[AWS] 🏴‍☠️ Athena 최적화 하기 (데이터 압축 / 데이터 파티셔닝)

2023년 6월 15일 목요일 오후 2:21

Athena 성능 향상 전략

Athena(아테나)는 서버리스 서비스로서, 실행한 쿼리에 대한 비용을 지불하면 된다.

요금은 실행한 쿼리가 스캔한 데이터의 용량만큼 비용이 청구된다.

하지만 실행한 쿼리의 데이터 용량이 어마무시하면 아무리 온디맨드 형식일 지라도 요금이 많이 들게 된다.

따라서 적절히 데이터 압축과 파티셔닝을 하여, 스캔하는 데이터의 양을 제한해 비용을 절감할 수 있는 전략으로 나아가야 한다.

Info

데이터 파티셔닝

용량이 큰 테이블이나 인덱스를 관리하기 쉬운 파티션(partition)이라는 작은 단위로 분할하는 것

Tip

아테나를 사용하기 위한 테이블 생성(CREATE)이나 파티셔닝을 위한 테이블 수정(ALTER)과 같은 DDL문과 실패한 쿼리문에 대한 비용은 청구되지 않는다.

데이터 압축

Athena의 비용은 스캔한 데이터의 크기를 기준으로 발생한다.

따라서 아테나를 이용할때는, S3에 있는 로그 데이터를 압축해 놓고 저장하는 것을 권장한다.

파일 용량이 작으면 네트워크 비용 감소, 쿼리속도 향상, 스캔되는 데이터의 크기 감소되어 비용절감 등 얻는 것이 많기 때문이다.

Athena가 인식 가능한 압축 포맷은 링크Visit Website 및 아래 사진을 참고한다.

- snappy : Parquet 데이터 스토리지 형식의 파일에 대한 기본압축형식
- zlip : ORC 데이터스토리지 형식의 파일에 대한 기본압축형식
- gzip : 데이터파일에 .gz확장프로그램 사용, 디폴트 값

	Avro	Ion	ORC	Parquet	텍스트 파일
bzip2	읽기 지원 전용. 쓰기는 지원되지 않음.	예	아니요	아니요	예
DEFLATE	예	아니요	아니요	아니요	아니요
GZIP	아니요	예	아니요	예	예
LZ4	아니요	Hadoop 호환 읽기 지원. 쓰기 지원 없음.	예(원시/프레이밍되지 않음)	아니요	Hadoop 호환 읽기 지원. 쓰기 지원 없음.
LZO	아니요	Hadoop 호환 읽기 지원. 쓰기 지원 없음.	아니요	예	Hadoop 호환 읽기 지원. 쓰기 지원 없음.
SNAPPY	원시/프레이밍되지 않는 읽기 지원. 쓰기는 지원되지 않음.	예(Hadoop 호환 프레이밍)	예(원시/프레이밍되지 않음)	예(원시/프레이밍되지 않음)	예(Hadoop 호환 프레이밍)
ZLIB	아니요	아니요	예	아니요	아니요
ZSTD	아니요	예	예	예	예
NONE	예	예	예	예	예

Athena 압축 지원 - Amazon Athena

docs.aws.amazon.com

Athena 압축 지원 Athena는 여러 압축 형식을 사용하는 테이블에서 데이터를 읽는 것을 비롯하여, 데이터 읽기 및 쓰기를 위한 다양한 압축 형식을 지원합니다. 예를 들어 Athena는 일부 Parquet 파일이

데이터 압축 설정하기

압축을 지원한다고 했으니 실전 적용해보자.

압축 형식은 TBLPROPERTIES 속성으로 지정할 수 있다.

CREATE문이나 ALTER 문에 TBLPROPERTIES 속성으로 orc.compress나 parquet.compression 값을 주고 압축 형식을 지정해주면 된다.

