

# Stat 테이블 SparkJob 개선

## Stat 테이블 생성 및 검증

NAVER WEBTOON 인턴 김형근

# 목차

## 1. 과제 배경

1) Data flow

2) 목표

3) 사용 환경

## 2. Stat 테이블 생성

1) 기본 구조

2) 코드 흐름

3) 결과

## 3. Stat 테이블 검증

1) 기본 구조

2) 코드 흐름

3) 예시

4) 결과

## 4. 기술적 이슈

1) 검증 로직 구상

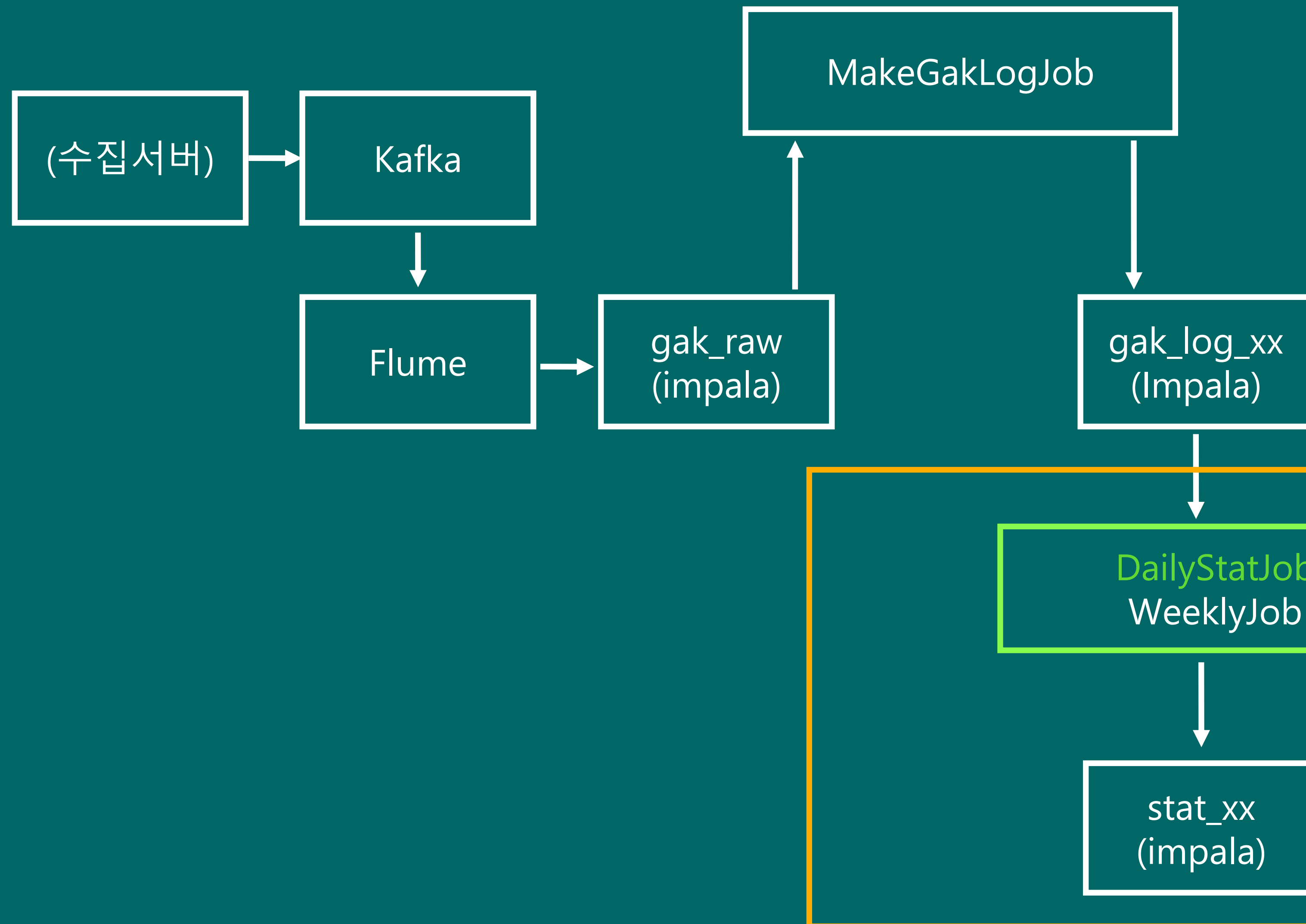
2) 테이블 생성 Job 재구성

3) Impala 명령어 전달 법

## 5. 배운점 및 아쉬운 점

# 1. 과제 배경

## 1) Data Flow



### 과제 배경

- Impala로직이 다른 로직으로도 대체 가능해야 함
- SQL을 이용해서 분석가 팀이 이해 할수 있도록 진행

### 목표

- **SparkJob**으로 기존 테이블 생성 과정을 재구성
- 기존 Job과 비교하여 **데이터 검증과정**이 필요

# 1. 과제 배경

## 2) 목표

- Impala 로직을 수행하는 기존 `DailyStatJob`의 `stat테이블` 생성 과정을 `SparkJob`으로 재구성
- 이 때, `Spark SQL`을 이용하여 로직을 구성할 것
- `SparkJob`으로 생성한 `stat테이블` 내용이 `DailyStatJob 테이블`과 동일한 내용인지 검증

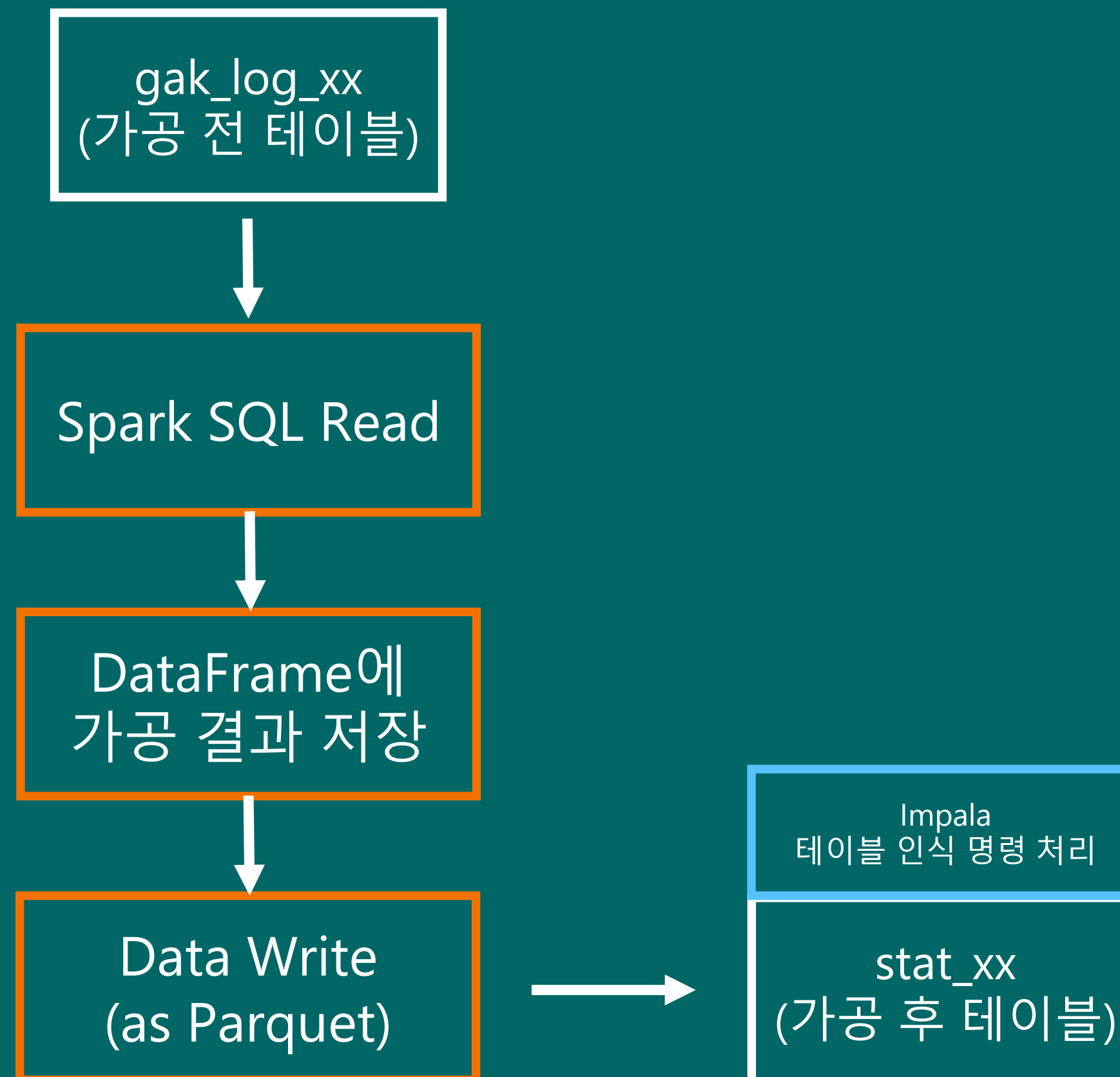
# 1. 과제 배경

## 3) 사용 환경

- Hadoop 2.6.0 (YARN 포함)
- Spark 2.3.0
- Hive 1.x
- Impala 2.x
- Scala 2.11.x
- IntelliJ에서 환경 개발

