데이터 엔지니어 초급 4 일차

데이터 적재 서비스 하이브

Park Suhyuk
Data Ingestion Team Leader


psyoblade


psyoblade

NCSOFT®

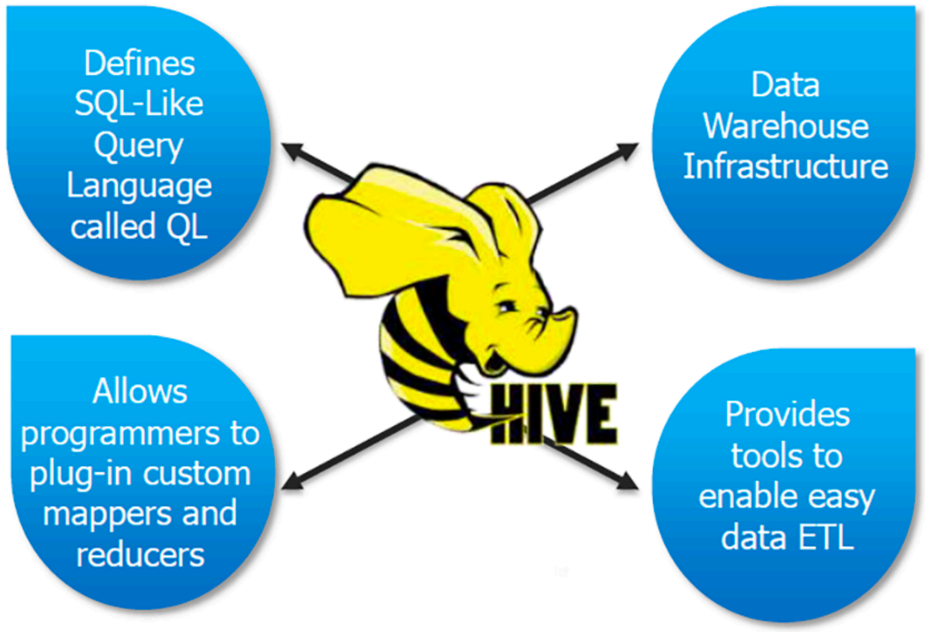
목차

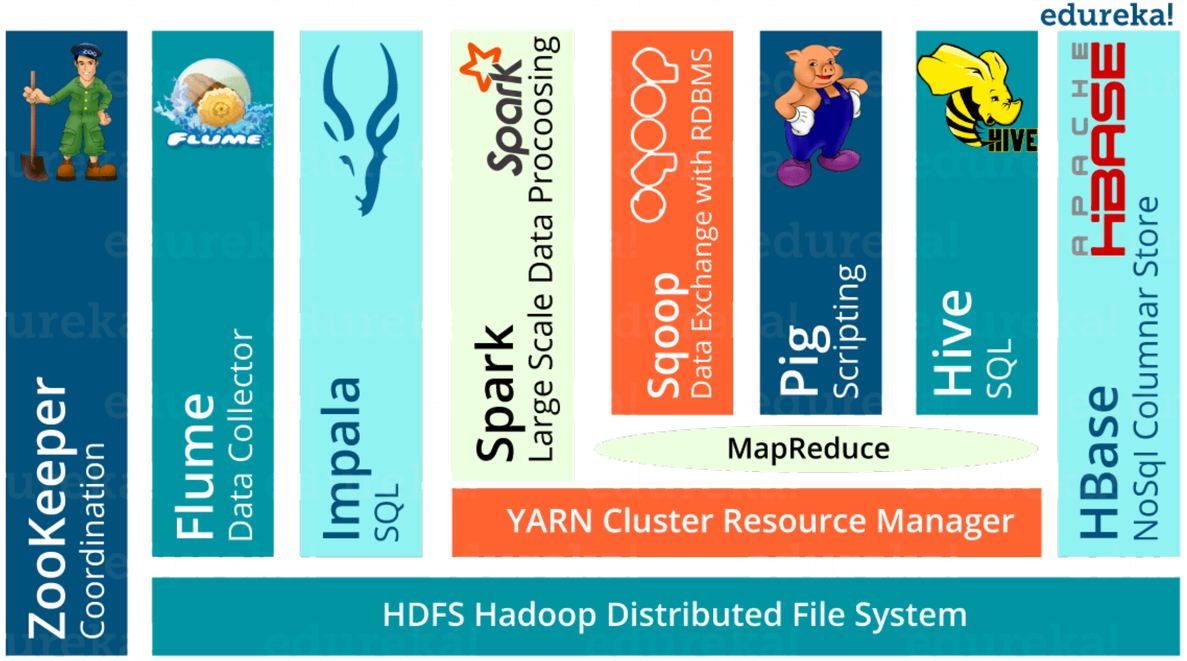
아파치 하이브

아파치 하이브 – 개요

What is 'Hive' ?