SQL

```
CREATE EXTERNAL TABLE IF NOT EXISTS test.myjson2 (
  `time` STRING,
  `user_id` STRING,
  `board_name` STRING,
  `action` STRING
)
LOCATION 's3://athena-log-test-11/jsonTest'
TBLPROPERTIES("PARQUET.COMPRESS"="GZIP"); -- GZIP으로 설정
```

Copy

SQL

```
ALTER TABLE test.myjson SET TBLPROPERTIES("PARQUET.COMPRESS"="GZIP");
```

Copy

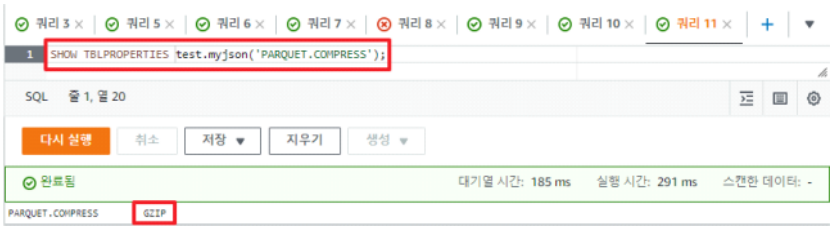
ALTER TABLE SET TBLPROPERTIES - Amazon Athena

docs.aws.amazon.com

Thanks for letting us know this page needs work. We're sorry we let you down. If you've got a moment, please tell us how we can make the documentation better.

테이블을 수정해주고 TBLPROPERTIES 속성을 조회해보면 GZIP이 적용된 걸 확인할 수 있다.

SQL
```
SHOW TBLPROPERTIES test.myjson('parquet.compression');
```
Copy

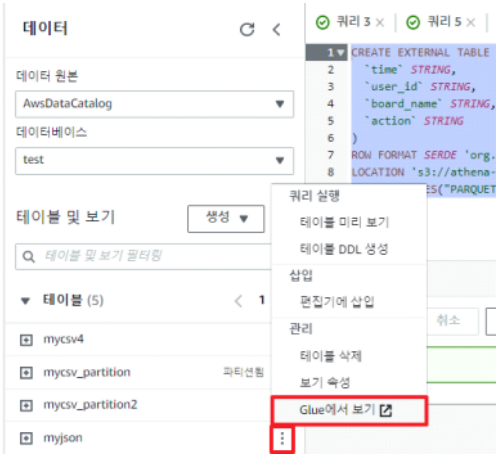


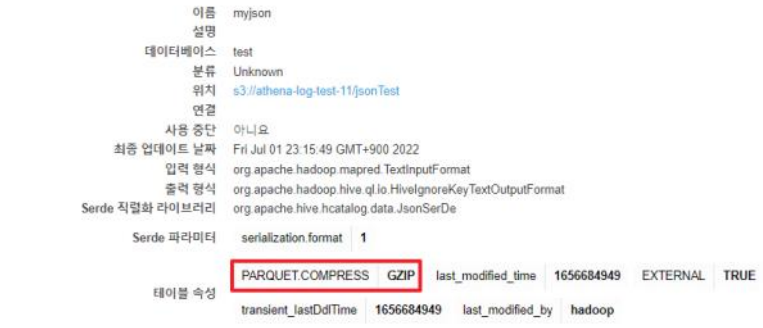
혹은 AWS GLUE에서도 확인이 가능하다.

Info

AWS Glue 서비스

AWS Glue는 완전 관리형 ETL(추출, 변환 및 로드) 서비스로, 간단하게 여러 데이터 스토어 및 스트림 간에 원하는 데이터를 분류, 정리, 보강, 이동한다. EC2가 인스턴스(컴퓨팅)을 모아놓은 것이고, 람다가 함수들을 모아놓은 것이라면, Glue는 데이터들을 모아서 관리한다고 보면 된다.





데이터 파티셔닝

데이터 파티셔닝(Data Partitioning)이란, 데이터를 분할 저장하여 조건에 따라 일부 데이터만 검색할수 있도록 하는 기법이다

그래서 S3에 쌓인 많은 로그를 조회할때, 파티셔닝을 하지 않으면 아테나 쿼리는 모든 데이터를 스캔하게 되지만, 파티셔닝을 통해 쿼리당 스캔하는 데이터의 양을 줄여 성능을 향상시키고, 비용을 절감할 수 있다.

파티셔닝을 적절히 해주면 테이블 조회 쿼리 실행 시간이 3~5분에서 5초로 절감되는 케이스가 있을 정도이다.

