

CSCI E-103: Data Engineering for Analytics

7주차 학습 노트: 데이터 파이프라인 운영 (Operationalizing Data Pipelines)

Harvard Extension School (Fall 2025)

October 26, 2025

■ 강의명: CSCI E-103: 재현 가능한 머신러닝

■ 주차: Lecture 07

■ 교수명: Anindita Mahapatra

Eric Gieseke

■ 목적: Lecture 07의 핵심 개념 학습

Contents

1	핵심 용어 정리	2
2	1부: 데이터 파이프라인 운영의 개념	4
2.1	왜 "운영(Operationalizing)"이 필요한가?	4
2.2	요구사항 정의: 모든 설계의 시작	4
2.3	데이터 파이프라인의 6가지 핵심 원칙	5
2.3.1	1. 데이터 큐레이션 및 신뢰할 수 있는 "데이터 제품" 제공	5
2.3.2	2. 데이터 사일로 제거 및 이동 최소화	5
2.3.3	3. 셀프 서비스 경험으로 가치 창출 민주화	5
2.3.4	4. 전사적 데이터 거버넌스 전략 채택	6
2.3.5	5. 개방형 인터페이스 및 포맷 사용	6
2.3.6	6. 확장성 및 비용/성능 최적화	6
3	2부: 데이터 품질 및 검증	7

3.1	데이터 품질이란? (Garbage In, Garbage Out)	7
3.2	손상된 레코드 처리 (Handling Corrupt Records)	7
3.3	결측치 및 중복 데이터 처리	8
3.4	DLT를 활용한 자동 데이터 품질 관리 (Expectations)	8
4	3부: Databricks Lakeflow 및 DLT 실습	10
4.1	명령형 vs 선언형 파이프라인	10
4.2	DLT의 핵심: CDC (Change Data Capture) 자동화	10
5	4부: 실습 오류 및 해결 (Checkpoint Q&A)	12
5.1	사례: 스트리밍 체크포인트 오류	12
5.2	해결 방안	12
6	5부: 요약 및 점검	14
6.1	운영 파이프라인 체크리스트	14
6.2	FAQ (자주 묻는 질문)	14
6.3	빠르게 훑어보기 (1페이지 요약)	15
A	부록: 과제 및 공지사항	17

□ 핵심 요약

이 문서는 데이터 파이프라인을 개념 증명(PoC) 단계에서 실제 운영(Production) 환경으로 전환하는 데 필요한 핵심 원칙과 기술을 요약합니다.

데이터 과학자가 만든 노트북(프로토타입)을 비즈니스에서 신뢰하고 사용할 수 있는 자동화된 시스템으로 만드는 과정을 다룹니다.

이 과정에는 명확한 요구사항 정의(기능적/비기능적), 데이터 품질 보장, 거버넌스 수립, 그리고 확장성 확보가 필수적입니다.

Databricks의 **Lakeflow** 및 **DLT(Delta Live Tables)**와 같은 도구가 이 복잡한 과정을 어떻게 '선언형'으로 단순화하는지 살펴봅니다.

1 핵심 용어 정리

데이터 파이프라인 운영을 이해하기 위해 다음 용어들을 먼저 숙지해야 합니다.

주요 용어 해설표

용어	원어	한 줄 요약
운영(화)	Operationalizing	프로토타입을 실제 서비스로
개념 증명	PoC (Proof of Concept)	아이디어가 기술적으로
기능적 요구사항	Functional Req.	시스템이 '무엇을' 해야
비기능적 요구사항	Non-Functional Req.	시스템이 '어떻게' 동작하
Medallion Arch.	데이터를 Bronze → Silver → Gold 3단계로 정제하는 구조.	원석(Bronze)을 다듬어
데이터 거버넌스	Data Governance	데이터의 품질, 보안, 접근
데이터 계보	Data Lineage	데이터가 어디서 와서 어
ETL / ELT	-	데이터 처리 순서. (Extrac
Lakeflow	Lakeflow	Databricks의 데이터 파이프
DLT	Delta Live Tables	선언형 ETL 프레임워크
선언형 파이프라인	Declarative Pipeline	'어떻게'가 아닌 '무엇을
오토로더	Autoloader	클라우드 스토리지에 새
CDC	Change Data Capture	원본 데이터의 변경(Inse
확장성	Scalability	시스템이 증가하는 작업
탄력성	Elasticity	작업량 변화에 따라 자원
체크포인트	Checkpoint	스트리밍 작업이 어디까

2 1부: 데이터 파이프라인 운영의 개념

2.1 왜 "운영(Operationalizing)"이 필요한가?