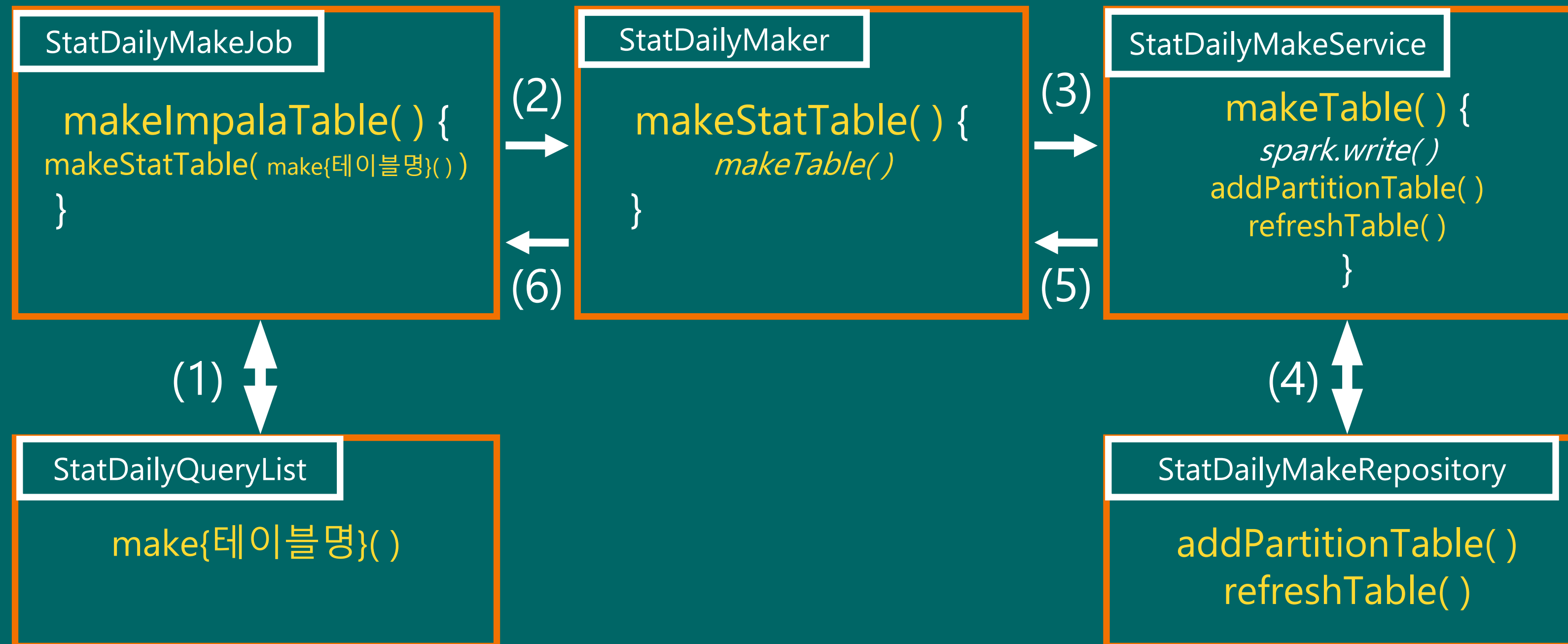
## 2. Stat 테이블 생성

### 1) 기본 구조



## 2. Stat 테이블 생성

### 2) 코드 흐름



(1) `makeImpalaTable()` 에서 `make{테이블명}()`를 호출하여 SQL 쿼리구문 반환

(2) 반환된 `make{테이블명}()` SQL 쿼리구문을 이용해 `makeStatTable()` 호출

(3) `makeStatTable()` 에서 `makeTable()`을 호출하여 상세 데이터 쓰기를 진행

(4) `addPartitionTable()` 을 호출하여 새로운 파티션 추가를 Impala에서도 인식토록 하고, 그 후 새로고침

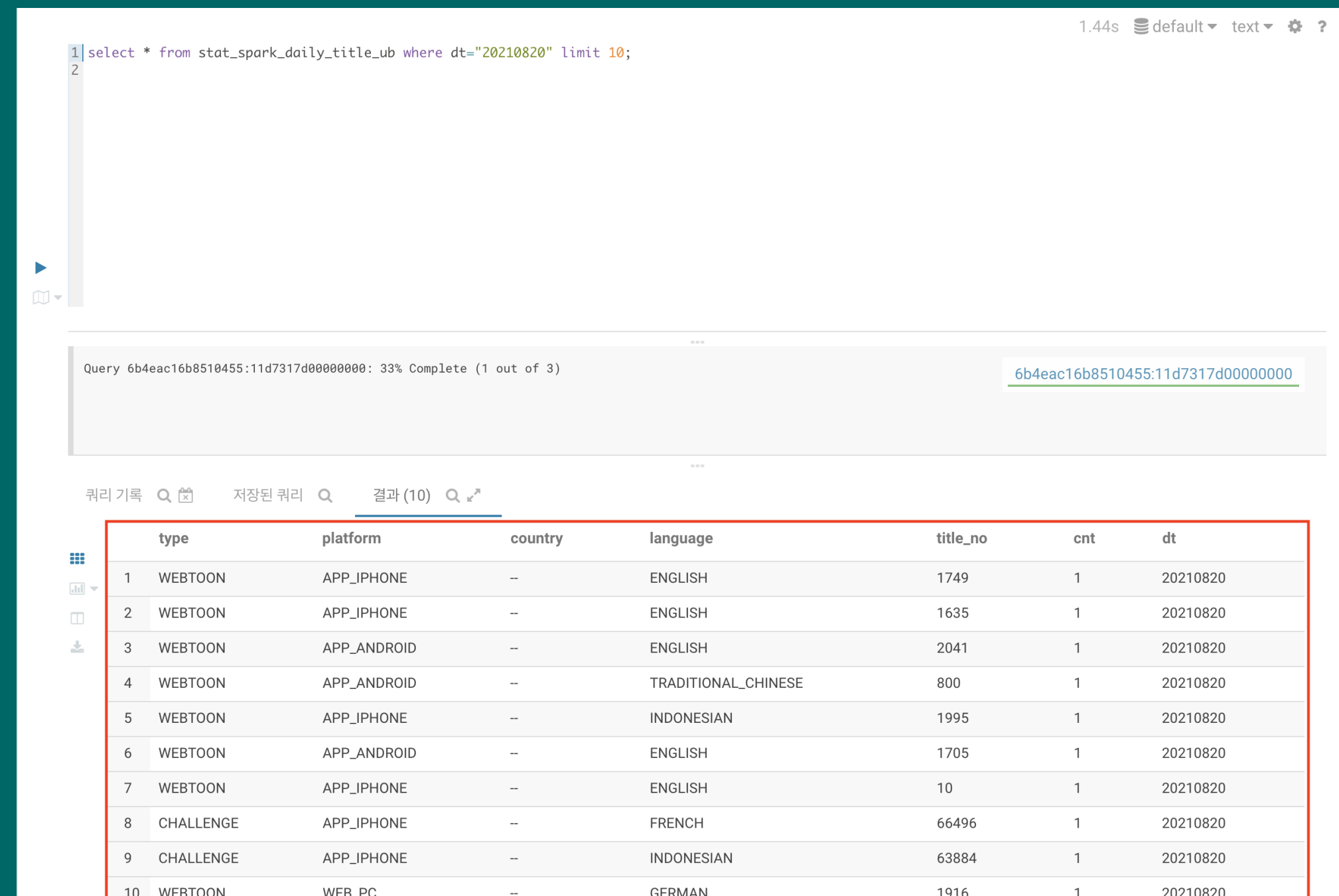
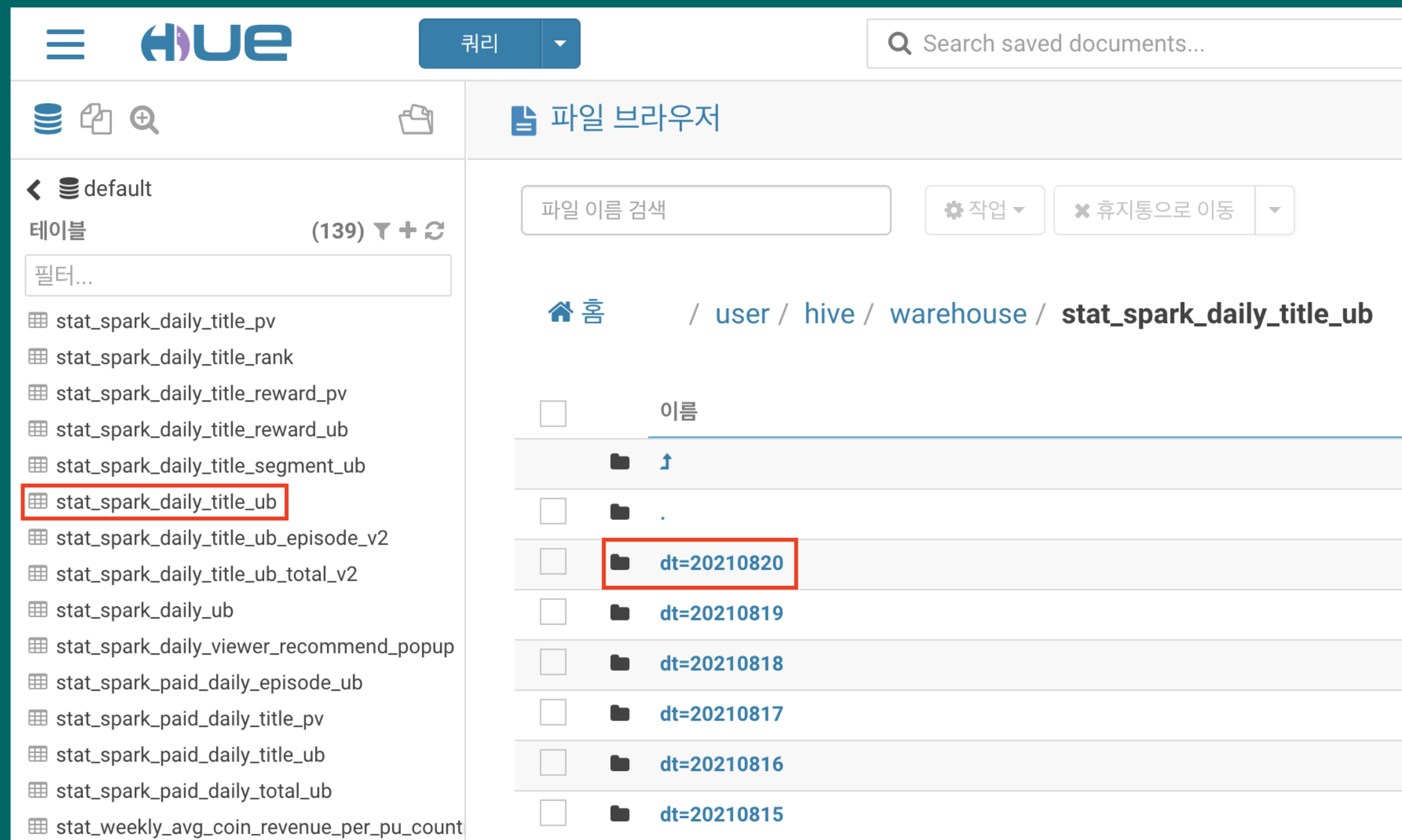
(5) `makeTable()` 기능이 모두 완료된 후, `makeStatTable()`로 돌아온다.

