데이터 엔지니어링 테이블 수집

관계형 데이터베이스 수집 개요

Park Suhyuk

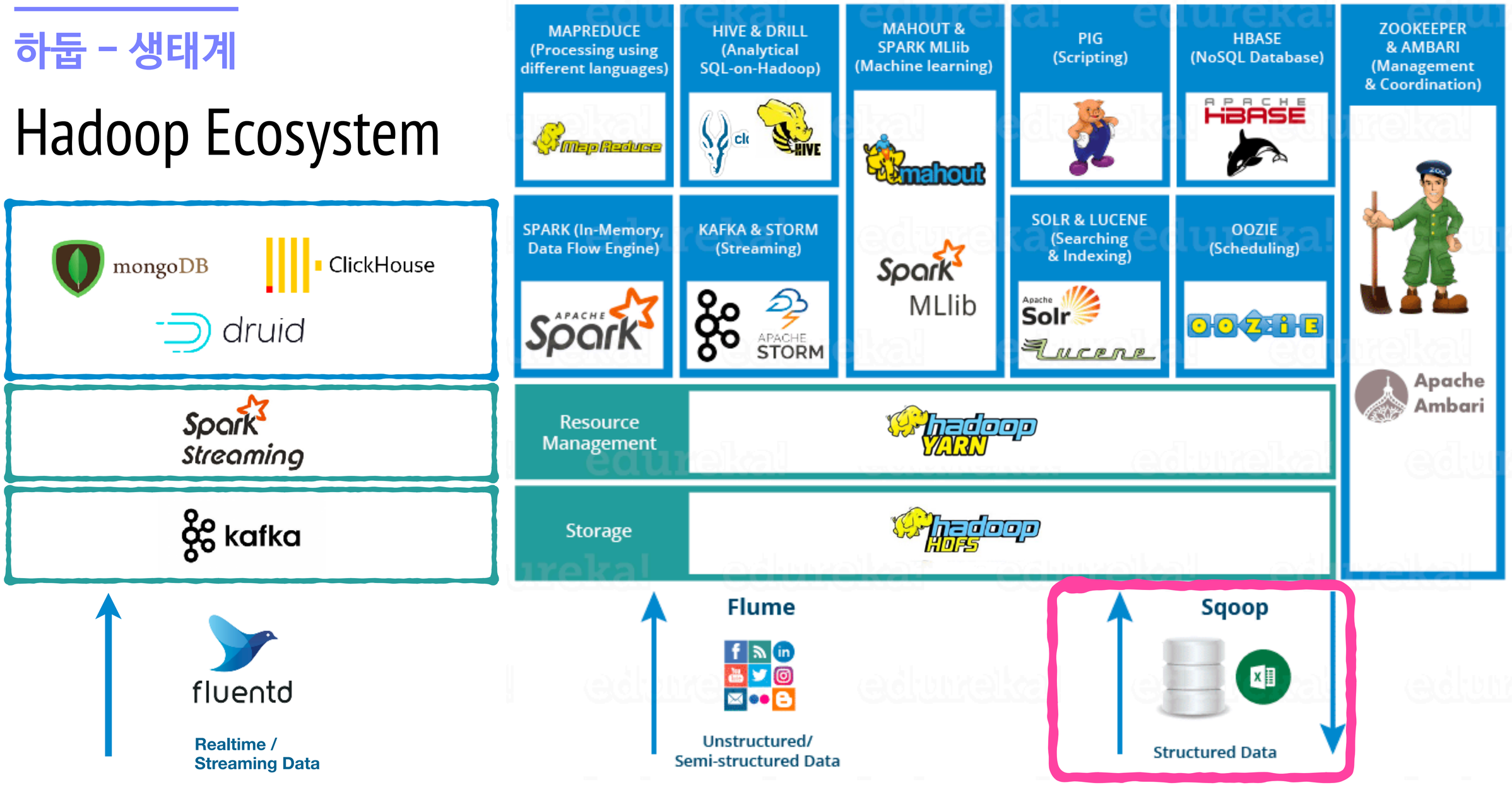
 psyoblade

 psyoblade

ubuntu 20.04 LTS

하둡 – 생태계

Hadoop Ecosystem



목차

About Relational Database

관계형 데이터베이스의 특성과 테이블 수집의 필요성

관계형 데이터베이스 수집

Collecting Relational Database

발생하는 모든 로그를 시간 순서대로 적재하는 로그와 다르게 관계형 데이터베이스에 저장된 데이터는 실시간으로 변경되고 있으며, 수 초 단위로 변경될 수 있습니다. 고객정보, 상품정보 혹은 장비 혹은 주문의 상태 정보 등, 대부분 현재의 상태를 나타내고 있어 시간이 흐르면 변경되어 과거의 상태를 확인하기 어려운 데이터의 특성을 가지고 있습니다. 로그 수집과 테이블 수집은 서로 다른 성질의 데이터를 수집하기 위한 도구로써, 상호 보완적인 특징을 가지므로, 얻고자 하는 데이터의 특성에 따라 설계되어야 합니다

	로그	테이블
정보의 휘발성	시간이 지나도 과거 이력 및 모든 정보가 유지	시간이 지나면 현재 상태만 유지
저장 유형 및 포맷	다양한 포맷의 형태로 저장이 가능함	정해진 스키마의 유형과 타입으로만 저장
데이터의 경계	끊임 없이 생성 늘어나는 경계가 모호함 (unbounded)	특정 시점의 스냅샷을 저장하여 경계가 명확 (bounded)
한 레코드 당 정보량	가능한 반드시 필요한 정보만 담도록 설계	정규화 등을 통해 상대적으로 많은 정보를 저장이 가능
데이터 특성	스트리밍 데이터	스냅샷 데이터
저장소 유형	대용량 스토리지, 분산 저장소 등	관계형 데이터베이스