데이터 과학자가 만든 프로토타입(PoC) 노트북은 특정 문제를 해결할 수 있음을 보여주지만, 비즈니스에서 매일 신뢰하며 사용하기에는 부족합니다.

□ 예제: title

- **PoC (노트북):** 셰프가 주방에서 실험적으로 만든 '신메뉴 레시피'와 같습니다. 맛은 있지만, 한 번에 1인분만 만들 수 있고, 셰프의 컨디션에 따라 품질이 달라질 수 있습니다.
- **운영 (Production):** 이 레시피를 '대규모 식품 공장'으로 가져와, 하루에 수만 개씩 일정한 품질로 자동 생산하는 시스템을 구축하는 것입니다.

운영 파이프라인은 신뢰성(Reliability), 자동화(Automation), 확장성(Scalability), 모니터링(Monitoring)을 갖추어야 합니다.

2.2 요구사항 정의: 모든 설계의 시작

설계를 시작하기 전에 비즈니스(고객)가 '무엇을' 원하고 '어떻게' 작동하길 기대하는지 명확히 해야 합니다. 이는 두 가지로 나뉩니다.

1. 기능적 요구사항 (Functional Requirements) - "무엇을?"

시스템이 사용자에게 제공해야 하는 구체적인 기능을 정의합니다.

- 비즈니스 규칙 (예: 특정 조건에서 거래 취소)
- 인증 및 권한 수준 (예: 관리자만 데이터 삭제 가능)
- 외부 인터페이스 (예: 여러 데이터 소스 연결)
- 리포팅 요구사항 (예: 일일 매출 요약 생성)
- 감사 추적 (예: 누가 데이터를 수정했는지 기록)

2. 비기능적 요구사항 (Non-Functional Requirements) - "어떻게?"

시스템의 품질, 성능, 제약 조건을 정의합니다. 기능적 요구사항만큼, 때로는 그 이상으로 중요합니다.

- **성능 (Performance):** 응답 시간(Latency), 처리량(Throughput) (예: 쿼리 응답은 5초 이내)
- **확장성 (Scalability):** 향후 데이터/사용자 증가에 대한 대비 (예: 10배 많은 데이터 처리 가능)
- **가용성 (Availability):** 시스템이 중단 없이 작동하는 시간의 비율
- **보안 (Security):** 데이터 접근 제어 및 규정 준수

비기능적 요구사항의 함정

비기능적 요구사항은 종종 간과되지만, 프로젝트 성패를 좌우합니다.

- **과잉 설계 (Over-design):** 비즈니스는 하루 지연된 데이터를 받아도 되는데, 엔지니어가 1초 미만 실시간 시스템을 구축하면 막대한 비용이 낭비됩니다.
- **미달 설계 (Under-design):** 1초 미만 응답을 기대하는 고객에게 10초 걸리는 시스템을 제공하면, 그 시스템은 비즈니스 가치를 잃게 됩니다.

예를 들어, 'Five Nines' (99.999%) 가용성은 1년 중 단 5분의 장애만 허용하며, 이는 엄청난 구축 비용을 요구합니다.

2.3 데이터 파이프라인의 6가지 핵심 원칙

효율적이고 견고한 데이터 레이크하우스(Data Lakehouse)를 구축하기 위한 6가지 지침입니다.

2.3.1 1. 데이터 큐레이션 및 신뢰할 수 있는 "데이터 제품" 제공

데이터를 목적에 맞게 정제하고 가공하여, 사용자가 신뢰하고 소비할 수 있는 '제품' 형태로 제공해야 합니다.

메달리온 아키텍처 (Medallion Architecture)

데이터를 품질 수준에 따라 3단계로 관리하는 표준 방식입니다.

