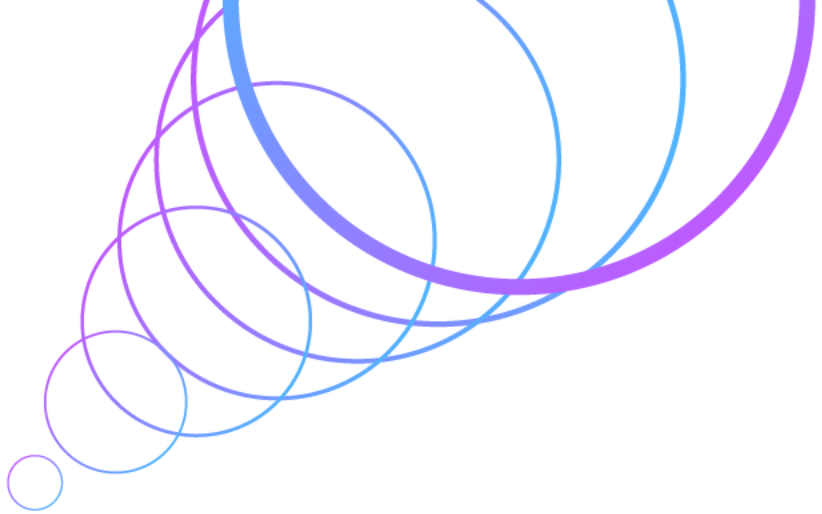
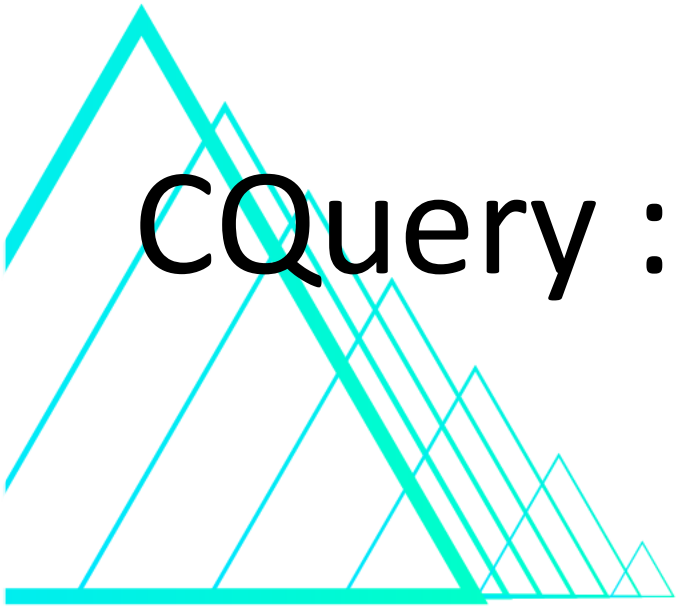
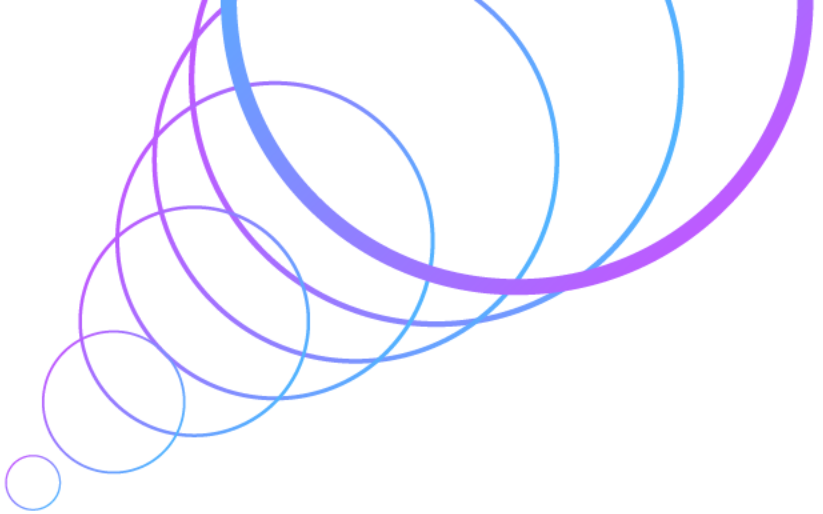


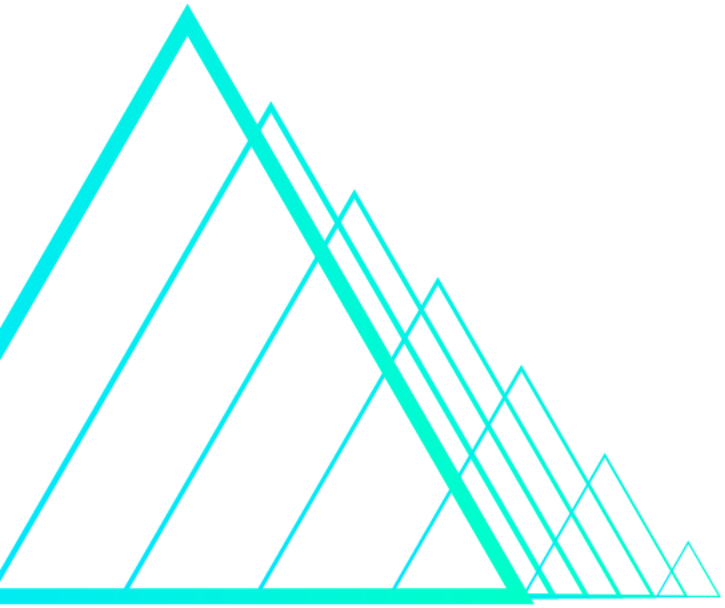


CQuery：Easy and Fast SQL-Based Analytics Solution

정의근, 신선우 AI & Data Platform

CONTENTS





1. CQuery Intro

1.1 CQuery

네이버 서비스에서 발생하는 대용량의 로그를

SQL로 조회, 분석하는 환경을 제공하는

멀티테넌트 플랫폼 & 통합 솔루션

1.2 CQuery History

2018 : SQL Interface(검색 로그 활용 시스템) 구축
Hive 을 포함한 Hadoop Eco Systems 기반

2019 : DEVIEW 2019 발표
https://deview.kr/2019/schedule/300
네이버 로그를 지탱하는 힘 (DataStore 로그 저장소)

2020.10: CQuery 로 탄생
Hue를 대체하는 SQL 웹 인터페이스 독자 개발

2021.05: Trino 도입

1.2.1 Future

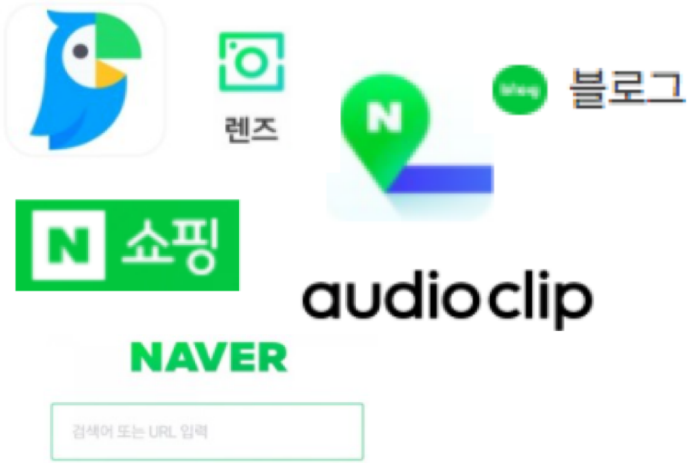
- 글로벌 서비스 레벨로

1.3 네이버 로그

필수 분석 대상입니다.

사용자 반응의 바로미터

새로운 서비스의 시작



대용량입니다.

모바일 통합검색 로그 수 일 수십억 건,

사이즈 일 수백 GB (압축된 ORC 파일 기준)

➢ CQuery를 통해서 대용량의 로그를 안정적으로 분석 가능

1.4 CQuery를 찾는 사람들

서비스 기획자 서비스 개발자 Data Scientist 검색 모델러 경영진 …

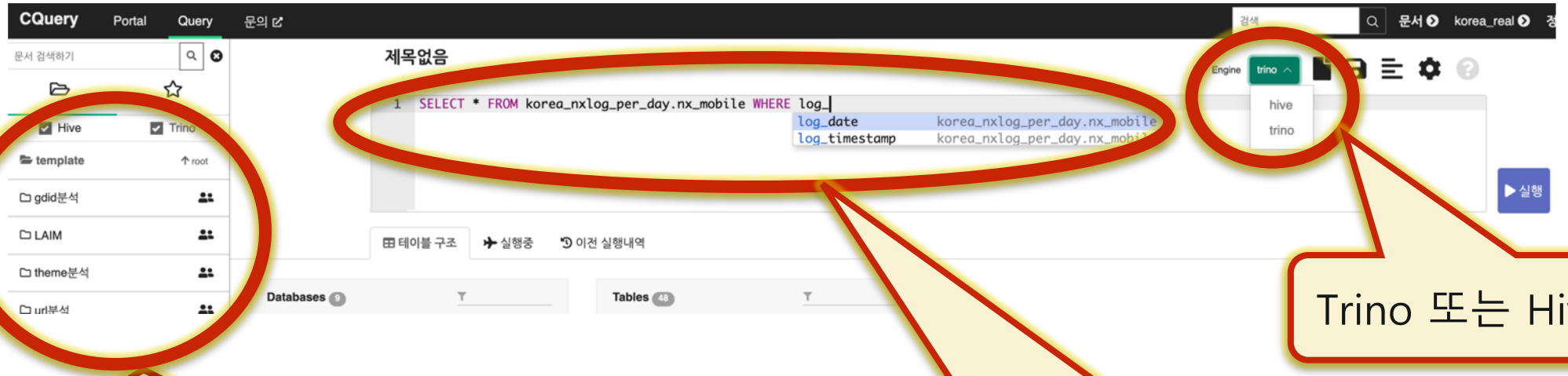
로그 분석을 원하는 모든 사람들

- 다수의 사용자가 동시에 사용하는 멀티테넌트 환경
- 개발자가 주로 사용하는 언어 대상으로 개발 툴 제공
- 비개발자가 사용하기 쉽도록 웹 인터페이스 제공

1.5 CQuery Web Interface

1.5 CQuery Web Interface

SQL 편집과 실행



SQL을 작성하기 어려워하는 사람들을 위한 네이버 로그 분석용 SQL Template 묶음

SQL 자동 완성, 실시간 SQL 에러 체크 등을 제공하는 SQL Editor

Trino 또는 Hive 선택 가능

1.6 Develop Tool

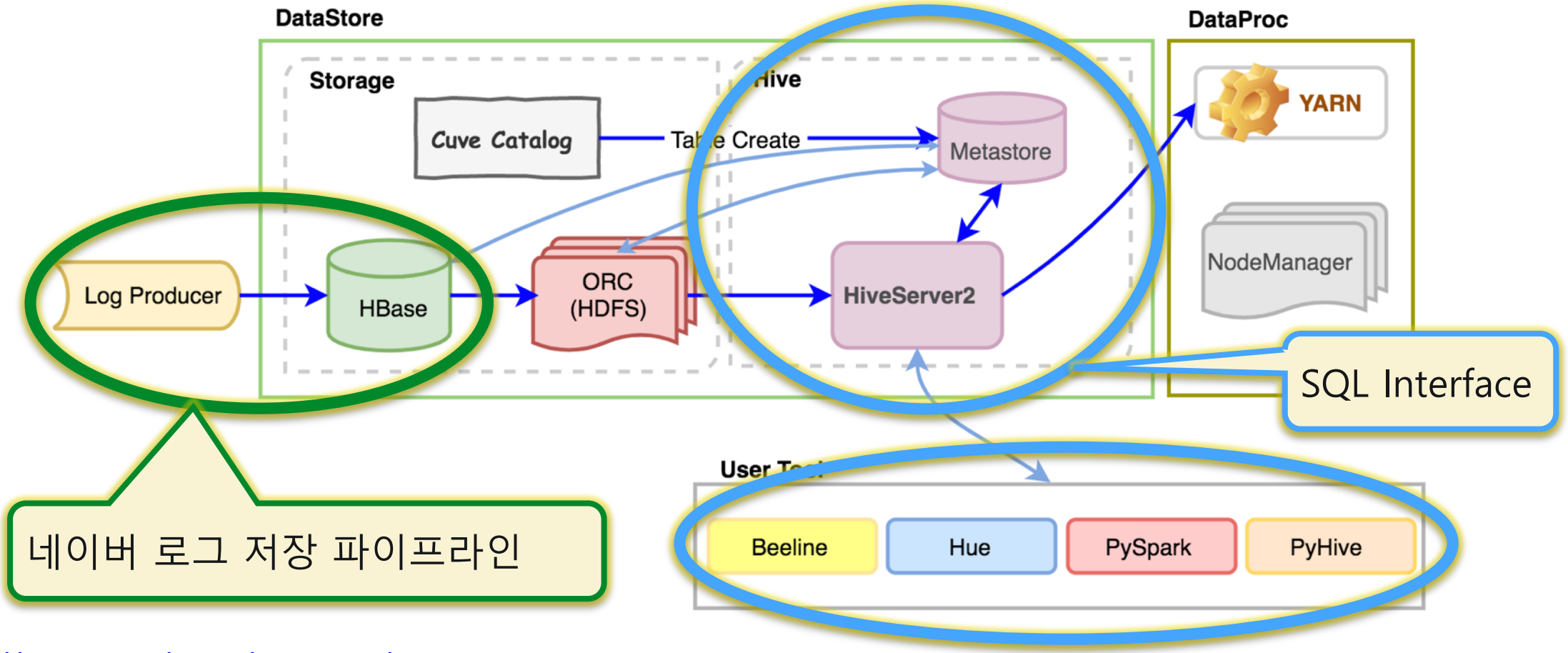
Command Line Interface(CLI)

JDBC Driver for Java

Python Driver
  https://github.com/naver/pycquery

Go Driver
  오픈 준비중

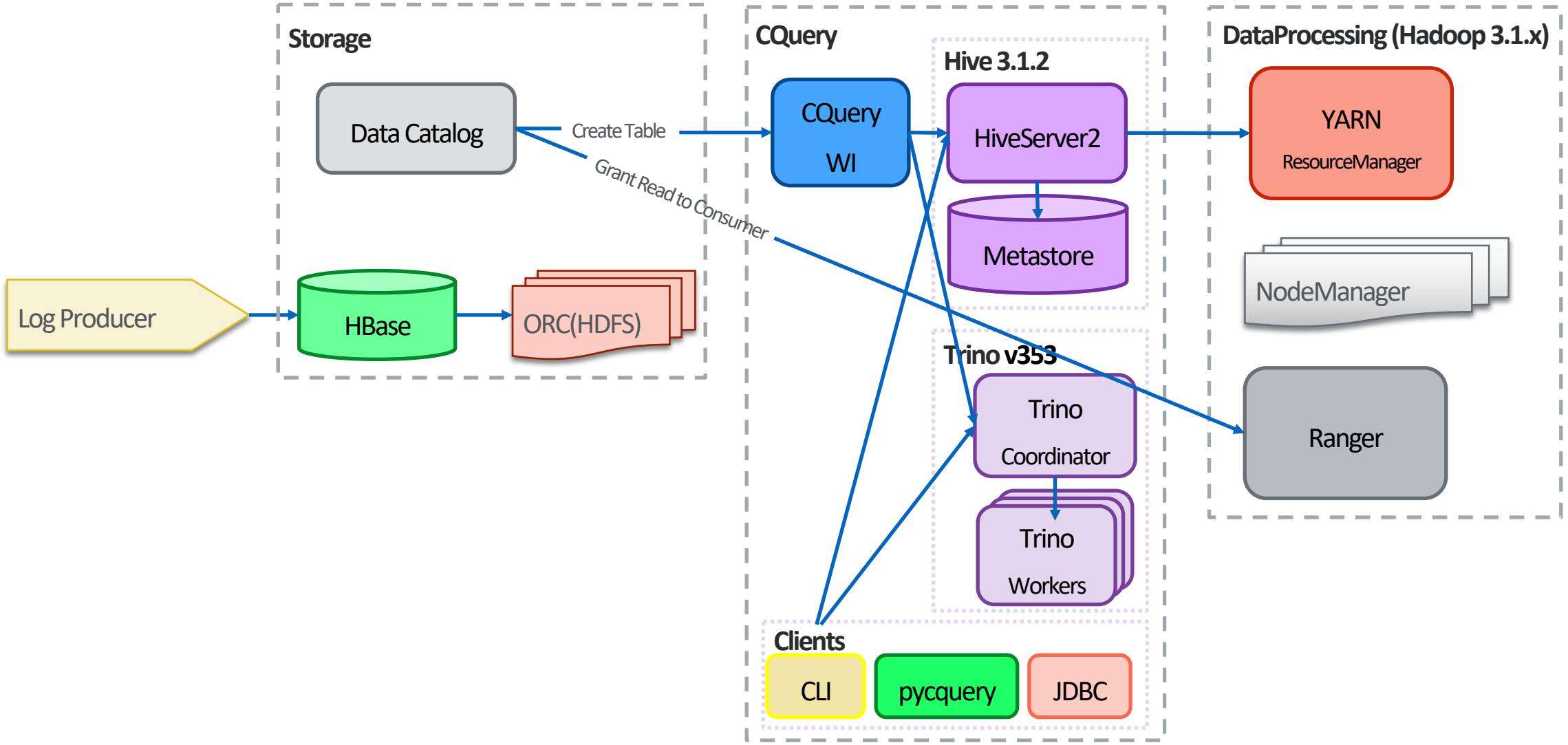
1.7 SQL Interface (CQuery 이전) 구성 요소



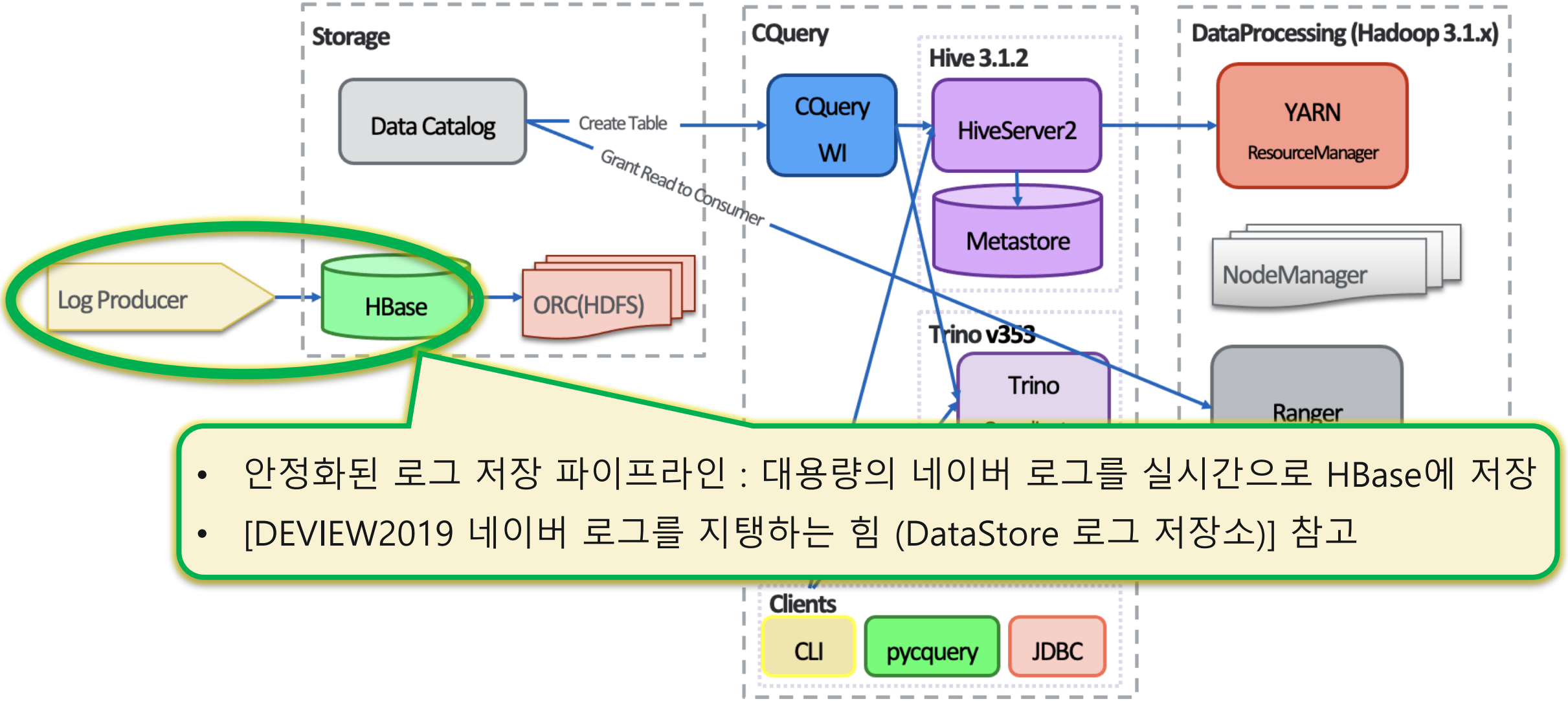
https://deview.kr/2019/schedule/300

DEVIEW2019 네이버 로그를 지탱하는 힘 (DataStore 로그 저장소)

