October 26, 2025

▣ **강의명:** CSCI E-103: 재현 가능한 머신러닝

▣ **주차:** Lecture 05

▣ **교수명:** Anindita Mahapatra

Eric Gieseke

▣ **목적:** Lecture 05의 핵심 개념 학습

Contents

1 개요: 데이터 레이크 강의 핵심 요약

□ 핵심 요약

이 문서는 5주차 '데이터 레이크' 강의의 핵심 내용을 정리합니다.
데이터가 고립되는 **'데이터 사일로'** 문제를 해결하기 위해 등장한 **'데이터 레이크'**의 개념을 배웁니다. 하지만 데이터 레이크가 적절한 관리 없이 방치되면 **'데이터 스웜프(늪)'**가 되는 위험성을 인지합니다.
이 문제를 해결하고 데이터 웨어하우스의 장점(신뢰성, 성능)을 결합한 **'데이터 레이크하우스'**가 왜 필요한지, 그리고 그 기반 기술인 **'델타 레이크(Delta Lake)'**가 어떻게 동작하는지(ACID, 타임 트래블) 학습합니다.
마지막으로, 실제 데이터 파이프라인을 구축하는 **'메달리온 아키텍처'**(Bronze/Silver/Gold)와 파이프라인의 성능을 저하하는 주요 원인들(Spill, Shuffle, Skew)을 이해합니다.

이 문서는 강의 내용과 실습 코드를 통합하여, 데이터 엔지니어링을 처음 접하는 사람도 쉽게 이해할 수 있도록 구성되었습니다.

2 핵심 용어 정리

데이터 엔지니어링 분야는 유사하지만 미묘하게 다른 용어들이 많아 혼란스러울 수 있습니다. 각 개념의 핵심 차이를 표로 정리했습니다.

용어 (원어)	핵심 설명 (비유)	주요 특징	관계
데이터 사일로 (Data Silo)	부서별로 따로 쓰는 '데이터 섬'. (예: 재무팀의 Excel, 마케팅팀의 DB)	- 데이터 고립 및 중복 - 전사적 분석 불가	데이터 레이크가 해결하려는 **문제**
데이터 레이크 (Data Lake)	모든 데이터를 원본(Raw) 그대로 저장하는 거대한 '저장용 호수'.	- 모든 데이터 유형 저장 - **스키마 온 리드 (Schema on Read)** - 저렴한 저장 비용	사일로를 해결. 스웜프가 될 수 있음.
데이터 스웜프 (Data Swamp)	무엇이 어디 있는지 모르는 '데이터 늪'.	- 거버넌스 부재 - 메타데이터 관리 실패 - 데이터 신뢰도 하락	데이터 레이크가 **실패**한 상태
데이터 웨어하우스 (Data Warehouse)	정제되고 구조화된 데이터만 보관하는 '제품 창고' 또는 '백화점'.	- 정형 데이터만 저장 - **스키마 온 라이트 (Schema on Write)** - BI 및 리포트용, 고성능	레이크와 상호 보완적. (비유: 페라리)
데이터 레이크하우스 (Data Lakehouse)	'호수 (Lake)'의 유연함과 '창고 (Warehouse)'의 신뢰성을 합친 '하이브리드'.	- 데이터 레이크 + 데이터 웨어하우스 - 단일 플랫폼에서 BI와 AI 모두 지원	레이크의 진화형. 델타 레이크가 핵심 기반.
델타 레이크 (Delta Lake)	데이터 레이크를 '신뢰할 수 있게' 만드는 '관리 프로토콜'.	- Parquet 기반 + 트랜잭션 로그 - **ACID 트랜잭션, 타임 트래블** 지원	레이크하우스를 구현하는 **핵심 기술**
메달리온 아키텍처 (Medallion Arch.)	데이터를 '동/은/금메달'처럼 3단계로 정제하는 파이프라인 구조.	- **Bronze**: 원본 (Raw) - **Silver**: 정제/통합 (Cleaned) - **Gold**: 집계/활용 (Aggregated)	레이크하우스에서 데이터를 관리하는 **방법론**
DAG (Directed Acyclic Graph)	'방향이 있고 순환하지 않는 그래프'. 작업의 의존 관계를 표현.	- (A → B), (A → C) → D - 작업 병렬성/의존성 정의	스파크 (Spark) 및 워크플로우의 **작동 원리**
데이터 메시 (Data Mesh)	데이터 관리를 중앙팀이 아닌 '각 현업 부서 (도메인)'가 맡는 **조직 문화**.	- 분산 소유권 (Domain Ownership) - 데이터 = 제품 (Data as a Product)	레이크하우스와 함께 적용 가능한 **조직/문화** 전략
데이터 패브릭 (Data Fabric)	여러 곳에 흩어진 데이터를 '하나의 천'처럼 연결하는 **기술 아키텍처**.	- 하이브리드/멀티 클라우드 통합 - 메타데이터 기반 자동화	데이터 메시보다 **기술/아키텍처**에 집중

3 데이터 저장소의 진화: 사일로에서 레이크하우스까지

데이터를 저장하고 활용하는 방식은 비즈니스 요구사항에 따라 진화해왔습니다.