(6) `makeStatTable()` 기능이 모두 완료된 후, `makeImpalaTable()`로 돌아온다. 그 후, Job 종료

# 2. Stat 테이블 생성

## 3) 결과

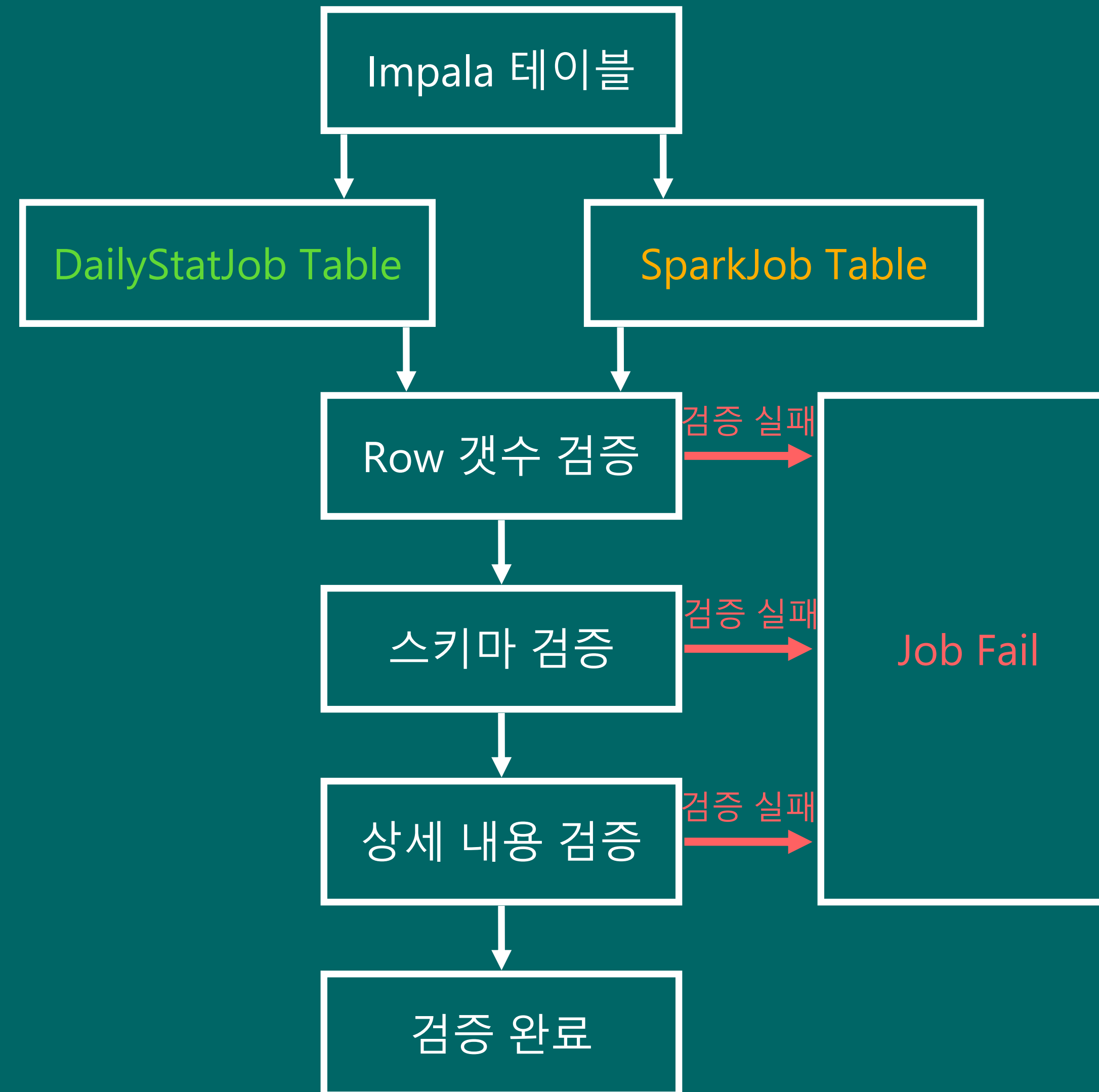
- 결과 : **SparkJob**을 이용한 Stat 테이블 생성 작업 **완료**
- 테이블 생성 : 22개 완료 (**DailyStatJob**의 Stat테이블 갯수가 22개)





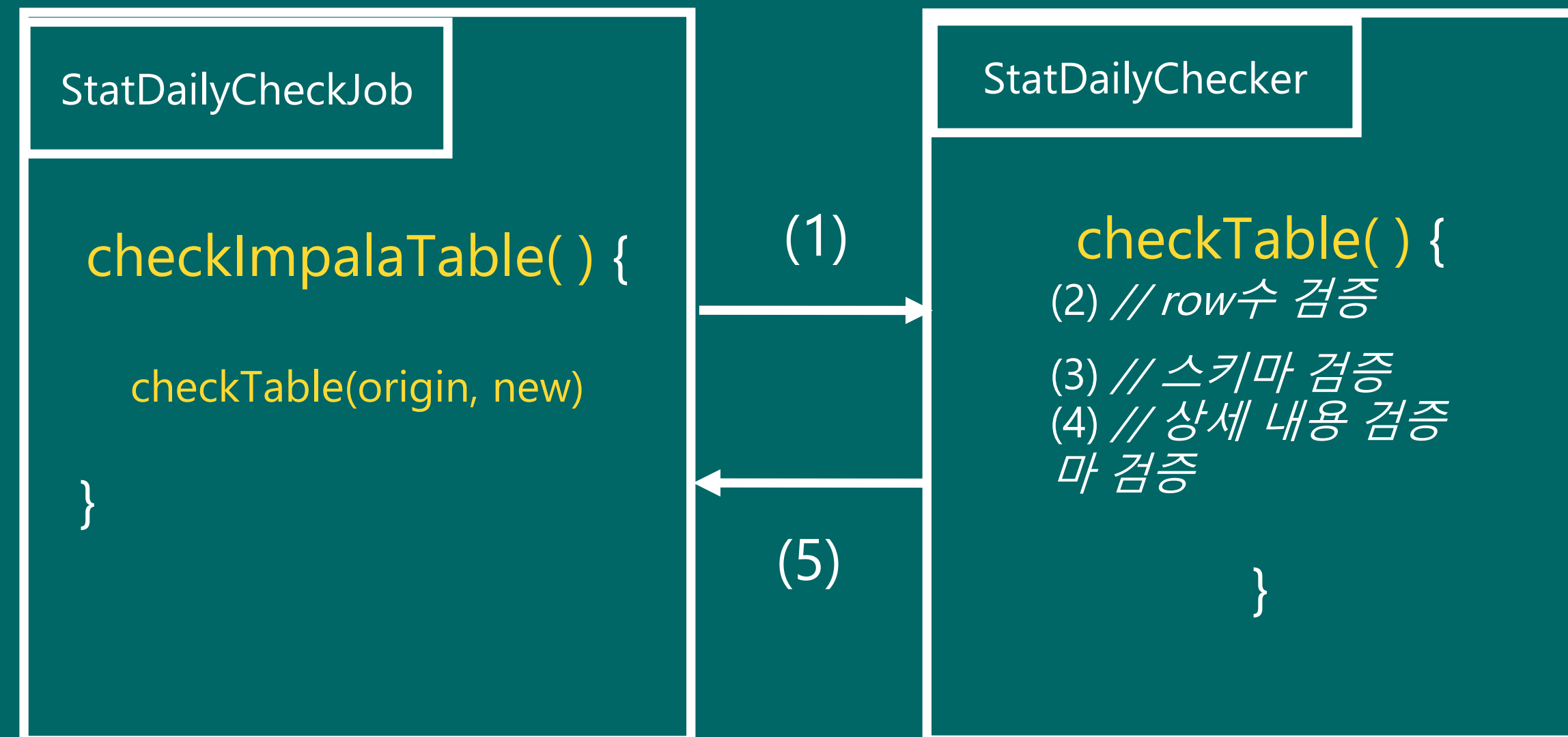
# 3. Stat 테이블 검증

## 1) 기본 구조



# 3. Stat 테이블 검증

## 2) 코드 흐름



- (1) `checkTable( )` 를 호출하여 테이블 검증 작업 시작
- (2) `checkTable( )` 에서 테이블 간의 row 수를 비교
- (3) `checkTable( )` 에서 테이블 간의 스키마를 비교
- (4) `checkTable( )` 에서 테이블 간의 상세 내용을 비교
- (5) 모든 검증이 완료되면 `checkImpalaTable( )`로 돌아온 후, Job 종료

# 3. Stat 테이블 검증

## 3) 예시 (검증성공)

```
INFO StatDailyChecker$:76 - "gw.stat daily title ub/20210820" and "default.stat_spark_daily_title_ub/20210820" are equal Table
```

=> 의미 : gw.stat\_daily\_title\_ub와 default.stat\_spark\_daily\_title\_ub는 서로 같은 테이블이다.

