

2

Lab 2

4. Spectrum을 활용한 현대화

이 실습에서는 데이터를 적재하거나 이동하지 않고 Amazon S3 데이터 레이크에 저장된 엑사바이트 규모의 데이터를 Amazon Redshift로 페타바이트 단위로 쿼리하는 방법을 보여줍니다. 또한 직접 연결 스토리지(direct attached storage)와 S3 데이터레이크에서 데이터를 통합하여 활용하여 단일 정보 소스로 생성하는 방법도 확인합니다. 마지막으로 오래된 데이터를 S3로 이동(Aging off)하고 Amazon Redshift 직접 연결 스토리지에서 가장 최근 데이터만 유지 관리하는 전략을 살펴보겠습니다.

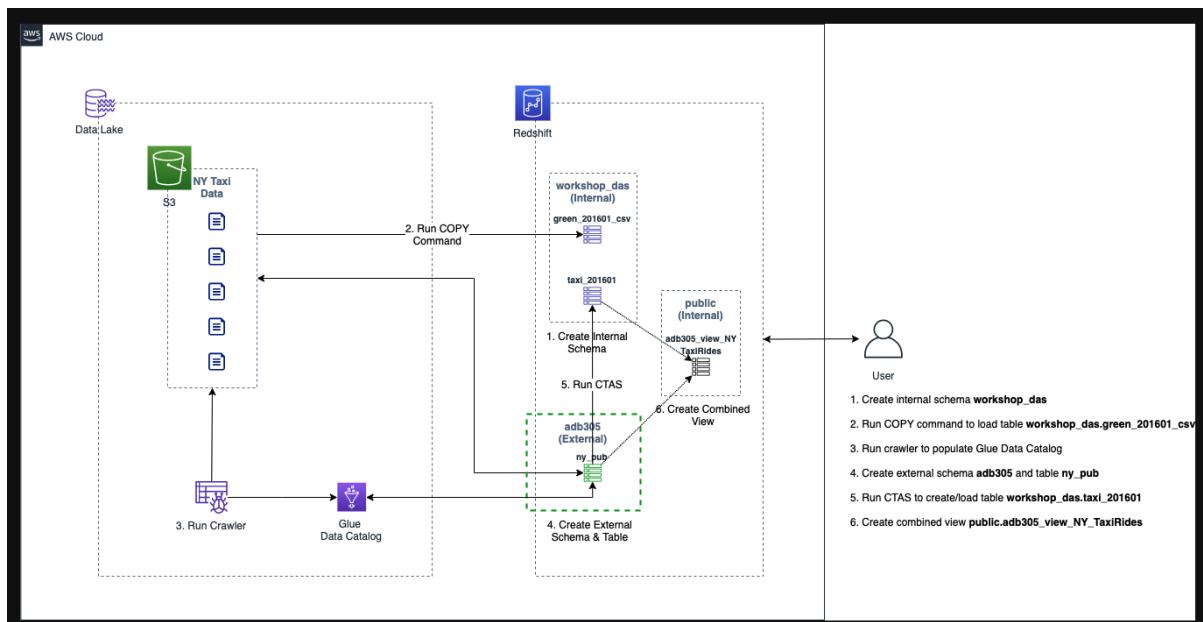
시작에 앞서

Before You Begin

- 이번 Lab을 수행하기 위해서 Redshift 클러스터와 클라이언트 도구 설정이 필요합니다. [Lab 1 - 클러스터 생성 및 연결](#), [Lab2 - 데이터 로딩](#) 이 먼저 선행되어야 합니다.
- 이번 Lab을 위해서는 [US-WEST-2\(Oregon\)](#) 지역에 클러스터를 구성해야 합니다.

Architecture

다음은 이번 실습과 관련된 아키텍처 및 각 단계를 설명한 개요입니다.



DAS(Direct Attached Storage) 활용

What Happened in 2016

- Green사의 2016년 1월 데이터를 Redshift direct-attached storage(DAS)에 COPY명령어로 로딩합니다
- 2016년 1월 눈보라가 택시 이용에 미친 영향에 대한 증거 데이터를 수집합니다.
- CSV 데이터는 Amazon S3의 월별 데이터입니다. 아래 스크립션을 참고하세요.

https://s3.console.aws.amazon.com/s3/buckets/us-west-2.serverless-analytics/NYC-Pub/green/?region=us-west-2&tab=overview&prefixSearch=green_tripdata_2016

The screenshot shows the AWS S3 console interface. In the top navigation bar, it says "Amazon S3 > us-west-2.serverless-analytics > NYC-Pub/ > green/". On the right, there is a "S3 URI 복사" (Copy S3 URI) button. Below the navigation, there are tabs for "객체" (Objects) and "속성" (Properties). The "객체" tab is selected, showing 41 objects. A search bar at the top of the list area contains "green_tripdata_2016". The list table has columns for "이름" (Name), "유형" (Type), "마지막 수정" (Last modified), "크기" (Size), and "스토리지 클래스" (Storage class). All 12 files are CSV type, modified on May 19, 2017, at various times between 12:41:40 AM KST and 12:45:50 AM KST, and are 220.3MB to 214.2MB in size, all in Standard storage class.