3.1 데이터 사일로 (Data Silos): 고립된 섬

데이터 사일로는 데이터가 조직 내의 여러 부서나 시스템에 고립되어 저장된 상태를 말합니다.

□ **예제:**

비유: 고립된 섬
A 회사의 재무팀은 재무 데이터(매출)를 자신들의 Excel 파일에, 마케팅팀은 고객 데이터(클릭 기록)를 별도의 마케팅 솔루션에 저장합니다. 두 팀의 데이터가 연결되지 않아 "어떤 마케팅 활동이 실제 매출로 이어졌는지" 분석하는 것이 거의 불가능합니다. 각 부서가 자신만의 '데이터 섬'에 갇혀있는 것입니다.

데이터 사일로의 주요 문제점:

- **구조적 문제 (Structural):** 시스템 자체가 데이터 공유를 고려하지 않고 설계되었습니다.
- **정치적 문제 (Political):** 부서가 데이터를 '소유물'로 여기고 공유를 거부합니다.
- **성장 문제 (Growth):** 조직이 성장하며 도입한 새 시스템이 기존 시스템과 호환되지 않습니다.
- **벤더 종속 (Vendor Lock-in):** 특정 솔루션(SaaS 등)이 데이터 외부 반출을 어렵게 만듭니다.

3.2 데이터 레이크 (Data Lake): 모든 것을 담는 호수

데이터 사일로 문제를 해결하기 위해 등장한 것이 **데이터 레이크(Data Lake)**입니다. "모든 종류의 데이터를(정형, 반정형, 비정형) 원본(Raw) 형태 그대로 한곳에 저장하자"는 개념입니다.

- **장점:** 저렴한 비용으로 모든 데이터를 한곳에 모아 사일로를 제거합니다. 데이터 과학자와 ML 엔지니어는 원본 데이터에 자유롭게 접근하여 새로운 가치를 탐색할 수 있습니다.
- **핵심 특징: 스키마 온 리드 (Schema on Read)**

개념 비교: 스키마 온 리드 vs. 스키마 온 라이트

스키마 온 리드 (Schema on Read)	**스키마 온 라이트 (Schema on Write)**
(데이터 레이크 방식)	(전통적 데이터베이스/웨어하우스 방식)
1. 저장 (Write): 일단 그냥 저장한다. (스키마 없음)	**1. 저장 (Write):** 엄격한 스키마(테이블 구조) 검사. (불일치 시 저장 실패)
2. 읽기 (Read): 읽는 시점에 스키마(구조)를 정의하여 해석한다.	**2. 읽기 (Read):** 이미 정의된 스키마대로 빠르게 읽는다.
장점: 빠른 수집, 유연함 (모든 데이터 수용)	**장점:** 데이터 품질 보장, 빠른 읽기 속도
단점: 읽기 복잡, 데이터 품질 보장 어려움	**단점:** 수집이 까다로움, 비정형 데이터 저장 불가

3.3 데이터 스웜프 (Data Swamp): 관리되지 않는 늪

데이터 레이크는 강력하지만, 치명적인 함정이 있습니다. '스키마 온 리드'의 유연함은 곧 '관리의 부재'를 의미할 수 있습니다.

데이터가 계속 쌓이지만,

- 어떤 데이터인지 설명하는 **메타데이터(Metadata)가 없거나,**
- 데이터의 품질을 관리하는 **거버넌스(Governance)가 없다면,**

데이터 레이크는 아무도 무엇이 어디 있는지 알 수 없는 **데이터 스웜프(Data Swamp, 늪)**가 되어버립니다.

□ 예제:

비유: 창고 vs. 폐기물 처리장
잘 관리된 데이터 레이크는 모든 물건에 '라벨'이 붙어있고 '위치'가 기록된 거대한 창고(아마존 물류센터)와 같습니다. 하지만 데이터 스웜프는 온갖 종류의 쓰레기를 한곳에 버리기만 한 폐기물 처리장과 같습니다. 무언가 유용한 것이 있을지도 모르지만, 찾을 수가 없습니다.