1.8 CQuery 구성 요소

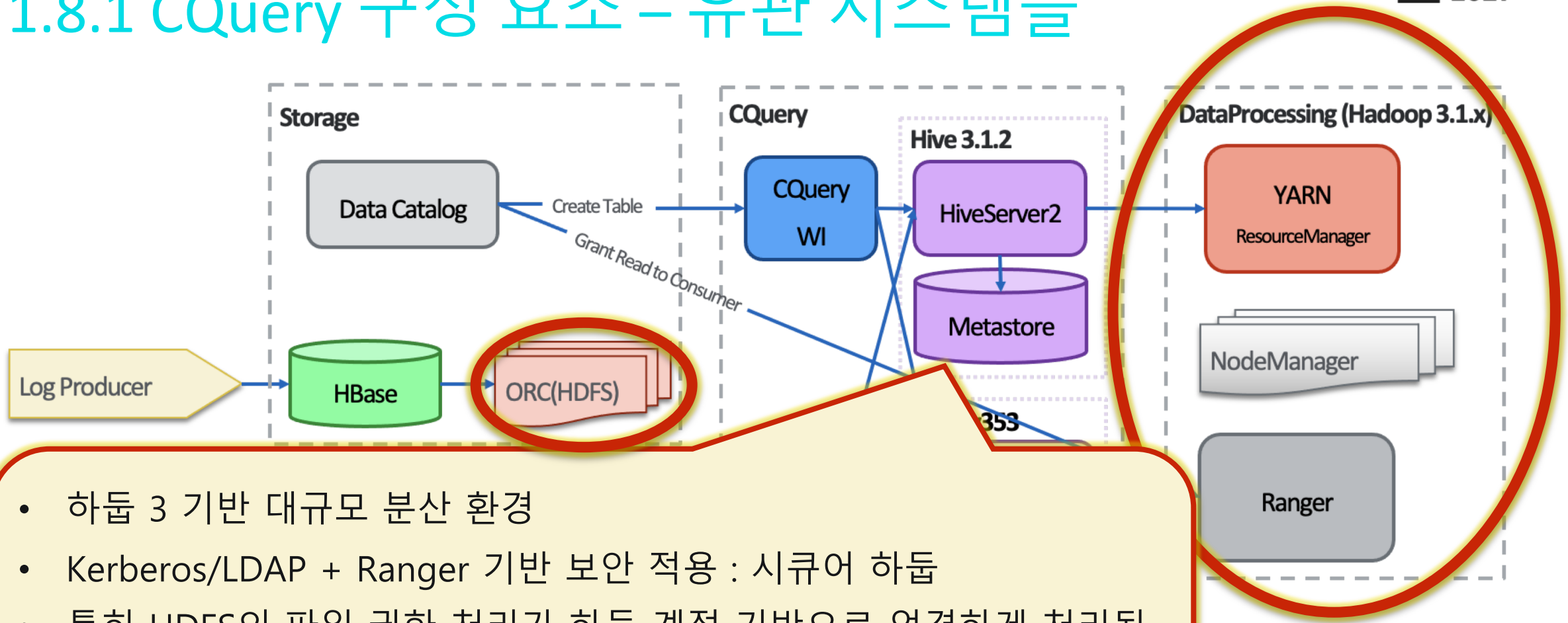


1.8.1 CQuery 구성 요소 – 유관 시스템들



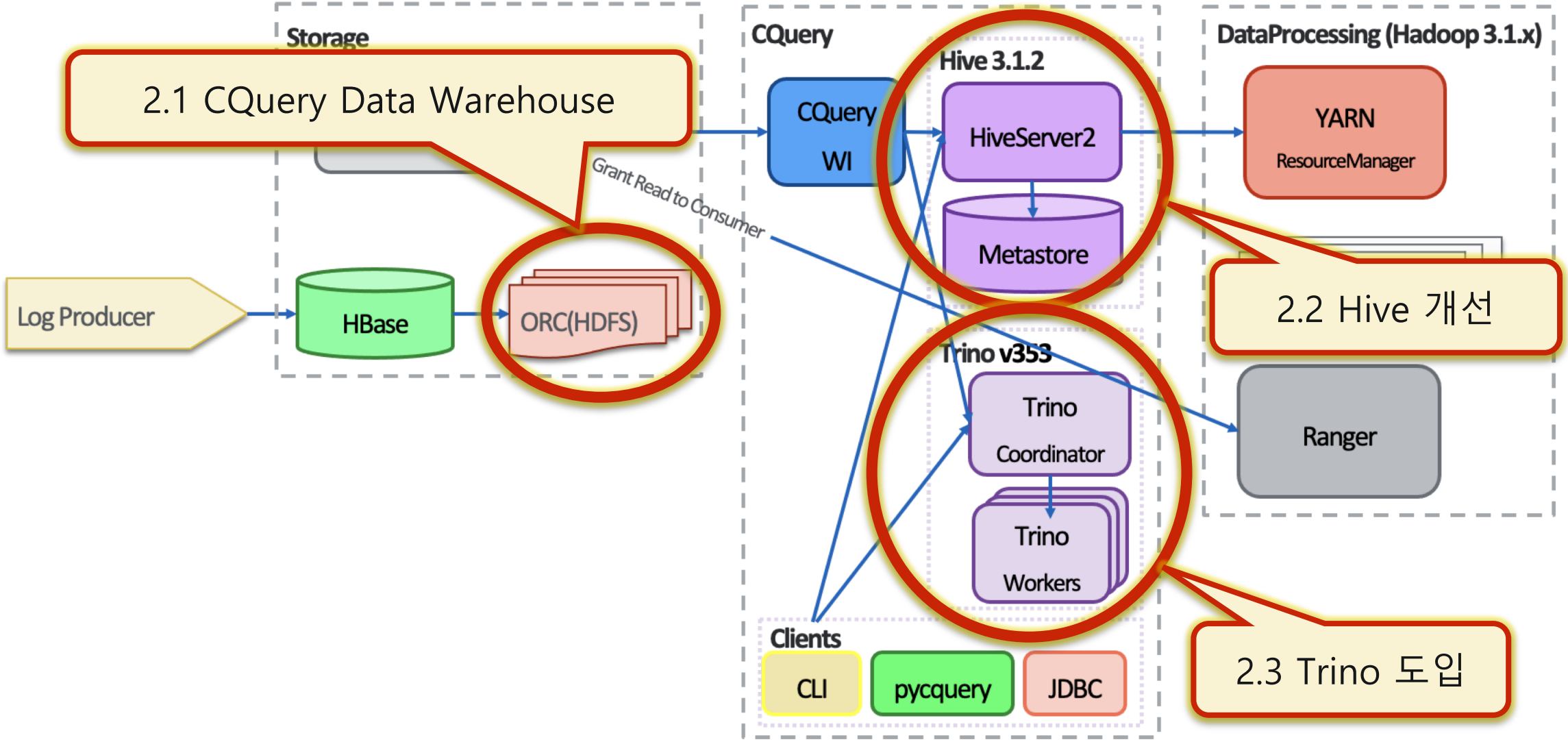
- 안정화된 로그 저장 파이프라인 : 대용량의 네이버 로그를 실시간으로 HBase에 저장
- [DEVIEW2019 네이버 로그를 지탱하는 힘 (DataStore 로그 저장소)] 참고

1.8.1 CQuery 구성 요소 – 유관 시스템들

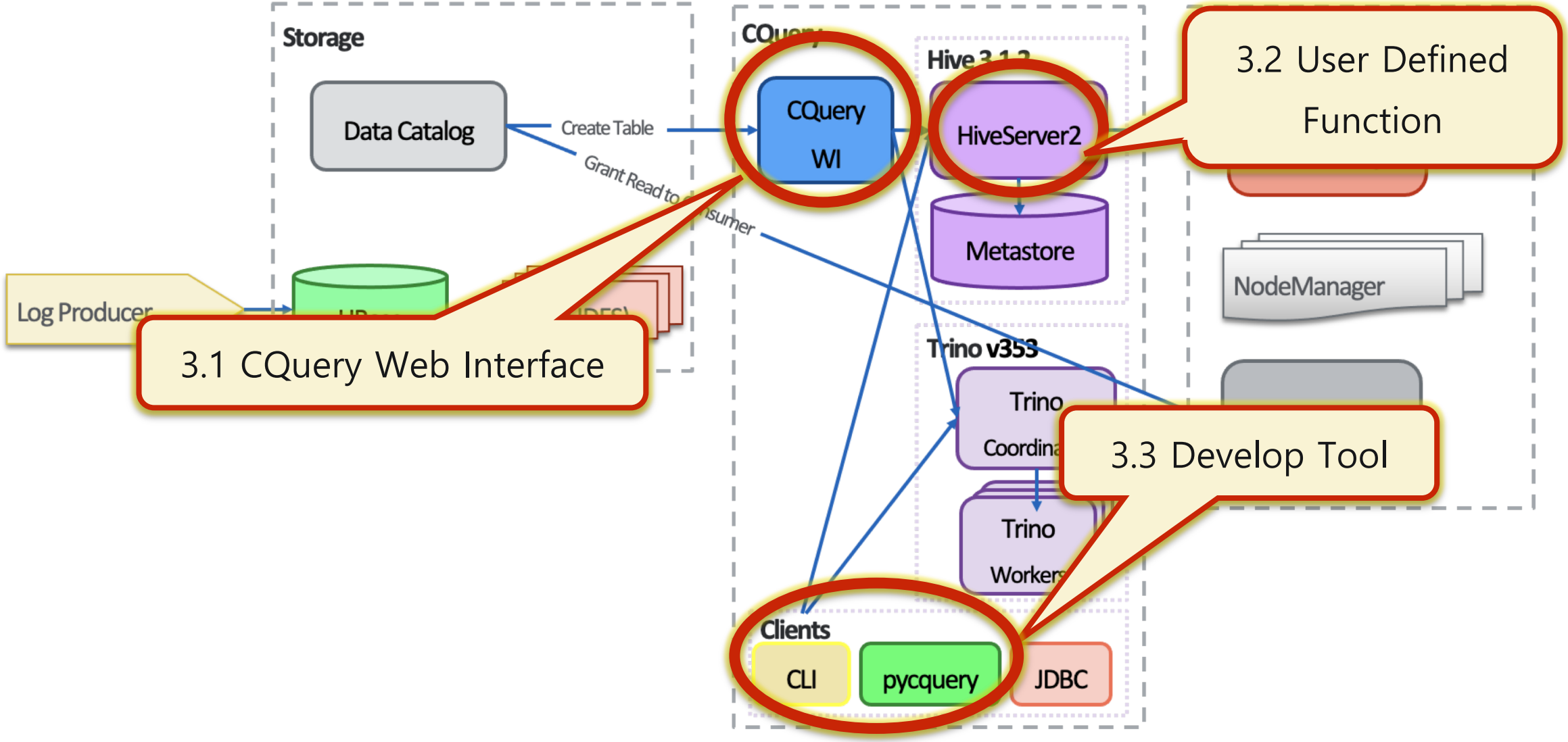


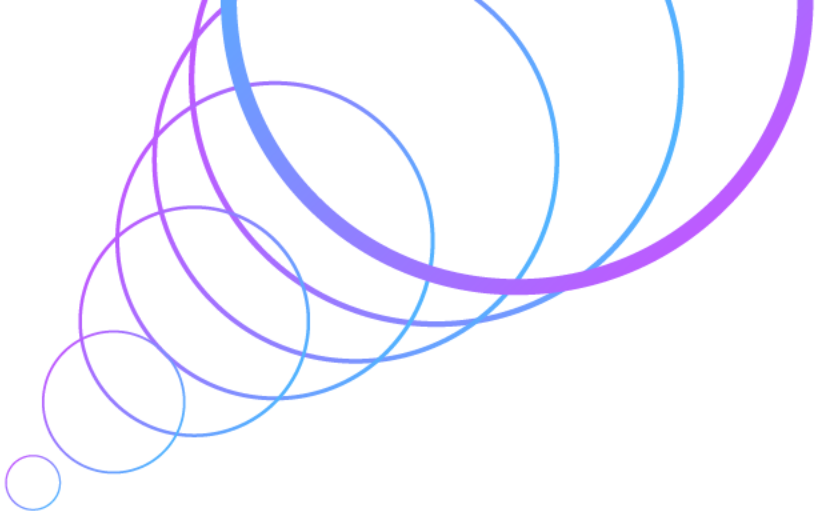
- 하둡 3 기반 대규모 분산 환경
- Kerberos/LDAP + Ranger 기반 보안 적용 : 시큐어 하둡
- 특히 HDFS의 파일 권한 처리가 하둡 계정 기반으로 엄격하게 처리됨
- [DEVIEW2019 대용량 멀티테넌트 시큐어 하둡 클러스터를 시행착오 없이 만들기] 참고

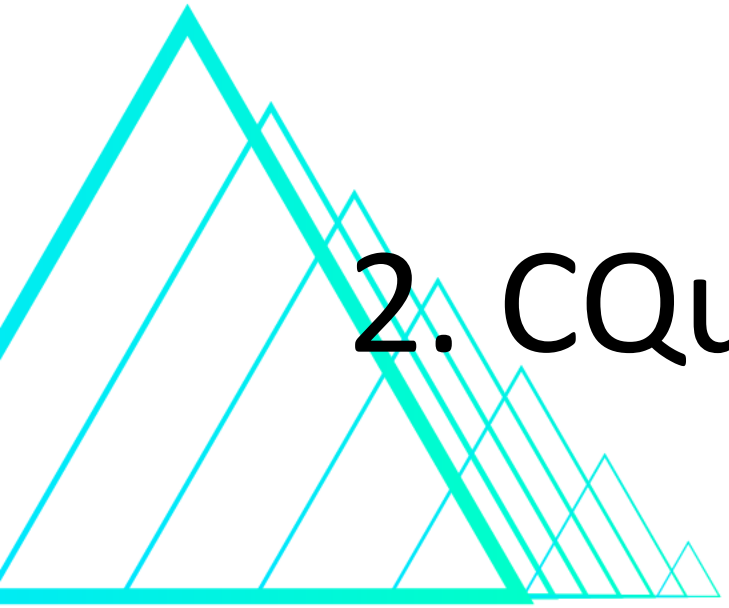
1.8.2 CQuery 구성 요소



1.8.2 CQuery 구성 요소





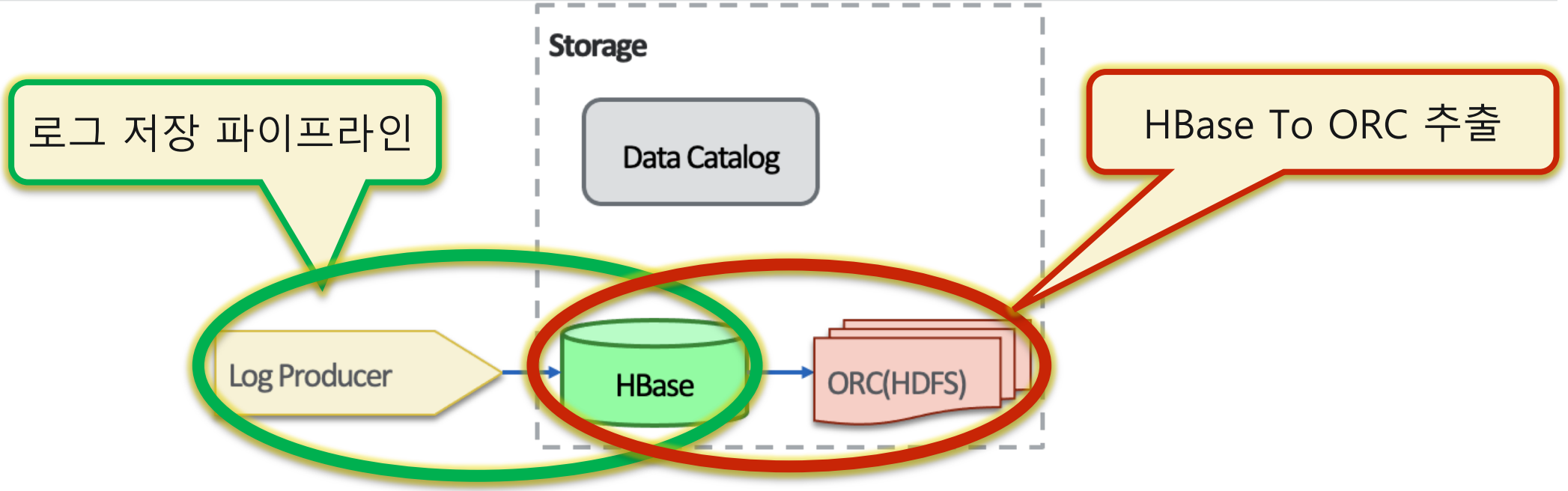


DEVIEW 2021

2. CQuery Internal (Feat. Fast)

2.1 CQuery Data Warehouse

- HDFS에서, 대용량 로그 분석에 적합한 ORC 파일들로 구성됨
- HBase에 저장된 로그를 10분/1시간/1일 단위로 ORC 파일 추출 수행
- 자체적으로 Business Analytics Warehouse(BAW)라고 명명



2.1.1 Business Analytics Warehouse (BAW)

- 비즈니스 가치가 높은 데이터들을 분석이 용이한 형태로 가공한 데이터 창고
- 간단한 가공을 거쳐서 HBase 저장 후 2, 3차 가공 로그 생성
- BAW의 관리 대상은 2, 3차

1차

- 원본 로그 (HBase)
- 기본적인 파싱만 처리
- 실시간 저장
- SQL 속도가 느림

2차

- 가공 로그 (ORC)
- 1차 로그를 ORC 파일로 저장
- 비실시간 생성
- SQL 속도가 빠름

3차

- Summary 로그 (ORC)
- 2차 로그들을 활용해 분석에 자주 사용되는 패턴에 따라 요약해서 저장

2.1.2 BAW 3차 가공 로그의 효과

DEVIEW 2021

대용량 로그 테이블 간의 JOIN 필요

2차 : 원본로그 ORC Ta

노출로그

sscode	query
tab.m.all	bts
tab.m_video.all	bts
tab.m_image.all	bts
tab.m.all	bts
⋮	
tab.m.all	bts

노출수 sum
N개

클릭수 sum
M개

클릭로그

sscode	query
tab.m.all	bts
tab.m_kdic.all	bts
tab.m.all	bts
tab.m.all	bts
⋮	
tab.m.all	bts

```
SELECT nx.query, nx.qc , cr.cc
FROM
  (
   SELECT query, count(*) qc                                  노출로그
   FROM
     korea_nxlog_per_day.nx_mobile
   WHERE log_date='2021-06-20' AND sscode='tab.m.all' AND query='bts'
   GROUP BY query
  ) nx
LEFT JOIN (
   SELECT query, count(*) cc
   FROM                                                       클릭로그
     korea_nxlog_per_day.cr_mobile
   WHERE log_date = '2021-06-20'AND sscode='tab.m.all' AND query='bts'
   GROUP BY query
  ) cr ON (nx.query = cr.query)
```

#	query		cc
1	bts	10,000	20,000

3차 : Summary Table

Summary Table

sscode	query	노출수 sum	클릭수 sum
tab.m.all	bts	10,000	20000
tab.m_kdic.all	bts	300	600
tab.m_video.all	bts	200	400
tab.m_image.all	bts	100	200

```
select query, sc, cc from korea_nxlog_per_day.ctr_query
where log_date='2021-06-20'
and query='bts'
and sscode='tab.m.all'
```

