CDC 기반의 스트리밍 ETL 파이프라인 구축기

발표자: KAFKA KRU 이상헌

목차

1. Introduction – 발표자 소개

- 소개 : 제조업 기반 회사에서 데이터 레이크 서비스 개발 및 운영 업무를 맡고 있다.
  실시간 CDC(Change Data Capture) 데이터 파이프라인 개발을 비롯하여 카프카 서비스에 관심이 많다. 현재 카프카 한국 사용자 모임 운영진으로 활동하고 있다.
  지식 전달에 관심이 많아서, 특허나 논문 쓰는 것을 좋아하려고 노력하고 있고,
  최근에는 Kafka Connect 도서를 번역했다.

1. Introduction – 주제 소개

밋업 주제 : CDC 기반의 스트리밍 ETL 파이프라인 구축

- 과제 진행 목적 :

1. 실시간 데이터를 Data Lake에 넣어서 사용자가 데이터를 빨리 활용할 수 있게!
2. 원천 DB의 부하를 줄이기 위해!

1. Introduction – 주제 소개

밋업 주제 : CDC 기반의 스트리밍 ETL 파이프라인 구축

(On-premise 환경, 타 부서 DB에 접근하는 조건에서) CDC 기반의 스트리밍 ETL 파이프라인 구축

- 과제 진행 목적 :

1. 실시간 데이터를 Data Lake에 넣어서 사용자가 데이터를 빨리 활용할 수 있게!
2. 원천 DB의 부하를 줄이기 위해!

2. 개요 – About CDC

CDC = 데이터베이스의 변경 사항을 추적, 원천 DB와 똑같게 복제!

<Query-Based CDC = batch ETL>

a. 주기적으로 테이블을 SELECT 쿼리로 조회하여 변경된 데이터를 확인
b. 일반적으로 “last_modified” 컬럼을 기반으로 WHERE 조건을 걸어 데이터 추출
c. 장점 : JDBC 기반으로 DB에 연결해서 쿼리. 초기 셋업이 간단
d. 단점 : 원천 DB에 부하 발생. 데이터 누락 가능성 높음. Data latency 발생

2. 개요 – About CDC

CDC = 데이터베이스의 변경 사항을 추적, 원천 DB와 똑같게 복제!

★ **<Log-Based CDC>**

a. 데이터베이스의 트랜잭션 로그(bin log, redo log 등)를 직접 읽어서 변경 사항을 추출
b. 장점 : 원천 DB에 부하를 주지 않음. 트랜잭션 순서를 보장. 삭제나 업데이트 반영 가능, 실시간 연계 가능
c. 단점 : DB에서 동작하는 process가 있고, 로그에 접근할 수 있어야 한다. 파이프라인을 구성하는 프레임워크가 많아진다.

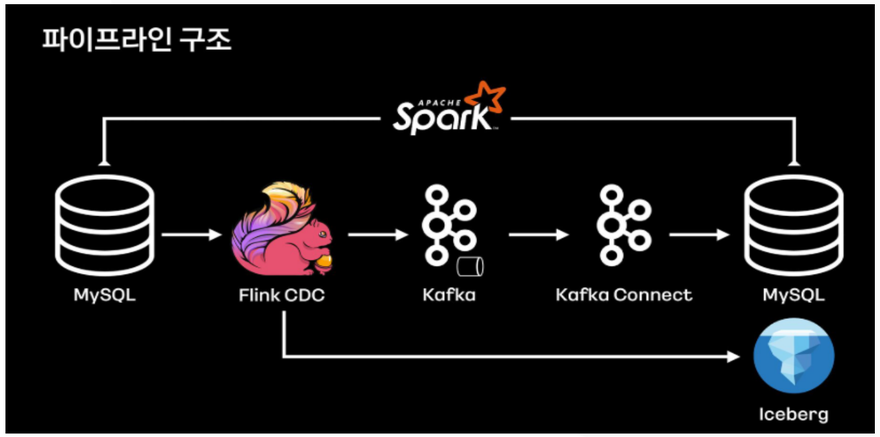

```
#051024 17:24:13 server id 1  end_log_pos 98
# Position  Timestamp   Type   Master ID        Size      Master Pos    Flags
# 00000004 9d fc 5c 43   0f   01 00 00 00   5e 00 00 00   62 00 00 00   00 00
# 00000017 04 00 35 2e 30 2e 31 35  2d 64 65 62 75 67 2d 6c |..5.0.15.debug.l|
# 00000027 6f 67 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |og..............|
# 00000037 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |................|
# 00000047 00 00 00 00 9d fc 5c 43  13 38 0d 00 08 00 12 00 |.......C.8......|
# 00000057 04 04 04 04 12 00 00 4b  00 04 1a               |.......K........|
```

