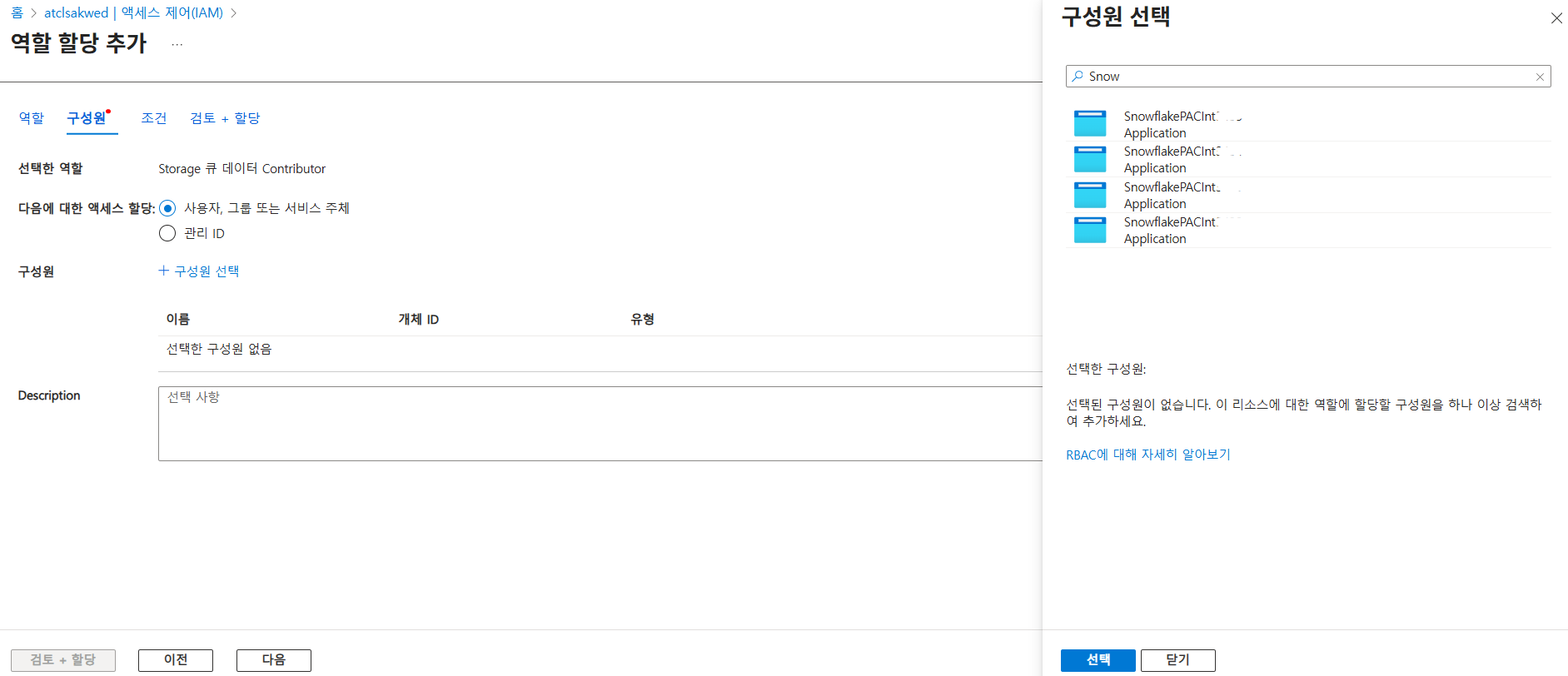
• **Azure Consent URL** 실행 - Snowflake가 Azure 접근을 허용받는 단계

• **AZURE\_MULTI\_TENANT\_APP\_NAME –** ex) SnowflakePACInt1234

Snowpipe가 Azure 리소스를 대신 접근할 때 사용하는 서비스 계정



• **Storage 큐 데이터 Contributor, Storage Blob 데이터 Contributor 역할 할당**



**\* 역할 할당 시 주의사항:**

**Integration**과 **Notification Integration**에서 생성된 **AZURE\_MULTI\_TENANT\_APP\_NAME** 전부 역할 부여

#### 6.1.2.6. **Snowpipe 생성**

-- HYBRID\_MANUFACTURING\_CATEGORICAL

CREATE OR REPLACE PIPE PIPE NAME

AUTO\_INGEST = TRUE

INTEGRATION = 'NOTIFICATION NAME'

AS

COPY INTO BRONZE\_DB.RAW\_CSV.HYBRID\_MANUFACTURING\_CATEGORICAL

FROM @STAGE NAME

PATTERN='(?i)^hybridmanufacturing/.\*\.csv$'

FILE\_FORMAT = (

TYPE = CSV,

FIELD\_OPTIONALLY\_ENCLOSED\_BY = '"',

SKIP\_HEADER = 1

);

• 파일을 지정된 컨테이너의 해당 디렉터리에 업로드한 후, Snowpipe가 해당 파일을

감지하여 지정 테이블로 COPY INTO 작업을 수행하는지 확인

## 6.2. **Stream+Task(CDC)**

• **Silver 단계 진행**

• Snowflake에서 **변경 데이터(CDC, Change Data Capture)**를 자동으로 추적하고

반영하는 방식

• **Stream**: 테이블에서 새로 추가·수정·삭제된 행만 기록해주는 **변경 추적 객체**

• **Task**: 일정 주기로 SQL을 자동 실행해 **Stream에 쌓인 변경분**을 **MERGE·INSERT·UPDATE** 등으로 대상 테이블에 반영하는 **스케줄러**

### 6.2.1. **Stream 생성**

CREATE OR REPLACE STREAM STREAM\_NAME

ON TABLE Bronze Table;

### 6.2.2. **Task 생성**

-- MANUFACTURING\_DEFECT\_CLEAN

CREATE OR REPLACE TASK TASK NAME

WAREHOUSE = COMPUTE\_WH

SCHEDULE = '1 MINUTE' -- 1분마다 실행

AS

MERGE INTO SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN T

USING (

SELECT \*

FROM STREAM NAME

WHERE PRODUCTION\_ID IS NOT NULL

QUALIFY ROW\_NUMBER() OVER (

PARTITION BY PRODUCTION\_ID

ORDER BY METADATA$ROW\_ID DESC

) = 1

) S

ON T.PRODUCTION\_ID = S.PRODUCTION\_ID

WHEN MATCHED AND S.METADATA$ACTION = 'UPDATE'

THEN UPDATE SET

T.OUTPUT\_LEVEL =

CASE

WHEN S.PRODUCTION\_VOLUME < 100 THEN 'LOW'

WHEN S.PRODUCTION\_VOLUME < 300 THEN 'MID\_LOW'

WHEN S.PRODUCTION\_VOLUME < 600 THEN 'MID'

WHEN S.PRODUCTION\_VOLUME < 900 THEN 'MID\_HIGH'

WHEN S.PRODUCTION\_VOLUME >= 900 THEN 'HIGH'

ELSE NULL

END,

T.PRODUCTION\_VOLUME = S.PRODUCTION\_VOLUME,

T.PRODUCTION\_COST\_USD = S.PRODUCTION\_COST,

T.SUPPLIER\_QUALITY\_PERCENT = S.SUPPLIER\_QUALITY,

T.DELIVERY\_DELAY\_DAYS = S.DELIVERY\_DELAY,

T.DEFECT\_RATE\_PERCENT = S.DEFECT\_RATE,

T.QUALITY\_SCORE\_PERCENT = S.QUALITY\_SCORE,

T.MAINTENANCE\_HOURS = S.MAINTENANCE\_HOURS,

T.DOWNTIME\_PERCENT = S.DOWNTIME\_PERCENTAGE,

T.INVENTORY\_TURNOVER\_RATIO = S.INVENTORY\_TURNOVER,

T.STOCKOUT\_RATE\_PERCENT = S.STOCKOUT\_RATE,

T.WORKER\_PRODUCTIVITY\_PERCENT = S.STOCKOUT\_RATE,

T.SAFETY\_INCIDENT\_COUNT = S.SAFETY\_INCIDENTS,

T.ENERGY\_CONSUMPTION\_KWH = S.ENERGY\_CONSUMPTION,

T.ENERGY\_EFFICIENCY\_RATIO = S.ENERGY\_EFFICIENCY,

T.ADDITIVE\_PROCESS\_HOURS = S.ADDITIVE\_PROCESS\_TIME,

T.ADDITIVE\_MATERIAL\_COST\_USD = S.ADDITIVE\_MATERIAL\_COST,

T.DEFECT\_STATUS =

CASE

WHEN S.DEFECT\_STATUS IN (FALSE, 0) THEN '낮음'