- 다음은 샘플 데이터의 일부입니다.

green_tripdata_2016-01																				
VendorID	Lpep_pickup_datetime	Lpep_dropoff_datetime	Store_and_fwd_flag	RateCodeID	Pickup_longitude	Pickup_latitude	Dropoff_longitude	Dropoff_latitude	Passenger_count	Trip_distance	Fare_amount	Extra	MTA_tax	Tip_amount	Tolls_amount	Ehail_fee	improvement_surcharge	Total_amount	Payment_type	Trip_type
2	2016-01-01 00:29:24	2016-01-01 00:29:36	N	1	-73.92884227294200	40.80010567738500	-73.92427825977300	40.89824382382200	1	1.48	8	0.5	0.5	1.86	0	0	0.3	11.16	1	1
2	2016-01-01 00:19:31	2016-01-01 00:30:16	N	1	-73.95874740229700	40.823104981500	-73.917747774900	40.7017951153400	1	3.68	15.5	0.5	0.5	0	0	0	0.3	16.8	2	1
2	2016-01-01 00:20:53	2016-01-01 00:30:48	N	1	-73.95874740229700	40.823104981500	-74.01307059482000	40.8497252978900	1	3.70	16.5	0.5	0.5	4.45	0	0	0.3	22.25	1	1
2	2016-01-01 00:25:12	2016-01-01 00:30:32	N	1	-73.986510583105	40.86907850246100	-74.00068480835200	40.8993330303000	1	3.01	13.5	0.5	0.5	0	0	0	0.3	14.8	2	1
2	2016-01-01 00:24:01	2016-01-01 00:30:22	N	1	-73.987203092000	40.86907850246100	-73.94071964492200	40.86921481852000	1	2.55	12	0.5	0.5	0	0	0	0.3	13.3	2	1
2	2016-01-01 00:22:05	2016-01-01 00:30:35	N	1	-73.98114379882100	40.82454514624200	-73.957744450900	40.74211126034700	1	1.37	7	0.5	0.5	0	0	0	0.3	8.3	2	1
2	2016-01-01 00:34:34	2016-01-01 00:30:21	N	1	-73.98687109863300	40.81619174826200	-73.98610203177300	40.74588030089800	1	0.57	5	0.5	0.5	0	0	0	0.3	6.3	2	1
2	2016-01-01 00:31:03	2016-01-01 00:30:36	N	1	-73.95238188186900	40.825516492000	-73.94190054540000	40.79120388512000	1	1.01	7	0.5	0.5	0	0	0	0.3	8.3	2	1
2	2016-01-01 00:24:40	2016-01-01 00:30:36	N	1	-73.99404613105400	40.820100052100	-73.91728787978000	40.87975464972700	1	2.46	12	0.5	0.5	2	0	0	0.3	15.3	1	1
2	2016-01-01 00:28:59	2016-01-01 00:30:23	N	1	-73.91413116450800	40.766413879300	-73.917491333000	40.73956555712900	1	1.61	9	0.5	0.5	1.6	0	0	0.3	11.9	1	1
2	2016-01-01 00:32:25	2016-01-01 00:30:48	N	1	-73.91117848872000	40.781837620700	-73.921081370700	40.76128373201000	1	0.72	6	0.5	0.5	1.82	0	0	0.3	9.12	1	1
2	2016-01-01 00:37:51	2016-01-01 00:30:31	N	1	-73.95816820978500	40.71532916552700	-73.96753205684400	40.718177756410200	1	0.32	3.5	0.5	0.5	0.96	0	0	0.3	5.76	1	1
2	2016-01-01 00:21:33	2016-01-01 00:30:37	N	1	-73.94687161621100	40.807806049700	-73.9408396947700	40.84276500910500	1	3.54	14.5	0.5	0.5	0	0	0	0.3	15.8	2	1
2	2016-01-01 00:34:04	2016-01-01 00:30:49	N	1	-73.91429139159300	40.76543017846600	-73.97783312982900	40.711020517700	1	1.10	6	0.5	0.5	0	0	0	0.3	7.3	2	1
2	2016-01-01 00:26:16	2016-01-01 00:30:46	N	1	-73.94337433798100	40.82143111743100	-73.92029792191100	40.85196936379800	1	2.28	11	0.5	0.5	2	0	0	0.3	14.3	1	1
2	2016-01-01 00:35:40	2016-01-01 00:30:33	N	1	-73.99970398618800	40.821555737200	-73.96688078338000	40.84015969339300	1	0.88	4.5	0.5	0.5	1.18	0	0	0.3	6.96	1	1
2	2016-01-01 00:25:05	2016-01-01 00:30:43	N	1	-73.93784323753900	40.814445449680500	-73.95188362304700	40.811886760233900	1	1.36	10	0.5	0.5	0	0	0	0.3	11.3	2	1
2	2016-01-01 00:17:40	2016-01-01 00:30:49	N	1	-73.99364074707000	40.890516016038800	-73.94774627688500	40.69325042724600	1	3.07	15.5	0.5	0.5	3	0	0	0.3	19.8	1	1