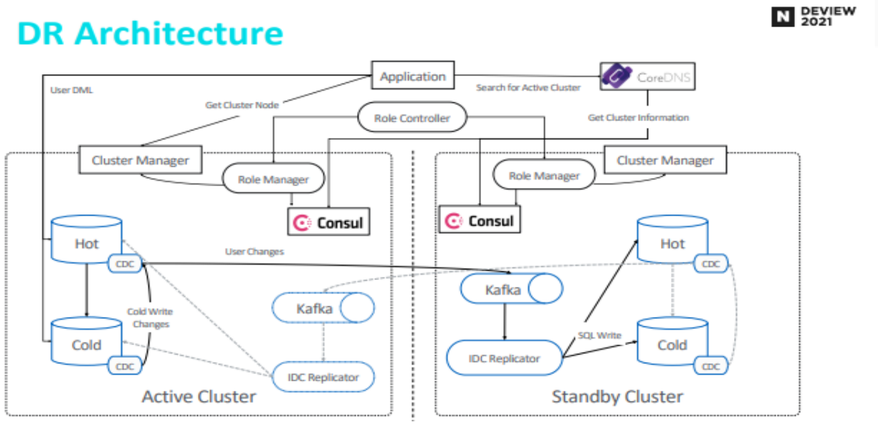
2. 개요 – 타사 case

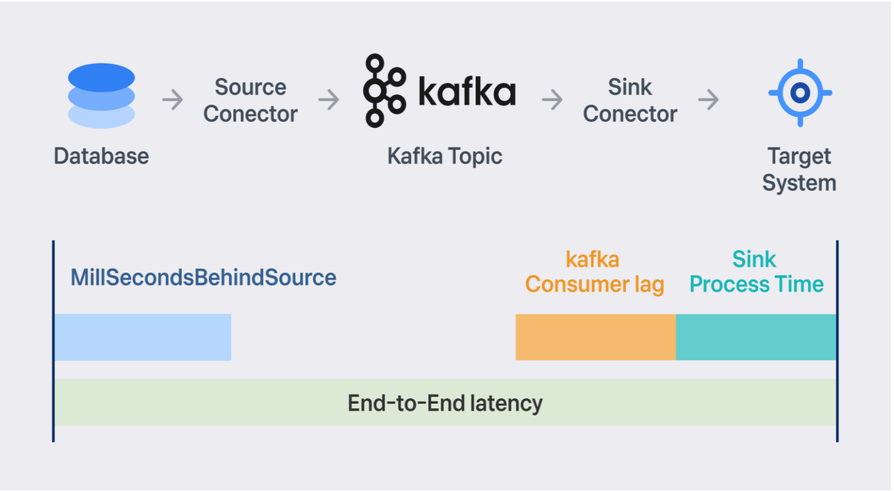
1. Kakao
- https://www.youtube.com/watch?v=PeNHKxadNos
- Flink CDC (Debezium)
- 대용량 데이터베이스 동기화