WHEN S.DEFECT\_STATUS IN (TRUE, 1) THEN '높음'

ELSE 'Undefined'

END

WHEN NOT MATCHED AND S.METADATA$ACTION = 'INSERT'

THEN INSERT (

PRODUCTION\_ID,

OUTPUT\_LEVEL,

PRODUCTION\_VOLUME,

PRODUCTION\_COST\_USD,

SUPPLIER\_QUALITY\_PERCENT,

DELIVERY\_DELAY\_DAYS,

DEFECT\_RATE\_PERCENT,

QUALITY\_SCORE\_PERCENT,

MAINTENANCE\_HOURS,

DOWNTIME\_PERCENT,

INVENTORY\_TURNOVER\_RATIO,

STOCKOUT\_RATE\_PERCENT,

WORKER\_PRODUCTIVITY\_PERCENT,

SAFETY\_INCIDENT\_COUNT,

ENERGY\_CONSUMPTION\_KWH,

ENERGY\_EFFICIENCY\_RATIO,

ADDITIVE\_PROCESS\_HOURS,

ADDITIVE\_MATERIAL\_COST\_USD,

DEFECT\_STATUS

)

VALUES (

S.PRODUCTION\_ID,

CASE

WHEN S.PRODUCTION\_VOLUME < 100 THEN 'LOW'

WHEN S.PRODUCTION\_VOLUME < 300 THEN 'MID\_LOW'

WHEN S.PRODUCTION\_VOLUME < 600 THEN 'MID'

WHEN S.PRODUCTION\_VOLUME < 900 THEN 'MID\_HIGH'

WHEN S.PRODUCTION\_VOLUME >= 900 THEN 'HIGH'

ELSE NULL

END,

S.PRODUCTION\_VOLUME,

S.PRODUCTION\_COST,

S.SUPPLIER\_QUALITY,

S.DELIVERY\_DELAY,

S.DEFECT\_RATE,

S.QUALITY\_SCORE,

S.MAINTENANCE\_HOURS,

S.DOWNTIME\_PERCENTAGE,

S.INVENTORY\_TURNOVER,

S.STOCKOUT\_RATE,

S.WORKER\_PRODUCTIVITY,

S.SAFETY\_INCIDENTS,

S.ENERGY\_CONSUMPTION,

S.ENERGY\_EFFICIENCY,

S.ADDITIVE\_PROCESS\_TIME,

S.ADDITIVE\_MATERIAL\_COST,

CASE

WHEN S.DEFECT\_STATUS IN (FALSE, 0) THEN '낮음'

WHEN S.DEFECT\_STATUS IN (TRUE, 1) THEN '높음'

ELSE 'Undefined'

END

);

-- HYBRID\_MANUFACTURING\_CLEAN

CREATE OR REPLACE TASK TASK NAME

WAREHOUSE = COMPUTE\_WH

SCHEDULE = '1 MINUTE' -- 1분마다 실행

AS

MERGE INTO SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN T

USING (

SELECT \*

FROM STREAM NAME

WHERE Job\_ID IS NOT NULL

QUALIFY ROW\_NUMBER() OVER (

PARTITION BY Job\_ID

ORDER BY METADATA$ROW\_ID DESC

) = 1

) S

ON T.Job\_ID = S.Job\_ID

WHEN MATCHED AND S.METADATA$ACTION = 'UPDATE'

THEN UPDATE SET

T.MACHINE\_ID = S.MACHINE\_ID,

T.OPERATION\_TYPE = S.OPERATION\_TYPE,

T.MATERIAL\_USED = S.MATERIAL\_USED,

T.PROCESSING\_TIME = S.PROCESSING\_TIME,

T.ENERGY\_CONSUMPTION = S.ENERGY\_CONSUMPTION,

T.MACHINE\_AVAILABILITY = S.MACHINE\_AVAILABILITY,

T.SCHEDULED\_START\_TS = S.SCHEDULED\_START,

T.SCHEDULED\_END\_TS = S.SCHEDULED\_END,

T.ACTUAL\_START\_TS = S.ACTUAL\_START,

T.ACTUAL\_END\_TS = S.ACTUAL\_END,

T.JOB\_STATUS = S.JOB\_STATUS,

T.OPTIMIZATION\_CATEGORY = S.OPTIMIZATION\_CATEGORY

WHEN NOT MATCHED AND S.METADATA$ACTION = 'INSERT'

THEN INSERT (

JOB\_ID,

MACHINE\_ID,

OPERATION\_TYPE,

MATERIAL\_USED,

PROCESSING\_TIME,

ENERGY\_CONSUMPTION,

MACHINE\_AVAILABILITY,

SCHEDULED\_START\_TS,

SCHEDULED\_END\_TS,

ACTUAL\_START\_TS,

ACTUAL\_END\_TS,

JOB\_STATUS,

OPTIMIZATION\_CATEGORY

)

VALUES (

S.JOB\_ID,

S.MACHINE\_ID,

S.OPERATION\_TYPE,

S.MATERIAL\_USED,

S.PROCESSING\_TIME,

S.ENERGY\_CONSUMPTION,

S.MACHINE\_AVAILABILITY,

S.SCHEDULED\_START,

S.SCHEDULED\_END,

S.ACTUAL\_START,

S.ACTUAL\_END,

S.JOB\_STATUS,

S.OPTIMIZATION\_CATEGORY

);

• Task는 기본적으로 **비활성화 상태**로 생성되므로, 실제로 실행되도록 하려면

**명시적으로 활성화(RESUME)**

-- Task 활성화

ALTER TASK HYBRID\_MANUFACTURING\_TASK RESUME;

-- Task 비활성화

ALTER TASK HYBRID\_MANUFACTURING\_TASK SUSPEND;

## 6.3. **Dynamic Table**

• **Gold 단계 진행**

• Snowflake가 설정된 주기에 따라 **SQL을 자동 실행**하여,

해당 결과를 항상 **최신 상태의 테이블로 유지**해주는 기능

-- MANUFACTURING\_DEFECT

CREATE OR REPLACE DYNAMIC TABLE GOLD\_DB.MANUFACTURING\_DEFECT\_KPI.FCT\_OPERATION\_SUMMARY\_DT

TARGET\_LAG = '1 MINUTE'

WAREHOUSE = COMPUTE\_WH

AS

SELECT

OUTPUT\_LEVEL,

-- 생산량 등급별 평균 KPI

ROUND(AVG(PRODUCTION\_COST\_USD), 2) AS PRODUCTION\_COST\_USD\_AVG,

ROUND(AVG(MAINTENANCE\_HOURS), 2) AS MAINTENANCE\_HOURS\_AVG,

ROUND(AVG(ENERGY\_CONSUMPTION\_KWH), 2) AS ENERGY\_CONSUMPTION\_KWH\_AVG,

ROUND(AVG(ADDITIVE\_PROCESS\_HOURS), 2) AS ADDITIVE\_PROCESS\_HOURS\_AVG,

ROUND(AVG(INVENTORY\_TURNOVER\_RATIO), 2) AS INVENTORY\_TURNOVER\_RATIO\_AVG,

ROUND(AVG(STOCKOUT\_RATE\_PERCENT), 2) AS STOCKOUT\_RATE\_PERCENT\_AVG,

ROUND(AVG(ADDITIVE\_MATERIAL\_COST\_USD), 2) AS ADDITIVE\_MATERIAL\_COST\_USD\_AVG,

-- 불량 상태 비율 (범주형 -> 비율로 변환)