DDL 생성

Redshift 컴퓨팅 노드에 적재할 테이블을 위한 `workshop_das` 스키마 및 `workshop_das.green_201601_csv` 을 생성합니다.(Redshift direct-attached storage(DAS) 테이블)

```
CREATE SCHEMA workshop_das;

CREATE TABLE workshop_das.green_201601_csv
(
    vendorid          VARCHAR(4),
    lpep_pickup_datetime  TIMESTAMP,
    lpep_dropoff_datetime  TIMESTAMP,
    store_and_fwd_flag  VARCHAR(1),
    ratecode          INT,
    pickup_longitude  FLOAT4,
    pickup_latitude   FLOAT4,
    dropoff_longitude FLOAT4,
    dropoff_latitude  FLOAT4,
    passenger_count   INT,
    trip_distance     FLOAT4,
    fare_amount       FLOAT4,
    extra             FLOAT4,
```

```

    mta_tax          FLOAT4,
    tip_amount      FLOAT4,
    tolls_amount    FLOAT4,
    ehail_fee       FLOAT4,
    improvement_surcharge  FLOAT4,
    total_amount    FLOAT4,
    payment_type   VARCHAR(4),
    trip_type      VARCHAR(4)
)
DISTSTYLE EVEN
SORTKEY (passenger_count,pickup_datetime);

```

COPY 명령어 생성

- Amazon S3로 부터 데이터를 복제하기 위해 COPY 명령어를 사용합니다. 해당 데이터셋은 2016년 1월 택시 이용 횟수입니다.

```

COPY workshop_das.green_201601_csv
FROM 's3://us-west-2.serverless-analytics/NYC-Pub/green/green_tripdata_2016-01.csv'
IAM_ROLE default
DATEFORMAT 'auto'
IGNOREHEADER 1
DELIMITER ','
IGNOREBLANKLINES
REGION 'us-west-2'
;

```

- 로딩된 데이터 갯수를 확인합니다.

```

select count(1) from workshop_das.green_201601_csv;
--1445285

```

폭설이 발생한 시기 찾기

해당 월에 폭설로 인해 택시 이용 횟수가 최소인 날짜를 확인해 보세요

```

SELECT TO_CHAR(pickup_datetime, 'YYYY-MM-DD'),
COUNT(*)
FROM workshop_das.green_201601_csv
GROUP BY 1
ORDER BY 2;

```

스펙트럼(Redshift Spectrum) 활용

Go Back in Time

- Redshift Spectrum을 위한 External DB를 생성하여 S3에 저장되어 있는 기록(Historical) 데이터 쿼리합니다
- 시간에 따른 추세와 다양한 정보를 보기 위해 새로운 방식으로 데이터를 룰업(rolling-up)하여 기록 데이터를 조사합니다
- Redshift Spectrum 관련 Query Monitoring Rules(QMR)을 적용해서 클러스트럴 효율적으로 활용합니다.
 - 과부하를 유발하는 쿼리를 사용하여 QMR을 설정합니다.

파티션은 년, 월, 타입(택시회사)로 구성되었습니다(스크린샷 참조)

<https://s3.console.aws.amazon.com/s3/buckets/us-west-2.serverless-analytics/canonical/NY-Pub/>

Amazon S3 > us-west-2.serverless-analytics > canonical/ > NY-Pub/

NY-Pub/

S3 URI 복사

객체 속성

객체 (19)

객체는 Amazon S3에 저장되어 있는 기본 엔터티입니다. [Amazon S3 인벤토리](#)를 사용하여 버킷에 있는 모든 객체의 목록을 얻을 수 있습니다. 다른 사용자가 객체에 액세스할 수 있게 하려면 명시적으로 권한을 부여해야 합니다.

