



Snowflake 데이터 이관(브랜드 점주 앱)

☰ 태그	AWS	ETL	데이터엔지니어	서클플랫폼
📅 날짜	@2024년 11월 1일			

요구사항

AS-IS

- 현황 1
- 문제점 1-1
- 문제점 1-2
- 문제점 1-3
- 현황 2
- 문제점 2-1
- 문제점 2-2

TO-BE

- 인프라
- 비용관리
- 유지보수
- 소감
- 앞으로 계획

요구사항

- 점주앱(가칭)이라는 여러브랜드를 통합으로 관리할 수 있는 앱을 개발하는 프로젝트가 시작되었고 해당 앱은 여러 브랜드의 모든 데이터를 조회해야하는 환경을 구축해야함
 - 각 점주별로 준실시간으로 주문에 대한 알림을 보내줘야함
- 고객사의 영업담당자가 요구하는 복잡한 대시보드구성시 Query가 너무 느려져 이에 대한 컴플레인을 해결 해야함

AS-IS

- 기존의 DB 구조는 다음과 같았습니다.
APP \leftrightarrow AWS AuroraDB(운영DB) \rightarrow RDS(분석용)

현황 1

운영 DB \rightarrow 분석용 DB로 이관하는 과정은 매일 새벽 최근 3일치의 데이터를
 DELETE 후 INSERT 하는 방식이었습니다. (주문건에 대한 환불, 취소를 최신화하기 위해 3일치를 진행)

문제점 1-1

 해당 과정은 최근 3일치 주문건에 대한 구매확정을 확신할 수 없어 고객사가 보는 통계데이터에 반영할 수 없던 문제가 있었습니다.

문제점 1-2

 또한 점점 주문이 많아지고 그에 따라 고객 CS또한 많아져 주문이 많이 몰리는 시간대(출퇴근시간대)에 운영 DB에 직접 SELECT 하는 작업이 부담이 되었으며, 심지어 늦어지는 경우도 있었습니다.

문제점 1-3

 주문이 많아지면서 주문데이터를 분석하거나 통계데이터를 추출하는 쿼리가 너무 오래걸리고 심지어 타임아웃까지 발생하기도 했습니다.

현황 2

 점주앱(브랜드통합앱)을 구성하는데 있어 5개가 넘는 RDS에 대한 연결관리

문제점 2-1

 브랜드를 추가할 때마다 엄청나게 늘어나는 공수

문제점 2-2

 개인화 마케팅 서비스를 제공하기위해 데이터를 통합으로 분석할 수 없는 문제

TO-BE

- 회사는 이러한 문제를 해결하기 위해 Snowflake를 도입하기로 하였으며, 데이터 이관 작업을 직접 진행하게 되었습니다.

인프라



Aurora(RDS) → S3(binlog) → snowflake

- 실시간 데이터 조회



주문취소건을 반영하기위해 **3일치씩 Delete & Insert**로 직대신 아래와 같은 방식으로 진행하였습니다.

1. Aurora(mysql)DB 에 직접조회하는 대신 Binlog 를 직접 S3 에 저장 (실시간)
2. AWS S3 의 데이터를 **Snowpipe** 를 활용하여 **STG_SCHEMA** 에 모두 적재합니다.



STG_SCHEMA 는 스테이지 단계로 ROW 단위로 **Op** 및 **updatedAt** 등이 기록되며 같은 PK가 중복으로 존재합니다.

3. STG_SCHEMA 에 쌓인 binlog 테이블을 **RAW_SCHEMA** 에 **TASK** 와 Snowflake 의 SQL 문법에 맞는 **merge into Query** 를 사용하여 **Insert** 와 **Update** 를 진행합니다.



STG(스테이지), RAW(운영DB와 일치)는 각 schema 마다 하나의 브랜드입니다.

- 통합 분석을 위한 CI_SCHEMA에는 모든 브랜드의 주문, 고객 데이터가 존재하는 DW 단계입니다.

RAW → CI 단계에서 각 브랜드에 맞는 BRAND_ID를 추가하여 구분하였으며
→ 통합분석과 브랜드조합의 분석이 가능해집니다.

- RAW_SCHEMA의 CI_SCHEMA로 Merge를 수행합니다.

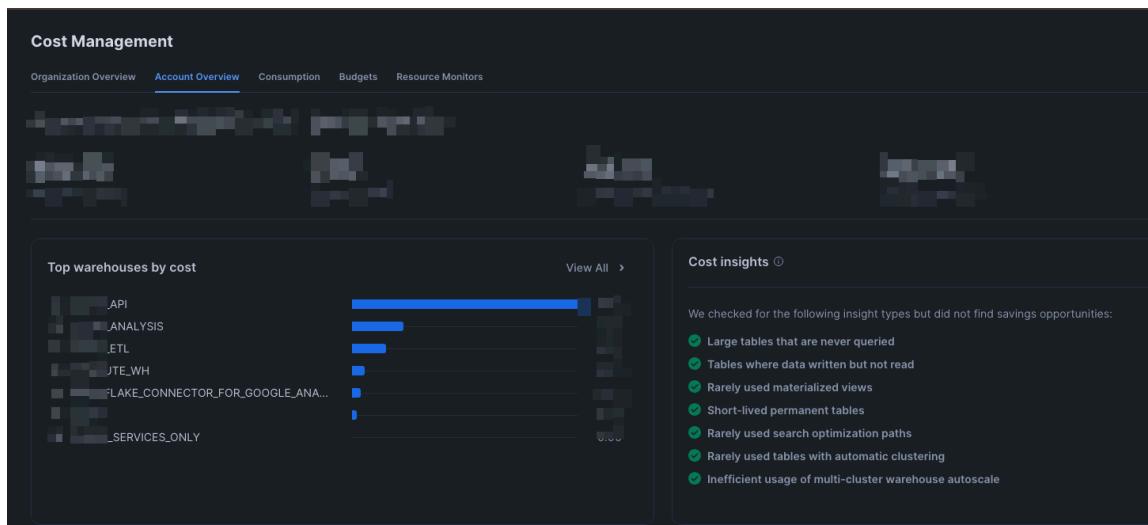
RAW_SCHEMA의 테이블에 스트림을 생성하면 소비할 수 있는 논리적 테이블이 생성되며 RAW → CI의 merge는 RAW 테이블이 아닌 Stream → CI를 수행합니다.