- 1) row 수 검증 (O) : gw.stat\_daily\_title\_ub와 default.stat\_spark\_daily\_title\_ub는 서로 같은 row를 가졌다.
- 2) 스키마 검증 (O) : gw.stat\_daily\_title\_ub와 default.stat\_spark\_daily\_title\_ub는 서로 같은 스키마를 가졌다.
- 3) 내용 검증 (O) : gw.stat\_daily\_title\_ub와 default.stat\_spark\_daily\_title\_ub는 서로 같은 테이블 내용을 가졌다.

# 3. Stat 테이블 검증

## 3) 예시 (검증성공)

```
INFO StatDailyCheckJob$:120 - All Table check is completed
```

=> 의미 : 모든 테이블들에 대한 **검증을 성공적으로 완료**하면 위와 같은 메시지를 출력

# 3. Stat 테이블 검증

## 3) 예시 (검증실패 : row count)

```
java.util.NoSuchElementException: Table row counts are not same :  
gw.stat_daily_title_ub/20210820 = 91  
default.stat_spark_daily_title_ub_countFail/20210820 = 297
```

- row 수 검증 (X) : gw.stat\_daily\_title\_ub와 default.stat\_spark\_daily\_title\_ub는 서로 다른 row를 가졌다.

# 3. Stat 테이블 검증

## 3) 예시 (검증실패 : schema 검증)

```
java.lang.RuntimeException: Table schemas are not same :  
gw.stat_daily_title_ub/20210820 = StructType(StructField(type,StringType,true), StructField(platform,StringType,true), StructField(country,StringType,true), StructField(language,  
default.stat_spark_daily_title_ub_schemaFail/20210820 = StructType(StructField(platform,StringType,true), StructField(type,StringType,true), StructField(country,StringType,true),
```

- row 수 검증 (O) : gw.stat\_daily\_title\_ub와 default.stat\_spark\_daily\_title\_ub는 서로 같은 row를 가졌다.

- 스키마 검증 (X) : gw.stat\_daily\_title\_ub와 default.stat\_spark\_daily\_title\_ub는 서로 다른 스키마를 가졌다.

\* 스키마는 컬럼 종류, 컬럼의 데이터 타입, 컬럼의 이름 뿐 아니라 컬럼 순서까지 모두 같아야 한다.

# 3. Stat 테이블 검증

## 3) 예시 (검증실패 : 상세내용 검증)

```
java.util.NoSuchElementException: Table data contents are not same :  
gw.stat_daily_title_ub/20210817 grouped rows: 90  
default.stat_spark_daily_title_ub_contentFail/20210817 grouped rows: 90  
Intersection grouped Rows : 5
```

- row 수 검증 (O) : gw.stat\_daily\_title\_ub와 default.stat\_spark\_daily\_title\_ub는 서로 같은 row를 가졌다.
- 스키마 검증 (O) : gw.stat\_daily\_title\_ub와 default.stat\_spark\_daily\_title\_ub는 서로 같은 스키마를 가졌다.
- 내용 검증 (X) : gw.stat\_daily\_title\_ub와 default.stat\_spark\_daily\_title\_ub는 서로 다른 테이블 내용을 가졌다.

# 3. Stat 테이블 검증

## 4) 결과

- 결과 : **SparkJob**을 이용한 Stat 테이블 검증 작업 **완료**
- 테이블 검증 : 22개 완료 (원본 **DailyStatJob**의 Stat테이블 갯수가 22개)

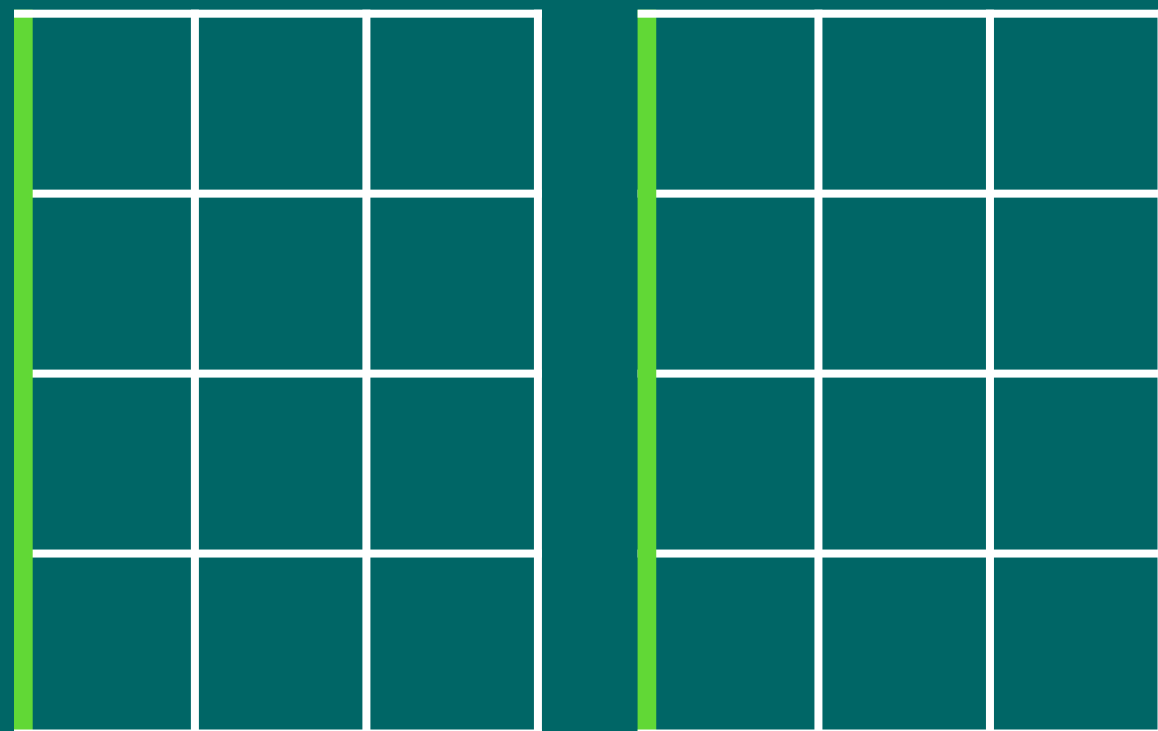


# 4. 기술적 이슈

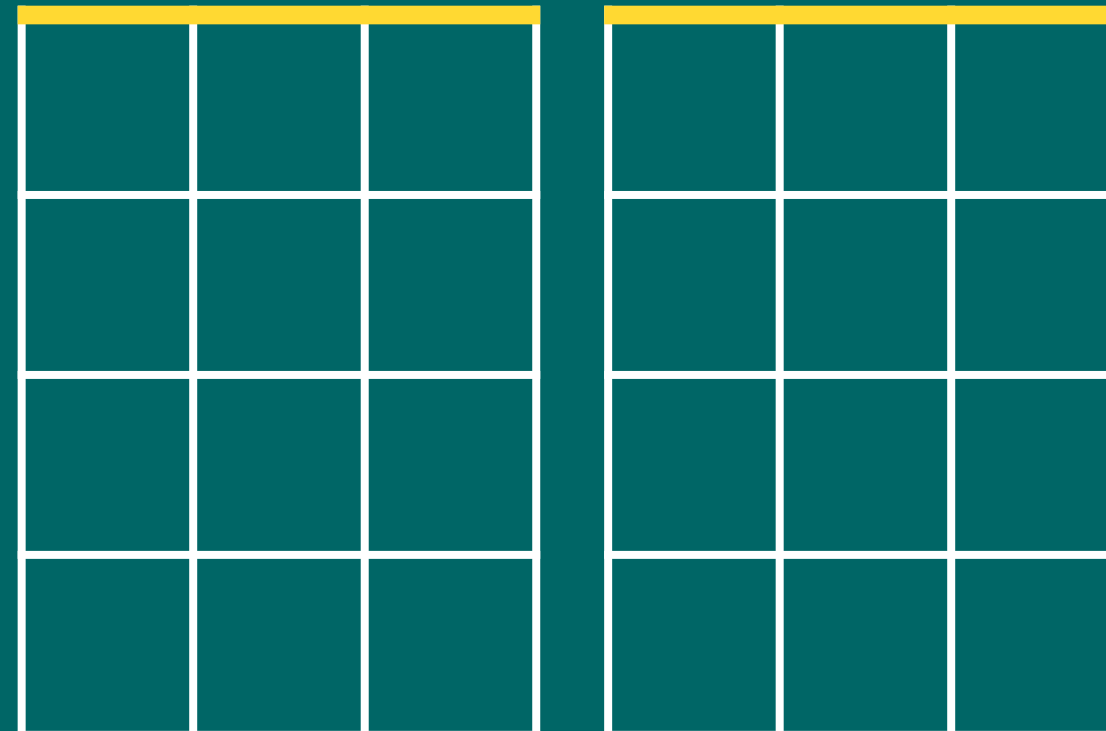
## 1) 검증 로직 구상

- 두 테이블이 같음을 보이기 위해서는 **형태와 내용이 모두 동일**해야 한다는 점을 캐치
- **1:1 매칭을 통해 테이블 모두 비교**해보기로 결정