하이브는 분산 저장소에 존재하는 대용량 데이터를 SQL을 통해 관리할 수 있는 SQL on Hadoop 데이터 웨어하우스 엔진입니다. 2010년 부터 Facebook 에서 개발하기 시작하여, 현재는 아파치 프로젝트인 하둡 기반의 데이터 웨어하우스 프레임워크로써 복잡한 MapReduce 코드 대신 SQL 만으로 대용량 데이터 분산처리가 가능하여, 기존의 DW 기반의 BI 개발자들도 손쉽게 사용할 수 있으며, SQL 에서 제공하는 다양한 함수 및 Optimizer 등이 거의 대부분 적용되어 확장성과 유지보수성이 뛰어납니다.





데이터 서비스 엔진 – Hive

Apache Hive

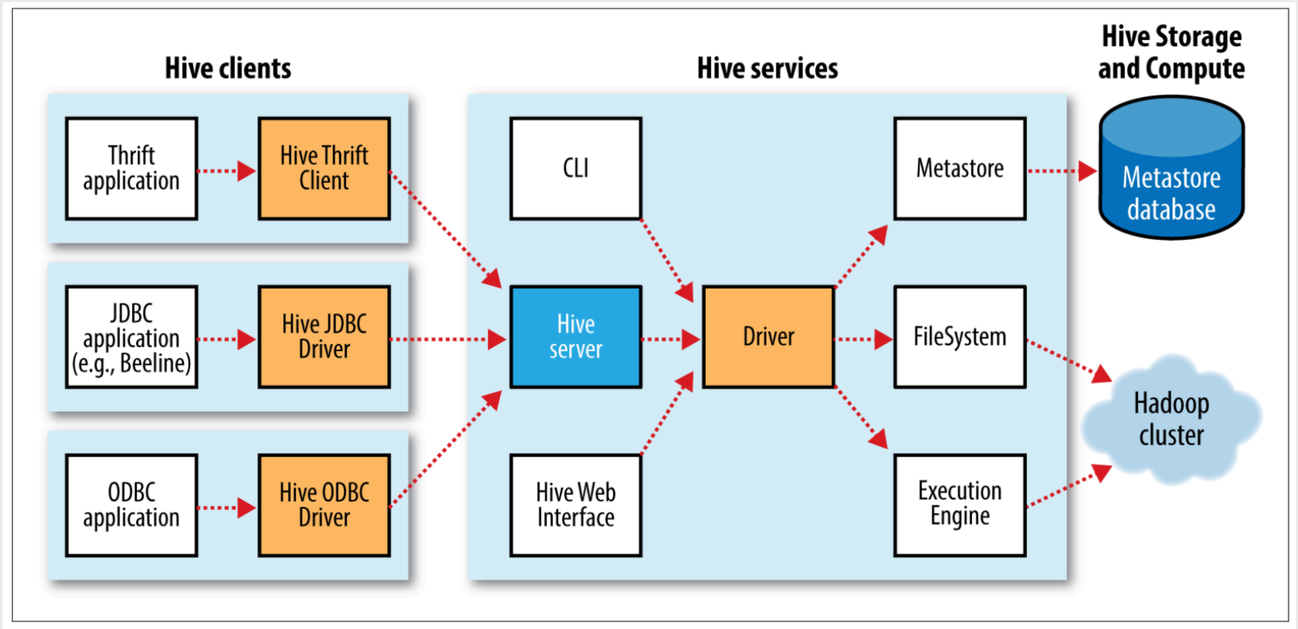
복잡한 MapReduce 코드 대신 SQL 만으로 대용량 데이터 분산처리가 가능하여, 기존의 DW 기반의 BI 개발자들도 손쉽게 사용할 수 있으며, SQL 에서 제공하는 다양한 함수 및 Optimizer 등이 거의 대부분 적용되어 확장성과 유지보수성이 뛰어나지만, 반복적인 처리와 업데이트가 자주 발생해야 하는 기계학습 혹은 그래프 분석 알고리즘에는 SQL 을 통한 데이터 파이프라인으로는 부족할 수 있습니다