Relational Database Snapshot

관계형 데이터베이스 스냅샷 수집 및 활용 사례

관계형 데이터베이스 수집

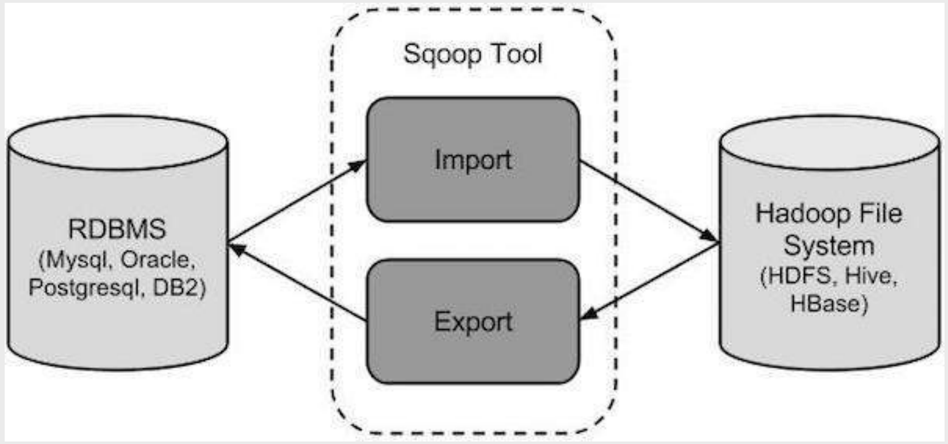
Relational Database Collection

전통적인 관계형 데이터베이스를 이용한 데이터웨어하우스의 경우 정형화 된 테이블 데이터 구조를 가지고 있어 데이터 분석이나 처리에 유리합니다. 다만, 데이터베이스 아키텍처 특성 상 (클러스터 구성을 하지 않는 경우) 너무 많은 커넥션이 몰리거나, 대용량 워크로드에 취약한 면이 있지만, 데이터의 변화 과정이나 트랜잭션 자체가 중요한 경우 사용합니다.

반면, 대용량 데이터의 저장 및 분산 처리 시스템인 하둡 플랫폼의 경우 빅 데이터를 통해 인사이트를 얻고 분석하기 위한 용도로 설계 되었으며, 수시로 변화하는 데이터를 보기 보다는 특정 시점의 스냅샷 데이터를 의사결정을 하기 위한 데이터 처리가 더 적합한 시스템이라고 말할 수 있습니다.

데이터 처리의 가장 첫번째 단계로 분산 저장소에 저장 을 위한 도구가 아파치 스쿱입니다.

* 기존의 분산되어 사일로 형태로 조직 별 데이터를 저장하던 구조의 문제점
* ERP, CRM, Logs, Web 등의 다양한 형태의 데이터를 통합 하여 보기 어려운 점
  - 저장공간, 처리 능력, 스키마, 인증 등
* 일원화된 데이터 수집, 처리, 적재 및 조회 인프라의 부족
  - 데이터베이스, XML, 웹, 웹로그, 인증로그 등
* 워크로드 증가에 따른 수평적 확장의 어려움



관계형 데이터베이스 활용

Join snapshot with logs

로그에 모든 정보를 다 남길 수 있다면 좋겠지만, 로그를 남기는 시점에 확인할 수 없거나, 추후에 변경될 수 있는 정보 (고객 등급, 상품 등급 등) 혹은 너무 많은 정보라서 로그에 모두 남기기 부담스러운 정보(상품 정보 등)들도 있습니다. 이러한 경우에도 다양한 차원에서 데이터를 분석하고, 집계할 필요가 있기 때문에 로그와 관계형 데이터베이스 정보를 모두 수집할 수 있어야 좀 더 정교한 분석이 가능합니다.