- **Bronze (Raw Layer):** 원본 소스 데이터를 변경 없이 그대로 저장합니다. (원석)
- **Silver (Curated Layer):** Bronze 데이터를 가져와 정제(Cleansing), 필터링, 보강(Enriching)합니다. (정제된 은)
- **Gold (Final Layer):** Silver 데이터를 비즈니스 목적(예: BI, 리포팅, ML)에 맞게 집계(Aggregation)하고 요약합니다. (최종 제품, 금)

Bronze → Silver → Gold로 갈수록 데이터의 품질과 신뢰도는 높아지지만, 가공 비용(Cost)도 증가합니다.

2.3.2 2. 데이터 사일로 제거 및 이동 최소화

데이터를 불필요하게 복제하고 이동시키면 비용, 지연 시간(Latency), 품질 문제가 발생하며, 부서 간 데이터가 고립되는 '사일로(Silo)'가 생깁니다.

- **해결책:** 데이터 복제 대신 얇은 복제(Shallow Clone, 메타데이터만 복사), 뷰(Views), 델타 타임 트래블(Delta Time Travel) 등을 활용하여 데이터 중복을 최소화합니다.

2.3.3 3. 셀프 서비스 경험으로 가치 창출 민주화

데이터 팀뿐만 아니라 현업의 비즈니스 사용자들도 필요한 데이터를 직접 탐색하고 활용할 수 있어야 합니다. (데이터 민주화)

- **필수 조건:** 사용자가 데이터를 함부로 변경하거나 삭제하지 못하도록 적절한 "가드레일(Guardrails)", 즉 데이터 거버넌스(보안 및 접근 제어)가 반드시 전제되어야 합니다.

2.3.4 4. 전사적 데이터 거버넌스 전략 채택

데이터 거버넌스는 단순한 보안이 아니라, 데이터의 수명 주기 전반을 관리하는 시스템입니다.

- **데이터 품질 (Quality):** 데이터가 정확하고 일관되도록 제약 조건(Constraints)을 적용합니다.
- **데이터 카탈로그 및 계보 (Catalog & Lineage):** 데이터의 의미(메타데이터)를 정의하고, 데이터의 출처와 변환 이력(Lineage)을 추적 가능하게 합니다.
- **접근 제어 (Access Control):** 누가(Who) 어떤(What) 데이터에 접근할 수 있는지 관리합니다. (예: PII-개인식별정보 마스킹, 행/열 단위 접근 제한)

2.3.5 5. 개방형 인터페이스 및 포맷 사용

특정 벤더(Vendor)에 종속되는 독점적(Proprietary) 포맷 대신, 델타(Delta), 파케이(Parquet) 같은 개방형 포맷(Open Format)을 사용해야 합니다.

- **이유:** 1. **벤더 종속 탈피 (No Lock-in):** 다른 플랫폼으로 자유롭게 이전할 수 있습니다. 2. **상호 운용성 (Interoperability):** 다양한 3rd-party 도구와 쉽게 연동됩니다. 3. **비용 절감 (Lower Cost):** 값비싼 독점 플랫폼 라이선스를 피할 수 있습니다.

2.3.6 6. 확장성 및 비용/성능 최적화

대규모 데이터를 효율적으로 처리하려면 확장성과 비용 효율성을 모두 고려해야 합니다.

핵심: 스토리지와 컴퓨트의 분리 (Decoupling)

가장 중요한 아키텍처 원칙 중 하나입니다.

- **전통적 시스템 (결합):** 데이터를 저장하는 디스크(스토리지)와 데이터를 처리하는 CPU(컴퓨트)가 한 서버에 묶여 있습니다. 데이터만 10배 늘어도, 비싼 CPU까지 10배 증설해야 했습니다. (예: Elasticsearch)
- **현대적 레이크하우스 (분리):** 데이터는 저렴한 클라우드 스토리지(S3, ADLS 등)에 무한히 저장하고, 데이터 처리가 필요할 때만 컴퓨트 클러스터를 빌려 씁니다.