2. Naver pay
- https://deview.kr/2021/sessions/495
- CDC 기반 Materialized view, Data Tiering, 재해 복구

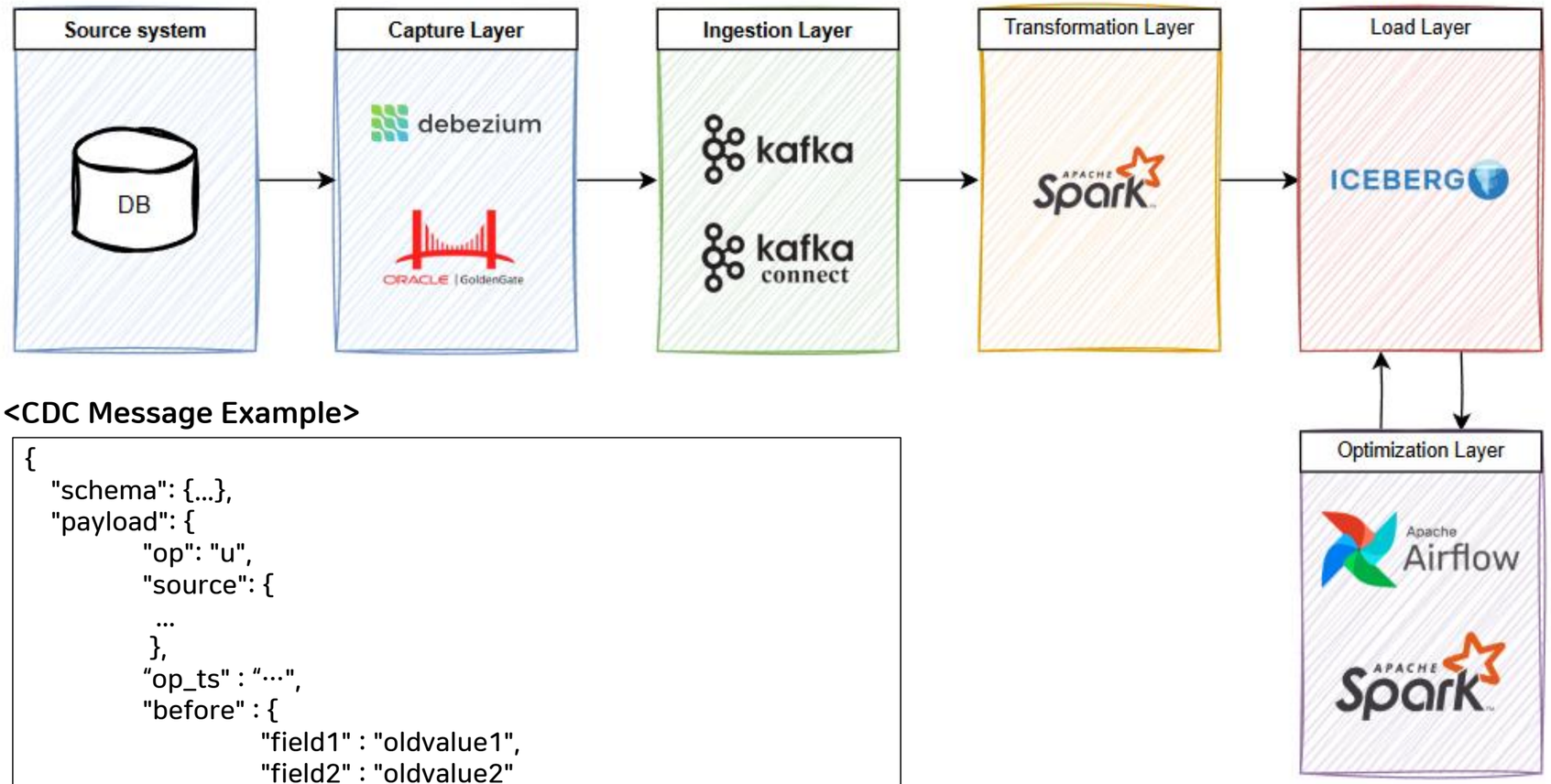
3. Toss
- https://toss.tech/article/cdc_pipeline
- Debezium CDC, 운영 관점 지표 관리