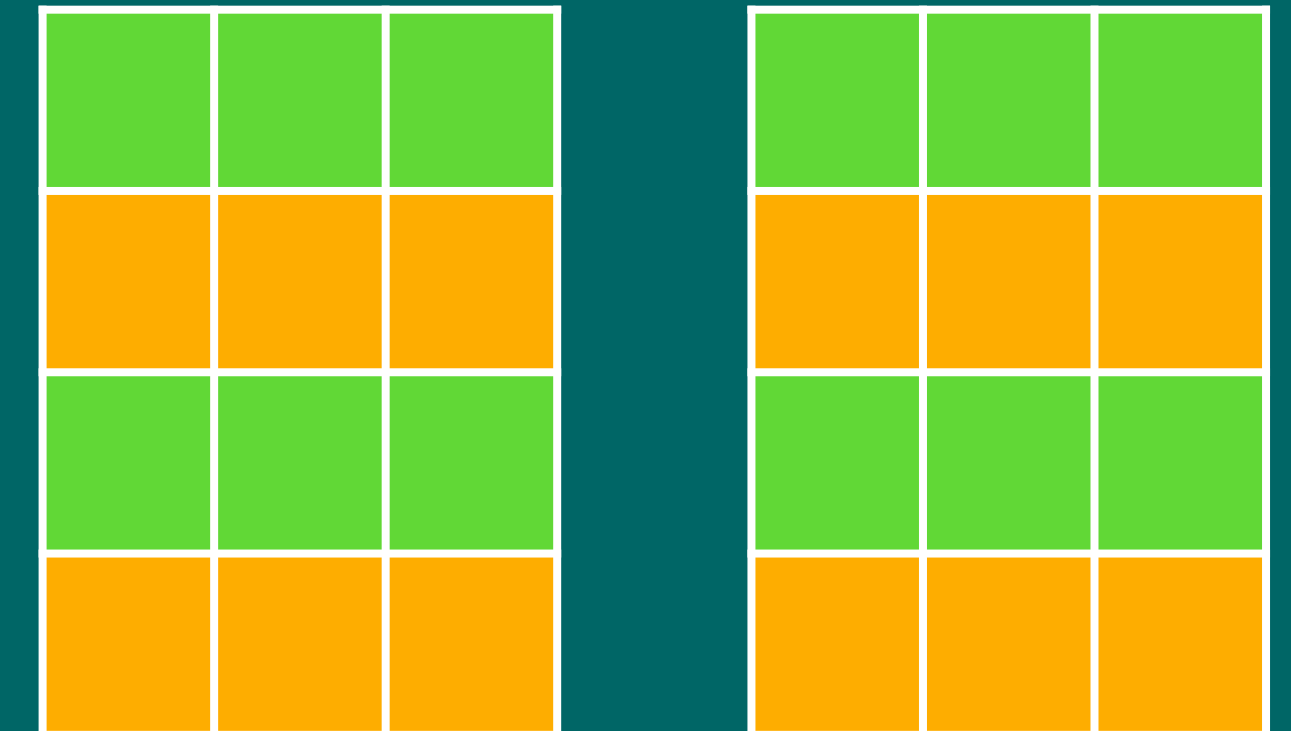
1단계 : Row 갯수 검증



2단계 : 스키마 검증(종류, 데이터타입, 컬럼순서)



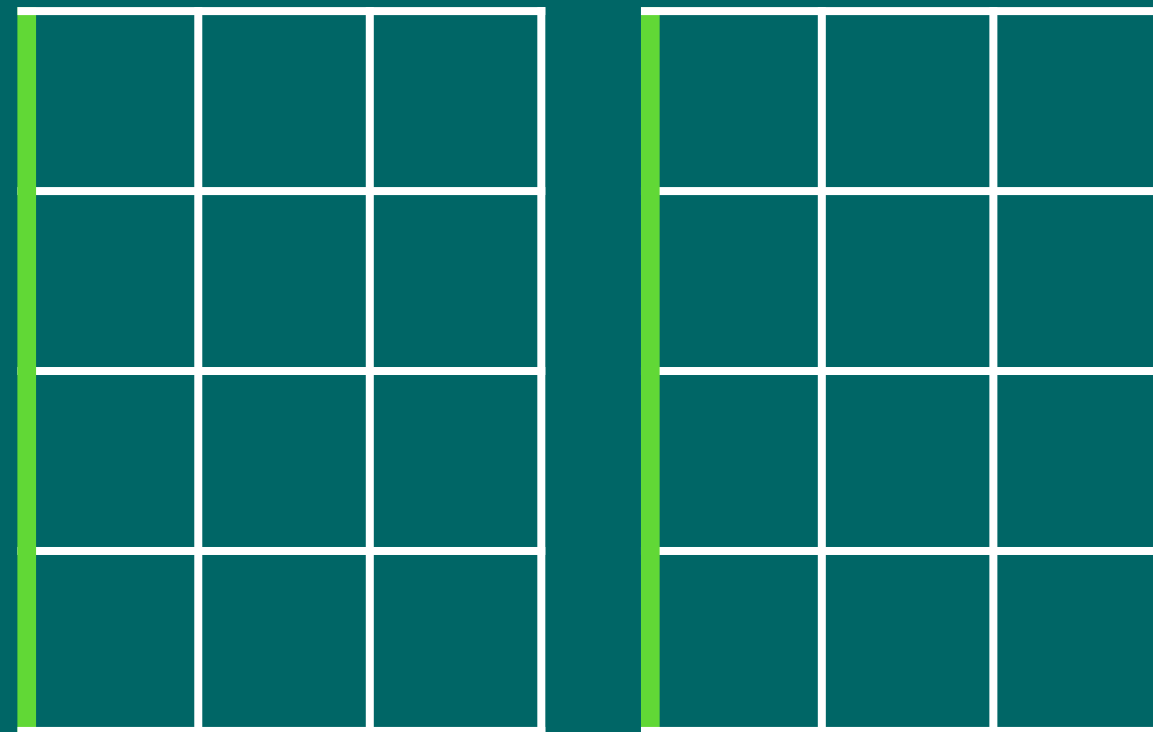
3단계 : 상세 내용 검증



# 4. 기술적 이슈

## 1-1) 검증 로직 구상 : row수 검증

- 두 테이블이 가진 데이터 갯수가 모두 같아야 함
- Spark 함수 중, count( ) 함수를 통해 두 테이블의 데이터 갯수를 구함





# 4. 기술적 이슈

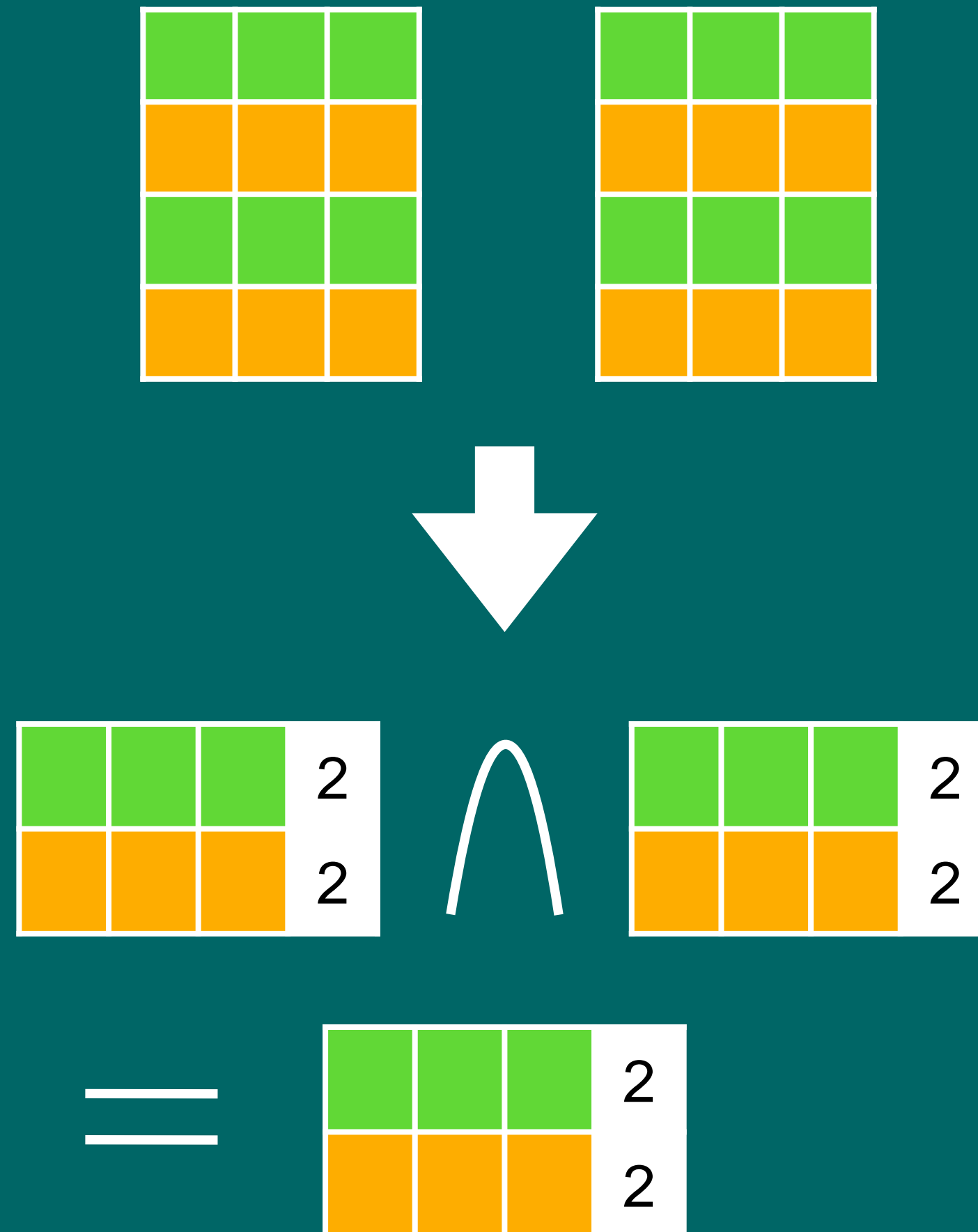
## 1-2) 검증 로직 구상 : 스키마 검증

- 두 테이블이 가진 **스키마가 같은지 확인**
- Spark 함수 중, **schema.eqauls()** 를 통해 **두 테이블의 스키마를 비교**  
(데이터타입, 종류, 순서까지 모두 같아야 한다.)