그래서 아테나를 사용할 때 파티셔닝은 필수로 하는 것이 좋다.

보통 파티셔닝(partitioning)은 날짜기준으로 쿼리가 스캔하는 범위를 제한하는 편이다.

예를들어 2022년 7월 1일의 로그 데이터를 수집할 때, 10년치 로그가 쌓여있는 한개의 폴더에서 찾을 경우 10년치 로그를 모두 스캔해야하지만, 로그가 각 년,월,일로 분할되어 있는 폴더에서 찾을 경우엔 bucket/2022/7/1 폴더에 있는 로그만 스캔하면 된다.

파티셔닝은 자동으로 맵핑하는 방법과 수동으로 맵핑하는 방법이 있다.

만약 이미 S3 버킷에 로그가 쌓여져 있으면 둘 중에 자신의 S3 구조에 맞는 방법을 사용하면 되고, 아직 로그가 쌓여지 않은 상태라면 자동 맵핑 구조에 맞게 데이터를 적재하면 편리하다.

자동 맵핑

데이터파티셔닝을 위한 자동매핑 구조일 경우, 공식문서에선 버킷에서 아래와 같은 구조로 파일이 위치해야 된다고 말한다.

SHELL
```
S3://your-bucket/pathToTable/<PARTITION_COLUMN_NAME>=<VALUE>/<PARTITION_COLUMN_NAME>=<VALUE>/
```
Copy

예를들면 아래와 같이 버킷/키 구조로 되어 있어야 한다.

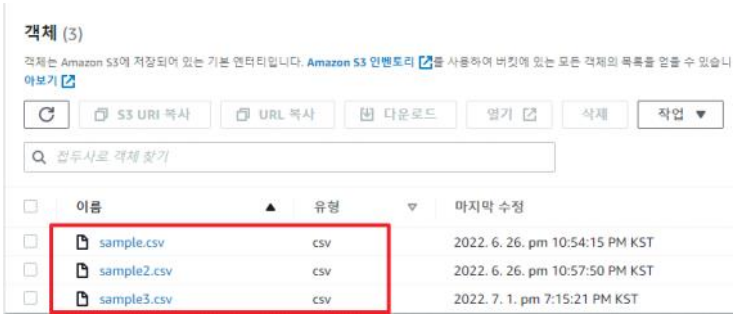
SHELL
```
S3://my-bucket/log-data/year=2019/month=3/day=7
```
Copy

아래와 같은 구조로 되어 있다면 year, month, day가 파티션컬럼이 된다.

쿼리가 가능하도록 구조만 똑같이 하면 되고 컬럼명과 형식은 달라도 된다.

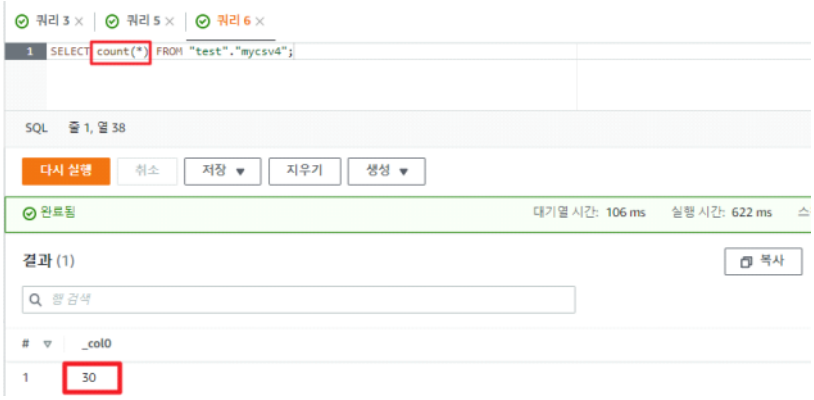
이게 당최 무슨말인지 도저히 모르겠다면, 바로 실전에 돌입하도록 하자.

S3에 다음과 같이 csv 파일이 3개가 있고, 이 버킷 경로를 Athena(아테나)로 맵핑 시켜 테이블로 만들어 놓았다고 가정하자. 각 csv 파일당 10개의 데이터가 있다.