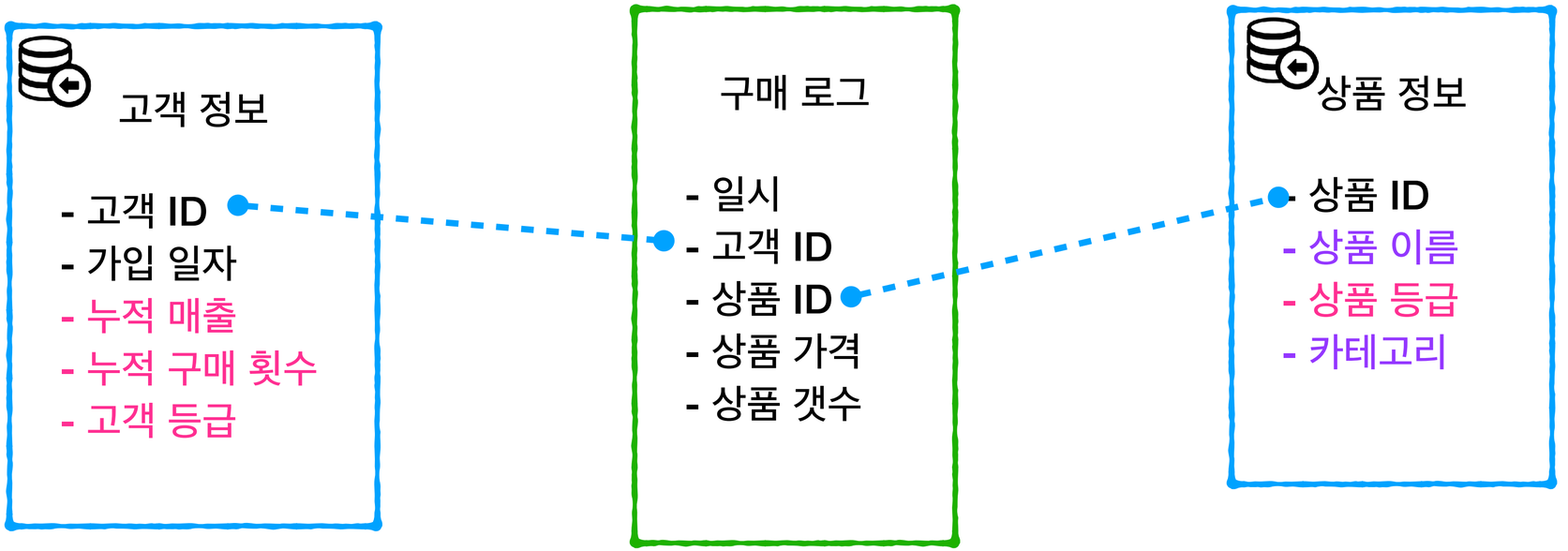
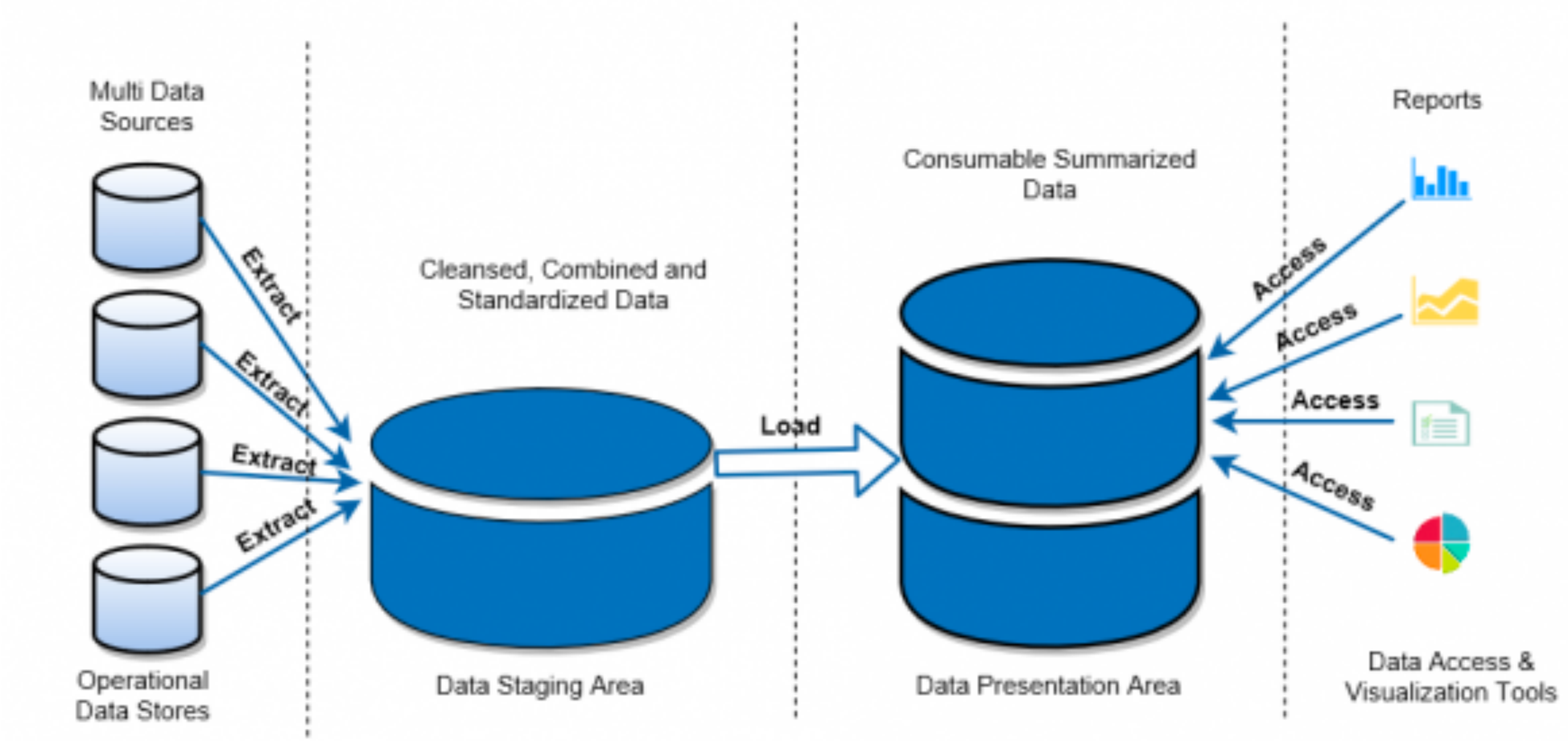