Pros

1. SQL on Hadoop 기능을 구현한 오픈소스 데이터 웨어하우스
2. SQL 엔진의 특징인 쿼리 옵티마이저에 따른 최적화
3. 유저, 그룹 별 접근 권한 및 스키마에 에볼루션 지원

Cons

1. 실시간 쿼리나 레코드 단위 업데이트 등의 OLTP 기능에 부적합
2. MapReduce 기반의 처리가 기본이므로 쿼리 레이턴시는 느림
3. SQL 기반의 데이터 처리로 파이프라인 구성 혹은 모듈화가 어려움



아파치 하이브 – 기동 방식

Hive Internal

Query 수행 절차

1. executeQuery → CLI 혹은 웹 UI 를 통해 HiveQL 을 Driver 로 전달
2. getPlan → 세션을 생성하고 컴파일러에 해당 쿼리를 넘겨서 실행 계획을 생성
3. getMetadata → 메타스토어로부터 메타를 가져와 실행 계획을 드라이버에 전달
4. executePlan → Yarn 혹은 MapReduce 작업을 통해서 실행 계획 수행
5. sendResult → 실행 결과를 반환합니다

Components

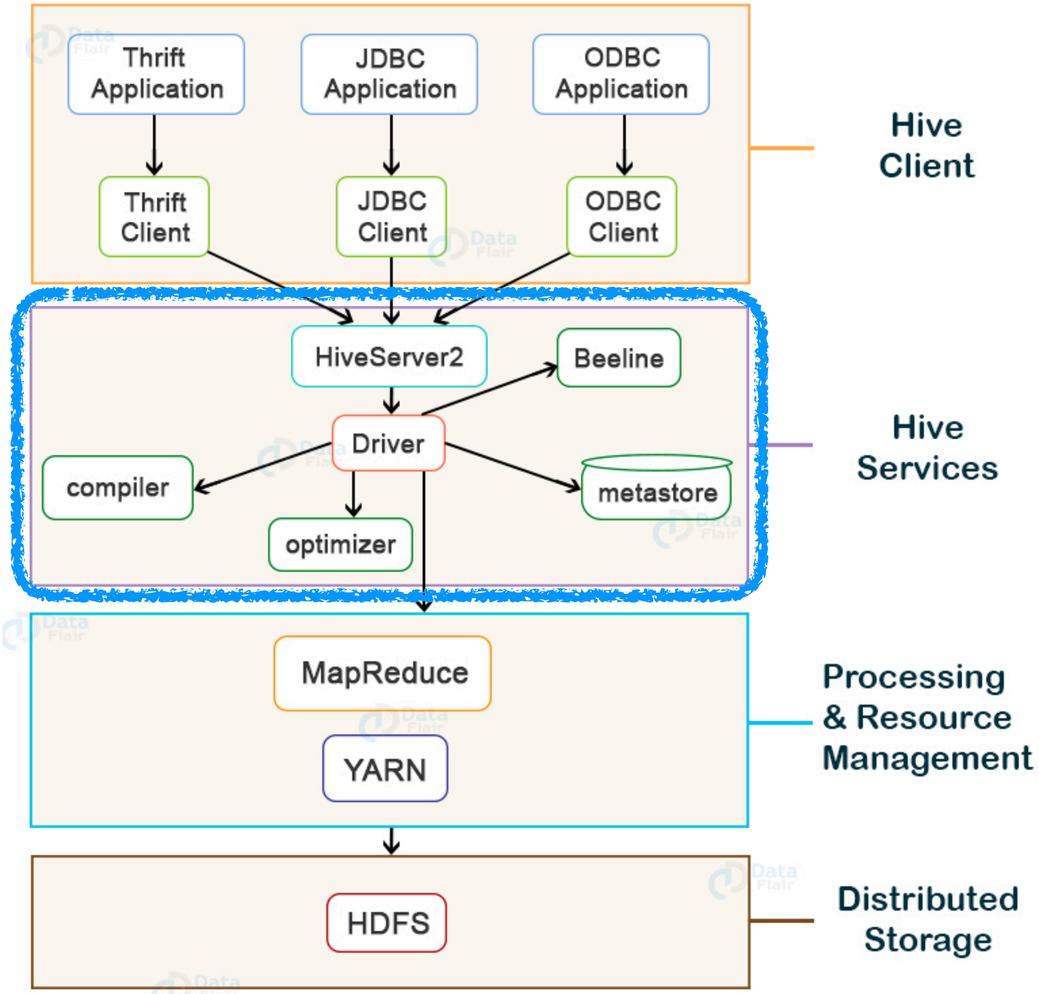
HiveServer2 : 클라이언트 쿼리 수신 및 생성된 결과를 전달하는 브로커