- STREAM은 binlog와 다르게 INSERT와 DELETE만 존재하여 STG → RAW와의 Merge Query는 다르게 구성해야합니다.

비용관리

아직 Snowflake에 대해 많이 공부하고 있어 계속해서 고도화하는 중입니다.

- warehouse 분리하여 각 Warehouse별로 비용을 쉽게 산정할 수 있도록 구성했습니다. (ETL, API, 분석 전용)



Warehouse별로 사용량 측정 가능

API Warehouse 의 Query 확인

SQL TEXT	QUERY ID	STATUS	USER	DURATION	STARTED	ROWS
SELECT ...	01b8b129-0000-a3c...	Success	[REDACTED]	31ms	11/29/2024, 6:13:04 PM	126
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	27ms	11/29/2024, 6:12:43 PM	5
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	31ms	11/29/2024, 6:12:43 PM	5
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	21ms	11/29/2024, 6:12:43 PM	5
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	28ms	11/29/2024, 6:12:43 PM	5
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	26ms	11/29/2024, 6:12:43 PM	4
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	81ms	11/29/2024, 6:12:43 PM	6
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	68ms	11/29/2024, 6:12:43 PM	6
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	19ms	11/29/2024, 6:12:43 PM	4
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	26ms	11/29/2024, 6:12:43 PM	3
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	18ms	11/29/2024, 6:12:43 PM	181
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	18ms	11/29/2024, 6:12:43 PM	181
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	20ms	11/29/2024, 6:12:43 PM	3
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	41ms	11/29/2024, 6:12:42 PM	6
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	77ms	11/29/2024, 6:12:42 PM	1
SELECT ...	01b8b128-0000-a3c...	Success	[REDACTED]	26ms	11/29/2024, 6:12:42 PM	25

API Warehouse 의 사용 쿼리 확인



ETL, API, 분석 전용 Warehouse 를 구분하여 각 warehouse 가 자주 사용하는 쿼리를 보면서

파티셔닝, 오토 클러스터링을 어떤 컬럼에 적용할 지 볼 수 있도록 했습니다.

- 또한 Warehouse 별로 Auto Suspend 시간을 최대한 타이트하게 구성하여 크레딧 낭비를 최소화 하였습니다.

2. ETL TASK 는 Serverless 활용



두 가지 방법을 모두 테스트한 결과

STG → RAW → CI 의 TASK 를 모두 Warehouse 를 지정했으나

유지보수

사례

- 해당 프로젝트를 수행하며 최초 수행했던 브랜드에서 어느날 갑자기 Stream → CI 로 가는 TASK 에서 Failure 가 발생했으며 이를 TASK 를 수행한지 4번째에나 확인할 수 있었습니다.

원인



STREAM 이 설정되어있던 TABLE 중 하나에 일정기간동안 데이터 변화가 없어 해당 STREAM 의 상태가 `stale` 이 되어

`SYSTEM$STREAM_HAS_DATA('BRAND_DB.RAW_SCHEMA.TABLE_A')` 를 사용하는 TASK의 Failure 가 발생했었습니다.

조치



빠르게 해당 STREAM 을 정상상태로 돌렸고 해당 TASK 를 수동으로 진행 시켜 파이프라인을 유지시켰습니다.

대응



이에 TASK 의 Failure 를 빠르게 알아차려야 할 필요성을 느끼게 되고 사내 메신저인 Slack 에 `snowflake-notification` 쓰레드를 구성하여 **ERROR NOTIFICATION** 을 구성하였습니다.

◦ 실제 구성한 테스트 notification



TASK 에 ERROR NOTIFICATION 을 AWS SNS 와 연결하여 AWS Lambda 의 트리거로 구성하여 에러 message 를 파싱하여 Slack 으로 보내도록 구성하였습니다.

소감

운영 DB 를 재시작하는 과정과 Snowflake 에 데이터를 이관하여 분석할 수 있는 초기환경을 직접 만들어 볼 수 있는 기회를 얻어 너무 감사했습니다. 또 문서를 보거나 혼자 토이 프로젝트로는 느낄 수 없는 책임감을 많이 가지게 되었고 이를 기회로 저는 한단계 더 성장했다고 생각합니다.

새로운 시도에 대한 두려움 보다는 설레임으로 항상 생각하는 마음이 이번 프로젝트도 잘 마무리할 수 있었다고 생각합니다.

앞으로 계획

Snowflake 에서 제공하는 CortexAI 를 활용한 SQL Agent 를 만들어 고객사 데이터분석 챗봇 등을 만들어 제공하고 싶습니다.