분석 summary table

#	query	sc	cc
1	bts	10,000	20,000

작은 사이즈의 테이블 조회로 동일한 결과 얻음

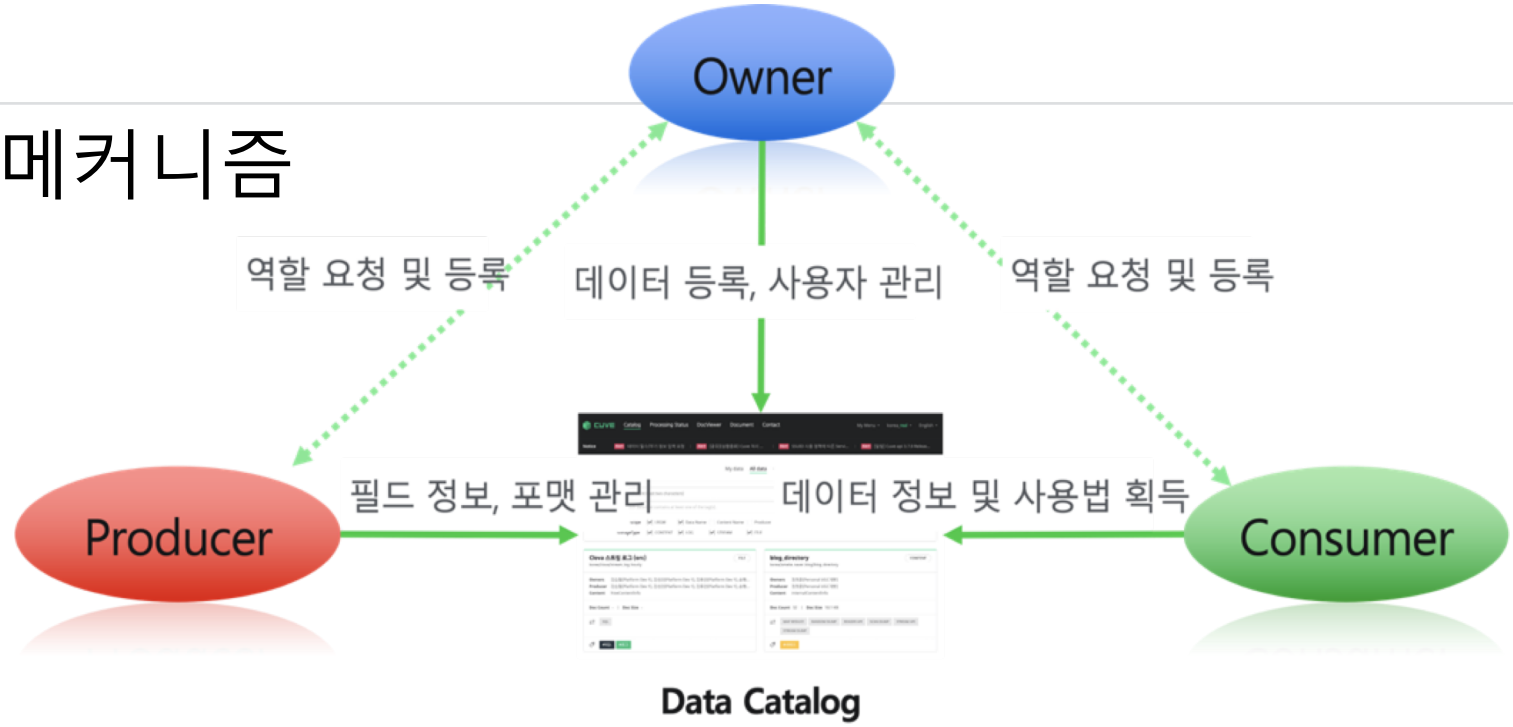
2.1.3 BAW의 3차 가공 주체 확장

- 현재 분석 니즈가 가장 많은 검색 서비스 로그에 대해서 CQuery 팀에서 직접 가공해서 제공
- 다른 수많은 서비스들의 로그는 어떻게 3차 가공할 것인가?
- CQuery 팀에서 모든 서비스의 분석 니즈를 파악해서 3차 가공 로그를 제공하는 것은 불가능
- CQuery Flow라는 이름으로, 사용자들이 2, 3차 가공 로그를 만들 수 있는 서비스 오픈 준비중

2.1.4 BAW에서의 데이터 접근 제어

DEVIEW 2021

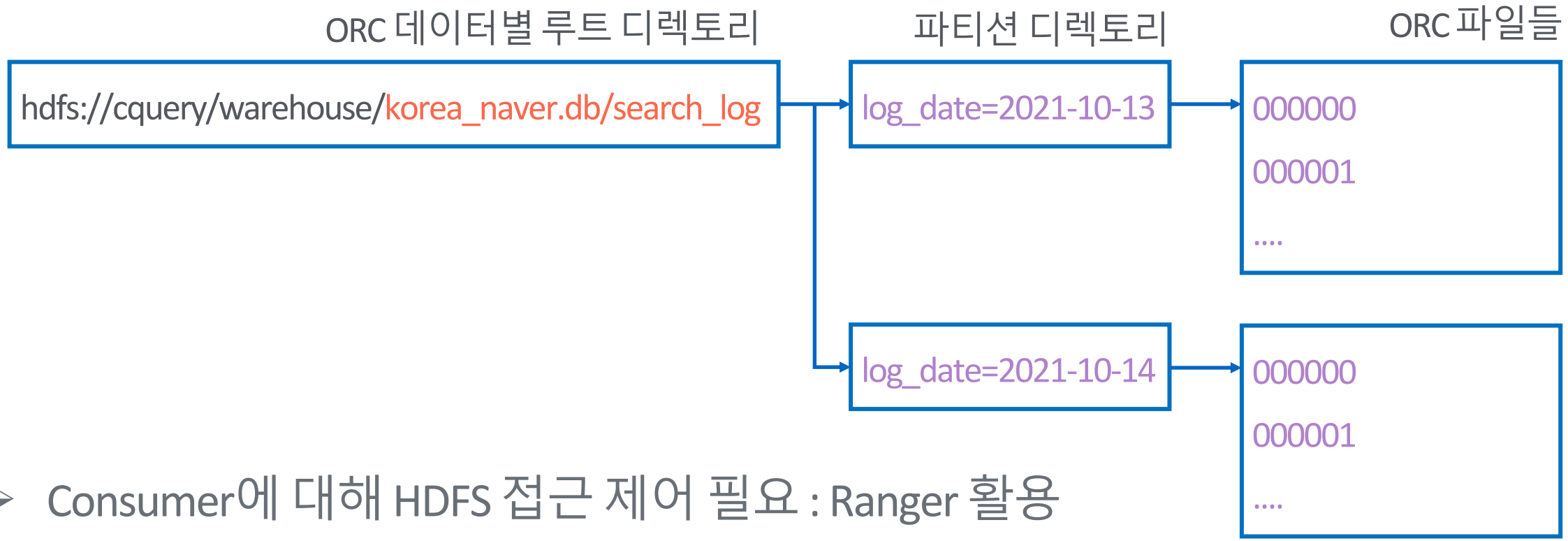
Catalog 기반 데이터 관리 메커니즘



- 데이터를 안전하고 합법적으로 사용할 수 있도록 하기 위한 메커니즘
- 자세한 내용은 DEVIEW2021 AIDA Project: Enterprise Data Governance and Access Management 참고
- BAW에서는 HDFS 상의 ORC 파일들에 대한 접근 제어 필요

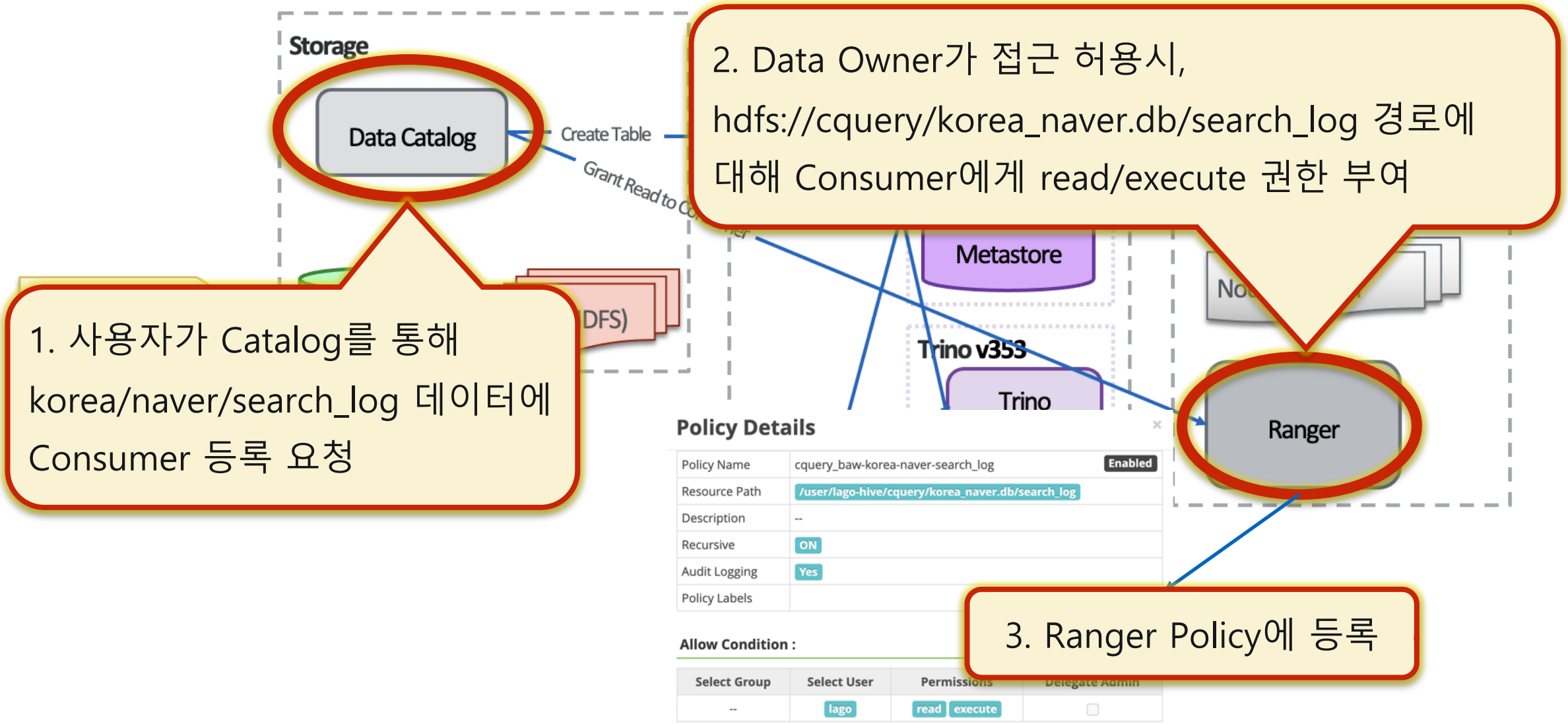
2.1.4.1 BAW와 HDFS

- Data Catalog에 등록된 데이터의 예 : korea/naver/search_log
- HDFS에서의 데이터 형태



➢ Consumer에 대해 HDFS 접근 제어 필요 : Ranger 활용

2.1.4.2 Ranger Policy 기반 데이터 접근 제어



2.2 Hive 개선

DEVIEW 2021

리소스 제어

- Hive Session 제어
- Abuse 성격의 SQL 방지

성능 개선

- Bucket Pruning 성능 개선
- Tez AM VCore 조절

2.2.1 리소스 제어 - Hive Session 제어

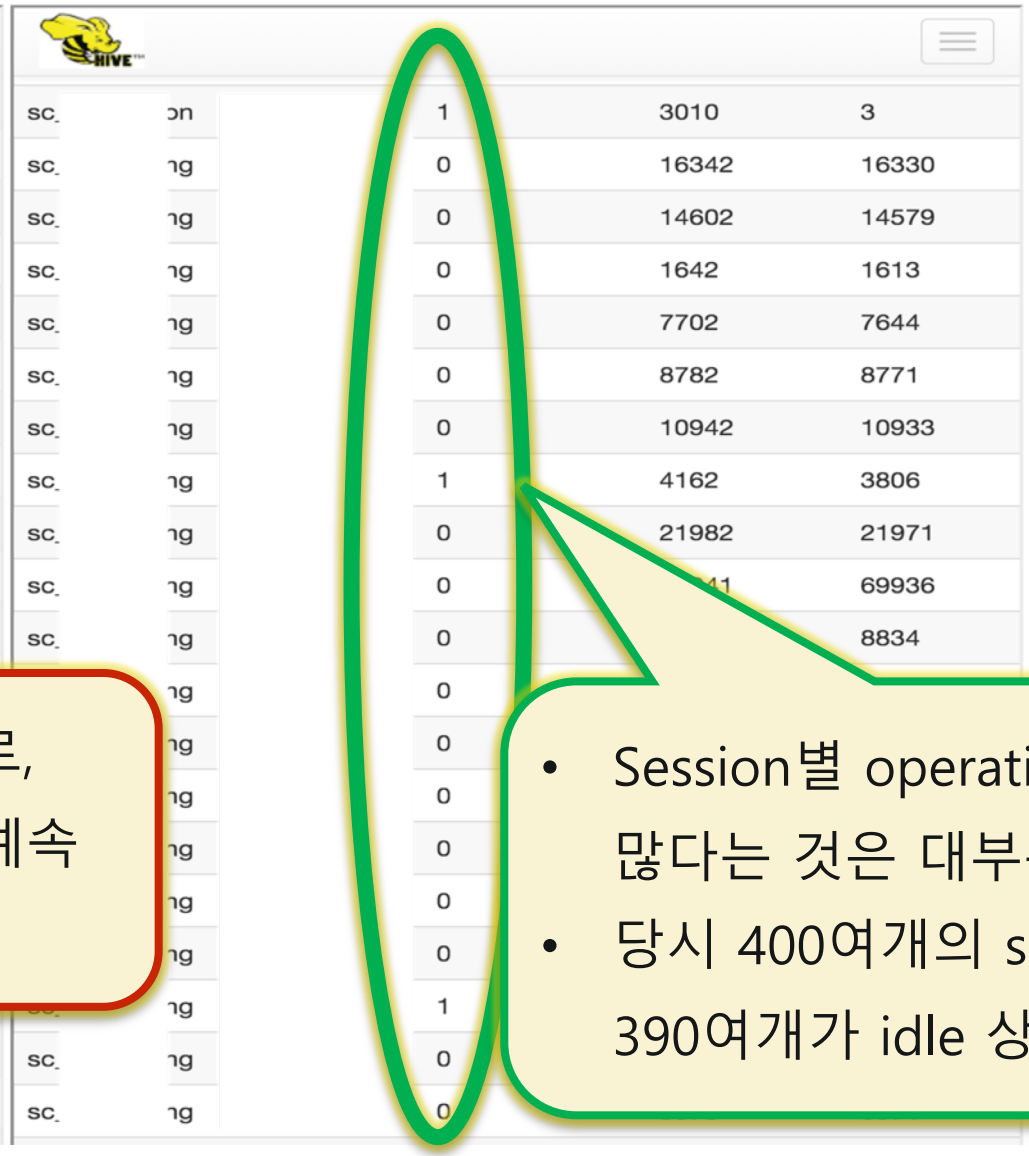
Hive Session 제어가 필요한 이유

- HiveServer2 입장에서 Hive Session은 값비싼 리소스
- 특히 SQL 실행 완료 후에는 Tez context, Temp table 등이 생성되어 메모리를 많이 차지하고 있을 가능성 높음
- Client에서 의도적으로 Hive Session을 장시간 열어놓는 경우가 있지만
- SQL 실행 후 Hive Session을 닫지 않고 계속 생성하여 장애가 발생하는 사례 존재 : Hive Session Leak

2.2.1.1 Hive Session Leak 사례

sc_l	ng	0	9622	9610
sc_l	ng	0	12562	12554
sc_l	ng	0	89302	88672
sc_l	ng	0	5182	5172
sc_l	ng	1	13821	6809
sc_l	ng	0	16342	16334
sc_l	ng	0	9742	9732
sc_l	ng	0	7402	7394
sc_l	ng	0	22042	22031
sc_l	ng	0	87741	86990
sc_l	ng	0	70102	70003
sc_l	ng			
sc_l	ng			
sc_l	ng			
sc_l	ng			
sc_l	ng			
sc_l	ng			
sc_l	ng	0	20661	20649
sc_l	ng	0	19462	19455

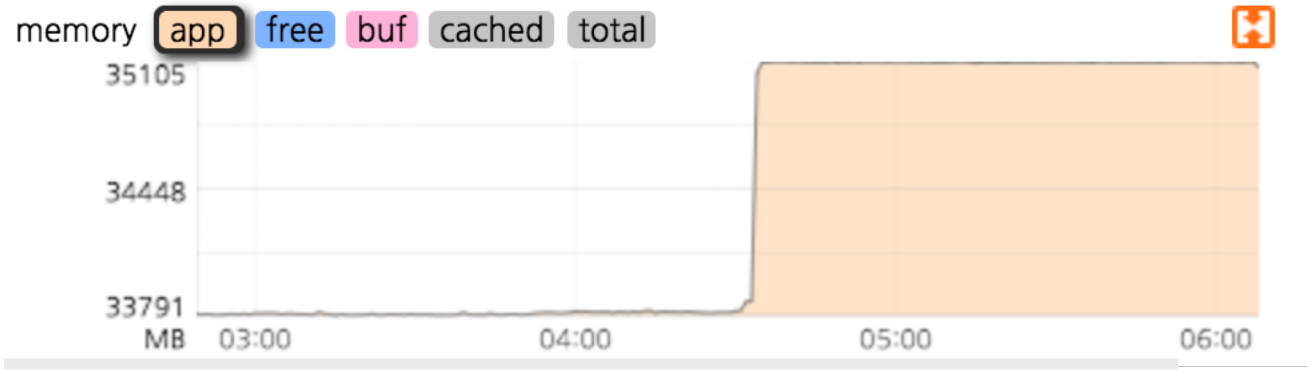
특정 사용자의 프로그램 오류로, Hive Session을 닫지 않은 채 계속 새로운 session에서 SQL 실행

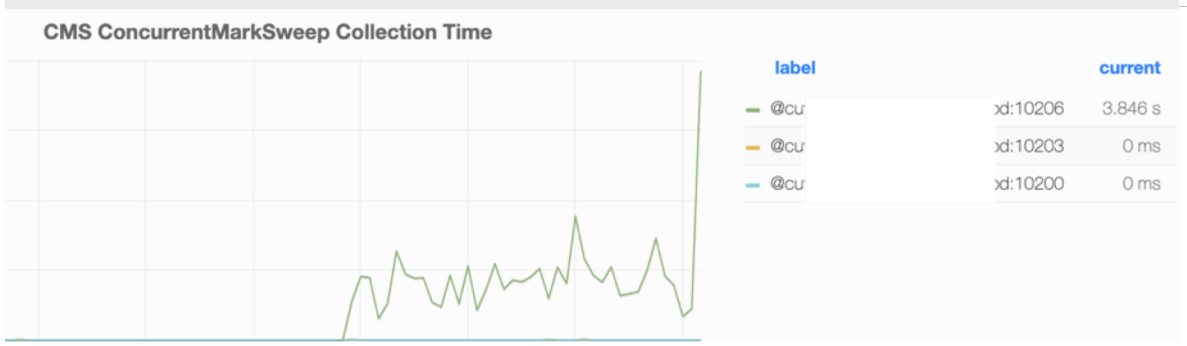


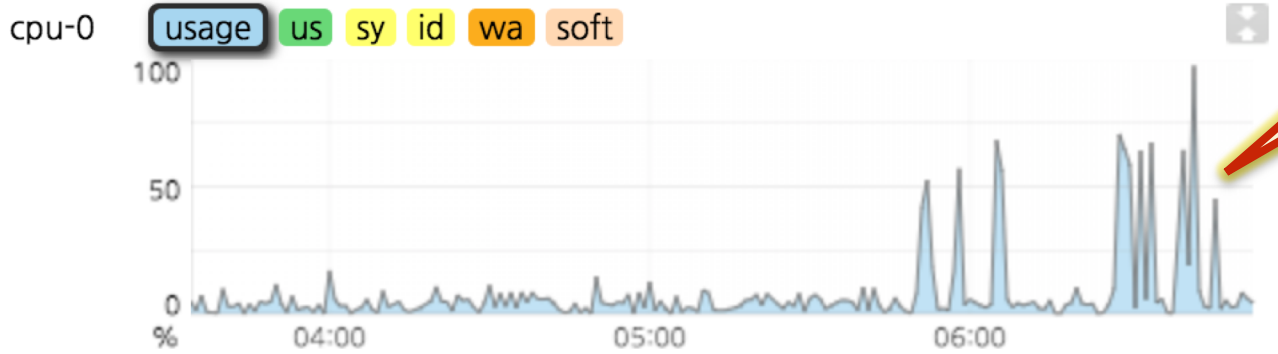
sc_	on	1	3010	3
sc_	ng	0	16342	16330
sc_	ng	0	14602	14579
sc_	ng	0	1642	1613
sc_	ng	0	7702	7644
sc_	ng	0	8782	8771
sc_	ng	0	10942	10933
sc_	ng	1	4162	3806
sc_	ng	0	21982	21971
sc_	ng	0	41	69936
sc_	ng	0		8834
	ng	0		
	ng	0		
	ng	0		
	ng	0		
	ng	0		
	ng	0		
	ng	1		
sc_	ng	0		
sc_	ng	0		