Hive Driver : 요청 쿼리에 대한 세션 생성 및 컴파일러 통한 실행계획 생성

Hive Compiler : 쿼리 구문분석, 데이터 타입 체크 및 메타 통한 물리 수행계획

Optimizer : 성능과 확장성을 검토하여 변환 작업을 통해 최적화 수행

Execution Engine : 물리 계획을 직접 수행하는 엔진



Hive Architecture & Its Components

https://data-flair.training/blogs/apache-hive-architecture/

아파치 하이브 - Hive vs. Spark

What is differences between Hive and Spark

Spark SQL 의 경우 Hive 엔진과 연동 시 Hive 의 Metastore 기반으로 동작하기 때문에 어플리케이션 수준에서 운영 시에 큰 차이점을 느끼지 못 합니다. 특히 JDBC, ODBC 지원이나, OLAP 데이터 처리 지향적이며 Batch 처리에 최적화 되어 있으며, 레코드 단위 업데이트를 지원하지 않는 점 등 Hive QL 과 Spark SQL 은 상당히 유사한 점이 많습니다. 하지만 실행 엔진은 설계 방향이 다르기 때문에 아래와 같은 다른 점들에 유의하여 사용 및 선택할 수 있습니다.

	Hive QL	Spark SQL
실행엔진	MapReduce or Tez 기반의 느리지만 안정적인 SQL 수행이 가능한 엔진	Memory 기반의 Spark Engine 을 통한 빠르지만 상대적으로 다소 민감한 엔진
플러그인	SQL 문법을 기본으로 UDF, UDAF 등은 다양한 언어를 통해 컴파일 후 배포	Scala, Java, Python 및 R 을 통한 런타임 시에 적용 가능
권한설정	유저, 그룹 별 접근 권한을 지정할 수 있습니다.	유저에 대한 권한 지정 기능은 없습니다
스키마	유연한 스키마와 추가된 스키마에 에볼루션 기능을 지원합니다	스키마 관련 기능은 없지만 Parquet 포맷 자체의 스키마 에볼루션을 활용합니다
메타데이터	테이블 수준에서 파티션, 버킷팅을 지원합니다	자체 메타 스토어가 없기 때문에 하이브와 연동하여 테이블 파티션, 버킷팅 지원