자세히 알아보기

C S3 URI 복사 URL 복사 다운로드 열기 삭제 작업 폴더 만들기 업로드

검색어로 객체 찾기 버전 표시

선택	이름	유형	마지막 수정	크기	스토리지 클래스
<input type="checkbox"/>	year=2009/	폴더	-	-	-
<input type="checkbox"/>	year=2010/	폴더	-	-	-
<input type="checkbox"/>	year=2011/	폴더	-	-	-
<input type="checkbox"/>	year=2012/	폴더	-	-	-
<input type="checkbox"/>	year=2013/	폴더	-	-	-
<input type="checkbox"/>	year=2014/	폴더	-	-	-
<input type="checkbox"/>	year=2015/	폴더	-	-	-
<input type="checkbox"/>	year=2016/	폴더	-	-	-

<https://s3.console.aws.amazon.com/s3/buckets/us-west-2.serverless-analyticscanonical/NY-Pub/year%253D2016/>

Amazon S3 > us-west-2.serverless-analytics > canonical/ > NY-Pub/ > year=2016/

year=2016/

S3 URI 복사

객체 속성

객체 (12)

객체는 Amazon S3에 저장되어 있는 기본 엔터티입니다. [Amazon S3 인벤토리](#)를 사용하여 버킷에 있는 모든 객체의 목록을 얻을 수 있습니다. 다른 사용자가 객체에 액세스할 수 있게 하려면 명시적으로 권한을 부여해야 합니다.

자세히 알아보기

C S3 URI 복사 URL 복사 다운로드 열기 삭제 작업 폴더 만들기 업로드

검색어로 객체 찾기 버전 표시

선택	이름	유형	마지막 수정	크기	스토리지 클래스
<input type="checkbox"/>	month=1/	폴더	-	-	-
<input type="checkbox"/>	month=10/	폴더	-	-	-
<input type="checkbox"/>	month=11/	폴더	-	-	-
<input type="checkbox"/>	month=12/	폴더	-	-	-
<input type="checkbox"/>	month=2/	폴더	-	-	-
<input type="checkbox"/>	month=3/	폴더	-	-	-
<input type="checkbox"/>	month=4/	폴더	-	-	-
<input type="checkbox"/>	month=5/	폴더	-	-	-
<input type="checkbox"/>	month=6/	폴더	-	-	-
<input type="checkbox"/>	month=7/	폴더	-	-	-
<input type="checkbox"/>	month=8/	폴더	-	-	-
<input type="checkbox"/>	month=9/	폴더	-	-	-

<https://s3.console.aws.amazon.com/s3/buckets/us-west-2.serverless-analyticscanonical/NY-Pub/year%253D2016/month%253D1/>

Amazon S3 > us-west-2.serverless-analytics > canonical/ > NY-Pub/ > year=2016/ > month=1/

month=1/

S3 URI 복사

속성

객체 (5)

객체는 Amazon S3에 저장되어 있는 기본 엔터티입니다. [Amazon S3 인벤토리](#)를 사용하여 버킷에 있는 모든 객체의 목록을 얻을 수 있습니다. 다른 사용자가 객체에 액세스할 수 있게 하려면 명시적으로 권한을 부여해야 합니다.

[자세히 알아보기](#)

C S3 URI 복사 URL 복사 다운로드 열기 삭제 작업 폴더 만들기 업로드

검색 접두사로 객체 찾기 버전 표시

이름	유형	마지막 수정	크기	스토리지 클래스
type=fhv_\$.folder\$	-	2017. 5. 19. am 11:43:22 AM KST	0B	Standard
type=fhv/	폴더	-	-	-
type=green/	폴더	-	-	-
type=yellow_\$.folder\$	-	2017. 5. 19. am 11:43:49 AM KST	0B	Standard
type=yellow/	폴더	-	-	-

External 스키마 및 DB 생성

External 테이블은 공유 Glue Catalog에 저장되어 있어 AWS 에코시스템내에서 활용할 수 있습니다. 예를 들면, Athena, Redshift, Glue 등에서 사용 가능합니다.

- AWS Glue Crawler를 사용하여 <s3://us-west-2.serverless-analytics/canonical/NY-Pub/> 위치에서 parquet 포맷으로 저장된 [adb305.ny_pub](#) 테이블을 생성합니다.

1. Glue Crawler 페이지로 이동합니다.

- <https://us-west-2.console.aws.amazon.com/glue/home?region=us-west-2#catalog:tab=crawlers>

AWS Glue

크롤러 크롤러 데이터 스트어에 연결하여, 우선 순위가 지정된 분류자의 목록을 기준으로 데이터의 스키마를 결정한 다음, 데이터 키탈로그에 메타데이터 테이블을 생성합니다.

크롤러 추가 크롤러 실행 작업 태그 및 속성 필터 표시: 0-0 사용자 기본 설정

이름	일정	상태	로그	마지막 빈타임	중간 빈타임	테이블 업데이트	테이블 추가됨

아직 크롤러 없습니다. **크롤러 추가**

2. 크롤러 추가를 클릭하고, 크롤러 이름에 [NYTaxiCrawler](#) 을 입력하고 다음을 클릭합니다.

