3일차. 데이터 엔지니어링 변환 도구 - Spark

> *아파치 스파크를 통해 다양한 변환 예제를 실습합니다 이번 장에서 사용하는 외부 오픈 포트는 4040, 4041, 8888 입니다*

스파크 실습을 위한 도커 컨테이너를 기동합니다

- 최신 소스를 내려 받습니다

```
cd /home/ubuntu/work/data-engineer-intermediate-training
git pull
```

- 스파크 워크스페이스로 이동하여 도커를 기동합니다

```
cd /home/ubuntu/work/data-engineer-intermediate-training/day3
docker-compose up -d
docker-compose logs -f notebook
```

- 출력되는 로그 가운데 마지막에 URL 이 출력되는데 해당 URL에서 127.0.0.1 값을 student#.lgebigdata.com 으로 변경하여 접속합니다
    - http://student#.lgebigdata.com:8888/?token=d508d3a860cbc00c1095b078f9f7bd755a3b3f95f715692e
    - 접속하면 jupyter notebook lab 이 열리고 work 폴더가 보이면 정상 기동 된 것입니다
    - 이제 마우스로 work 폴더를 클릭하고 해당 경로 안에서 노트북 페이지를 생성합니다
- 목차
    - 스파크 기본 명령어 이해
        - 1. 스파크 기본 명령어 이해
        - 2. 기본 연산 다루기
        - 3. 데이터 타입 다루기
        - 4. 조인 연산 다루기
        - 5. 집계 연산 다루기
        - 6. 스파크 JDBC to MySQL
        - 7. 스파크 JDBC to MongoDB
    - 스파크 고급 명령어 이해
        - 1. Repartition vs. Coalesce Explained
        - 2. Skewness Problem Resolutions
        - 3. Cache, Persist and Unpersist
        - 4. Partitioning Explained
        - 5. Bucketing Explained
    - 파이썬 첫걸음 (외부 문서링크) - Python 3.x 기준
        - 1. 파이썬 기초문법 1부
        - 2. 파이썬 기초문법 2부
        - 3. 코딩을 몰라도 쉽게 만드는 데이터수집기 만들기
    - 파이썬 분석 첫걸음 (외부 문서링킁)
        - 1. Pandas 기초
        - 2. Numpy 기초
        - 3. Plotly 기초

스파크 성능 개선 방향

> *과거 한 대의 장비에서 최대한 높은 성능의 장비에서 모든 데이터를 넣어두고 데이터 처리를 하는 환경에서 분산 저장소에 데이터를 저장하고 병렬처리를 하는 환경에서는 접근 방법이 다를 수 밖에 없습니다. 기본적으로 하둡 기반의 데이터 처리에 있어서는 모든 데이터가 물리적으로 다른 장비에 고르게 분산되어 저장되어 있다는 가정이 있으며 특정 레코드나 특정 범위의 데이터를 가져오는 것 자체가 어려울 수 있습니다. 하여 분산 환경에서 특히 하둡 + 스파크 환경에서 성능을 개선할 수 있는 방향에 대해 정리해 보았습니다 실무에서 사용하는 대부분의 작업은 데이터를 통해 인사이트를 얻기 위한 탐험적 분석(EDA) 혹은 그러한 탐험적 분석을 통해 의미있는 지표가 나왔다면 이러한 지표를 KPI 로 정하고 일간, 주간, 월간 지표를 뽑기 위한 정형화된 데이터 처리를 수행하는 것이 일반적인 데이터웨어하우스의 사용접근 방법일 것입니다. 그리고 이러한 지표들은 단순한 조회가 아니라 Group By, Join 혹은 Union 등의 복잡한 데이터 처리 과정을 거쳐서 생성되기 마련인데 이러한 연산에는 반드시 Sorting (Shuffling)이 발생할 수 밖에 없으며 이러한 것이 전체적인 성능을 떨어뜨리는 가장 큰 요인입니다. 그렇다면 이러한 정렬과 많은 데이터를 빠르게 처리하기 위한 분산 환경에서 가장 좋은 성능을 낼 수 있는 방법은 정렬 혹은 집계를 빠르게 하고 병렬처리를 최대한 잘하면 되는 것이라고 보입니다. 즉 정말 필요한 데이터만 데이터소스에서 읽어오고, 데이터 처리시에 최대한 적은 데이터만 노드간에 전달하고, 데이터 처리는 최대한 병렬로 처리하고, 저장 시에 다음에 읽기에 최적화된 상태로 저장하는 것입니다. 참 쉽죠~ ;-)*