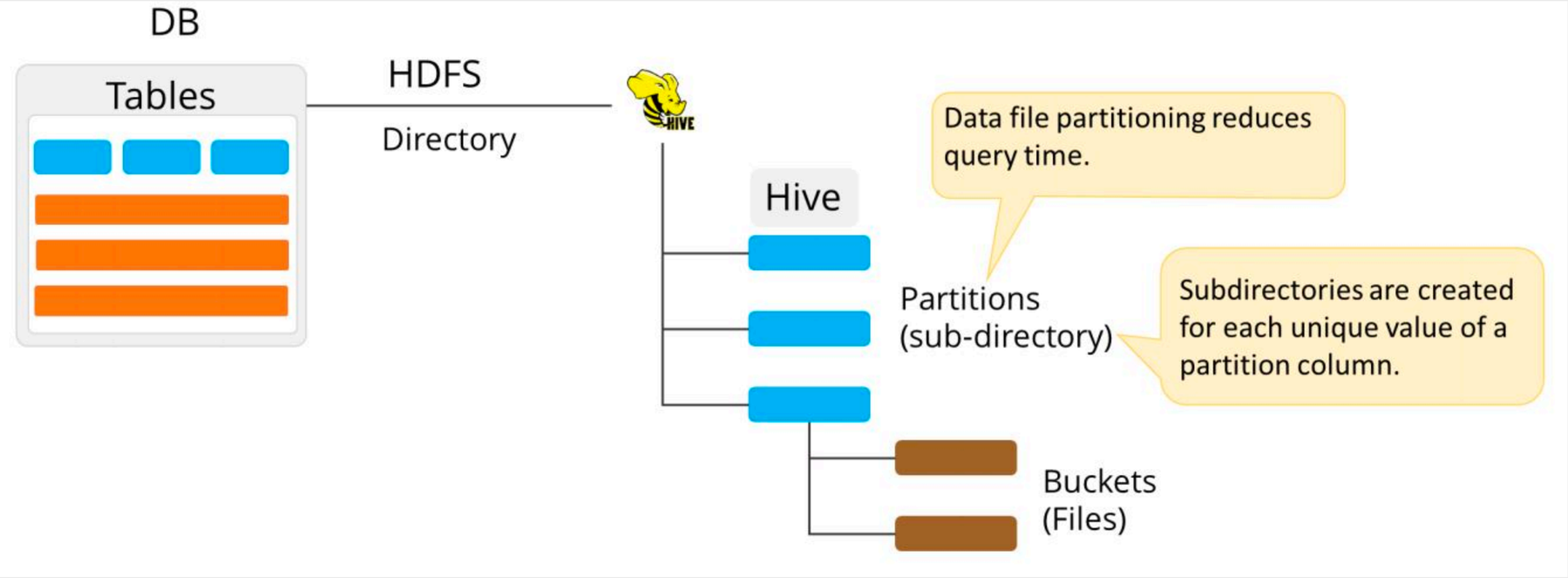
Apache Hive Troubleshooting

하이브 서비스 운영 시에 가장 흔히 발생하는 문제점은 Multitenancy 환경에서 대량의 쿼리가 몰려서 Pending 상태에 걸리거나, 일부 공통 테이블의 CRUD 작업의 Lock 에 의한 지연 그리고 잘못된 질의 혹은 대용량 데이터 처리에 의한 리소스 부족 등이 주요 원인입니다. 이러한 다양한 문제점들을 어떻게 디버깅 할 수 있으며, 일반적으로 잘 알려진 방법을 적용함으로써 해결할 수 있습니다. 또한 튜닝 시에는 Fine-tuning 보다는 Coarse-grained-tuning 이 될 수 있도록 해야 향후 환경적인 변화에도 유연하게 대응할 수 있으며 유지보수에 따른 비용도 줄일 수 있습니다.

1. 대상 테이블이 너무 커서 조회 혹은 분석 시에 너무 오랜 시간이 걸리는 경우 - **Partition or Bucket**
2. 테이블의 용량이 너무 크거나, 많은 파일에 의해 조회 뿐만 아니라 저장 공간을 너무 많이 차지하는 경우 - **Parquet or ORC**
3. 조인 혹은 집계 연산 작업의 비용이 크거나 느린 경우 - **Denormalize**
4. 정렬, 집계 시에 기존의 SQL 문을 조금 더 최적화 하고 싶은 경우 - Order By, Group By, Distribute By, **Partition By, Sort By**, Cluster By 검토
5. 평소에는 잘 조회 되지만 특정 시점에만 느리거나 지연되는 불규칙한 성능을 보이는 경우 - **Avoid Locking**
6. 현재 수행되는 질의가 잘 수행되는 지 예측하기 어려운 경우 - **Explain**, Debug

Slow queries - table partitioning and bucketing

하이브는 MapReduce 기반의 엔진이므로 한 번에 읽어야 할 데이터의 크기가 너무 크거나 제대로 구조화되어 있지 않다면 모든 데이터를 읽어올 수 밖에 없으므로 이러한 부분을 조정하는 것이 Partitioning 과 Bucketing 의 개념 입니다. 여기서 파티셔닝은 말 그대로 특정 값을 기준으로 물리적으로 다른 경로에 저장하는 방식이며 Bucketing 은 특정 값을 기준으로 파일을 분리해서 저장해두는 기법입니다.



https://www.simplilearn.com/advanced-hive-concept-and-data-file-partitioning-tutorial