3. CDC 파이프라인 아키텍처 – 전체 아키텍처 소개



<CDC Message Example>

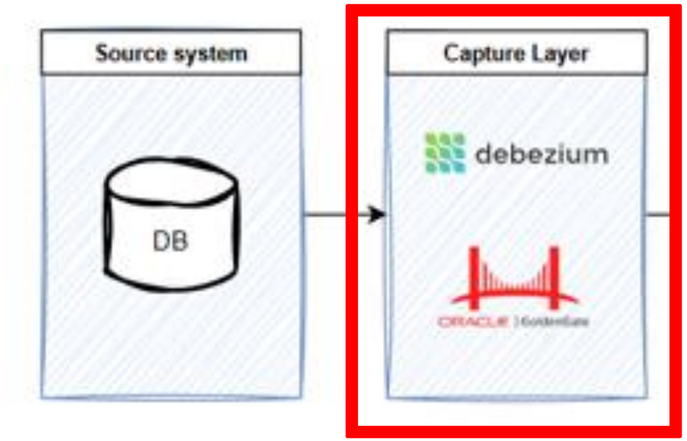
```
{
   "schema": {...},
   "payload": {
              "op": "u",
              "source": {
                ...
               },
              "op_ts" : "…",
              "before" : {
                            "field1" : "oldvalue1",
                            "field2" : "oldvalue2"
                            },
               "after" : {
                             "field1" : "newvalue1",
                             "field2" : "newvalue2"
                             }
              }
}
```

3. CDC 파이프라인 아키텍처 – 요구 사항 및 고려 사항

<구축 당시 상황>

1. ETL Latency 최소화 요구사항 증가
2. 초대규모 트랜잭션 처리 필요
3. 적은 개발 인력
4. Source는 Only Oracle DB
5. 여유 있는 Spark Cluster 자원
6. 운영 DB 접근의 어려움
7. Iceberg 도입 가능성

4. Layer 별 세부 사항 – Capture Layer



1. CDC Framework
   A. Oracle Golden Gate + Oracle Golden Gate for bigdata
   B. Debezium
      a. Kafka Connect / Flink CDC
      b. Oracle의 경우 XStream API / LogMiner

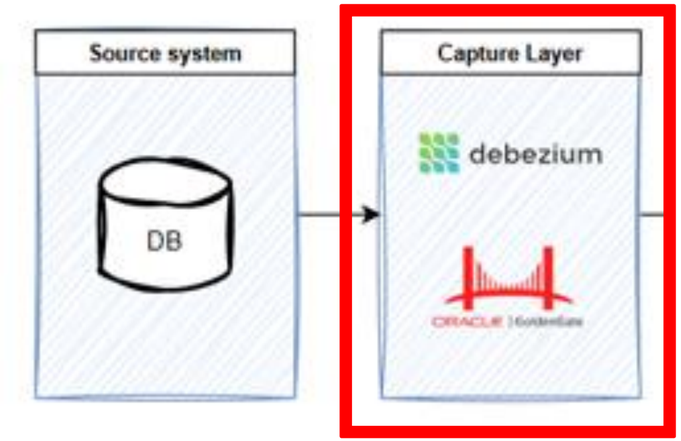
2. 계정에 log 접근 권한 부여 + 설정 값 변경(supplemental logging 활성화)

3. 로그 보존 주기 설정 + 관리

4. 문제 발생 시 트러블 슈팅
   A. 주로 disk full 문제나 DB 작업 시 영향에 대한 대응

5. 최초 셋업 시, 영향도 평가 및 DB 옵션에 따른 Framework 선택 고려
   A. Table compression method 고려 (EX : COMPRESS FOR OLTP)
   B. DB 구성(DR인지? 그 DR DB는 어떤 방식으로 구성되어 있는지? 등)