AWS Glue

크롤러 추가

크롤러 정보 추가

크롤러 이름 태그, 설명, 보안 구성, 분류자(선택 항목)

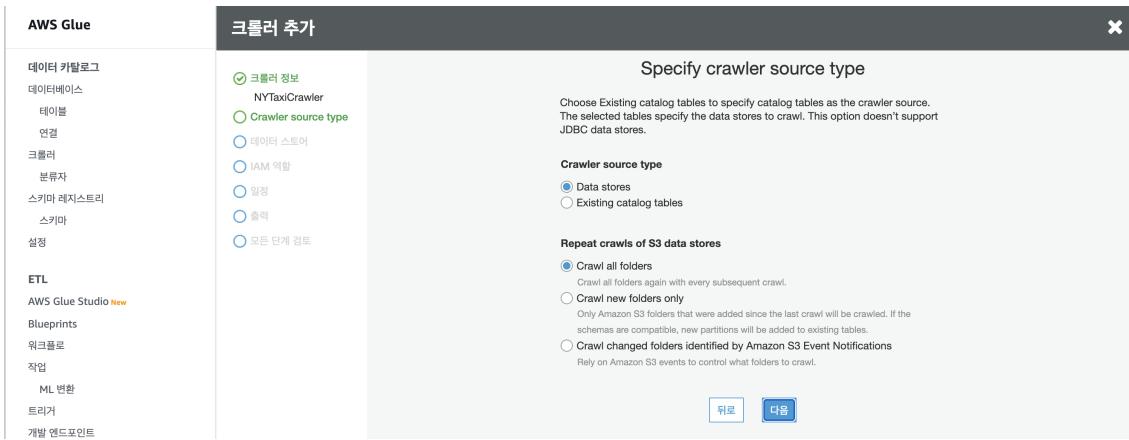
다음

데이터 카탈로그 데이터베이스 테이블 연결 크롤러 분류자 스키마 레지스트리 스키마 설정

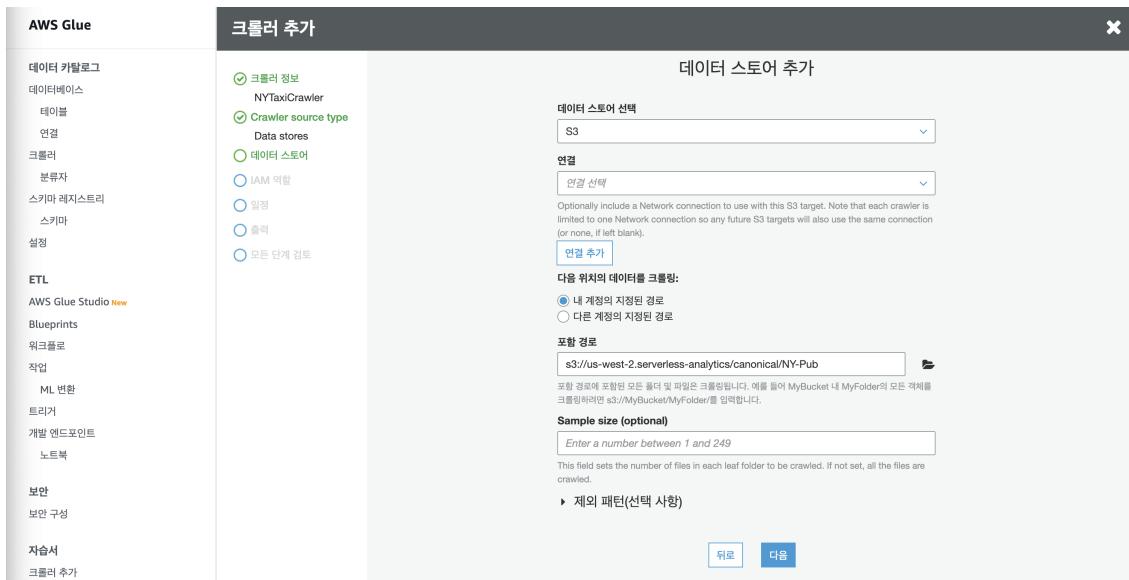
크롤러 정보

 Crawler source type
 데이터 스트어
 IAM 역할
 일정
 출처
 모든 단계 검토

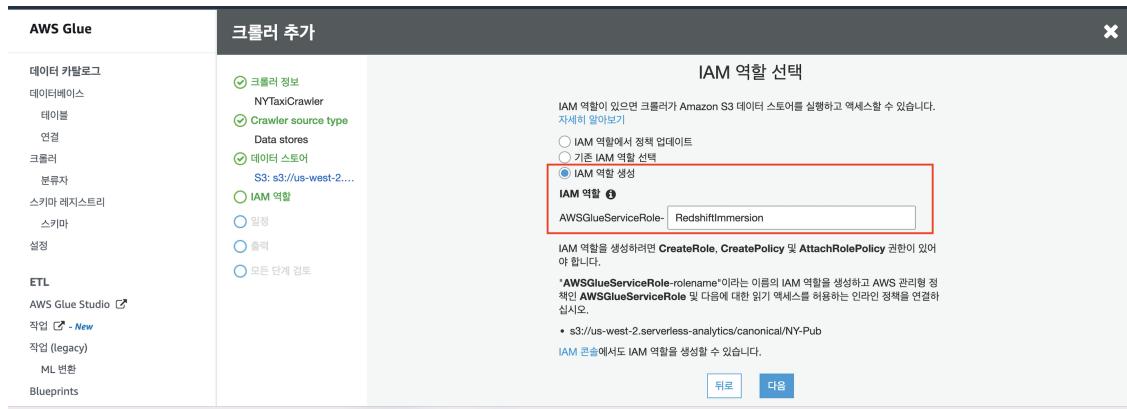