아파치 하이브 – 트러블 슈팅 #2

Too Big table - Parquet or ORC format

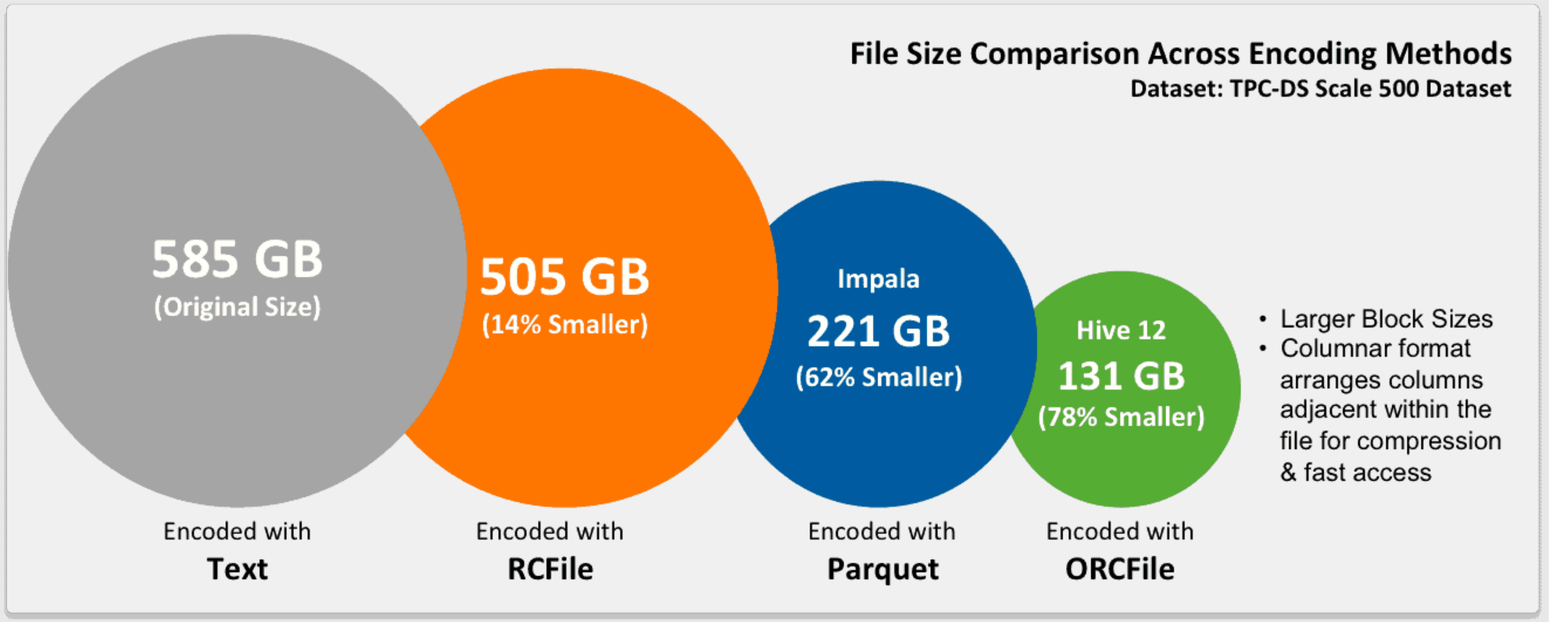
하이브 테이블의 경우 텍스트 포맷(CSV, TSV 혹은 JSON)도 지원하지만 다양한 바이너리 압축 포맷도 (Avro, Parquet, ORC 등) 지원합니다. 단순히 파일로 저장하는 경우 디버깅 하기에도 좋고, 직관적으로 확인할 수 있어 용이한 점도 많지만, 텍스트 파일의 경우 압축되어 있지 않은 경우 압축된 테이블에 비해 약 2~3배 이상 커질 수 있으며, 단순 압축(gzip 등)을 수행하는 경우 Splittable 하지 않아 성능에 큰 영향을 줄 수 있습니다. Column 기반 저장 포맷인 Parquet 와 Hive 에서 지원하는 Row 기반 저장 포맷인 ORC가 있는데 일반적으로 Spark 과 연동해서 사용하는 경우라면 Parquet 포맷을 사용하는 것을 추천 드립니다.

Pros

1. 데이터 저장 공간을 약 60~70% 절약
2. 데이터 조회 성능 향상
   - Predicate pushdown
   - Column Pruning
3. 스키마 에볼루션 및 데이터 타입 유지

