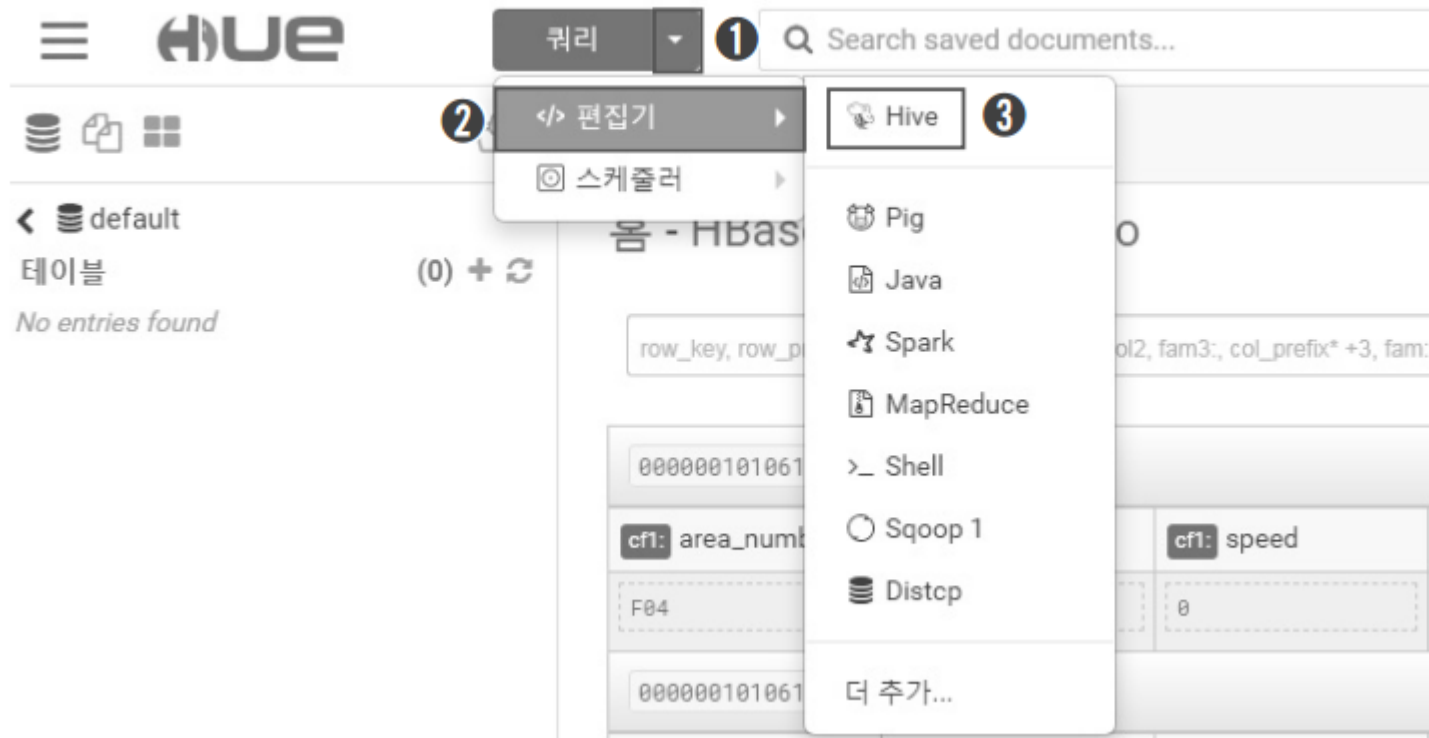


6.5 탐색 파일럿 실행 3단계-데이터 탐색&처리 2

하이프 이용한 External 데이터 탐색(2/2)