- Session별 operation 수로, 0이 많다는 것은 대부분 idle 상태
- 당시 400여개의 session 중 390여개가 idle 상태

2.2.1.2 Leaked Session이 많은 HiveServer2 상태







Memory 34GB 차지

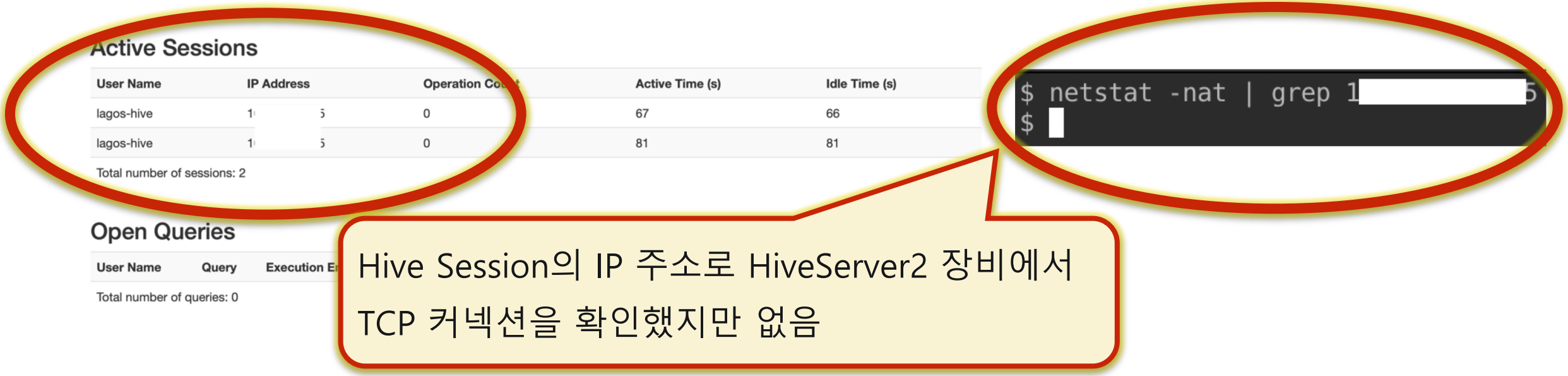
JVM GC Time 및 오버헤드 증가

장비 CPU 사용률 증가

명백한 HiveServer2 장애

2.2.1.3 제 코드는 Session Close를 하고 있어요!

- Client 실행 환경에 따라 프로세스 kill 방식으로 실행을 종료하는 사례 있음
- 프로세스 종료시 Hive Session을 생성할 때 맺은 TCP 커넥션은 정리됨
- 그러나 Hive Session과 TCP 커넥션은 1:1 대응 관계가 아니라서 Hive Session은 남아 있음!



2.2.1.4 HiveServer2 차원에서의 제어 필요

- CQuery는 다수의 사용자에게 서비스를 제공하는 멀티테넌트 시스템
- 사용자들의 선의에만 맡길 수 없음
- 일부 사용자의 실수가 전체 사용자에게 영향을 주면 안 됨
- 즉, 시스템 차원에서 Hive Session 제어가 되어야 함

2.2.1.5 Hive Session 제어 설정

사용자별 동시 접속 세션 수 제한

- hive.server2.limit.connections.per.user=5 (Since Hive 3.x)

Hive Session Idle Timeout

- hive.server2.idle.session.timeout=10m

2.2.1.5 Hive Session 제어 설정

사용자별 동시 접속 세션 수 제한

- hive.server2.limit.connections.per.user=5 (Since Hive 3.x)

Hi

-

하나의 HiveServer2에 적용하는 것으로, 현재 CQuery는 여러 대의 HiveServer2 운영중

2.2.1.5 Hive Session 제어 설정

사용자별 동시 접속 세션 수 제한

- hive.server2.limit.connections.per.user=5 (Since Hive 3.x)

Hive Session Idle Timeout

- hive.server2.idle.session.timeout=10m

- 더 짧게 줄일 수 있지만 기본값 4시간(4h)에서 급격하게 줄인 상황
- 10분(10m)으로 운영하며 모니터링중

2.2.2 리소스 제어 – Abuse 성격의 SQL 방지

SQL 방지가 필요한 이유

- 일부 사용자가 작성한 abuse 성격의 SQL에 의해 HiveServer2 장애 발생 가능성 차단
- CPU 과다, 메모리 과다 사용 등의 문제를 일으킬 수 있음

방지가 필요한 SQL들

- 너무 긴 SQL
- UNION 폭풍

2.2.2.1 너무 긴 SQL

Hive의 SQL 처리 과정 요약

1. Syntax Parsing by ANTLR → Abtract Syntax Tree (AST)
2. Semantic 분석 → SQL Plan (Tree 형태)
3. Tez DAG 생성 → YARN Application 제출

2.2.2.1 너무 긴 SQL

Hive의 SQL 처리 과정 요약

1. Syntax Parsing by ANTLR → Abtract Syntax Tree (AST)
2. Semantic 분석 → SQL Plan (Tree 형태)
3. Tez DAG 생성 → YARN Application 제출

```
SELECT count(*) AS qc
FROM korea_nxlog_per_day.nx_mobile
WHERE
 log_date BETWEEN '2021-01-01'
          AND '2021-01-31'
 AND refine_query='네이버';
```

AST

```
TOK_QUERY
  TOK_FROM
    TOK_TABREF
      TOK_TABNAME
        korea_nxlog_per_day
        nx_mobile
...
    TOK_WHERE
      AND
        AND
          TOK_FUNCTION
            between
            KW_FALSE
            TOK_TABLE_OR_COL
              log_date
            '2021-01-01'
            '2021-01-31'
        =
          TOK_TABLE_OR_COL
            refine_query
          '네이버'
```

2.2.2.1 너무 긴 SQL

어떤 문제를 일으켰는가?

- AST, SQL plan 생성 과정에서 JVM 메모리 과다 사용
- 잦은 GC 유발로 인한 CPU 오버헤드 발생

2.2.2.1 너무 긴 SQL

해결책

- hive.query.max.length=2MB
- 현재 개발중인 Hive 4.0에 있는 설정
- https://issues.apache.org/jira/browse/HIVE-22901

2.2.2.2 UNION 폭풍

이런 사례가 있었습니다.

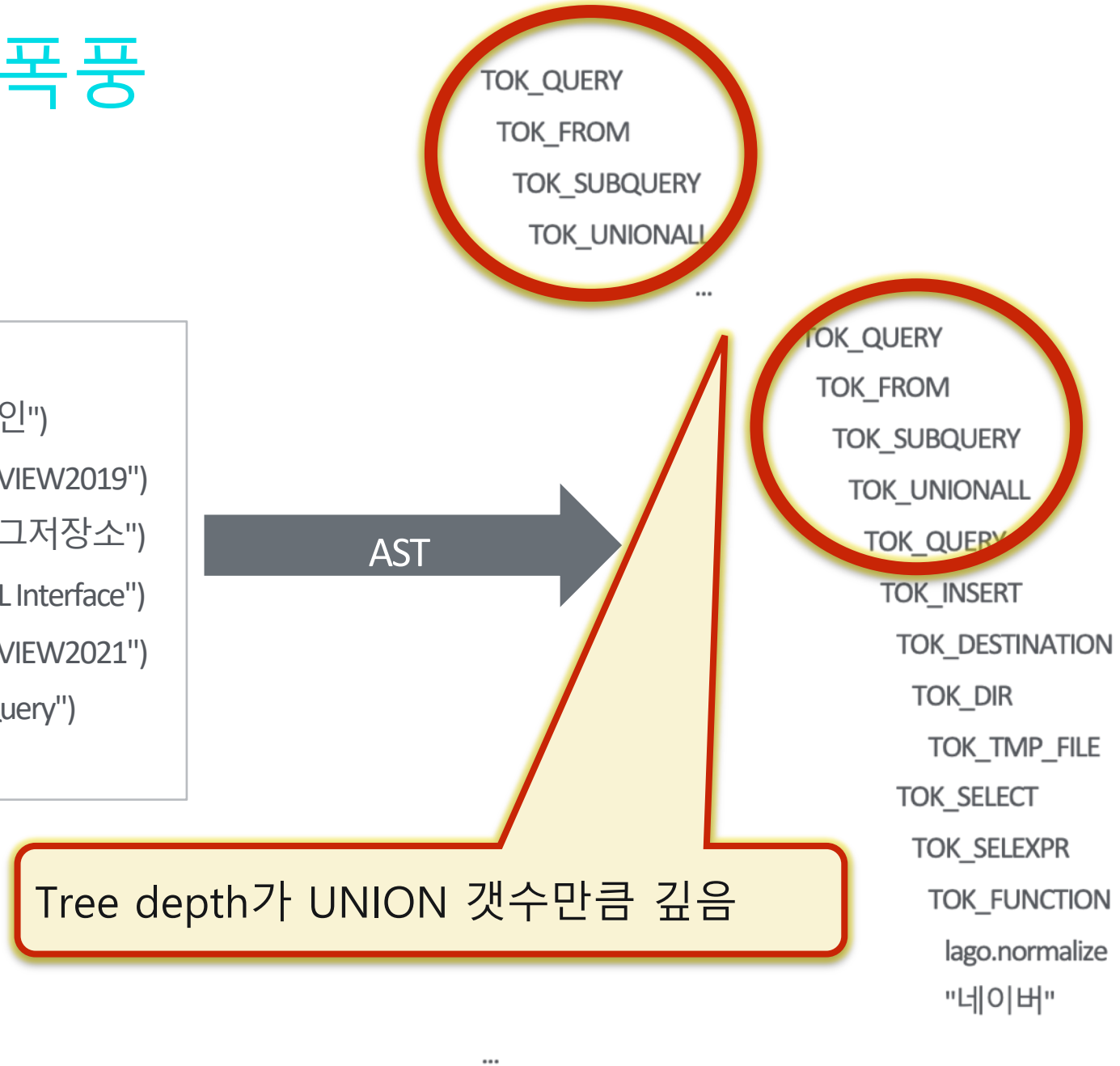
```
SELECT lago.normalize("네이버")
UNION SELECT lago.normalize("라인")
UNION SELECT lago.normalize("DEVIEW2019")
UNION SELECT lago.normalize("로그저장소")
UNION SELECT lago.normalize("SQL Interface")
UNION SELECT lago.normalize("DEVIEW2021")
UNION SELECT lago.normalize("CQuery")
... 600여개
```

2.2.2.2 UNION 폭풍

AST

```
SELECT lago.normalize("네이버")
UNION SELECT lago.normalize("라인")
UNION SELECT lago.normalize("DEVIEW2019")
UNION SELECT lago.normalize("로그저장소")
UNION SELECT lago.normalize("SQL Interface")
UNION SELECT lago.normalize("DEVIEW2021")
UNION SELECT lago.normalize("CQuery")
... 600여개
```

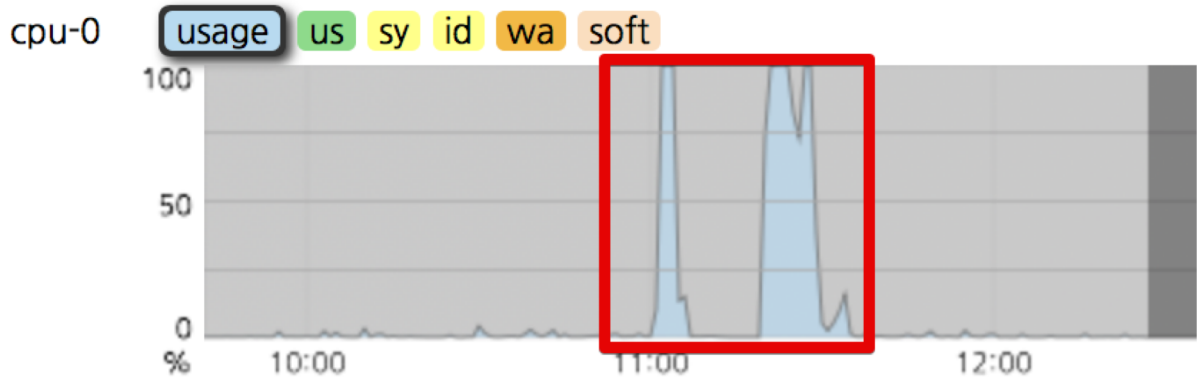
AST



2.2.2.2 UNION 폭풍

어떤 문제를 일으켰는가?

- Semantic 분석은 AST를 traversal 하면서 수행하는 과정
- AST traversal 과정에서 CPU 과다 사용으로 HiveServer2 장애 일으킴



2.2.2.2 UNION 폭풍

해결책

- Hive Hook을 통해 SQL AST 체크 후 필요하면 에러 처리
- org.apache.hadoop.hive.ql.parse.HiveSemanticAnalyzerHook

```
import org.apache.hadoop.hive.ql.parse.ASTNode;


public class LagoAbuseOperatorChecker implements HiveSemanticAnalyzerHook {
    ...
    public ASTNode preAnalyze(HiveSemanticAnalyzerHookContext context, ASTNode ast) ... {
    }
}
```

2.2.2.2 UNION 폭풍

해결책

```
import org.apache.hadoop.hive.ql.parse.ASTNode;


public class LagoAbuseOperatorChecker implements HiveSemanticAnalyzerHook {
   ...
   public ASTNode preAnalyze(HiveSemanticAnalyzerHookContext context, ASTNode ast) ... {
        // AST를 traversal 하면서 UNION과 JOIN 사용 수 측정
        // 현재 최대 10개까지  허용
   }
}
```

2.2.3 Hive 성능 개선

소개할 개선 사항

- Bucket Pruning 개선
- Tez ApplicationMaster(AM) VCore 수 조정

개선 포인트

- SQL input을 줄여보자.
- SQL input을 좀더 효율적으로 처리하자.

2.2.3.1 Bucket Pruning 개선

Hive ORC Table의 Partition과 Bucket

- Bucket pruning을 이해하기 위해서 알아야 할 개념들

```
CREATE EXTERNAL TABLE `korea_naver.search_log`(
  ...
  `search_query` string,
  ...
)
STORED AS ORC
...
PARTITIONED BY (`log_date` string) -- Partition
CLUSTERED BY (search_query)
INTO 240 BUCKETS                   -- Bucket
```

Hive ORC Table

2.2.3.1 Bucket Pruning 개선

DEVIEW 2021

HDFS에서의 Hive ORC Table

```
CREATE EXTERNAL TABLE `korea_naver.search_log`
  ...
  `search_query` string,
  ...
)
STORED AS ORC
...
PARTITIONED BY (`log_date` string) -- Partition
CLUSTERED BY (search_query)
INTO 240 BUCKETS                    -- Bucket
```

Directory

HDFS

hdfs://cquery/warehouse/korea_naver.db/search_log/log_date=2021-10-14

000000

000001

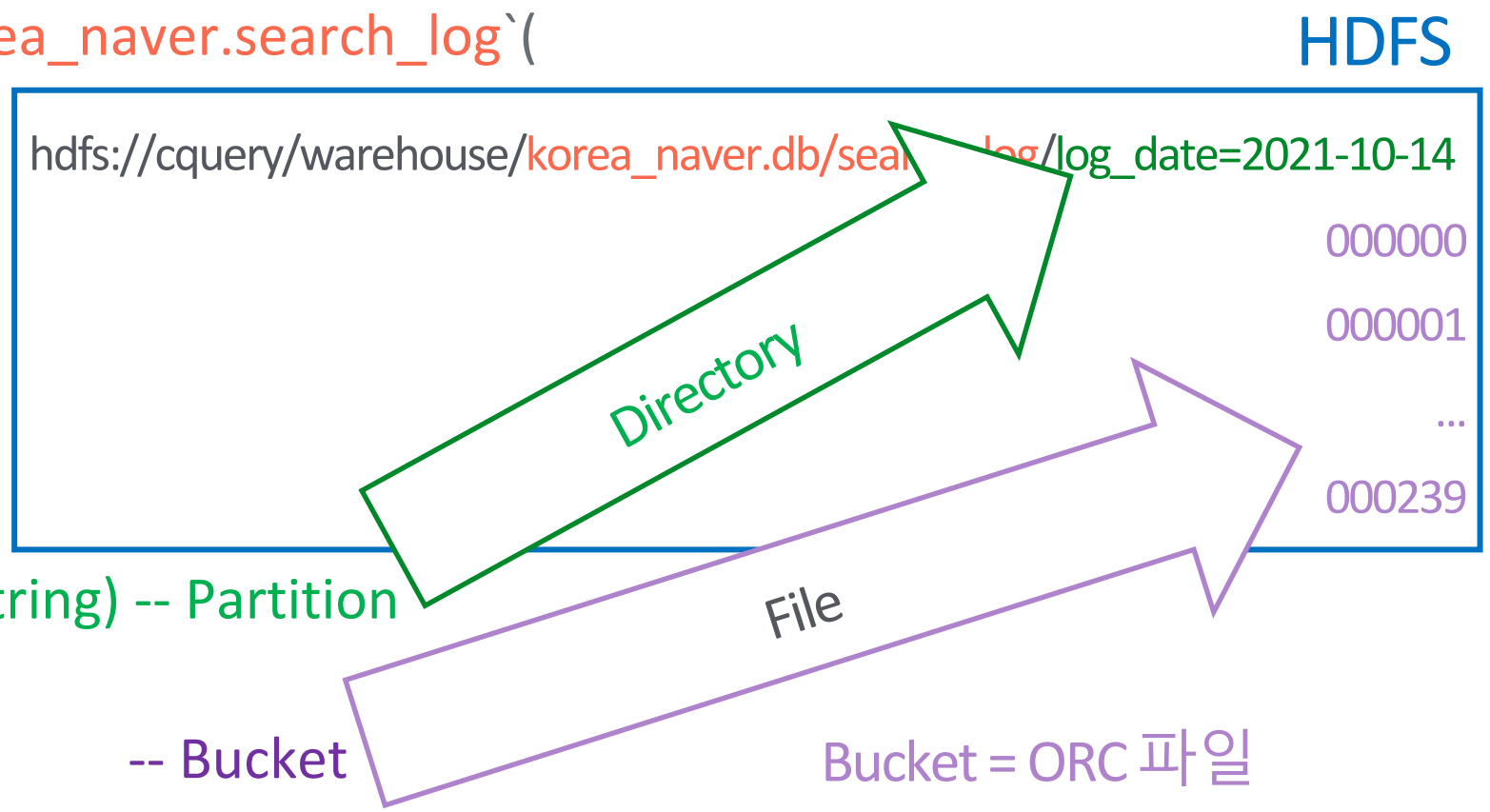
...