Table snapshot on Distributed Environment

분산환경에서의 테이블 스냅샷 활용 방식

고전적 데이터 웨어하우스 비교

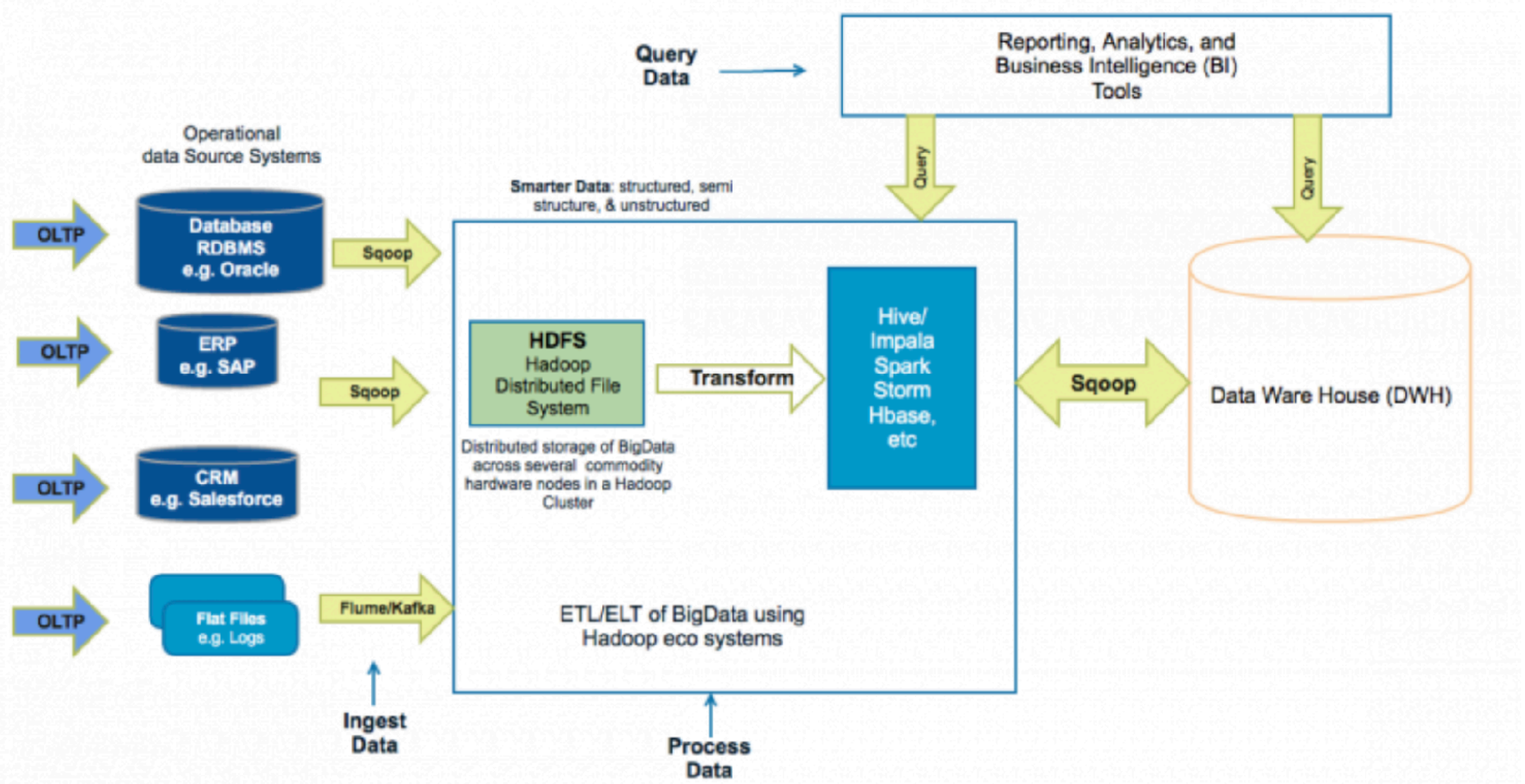
What is 'Data Warehouse' ?



Conventional Data Warehouse Architecture

관계형 데이터베이스 수집

Relational Database Collection



관계형 데이터베이스 수집

Relational Database on Distributed System

아파치 스쿱의 경우 데이터를 수집하는 도구이며, 이를 조회할 수 있는 방법은 없기 때문에 가장 널리 알려진 SQL on Hadoop 엔진인 Hive 테이블 생성을 엔진에서 지원하며, 테이블 수집과 동시에 하이브에서 조회할 수 있도록 설계 되었습니다. 아파치 하이브는 관계형 데이터베이스의 거의 대부분의 SQL 지원이 가능하며, 데이터 수집과 적재만 잘 설계 했다면, 분산 환경에서 기존의 DW 기반의 BI 개발자들도 손쉽게 사용할 수 있으며, SQL 에서 제공하는 다양한 함수 및 Optimizer 등이 거의 대부분 적용되어 확장성과 유지보수성이 뛰어납니다.

	SQL	HIVE QL
Language	SQL-92 Standard	SQL-92 표준을 거의 따르며 추가적인 Hive Spec
CRUD	INSERT, UPDATE and DELETE	INSERT, DELETE and UPDATE (ORC, Delta)
Transaction	Yes	Yes (ORC, Delta)
Latency	Sub-second	Minutes or more
Indexes	Any number of indexes	Second order indexes (ORC, Parquet)
Data size	TBs	PBs
Data per query	GBs	PBs

https://hevodata.com/learn/hive-sql/

Tools for Relational Database

관계형 데이터베이스 수집을 위한 도구

Database Client, Embulk, Sqoop, Spark JDBC Connector

오픈소스 수집도구 - Database Client

sql*Plus, iSQL, pSQL

개별 클라이언트 설치가 필요하고, 수집 도구로 제공되는 툴이 아니다 보니 대용량 처리나 운영 차원에서 어려움이 발생할 수 있습니다

다만, 이미 설치되어 있거나, 부가적인 설정이나 인프라를 설치 하기에는 운영 및 관리 부담 되는 경우, 클라이언트 모듈만 빠르게 설치해서 사용하는 경우 추천

iSQL - https://stackoverflow.com/questions/4355080/how-to-output-data-from-isql-to-csv-file-with-headings