Cons

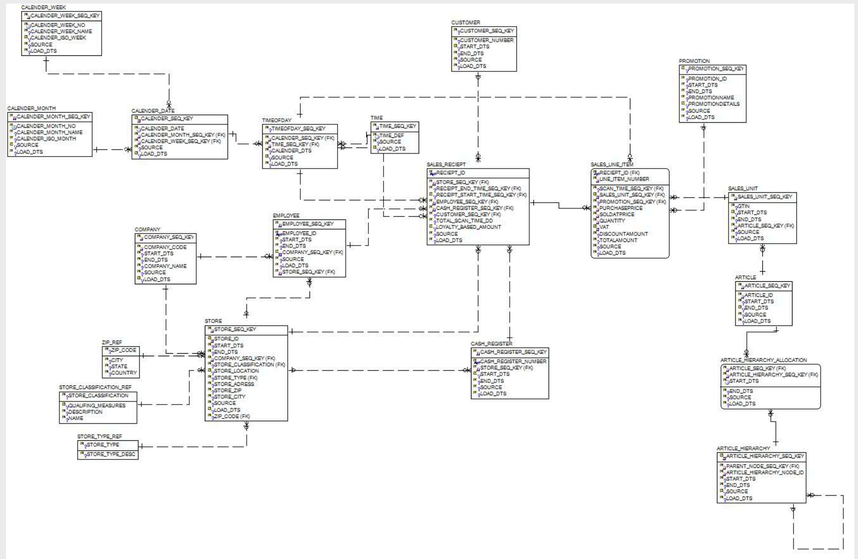
1. 관련 라이브러리를 통해서만 확인 및 변경 가능

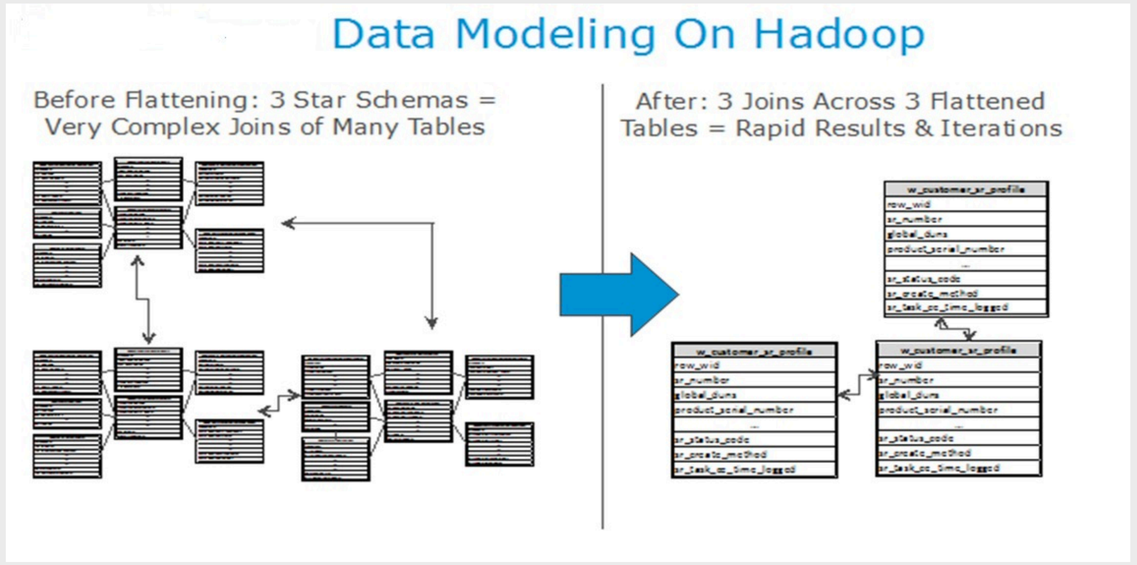


https://blog.cloudera.com/orcfile-in-hdp-2-better-compression-better-performance/

Slow Join and Aggregation - Denormalize

빅 데이터 영역에 있어 데이터 분석의 경우 집계 연산의 성능이 가장 중요하다고 말할 수 있습니다. 레거시 테이블의 스냅샷 혹은 각종 로그를 하둡 클러스터에 저장된 상황에서는 기존 관계형 데이터베이스의 정규화 테이블을 그대로 사용할 수 밖에 없는데 이러한 경우 너무 많은 조인에 의한 셔플링이 성능을 떨어뜨리는 큰 요인 중에 하나 입니다. 하여 ELT (Extraction, Load, Transformation) 즉, 데이터 적재 이후에 적절한 후처리를 통해 Denormalization 된 테이블을 생성하는 것이 중요합니다. 기존의 레거시 테이블을 하나의 아주 큰 Dimension 혹은 Fact 테이블을 생성하여 하나의 테이블에 약 50~100개 이상의 컬럼을 생성하고 이를 통해 각종 Summary 테이블 혹은 또 다른 Fact 테이블을 추출해 낼 수 있습니다.





http://blog.dbtechnosolutions.com/2015/08/data-modelling-in-hadoop.html

아파치 하이브 - 트러블 슈팅 #4

Top 10 + OrderBy == Rank 10 + PartitionBy + SortBy

Q1. 전체 사원 중에서 가장 나이가 많은 10명의 이름과 나이는?