000239

2.2.3.1 Bucket Pruning 개선

HDFS에서의 Partition과 Bucket

```
CREATE EXTERNAL TABLE `korea_naver.search_log`(
  ...
  `search_query` string,
  ...
)
STORED AS ORC
...
PARTITIONED BY (`log_date` string) -- Partition
CLUSTERED BY (search_query)
INTO 240 BUCKETS                   -- Bucket
```

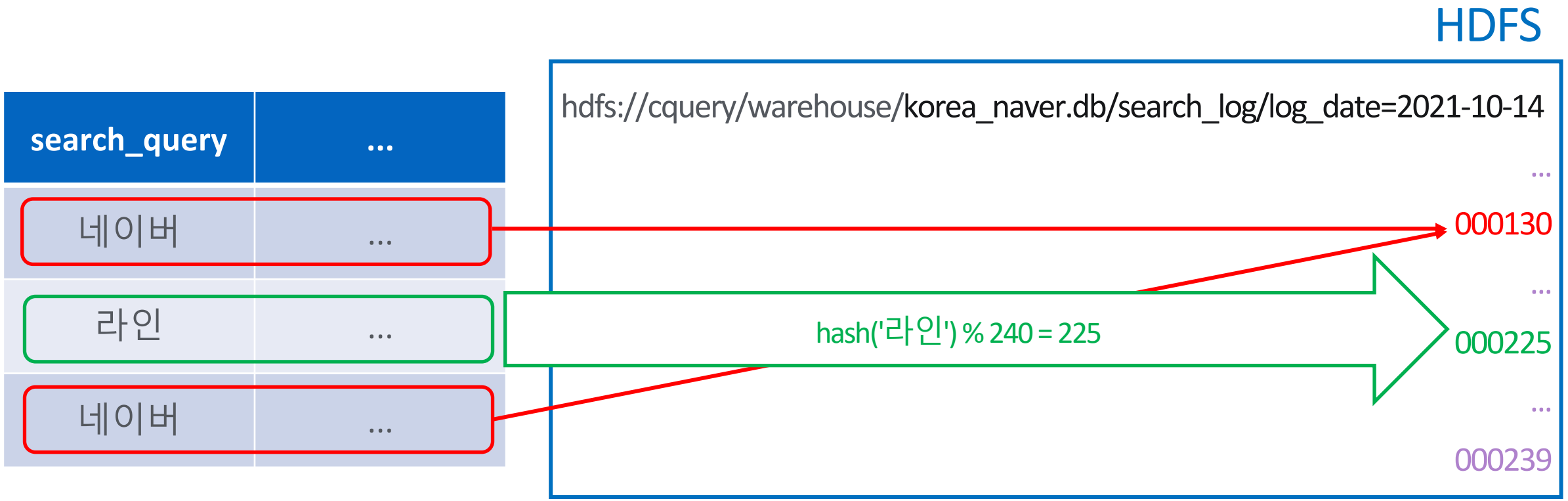


2.2.3.1 Bucket Pruning 개선

DEVIEW 2021

Bucket은 어떻게 구성되는가

CLUSTERED BY (search_query) INTO 240 BUCKETS

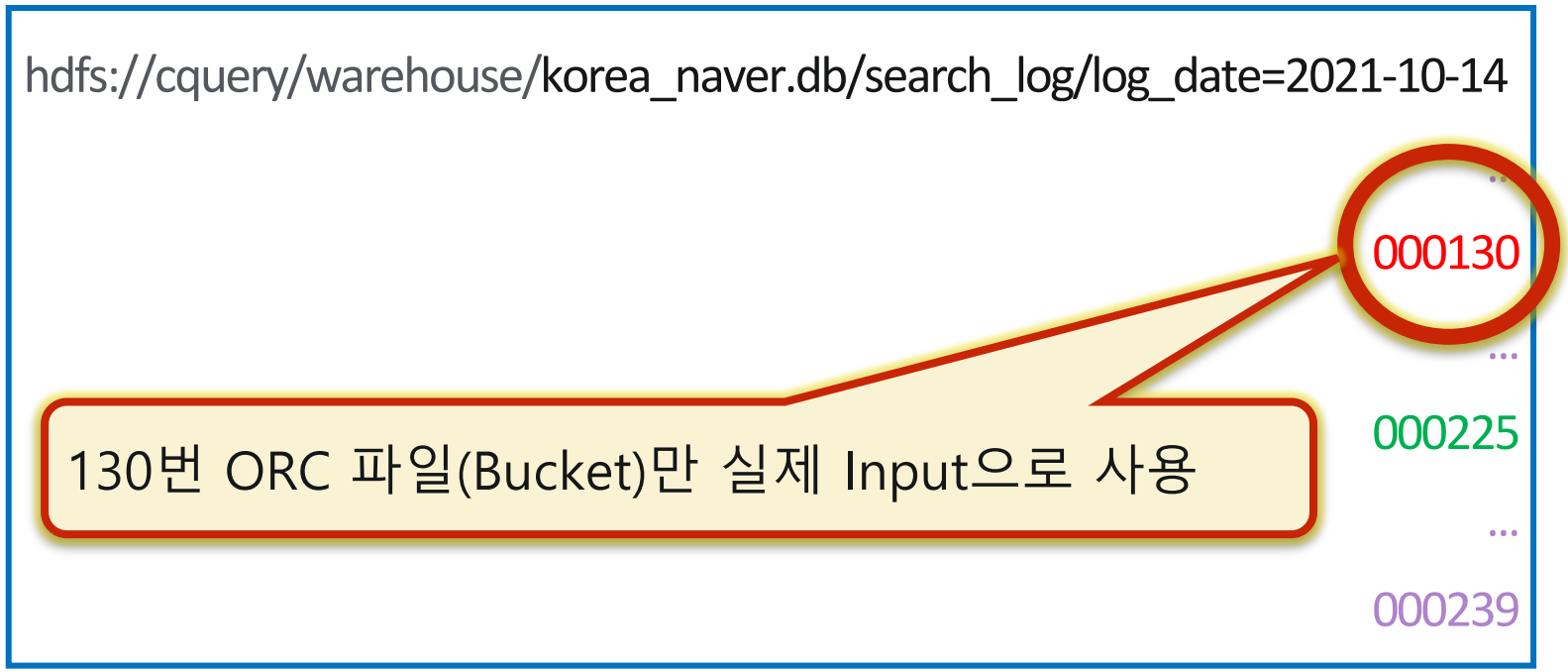


2.2.3.1 Bucket Pruning 개선

Bucket Pruning이란 무엇인가

- WHERE 절에 bucketing column에 대해 equality 비교를 할 때 적용
- Equality 비교 대상 값이 들어있는 ORC 파일들만 실제 input으로 사용

```
SELECT search_query, count(*)
FROM korea_naver.search_log
WHERE log_date='2021-10-14'
  AND search_query='네이버'
GROUP BY 1;
```



2.2.3.1 Bucket Pruning 개선

Bucket Pruning 실행 과정 요약

1. Hive : Bucket 번호를 Tez DAG context에 포함시킴
2. YARN ResourceManager(RM) : Tez ApplicationMaster(AM) 실행
3. Tez AM : SQL 실행 스케쥴러 역할

3.1 Partition에 포함된 모든 ORC 파일들의 목록 작성

3.2 목록 상의 ORC 파일들 메타 분석 후 InputSplit 목록 생성

3.3 Bucket 번호로 InputSplit 목록 필터링해서 최종 목록 생성

3.4 Tez tasks(=YARN Containers) 할당 및 실행

2.2.3.1 Bucket Pruning 개선

Bucket Pruning 실행 과정

1. Hive : Bucket 번호를 Tez DAG context에 포함시킴

Hive

```
SELECT search_query, count(*)
FROM korea_naver.search_log
WHERE log_date='2021-10-14'
AND search_query='네이버'
GROUP BY 1;
```

Tez DAG Context

← BucketNumbers = 130

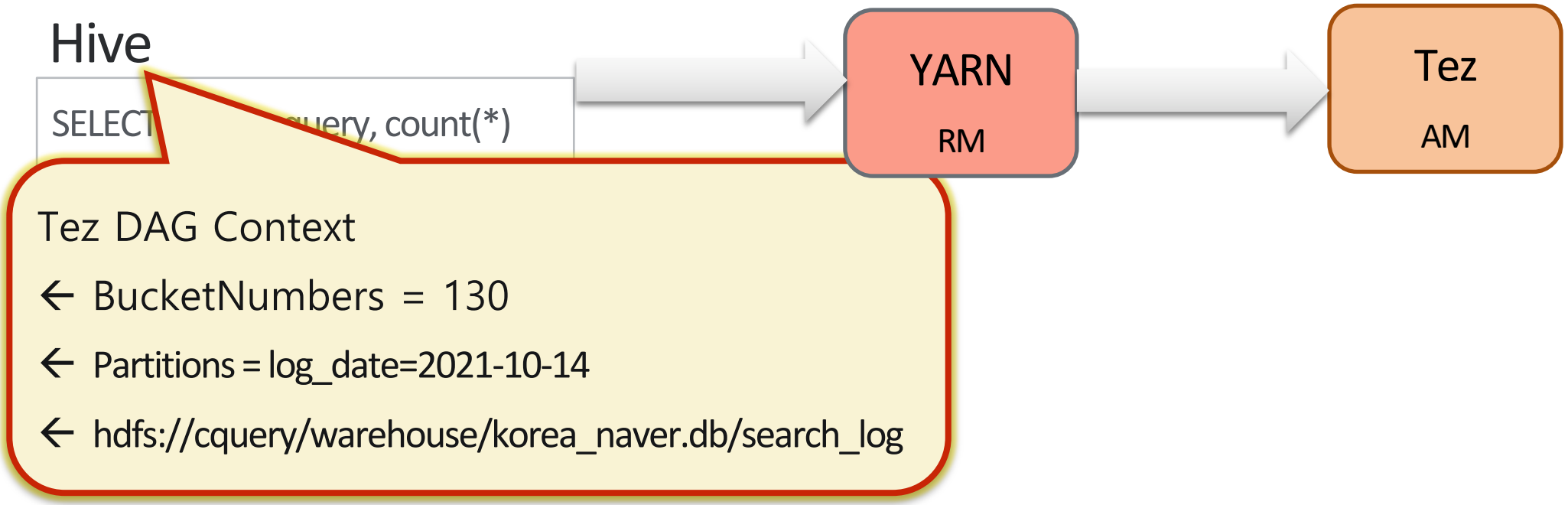
← Partitions = log_date=2021-10-14

← hdfs://cquery/warehouse/korea_naver.db/search_log

2.2.3.1 Bucket Pruning 개선

Bucket Pruning 실행 과정

2. YARN ResourceManager(RM) : Tez ApplicationMaster(AM) 실행

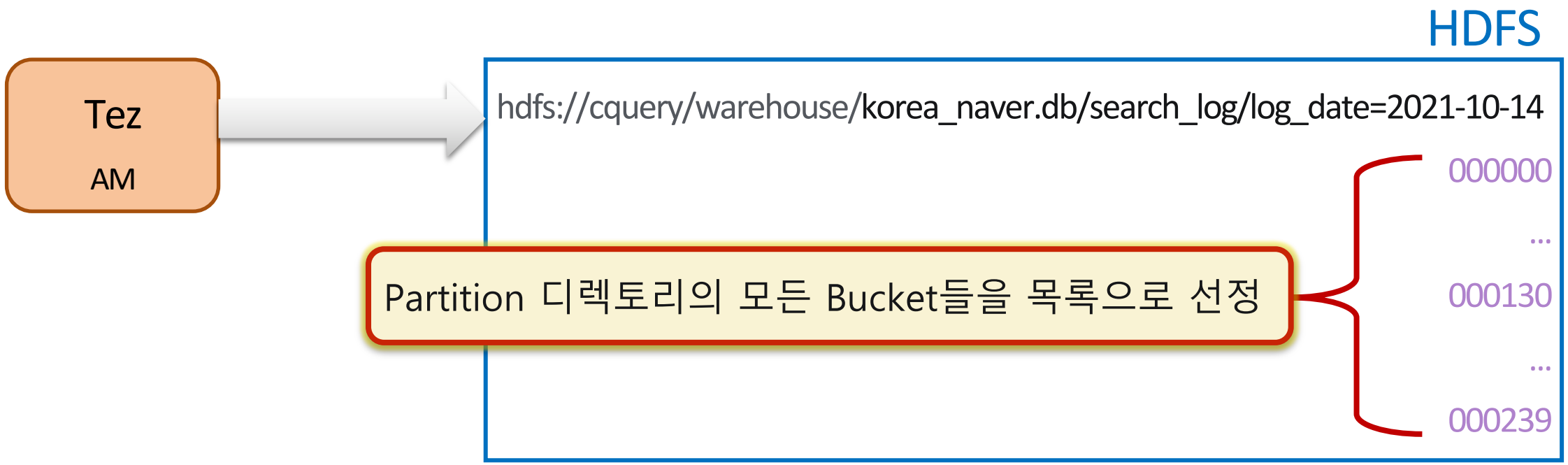


2.2.3.1 Bucket Pruning 개선

Bucket Pruning 실행 과정

3. Tez AM : SQL 실행 스케쥴러 역할

3.1 Partition에 포함된 모든 ORC 파일들의 목록 작성

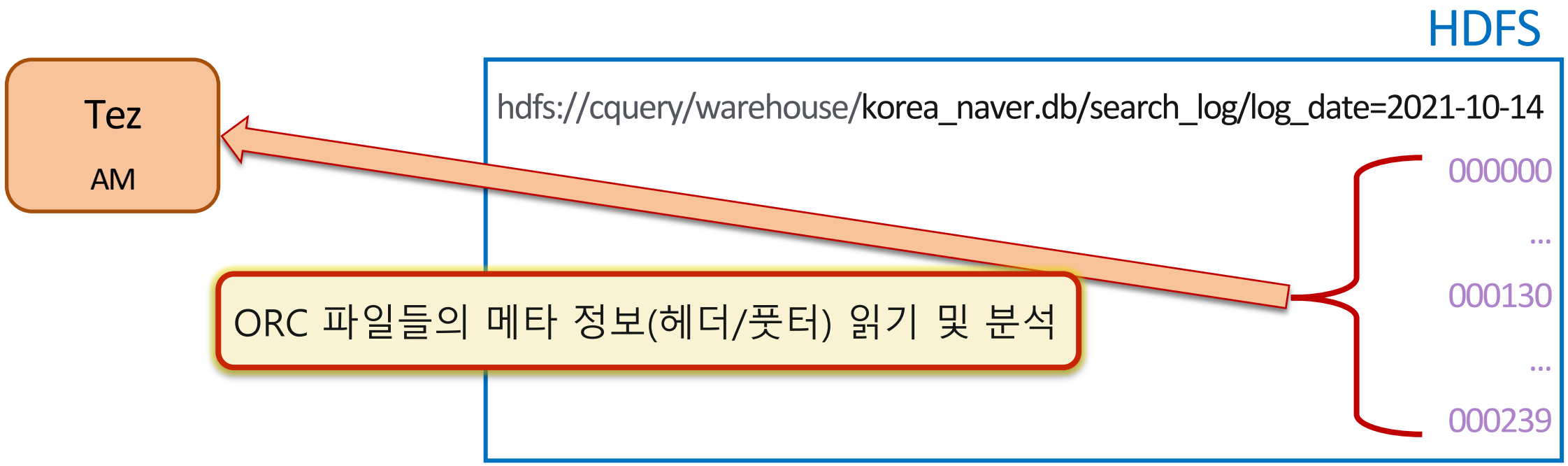


2.2.3.1 Bucket Pruning 개선

Bucket Pruning 실행 과정

3. Tez AM : SQL 실행 스케쥴러 역할

3.2 목록 상의 ORC 파일들 메타 분석 후 InputSplit 목록 생성



2.2.3.1 Bucket Pruning 개선

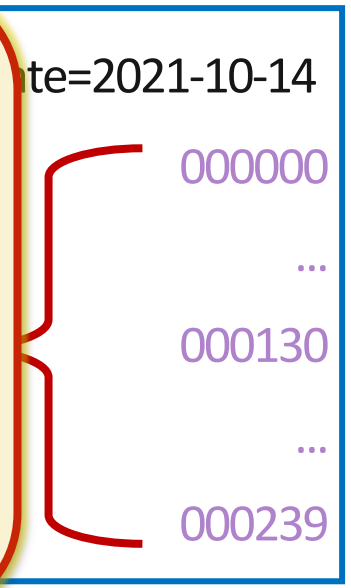
Bucket Pruning 실행 과정

3. Tez AM : SQL 실행 스케쥴러 역할

3.2 목록 상의 ORC 파일들 메타 분석 후 InputSplit 목록 생성

여기서 잠깐! InputSplit이란?

- 전체 input을 parallel하게 처리하기 위해서, 적절한 사이즈로 쪼개놓은 것
- 예를 들어 전체 input이 100GB라면, 100개로 나눠서 처리하려면 1GB 짜리 InputSplit 100개로 구성
- ORC 파일 하나가 InputSplit이 될 수 있고, 파일 사이즈가 큰 경우 여러 개의 InputSplit들이 될 수 있음

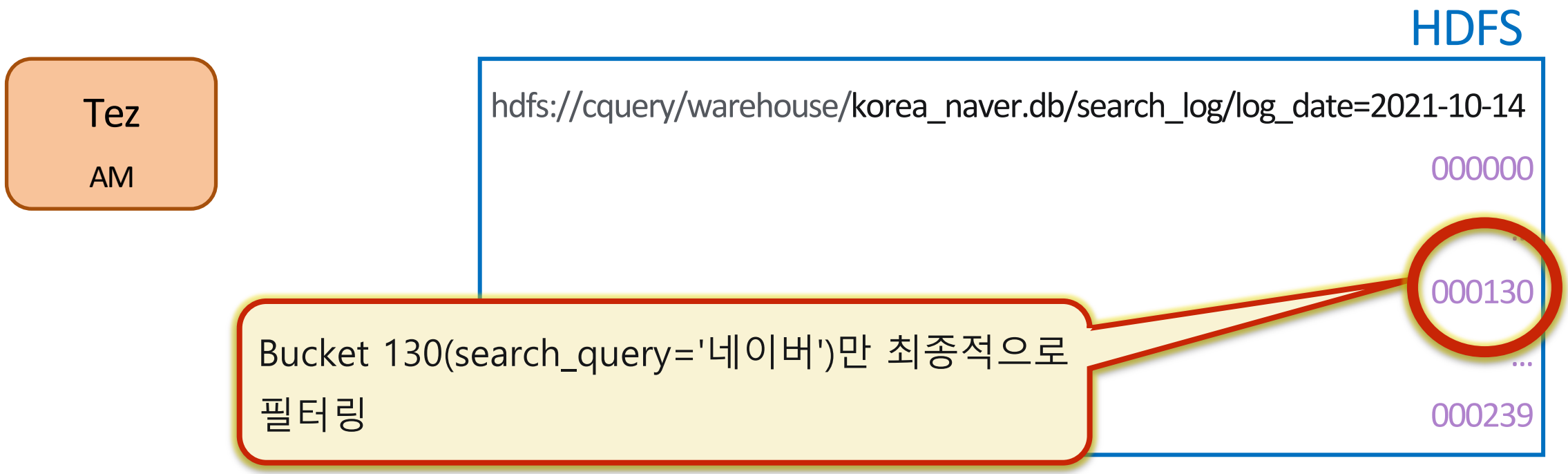


2.2.3.1 Bucket Pruning 개선

Bucket Pruning 실행 과정

3. Tez AM : SQL 실행 스케쥴러 역할

3.3 Bucket 번호로 InputSplit 목록 필터링해서 최종 목록 생성



2.2.3.1 Bucket Pruning 개선

Bucket Pruning 실행 과정에서의 의문 사항

1. Hive : Bucket 번호를 Tez DAG Context에 포함시키고 YARN ResourceMan
2. YARN RM : Te
3. Tez AM : SQL 실행 스케줄러 실
3.1 Partition에 포함된 모든 ORC 파일들의 목록 작성
3.2 목록 상의 ORC 파일들 메타 분석 후 InputSplit 목록 생성
3.3 Bucket 번호로 InputSplit 목록 필터링해서 최종 목록 생성
3.4 Tez Tasks(=YARN Containers) 할당 및 실행

- 왜 모든 ORC 파일들의 메타 정보를 읽고 나서 필터링을 하는 걸까?
- 처음부터 bucket 번호에 해당하는 파일만 메타를 읽으면 안 되나?