# 4. 기술적 이슈

## 1-3) 검증 로직 구상 : 내용 검증

- row 별 빈도수는 모든 컬럼을 group by 한 후, count(\*)을 통해 구함



# 4. 기술적 이슈

## 1-4) 검증 로직 구상 : 로직의 장단점

- 장점

- 1) schema.equal를 통해서 컬럼 종류, 컬럼 데이터타입 뿐 아니라 **순서까지 완전히 동일함을 검증**하므로 우려 없음
- 2) 스키마의 동일함을 전제하에, 교집합 operator를 통해 **group by된 row 빈도수까지 모두 계산**하므로 정확성이 높음

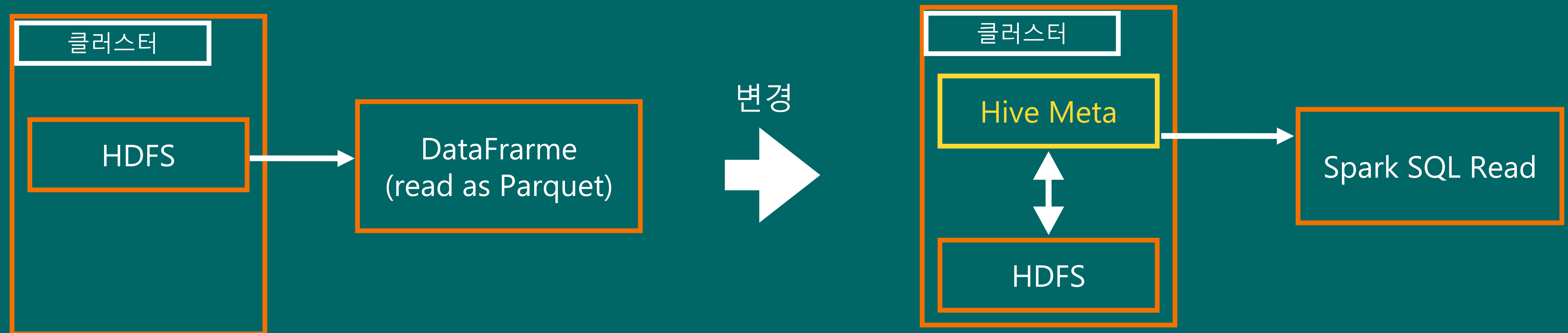
- 단점

- 1) 적은 수의 데이터로는 큰 문제는 없으나, **대용량 데이터에 대해서는 리소스 측면에서 우려**
- 2) 특히, row를 그룹 짓는 group by 연산은 **클러스터 내에서 수많은 shuffle이 발생**하므로 부담 증가율 높음

# 4. 기술적 이슈

## 2) 테이블 생성 Job 재구성

- 중간리뷰 이전까지는 `spark.read.parquet()`을 통해 직접 파일단위로 읽음
- 이후, `hive-site.xml` 설정파일을 통해 **hive metastore**에 직접 접근할 수 있게되었음
- 직접 **hive metastore**에 접근하여 SQL을 이용해 데이터를 읽을 수 있게 되었음



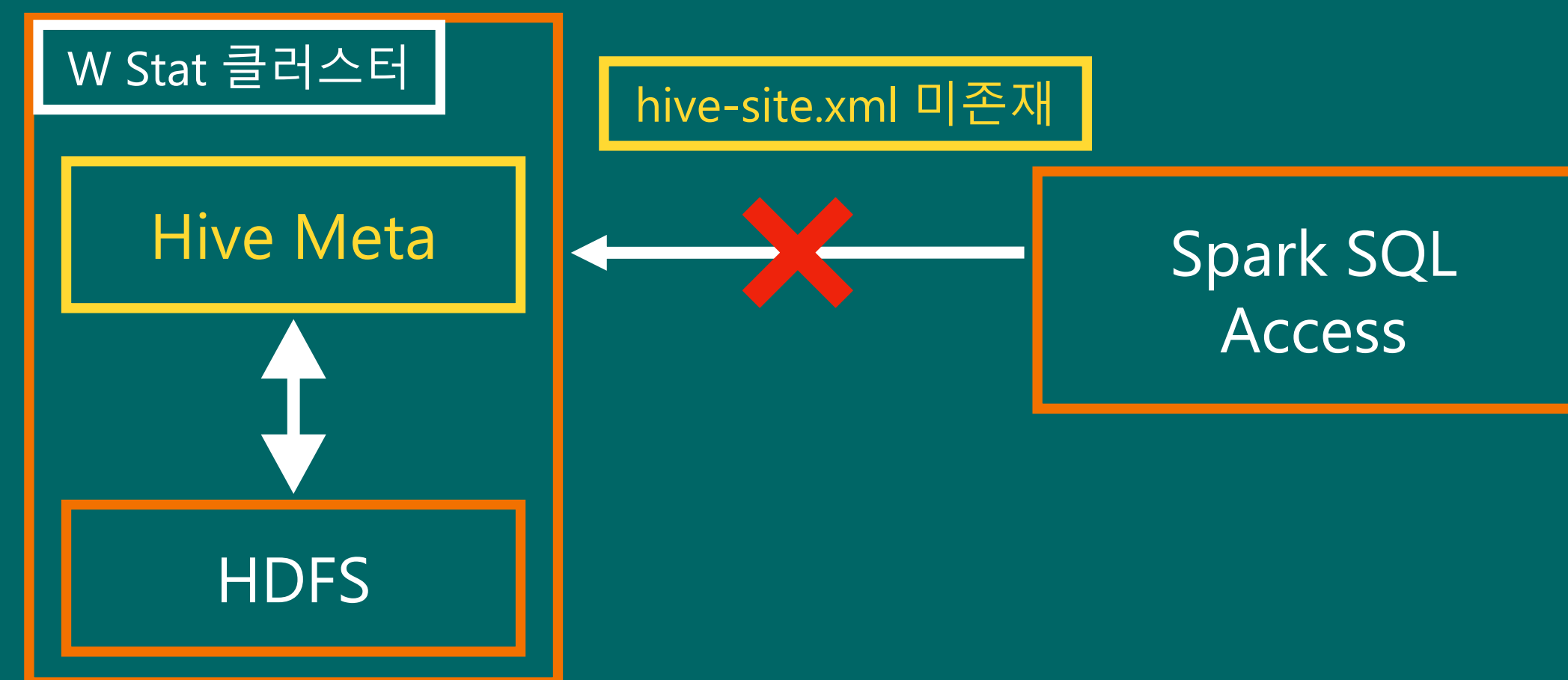
\* SQL을 이용한 Hive테이블 접근 : <https://spark.apache.org/docs/latest/sql-data-sources-hive-tables.html>

\* 현재 2.3.0 버전 기준 : <https://spark.apache.org/docs/2.3.0/sql-programming-guide.html>

# 4. 기술적 이슈

## 2-1) 테이블 생성 Job 재구성 : Data Read시 SQL 미사용 이유

- 초기에 Spark SQL을 이용해 접근 하는 방법도 찾아보았으나, 당시에는 **hive-site.xml** 설정파일을 찾지 못했음
- 또한, 클러스터는 공용으로 쓰기 때문에 개인이 **임의로 상세 값 설정을** 진행할 수는 없었음
- 추후, **hive-site.xml** 설정 파일이 뒤늦게 발견 되었고, **설정 파일을 이용해** 진행이 가능 했음



\* SQL을 이용한 Hive테이블 접근 : <https://spark.apache.org/docs/latest/sql-data-sources-hive-tables.html>

\* 현재 2.3.0 버전 기준 : <https://spark.apache.org/docs/2.3.0/sql-programming-guide.html>