ROUND(

100 \* SUM(CASE WHEN DEFECT\_STATUS = '높음' THEN 1 ELSE 0 END)

/ NULLIF(COUNT(\*), 0),

2

) AS DEFECT\_HIGH\_PERCENT,

ROUND(

100 \* SUM(CASE WHEN DEFECT\_STATUS = '낮음' THEN 1 ELSE 0 END)

/ NULLIF(COUNT(\*), 0),

2

) AS DEFECT\_LOW\_PERCENT

FROM SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN

GROUP BY OUTPUT\_LEVEL

ORDER BY

CASE

WHEN OUTPUT\_LEVEL = 'LOW' THEN 1

WHEN OUTPUT\_LEVEL = 'MID\_LOW' THEN 2

WHEN OUTPUT\_LEVEL = 'MID\_HIGH' THEN 3

WHEN OUTPUT\_LEVEL = 'HIGH' THEN 4

ELSE NULL

END;

-- HYBRID\_MANUFACTURING(FCT\_MACHINE\_SUMMARY)

CREATE OR REPLACE DYNAMIC TABLE GOLD\_DB.HYBRID\_MANUFACTURING\_KPI.FCT\_MACHINE\_SUMMARY\_DT

TARGET\_LAG = '1 MINUTE'

WAREHOUSE = COMPUTE\_WH

AS

WITH operation\_ratio AS (

-- 각 Machine\_ID × Operation\_Type별 비율(%) 계산

SELECT

Machine\_ID,

Operation\_Type,

COUNT(\*) AS operation\_count,

ROUND(

COUNT(\*) \* 100.0 / NULLIF(SUM(COUNT(\*)) OVER (PARTITION BY Machine\_ID), 0),

2

) AS operation\_ratio\_percent

FROM SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN

GROUP BY Machine\_ID, Operation\_Type

)

-- 기계별 주요 KPI + 공정 비율 결합

SELECT

h.Machine\_ID,

o.Operation\_Type,

o.operation\_ratio\_percent,

ROUND(AVG(h.Machine\_Availability), 2) AS Machine\_Availability\_avg,

ROUND(AVG(h.Energy\_Consumption), 2) AS Energy\_Consumption\_avg,

ROUND(AVG(h.Material\_Used), 2) AS Material\_Used\_avg,

FROM SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN h

LEFT JOIN operation\_ratio o

ON h.Machine\_ID = o.Machine\_ID

GROUP BY

h.Machine\_ID,

o.Operation\_Type,

o.operation\_ratio\_percent

ORDER BY

h.Machine\_ID ASC,

o.operation\_ratio\_percent DESC;

-- HYBRID\_MANUFACTURING(FCT\_JOB\_STATUS\_SUMMARY)

CREATE OR REPLACE DYNAMIC TABLE GOLD\_DB.HYBRID\_MANUFACTURING\_KPI.FCT\_JOB\_STATUS\_SUMMARY\_DT

TARGET\_LAG = '1 MINUTE'

WAREHOUSE = COMPUTE\_WH

AS

WITH

-- Optimization\_Category 비율 계산

opt\_ratio AS (

SELECT

Job\_Status,

Optimization\_Category,

COUNT(\*) AS cnt,

ROUND(

COUNT(\*) \* 100.0 / NULLIF(SUM(COUNT(\*)) OVER (PARTITION BY Job\_Status), 0),

2

) AS category\_ratio\_percent

FROM SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN

GROUP BY Job\_Status, Optimization\_Category

),

-- Job\_Status별 전체 건수 계산

status\_total AS (

SELECT

Job\_Status,

COUNT(\*) AS total\_count

FROM SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN

GROUP BY Job\_Status

)

-- 최종 테이블 생성

SELECT

h.Job\_Status,

s.total\_count,

o.Optimization\_Category,

o.category\_ratio\_percent,

ROUND(AVG(DATEDIFF(minute, h.Scheduled\_End\_TS, h.Actual\_End\_TS)), 2) AS avg\_delay\_min,

ROUND(AVG(h.Processing\_Time), 2) AS processing\_time\_avg,

ROUND(AVG(h.Energy\_Consumption), 2) AS energy\_consumption\_avg,

ROUND(AVG(h.Machine\_Availability), 2) AS machine\_availability\_avg

FROM SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN h

LEFT JOIN opt\_ratio o

ON h.Job\_Status = o.Job\_Status

AND h.Optimization\_Category = o.Optimization\_Category

LEFT JOIN status\_total s

ON h.Job\_Status = s.Job\_Status

GROUP BY

h.Job\_Status,

s.total\_count,

o.Optimization\_Category,

o.category\_ratio\_percent

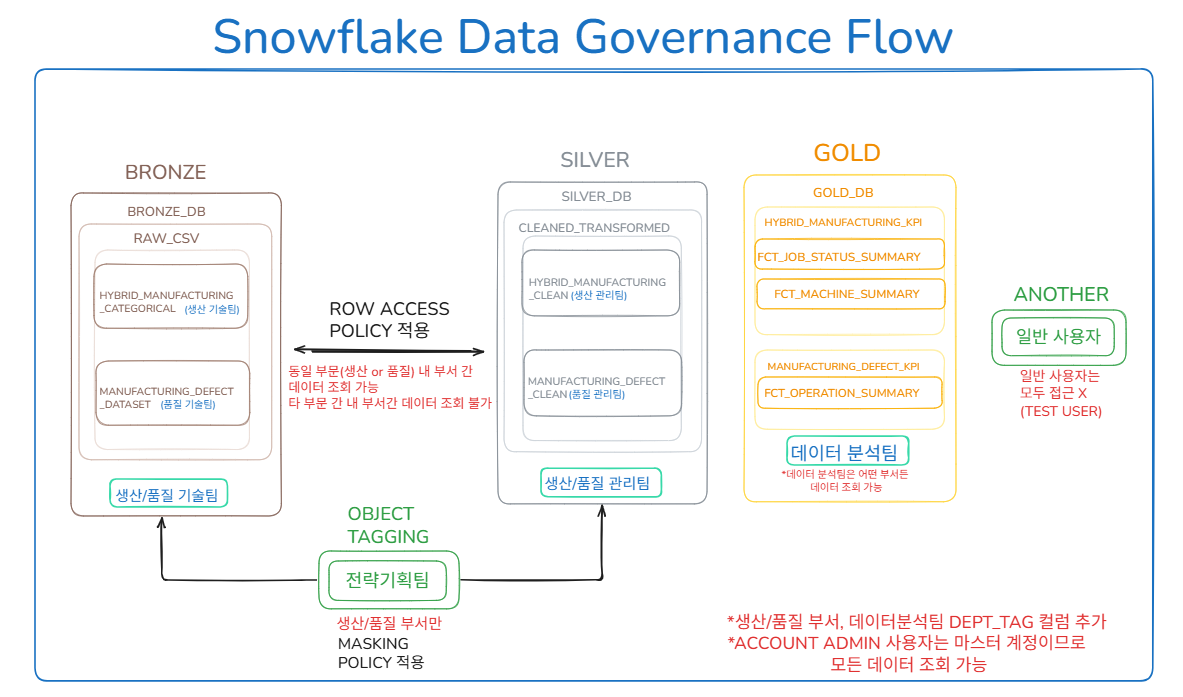
ORDER BY

h.Job\_Status,

o.category\_ratio\_percent DESC;

• 기존에 만들어둔 집계·요약 테이블은 **백업 및 참고용**으로 유지하고,  
Dynamic Table은 최신 데이터가 반영된 집계·요약 결과를 실시간에 가깝게 조회하는 방식으로 활용

# 7. **5주차: Data Governace**



**거버넌스 & 메타데이터 관리**

• Snowflake의 **데이터 거버넌스 기능(Object Tagging, Masking/Row Access Policy)**을

실습을 통해 적용

• 조직 내 데이터 접근·보안 **정책의 관리 체계 구축**

• **INFORMATION\_SCHEMA 및 ACCOUNT\_USAGE 뷰**를 활용해 **메타데이터 관리**와

**정책 적용 결과를 검증**

## 7.1. **부서별 ROLE1) 생성 및 권한 부여**

• **ROLE**: 사용자 접근 권한을 제어하는 역할 단위

\* ROLE 생성 및 권한 부여 작업은 ACCOUNTADMIN 권한으로 수행

### 7.1.1. **부서별 ROLE 생성**

-- 생산/품질 기술팀(BRONZE)