버킷에 현재 저장되어 있는 csv 파일이 3개인데 count 쿼리를 실행하면 30개가 나오는데, Data scanned과 3개 csv 파일 사이즈의 합이 동일함을 확인할 수 있다.

만일 sample3.csv에만 있는 데이터를 조회할 필요가 있을때, 적절히 SQL문을 쿼리하면 원하는 결과는 얻을지언정 아테나는 버킷 경로를 기준으로 매핑하기 때문에 쿼리 실행시마다 모든 csv 파일을 검색하게 된다.



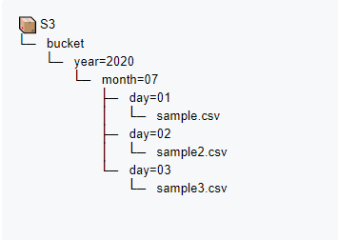
즉, csv 파일이 나뉘어져있음에 불구하고 아테나는 이를 하나의 파일로 쳐서 발생되는 문제인 것이다.

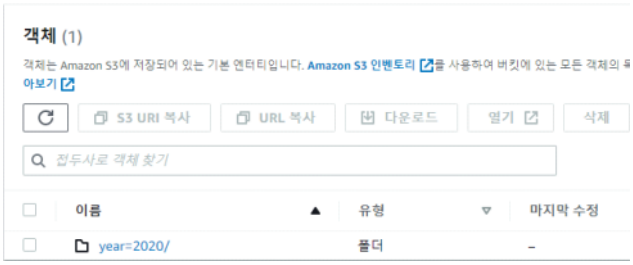
이는 곧 비용이므로 조건에 따라 일부 csv 파일만 검색하도록 partitioning 을 해보자.

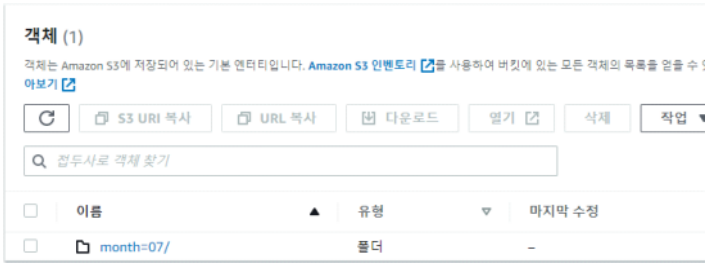
이를 위해서는 아래 작업을 수행해야 한다.

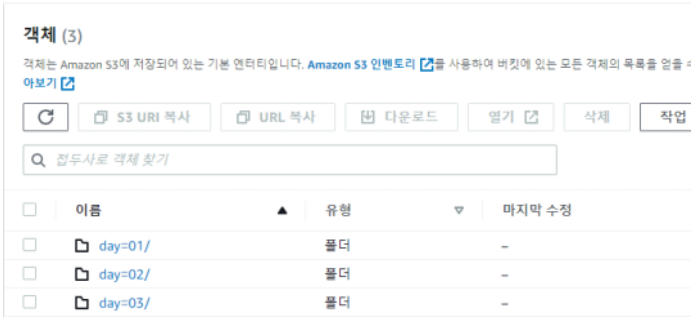
1. csv 파일을 디렉토리별로 그룹핑 해서 저장
2. 디렉토리명을 partition 조건으로 활용

다음과 같이 버킷 디렉토리를 년도별, 월별, 일별 로 나누고 각 일별 폴더에 csv 파일을 위치시킨다.









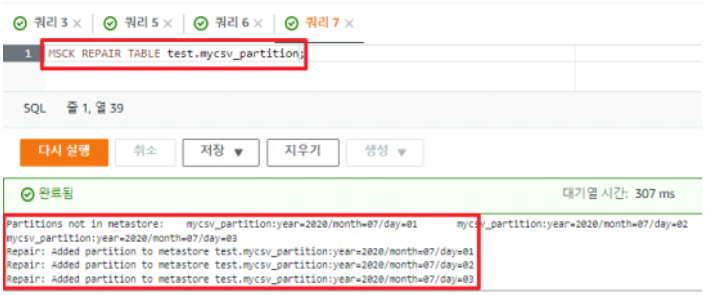
이렇게 버킷 폴더 별로 키(파일)이 위치한 상태에서, 테이블 생성 시 파티션 컬럼을 설정해 줄 수 있다.

```SQL
CREATE EXTERNAL TABLE IF NOT EXISTS test.mycsv_partition (
  `time` STRING,
  `user_id` STRING,
  `board_name` STRING,
  `action` STRING
)
PARTITIONED BY (year string, month string, day string) -- 파티션 추가
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
ESCAPED BY '\\'
LINES TERMINATED BY '\n'
LOCATION 's3://athena-log-test-11/csvTest/'
TBLPROPERTIES ('skip.header.line.count'='1');
```