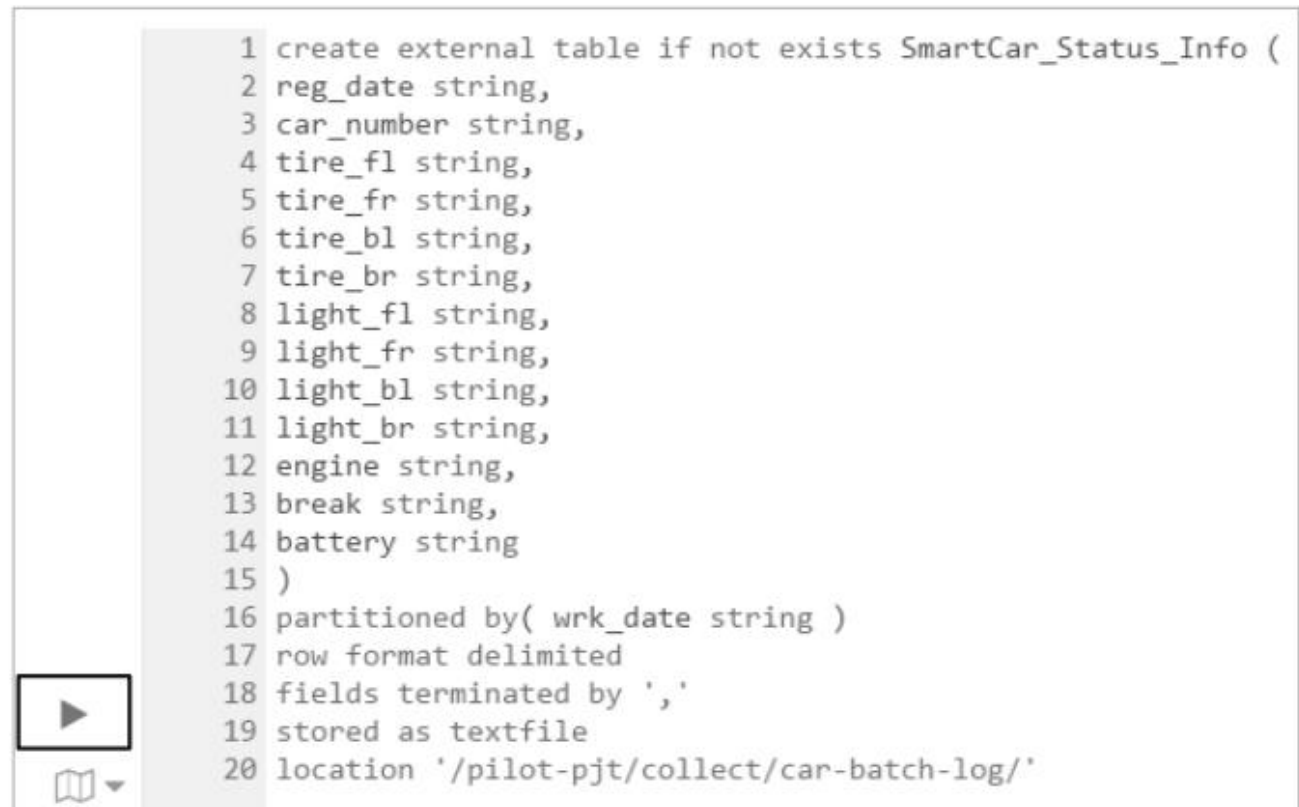
01. 상단 중앙의 쿼리 콤보박스를 내려서 [편집기] → [Hive]를 선택해 하이브 에디터로 이동한다.