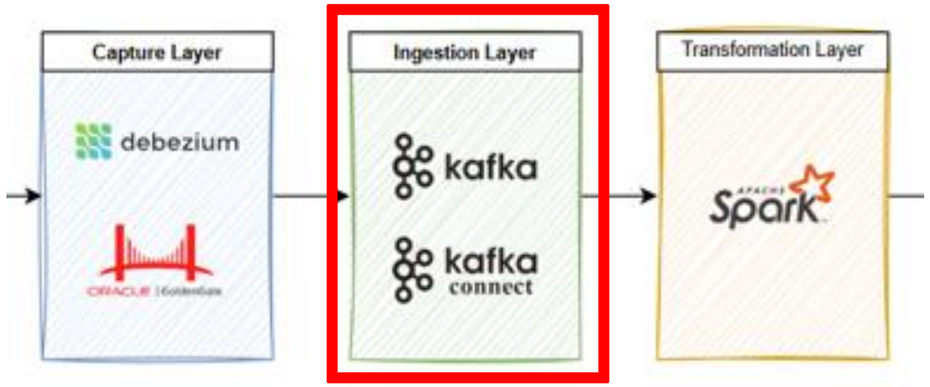
4. Layer 별 세부 사항 – Capture Layer



6. (Oracle DB의 경우) Debezium 성능 고려
   A. 특히 Oracle DB의 경우,
      LogMiner 방식을 사용하냐?
      Xstreams 방식을 사용하냐? 에 따라 크게 성능 차이가 난다.
   B. LogMiner 방식
      a. Oracle에서 기본 제공하는 LogMiner를 통해 redo log를 조회 (SQL)
   C. XStream 방식
      a. GoldenGate에서 사용하는 내부 API (XStream) 활용

항목	LogMiner	XStream
접근 방식	SQL 기반 redo log 조회	Binary 파일 스트리밍 (GoldenGate)
라이센스	무료	유료 (GoldenGate 라이선스 필요)
성능	낮음 (SQL 파싱)	높음 (스트리밍)
설정 난이도	낮음	높음
대용량 처리	어려움	가능

4. Layer 별 세부 사항 – Ingestion Layer

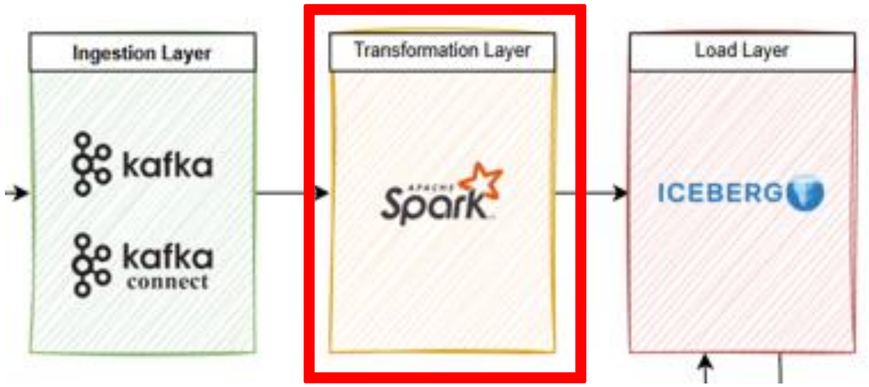


1. Kafka
   A. CDC 메시지를 Kafka broker로 전송
      a. 이때는 Kafka Connect 이용

   B. 테이블 별 토픽 생성 및 관리

   C. 토픽별 파티션 키 선정 및 카프카 파티션 관리
      a. 파티션 증설 시, 기존 파티션은 그대로 유지한 채로 증설하는 것이 필요함.
      b. 또한 파티션 증설 시, consume하는 application이 있다면 기존에 있던 파티션의 데이터를 전부 consume 했는지 확인.
      c. 전부 consume하지 못했다면, 별도 후처리 로직을 적용시켜야 한다.