□ 예제: title

- **스토리지 (Storage):** 물건을 보관하는 '창고'. 창고 크기는 저렴하게 무한히 늘릴 수 있습니다.
 - **컴퓨트 (Compute):** 물건을 가공하는 '공장 기계'. 비싸지만, 물건을 가공할 때만 기계를 켜서 사용료를 냅니다.
- 이 둘을 분리하면, 창고가 아무리 커져도 공장 운영비는 필요한 만큼만 지불하게 되어 매우 효율적입니다.

3 2부: 데이터 품질 및 검증

3.1 데이터 품질이란? (Garbage In, Garbage Out)

”쓰레기가 들어가면, 쓰레기가 나온다 (GIGO)” 데이터 파이프라인의 핵심은 데이터 품질을 보장하는 것입니다. 부정확한 데이터를 기반으로 한 분석이나 AI 모델은 잘못된 비즈니스 결정으로 이어집니다.

- **정확성 (Accuracy):** 데이터가 실제 값과 일치하는가?
- **일관성 (Consistency):** 데이터가 시스템 내에서 모순 없이 일관되는가? (예: 'NY'와 'New York'이 혼용)
- **완전성 (Completeness):** 필수 데이터가 누락되지 않았는가?
- **적시성 (Timeliness):** 데이터가 필요한 시점에 제공되는가? (신선도)
- **무결성 (Integrity):** 데이터 간의 관계(예: FK)가 올바른가?

3.2 손상된 레코드 처리 (Handling Corrupt Records)

소스 데이터는 스키마 불일치, 형식 오류, 누락 값 등 다양한 이유로 '손상될' 수 있습니다. Spark은 이러한 데이터를 처리하는 3가지 모드(ParseMode)를 제공합니다.

Spark의 3가지 오류 처리 모드

1. PERMISSIVE (기본값)

- **동작:** 오류가 발생한 레코드를 `_corrupt_record`라는 별도 컬럼에 저장하고, 파싱 가능한 컬럼은 null로 채웁니다. 작업은 중단되지 않습니다.
- **용도:** 파이프라인 중단 없이 모든 데이터를 일단 수집한 후, 나중에 손상된 데이터를 분석하거나 수정할 때 유용합니다.

2. DROPMALFORMED

- **동작:** 손상된 레코드를 즉시 삭제(무시)합니다.
- **용도:** 품질이 중요하지 않거나, 일부 데이터가 손실되어도 무방한 로그 분석 등에 사용됩니다.

3. FAILFAST

- **동작:** 손상된 레코드를 만나는 즉시 작업을 중단시키고 예외(Exception)를 발생시킵니다.
- **용도:** 데이터 품질이 매우 엄격하게 요구되는 금융 거래 데이터 등, 단 하나의 오류도 허용되지 않는 경우에 사용됩니다.

또한, `option("badRecordsPath", "경로")`를 지정하여 오류 레코드의 원본 파일을 별도 위치에 기록할 수 있습니다.

3.3 결측치 및 중복 데이터 처리

- 중복 데이터: `dropDuplicates(["id", "color"])`와 같이 고유 키를 기준으로 중복 행을 제거합니다.
- 결측치 (Missing Values):
 - 행 삭제 (Drop): `dropna()`를 사용하여 결측치가 있는 행을 무시합니다. (정보 손실 위험)
 - 대체값 (Placeholder): 스키마를 위반하지 않도록 -1이나 'N/A' 같은 값으로 채웁니다.
 - 기본 대체 (Basic Imputing): `na.fill()`을 사용하여 전체 평균값이나 중앙값으로 채웁니다.
 - 고급 대체 (Advanced Imputing): ML 모델(예: 회귀)을 사용하여 결측치를 예측합니다.

3.4 DLT를 활용한 자동 데이터 품질 관리 (Expectations)

Delta Live Tables (DLT)는 데이터 품질 규칙을 파이프라인 정의에 직접 '기대(Expectations)'로 선언할 수 있게 해줍니다.

DLT Expectations: 데이터 품질 선언

품질 규칙(제약 조건)을 정의하고, 위반 시(On Violation) 어떻게 처리할지 지정합니다.