6.5 탐색 파일럿 실행 3단계-데이터 탐색&처리 2

하이프 이용한 External 데이터 탐색(1/5)

02. 하이브 쿼리를 이용해 스마트카 상태 정보를 탐색하기 위한 하이브의 External 테이블을 생성하고 실행해 보자. 쿼리 에디트 창에 그림 6.44와 똑같이 입력한 후 [실행] 버튼을 누른다. 참고로 해당 QL은 C://예제소스/bigdata-master/CH06/HiveQL/그림-6.44.hql 파일로 제공된다. 해당 파일을 하이브 에디터로 드래그 앤드 드롭하면 편리하게 이용할 수 있다.



```

1 create external table if not exists SmartCar_Status_Info (
2   reg_date string,
3   car_number string,
4   tire_fl string,
5   tire_fr string,
6   tire_bl string,
7   tire_br string,
8   light_fl string,
9   light_fr string,
10  light_bl string,
11  light_br string,
12  engine string,
13  break string,
14  battery string
15 )
16 partitioned by( wrk_date string )
17 row format delimited
18 fields terminated by ','
19 stored as textfile
20 location '/pilot-pjt/collect/car-batch-log/'

```

그림 6.44 스마트카 상태 정보 테이블을 External로 생성

6.5 탐색 파일럿 실행 3단계-데이터 탐색&처리 2

하이프 이용한 External 데이터 탐색(2/5)

03. “스마트카 상태 정보”를 위한 하이브 테이블을 생성했으면 이제 작업일자를 기준으로 파티션 정보를 생성해 보자. 필자의 파일럿 환경에서 스마트카 상태 정보 파일을 수집한 날짜는 “2020년 03월 11일”이고, 수집한 데이터를 후 처리한 작업 일자도 “2020년 03월 11일”이다. 아래와 같이 Alter Table 명령을 실행해 작업 일자 기준으로 파티션 정보를 추가한다. 참고로 파티션 일자는 독자의 파일럿 환경에서 작업했던 날짜로 입력해야 한다.

```
1 ALTER TABLE SmartCar_Status_Info ADD PARTITION(wrk_date='20200311');
```

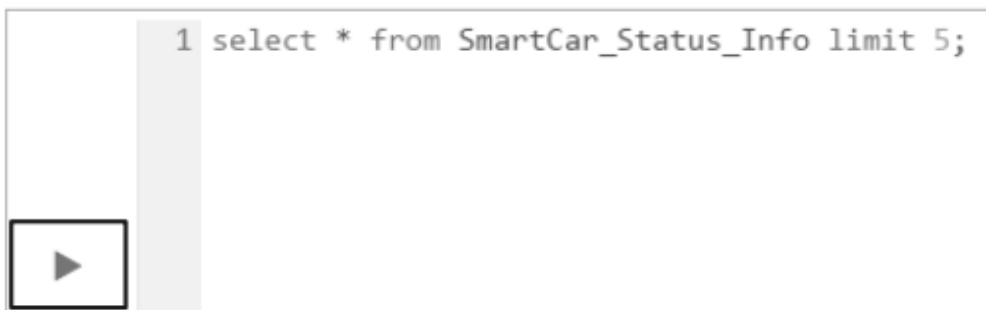


그림 6.45 스마트카 상태 정보 테이블에 파티션 정보 추가

6.5 탐색 파일럿 실행 3단계-데이터 탐색&처리 2




하이프 이용한 External 데이터 탐색(3/5)

04. 하이브의 External 테이블로 만들어진 SmartCar_Status_Info 테이블에서 단순 SELECT 쿼리를 실행해 보자. 쿼리 절의 끝에 “limit 5”로 조회 개수를 제한했다. “limit” 절을 이용해 부하를 최소화하고 빠르게 데이터를 조회해 볼 수 있다. (참고로 Ctrl + Enter 단축키를 이용하면 편집기 상에서 커서가 위치한 곳의 쿼리가 실행된다.)



```
1 select * from SmartCar_Status_Info limit 5;
```

그림 6.46 스마트카 상태 정보 테이블 조회

Query History Q  저장된 쿼리 Q  결과 Q 				
	smartcar_status_info.reg_date	smartcar_status_info.car_number	smartcar_status_info.tire_fl	smartcar_status_info.tire_fr
1	20160101000000	F0001	91	76
2	20160101000004	F0001	81	97
3	20160101000008	F0001	87	100
4	20160101000012	F0001	100	84
5	20160101000016	F0001	88	100