4. Layer 별 세부 사항 – Transformation Layer



1. Spark Streaming
   A. CDC 파이프라인에서 가장 핵심
   B. Kafka 로 유입된 메시지들을 정형화(Struct)시킨 후 Merge Into 구문을 사용해서 적재
   C. 모든 트랜잭션의 순서를 보장하게 해서 그대로 구현시키는 로직을 넣는 것이 중요함.
      a. What's the Difference? Incremental Processing with Change Queries in Snowflake 논문 참고

2. Transformation 과정
   A. Json 포멧의 cdc 메시지를 정형 데이터로 저장할 수 있도록 변환
   B. 각종 전처리(중복 값 제거, op 시간순 정렬 등)
   C. Merge Into 문 작성

4. Layer 별 세부 사항 – Transformation Layer : Merge Into 문 작성

<MERGE INTO SQL>

```
MERGE INTO target USING source
ON source.key = target.key
WHEN MATCHED AND source.action = "UPDATE" THEN UPDATE SET *
WHEN MATCHED AND source.action = "DELETE" THEN DELETE
WHEN NOT MATCHED AND source.action = "INSERT" THEN INSERT *
```

1. source 테이블을 사용해서 target 테이블로 적재.
2. 키 매칭 여부(source.key = target.key)로 분기
3. 매칭된 경우, source.action을 확인
   - source.action 이 "UPDATE"이면 UPDATE SET *
   - source.action 이 “DELETE”이면 DELETE
4. 매칭되지 않은 경우, source.action이 "INSERT"인지 확인
   - "INSERT"이면 INSERT *

4. Layer 별 세부 사항 – Transformation Layer : Merge Into 문 작성

<MERGE INTO SQL>

```
MERGE INTO target USING source
ON source.key = target.key
WHEN MATCHED AND source.action = "UPDATE" THEN UPDATE SET *
WHEN MATCHED AND source.action = "DELETE" THEN DELETE
WHEN NOT MATCHED AND source.action = "INSERT" THEN INSERT *
```

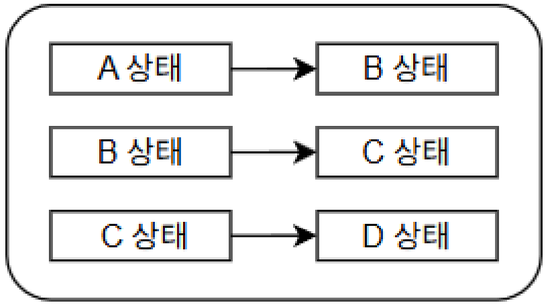
-> 잘 될까?

4. Layer 별 세부 사항 – Transformation Layer : Merge Into 문 작성

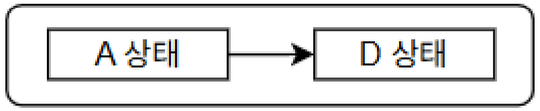
* Spark Streaming의 micro batch가 n분으로 가정

필요 없는 데이터가 쌓일 수도 있는 상황 예시

<상황 설정>



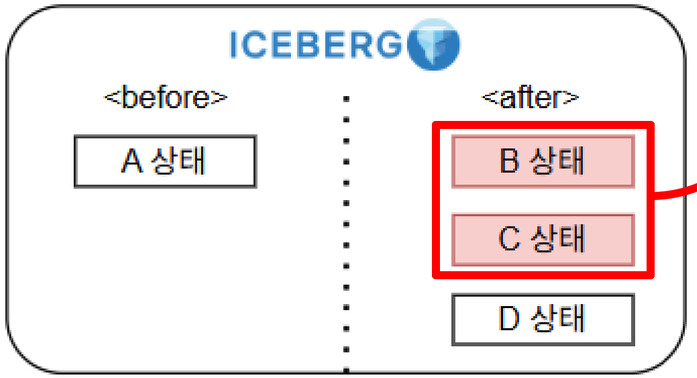
<최종 상황>



* 상태 : 테이블 각 Row의 버전(update라 가정)