주의사항

진자의 비유 (The Pendulum)
데이터 관리는 양극단을 오가는 진자와 같습니다.

- **데이터 사일로 (Silos):** 너무 엄격하게 분리되어 활용이 불가능.
- **데이터 스웜프 (Swamps):** 너무 자유롭게 방치되어 활용이 불가능.

우리의 목표는 이 진자의 중간 지점, 즉 **관리되는 데이터 레이크(Governed Data Lake)**를 찾는 것입니다.

3.4 데이터 웨어하우스 (Data Warehouse): 정제된 백화점

데이터 레이크와 자주 비교되는 개념이 **데이터 웨어하우스(Data Warehouse, DW)**입니다. DW는 비즈니스 인텔리전스(BI) 및 리포팅을 위해 '정제되고', '구조화된' 데이터만 저장하는 시스템입니다.

- **핵심 특징:** 스키마 온 라이트 (Schema on Write)
- **주요 사용자:** 비즈니스 분석가 (Business Analysts)
- **장점:** 데이터가 신뢰할 수 있고, 쿼리(조회) 속도가 매우 빠릅니다.
- **단점:** 비싸고, 비정형 데이터를 처리할 수 없으며, 데이터 과학(ML) 활용에 제한적입니다.

3.5 비교: 데이터 레이크 (DL) vs. 데이터 웨어하우스 (DW)

□ **예제:**

비유: 페라리 vs. 트랙터 트레일러

- **데이터 웨어하우스 (페라리):** 매우 빠르고 멋지지만(고성능 쿼리), 2명밖에 못 타고(정형 데이터), 짐을 실을 수 없습니다(유연성 낮음).
- **데이터 레이크 (트랙터 트레일러):** 페라리만큼 빠르진 않지만, 모든 종류의 짐(모든 데이터)을 원하는 만큼(대용량) 실어 나를 수 있습니다.

특징	데이터 레이크 (Data Lake)	데이터 웨어하우스 (Data Warehouse)
데이터 종류	모든 데이터 (정형, 반정형, 비정형)	정형 데이터 (Structured)
데이터 상태	원본 (Raw), 정제된 데이터 모두	정제되고 변환된 데이터 (Processed)
스키마	**Schema on Read** (읽을 때 정의)	**Schema on Write** (저장할 때 정의)
프로세스	ELT (Extract, Load → Transform)	ETL (Extract, Transform → Load)
주요 사용자	데이터 과학자, ML 엔지니어	비즈니스 분석가, SQL 사용자
주요 용도	AI / ML 모델링, 데이터 탐색	BI 리포트, 대시보드
비용	저렴 (스토리지/컴퓨트 분리)	고비용 (스토리지/컴퓨트 결합)
형식	오픈 포맷 (Open Formats)	독점 포맷 (Proprietary)

3.6 데이터 레이크하우스 (Data Lakehouse): 두 세계의 통합

과거에는 BI를 위해 DW를, AI/ML을 위해 DL을 따로 구축했습니다. 하지만 이로 인해 데이터가 다시 사일로화되고(DW/DL 사일로), 비용이 이중으로 드는 문제가 발생했습니다.

데이터 레이크하우스(Data Lakehouse)는 이 두 시스템을 하나로 통합하려는 시도입니다. 즉, 데이터 레이크의 **저렴한 비용과 유연성** 위에 데이터 웨어하우스의 **신뢰성, 거버넌스, 성능**을 구현한 것입니다.

이를 가능하게 하는 핵심 기술이 바로 **델타 레이크(Delta Lake)**입니다.

4 델타 레이크: 강력한 레이크하우스의 기반 (Delta Lake)

4.1 델타 레이크란? (포맷이 아닌 프로토콜)

주의사항

오해 바로잡기: 델타 레이크는 파일 포맷이 아닙니다.
델타 레이크(Delta Lake)는 **프로토콜(Protocol) 또는 관리 계층**입니다. 여러분이 저장하는 실제 데이터 파일은 여전히 효율적인 **Parquet** 포맷입니다.
델타 레이크는 이 Parquet 파일들 위에 **_delta_log**라는 트랜잭션 로그 폴더를 추가하여, Parquet 파일만으로는 불가능했던 강력한 기능들을 제공합니다.

전통적인 데이터 레이크(Hadoop, S3)의 파일(Parquet, CSV)은 한번 쓰면 수정이 불가능(Immutable)한 경우가 많아 다음과 같은 치명적인 문제들이 있었습니다.