2.2.3.1 Bucket Pruning 개선

개선된 Bucket Pruning 실행 과정

3. Tez AM : SQL 실행 스케줄링 역할

3.1 Bucket 번호로 필터링하면서 ORC 파일 목록 파악

3.2 목록 상의 ORC 파일들 메타 분석 후 InputSplit 목록 생성

3.4 Tez Tasks(=YARN [illegible]ners) 할당 및 실행

- https://issues.apache.org/jira/browse/HIVE-24948
- Hive master branch에 merge 되지는 않았으나 CQuery 에서 충분히 검증된 코드

2.2.3.1 Bucket Pruning 개선

Bucket Pruning 개선 전후 성능 비교

- ORC 파일 수가 충분히 많아야 개선 전 문제점이 드러남
- 성능 비교에 사용한 SQL

```
SELECT count(*) as qc
FROM korea_naver.search_log
WHERE log_date BETWEEN '2021-01-01' AND '2021-06-30'
...
AND search_query='로또'
```

- 총 ORC 파일 갯수 : 240 X 30일 X 6개월 = 43200
- Bucket Pruning된 갯수 : 1 X 30 X 6 = 180

2.2.3.1 Bucket Pruning 개선

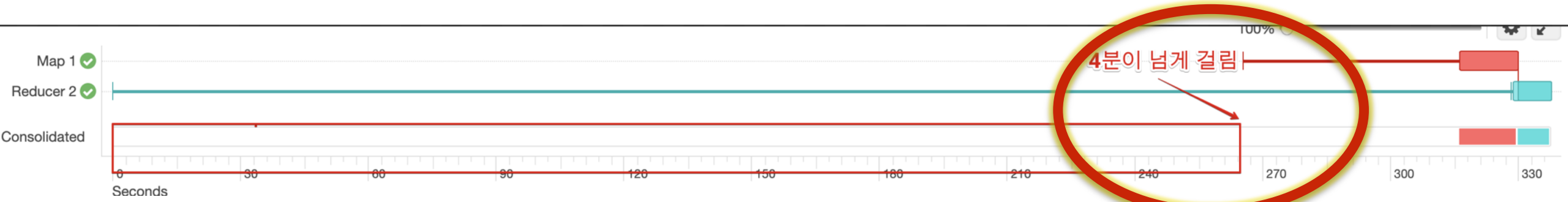
DEVIEW 2021

Bucket Pruning 개선 전 성능

Duration	5m 35s 662ms
Queue	lago
Logs	1

- 총 5분 35초 중 Tez AM에서 초반 4분 20초 소요
- 43200개의 ORC 파일 메타를 읽는데 걸린 시간이 약 4분

Vertex Name	Status	Progress	Total Tasks	Succeede
Map 1	✔ SUCCEEDED	100%	183	183
Reducer 2	✔ SUCCEEDED	100%	253	253



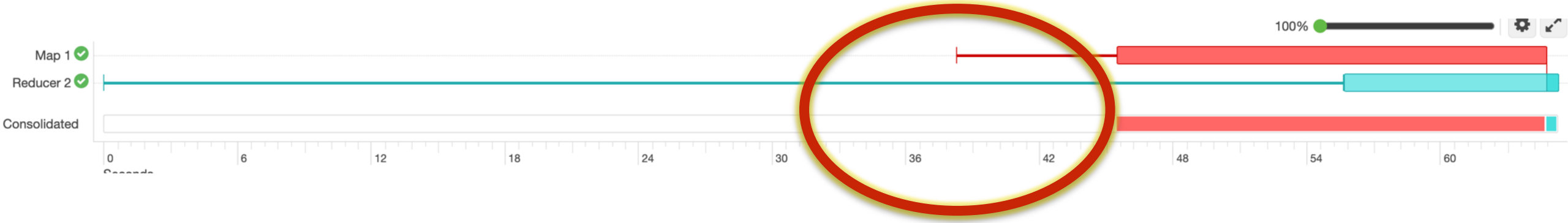
2.2.3.1 Bucket Pruning 개선

Bucket Pruning 개선 후 성능

Duratio	1m 5s 276ms
Queue	lago
Logs	1

메타를 읽어야 할 ORC 파일이 180개로 줄면서 4분 이상 단축됨

Vertex Name	Status	Progress	Total Tasks	Succee
Map 1	SUCCEEDED	100%	146	146
Reducer 2	SUCCEEDED	100%	253	253



2.2.3.2 Tez AM VCore 수 조정

SQL 실행 스케쥴러로써의 Tez AM

1. Partition에 포함된 모든 ORC 파일들의 목록 작성
2. 목록 상의 ORC 파일들 메타 분석 후 InputSplit 목록 생성

…

Bucket Pruning이 적용 안 되는 일반적인 경우

- 위의 1, 2번 과정을 항상 수행
- ORC 파일 갯수가 많은 경우 Tez AM에서 소요되는 시간이 항상 큼

2.2.3.2 Tez AM VCore 수 조정

hive.orc.compute.splits.num.threads

- thread pool 크기, 기본값 = 10
- Tez AM에서 ORC 파일 메타 읽을 때 thread pool을 사용하여 parallel하게 처리
- 그렇다면 CPU core 수는?

tez.am.resource.cpu.vcores

- 기본값 = 1
- YARN RM에 Tez AM 생성시 요청하는 virtual CPU core 수

2.2.3.2 Tez AM VCore 수 조정

tez.am.resource.cpu.vcores 설정 방식

- 사용자들이 직접 설정하기 어려우므로 Hive Hook으로 처리중
- SQL 분석을 통해 ORC 파일 갯수 파악 후, 그에 따라 AM vcore 수 세팅
- hive.orc.compute.num.threads는 AM vcore 수의 3배로 설정

2.3 Trino 도입

왜 Trino 엔진을 도입하게 되었는가?

- Hive+Tez 조합 대비 매우 빠름
- 사용자들의 SQL 실행 속도 개선 요구 사항을 만족하기 위해서 도입

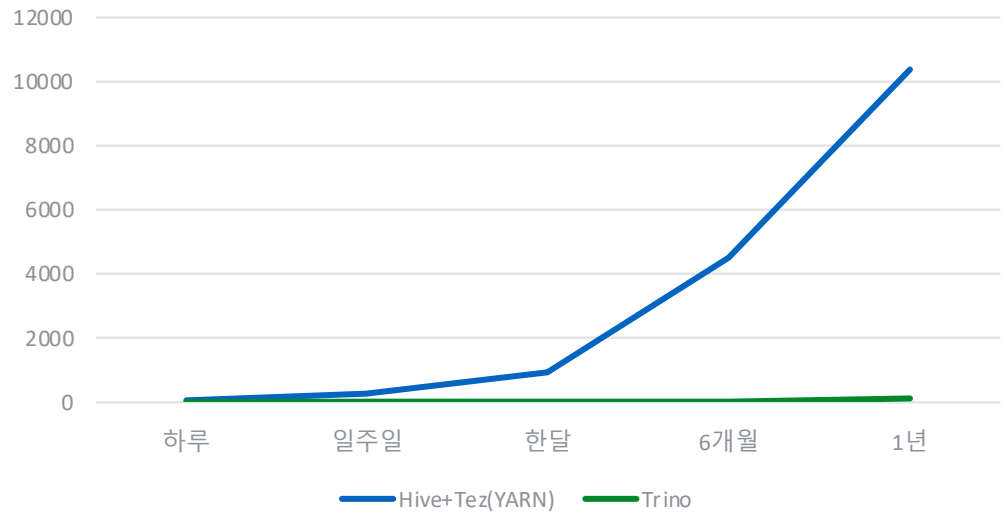
2.3.1 Hive+Tez vs Trino 성능 비교

DEVIEW 2021

비교 대상 1 : 모바일 통합검색 클릭 로그 수

```
SELECT count(*)
FROM korea_naver.mobile_click_log
WHERE log_date = '2021-06-27’
```

기간	데이터 크기	Hive+Tez	Trino	소요 시간 비교
하루	수십 GB	1분(60초)	2초	**98% 감소**
일주일	수백 GB	2분(120초)	10초	**91% 감소**
한달	수 TB	6분(360초)	43초	**88% 감소**
6개월	수십 TB	23분(1380초)	2분(120초)	**91% 감소**
1년	수십 TB	104분(6240초)	4분(240초)	**96% 감소**



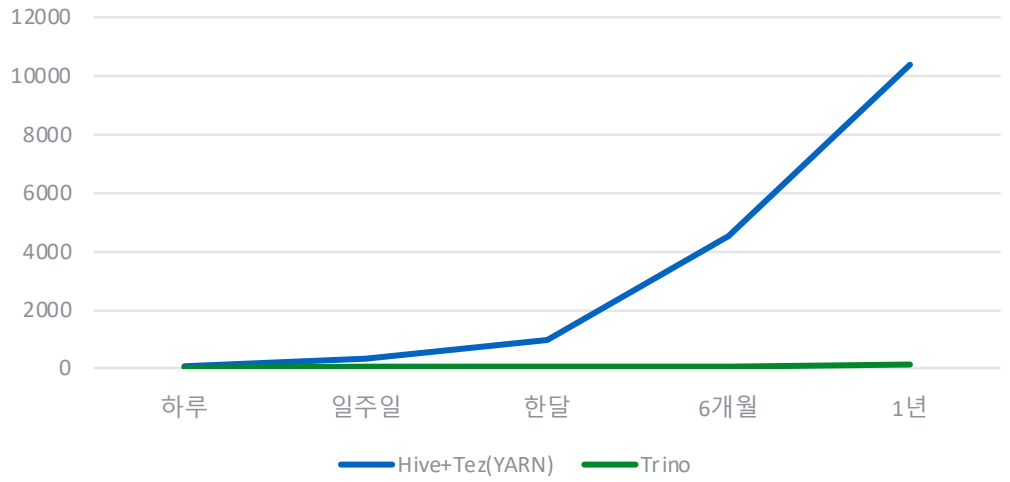
2.3.1 Hive+Tez vs Trino 성능 비교

DEVIEW 2021

비교 대상 2 : CQuery 사용자들이 가장 많이 쓰는 SQL Template

다수의 Inner Select, FULL OUTER JOIN, ORDER BY로 조합된 SQL

기간	데이터 크기	Hive+Tez	Trino	소요 시간 비교
하루	수 GB	1분(60초)	6초	**90% 감소**
일주일	수십 GB	5분(300초)	10초	**96% 감소**
한달	수백 GB	16분(960초)	18초	**98% 감소**
6개월	수 TB	75분(4500초)	1분(60초)	**98% 감소**
1년	수 TB	173분(10380초)	2분(120초)	**98.8% 감소**



2.3.2 Hive+Tez vs Trino 실행 방식 비교

	Hive+Tez	Trino
SQL 분석	HiveServer2	Trino Coordinator
SQL 실행 과정 스케줄링	Tez ApplicationMaster	Trino Coordinator
SQL 실제 실행	Tez Tasks(=YARN Containers)	Trino Workers
SQL 실행 전 준비 과정이 필요한가?	**필요** YARN RM에서 Tez AM 실행	**필요없음**
SQL 실제 실행시 리소스 할당이 필요한가?	**필요** Tez AM이 YARN RM에 요청해서 YARN Container 할당	**필요없음** Trino Workers는 항상 떠 있음 각 Worker는 thread pool 사용

2.3.2 Hive+Tez vs Trino 실행 방식 비교

	Hive+Tez	Trino
SQL 실행 전 준비 과정이 필요한가?	**필요** YARN RM에서 Tez AM 실행	**필요없음**
SQL 실제 실행시 리소스 할당이 필요한가?	**필요** Tez AM이 YARN RM에 요청해서 YARN Container 할당	**필요없음** Trino Workers는 항상 떠 있음 각 Worker는 thread pool 사용

리소스 할당 과정, 별도의 프로세스 생성 과정 등의 유무가 성능 차이에 큰 영향을 주는 것으로 예상

2.3.3 Trino 운영시 고려 사항

- 메모리 사용 문제
- 특정 SQL 실행 오류에 의한 Trino Worker 장애
- Hook 매커니즘 부재

2.3.3.1 메모리 사용 문제

- 무조건 메모리에서 처리하는 것이 기본 동작
- SQL에 따라 메모리 부족으로 인해 쿼리가 실패할 수 있음
- Spill to disk : 메모리 부족 에러 회피 옵션

  spill-enabled=true

  spiller-spill-path=/local-file-system-path

2.3.3.2 특정 SQL 실행 오류에 의한 Worker 장애

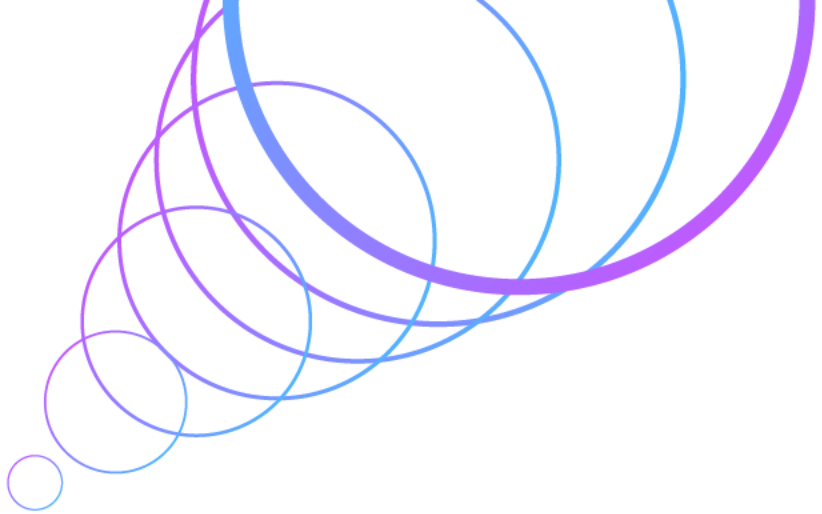
- 메모리를 많이 사용하는 SQL에 의해 문제 발생시 Worker 프로세스 장애가 발생할 수 있음
  JVM OutOfMemoryError에 의해 JVM이 사실상 죽은 경우
  Linux OOM Killer에 의해 프로세스가 kill 되는 경우
- 동일한 Worker들에서 병렬적으로 SQL들이 처리되므로 일부 SQL로 인해 전체 SQL들이 실패할 수 있음
- Resource groups : 대응 방안으로 검토중
  resource-groups. configuration-manager=file
  resource-groups. config-file=...
  hardConcurrencyLimit, softMemoryLimit 등 리소스 제어 설정 가능

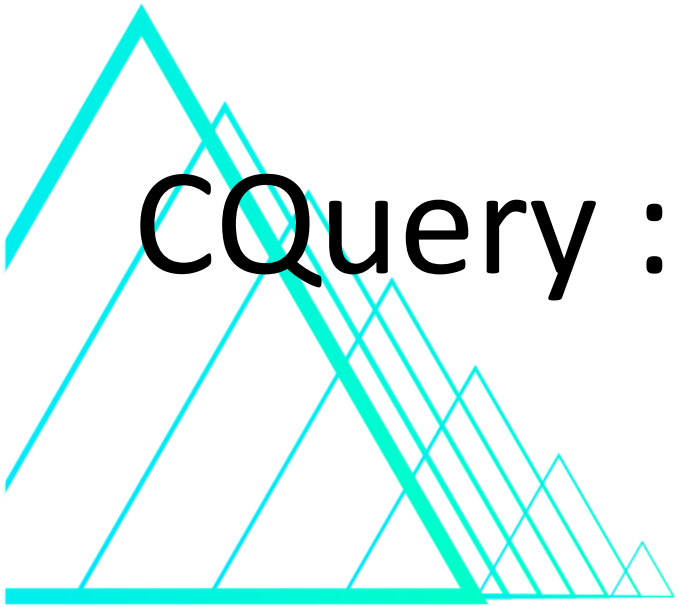
2.3.3.3 Hook 매커니즘 부재

- Hive의 Hook 매커니즘이 없어서, 서버 단에서의 제어를 할 수 없음
- Trino EventListener는 Query 처리 시작/종료 이벤트만 받을 수 있음
- ASM 또는 BCEL 기반으로 Java runtime hooking 적용 검토중

  Trino Coordinator 소스를 분석하여 소스 레벨에서 SQL 처리 위치 파악

  Coordinator JVM 실행시 javaagent 옵션을 통해 runtime hooking 로직 실행





CQuery：Easy and Fast SQL-Based Analytics Solution

정의근, 신선우 AI & Data Platform

3. CQuery Functionality (Feat. Easy)

3.1 CQuery Web Interface

CQuery Web Interface 란?

- 테이블 카탈로그 제공
- Hive MetaStore에 구조화된 데이터를 SQL로 실행할 수 있는 에디터

자체적으로 개발한 이유?

- 사용자에게 쉽게 SQL을 사용할 수 있도록 하기 위해서
  - 쉬운 분석 환경 제공
  - 쉬운 SQL 작성

3.1.1 기능 설명

쉬운 분석 환경 제공

- Use case 목록 제공
- 개인 문서 저장

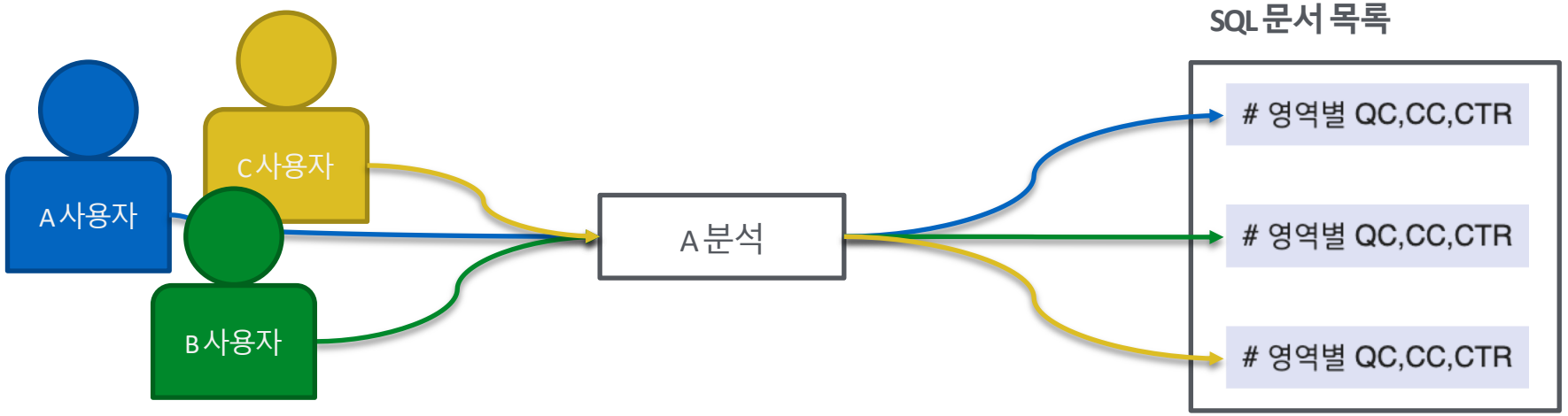
쉬운 SQL 작성

- 실시간 문법 체크 기능
  - 쿼리 실행 제어
  - 쿼리 정보 제공

3.1.2 쉬운 분석 환경 제공

Use case 문서를 고민하게 된 이유

- 여러 사용자가 비슷한 요구사항으로 SQL을 각각 작성
- 비슷한 요구사항이라면 기존 분석 SQL을 재사용하면 어떨까?
- 더욱이 SQL에 익숙하지 않은 사용자라면 기존 분석 자료가 큰 도움이 될 것이라 예상
- 분석에 많이 활용된 SQL을 문서화 하자.

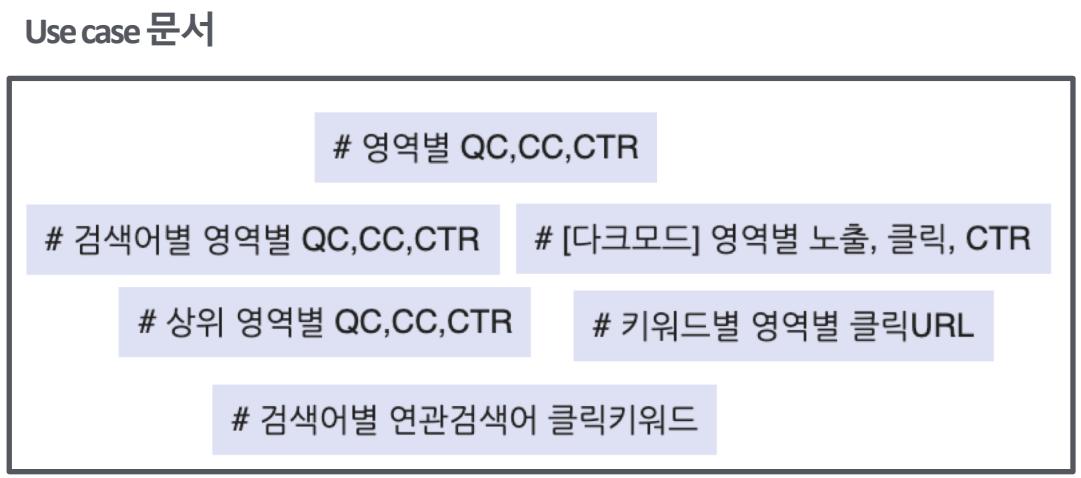


3.1.2 쉬운 분석 환경 제공

DEVIEW 2021

Use case 문서란?