Copy

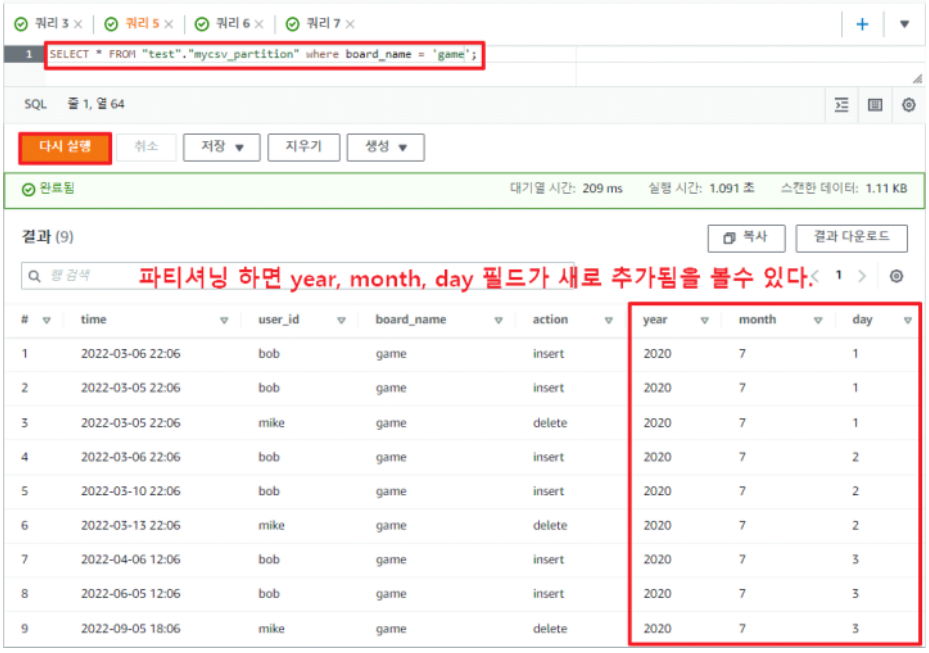
테이블 생성했으면, 자동으로 파티션을 지정할 수 있도록 아래의 명령 수행한다.

SQL
```
MSCK REPAIR TABLE test.mycsv_partition;
```
Copy



테이블 필드를 전체 조회해보면, 우측에 새로운 필드가 추가됐음을 확인 할 수 있다.

바로 **<u>이 추가된 파티션 필드를 이용해서 조건 검색</u>**을 하면 되는 것이다.

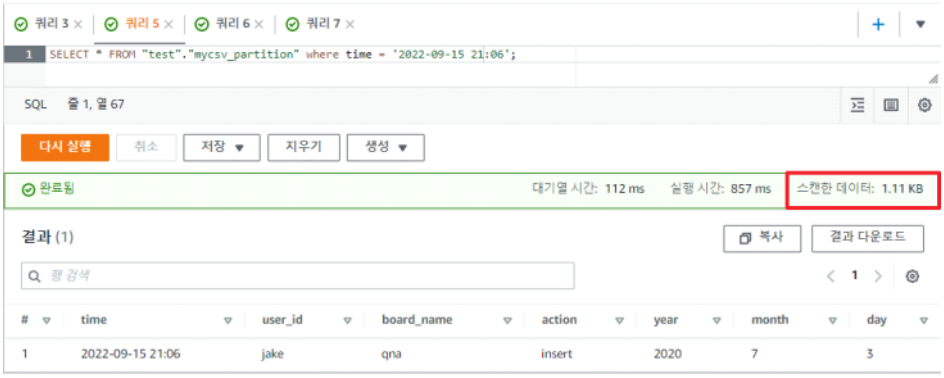


만일 time 필드가 2022-09-15 21:06 인 값을 검색하고 싶다고 하자.