- 데이터 일부만 수정(UPDATE)하거나 삭제(DELETE)하기가 극도로 어렵습니다.
- 작업이 중간에 실패하면 데이터가 오염됩니다 (예: 중복 데이터).
- 여러 사용자가 동시에 데이터를 읽고 쓸 때 일관성이 깨집니다.

델타 레이크는 이 모든 문제를 해결하여 데이터 레이크를 신뢰할 수 있는 저장소로 만듭니다. (대안 기술: Apache Iceberg, Apache Hudi)

4.2 델타 레이크의 핵심 기능 (The Guarantees)

- **ACID 트랜잭션 (ACID Transactions):** 전통적인 데이터베이스처럼 원자성(Atomicity), 일관성(Consistency), 고립성(Isolation), 지속성(Durability)을 보장합니다. 작업이 중간에 실패해도 데이터가 오염되지 않으며, 여러 사용자가 동시에 접근해도 데이터가 깨지지 않습니다.
- **스키마 관리 (Schema Evolution & Enforcement):** 데이터를 쓸 때 스키마가 일치하는지 검사(Enforcement)할 수 있으며, 의도적으로 스키마(컬럼)를 변경(Evolution)하는 것도 허용합니다.
- **타임 트래블 (Time Travel):** 모든 변경 이력(_delta_log)을 저장하므로, "어제 날짜의 데이터" 또는 "버전 5의 데이터"처럼 과거의 특정 시점으로 데이터를 되돌려 조회할 수 있습니다. (예: 'VERSION AS OF 5')
- **DML 지원 (UPDATE, DELETE, MERGE):** 데이터 레이크의 파일에 직접 SQL의 'UPDATE', 'DELETE', 'MERGE' (Upsert) 명령을 실행할 수 있습니다. (예: GDPR로 인한 사용자 데이터 삭제)
- **성능 최적화 (Optimization):** 데이터 스키핑(Data Skipping, 통계정보 기반 파일 필터링), Z-Ordering(다차원 정렬) 등을 통해 쿼리 성능을 향상시킵니다.

5 데이터 아키텍처 및 원칙

5.1 메달리온 아키텍처 (Medallion Architecture)

레이크하우스에서 데이터를 효과적으로 정제하고 관리하기 위해 가장 널리 쓰이는 패턴이 **메달리온 아키텍처**입니다. 데이터의 품질 수준에 따라 3개의 레이어(테이블)로 구분합니다.

□ 예제:

비유: 광물 제련소

- **Bronze (동메달):** 광산에서 막 캐낸 '원본 광석' (Raw Data). 어떤 가공도 하지 않고 그대로 쌓아둡니다.
- **Silver (은메달):** 원본 광석에서 불순물을 제거하고(Null, 중복 제거) 필요한 것끼리 합친 '제련된 금속' (Cleaned, Filtered).
- **Gold (금메달):** 제련된 금속을 특정 목적에 맞게 가공한 '최종 완제품' (Aggregated, Business-ready).

레이어	Bronze	Silver	Gold
데이터 상태	원본 (Raw)	정제됨 (Cleaned)	집계됨 (Aggregated)
주요 작업	- 모든 소스 데이터 수집- (예: JSON, CSV, Parquet)	- Null/중복 데이터 제거- 데이터 유형(Type) 통일- 여러 소스 조인(Join)- (이때부터 Delta 포맷)	- BI/리포트를 위한 집계- (예: 부서별 월간 매출)- AI/ML용 피처(Feature) 생성
데이터 구조	소스 시스템과 동일	3정규화(3NF) 또는 유사	비정규화(Denormalized), 스타 스키마
주요 사용자	데이터 엔지니어	데이터 엔지니어, 데이터 과학자	비즈니스 분석가, 데이터 과학자
업데이트 빈도	실시간 또는 배치	배치 (Bronze → Silver)	배치 (Silver → Gold)

5.2 레이크하우스 6대 가이드 원칙

성공적인 레이크하우스를 구축하기 위한 6가지 핵심 원칙입니다.