- 사용자가 많이 분석하는 다양한 사례에 대해서 관리하는 SQL 문서

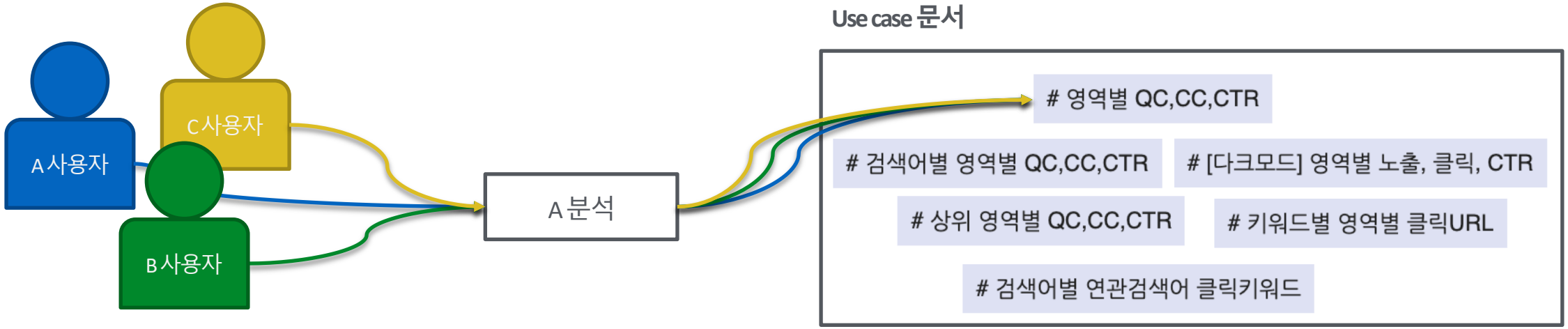


3.1.2 쉬운 분석 환경 제공

DEVIEW 2021

Use case 문서를 제공할 때 발생하는 이점

- 최적화된 SQL을 제공 -> 추출 속도와 리소스 절감
- SQL 재사용 증가 -> 분석 리소스 절감

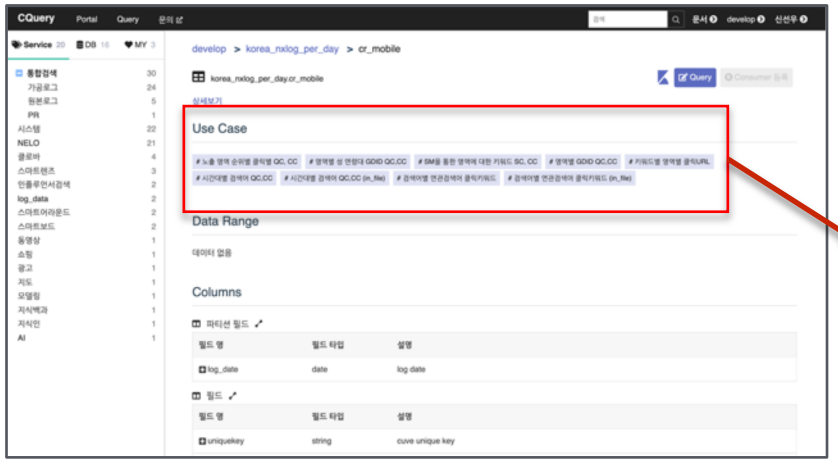


3.1.2 쉬운 분석 환경 제공

Use case 문서 제공 방식

- 테이블 정보를 확인할 때 해당 테이블로 어떠한 분석 사례가 있는지 쉽게 파악할 수 있도록 “테이블 상세 정보 페이지”에서 해당 테이블이 사용된 모든 Use case 목록 제공

테이블 상세 정보 페이지



해당 테이블이 사용된 모든 Use case 목록을 제공

Use Case

노출 영역 순위별 클릭별 QC, CC
영역별 성 연령대 GDID QC,CC
SM을 통한 영역에 대한 키워드 SC, CC
영역별 GDID QC,CC
키워드별 영역별 클릭URL
시간대별 검색어 QC,CC
시간대별 검색어 QC,CC (in_file)
검색어별 연관검색어 클릭키워드
검색어별 연관검색어 클릭키워드 (in_file)

3.1.2 쉬운 분석 환경 제공

DEVIEW 2021

Use case 문서 제공 방식

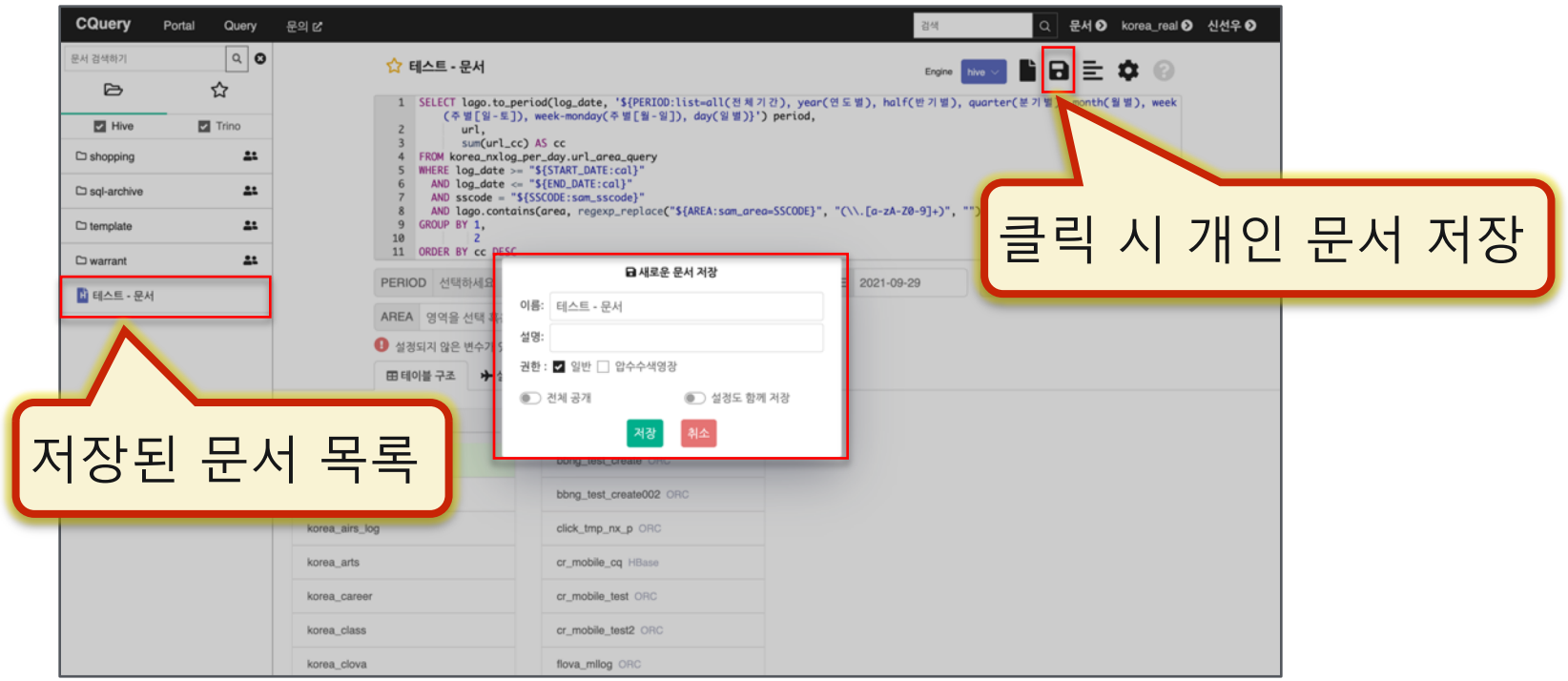
- Use case 문서 검색 기능을 통해서 원하는 문서를 찾을 수 있도록 제공

Use case 검색 페이지

3.1.2 쉬운 분석 환경 제공

개인 문서 저장

- Use case 문서에 일부 수정이 필요할 경우 수정하여 개인 문서로 저장



3.1.3 쉬운 SQL 작성

실시간 파싱 기능이 필요했던 이유

- 쿼리 실행 제어
  - SQL 문법 체크
  - 테이블 권한 제어

- 쿼리 정보 제공
  - 자동 완성 기능
  - 데이터 용량 산정 기능

3.1.3 쉬운 SQL 작성

실시간 파싱 방법

- Web Worker와 Antlr 파서를 활용하여 실시간 파싱 기능을 구현

- Web Worker
  - 특정 작업에 대해서 Main Thread와는 별도의 Thread 로 작업을 실행하는 기능

- Antlr 파서 (https://github.com/antlr/antlr4)
  - 구문 분석을 지원하는 파서
  - 특정 구문을 분석하여 AST 형태로 변환

3.1.3 쉬운 SQL 작성

Antlr 파서 활용 방식

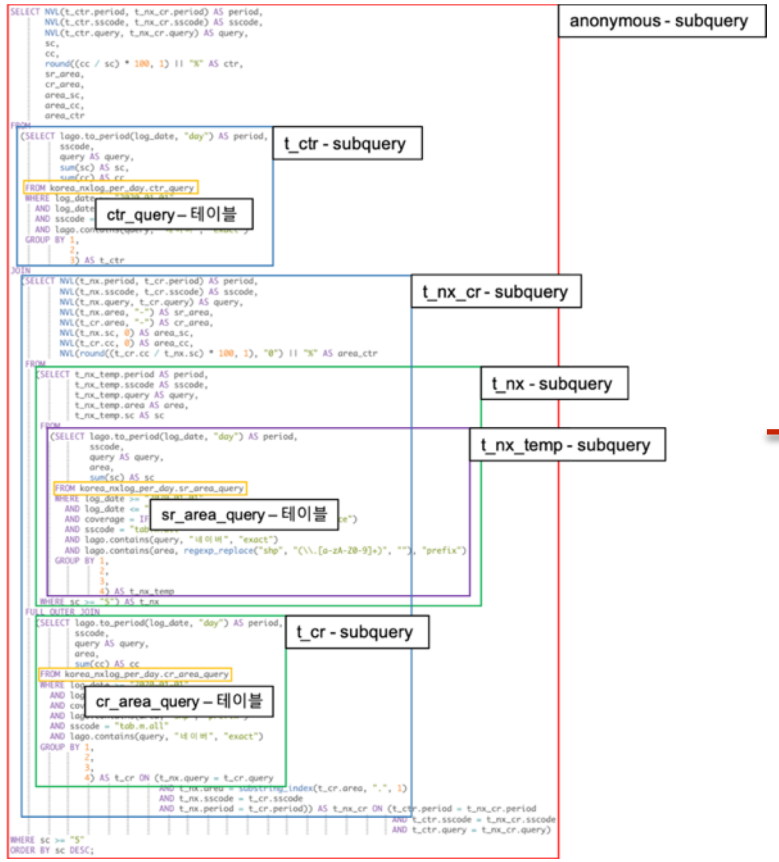
- Antlr 파서는 Listener 기능을 활용
  - Listener는 반환된 AST를 순회하여 원하는 구문에 필요한 동작이 실행되도록 지원

- Listener 기능을 활용하여 얻는 정보
  - 서브 쿼리의 필드 정보
  - Where 조건의 파티션 정보

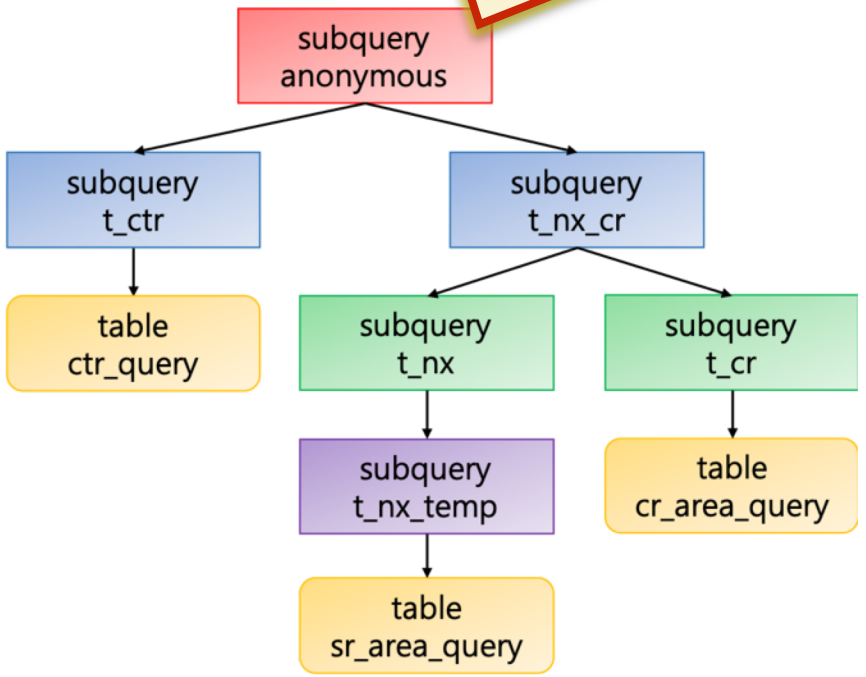
3.1.3 쉬운 SQL 작성

서브 쿼리의 필드 정보

- 모든 테이블 및 서브 쿼리의 필드 정보를 획득



서브 쿼리의 에디터 위치 정보와 필드 정보를 저장하고 있음



서브 쿼리 필드 정보 트리 구조

3.1.3 쉬운 SQL 작성

Where 조건의 파티션 정보

- 테이블을 활용한 From 절의 Where 조건 파티션 정보 획득

```
SELECT period,
       count(DISTINCT refine_query) AS u_query_count
FROM
  (SELECT lago.to_period(log_date, 'all') AS period,
          sscode,
          refine_query,
          substr(area, 1, 3) AS area,
          SUM(url_cc) AS area_url_cc
   FROM korea_nxlog_per_day.url_area_query
   WHERE log_date >= "${START_DATE:cal}"
     AND log_date <= "${END_DATE:cal}"
     AND sscode="${SSCODE:sam_sscode}"
     AND lago.match(area, "${AREA=ex. shp}", "prefix")
     AND url LIKE "${URL=%shopping.naver.com%}"
   GROUP BY 1,
            2,
            3,
            4) AS t_temp
WHERE area_url_cc >= ${U_CC_LIMIT=1000}
GROUP BY 1
ORDER BY 1;
```

Where 조건 안에 파악된 파티션 필드 조건

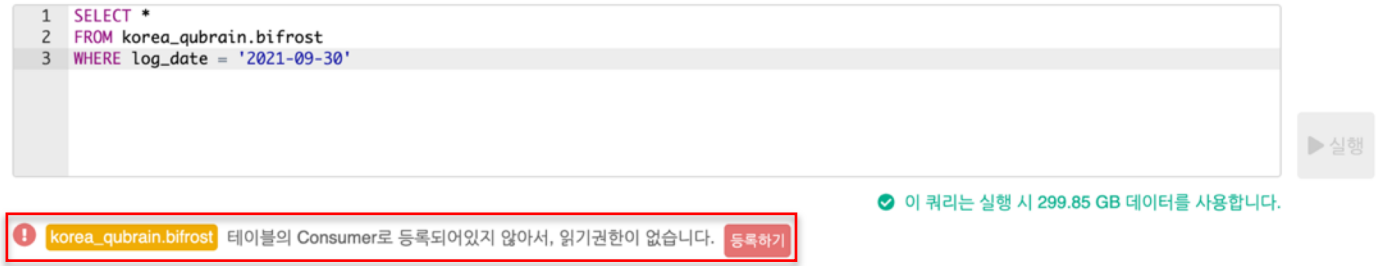
```
IERE log_date >= "${START_DATE:cal}"
AND log_date <= "${END_DATE:cal}"
```

3.1.3 쉬운 SQL 작성

쿼리 실행 제어

- SQL 문법 체크
  - 잘못된 SQL을 입력 시 오류 정보 제공

- 테이블 권한 제어
  - 권한이 없는 테이블을 사용 시 정보 제공



3.1.3 쉬운 SQL 작성

쿼리 정보 제공

- 자동완성 기능 제공
  - 테이블 및 필드 명 자동완성 정보 제공

- 데이터 용량 산정 기능 제공
  - 실시간으로 작성된 SQL에서 사용된 데이터 용량 제공
  - Hive MetaStore에 저장된 용량 정보를 활용

3.1.3 쉬운 SQL 작성

Hive MetaStore에 파티션 통계 정보 입력하는 방법

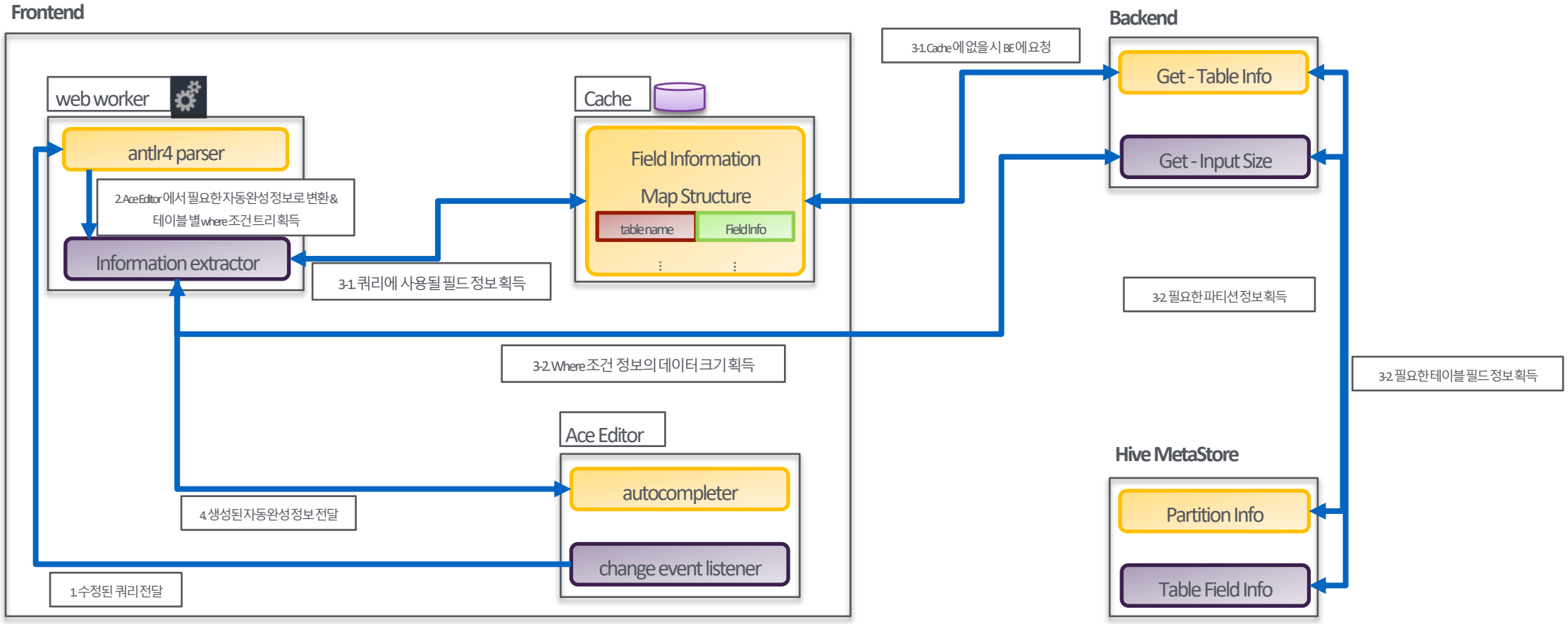
- hive.stats.autogather 옵션을 켠 상태에서 Insert 명령어 실행 시 자동으로 통계 정보가 입력됨

- analyze table 명령어로 직접 통계 정보를 입력할 수 있음

```
ANALYZE TABLE `korea_nxlog_per_day`.`nx_mobile`
PARTITION(`log_date`='2021-01-01')
COMPUTE STATISTICS;
```