3. Source Type으로 Data stores 를 선택하고 다음을 클릭합니다.



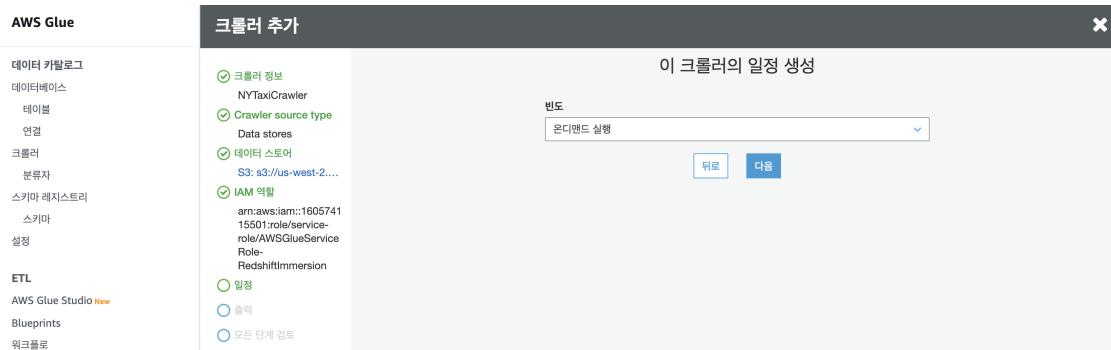
4. 데이터 저장소로 S3를 선택하고 포함경로(Include path) `s3://us-west-2.serverless-analytics/canonical/NY-Pub`의 경로를 입력합니다.



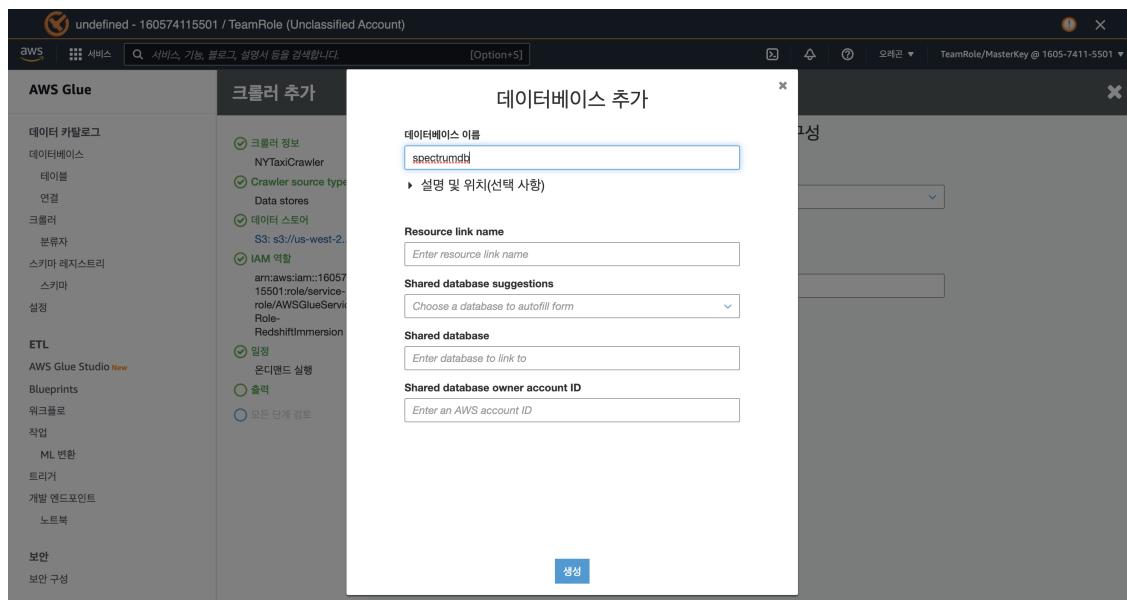
5. IAM 역할 생성(Create an IAM Role)을 선택하고, AWSGlueServiceRole- `RedshiftImmersion` 을 입력합니다.