1. 데이터 I/O 를 최대한 줄이는 것

- 파티셔닝을 통한 데이터 범위를 줄이는 것
  - 데이터베이스의 인덱스와 마찬가지로 쿼리 분석을 통해, 가장 자주 많이 조회되는 쿼리의 Where 조건에 사용되는 Boundary Query Column 을 Key Partition 으로 지정합니다
- 인덱스를 가진 저장 포맷으로 저장하는 것
  - Parquet 혹은 ORC 와 같은 색인을 별도로 가진 데이터 포맷으로 저장하여 데이터소스의 색인을 통한 필터 조건이 적용될 수 있도록 구성합니다
  - Columnar Storage 포맷인 Parquet 를 선택하는 경우 Column Pruning 효과를 볼 수 있어 데이터소스로부터 가져오는 데이터 크기를 줄이는 효과를 가집니다
  - 내장된 색인구조를 통해 조회 시에 Predicate Pushdown 적용을 통해 필요한 데이터 블록만 읽어올 수 있는데 반드시 쿼리 분석을 통해 First-Index 를 해당 Key 값으로 저장해야만 합니다

2. 네트워크 I/O 를 최대한 줄이는 것

- Broadcast 조인을 통한 셔플링을 줄이는 것
  - 하나의 테이블이 충분히 작은 경우는 작은 테이블 데이터 전체를 조인이 발생하는 모든 노드에 전송을 통해 셔플링을 줄일 수 있습니다
- 버킷팅을 통해 셔플링을 줄이는 것
  - 조인 시에 2개의 테이블 모두가 큰 테이블의 경우 Join 되는 킷 값을 기준으로 버킷을 생성해 두는 경우 Sort Merge Join 을 통해 셔플링을 줄일 수 있습니다

3. 최대한 많은 데이터를 동시에 빠르게 처리하는 것

- 코어, 메모리 그리고 오프힙 설정을 최대한 활용하는 것
  - 리소스가 충분하다면 최대한 사용가능한 리소스를 활용할 수 있도록 메모리, 코어 및 오프힙 설정 조정을 통해 짧은 시간에 최대한 모든 리소스를 사용합니다
- 특정 값에 외곡된 데이터의 경우 예외적인 로직을 적용합니다
  - Null 값이 많은 컬럼에 대한 집계처리의 경우 해당 값의 Reduce 작업이 지연되어 전체 작업이 지연될 수 있으며 크게 2가지 해결방안이 있습니다
    - 첫번째는 Null 값을 필터하여 2개의 파이프라인을 동시에 처리하고 2개의 파이프라인을 Union 하는 방법
    - 두번째는 Salting 기법으로 원 킷값에 잘 분산될 수 있는 해시 키값을 섞어서 셔플링이 충분히 잘 되게 하고 처리 후 다시 처리하는 방법

4. 처리가 완료된 데이터를 최적화된 상태로 저장하는 것

- 충분히 작은 파일로 저장될 수 있도록 파일 수를 조정하는 것

- M/R 작업이 완료되고난 경우 병렬처리의 수에 따라 그대로 저장되는 경우가 대부분이므로 반드시 Coerce 를 통해 파일 수를 줄여야 합니다
- Coerce 의 경우 이전 단계의 Reduce 작업의 수에 영향을 미칠 수 있으므로 유의해서 사용해야 합니다
- Coerce 가 어려운 상황이라면 Repartitioning 을 통해 Shuffle 이 일어나더라도 더 나은 성능과 결과물을 만들 수 있습니다