11. 스트림 처리

태그	
생성일	@2021년 8월 29일 오후 2:53

스트림 처리

- 일괄 처리는 처리할 데이터의 양이 정해져 있어야함
- 스트림은 시간 흐름에 따라 점진적으로 생산된 데이터 → 기본적으로 이 데이터의 단위를 “이벤트”라고 함

기본적인 용어

- producer - consumer
- publisher - subscriber
- 데이터 일련의 단위를 스트림 혹은 Topic 이라고함 (AWS SNS, Apache Kafka에서 Topic이라고 말함)

→ 이벤트 스트림은 데이터가 있을 때 처리한다 → Event Driven 하다 (DB의 Trigger 같은... ;;)

메시징 시스템

- 발행/구독 모델
  - AWS SNS, Redis Pubsub
- 생산자에서 소비자로 메시지 직접 전달
  - 소비자의 API에 직접 Call ..
- 메시지 브로커 (최근 드는 생각인데 메시지 큐와 이벤트 스트림은 분리가 필요하다고 생각 ex : Apache Kafak, Rabbit MQ 이 둘을 메시지 브로커라고 퉁치는 것은 살짝 아쉬운 느낌)
  - Rabbit MQ
  - Apache Kafka

다수의 컨슈머 (복수 소비자)

- 로드 밸런싱 → 순서보장 안됨
- 팬아웃
  - Redis Pub/Sub
  - AWS SNS

확인 응답과 재전송

- 실패했을 때 그냥 DLQ(Dead Letter Queue)를 사용하면 메시지 순서를 보장할 수 없다

로그기반 메시지 브로커

- 메시지 소비를 성공해도 메시지를 지우지 않는다 → Kafka, AWS Kinesis
- 성공하면 메시지를 지우는 것이 아니고 Offset Commit 한다
- 성공했는데 Offset Commit 실패할 경우 두번처리될 수 있음

로그기반 메시지 브로커에서는 파티션단위로 컨슘한다(중요)

모임 사용자

모임 신청 → 모임 + 사용자의 릴레이션 → 파티션 이 때 모임 신청도 PK가 있을거에요 (Auto Increment 인조키) → 이 키로 파티션 하면 → 한 모임에서 순서보장 안됨 → 그러면 모임 ID를 가지고 파티셔닝하면 모임신청 이벤트들을 순서보장 할 수 있음 한 모임에서는

Co-Partition → 재미있는 개념.. 어려움 → 어떻게 구현하지?

소비자가 생산자를 따라갈 수 없을 때 → 메시지 버리기, 버퍼링, 배압 → bulk → Log Compaction

오래된 메시지 재생 → binlog 처음부터 읽어서 follower를 만드는 것처럼 로그기반 메시지 브로커도 처음부터 읽어서 복원가능

데이터베이스와 스트림

- 데이터베이스의 변경사항이 사실 이벤트일 수도 있다 → Connector기반 CDC의 등장 이유가 될 수 있는듯

여러 시스템의 동기화

- 로그기반 메시지 브로커를 이용하면 각 분산 데이터 시스템들의 Eventual Consistency를 보장할 수 있음

CDC (Change Data Capture)

- 개인적으로 Event Stream 시작의 꽃이라고 생각함
- MySQL의 binlog MongoDB의 oplog를 읽어서 메시지 브로커에 넣어줍니다 → Connector, Confluence, debezium
- 그러면 메시지 브로커 = binlog(write-ahead log) 라고 볼 수 있음
- 데이터 너무 크면 초기 스냅숏 사용해야함 → Log Compaction이라는 대안이 있음 (Redis의 AOF Compact 과정과 유사)

이벤트 소싱

- Event Stream, Event Driven Architecture를 관통하는 개념임
- “이벤트 스트림”이 “단일 진실 공급원”
- CQRS → Kafka Sink Connector → Event Stream → 다양한 View
- DB → Stream 데이터 해방 (Data liberation pattern)
    - Query Liberation → LogStash DB Query Status Column
    - (CDC) Connector Liberation → 모든 테이블의 변경이 사실 이벤트가 아닐 수 있다
    - OutBox Pattern → Implementation Application → 이상적인데 구현 해야함

Event Driven

- 불변성, 멱등성 있는 로직(operator, job) 작성이 가능함

스트림 처리

- 복잡한 이벤트 처리(Complex Event Processing, CEP) → 여기 나온 내용 KSQL로 대체 가능
- 시간 관련해서서는 구현때도 잘 안하는 고민이긴합니다(클라우드쓰니까)
- 윈도우 → 일정 기간
    - 텀블링 윈도우
    - 홉핑 윈도우
    - 슬라이딩 윈도우
    - 세션 윈도우

스트림 조인

- KSQL, KSQLDB를 봅시다 아래는 제가 실습했던 예제임
  (MySQL에서 2개의 테이블을 MongoDB로 동기화)
- https://github.com/labyu/playground/tree/main/spring-ddd