# 4. 기술적 이슈

## 2-2) 테이블 생성 Job 재구성 : 이유

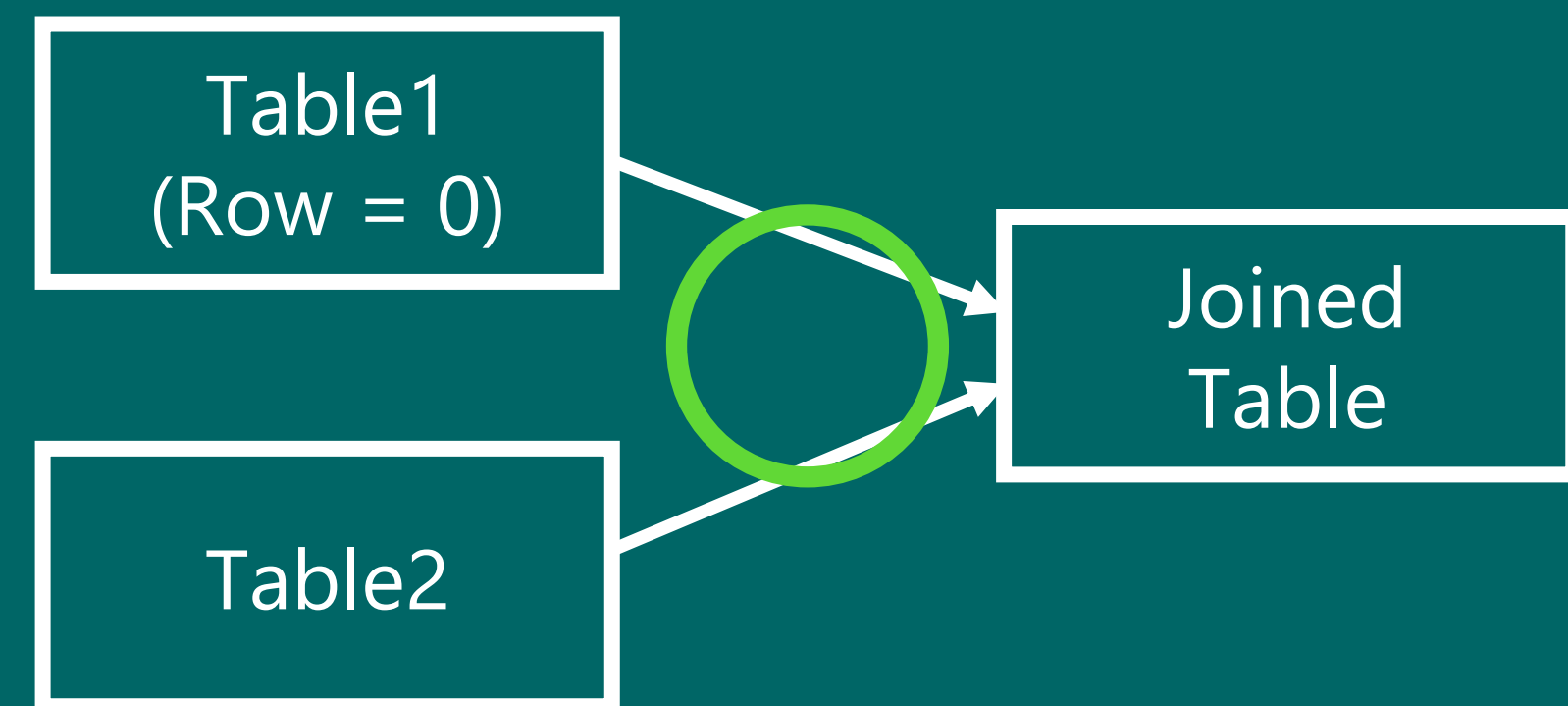
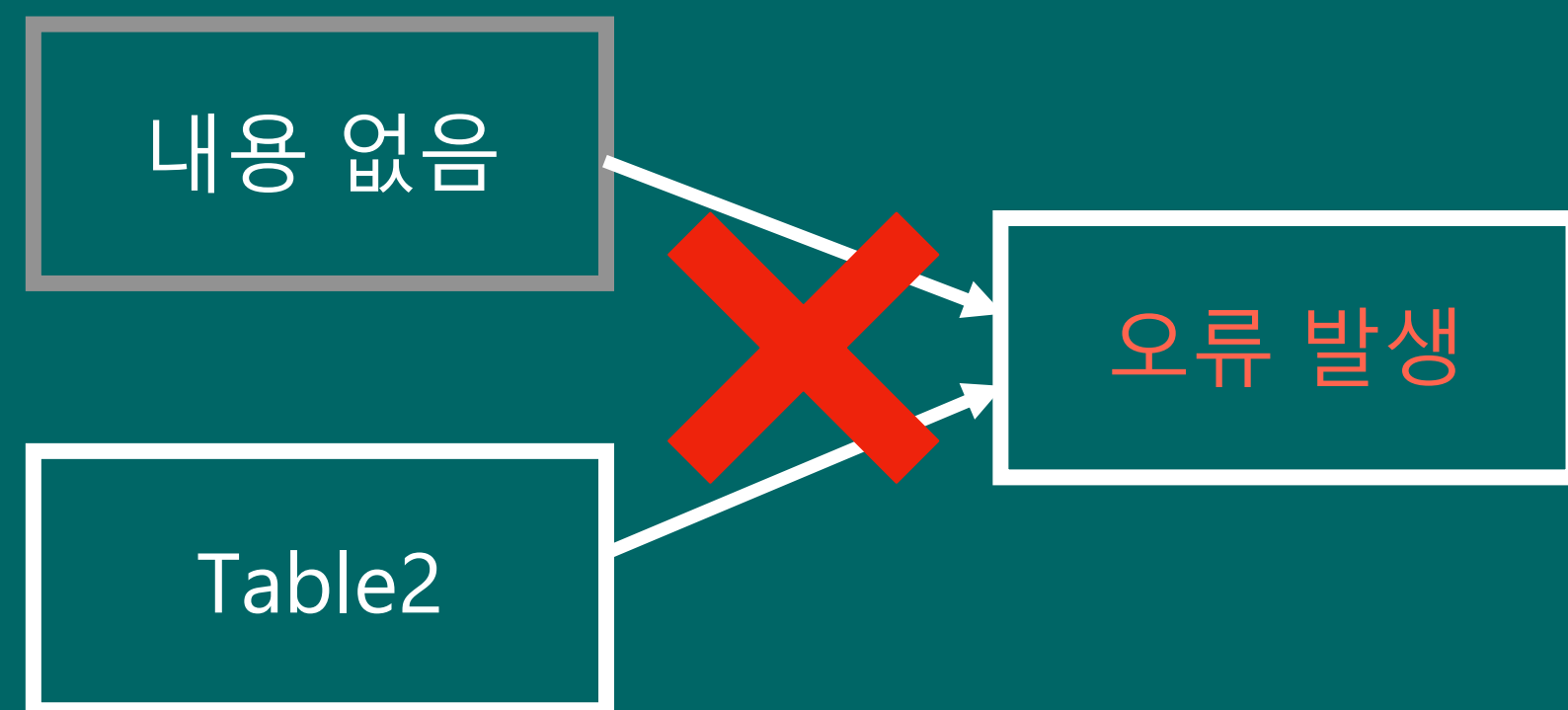
- 재구성 이유1 ) 과제 목표에 맞추기 위함
  - Spark SQL을 이용해 로직 구성하는것이 과제 목표중 하나



# 4. 기술적 이슈

## 2-2) 테이블 생성 Job 재구성 : 이유

- 재구성 이유2 ) spark.read.parquet( ) 비어있는 파일 read 에러 이슈 해결
  - 빈 파일을 읽을 시 *"Unable to infer schema for Parquet. It must be specified manually."* 이슈가 발생
  - 특히 Join의 경우에는 row가 0일지라도, 스키마는 존재하는 Empty DataFrame 형태로 연산에 필요



# 4. 기술적 이슈

## 2-2) 테이블 생성 Job 재구성 : 이유

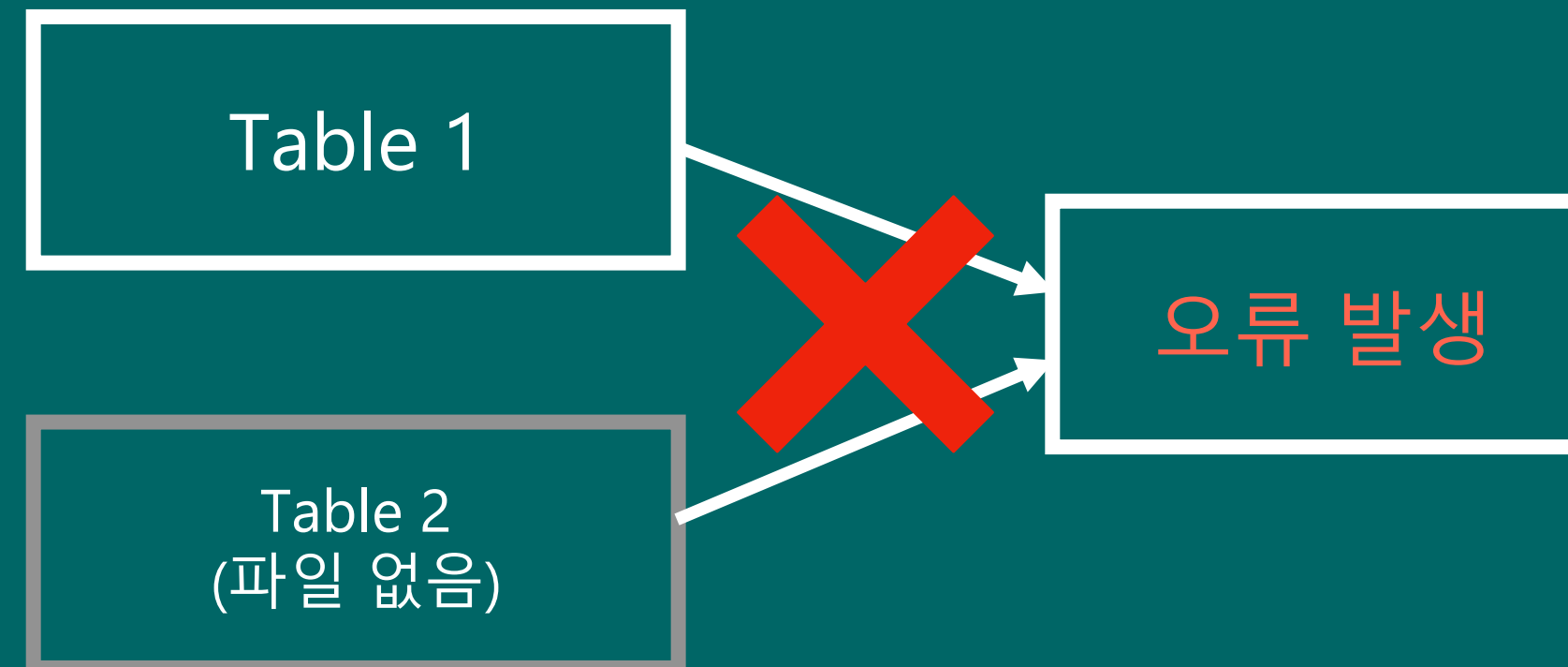
- 재구성 이유2 ) spark.read.parquet( ) 비어있는 파일 read 에러 이슈 해결 (Before)