- **EXPECT ... ON VIOLATION FAIL UPDATE:** 규칙 위반 시 파이프라인을 중단합니다. (Spark의 FAILFAST와 유사)
- **EXPECT ... ON VIOLATION DROP ROW:** 규칙 위반 시 해당 행을 삭제합니다. (Spark의 DROPMALFORMED와 유사)
- **EXPECT ... (처리 지정 없음):** 규칙 위반 시 데이터는 통과시키되, 위반 내역을 메트릭으로 기록하여 모니터링할 수 있게 합니다.

```

1 import dlt
2 from pyspark.sql.functions import col
3
4 # @dlt.expect_or_drop: 규칙위반시행삭제
5 @dlt.expect_or_drop("valid_age", "age > 0 AND age < 120")
6
7 # @dlt.expect_or_fail: 규칙위반시파이프라인중단
8 @dlt.expect_or_fail("valid_email", "email IS NOT NULL")
9
10 # @dlt.expect: 위반시기록만함
11 @dlt.expect("reasonable_score", "score >= 50")
12
13 @dlt.table(
14     comment="데이터 품질제약조건이적용된 Silver 테이블"
15 )
16 def users_silver():
17     return (
18         dlt.read_stream("users_bronze")
19         .select("id", "age", "email", "score")

```

20

)

Listing 1: DLT Python에서 Expectations를 선언하는 예시

데이터 격리 (Quarantining)

규칙을 위반한(실패한) 데이터를 DROP하거나 FAIL시키는 대신, 별도의 '격리(Quarantine) 테이블'로 라우팅하는 패턴도 많이 사용됩니다.

이를 통해 메인 파이프라인은 계속 실행하면서, 실패한 데이터는 나중에 수동으로 검토하고 수정하여 다시 파이프라인에 주입(re-inject)할 수 있습니다.

4 3부: Databricks Lakeflow 및 DLT 실습

Lakeflow는 데이터 수집(Ingest), 변환(Transform), 오케스트레이션(Orchestrate)을 통합 관리하는 Databricks의 플랫폼입니다. DLT는 Lakeflow의 핵심 기능으로, '선언형' 파이프라인을 구축합니다.

4.1 명령형 vs 선언형 파이프라인

전통적인 파이프라인 코딩과 DLT의 차이점입니다.

□ 예제: title

- **명령형 (Imperative - 전통 방식):** 셰프에게 "1. 소고기 200g을 굽고, 2. 야채를 씻고, 3. 빵을 데우고, 4..." 라며 '모든 절차(How)'를 직접 지시합니다. (개발자가 스케일링, 오류 처리, 리트라이, 상태 관리 코드를 모두 작성)
- **선언형 (Declarative - DLT 방식):** 메뉴판을 보고 "스테이크 주세요"라고 '원하는 결과(What)'만 주문합니다. 주방(DLT 엔진)이 최적의 레시피로 요리(파이프라인 실행), 품질 검사, 서빙(데이터 저장)을 알아서 처리합니다.

DLT를 사용하면 개발자는 비즈니스 로직(변환)에만 집중하고, 복잡한 인프라 관리(오류 처리, 확장, 최적화, 모니터링)는 DLT 엔진에 맡길 수 있습니다.

4.2 DLT의 핵심: CDC (Change Data Capture) 자동화

DLT의 가장 강력한 기능 중 하나는 원본 데이터의 변경(Insert, Update, Delete)을 자동으로 감지하고 Silver 테이블에 반영하는 `dlt.apply_changes()` 입니다.