1. **데이터를 제품(Data-as-Products)으로 취급하라:** (메달리온 아키텍처) 데이터를 단순히 쌓아두는 것이 아니라, 신뢰할 수 있는 '제품'으로 가공하여 제공해야 합니다.
2. **데이터 사일로를 제거하고 데이터 이동을 최소화하라:** 데이터를 한곳(레이크)에 보관하고 여러 시스템이 직접 접근하게 하여, 데이터를 복제/이동하면서 발생하는 비효율과 데이터 불일치(Stale data) 문제를 막아야 합니다.
3. **셀프 서비스(Self-Service) 경험으로 가치 창출을 민주화하라:** 적절한 거버넌스 하에 현업 사용자들이 직접 데이터에 접근하고 분석할 수 있는 환경을 제공해야 합니다.
4. **전사적인 데이터 거버넌스 전략을 채택하라:** 데이터 접근 제어(Access Control), 품질 관리, 카탈로그(Unity Catalog) 등 중앙화된 거버넌스 전략이 필수입니다.
5. **개방형 인터페이스와 포맷(Open Formats)을 사용하라:** 특정 벤더에 종속되지 않는 Parquet, Delta Lake 같은 오픈 포맷을 사용하여 유연성을 확보해야 합니다.
6. **확장성, 성능, 비용을 고려하여 구축하라:** 데이터와 사용자가 증가할 것을 대비하여 확장 가능하게 설계하고, 성능과 비용 간의 균형을 맞춰야 합니다.

5.3 데이터 메시 vs. 데이터 패브릭

레이크하우스와 함께 자주 언급되는 두 가지 상위 개념입니다.

주의사항

핵심 차이: 조직 vs. 기술

- **데이터 메시 (Data Mesh): 조직 문화**에 가깝습니다. "중앙 데이터팀이 모든 것을 관리하는 것은 비효율적이니, 데이터 관리를 각 현업 부서(도메인)에 맡기자"는 **분산형 조직/프로세스** 철학입니다.
- **데이터 패브릭 (Data Fabric): 기술 아키텍처**에 가깝습니다. "회사의 데이터가 여러 클라우드와 온프레미스(사내 서버)에 흩어져 있으니, 이를 기술적으로 연결하여 마치 하나의 거대한 '천(Fabric)'처럼 보이게 만들자"는 **통합 아키텍처**입니다.

관점	데이터 메시 (Data Mesh)	데이터 패브릭 (Data Fabric)
핵심 아이디어	분산화 (Decentralization)	통합 및 연결 (Integration)
주요 초점	**조직, 사람, 프로세스**	**기술, 아키텍처, 자동화**
데이터 소유권	각 도메인 팀 (현업 부서)	중앙팀 (IT) 또는 위임
거버넌스	연합 거버넌스 (Federated)	중앙 집중식 거버넌스 (Centralized)
데이터 취급	데이터 = 제품 (Data as a Product)	데이터 = 접근 가능한 자산
적용 대상	조직의 복잡성이 높은 대기업	기술/환경의 복잡성이 높은 기업 (하이브리드/멀티 클라우드)

6 데이터 파이프라인 구축 및 운영

6.1 데이터 수집 도구 (Consolidation Tools)

데이터 레이크로 데이터를 가져오는(수집/적재) 방법들입니다.

- **AutoLoader (오토로더): 지속적, 증분(Incremental) 수집**에 사용됩니다. 클라우드 스토리지(S3, ADLS)의 특정 폴더를 모니터링하다가 **새로운 파일이 도착하면** 자동으로 감지하여 레이크로 적재합니다. 스트리밍 방식이며 'cloudFiles' 포맷을 사용합니다.
- **COPY INTO (카피 인투): 대용량 일괄(Bulk Batch) 수집**에 사용되는 SQL 명령어입니다. 이미 적재된 파일은 건너뛰는 **멱등성(Idempotent)**을 보장하여, 명령을 여러 번 실행해도 데이터 중복이 발생하지 않습니다.
- **DLT (Delta Live Tables):** 차세대 ETL 도구입니다. 선언적(Declarative)으로 파이프라인을 정의하면, 복잡한 처리와 **데이터 품질 검사(Quality Constraints)**를 자동화해줍니다.
- **Local File Upload (로컬 파일 업로드):** Databricks UI를 통해 로컬 PC의 **작은 파일**(예: 테스트용 CSV)을 업로드**하는 기능입니다. (최대 10개 파일, 총 2GB 제한)

6.2 작업 오케스트레이션 (Job Orchestration)

단일 작업이 아닌, 여러 작업(Task)이 연결된 복잡한 파이프라인을 **워크플로우(Workflow)**라고 부릅니다. 이 워크플로우를 관리하고 스케줄링하는 것을 **작업 오케스트레이션(Job Orchestration)**이라고 합니다.