CREATE ROLE PRODUCTION\_ENGINEERING\_TEAM;

CREATE ROLE QUALITY\_ENGINEERING\_TEAM;

-- 생산/품질 관리팀(SILVER)

CREATE ROLE PRODUCTION\_MANAGEMENT\_TEAM;

CREATE ROLE QUALITY\_MANAGEMENT\_TEAM;

-- 데이터 분석팀(GOLD)

CREATE ROLE DATA\_ANALYTICS\_TEAM;

-- 전략 기획팀

CREATE ROLE STRATEGIC\_PLANNING\_TEAM;

-- 일반 사용자(TEST\_USER)

CREATE ROLE TEST\_USER;

### 7.1.2. **부서별 ROLE 권한 매핑**

-- 생산(HYBRID\_MANUFACTURING\_CLEAN)/품질(MANUFACTURING\_DEFECT\_CLEAN)기술팀 테이블 접근 권한 부여

GRANT SELECT ON TABLE BRONZE\_DB.RAW\_CSV.HYBRID\_MANUFACTURING\_CATEGORICAL TO ROLE PRODUCTION\_ENGINEERING\_TEAM;

GRANT SELECT ON TABLE BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET TO ROLE QUALITY\_ENGINEERING\_TEAM;

-- 생산(HYBRID\_MANUFACTURING\_CLEAN)/품질(MANUFACTURING\_DEFECT\_CLEAN)관리팀 테이블 접근 권한 부여

GRANT SELECT ON TABLE SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN TO ROLE PRODUCTION\_MANAGEMENT\_TEAM;

GRANT SELECT ON TABLE SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN TO ROLE QUALITY\_MANAGEMENT\_TEAM;

-- GOLD\_DB는 데이터 분석팀 테이블 접근 권한 부여

GRANT USAGE ON DATABASE GOLD\_DB TO ROLE DATA\_ANALYTICS\_TEAM;

### 7.1.3. **사용자에게 부서 ROLE 할당**

-- 각 부서 ROLE LIST

-- 생산/품질 기술팀(BRONZE): PRODUCTION\_ENGINEERING\_TEAM, QUALITY\_ENGINEERING\_TEAM

-- 생산/품질 관리팀(SILVER): PRODUCTION\_MANAGEMENT\_TEAM, QUALITY\_MANAGEMENT\_TEAM

-- 데이터 분석팀(GOLD): DATA\_ANALYTICS\_TEAM

-- 전략 기획팀: STRATEGIC\_PLANNING\_TEAM

-- 일반 사용자(TEST): TEST\_USER

SHOW USERS; -- 사용자 이름 확인하기

GRANT ROLE 부서 TO USER 사용자 이름;

• 일반적으로는 부서별로 사용자를 구분하여 각자 해당 ROLE만 부여하지만,

실습 목적이므로 **모든 부서 ROLE을 동일 사용자에게 할당**

**Role 기반 권한 관리 시 유의사항:**

• 한 사용자에게 여러 ROLE이 부여되었다고 해서 **모든 부서의 권한이 동시에**

**활성화되는 것은 아님**

• Snowflake에서는 사용자가 현재 활성화한 **ROLE에 따라 조회 가능한 데이터와**

**적용되는 정책이 달라짐**

## 7.2. **Object Tagging**

• **Object Tagging**: 데이터 오브젝트에 의미나 속성을 부여하는 메타데이터 관리 기능

• 이는 Object Tagging 기능의 구조와 적용 방법을 이해하기 위한 실습으로,

사용 목적이나 데이터 특성에 따라 **사용자 임의**로 태그를 정의·부여해도 무방

### 7.2.1. **Object Tagging 생성**

-- Object Tagging 생성

CREATE OR REPLACE TAG TAG\_SENSITIVITY COMMENT='마스킹 처리 대상';

### 7.2.2. **태그 값 부여**

-- 생산 기술팀(HYBRID\_MANUFACTURING\_CATEGORICAL) 마스킹 처리 대상 태그 부여

ALTER TABLE BRONZE\_DB.RAW\_CSV.HYBRID\_MANUFACTURING\_CATEGORICAL

MODIFY COLUMN MACHINE\_ID SET TAG TAG\_SENSITIVITY = 'MEDIUM';

ALTER TABLE BRONZE\_DB.RAW\_CSV.HYBRID\_MANUFACTURING\_CATEGORICAL

MODIFY COLUMN SCHEDULED\_START SET TAG TAG\_SENSITIVITY = 'MEDIUM';

ALTER TABLE BRONZE\_DB.RAW\_CSV.HYBRID\_MANUFACTURING\_CATEGORICAL

MODIFY COLUMN SCHEDULED\_END SET TAG TAG\_SENSITIVITY = 'MEDIUM';

-- 품질 기술팀(MANUFACTURING\_DEFECT\_DATASET) 마스킹 처리 대상 태그 부여

ALTER TABLE BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET

MODIFY COLUMN SUPPLIER\_QUALITY SET TAG TAG\_SENSITIVITY = 'HIGH';

ALTER TABLE BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET

MODIFY COLUMN DEFECT\_RATE SET TAG TAG\_SENSITIVITY = 'HIGH';

ALTER TABLE BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET

MODIFY COLUMN QUALITY\_SCORE SET TAG TAG\_SENSITIVITY = 'HIGH';

ALTER TABLE BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET

MODIFY COLUMN WORKER\_PRODUCTIVITY SET TAG TAG\_SENSITIVITY = 'HIGH';

ALTER TABLE BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET

MODIFY COLUMN SAFETY\_INCIDENTS SET TAG TAG\_SENSITIVITY = 'HIGH';

-- 생산 관리팀(HYBRID\_MANUFACTURING\_CLEAN) 마스킹 처리 대상 태그 부여

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN

MODIFY COLUMN MACHINE\_ID SET TAG TAG\_SENSITIVITY = 'MEDIUM';

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN

MODIFY COLUMN SCHEDULED\_START\_TS SET TAG TAG\_SENSITIVITY = 'MEDIUM';

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN

MODIFY COLUMN SCHEDULED\_END\_TS SET TAG TAG\_SENSITIVITY = 'MEDIUM';

-- 품질 관리팀() 마스킹 처리 대상 태그 부여

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN

MODIFY COLUMN OUTPUT\_LEVEL SET TAG TAG\_SENSITIVITY = 'HIGH';

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN

MODIFY COLUMN SUPPLIER\_QUALITY\_PERCENT SET TAG TAG\_SENSITIVITY = 'HIGH';

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN

MODIFY COLUMN DEFECT\_RATE\_PERCENT SET TAG TAG\_SENSITIVITY = 'HIGH';

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN

MODIFY COLUMN QUALITY\_SCORE\_PERCENT SET TAG TAG\_SENSITIVITY = 'HIGH';

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN

MODIFY COLUMN WORKER\_PRODUCTIVITY\_PERCENT SET TAG TAG\_SENSITIVITY = 'HIGH';

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN

MODIFY COLUMN SAFETY\_INCIDENT\_COUNT SET TAG TAG\_SENSITIVITY = 'HIGH';

• 현재 단계에서 생성하고 부여한 태그는 이후 Masking Policy 정의 및 적용 과정에서 활용

• 생성된 태그를 Masking Policy 정의·적용 시 연동하여 **자동화 기반**으로 활용

## 7.3. **Masking Policy**

• **Masking Policy**: ROLE이나 조건에 따라 컬럼 값을 마스킹하는 보안 정책

### 7.3.1. **Masking Policy 정의**

\*마스킹 처리하고 싶은 컬럼들을 **타입별로 Masking Policy 따로 정의**해야 함

-- 문자/숫자형(NUMBER/FLOAT)/날짜형 타입의 컬럼 별도 마스킹 처리

-- 문자(STRING)타입의 컬럼 마스킹 처리

CREATE OR REPLACE MASKING POLICY MASK\_POLICY\_SENSITIVITY\_STRING

AS (val STRING)