일반적으로 다음과 같이 쿼리를 할것이고 원하는 결과를 얻을 수 있을 것이다.

하지만 여기서 중요한건 스캔한 데이터 부분이다. 스캔한 용량을 보니 1.11KB 이다.

SQL
```
SELECT * FROM "test"."mycsv_partition" where time='2022-09-15 21:06';
```
Copy

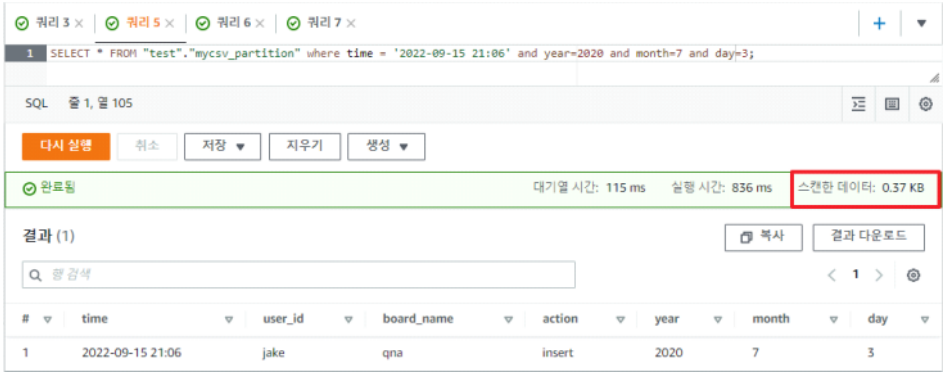


이번엔 위에서 파티셔닝하여 추가된 필드를 조건 검색하여 쿼리 해보자.

SQL

```
SELECT * FROM "test"."mycsv_partition" where time='2022-09-15 21:06' and year=2020 and month=7 and day=3;
```

Copy

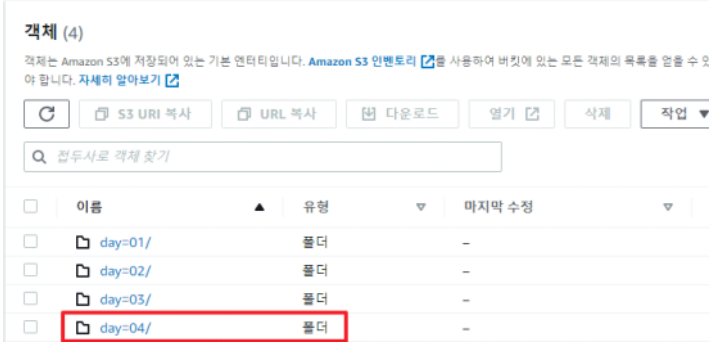


쿼리 결과는 똑같이 나왔지만, 스캔한 데이터는 0.37KB로 줄어든걸 확인할 수 있다.

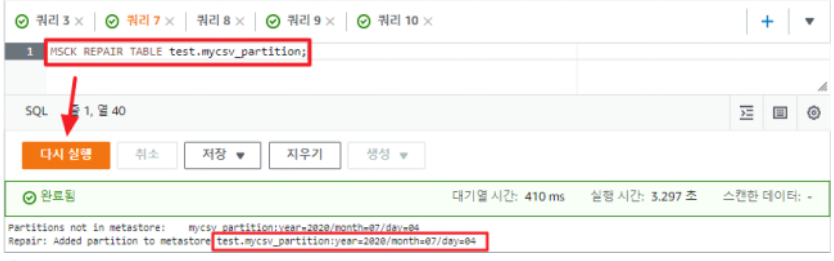
즉, 파티셔닝 필드 조건을 추가하지 않으면 버킷에 있는 전체 csv를 뒤져서 질의하지만, 파티셔닝 필드 조건을 추가하게 된다면라 Athena가 알아서 자동 맵핑된 버킷 폴더 경로로 찾아들어가 그 폴더 안에있는 csv만을 뒤지기 때문에 스캔한 데이터 용량이 줄어들게 되는 것이다.