6.5 탐색 파일럿 실행 3단계-데이터 탐색&처리 2

하이프 이용한 External 데이터 탐색(4/5)

05. 이번에는 조건절을 포함하는 하이브 쿼리를 실행한다. 스마트카의 배터리 상태가 “60” 이하였던 차량들을 찾아보고, 배터리 평균값도 구해본다. 그림 6.48의 쿼리를 실행할 때는 높은 지연 시간이 발생할 것이다. 필자의 파일럿 환경에서는 216만 건이 적재돼 있는데 해당 쿼리를 수행할 때 총 2분 4초가 걸렸다.

```

1 select car_number, avg(battery) as battery_avg
2 from SmartCar_Status_Info
3 where battery < 60
4 group by car_number;

```

그림 6.48 스마트카 상태 정보 테이블에 대한 조건 조회

Query History

저장된 쿼리

결과

	car_number	battery_avg
1	A0061	54.413698630136984
2	A0096	54.601503759398497
3	B0057	54.441958041958038
4	B0066	54.360273972602741
5	B0073	54.385714285714286
6	C0074	54.557103064066851
7	D0008	54.608870967741936

그림 6.49 스마트카 상태 정보 테이블에 대한 조건 조회 결과

6.5 탐색 파일럿 실행 3단계-데이터 탐색&처리 2

하이프 이용한 External 데이터 탐색(5/5)

06. 실행된 하이브 쿼리 모니터링은 출력되는 로그창을 통해서도 확인할 수 있지만 그림 6.50처럼 [Job Browser] 메뉴를 통해 맵리듀스로 변환되어 실행된 Job 정보도 확인할 수 있다.

Job 1 ↺ admin

Job Browser
Job
Workflow
일정
Bundle
SLAs

user:admin
☐ 성공
☐ 실행 중
☐ 실패함 지난
7
일
↺
중지

실행 중

이름	사용자	유형	상태	진행률	그룹	시작됨	소요 시간	ID
select car_number, avg(battery)...car_number (Stage-1)	admin	MAPREDUCE	ACCEPTED	0%	default	2020년 3월 21일 오전 1시 18분	23.82s	application_1584711074926_0002

Completed

이름	사용자	유형	상태	진행률	그룹	시작됨	소요 시간	ID
select car_number, avg(battery)...car_number (Stage-1)	admin	MAPREDUCE	SUCCEEDED	100%	default	2020년 3월 21일 오전 1시 14분	2m, 32s	application_1584711074926_0001

2 Job

그림 6.50 하이브 실행 모니터링 – Job Browser

Tip _ 하이브의 특징

하이브는 하둡에 적재돼 있는 파일의 메타정보(파일의 위치, 이름, 포맷 등)를 테이블의 스키마 정보와 함께 메타스토어에 등록하고, 하이브 쿼리를 수행할 때 메타스토어의 정보를 참조해 마치 RDBMS에서 데이터를 조회 및 탐색하는 것 같은 기능을 제공한다. 하지만 하이브 쿼리는 RDBMS 쿼리와 외형만 유사할 뿐 내부 메커니즘은 전혀 다르다. 다음은 하이브 QL의 5가지 주요 특징이다.