A1. select user_age, user_name from employee order by user_age desc limit 10;

A2. select dept_id, user_id, user_age from (

    select dept_id, user_id, user_age, rank() over (partition by dept_id order by user_age desc) as rank from employee

) t where rank < 11 order by user_age desc limit 10;

	특징	예제
Order By	모든 데이터가 해당 키에 대해 정렬됨을 보장합니다	select * from employee order by dept_id
Group By	군집 후 집계함수를 사용할 수 있습니다	select dept_id, count(*) from employee group by dept_id
Sort By	해당 파티션 내에서만 정렬을 보장 (set mapred.reduce.task = 2)	select * from employee sort by dept_id desc
Distribute By	단순히 해당 파티션 별로 구분되어 실행됨 보장, 정렬 보장 안함	select * from employee distribute by dept_id
Distribute By Sort By	파티션과 해당 필드에 대해 모두 정렬을 보장	select * from employee distribute by dept_id sort by dept_id asc, seq desc
Cluster By	파티션 정렬 까지만 보장 합니다	select * from employee cluster by dept_id

아파치 하이브 - 트러블 슈팅 #5

Avoid table locking

하이브의 경우 2가지 동시성 모델(Shared Lock, Exclusive Lock)을 지원합니다. 즉, 테이블을 읽고 있는 경우 해당 테이블에 대한 변경(ALTER 등)이 불가능하기 때문에 자주 사용하는 테이블 혹은 변경이 자주 발생하는 (파티션 추가 등) 테이블의 경우 동시성 이슈로 조회나 사용에 있어 지연되는 경우가 빈번하게 발생할 수 있습니다. 이러한 문제를 회피하기 위해서 데이터베이스와 마찬가지로 해당 동시성 문제를 최대한 발생시키지 않도록 모델링 혹은 구현하거나 최소화 하여 운영하는 방안을 검토해야 합니다.

Use Cases

1. 자주 읽기(S on T1)가 발생하는 테이블에 대해 야간 배치 파티션이 추가 (X on T1) 작업
   - 조회가 빈번하게 사용되지 않는 시점으로 배치 작업 스케줄을 조정
2. 시간 단위로 파티션이 추가(X on T2)되는 테이블에 대해 비정기적으로 분석 질의(S on T1)가 수행
   - ALTER TABLE orders ADD PARTITION (dt = '2016-05-14') .... PARTITION (dt = '2016-05-20'); // 배치 튜닝
   - hive.support.concurrency = false

Debugging

- SHOW LOCKS <TABLE_NAME> ;
- SHOW LOCKS <TABLE_NAME> PARTITION (<PARTITION_DESC>) ;

The compatibility matrix is as follows:

Lock Compatibility		Existing Lock	
		S	X
Requested Lock	S	True	False
	X	False	False

https://cwiki.apache.org/confluence/display/Hive/Locking

Avoid table locking

For example

```
EXPLAIN LOCKS UPDATE target SET b = 1 WHERE p IN (SELECT t.q1
```

Will produce output like this.

```
LOCK INFORMATION:
default.source -> SHARED_READ
default.target.p=1/q=2 -> SHARED_READ
default.target.p=1/q=3 -> SHARED_READ
default.target.p=2/q=2 -> SHARED_READ
default.target.p=2/q=2 -> SHARED_WRITE
default.target.p=1/q=3 -> SHARED_WRITE
default.target.p=1/q=2 -> SHARED_WRITE
```

Hive Command	Locks Acquired
select .. T1 partition P1	S on T1, T1.P1
insert into T2(partition P2) select .. T1 partition P1	S on T2, T1, T1.P1 and X on T2.P2
insert into T2(partition P.Q) select .. T1 partition P1	S on T2, T2.P, T1, T1.P1 and X on T2.P.Q
alter table T1 rename T2	X on T1
alter table T1 add cols	X on T1
alter table T1 replace cols	X on T1
alter table T1 change cols	X on T1
alter table T1 concatenate	X on T1
alter table T1 add partition P1	S on T1, X on T1.P1
alter table T1 drop partition P1	S on T1, X on T1.P1
alter table T1 touch partition P1	S on T1, X on T1.P1
alter table T1 set serdeproperties	S on T1
alter table T1 set serializer	S on T1
alter table T1 set file format	S on T1
alter table T1 set tblproperties	X on T1
alter table T1 partition P1 concatenate	X on T1.P1
drop table T1	X on T1

https://cwiki.apache.org/confluence/display/Hive/Locking

Q&A