만일 로그를 버킷에 추가하고 Athena로 관리하게 만들고 싶으면 어떻게 할까?

다음과 같이 버킷에 day=04/ 디렉토리를 만들고 안에 sample4.csv 파일을 넣어보자.



그리고 자동 파티션 쿼리를 실행하면 새로 추가된 폴더의 데이터가 테이블에 추가되게 된다.



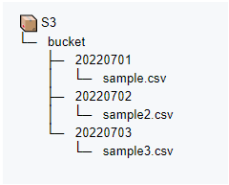
Tip

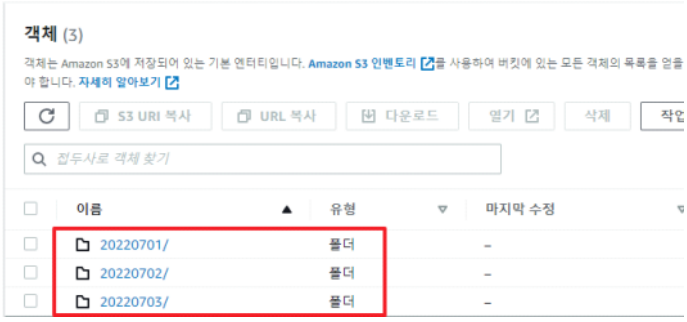
하지만 몇 천 개 이상의 파티션이 있는 경우, MSCK REPAIR TABLE 방법은 DDL 쿼리에 시간 초과 문제가 발생할 수 있어 모범 사례가 아니다.
이때는 수동 맵핑의 alter문으로 파티션 테이블을 업데이트 해줘야 한다.
이에 대해선 바로 다음 수동 맵핑 섹션에서 다룬다.

수동 맵핑

년도, 월별, 일자 별로 세세히 폴더를 하나하나 나누는 자동 맵핑 형식이 마음에 들지 않는다면, 내가 원하는 대로 디렉토리를 만들고 수동 매핑을 해주면 된다.

다음과 같이 아예 날짜 자체를 폴더명으로 하여 심플하게 로그를 관리하고 싶다고 가정하자.





디렉토리 구조에 맞게 파티션 테이블을 생성한다.

SQL

```
CREATE EXTERNAL TABLE IF NOT EXISTS test.mycsv_partition2 (
  `time` STRING,
  `user_id` STRING,
  `board_name` STRING,
  `action` STRING
)
PARTITIONED BY (date string) -- 파티셔닝
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
ESCAPED BY '\\'
LINES TERMINATED BY '\n'
LOCATION 's3://athena-log-test-11/csvTest/'
TBLPROPERTIES ('skip.header.line.count'='1');
```

Copy

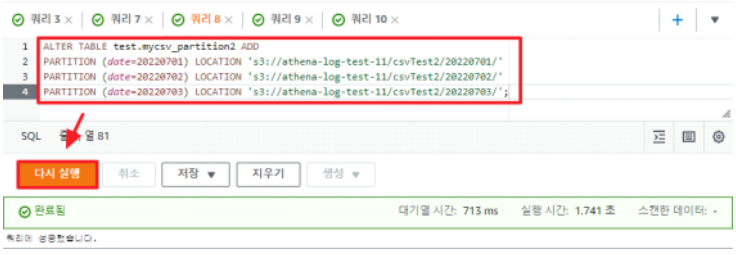
그리고 파티션을 등록해주어야 하는데, 자동 맵핑과는 달리 일일히 파티션을 지정하여 쿼리를 날려주어야 한다.