RETURNS STRING ->

CASE

WHEN CURRENT\_ROLE() IN ('PRODUCTION\_ENGINEERING\_TEAM', 'PRODUCTION\_MANAGEMENT\_TEAM',

'QUALITY\_ENGINEERING\_TEAM', 'QUALITY\_MANAGEMENT\_TEAM','DATA\_ANALYTICS\_TEAM')

THEN val

WHEN CURRENT\_ROLE() = 'STRATEGIC\_PLANNING\_TEAM'

THEN '\*\*No permission\*\*'

ELSE NULL

END;

-- 숫자(NUMBER)타입의 컬럼 마스킹 처리

CREATE OR REPLACE MASKING POLICY MASK\_POLICY\_SENSITIVITY\_NUMBER

AS (val NUMBER)

RETURNS NUMBER ->

CASE

-- 원본 값을 숨기고 변환된 값 제공

WHEN CURRENT\_ROLE() IN ('PRODUCTION\_ENGINEERING\_TEAM', 'PRODUCTION\_MANAGEMENT\_TEAM',

'QUALITY\_ENGINEERING\_TEAM', 'QUALITY\_MANAGEMENT\_TEAM','DATA\_ANALYTICS\_TEAM')

THEN val

WHEN CURRENT\_ROLE() = 'STRATEGIC\_PLANNING\_TEAM'

THEN ROUND(val \* 0.1, 2)

ELSE NULL

END;

-- 숫자(FLOAT)타입의 컬럼 마스킹 처리

CREATE OR REPLACE MASKING POLICY MASK\_POLICY\_SENSITIVITY\_FLOAT

AS (val FLOAT)

RETURNS FLOAT ->

CASE

-- 원본 값을 숨기고 변환된 값 제공

WHEN CURRENT\_ROLE() IN ('PRODUCTION\_ENGINEERING\_TEAM', 'PRODUCTION\_MANAGEMENT\_TEAM',

'QUALITY\_ENGINEERING\_TEAM', 'QUALITY\_MANAGEMENT\_TEAM','DATA\_ANALYTICS\_TEAM')

THEN val

WHEN CURRENT\_ROLE() = 'STRATEGIC\_PLANNING\_TEAM'

THEN ROUND(val \* 0.1, 2)

ELSE NULL

END;

-- 날짜(TIMESTAMP\_NTZ)타입의 컬럼 마스킹 처리

CREATE OR REPLACE MASKING POLICY MASK\_POLICY\_SENSITIVITY\_DATE

AS (val TIMESTAMP\_NTZ)

RETURNS TIMESTAMP\_NTZ ->

CASE

WHEN CURRENT\_ROLE() IN ('PRODUCTION\_ENGINEERING\_TEAM', 'PRODUCTION\_MANAGEMENT\_TEAM',

'QUALITY\_ENGINEERING\_TEAM', 'QUALITY\_MANAGEMENT\_TEAM','DATA\_ANALYTICS\_TEAM')

THEN val

WHEN CURRENT\_ROLE() = 'STRATEGIC\_PLANNING\_TEAM'

-- 날짜만 보여주고 시간 부분은 숨김

THEN DATE\_TRUNC('DAY', val)

ELSE NULL

END;

### 7.3.2. **TAG 기반 자동 Masking 확장**

-- 정의한 Masking Policy를 이전에 생성한 TAG에 설정

ALTER TAG TAG\_SENSITIVITY SET MASKING POLICY MASK\_POLICY\_SENSITIVITY\_STRING;

ALTER TAG TAG\_SENSITIVITY SET MASKING POLICY MASK\_POLICY\_SENSITIVITY\_NUMBER;

ALTER TAG TAG\_SENSITIVITY SET MASKING POLICY MASK\_POLICY\_SENSITIVITY\_FLOAT;

ALTER TAG TAG\_SENSITIVITY SET MASKING POLICY MASK\_POLICY\_SENSITIVITY\_DATE;

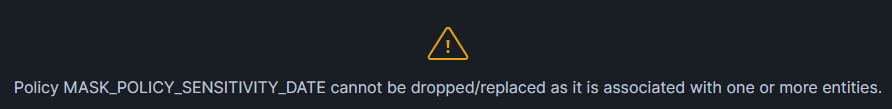
**\*Masking Policy를 잘못 정의했거나 수정이 필요한 경우:**

• 기존 TAG에 설정되어 있는 Masking Policy를 먼저 **해제**한 후 수정 작업을

진행해야 함

• 정책 해제 후에는 DROP MASKING POLICY 또는 CREATE OR REPLACE MASKING POLICY와 같은 **DDL 명령어를 사용하여 정책을 재정의**할 수 있음

**Masking Policy 연결로 인한 TAG 변경 제한 오류:**



### 7.3.3. **Masking Policy 적용 확인**

-- 각 부서 ROLE LIST

-- 생산/품질 기술팀(BRONZE): PRODUCTION\_ENGINEERING\_TEAM, QUALITY\_ENGINEERING\_TEAM

-- 생산/품질 관리팀(SILVER): PRODUCTION\_MANAGEMENT\_TEAM, QUALITY\_MANAGEMENT\_TEAM

-- 데이터 분석팀(GOLD): DATA\_ANALYTICS\_TEAM

-- 전략 기획팀: STRATEGIC\_PLANNING\_TEAM

-- 일반 사용자(TEST): TEST\_USER

-- 각 부서 TABLE LIST(마스킹 처리 대상 부서 한정)

-- 생산 기술팀(BRONZE): BRONZE\_DB.RAW\_CSV.HYBRID\_MANUFACTURING\_CATEGORICAL

-- 품질 기술팀(BRONZE): BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET

-- 생산 관리팀(SILVER): SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN

-- 품질 관리팀(SILVER): SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN

-- 부서 ROLE 전환 후 마스킹 정책 검증

USE ROLE 부서;

-- 부서별 ROLE 전환 후 데이터 표시 결과 확인

SELECT \* FROM DB.SCHEMA.TABLE;

전략 기획팀에서 품질 관리팀 부서 테이블 조회 결과:



## 7.4. **Row Access Policy**

**• Row Acces Policy**: Snowflake에서 사용자 또는 ROLE에 따라 조회 가능한 행(Row)을

제한하기 위한 보안 정책

### 7.4.1. **각 부서별 테이블에 DEPT\_TAG 컬럼 추가**

Row Access Policy 적용을 위한 **DEPT\_TAG 컬럼** 추가:

• 일반적으로 Row Access Policy는 하나의 테이블 내에 여러 부서의 데이터가 존재할 때, 부서별로 조회 가능한 행(Row)을 제한하기 위해 사용

• 본 실습에서는 각 부서별로 별도의 테이블이 존재하므로, Row Access Policy 적용을

실습하기 위해 부서 식별 컬럼(DEPT\_TAG)을 추가

-- Row Access Policy 정의를 위한 각 부서별 테이블에 DEPT\_TAG 컬럼 추가

-- 생산 관련 부서는 DEPT\_TAG의 컬럼값 PRODUCTION 지정

ALTER TABLE BRONZE\_DB.RAW\_CSV.HYBRID\_MANUFACTURING\_CATEGORICAL ADD COLUMN DEPT\_TAG STRING DEFAULT 'PRODUCTION';

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN ADD COLUMN DEPT\_TAG STRING DEFAULT 'PRODUCTION';

-- 품질 관련 부서는 DEPT\_TAG의 컬럼값 QUALITY 지정

ALTER TABLE BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET ADD COLUMN DEPT\_TAG STRING DEFAULT 'QUALITY';

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN ADD COLUMN DEPT\_TAG STRING DEFAULT 'QUALITY';