$> isql <DATABASE> <USERNAME> <PASSWORD> -b -d<DELIMITER> -q -c<COLUMNS>

SQL*Plus - https://chartio.com/resources/tutorials/how-to-write-to-a-csv-file-using-oracle-sql-plus/

```
set colsep ,
spool books.csv
SELECT title, primary_author FROM books;
```

PostgreSQL - https://stackoverflow.com/questions/1517635/save-pl-pgsql-output-from-postgresql-to-a-csv-file

$> Copy (Select * From foo) To '/tmp/test.csv' With CSV DELIMITER ',' HEADER;

오픈소스 수집도구 - Embulk

TreasureData Embulk

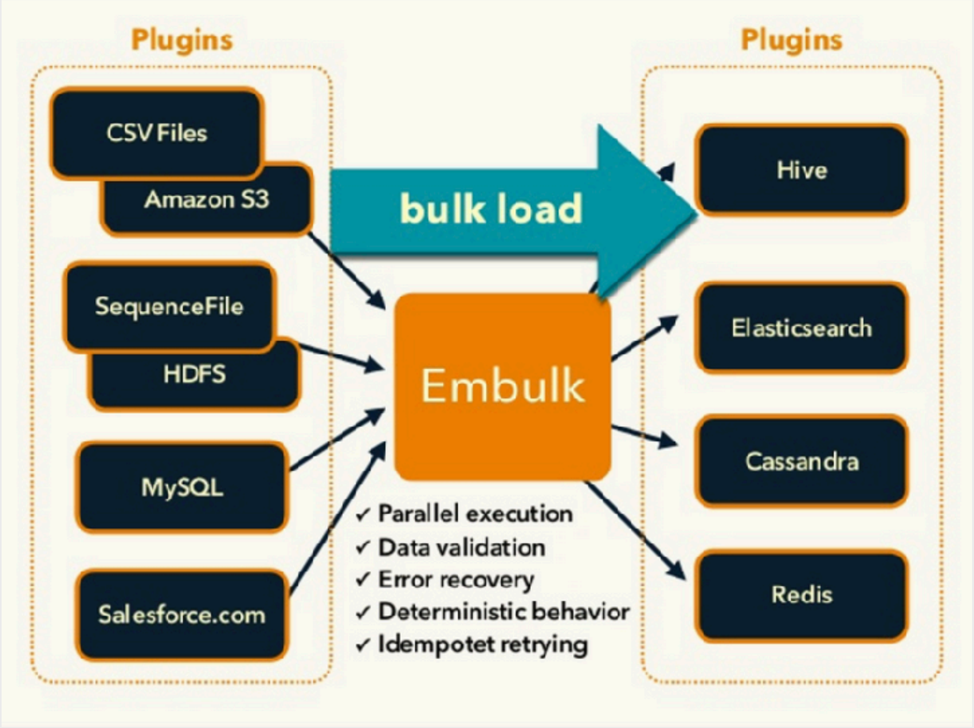
fluetnd 와 유사한 인터페이스와 아키텍처, 플러그인이 다양하여 데이터베이스 뿐만 아니라 bulk 적재에 컨셉이나 별도의 클러스터를 구성해야 하고 운영 비용이 들고, 수집 단위로 yml 설정이 필요한 점이 단점, 왠만한 상용 데이터 베이스를 지원하고 있어 sqoop 대비 큰 차이는 없음 (최근 몇 년간 지속적인 개발이 되고 있음), 대부분의 엔진이 ruby 로 개발되어 있어 운영 및 트러블 슈팅에 언어적인 제약이 있을 수 있습니다

Pros

* 테이블 뿐만 아니라 다양한 데이터 소스를 지원 (file, nosql, rdb 등)
* fluentd 와 유사한 인터페이스를 가지므로 통합 관리 시에 유리할 수 있음

Cons

* Sqoop 에 비해 널리 알려져 있지 않아 안정성 면에서 확인이 필요하다
* 기존 시스템과 연동 혹은 통합하기가 까다롭다 (독립적인 제품으로는 유용함)