이 기능을 사용하면 복잡한 Merge/Upsert 로직을 직접 코딩할 필요가 없습니다.

```

1 import dlt
2 from pyspark.sql.functions import col, expr
3
4 # 소스데이터경로에 (: S3, ADLS)
5 SOURCE_PATH = "/path/to/source-data/"
6
7 # 1. Bronze: 로Autoloader 원시데이터스트리밍 (JSON, CSV 등)
8 @dlt.table(
9     comment="로Autoloader 원시트랜잭션수집 "
10 )
11 def bronze_transactions():
12     return (
13         spark.readStream
14             .format("cloudFiles") # Autoloader 사용
15             .option("cloudFiles.format", "json")
16             .load(SOURCE_PATH)
17     )
18 
```

```

19 # 2. View: Bronze 데이터를 Silver 적용하기전정제
20 # 뷰는 ( 디스크에저장되지않고인메모리에서사용됨 )
21 @dlt.view(
22     comment="Bronze 데이터정제및시퀀스컬럼추가 "
23 )
24 def clean_transactions():
25     return (
26         dlt.read_stream("bronze_transactions")
27         .selectExpr(
28             "transaction_id",
29             "CAST(amount AS DOUBLE)",
30             "operation_type", # 'INSERT', 'UPDATE', 'DELETE'
31             "CAST(update_timestamp AS LONG) as sequence" # 변경순서
32         )
33     )
34
35 # 3. Silver: CDC (Change Data Capture) 적용
36 @dlt.table(
37     comment="정제된 트랜잭션데이터를로 CDC 에Silver 적용"
38 )
39 def silver_transactions():
40     dlt.apply_changes(
41         target = "silver_transactions", # 최종대상테이블
42         source = dlt.read_stream("clean_transactions"), # 소스정제된 ( 뷰 )
43         keys = ["transaction_id"], # 레코드의고유키 (PK)
44         sequence_by = col("sequence"), # 변경순서식별컬럼
45         apply_as_deletes = expr("operation_type = 'DELETE'") # 삭제조건
46         # 이operation_type 'DELETE'인 경우, 해당키를가진레코드를삭제
47         # INSERT, 는UPDATE 자동으로처리됨 (SCD Type 1)
48     )

```

Listing 2: DLT를 사용한 Bronze → Silver CDC 파이프라인 예시

dlt.apply_changes() 상세

- **target:** 데이터가 최종적으로 저장될 테이블 이름.
- **source:** 변경 사항이 포함된 스트리밍 데이터프레임.
- **keys:** 레코드를 고유하게 식별하는 키(PK) 리스트.
- **sequence_by:** 변경 순서를 나타내는 컬럼 (예: 타임스탬프). 동일한 키의 레코드가 여러 개일 경우, 이 값이 가장 높은(최신) 레코드를 적용합니다.
- **apply_as_deletes:** 이 조건이 참(True)이 되는 레코드는 'DELETE'로 처리됩니다.

5 4부: 실습 오류 및 해결 (Checkpoint Q&A)

과제 수행 중 흔히 발생하는 ‘checkpointLocation’ 오류는 스트리밍 파이프라인의 상태 관리와 밀접한 관련이 있습니다.

5.1 사례: 스트리밍 체크포인트 오류

Q: 증상 (Symptom)

과제 3번에서 `spark.readStream.format("delta").load(...).writeStream.option("checkpointLocation", ...).start()` 코드를 실행할 때, 체크포인트 위치(checkpointLocation) 관련 오류가 계속 발생합니다. 경로를 바꿔봐도 해결되지 않습니다.

원인 분석 (Cause Analysis)

스트리밍 쿼리는 '체크포인트(Checkpoint)'에 자신이 어디까지 데이터를 처리했는지 '상태(State)'를 기록합니다. 이 오류는 주로 두 가지 이유로 발생합니다.

1. **체크포인트 충돌:** 이전에 실행했던 스트리밍 쿼리의 체크포인트가 남아있는 상태에서, 파이프라인 코드(로직)를 수정하거나 소스 데이터를 삭제한 후 다시 실행하면, 이전 상태와 현재 상태가 충돌하여 오류가 발생합니다.
2. **잘못된 경로 사용:** (Databricks 환경) `dbfs:/...` 같은 경로 대신, Unity Catalog (UC)에서 관리하는 `/Volumes/...` 경로를 사용해야 할 수 있습니다.

5.2 해결 방안

□ 예제: title

스트리밍 쿼리를 새로 시작하기 전에, 이전 체크포인트 디렉토리를 수동으로 삭제하여 상태를 초기화합니다.

```

1 # 체크포인트경로정의
2 checkpoint_path = "/path/to/my/checkpoint"
3
4 # (1) 스트리밍쿼리중지이미 ( 실행중이라면 )
5 # for s in spark.streams.active:
6 #     s.stop()
7
8 # (2) 체크포인트디렉토리강제삭제
9 # 주의 (: 실제운영환경에서는복구불가능하므로신중해야함 )
10 dbutils.fs.rm(checkpoint_path, recurse=True)
11
12 # (3) 스트리밍쿼리다시시작
13 (
14     spark.readStream
15         .format("delta")
16         .load(...)