### 7.4.2. **Row Access Policy 정의**

-- Row Access Policy 정의

CREATE OR REPLACE ROW ACCESS POLICY POLICY\_TEAM\_ACCESS – 사용자 정의 ROW Access Policy 이름

AS (DEPT\_TAG STRING)

RETURNS BOOLEAN ->

CASE

WHEN CURRENT\_ROLE() IN ('PRODUCTION\_ENGINEERING\_TEAM', 'PRODUCTION\_MANAGEMENT\_TEAM')

AND DEPT\_TAG IN ('PRODUCTION') THEN TRUE

WHEN CURRENT\_ROLE() IN ('QUALITY\_ENGINEERING\_TEAM', 'QUALITY\_MANAGEMENT\_TEAM')

AND DEPT\_TAG IN ('QUALITY') THEN TRUE

WHEN CURRENT\_ROLE() = 'DATA\_ANALYTICS\_TEAM' THEN TRUE

WHEN CURRENT\_ROLE() = 'STRATEGIC\_PLANNING\_TEAM' THEN TRUE

ELSE FALSE

END;

### 7.4.3. **Row Access Policy 적용**

-- 생산 기술/관리팀 정책 적용

ALTER TABLE BRONZE\_DB.RAW\_CSV.HYBRID\_MANUFACTURING\_CATEGORICAL ADD ROW ACCESS POLICY POLICY\_TEAM\_ACCESS ON (DEPT\_TAG);

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN ADD ROW ACCESS POLICY POLICY\_TEAM\_ACCESS ON (DEPT\_TAG);

-- 품질 기술/관리팀 정책 적용

ALTER TABLE BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET ADD ROW ACCESS POLICY POLICY\_TEAM\_ACCESS ON (DEPT\_TAG);

ALTER TABLE SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN ADD ROW ACCESS POLICY POLICY\_TEAM\_ACCESS ON (DEPT\_TAG);

\* Row Access Policy를 잘못 정의했거나 수정이 필요한 경우:

-- 테이블에 적용된 Row Access Policy 해제

ALTER TABLE DB.SCHEMA.TABLE DROP ROW ACCESS POLICY 정책 이름;

-- Row Access Policy 제거

DROP ROW ACCESS POLICY IF EXISTS 정책 이름;

### 7.4.4. **Row Access Policy 적용 확인**

-- 각 부서 ROLE LIST

-- 생산/품질 기술팀(BRONZE): PRODUCTION\_ENGINEERING\_TEAM, QUALITY\_ENGINEERING\_TEAM

-- 생산/품질 관리팀(SILVER): PRODUCTION\_MANAGEMENT\_TEAM, QUALITY\_MANAGEMENT\_TEAM

-- 데이터 분석팀(GOLD): DATA\_ANALYTICS\_TEAM

-- 전략 기획팀: STRATEGIC\_PLANNING\_TEAM

-- 일반 사용자(TEST): TEST\_USER

-- 각 부서 TABLE LIST(Row Access Policy 대상 부서 한정)

-- 생산 기술팀(BRONZE): BRONZE\_DB.RAW\_CSV.HYBRID\_MANUFACTURING\_CATEGORICAL

-- 품질 기술팀(BRONZE): BRONZE\_DB.RAW\_CSV.MANUFACTURING\_DEFECT\_DATASET

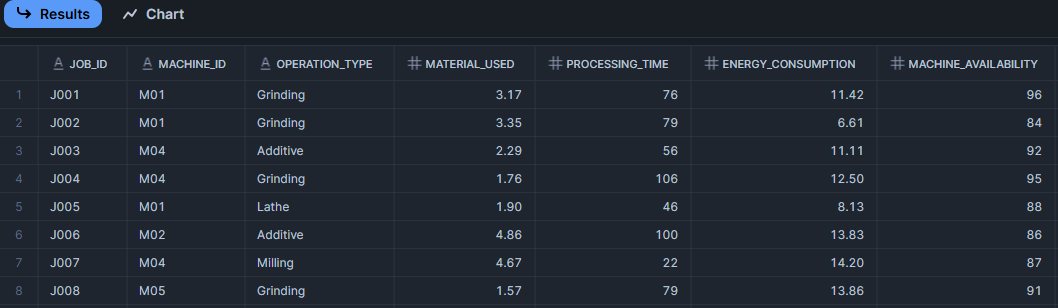
-- 생산 관리팀(SILVER): SILVER\_DB.CLEANED\_TRANSFORMED.HYBRID\_MANUFACTURING\_CLEAN

-- 품질 관리팀(SILVER): SILVER\_DB.CLEANED\_TRANSFORMED.MANUFACTURING\_DEFECT\_CLEAN

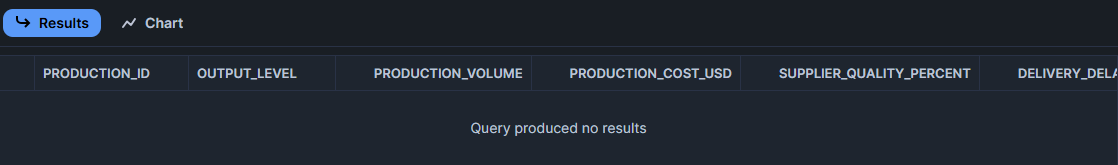
USE ROLE 부서;

SELECT \* FROM DB.SCHEMA.TABLE;

생산 기술팀에서 생산 관리팀 부서 테이블 조회 결과:



생산 기술팀에서 품질 관리팀 부서 테이블 조회 결과:



## 7.5. **메타데이터 기반 확인 및 정책 적용 결과 검증**

### 7.5.1. **INFORMATION\_SCHEMA**

#### 7.5.1.1. **모든 테이블의 기본 메타데이터 조회**

-- 모든 테이블의 기본 메타데이터를 조회

SELECT TABLE\_SCHEMA, TABLE\_NAME, ROW\_COUNT, CREATED

FROM 조회 대상 DB.INFORMATION\_SCHEMA.TABLES;

GOLD\_DB 조회 결과:



• **스키마명**, **테이블명**, **전체 행 수**, **생성 일자**를 확인

#### 7.5.1.2. **특정 테이블의 컬럼 구조 확인**

-- 특정 테이블의 컬럼 구조 확인

SELECT

COLUMN\_NAME,

DATA\_TYPE,

IS\_NULLABLE,

ORDINAL\_POSITION

FROM 대상 DB.INFORMATION\_SCHEMA.COLUMNS

WHERE

TABLE\_NAME = '대상 테이블'

ORDER BY

ORDINAL\_POSITION;

HYBRID\_MANUFACTURING\_CLEAN 조회 결과:



• **컬럼명**, **컬럼 타입**, **NULL 허용 여부**, **컬럼 순서**를 확인

### 7.5.2 **ACCOUNT\_USAGE**

#### 7.5.2.1. **쿼리 실행 이력 분석**

-- 쿼리 실행 이력 분석

SELECT

USER\_NAME,

QUERY\_TEXT,

START\_TIME,

END\_TIME,

TOTAL\_ELAPSED\_TIME/1000 AS exec\_time\_sec,

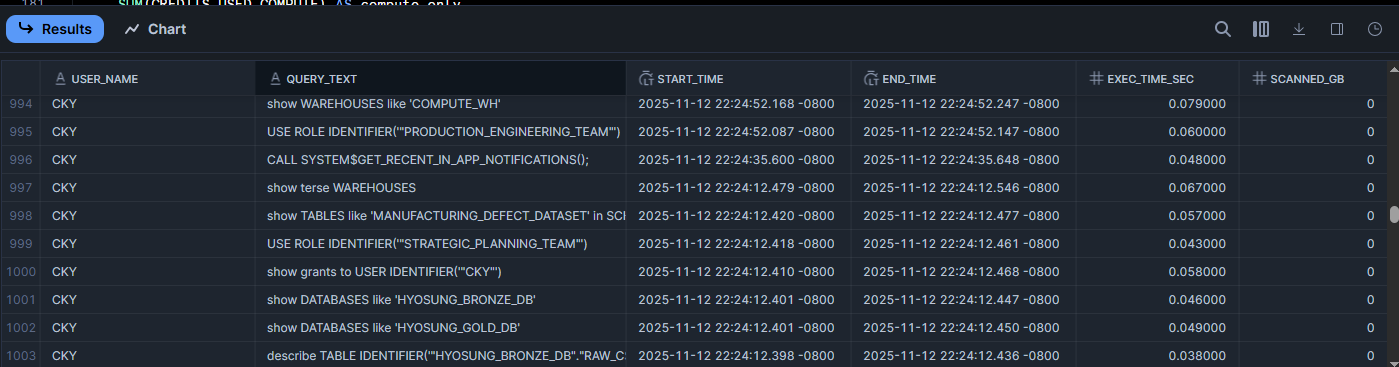
BYTES\_SCANNED/POWER(1024,3) AS scanned\_gb