오픈소스 수집도구 - Sqoop

Apache Sqoop

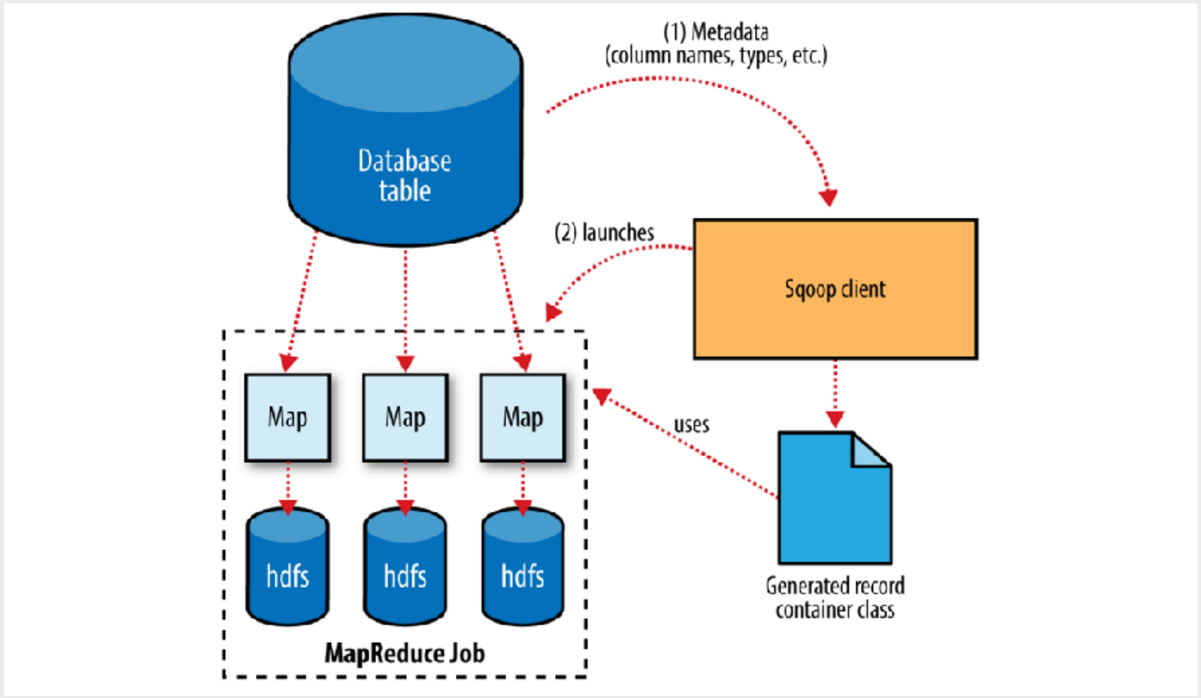
커맨드라인 도구로 사용 및 오픈소스 관계형 데이터베이스 수집이 가능, 가장 널리 사용되는 오픈소스이며, 하둡 맵리듀스를 통해 분산 처리가 가능하기 때문에 설치가 간편하고, 운영 비용이 적다 (최근 거의 업데이트가 없음, 안정적이라고 볼 수도..), 최근 NoSQL 에 대한 수집의 요구사항도 늘어나고 있어 관계형 테이블에 대해서만 사용이 가능한 점은 아쉽습니다. 하지만 가장 오래 되었고 안정적이며, 하둡 에코시스템 환경에서 가장 적합한 도구입니다

Pros

* 상용 DB 뿐만 아니라 대부분의 RDB 수집 지원
* 관계형 데이터 수집 (import), 적재 (export) 가 가능한 도구
* MapReduce 기반의 병렬 분산처리가 가능
* Parquet 및 다양한 저장 및 Hive 연동 가능

Cons

* 관계형 데이터베이스 수집만 가능함
* Incremental 수집의 기능은 약함



오픈소스 수집도구 - Spark

Apache Spark JDBC Connector

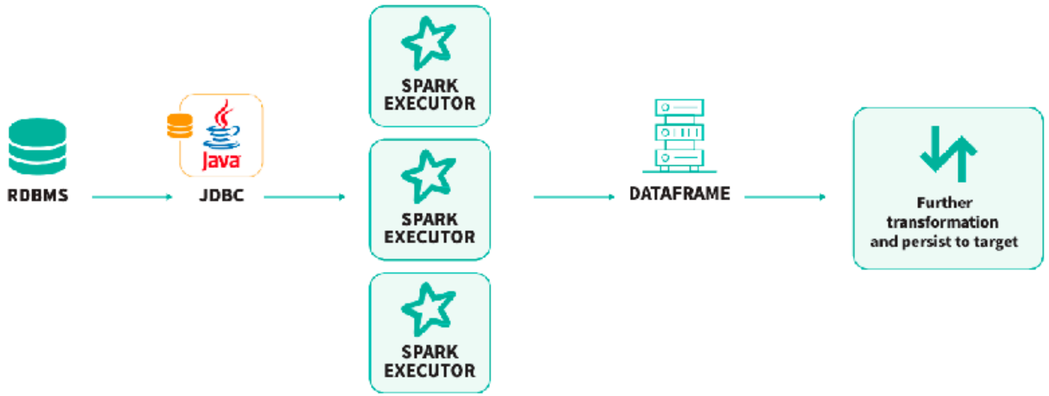
스파크를 사용할 수 있는 환경에서 별도로 설치 과정 없이 가장 빠르게 적용할 수 있다는 장점이 있고, 수집하고 적재하는 과정도 프로그램으로 관리할 수 있어 기존의 레거시 어플리케이션과 연동도 쉽다. 다만, 너무 많은 커넥션을 맺는 경우 데이터베이스에 부하를 줄 수 있고, 수집 대상 테이블이 늘어나는 경우 SQL 을 따로 관리해야 하거나 프로그램을 따로 배포해야 하는 상황이 발생할 수 있어 번거로울 수 있으며, 스파크 프로그래밍에 대한 학습과 이해도 필요하다는 점이 허들이다. 기존 어플리케이션과 연동을 하거나, 스파크 프로그래밍에 익숙하고, 관리 대상 테이블이 충분히 많지 않은 경우 2~30 개 미만인 경우 아주 편하게 사용할 수 있으며, NoSQL 과 같은 경우도 쉽게 연동이 가능하여 추천할 만합니다