- **의존성 (Dependencies):** 작업 간의 의존 관계를 **DAG (Directed Acyclic Graph)**로 정의합니다.
  - **병렬(Parallel) 수행:** 서로 의존성이 없는 작업들 (예: '고객 데이터 수집'과 '주문 데이터 수집')
  - **순차(Sequential) 수행:** 선행 작업이 끝나야만 시작할 수 있는 작업 (예: '고객 클릭 수집' → '클릭 세션화')
- **주요 기능:**
  - **스케줄링 (Scheduling):** 특정 시간(예: 매일 오전 3시)에 실행되도록 예약 (Cron 표현식 사용).
  - **트리거 (Triggers):** 특정 이벤트(예: 새 파일 도착)에 의해 실행.
  - **재시도/타임아웃 (Retry/Timeout):** 작업 실패 시 자동 재시도, 또는 일정 시간 초과 시 강제 종료.
  - **복구 및 재실행 (Repair and Run):** 파이프라인의 10개 작업 중 8번째 작업만 실패했을 때, 1 7번 작업을 재사용하고 8번부터 다시 실행하여 시간과 비용을 절약합니다.

6.3 작업 성능 문제 해결 (Debugging Job Slowness)

Spark 작업이 유난히 느리게 실행될 때 확인해야 할 4가지 주요 원인 (The 4 S's)입니다.

□ **예제:**

비유로 이해하는 Spark 성능 저하 원인

- **1. 스필 (Spill): (증상)** 메모리(RAM)가 부족하여 데이터를 디스크(Disk)에 임시로 썼다가 다시 읽어오는 현상. **(비유)** 요리하는데 조리대(RAM)가 너무 좁아서 재료를 바닥(Disk)에 내려놓았다가 다시 집어 드는 것. 매우 느리고 비효율적입니다. **(해결)** 더 많은 메모리를 가진 노드를 사용하거나, T-shirt 사이즈(Serverless)를 늘립니다.
- **2. 셔플 (Shuffle): (증상)** 정렬(Sort)이나 조인(Join) 시 여러 노드(컴퓨터) 간에 대규모 데이터 교환이 발생하는 현상. **(비유)** 팀 프로젝트를 하는데, 필요한 자료가 여러 팀원에게 흩어져 있어(Nodes) 서로 카톡으로 자료를 주고받느라(Network) 시간을 다 보내는 것. **(해결)** 데이터 파티셔닝 전략을 최적화하여 노드 간 데이터 이동을 최소화합니다.
- **3. 스큐 / 스트래글러 (Skew / Stragglers): (증상)** 데이터가 불균등하게 분배(Skew)되어 특정 노드(Straggler) 하나에만 일이 몰리는 현상. **(비유)** 팀 프로젝트 5명 중 1명(Straggler)에게만 모든 일이 몰리고, 나머지 4명은 일찍 끝나서 놀고 있는 것. 전체 프로젝트는 그 1명이 끝나야 끝납니다. **(해결)** 파티션 키를 변경하여 데이터가 고르게 분배되도록 합니다. (예: '국가' 대신 '국가+도시')
- **4. 작은 파일 문제 (Small Files): (증상)** 수백만 개의 아주 작은 파일들을 처리할 때, 실제 데이터 처리 시간보다 파일을 '열고/닫는' I/O 오버헤드가 더 커지는 현상. **(비유)** 책 1권(1GB)을 읽는데, 1권짜리 책이 아니라 1페이지짜리 100만 개의 파일로 쪼개져 있어서, 파일을 100만 번 '열고-읽고-닫는' 것. **(해결)** 'OPTIMIZE' 명령 등으로 작은 파일들을 적절한 크기의 큰 파일로 병합(Compaction)합니다.

6.4 모니터링 (Monitoring)

- **전통적 방식 (Ganglia Metrics):** 클러스터의 CPU, 메모리, 네트워크 사용량을 시각적으로 모니터링합니다. (예: Spill이 발생하면 메모리 사용량이 100%에 육박하고 디스크 I/O가 급증합니다.)
- **서버리스 (Serverless):** Ganglia 같은 세부적인 인프라 모니터링 대신, **T-shirt Sizing** (예: Small, Medium, Large)을 통해 워크로드에 맞는 리소스를 선택합니다. 관리는 단순화되지만, 세밀한 메모리 튜닝은 어려울 수 있습니다.