FROM SNOWFLAKE.ACCOUNT\_USAGE.QUERY\_HISTORY

WHERE START\_TIME >= DATEADD('day', -1, CURRENT\_TIMESTAMP())

ORDER BY START\_TIME DESC;

쿼리 실행 이력 분석 결과:



• 최근 1일 동안 실행된 쿼리들의 사용자 정보, 실행 시각, 소요 시간,

스캔된 데이터량 등 확인

#### 7.5.2.2. **Warehouse별 크레딧 사용량**

-- Warehouse별 크레딧 사용량

SELECT

WAREHOUSE\_NAME,

SUM(CREDITS\_USED) AS total\_credits,

SUM(CREDITS\_USED\_COMPUTE) AS compute\_only,

SUM(CREDITS\_USED\_CLOUD\_SERVICES) AS cloud\_service

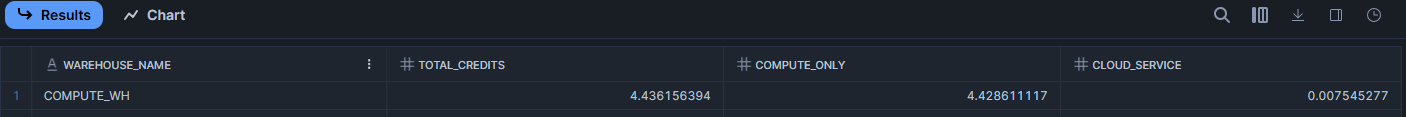
FROM SNOWFLAKE.ACCOUNT\_USAGE.WAREHOUSE\_METERING\_HISTORY

WHERE START\_TIME >= DATEADD('day', -7, CURRENT\_TIMESTAMP())

GROUP BY WAREHOUSE\_NAME

ORDER BY total\_credits DESC;

웨어하우스별 크레딧 사용량 결과:



• 최근 7일 동안 실행된 **Warehouse 총 크레딧 사용량**과 **컴퓨트 비용**,

**클라우드 서비스 비용** 확인

• Warehouse 운영에 소요된 리소스 및 비용 구조 파악

#### 7.5.2.3. **로그인 시간대 모니터링**

-- 로그인 시간대 모니터링

SELECT

USER\_NAME,

IS\_SUCCESS,

ERROR\_MESSAGE,

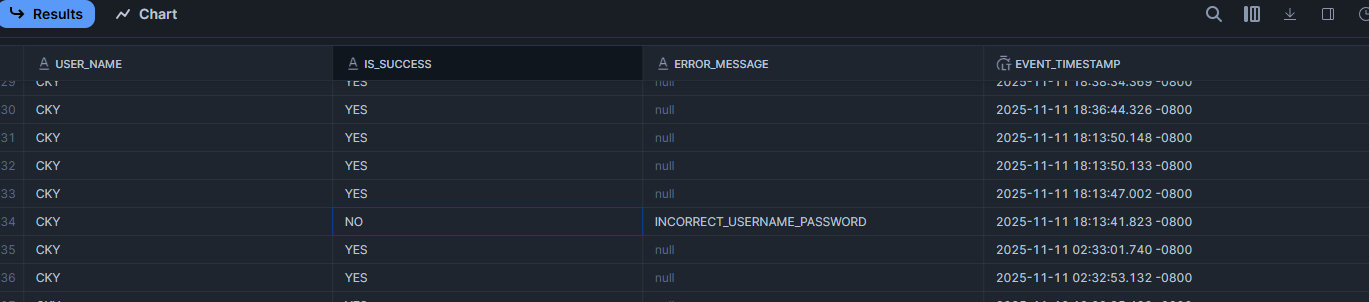
EVENT\_TIMESTAMP

FROM SNOWFLAKE.ACCOUNT\_USAGE.LOGIN\_HISTORY

WHERE EVENT\_TIMESTAMP >= DATEADD('day', -3, CURRENT\_TIMESTAMP())

ORDER BY EVENT\_TIMESTAMP DESC;

로그인 시간대 모니터링 결과:



• **로그인한 사용자**, **로그인 성공 여부**, 오류 발생 시 에러 메시지(**로그인 실패 사유**),

**로그인 시각** 확인

#### 7.5.2.4. **테이블 스토리지 사용량 모니터링**

-- 테이블 스토리지 사용량 모니터링

SELECT

TABLE\_NAME,

TABLE\_SCHEMA,

ACTIVE\_BYTES / POWER(1024,3) AS active\_gb,

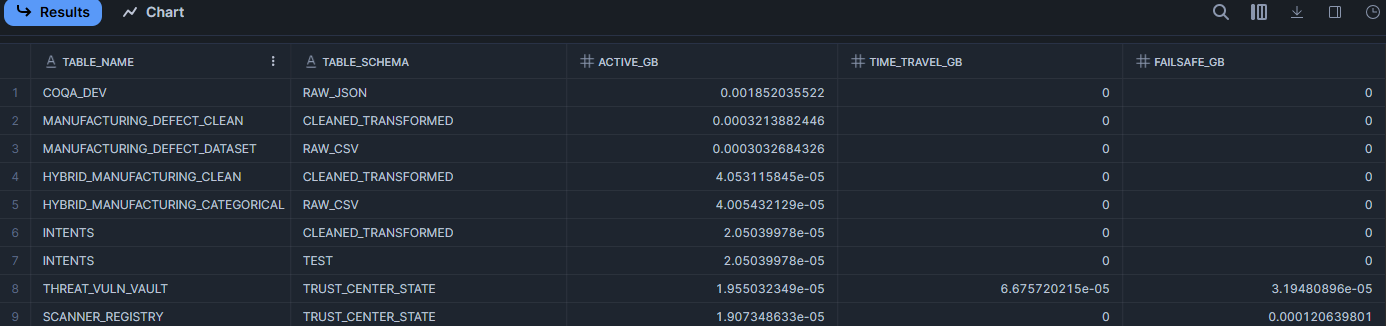
TIME\_TRAVEL\_BYTES / POWER(1024,3) AS time\_travel\_gb,

FAILSAFE\_BYTES / POWER(1024,3) AS failsafe\_gb

FROM SNOWFLAKE.ACCOUNT\_USAGE.TABLE\_STORAGE\_METRICS

ORDER BY active\_gb DESC;

테이블 스토리지 사용량 모니터링 결과:



• 각 **테이블 이름**, **테이블 스키마**, 실제 저장된 **데이터 용량**,

**Time Travel 보관 용량**, **Fail-safe 보관 용량** 확인

• 테이블 단위의 저장공간 사용량을 파악

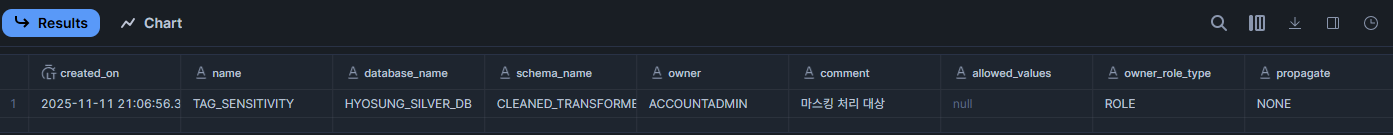
### 7.5.3. **정책 적용 결과 검증**

#### 7.5.3.1. **Object Tagging**

-- 모든 태그 조회

SHOW TAGS;