Pros

* YARN 환경에서 손쉽게 적용할 수 있고 성능도 준수합니다
* 레거시 어플리케이션과 쉽게 연동 및 메타데이터 관리가 가능합니다

Cons

* 스파크 프로그래밍에 대한 이해와 학습이 필요합니다
* 수정 시에 프로그램 빌드 및 배포의 과정이 필요할 수 있습니다



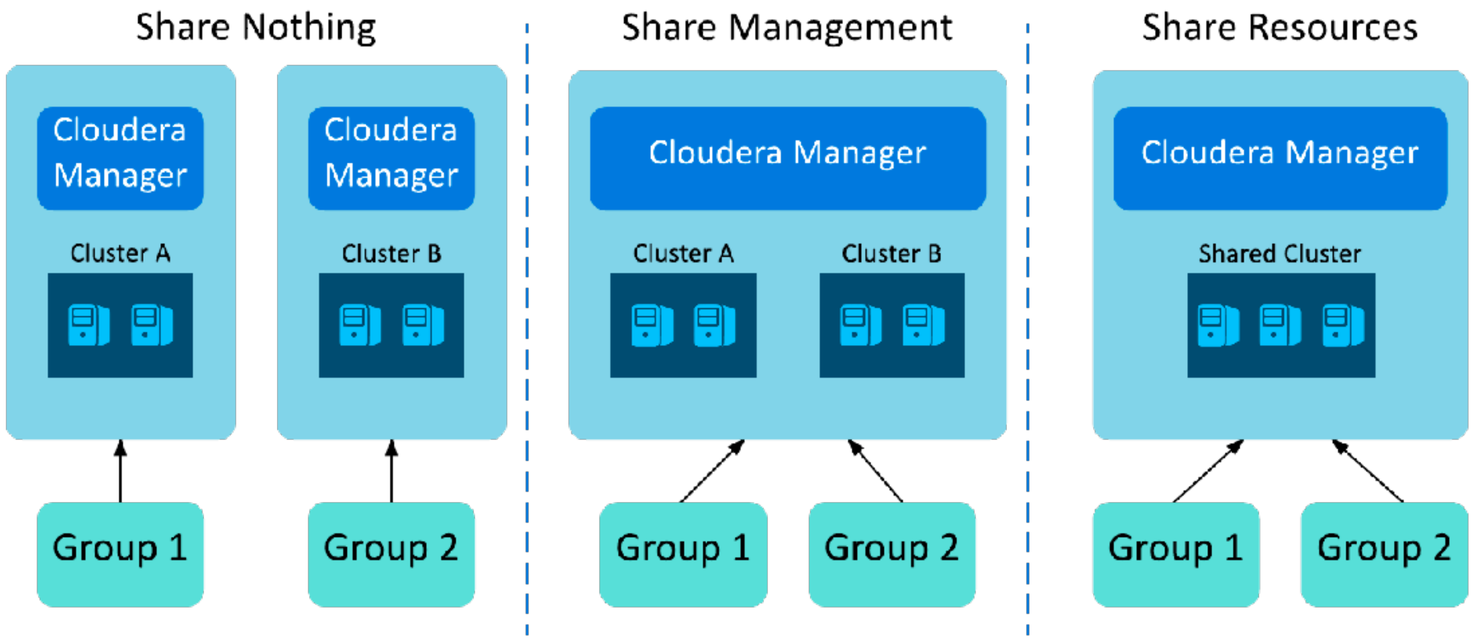
Building Ingestion Architecture

관계형 데이터베이스의 수집 아키텍처 설계의 기초

데이터 수집 아키텍처 설계 #1

Cluster Isolation

On-premise 레거시 데이터베이스에 접근하는 것은 여러가지 위험 요소가 있으며, 라이브 서버에 영향을 줄 수도 있기 때문에 수집을 위한 격리된 환경의 클러스터의 설계를 해야할 수도 있습니다. 특히 ACL 문제나 접속 커넥션 수 등의 리소스 관리가 특히 중요합니다. 일반적으로 라이브 서버에 영향을 최소화 하기 위해 복제 디비를 구성하거나 데이터베이스 사용량이 적은 시간 대에 스케줄을 거는 제약이 필요하며, 테이블 조회 시에도 with (nolock) 등의 옵션을 통해 동시성 문제가 없도록 가이드 해야만 합니다. ACL 의 경우 대상 클러스터에와 정해진 포트에 대해서만 접근이 가능하도록 Whitelist 관리가 되어야 합니다.



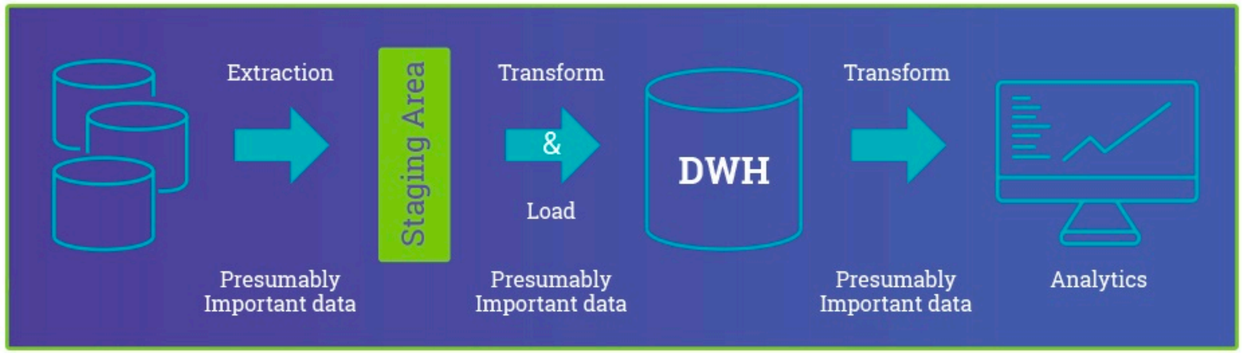
데이터 수집 아키텍처 설계 #2

Data Modeling - Schema on Write vs. Read

하둡과 같은 분산 처리 시스템에서는 주기적으로 스냅샷 데이터를 적재하고 이를 분산 처리 플랫폼을 통해서 데이터를 가공하고 적재하는 방식을 취하고 있습니다. 모델링 관점에 있어 기존 데이터베이스와 가장 큰 차이점은 스키마 정의 및 적용 시점입니다. 관계형 데이터베이스의 경우 저장 시에 모든 스키마 Validation 및 제약조건을 걸지만 하둡의 경우 대용량 데이터를 처리하는 데에 저장 시에 Validation 은 Throughput 에 너무 큰 영향을 주기 때문에 일단 모든 데이터를 저장하고 난 이후에 변환을 통해 스키마를 조정하거나, 읽는 시점에 Casting 을 하는 경우가 더 효과적이라고 말할 수 있습니다.