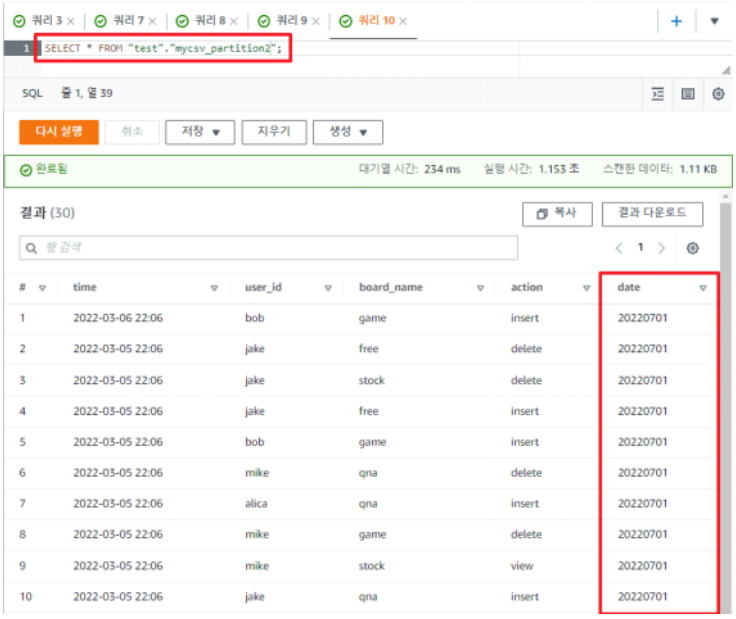
Tip
자동 매핑 방식도 alter문으로 파티션 업데이트가 가능하다.

SQL

```
ALTER TABLE test.mycsv_partition2 ADD
PARTITION (date=20220701) LOCATION 's3://athena-log-test-11/csvTest2/20220701/'
PARTITION (date=20220702) LOCATION 's3://athena-log-test-11/csvTest2/20220702/'
PARTITION (date=20220703) LOCATION 's3://athena-log-test-11/csvTest2/20220703/';
```

Copy





그러면 S3에 로그가 추가될때 마다 일일히 alter문을 날려야 되냐고 의문이 생길수 있을텐데, 그럴수 밖에 없다.

일일별로 하루에 한번 파티션 로드를 해줘야 한다.

그래서 개발자들은 보통 람다나 서버단(aws cli)에서 이러한 과정을 자동으로 처리되게 하는 편이다.

SHELL
```
$ aws athena start-query-execution --query-string "MSCK REPAIR TABLE some_database.some_table" --result-configuration "OutputLocation=s3://SOMEPLACE"
```
Copy

PYTHON
```
# 람다
import boto3

def lambda_handler(event, context):
    bucket_name = 'some_bucket'

client = boto3.client('athena')

config = {
        'OutputLocation': 's3://' + bucket_name + '/',
        'EncryptionConfiguration': {'EncryptionOption': 'SSE_S3'}

}

# Query Execution Parameters
    sql = 'MSCK REPAIR TABLE some_database.some_table'
    context = {'Database': 'some_database'}

client.start_query_execution(QueryString = sql,
                             QueryExecutionContext = context,
                             ResultConfiguration = config)
```
Copy

파티션 삭제

만일 파티션을 잘못 지정하여 삭제하고 싶을 경우 다음 쿼리를 날리면 된다.

SQL
```
ALTER TABLE orders
DROP PARTITION (dt = '2014-05-14', country = 'IN'), PARTITION (dt = '2014-05-15', country = 'IN');
```
Copy

Partition Projection으로 파티셔닝 자동화

Amazon Athena에 고도로 분할된 테이블의 쿼리 처리 속도를 빠르게 해주고 파티션 관리를 자동화하는 데 쓸 수 있는 새로운 기능인 Partition Projection 기능이 최근에 추가되었다.

위에서 자동 맵핑이든 수동 맵핑이든 버킷에 새로운 로그 파일과 폴더가 추가될경우 `MSCK REPAIR TABLE` 혹은 ~~alter문~~ 으로 파티셔닝을 업데이트해줘서 추가된 파일을 인식시켜야 했다.

그리고 이를 자동화 하기위해 보통 쉘 스크립트로 처리하거나 람다로 처리한다고 위에서 한번 언급한 바 있다.

하지만 Partition Projection을 사용하면 파티션을 형성하는 데 공통적으로 사용된 패턴(예를 들어 YYYY/MM/DD)과 같은 구성 정보를 지정할 수 있다.

즉, Partition Projection을 지정하면 새로운 폴더와 파일이 추가되도 따로 alter문으로 파티셔닝을 일일히 업데이트 해주지 않아도 알아서 자동으로 감지하여 Athena 테이블에 추가한다는 말이다.