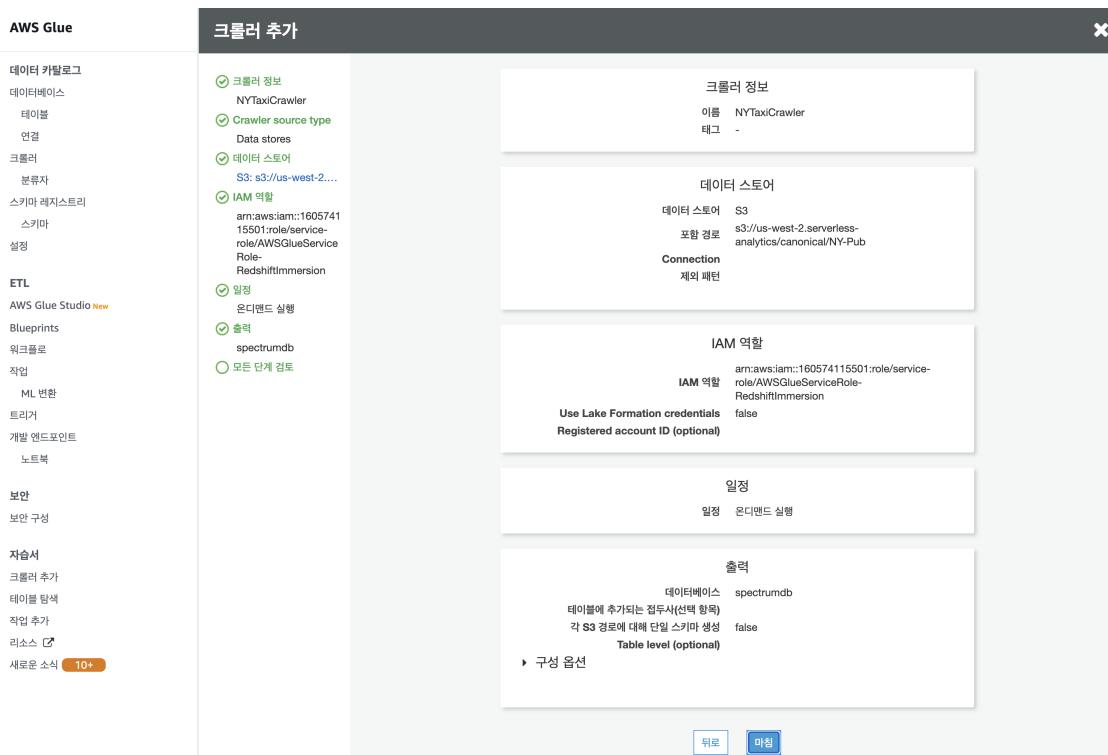
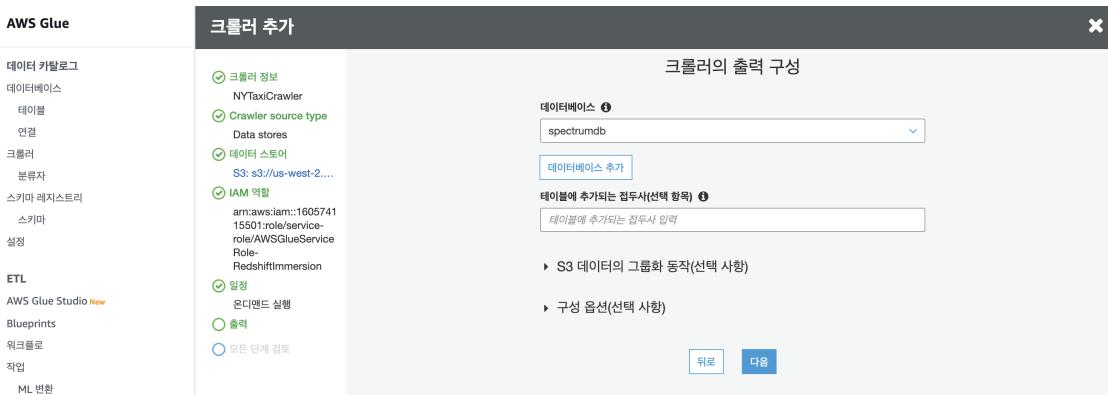


6. 빈도(Frequency)에 온디맨드 실행(Run on demand)를 선택합니다.