Schema on Write

* 데이터를 스테이징 하고 변경 가능성이 적은 경우
* 데이터 수집 이후에 즉시 사용이 필요한 경우
* 데이터 스키마가 이후 ETL 작업에 의존성이 큰 경우



https://blog.panoply.io/etl-vs-elt-the-difference-is-in-the-how

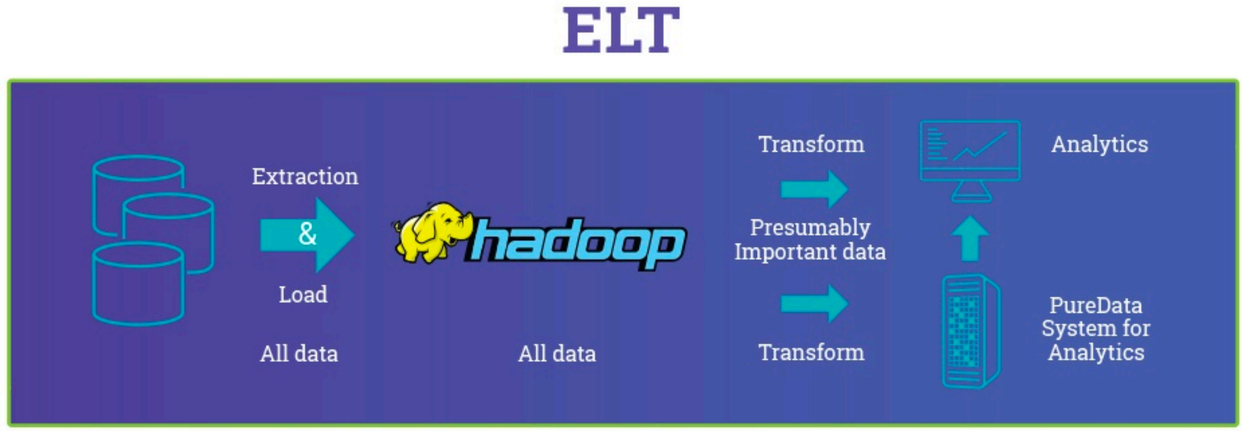
데이터 수집 아키텍처 설계 #2

Data Modeling - Schema on Write vs. Read

이는 구조상의 차이점이라기 보다는 데이터 처리의 방식이나 성격에 따라 선택할 수 있는 전략에 가깝습니다. 원본 데이터를 손실 없이 그대로 저장하는 것 자체에 의미가 있다면 ELT 가 적절하고, 매번 데이터 변환 과정을 개별 처리 수준에서 해야 한다면 미리 해두는 ETL 과 같은 전처리를 하면서 적재하는 방식이 더 효과적이라 말할 수 있습니다

Schema on Read

* 데이터 저장 이후에 가공 및 변환이 필수인 경우
* 스키마를 저장시에 고려하기에는 너무 큰 경우
* 스키마의 정밀도가 데이터 분석에 큰 영향이 없는 경우



https://blog.panoply.io/etl-vs-elt-the-difference-is-in-the-how

데이터 수집 아키텍처 설계 #3

Data Ingestion - Idempotence

장애를 빠르게 복구하는 방법은 ETL 전반에 걸쳐서 항상 검토되어야 하는 요소 중에 하나인데, 특히 데이터 수집, 변환 및 적재의 **멱등성(idempotence)**을 유지하는 데이터 프로세싱은 설계하는 것이 가장 중요합니다. 데이터 ETL 작업의 10~20% 빠른 것 보다 장애 시에 정확하게 복구 되는 것을 보장하는 것이 더 중요하며, 매번 수행 시 마다 다른 결과가 나온다면 해당 데이터를 근간으로 하는 지표의 신뢰성은 떨어질 수 밖에 없습니다. 특히 데이터 수집의 경우 스냅샷을 저장하는 경우가 많기 때문에 항상 멱등하지 않은 경우가 많기 때문에 고려해 두면 좋을 만한 사항들이 몇 가지 있습니다.

Idempotence of Data Ingestion

* 데이터 소스에 Timestamp 를 가지고 있다면 해당 필드를 Where 절로 파티셔닝 하여 수집할 것
* 하나의 논리 테이블에 여러 파티션을 가져오는 경우에는 최대한 병렬로 수집하여 시점을 일치시킬 것
* 적재 시점을 정확히 알 수 있도록 반드시 LOAD_DATE 와 같은 sysdate 컬럼을 유지할 것
* 일부 파티션 장애 시에는 유관부서와 협의 하에 필요한 경우 전체 파티션 범위를 복구할 것
* 가능하다면 별도의 형상관리가 가능한 저장소에 수집을 위한 메타데이터를 관리할 것
* 수집에 관련된 모든 상태를 별도의 메타데이터를 통해 저장관리하고 모니터링 할 수 있을 것

https://en.wikipedia.org/wiki/Idempotence

Q&A