```

```
17     .writeStream
18     .option("checkpointLocation", checkpoint_path)
19     .table(...)
20 )
```

Listing 3: 스트리밍 시작 전 체크포인트 강제 삭제

해결책 2: DLT/Lakeflow 사용 (자동 방식)

근본적인 해결책: **DLT가 상태 관리를 자동화합니다.**

이러한 수동 체크포인트 관리는 번거롭고 오류를 유발하기 쉽습니다.

DLT/Lakeflow는 파이프라인의 상태를 엔진이 직접 관리합니다. 만약 파이프라인을 처음부터 다시 실행하고 싶다면, DLT 실행 시 **"Full Refresh (전체 새로 고침)"** 옵션을 선택하면 됩니다.

"Full Refresh"는 DLT 엔진이 **자동으로** 모든 이전 상태와 체크포인트를 삭제하고, 소스 데이터를 처음부터 다시 처리하도록 지시합니다. (수동 `dbutils.fs.rm` 불필요)

6 5부: 요약 및 점검

6.1 운영 파이프라인 체크리스트

- ✓ **요구사항**: 기능적 요구사항(What)과 비기능적 요구사항(How - 성능, 가용성, 비용)이 명확하게 정의되었는가?
- ✓ **아키텍처**: Medallion 아키텍처(Bronze/Silver/Gold)가 설계되었는가?
- ✓ **품질**: 데이터 품질 규칙(Expectations)이 정의되었고, 위반 시 처리(Fail/Drop/Log) 전략이 수립되었는가?
- ✓ **오류 처리**: 손상된 레코드(Corrupt records) 및 파이프라인 실패 시 복구/재시도(Retry) 전략이 있는가?
- ✓ **거버넌스**: 데이터 접근 제어, PII 마스킹, 데이터 계보(Lineage)가 고려되었는가?
- ✓ **최적화**: 스토리지와 컴퓨트가 분리되었는가? 워크로드에 맞게 탄력적(Elastic)으로 자원이 조절되는가?
- ✓ **자동화**: 수동 개입 없이 파이프라인이 스케줄링(Orchestration)되고 자동 실행되는가?
- ✓ **모니터링**: 파이프라인 상태, 성능, 데이터 품질 메트릭이 모니터링되고 알람이 설정되었는가?

6.2 FAQ (자주 묻는 질문)

Q: ETL과 ELT 중 무엇을 사용해야 하나요?

A: 상황에 따라 다릅니다. 원본 데이터가 이미 잘 구조화되어 있고(예: RDBMS) 스키마가 명확하면 전통적인 **ETL**(추출 → 변환 → 적재)이 효율적일 수 있습니다.

하지만 대부분의 현대 데이터(비정형, 반정형)는 스키마가 불명확하므로, 일단 원본(Bronze)을 적재(**L**)한 후 Lakehouse의 강력한 컴퓨팅 파워를 이용해 **변환(T)**하는 **ELT** 방식이 더 유연하고 권장됩니다.

Q: DLT는 SQL과 Python 중 무엇으로 작성해야 하나요?

A: 둘 다 가능하며, 동일한 파이프라인 내에서 혼용할 수도 있습니다. 간단한 변환이나 SQL에 익숙한 분석가들은 SQL을, 복잡한 로직, ML 전처리, Python 라이브러리 활용이 필요하면 Python을 사용합니다.

Q: 파이프라인은 항상 처음부터 다시 실행해야 하나요?

A: 아닙니다. 그럴 필요가 없습니다. DLT와 같은 현대적 파이프라인은 **증분 처리(Incremental Processing)**를 기본으로 합니다. 즉, 마지막 실행 이후 새롭게 추가되거나 변경된 데이터만 처리하여 비용과 시간을 획기적으로 줄입니다.