<기대한 상황>

<실제 상황>



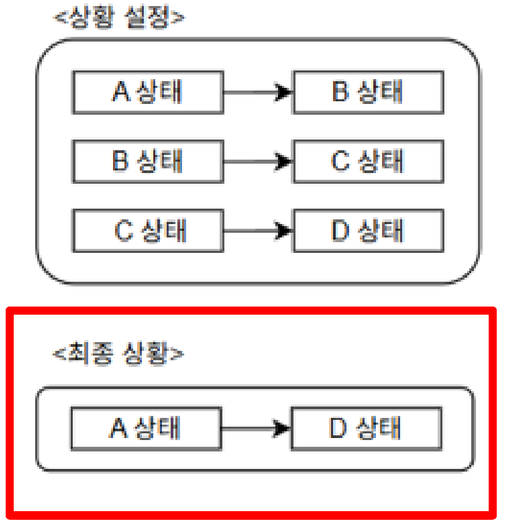
필요 없는 데이터(예전 데이터)

4. Layer 별 세부 사항 – Transformation Layer : Merge Into 문 작성

<Minimum-delta changes>

- 두 시점 간의 차이를 나타내기 위한 최소한의 데이터 변경을 의미하며, 중복된 수정 사항을 통합하고, 부분적인 업데이트 대신 전체 행을 처리하여 효율성을 높이는 방법

- 소스 테이블에 같은 기본 키를 가진 여러 행이 있을 경우 MERGE의 동작이 정의되지 않기 때문에, 최소 델타 변경을 사용하는 것이 더 유리하다.



4. Layer 별 세부 사항 – Transformation Layer : Merge Into 문 작성

<MERGE INTO SQL - Minimum-delta changes 반영>

```
MERGE INTO target USING
(SELECT key,
        MAX_BY(action, event_time) as action,
        MAX_BY(value1, event_time) as value1,
        MAX_BY(value2, event_time) as value2,
        MAX_BY(value3, event_time) as value3
 GROUP BY 1
) source
ON source.key = target.key
WHEN MATCHED AND source.action = "UPDATE" THEN UPDATE SET *
WHEN MATCHED AND source.action = "DELETE" THEN DELETE
WHEN NOT MATCHED AND source.action = "INSERT" THEN INSERT *
```

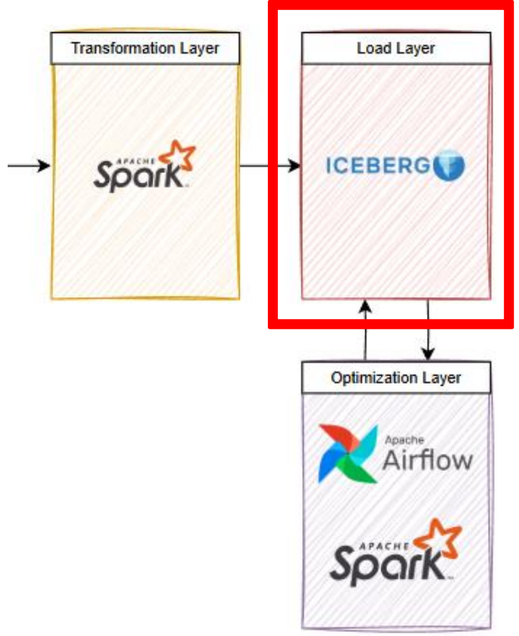
4. Layer 별 세부 사항 – Transformation Layer : Merge Into 문 작성

<최종 – MATCH 되면서 INSERT인 경우와 NOT MATCH 되면서 UPDATE인 경우 반영>

```
MERGE INTO target USING
 (SELECT key,
        MAX_BY(action, event_time) as action,
        MAX_BY(value1, event_time) as value1,
        MAX_BY(value2, event_time) as value2,
        MAX_BY(value3, event_time) as value3
  GROUP BY 1
 ) source
ON source.key = target.key
WHEN MATCHED AND source.action = "UPDATE" THEN UPDATE SET *
WHEN MATCHED AND source.action = "DELETE" THEN DELETE
WHEN MATCHED AND source.action = "INSERT" THEN UPDATE SET *
WHEN NOT MATCHED AND source.action in ("INSERT", "UPDATE") THEN INSERT *
```