1. **하이브 쿼리는 맵리듀스로 변환되어 실행:** 일부 하이브 쿼리를 제외한 대부분의 하이브 쿼리는 맵리듀스로 변환되어 하둡의 잡에서 실행 및 관리된다. 하나의 하이브 쿼리는 복잡도에 따라 여러 개의 잡이 순차적으로 만들어지고, 다시 잡 안에서는 데이터의 크기와 조건절에 따라 여러 개의 맵과 리듀스 작업이 생성되어 실행된다.
2. **대화형 온라인 쿼리 사용에 부적합:** 하이브 쿼리는 맵리듀스로 변환되어 실행되기까지는 최소 10~30초 이상의 준비 시간이 필요하며, 처리량이 높은 대규모 배치 작업에 최적화돼 있다. 극단적인 예를 들면 RDBMS에서 0.5초면 수행되는 가벼운 쿼리를 하이브에서 실행하면 50여 초 이상 걸릴 수 있다. 하지만 RDBMS에서 50여분 수행되던 무거운 쿼리를 하이브에서 실행하면 단 50여 초만에 수행이 완료될 수 있다.
3. **데이터의 부분적인 수정(Update/Delete) 불가:** 데이터의 부분적인 수정 불가는 HDFS의 특징이다. HDFS를 기반으로 작동하는 하이브는 그 특징을 그대로 계승했다고 할 수 있다. 그로 인해 하이브 테이블에서는 부분 수정 및 삭제 처리를 할 수 없고 전체 데이터를 덮어쓰기해야 한다. 이러한 문제로 하이브로 처리할 데이터를 적재할 때는 특정 파티션 단위로 적재해 필요 시 해당 파티션만 덮어쓰는 방식의 데이터 설계를 한다. 참고로 하이브 0.14부터는 INSERT, UPDATE, DELETE, MERGE 문을 ORC Stored 형식에서 부분적으로 지원한다.

4. 대규모 병렬분산 처리가 불가능한 경우: 하이브는 처리량이 높은 대규모 병렬분산 처리에 최적화돼 있지만 일부 요건에 따라 대규모 분산 처리가 어려울 수 있다. 예를 들어, RDBMS에서 주로 사용되는 전체 정렬을 단순 하이브 QL로 실행하면 하나의 리듀스만 실행되어 급격한 성능 저하가 발생한다. 또한 대규모 조인이나, 그룹핑이 발생하면 분산된 노드들끼리 대량의 데이터를 주고받으면서 네트워크 레이턴시가 높아져 응답 속도가 크게 떨어진다. 대용량 하이브 QL을 수행할 때는 데이터의 분산과 로컬리티를 반드시 고려한 쿼리 플랜을 세워야 한다.
5. 트랜잭션 관리 기능이 없어 롤백 처리 불가: 하나의 하이브 쿼리는 여러 개의 잡과 맵리듀스 프로그램으로 실행되며, 로컬 디스크에 중간 파일들을 만들어낸다. 이때 특정 맵리듀스 작업 하나가 실패하면 이미 성공한 잡들이 롤백 처리되지 않는다. 단지 실패한 잡을 재실행하기 위해 노력할 뿐이다. 이러한 기본 원칙으로 인해 하이브에서는 트랜잭션 관리 기능이 존재하지 않는다. 참고로 하이브 0.14부터는 ACID를 부분적으로 지원하고 있으나 제약사항이 많아 자주 사용되지는 않는다.

하이브가 SQL과 유사한 인터페이스를 제공하기 시작하면서 하둡 에코시스템들이 급성장하게 됐고, 빅데이터 벤더들은 하이브의 단점을 극복한 소프트웨어들을 만들기 시작했다. 관련 제품으로 임팔라, 테즈(Tez), 스파크 SQL, 드릴(Drill) 등이 있다. 이 제품들의 공통적인 특징은 인메모리 또는 DAG(Directed Acyclic Graph) 기술로 대용량의 배치 처리에 대해서도 빠른 응답속도를 보장한다는 것이다. 하지만 최근 동향을 보면 해당 제품들이 하이브를 완전히 대체하지는 못하고, 하이브리드 형태로 하이브와 공존하고 있는 모습이다. 빅데이터의 특성상 장시간에 걸친 배치성 잡에는 하이브를 이용하고, 빠른 의사결정이 필요한 잡에는 임팔라, 테즈 등을 선택적으로 사용한다.

6.5 탐색 파일럿 실행 3단계-데이터 탐색&처리 2

 하이브 이용한 External 데이터 탐색

실습