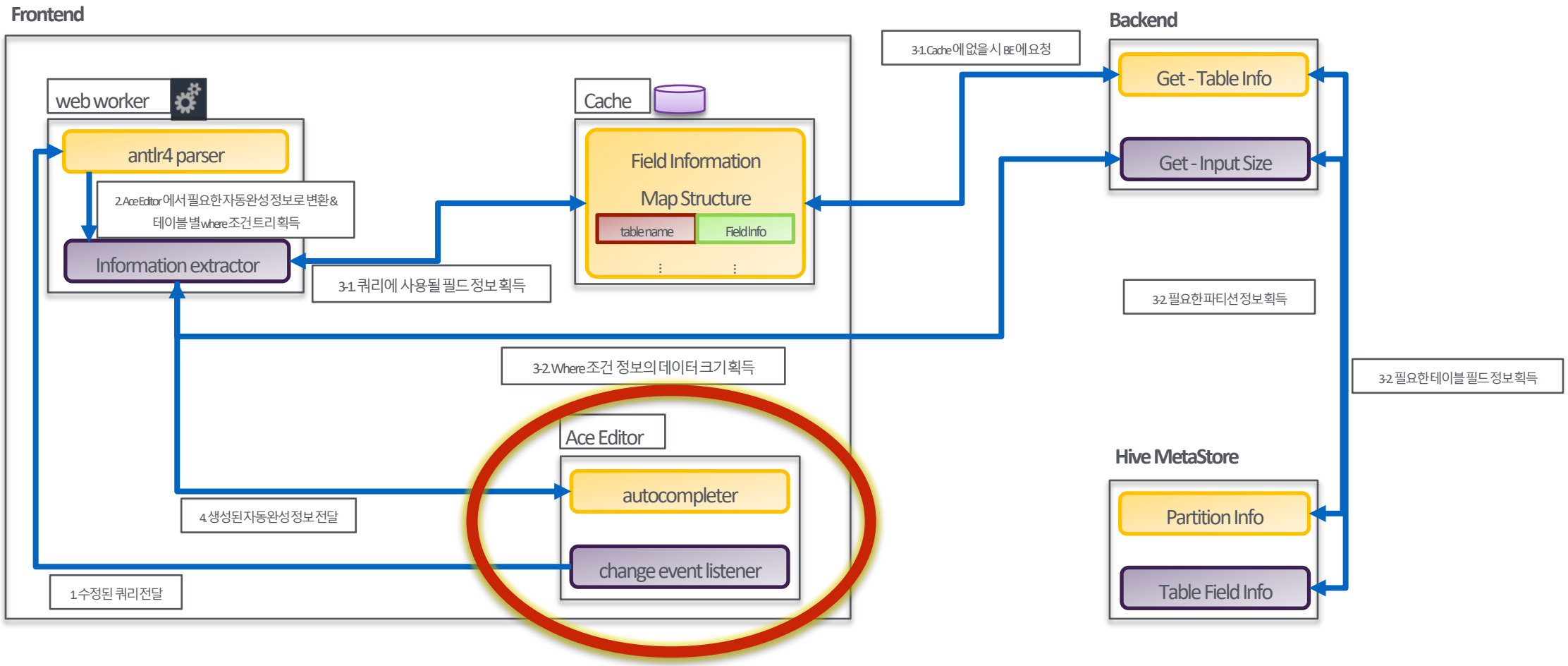
3.1.3 쉬운 SQL 작성

실시간 파싱 아키텍처 흐름



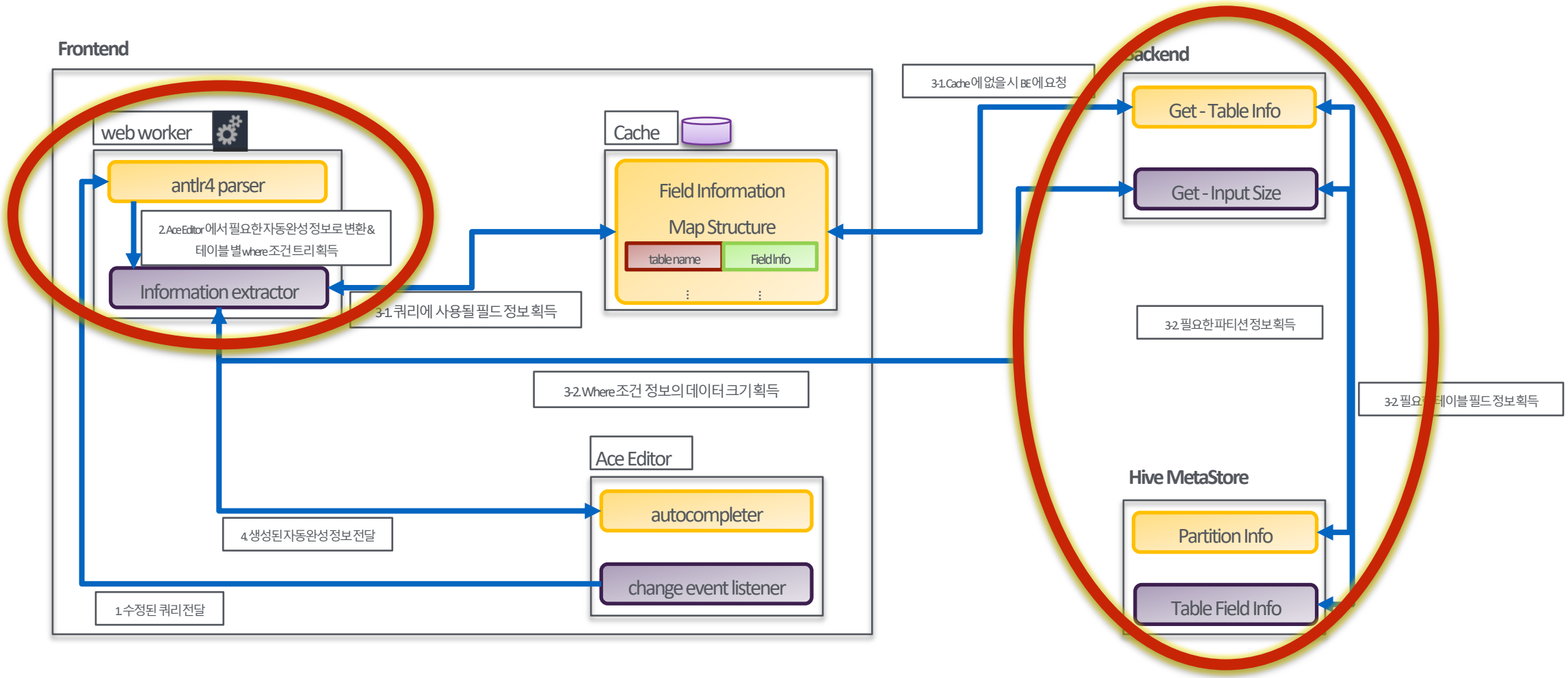
3.1.3 쉬운 SQL 작성

전체 적용된 아키텍처 흐름



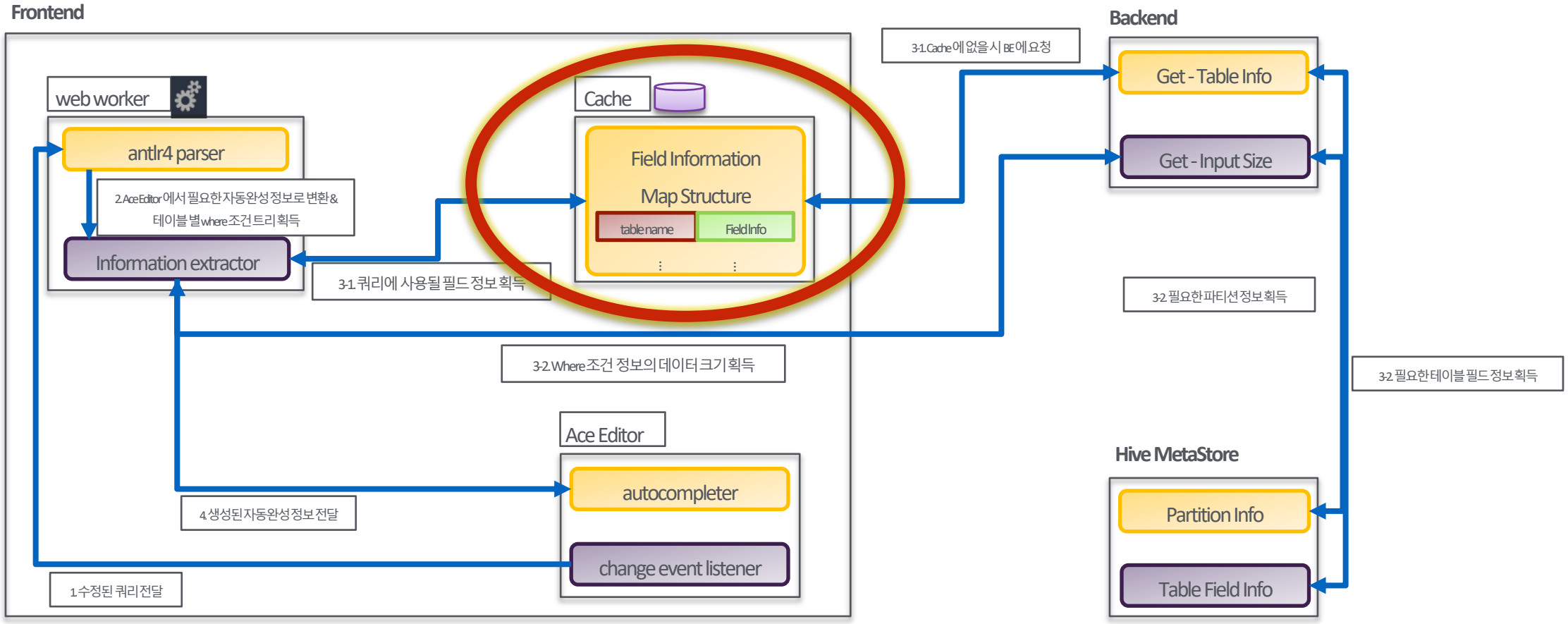
3.1.3 쉬운 SQL 작성

전체 적용된 아키텍처 흐름



3.1.3 쉬운 SQL 작성

전체 적용된 아키텍처 흐름



3.2 User Defined Function

UDF를 사용한 이유

- 비지니스 로직을 SQL로 쉽게 작성할 수 있도록 하기 위해서
- 예를 들어서 검색 로그에 사용자가 직접 입력한 검색어를 정규화하는 함수 등을 제공

UDF (User Defined Function) 란?

- 기본적으로 제공하는 함수(Function) 이외의 사용자가 직접 Custom하게 만든 함수(Function)

3.2.1 UDF 종류

Generic Function (UDF)

- 하나의 행에서 동작하며 하나의 출력을 반환

Aggregate Function (UDAF)

- 여러 행의 데이터를 집계하여 하나의 출력을 반환

Table Function (UDTF)

- 하나의 행에서 동작하며 여러 행의 데이터를 반환
- Trino는 UDTF 함수를 지원하지 않고 UNNEST 함수를 통해서 지원

3.2.2 등록 방법

DEVIEW 2021

Hive 등록 방법

- hive-site.xml 설정 값 중 hive.aux.jars.path 혹은 hive.reloadable.aux.jars.path 설정된 경로에 개발한 Jar 파일 추가

```
<property>
  <name>hive.aux.jars.path</name>
  <value>/naver/search-env/package/hive3/auxlib</value>
</property>
<property>
  <name>hive.reloadable.aux.jars.path</name>
  <value>/naver/search-env/package/hive3/auxlib</value>
</property>
```

해당 옵션은 reload 명령어를 사용하여 서버 재시작 없이 신규 jar 파일 가능

- Create Function 명령어로 UDF 등록

```
CREATE FUNCTION functionName AS 'classpath';
```

3.2.2 등록 방법

DEVIEW 2021

Trino 등록 방법

- ${TRINO_HOME}/plugins 경로에 Jar 파일을 추가
- Plugin 클래스를 implements 받아 getFunctions 함수에 해당 클래스 등록

```
public class UdfPlugin implements Plugin {

    @Override
    public Set<Class<?>> getFunctions() {
        return ImmutableSet.<Class<?>>builder()
            .add(LagoUDFConvertRangeLong.class)
            .add(LagoUDFDecodeURL.class)
            .add(LagoUDFFilter.class)
            .add(LagoUDFFlatMapSource.class)
```

3.3 Develop Tool

PyCQuery 툴 (https://github.com/naver/PyCQuery)

- Python 환경 지원
- PyHive와 minikerberos를 사용하여 naver opensource로 공개
  - PyHive (https://github.com/dropbox/PyHive)
  - minikerberos (https://github.com/skelsec/minikerberos)
- 추가 기능
  - Zookeeper로 HA가 적용된 Hiveserver2 지원
  - pure python Kerberos 인증 지원 (HTTP 방식만)

3.3.1 PyCQuery 개발

Zookeeper로 HA가 적용된 Hiveserver2 지원

- Hive Jdbc를 참고하여 Hiveserver2 접근하는 방식과 동일하게 구현
- Hive Jdbc는 Zookeeper에 등록된 정보를 “랜덤”하게 들고 오는 방식

```
if is_zookeeper:
        # It randomly shuffles node information stored in zookeeper.
        remaining_nodes = self._get_hiveserver2_info_with_zookeeper(host, port, zookeeper_name_space)
        random.shuffle(remaining_nodes)
else:
        # Direct access to host and port if not zookeeper.
        remaining_nodes = [{'host': host, 'port': port}]

# Access nodes sequentially and if they fail, access other nodes.
while len(remaining_nodes) > 0:
        node = remaining_nodes.pop()
        ...
```

Zookeeper 에 등록된 장비 정보 획득

랜덤하게 하나씩 장비를 접근

3.3.1 PyCQuery 개발

DEVIEW 2021

Pure Python Kerberos 라이브러리 적용

- 기존 Python kerberos 라이브러리를 활용하게 되면 native gssapi 라이브러리를 활용하기 때문에 해당 라이브러리에 의존성이 발생

- 일부 실행 환경에서 정상 동작하지 않는 경우가 발생

```
Building from Cython files...
Traceback (most recent call last):
  File "setup.py", line 173, in <module>
    GSSAPI_LIB = ctypes.CDLL(os.path.join(main_path, main_lib))
  File "/naver/search-env/package/python-3.5.4/lib/python3.5/ctypes/__init__.py", line 351, in __init__
    self._handle = _dlopen(self._name, mode)
OSError: /usr/lib64/libgssapi_krb5.so: symbol krb5_authdata_import_attributes, version krb5_3_MIT not defined in file libkrb5.so.3 with link time reference
```

3.3.1 PyCQuery 개발

Pure Python Kerberos 라이브러리 적용

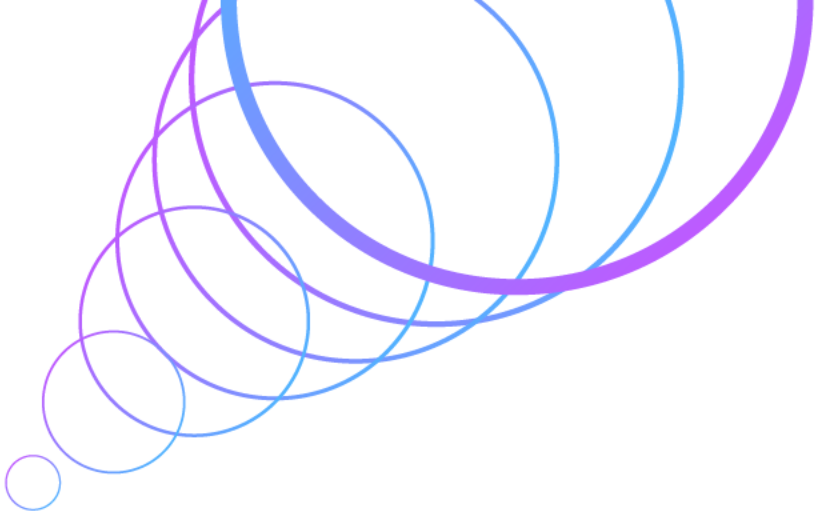
- Pure python kerberos 로 개발된 minikerberos 라이브러리를 활용

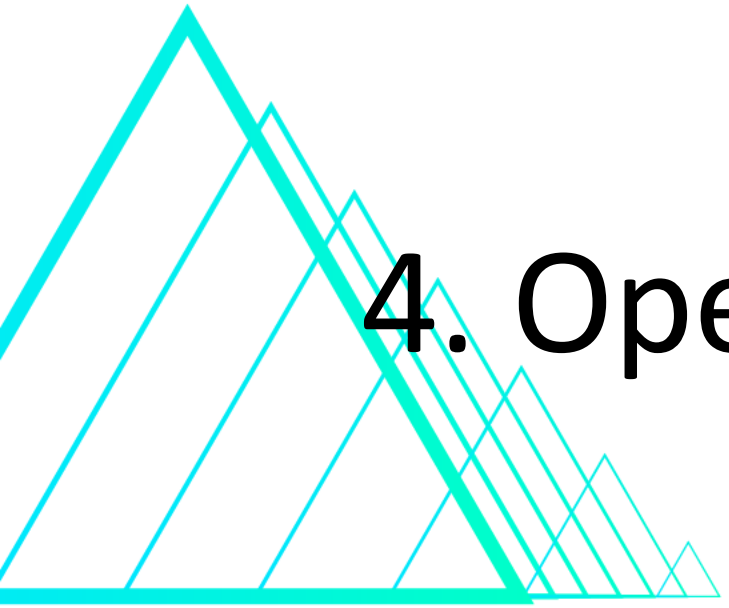
```
target = KerberosTarget()
...
client.get_TGT()
spn = KerberosSPN()
...
tgs, enc, key = self.kerb_client.get_TGS(spn)
token = self.kerb_client.construct_apreq(tgs=tgs, encTGSRepPart=enc, sessionkey=key, flags=0)
headers = {'Authorization': 'Negotiate ' + KRB5Token(token).get_apreq_token()}
self._transport.setCustomHeaders(headers)
```

TGT 인증

TGS 인증

AP 인증





4. Open Source Contributions

4.1 기능 추가/성능 개선

DEVIEW 2021

Hive

- https://issues.apache.org/jira/browse/HIVE-24948 2.2.3.1 Bucket Pruning 개선
- https://issues.apache.org/jira/browse/HIVE-23458 HiveServer2에서 ScheduledThreadPoolExecutor 매번 생성하지 않도록 통합 Scheduler Pool 적용

Tez

- https://issues.apache.org/jira/browse/TEZ-4205 Tez Task 상에서 RM DT 사용할 수 있도록 기능 추가

gohive

- https://github.com/beltran/gohive/pull/87 Hive Zookeeper 기반 HiveServer2 HA 연결 기능 추가

minikerberos

- https://github.com/skelsec/minikerberos/pull/20 GSS-API AP-REQ 토큰 생성 로직 추가

4.2 버그 수정

DEVIEW 2021

Hive

- https://issues.apache.org/jira/browse/HIVE-23153 Zookeeper 기반 Hive HA 기능 : Kerberos 적용 환경에서 deregister 명령 오류
- https://issues.apache.org/jira/browse/HIVE-24713 Zookeeper 기반 Hive HA 기능 : ZK 장애 후 복구시 deregister 명령에 의해 HiveServer2가 안 내려가는 문제
- https://issues.apache.org/jira/browse/HIVE-23581 Zookeeper 기반 Hive HA 기능 : JDBC Driver 버그
- https://issues.apache.org/jira/browse/HIVE-23954 Hive 3에서 distinct count SQL 수행이 잘못되는 에러

trino-go-client

- https://github.com/trinodb/trino-go-client/pull/5 hidden arg 관련 버그 1
- https://github.com/trinodb/trino-go-client/pull/7 hidden arg 관련 버그 2

minikerberos

- https://github.com/skelsec/minikerberos/pull/19 ccache 에러 처리 로직 강화 1
- https://github.com/skelsec/minikerberos/pull/21 ccache 에러 처리 로직 강화 2

DEVIEW 2021

We're hiring

AI & Data Platform 소개

https://naver-career.gitbook.io/kr/service/search/ai-and-data-platform

경력 등록

https://d2.naver.com/news/7591059

Thank You