모든 태그 조회 결과:



• **CREATED\_ON**: 태그가 생성된 일시

• **NAME**: 태그 이름

• **DATABASE\_NAME**/**SCHEMA\_NAME**: 태그가 속한 데이터베이스 및 스키마

• **OWNER**: 태그를 소유한 역할(Role)

• **COMMENT**: 태그에 부여된 설명 또는 용도

• **ALLOWED\_VALUES**: 태그 값에 허용되는 값 목록(지정된 경우)

• **OWNER\_ROLE\_TYPE**: 태그를 관리하는 역할 유형

• **PROPAGATE**: 태그 상속(전파) 여부 설정

-- 특정 태그의 세부정보 보기

SELECT

TAG\_DATABASE,

TAG\_SCHEMA,

TAG\_NAME,

OBJECT\_DATABASE,

OBJECT\_SCHEMA,

OBJECT\_NAME,

COLUMN\_NAME,

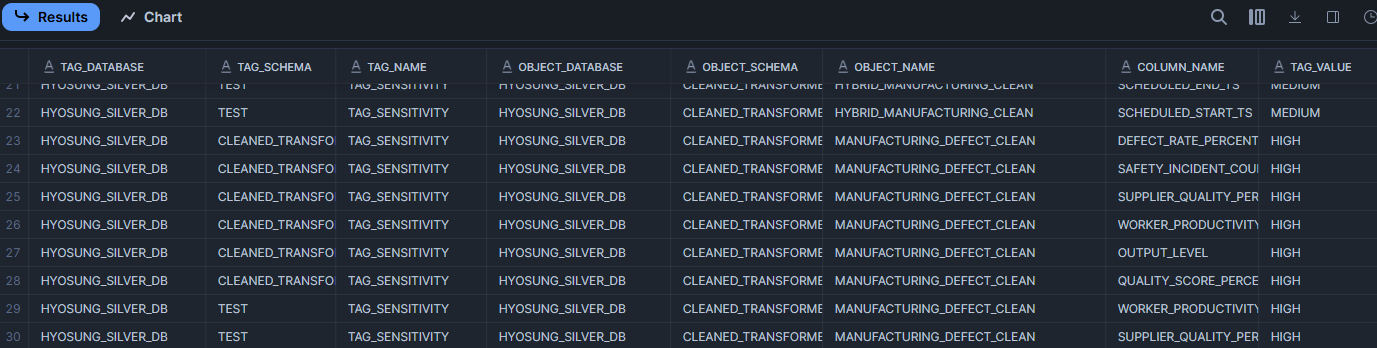
TAG\_VALUE

FROM SNOWFLAKE.ACCOUNT\_USAGE.TAG\_REFERENCES

WHERE TAG\_NAME = 'TAG\_SENSITIVITY'

ORDER BY OBJECT\_DATABASE, OBJECT\_SCHEMA, OBJECT\_NAME;

특정 태그의 세부정보 보기 결과:



• **TAG\_DATABASE**/**TAG\_SCHEMA**: 태그가 속한 데이터베이스와 스키마

• **TAG\_NAME**: 적용된 태그 이름

• **OBJECT\_DATABAS**/**OBJECT\_SCHEMA**/**OBJECT\_NAME**: 태그가 부여된 테이블의

위치 및 이름

• **COLUMN\_NAME**: 태그가 적용된 구체적인 컬럼

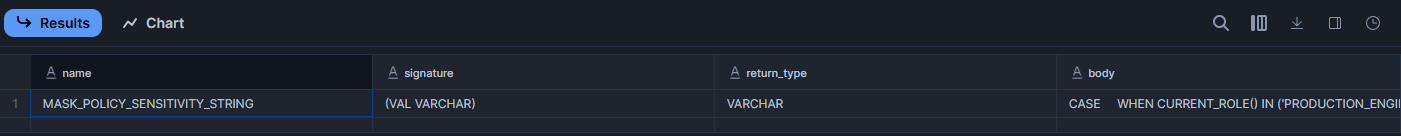
• **TAG\_VALUE**: 해당 컬럼에 설정된 태그 값

#### 7.5.3.2. **Masking Policy**

-- Masking Policy 상세 정의 조회

DESCRIBE MASKING POLICY 정책명;

Masking Policy 상세 정의 조회 결과:



• **정책 이름**, **반환 타입**, **조건** 확인

-- Masking Policy 적용 대상 조회

SELECT

POLICY\_NAME,

POLICY\_KIND,

REF\_DATABASE\_NAME,

REF\_SCHEMA\_NAME,

REF\_ENTITY\_NAME AS TABLE\_NAME,

REF\_COLUMN\_NAME

FROM SNOWFLAKE.ACCOUNT\_USAGE.POLICY\_REFERENCES

WHERE POLICY\_NAME = '정책 이름';

Masking Policy 적용 대상 조회 결과:



• **POLICY\_NAME**/**POLICY\_KIND**: 적용된 정책의 이름과 종류(마스킹 정책)

• **REF\_DATABASE\_NAME**/**REF\_SCHEMA\_NAME**: 정책이 적용된 객체가 속한

데이터베이스 및 스키마

• **TABLE\_NAME**: 마스킹 정책이 연결된 테이블

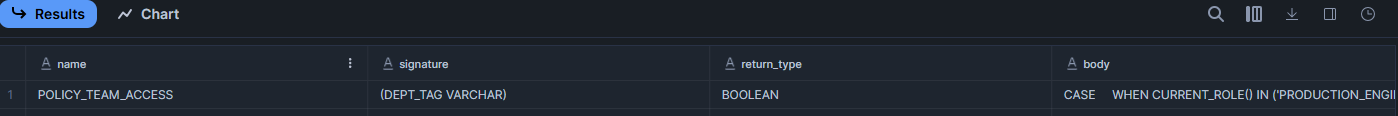
• **REF\_COLUMN\_NAME**: 정책이 적용된 구체적인 컬럼(민감 데이터 컬럼 등)

#### 7.5.3.3**. Row Access Policy**

-- Row Access Policy 상세 정의 조회

DESC ROW ACCESS POLICY 정책 이름;

Row Access Policy 상세 정의 조회 결과:



• **정책 이름**, **반환 타입**, **조건** 확인

-- Row Access Policy 적용된 오브젝트 조회

SELECT

POLICY\_NAME,

POLICY\_KIND,

REF\_DATABASE\_NAME,

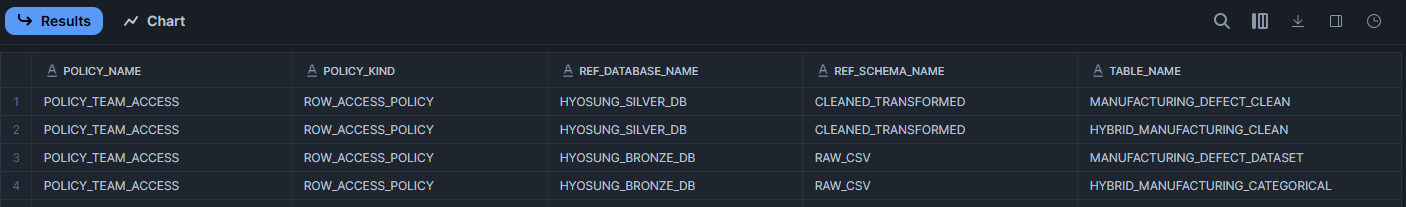
REF\_SCHEMA\_NAME,

REF\_ENTITY\_NAME AS TABLE\_NAME

FROM SNOWFLAKE.ACCOUNT\_USAGE.POLICY\_REFERENCES

WHERE POLICY\_NAME = '정책 이름';

Row Access Policy 적용된 오브젝트 조회 결과:



• **POLICY\_NAME**/**POLICY\_KIND**: 적용된 정책의 이름과 종류

• **REF\_DATABASE\_NAME**/**REF\_SCHEMA\_NAME**: 정책이 적용된 객체가 속한

데이터베이스 및 스키마

• **TABLE\_NAME**: 행 단위 접근 제어가 적용된 테이블