Top Screenshot:

이름	크기
11425995a8ec5cec-e07ae88700000005_1053476090_data.0.parq	2.0 KB
11425995a8ec5cec-e07ae88700000004_2089036932_data.0.parq	1.9 KB
11425995a8ec5cec-e07ae88700000003_1168509732_data.0.parq	1.9 KB

Bottom Screenshot:

이름	크기
----	----



```
2021-08-24 17:47:37 ERROR ApplicationMaster:91 - User class threw exception: org.apache.spark.sql.AnalysisException
org.apache.spark.sql.AnalysisException: Unable to infer schema for Parquet. It must be specified manually.;
```

# 4. 기술적 이슈

## 2-2) 테이블 생성 Job 재구성 : 이유

- 재구성 이유2 ) spark.read.parquet( ) 비어있는 파일 read 에러 이슈 해결 (After)



country	platform	language	user_cnt	new_user_cnt
--	APP_IPHONE	ENGLISH	1	0
--	WEB_MOBILE	ENGLISH	1	0
--	APP_IPHONE	TRADITIONAL_CHINESE	1	0
--	WEB_PC	FRENCH	1	0
--	APP_ANDROID	ENGLISH	5	0
--	APP_ANDROID	GERMAN	1	0

# 4. 기술적 이슈

## 3) Impala 명령 전달법

\* 조치 : scalikeJDBC를 이용해 연결된 DB에 하단의 SQL쿼리를 날릴 수 있도록 한다. (Impala 테이블 인식을 위한 명령어)

```
ALTER TABLE ${DB}.${Table} ADD IF NOT EXISTS PARTITION ("파티션컬럼이름"="${targetDate}");  
REFRESH ${DB}.${Table};
```

\* scalikeJDBC 개략적 흐름 :

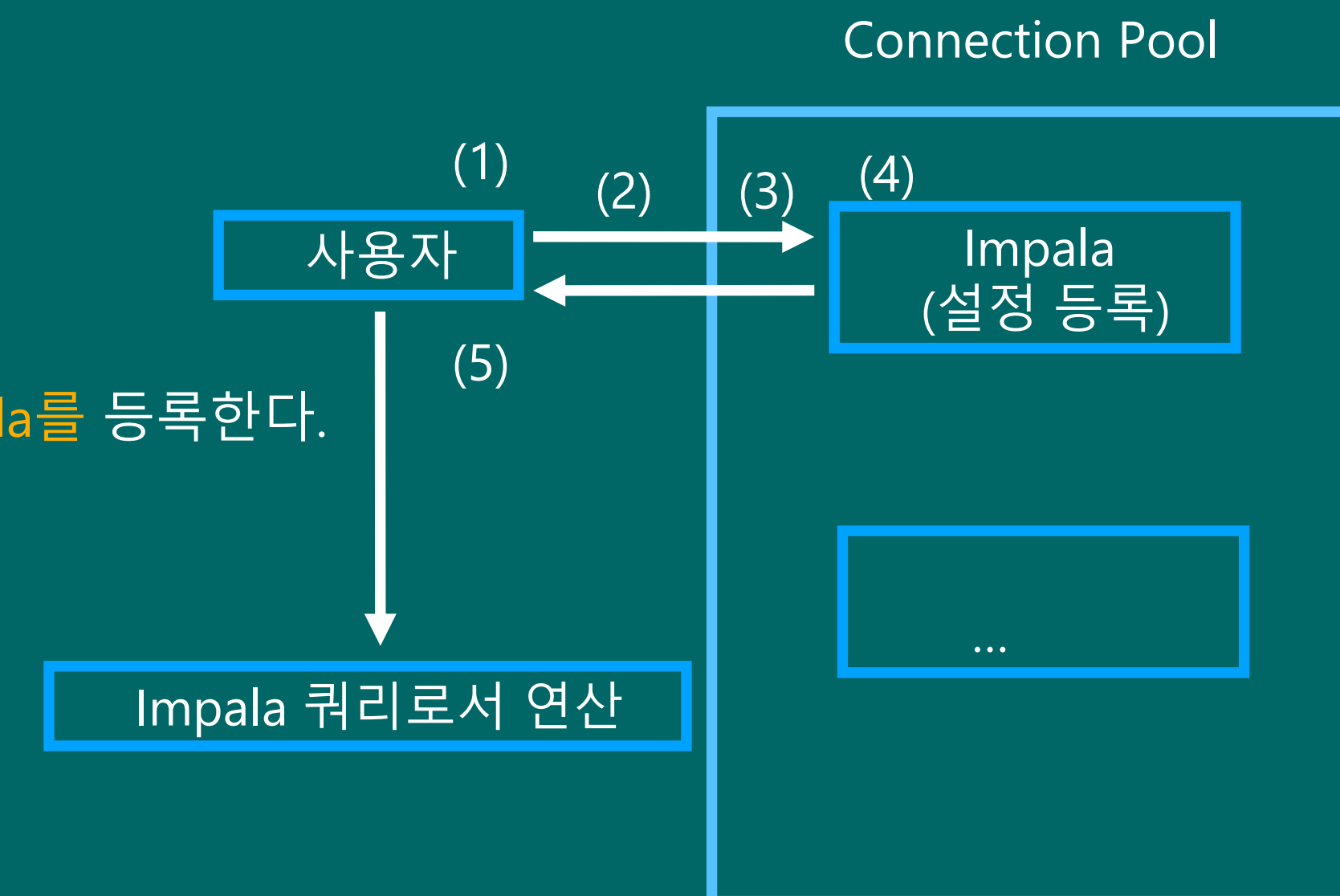
(1) 연결할 jdbc 드라이버 이름과 url을 설정한다.

(2) DB 연결에 있어 필요한 user, password를 입력한다.

(3) \*Connection Pool에 있어 필요한 설정을 셋팅한다. (INIT\_SIZE, MAX\_SIZE, TIMEOUT 등)

(4) scalike에 내장되어있는 ConnectionPool.add( ) 를 통해 1), 2), 3) 들을 적용시켜 Connection Pool에 Impala를 등록한다.

(5) 그 후, 사용자가 원하는 SQL 구문을 연결한 DB에 보낸다.



\* Connection Pool : 사용자가 원하는 DB 접근에 있어 필요한 설정들을 일정 공간(Pool)에 기록 하여 원할때마다 바로 쓸 수 있도록 한다.

\* 참고 : <http://scalikejdbc.org/>

# 5. 배운 점 및 느낀 점

## 1-1) 배운 점

- 실제 Spark Job 자체를 구성해보고 다루어 볼 수 있었다는 점
  - Spark Job은 Spark-submit 구문을 통해 연산장소에 제출되며, 실제로는 **jar파일** 형태를 갖추
  - Spark Job 로직 구성에 있어 **Scala 언어**를 다뤄볼 수 있는 기회가 되었음
  - Spark-submit 제출 구문 옵션을 Spark 공식 Document를 통해 참고하며 진행할 수 있었음
  - **jar파일** 구성을 위해 개발 환경은 IntelliJ를 사용했고, **maven**을 이용해 필요한 **설정**을 관리해 볼 수 있었음

참고 : <https://spark.apache.org/docs/2.3.0/submitting-applications.html>

# 5. 배운 점 및 느낀 점

## 1-2) 배운 점

- 유지보수성을 위해 더욱 신경써야한다는 점
  - 원하는 대로 구현도 중요하지만, 훗날의 코드 재구성 및 확장을 위해서는 코드의 형태도 중요
  - 바로 직관적으로 알 수 있는 **변수 및 메소드 네이밍이 중요**하며, 개발 비용을 최소화 시킬 수도 있음

Ex) DataCheckJob => **StatDailyCheckJob** : StatDaily 테이블 검증 기능을 진행하는 오브젝트를 의미

Ex) MainStatJob => **StatDailyMakeJob** : StatDaily 테이블 생성 기능을 진행하는 오브젝트를 의미

# 5. 배운 점 및 아쉬운 점

## 2-1) 아쉬운 점

- 현재 DailyStatJob에는 알맞으나 추후 다른 Job에 대한 확장성이 떨어짐
    - 현재, DailyStatJob은 YYYYMMDD 패턴만 만족한다면 바로 작동함
    - 그러나, 추후 hour나 week와 같이 다양한 시간 패턴이 들어있는 Job도 존재하므로 이에 대한 확장성 필요
- => 시간이 좀 더 주어졌다면, 기존의 시간 패턴에서 다양한 케이스가 들어올 수 있도록 날짜 변수를 동적으로 받거나 key-value 형태의 자료집합으로 한 데 묶어서 관리하는 방법을 생각 했을 것임

# 5. 배운 점 및 아쉬운 점

## 2-2) 아쉬운 점

- 현재의 검증 Job이 지금보다 더욱 대규모 데이터를 다룬다면 리소스 측면에서 효율이 낮아짐

=> 만약, 추가적으로 시간이 더 있었다면 기존의 검증 과정 중, 상세내용 검증 방식을 재검토 할 예정

1) Spark 에서는 `minus all` 이란 연산이 존재하는데, 중복을 포함한 테이블간의 중복교집합 요소를 모두 제외하는 형식

두 테이블 간의 연산 결과 차이를 비교하여 동등 여부를 결정하는 방법도 고려해볼만한 요소

( 2.3.0 에서는 아직 지원하지 않았기 때문에 테이블 간의 row 빈도수 테이블을 교집합하여 공통 요소를 가져오는 방식을 선택 )

2) DataFrame은 데이터 내용을 메모리에 저장하고 있는데

지금보다 대용량의 데이터가 들어올 경우에 대한 대비책을 중심으로 리서치를 진행했을 예정