- 참고 : https://medium.com/@flyws1993/traps-in-applying-cdc-changes-into-a-streaming-data-warehouse-cd43e53ff816

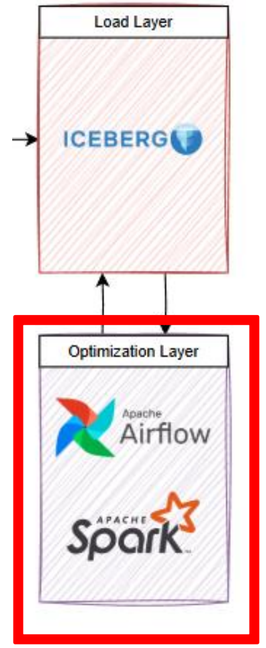
4. Layer 별 세부 사항 – Load Layer



1. Iceberg
   A. 각 테이블에 대한 파티션 키 선정 -> ‘매우 중요’
   B. 무수히 생성되는 small file 관리 (merge)
   C. 만료된 manifest, snapshot file 삭제
   D. rewrite 시 정규 CDC job도 동시에 write될 때, 정규 CDC job에 에러가 발생할 수 있으므로 주의.
   E. 본인이 사용하는 processing engine과 호환되는 버전인지 check! (특히 Upsert 지원 여부)

- 참고하면 좋은 사이트 : https://toss.tech/article/datalake-iceberg

4. Layer 별 세부 사항 – Optimization Layer



1. Spark + Airflow
   A. Iceberg를 유지 보수하기 위한 프레임워크로는 Spark를 채택함
      a. rewrite_data_files :
         작은 데이터 파일들을 병합하여 읽기 성능을 향상시키고 파일 수를 줄임
      b. rewrite_manifests :
         매니페스트 파일을 압축 및 최적화하여 메타데이터 처리 속도를 개선
      c. expire_snapshots :
         오래된 스냅샷을 제거하여 메타데이터 크기를 줄이고 저장 공간을 확보
      d. remove_orphan_file :
         메타 데이터와 연결되지 않은 파일들을 삭제하여 저장 공간을 확보
   B. 주기적으로 전체 테이블에 대한 Spark Procedures call.
      a. 세부 내용은 Iceberg doc 참고.

- 참고하면 좋은 사이트 : https://iceberg.apache.org/docs/nightly/spark-procedures/

5. 어려웠던 점 – 구축 시 고려할만한 점

1. 원천 DB에 application이 설치되고 process가 실행되어야 해서 DBA와 협의 + 세팅 시 DB 지식 필요
   - CDC 장애 시 직접 DB 서버 접속 후 명령어 수행해야 해서 부담... (내 서버가 아니니)
   - 원천 DB 세팅에 따라 선택할 수 있는 옵션이 좁아짐.

2. 관리 포인트가 많은 파이프라인
   - CDC, Kafka, Spark, Iceberg, Airflow 등등 사용하는 프레임워크가 많고 익숙하지 않은 프레임워크 (EX: Iceberg)는 특히 신경 쓸 부분이 많다.
   - 만약 CDC 프로세스가 죽게 되면, downstream에 영향을 크게 미쳐서, Query-based CDC를 통해 소급해야 한다.
   - 특히 스토리지 관리도 필요하다(small file이 다수 생성됨)
   - 파티션 컬럼 선정을 신경써서 쿼리 엔진이 탐색하는 범위를 최대한 줄여줘야 한다.

3. 트랜잭션 양이 많은 테이블은 Kafka topic의 파티션을 늘려줘야 한다.
   - 기존에 데이터가 쌓여 있는 파티션 및 브로커 id의 변화가 없도록 신경 써준다.
   - 아무래도 파티션 증설 시, consuming을 멈추는 편이 안전하다.
   - 파티션 키가 명확한 테이블만 증설 가능하다. (Merge Into 로직 망가져 버림)

6. Q&A

Q&A

감사합니다.