7 [실습] 델타 레이크 핵심 기능 (Lab: Delta Lake Features)

7.1 Parquet에서 Delta로 변환

기존 Parquet 데이터를 Delta Lake 형식으로 변환하는 것은 매우 간단합니다. 'CREATE TABLE ... USING delta AS ...' 구문을 사용합니다.

```
-- Lending Club 데이터셋(Parquet)을 읽어와
-- 'loans_by_state_delta라는' Delta Lake 테이블로생성
CREATE TABLE IF NOT EXISTS loans_by_state_delta
USING delta
LOCATION '/path/to/delta/table'
AS SELECT * FROM parquet_table_view;
```

Listing 1: Parquet 파일을 Delta Lake 테이블로 변환

테이블이 Delta 형식인지 확인하려면 'DESCRIBE DETAIL'을 사용합니다.

```
DESCRIBE DETAIL loans_by_state_delta;
```

Listing 2: 테이블 상세 정보 확인 (Delta 확인)

7.2 동시 작업 (Streaming + Batch)

Delta Lake의 ACID 트랜잭션 덕분에, 한쪽에서는 데이터를 스트리밍으로 읽는('readStream') 동시에 다른 쪽에서는 배치(Batch)로 데이터를 삽입('INSERT')할 수 있습니다.

```
# streaming_query = (
#   spark.readStream
#     .format("delta")
#     .load("/path/to/delta/table")
#     .writeStream
#     ...
#     .start()
# )
```

Listing 3: Delta 테이블에서 스트리밍 읽기 시작 (가상 코드)

```
-- Iowa (IA) 주에건의 450 대출데이터를번  6 반복삽입
INSERT INTO loans_by_state_delta VALUES ('IA', 450);
INSERT INTO loans_by_state_delta VALUES ('IA', 450);
... (6 times)
```

Listing 4: 동시에 배치 데이터 삽입 (Iowa 데이터 추가)

스트리밍 쿼리는 중단 없이 이 새로운 삽입을 감지하여 처리합니다.

7.3 DML (DELETE, UPDATE)

전통적인 Parquet 파일에서는 불가능했던 행(Row) 단위 데이터 삭제가 가능합니다.

```sql
-- Iowa (IA) 주의모든데이터를삭제
DELETE FROM loans_by_state_delta WHERE state = 'IA';
```

Listing 5: Delta 테이블에서 특정 데이터 삭제

이 명령은 일반 Parquet 테이블에서는 실패하지만, Delta Lake에서는 성공합니다.

7.4 MERGE (Upsert)

'MERGE'는 'Upsert' (Update + Insert) 작업을 원자적으로 수행합니다. 새로운 데이터(Source)를 기존 테이블(Target)과 비교하여,

- 조건이 일치하면 ('WHEN MATCHED') → **UPDATE**
- 조건이 일치하지 않으면 ('WHEN NOT MATCHED') → **INSERT**

```sql
-- 'new_updates_table' (Source)의 데이터를
-- 'loans_by_state_delta' (Target)에 병합
MERGE INTO loans_by_state_delta AS target
USING new_updates_table AS source
ON target.state = source.state -- state 컬럼을기준으로비교

WHEN MATCHED THEN
  -- 주(state)가 이미존재하면를  count 업데이트
  UPDATE SET target.count = source.new_count

WHEN NOT MATCHED THEN
  -- 주(state)가 존재하지않으면새로삽입
  INSERT (state, count) VALUES (source.state, source.new_count);
```

Listing 6: MERGE를 사용한 Upsert 작업

7.5 스키마 진화 (Schema Evolution)

기본적으로 Delta Lake는 기존 테이블 스키마와 다른 데이터가 들어오면 작업을 실패시킵니다 (Schema Enforcement).

주의사항

상황: 기존 테이블은 '(state, count)' 2개 컬럼인데, '(state, count, amount)' 3개 컬럼의 새 데이터를 추가(append)하려고 합니다.
기본 동작 (실패): 'spark.write.format("delta").mode("append").save(...)' → **오류 발생!** (Schema mismatch)
해결 (스키마 진화 옵션): 데이터프레임 쓰기 옵션에 'option("mergeSchema", "true")'를 추가하면, Delta Lake가 기존 스키마에 'amount' 컬럼을 자동으로 추가하여 작업을 성공시킵니다.