"Full Refresh"는 스키마가 크게 변경되거나 로직을 근본적으로 수정했을 때만 예외적으로 사용합니다.

Q: 확장성(Scalability)과 탄력성(Elasticity)의 차이는 무엇인가요?

A:

- **확장성(Scalability):** 시스템의 최대 용량을 늘릴 수 있는 능력입니다. (예: 100명용 → 1000명용 시스템으로 업그레이드)
- **탄력성(Elasticity):** 현재 수요에 맞춰 자원을 동적으로 늘리거나 줄이는 능력입니다. (예: 낮에 1000명이 쓰면 1000명용, 밤에 10명만 쓰면 10명용으로 자동 축소)

탄력성은 클라우드 환경에서 비용을 최적화하는 핵심 개념입니다. Databricks의 Serverless 기능이 대표적인 예입니다.

6.3 빠르게 훑어보기 (1 페이지 요약)

PoC (프로토타입) vs Production (운영)

PoC: 수동 실행, 불안정, 품질 보장 X, 1회성 분석 (예: 노트북)

↓ **Operationalizing** ↓

Production: 자동화, 신뢰성, 품질 보장(Quality), 확장성, 모니터링 (예: DLT)

요구사항 (Requirements)

- **기능적 (Functional):** 무엇을? (기능)
- **비기능적 (Non-Func.):** 어떻게? (성능, 품질, 비용) ← 놓치기 쉬움!

Medallion 아키텍처 (품질 향상)

Bronze (원시) → **Silver** (정제/검증) → **Gold** (집계/최종 제품)

6대 핵심 원칙 (Guiding Principles)

1. 데이터 제품 (Data-as-Products)
2. 사일로 제거 (No Silos)
3. 셀프 서비스 (Self-Service) + 거버넌스
4. 전사적 거버넌스 (Governance)
5. 개방형 포맷 (Open Formats)
6. 확장 및 최적화 (Scale & Cost)

DLT (Delta Live Tables) 핵심 기능

- **선언형 (Declarative):** "What"만 정의 (엔진이 "How" 처리)
- **CDC 자동화:** `dlt.apply_changes()`
- **품질 자동화:** `@dlt.expect_...` (Expectations)
- **상태 관리:** 체크포인트, Full Refresh 자동화

확장성 (Scaling)

- **수평적 확장 (Scale Out):** 서버 대수 추가 (권장)
- **수직적 확장 (Scale Up):** 서버 사양 업그레이드
- **핵심:** 스토리지(창고)와 컴퓨트(공장)는 반드시 분리!

A 부록: 과제 및 공지사항

다음은 강의 초반에 공지된 행정 사항 요약입니다.

- **Assignment 1 (Spark):** 채점이 완료되었으며 성적이 배포되었습니다. 성적을 받지 못한 경우 교수진에게 연락 바랍니다.
- **Assignment 2 (ETL/EDA):** 채점이 진행 중이며, 곧 완료될 예정입니다.
- **Assignment 3 (Batch & Streaming):** 과제가 공개되었으며, 현재 진행 중이어야 합니다.
- **Quiz 1:** 퀴즈가 이번 주에 공개되었습니다. (시간 제한이 있는 퀴즈가 아니라 며칠 또는 1주일 정도의 시간을 두고 푸는 방식입니다.)
- **Case Study 1 (Data Architectures):**
 - 이번 주 말(목요일 또는 금요일)에 공개될 예정입니다.
 - 팀 프로젝트로 진행되며, 팀은 Canvas의 그룹 도구를 통해 랜덤으로 배정됩니다.
 - 과제 완료까지는 최소 2주의 시간이 주어질 것입니다.
 - 과제 수행을 위한 템플릿(Deck)이 제공될 예정입니다.
 - (참고: 여러 번의 케이스 스터디가 있을 예정이며, 이는 다른 학생들과 네트워킹할 기회입니다.)
- **질문:** 질문은 **Slack**을 이용하는 것이 가장 좋습니다. Slack을 통해 질문과 답변을 공유하면 모든 학생이 도움을 받을 수 있습니다. 개인적인 질문은 교수 또는 TA에게 직접 연락 가능합니다.