7.6 타임 트래블 (Time Travel)

Delta Lake는 모든 변경 이력(로그)을 보관합니다. 'DESCRIBE HISTORY' 명령으로 이력을 볼 수 있습니다.

```
DESCRIBE HISTORY loans_by_state_delta;
```

Listing 7: 테이블 변경 이력 조회

이 버전을 사용하여 과거 데이터를 조회할 수 있습니다.

```
-- 가장처음버전 ( 0)의 데이터조회  (Iowa 데이터가없던시점  )
SELECT * FROM loans_by_state_delta VERSION AS OF 0;

-- 가MERGE 완료된시점버전 ( 9)의 데이터조회
SELECT * FROM loans_by_state_delta VERSION AS OF 9;
```

Listing 8: 특정 버전(Version)으로 과거 데이터 조회

버전 번호 대신 타임스탬프(시간)를 사용할 수도 있습니다.

```
-- 특정날짜시간 / 기준의데이터조회
SELECT * FROM loans_by_state_delta
TIMESTAMP AS OF '2025-10-26 03:00:00';
```

Listing 9: 특정 시간(Timestamp)으로 과거 데이터 조회

8 [실습] Databricks 워크플로우 (Lab: Workflows)

Databricks Jobs UI를 사용하여 여러 노트북을 연결하는 파이프라인(워크플로우)을 만드는 방법입니다.

1. 작업(Job) 생성: Databricks UI의 'Jobs' 탭에서 새 작업을 생성합니다.

2. 태스크(Task) 추가: 작업 내부에 여러 태스크를 추가합니다. 각 태스크는 노트북, SQL 스크립트, Python 파일 등이 될 수 있습니다.

3. 의존성(Dependency) 설정: 태스크 간의 실행 순서를 정의합니다.

- **Task B**가 **Task A**에 의존하도록 설정하면, Task A가 성공적으로 완료되어야만 Task B가 실행됩니다.
- 의존성을 설정하지 않으면 두 태스크는 병렬로 실행됩니다.

4. 파라미터(Parameter) 전달: 워크플로우 실행 시 동적으로 값을 전달할 수 있습니다.

- **작업 파라미터 (Job Parameters):** 워크플로우 전체에서 사용되는 전역 변수.
- **태스크 파라미터 (Task Parameters):** 특정 태스크에만 전달되는 지역 변수.

노트북 내에서 이 파라미터를 받으려면 'dbutils.widgets.get()'을 사용합니다.

```
# 'param1'이라는' 이름으로전달된태스크파라미터값을가져옴
param_value = dbutils.widgets.get("param1")

# 'job_param1'이라는' 이름으로전달된작업파라미터값을가져옴
job_param_value = dbutils.widgets.get("job_param1")

print(f"태스크 파라미터값 : {param_value}")
print(f"작업 파라미터값 : {job_param_value}")
```

Listing 10: 노트북에서 워크플로우 파라미터 받기

5. 스케줄링: 'Cron' 구문을 사용하여 "매주 월요일 오전 9시"처럼 주기적으로 작업을 실행하도록 예약할 수 있습니다.

9 빠르게 훑어보기 (1페이지 요약)

데이터 저장소의 진화

1. 데이터 사일로 (Silo):
고립된 '데이터 섬'. 부서 간 데이터 공유 불가. **(문제)** ↓ **2. 데이터 레이크 (Lake):**
모든 데이터를 원본(Raw) 그대로 한곳에 저장하는 '호수'. (Schema on Read) **(해결책)** ↓ **3. 데이터 스웜프 (Swamp):**
관리가 안 되어 아무도 못 쓰는 '데이터 늪'. (거버넌스 실패) **(위험)**

웨어하우스 vs. 레이크하우스

4. 데이터 웨어하우스 (Warehouse):
정제된 데이터만 저장하는 '백화점'. (Schema on Write) BI에 빠르지만 비싸고 비유연. ↓
5. 데이터 레이크하우스 (Lakehouse):
레이크(유연성/저비용)와 웨어하우스(신뢰성/성능)의 장점을 합친 **통합 플랫폼**.

레이크하우스 핵심 기술 및 방법론

Delta Lake (델타 레이크):
Parquet 파일 위에서 **ACID 트랜잭션, 타임 트래블, MERGE/UPDATE/DELETE**를 가능하게 하는 '프로토콜'. 레이크를 신뢰할 수 있게 만듦.
Medallion Architecture (메달리온 아키텍처):
데이터를 3단계로 정제하는 파이프라인.

- **Bronze:** 원본 (Raw)
- **Silver:** 정제 (Cleaned)
- **Gold:** 집계 (Aggregated)

파이프라인 성능 저하 4대 원인

- **Spill (스필):** 메모리 부족 → 디스크 사용 (느림)
- **Shuffle (셔플):** 노드 간 불필요한 데이터 교환 (네트워크 병목)
- **Skew (스큐):** 데이터 불균형 → 특정 노드만 바쁨 (작업 지연)
- **Small Files (작은 파일):** 너무 많은 작은 파일 → I/O 오버헤드