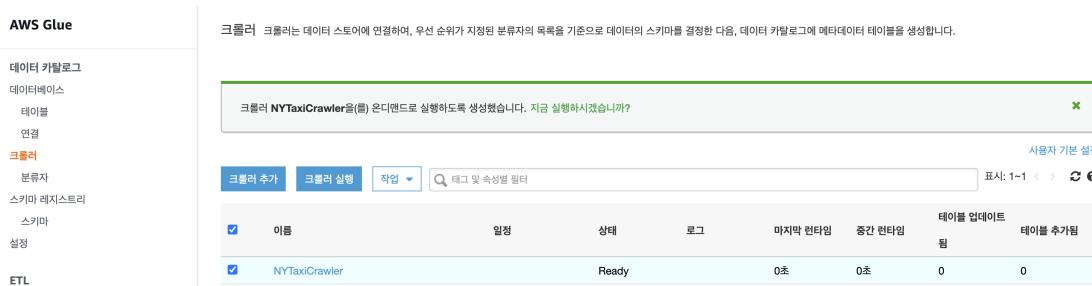


7. 데이터베이스 추가(Add database)를 클릭하고, Database 항목에 `spectrumdb`를 입력합니다.





8. Crawler가 생성되면, 크롤러 실행(Run Crawler)를 클릭합니다



AWS Glue

크롤러 크롤러는 데이터 스토어에 연결하여, 우선 순위가 지정된 분류자의 목록을 기준으로 데이터의 스키마를 결정한 다음, 데이터 카탈로그에 메타데이터 테이블을 생성합니다.

크롤러 "NYTaxiCrawler"이(가) 인료되었으며 다음과 같이 변경했습니다. 1개 테이블 생성, 0개 테이블 업데이트. 데이터베이스에서 생성된 테이블을 확인하십시오. spectrumdb.

크롤러 추가	크롤러 실행	작업	태그 및 속성 필터	표시: 1~1 < > ⌂ ⌂ ⌂				
<input type="checkbox"/>	이름	일정	상태	로그	마지막 빈티임	중간 빈티임	테이블 업데이트	테이블 추가됨
<input type="checkbox"/>	NYTaxiCrawler		Ready	로그	1분	1분	0	1

9. Crawler 실행이 완료되면, Glue Catalog에서 생성된 테이블을 확인할 수 있습니다.

- <https://us-west-2.console.aws.amazon.com/glue/home?region=us-west-2#catalog:tab=tables>

AWS Glue

테이블 테이블은 스키마를 비롯한 데이터를 표현하는 메타데이터 정의입니다. 테이블은 작업 정의의 원본 또는 대상으로 사용할 수 있습니다.

테이블 추가	작업	속성별 필터 또는 키워드 검색	보기 저장	표시: 1~1 < > ⌂ ⌂ ⌂		
<input type="checkbox"/>	이름	데이터베이스	위치	분류	최종 업데이트 날짜	사용 중단
<input type="checkbox"/>	ny_pub	spectrumdb	s3://us-west-2.serverless-analytics/canonical/NY-Pub/	parquet	4 2월 2022 11:38 오전 UTC+9	

10. ny_pub 테이블을 클릭하여 recordCount가 28억 여개가 입력되었음을 확인합니다.

AWS Glue

테이블 > ny_pub

최종 업데이트 날짜 4 2월 2022 11:38 오전 테이블 버전 (현재 버전) ▾

Partitions and indices	파티션 보기	버전 비교	스키마 편집

이름 ny_pub
설명
데이터베이스 spectrumdb
분류 parquet
위치 s3://us-west-2.serverless-analytics/canonical/NY-Pub/
연결
사용 중단 아니요
최종 업데이트 날짜 Fri Feb 04 11:38:53 GMT+900 2022
입력 형식 org.apache.hadoop.hive ql.io.parquet.MapredParquetInputFormat
출력 형식 org.apache.hadoop.hive ql.io.parquet.MapredParquetOutputFormat
Serde 직렬화 라이브러리 org.apache.hadoop.hive ql.io.parquet.serde.ParquetHiveSerDe
Serde 파라미터 serialization.format 1
sizeKey 32032431007 objectCount 7045 UPDATED_BY_CRAWLER NYTaxiCrawler
테이블 속성 CrawlerSchemaSerializerVersion 1.0 recordCount 2870781820 averageRecordSize 11
CrawlerSchemaDeserializerVersion 1.0 compressionType none typeOfData file

스키마

열 이름	데이터 형식	파티션 키	설명
1	vendorid	string	
2	pickup_datetime	timestamp	
3	dropoff_datetime	timestamp	
4	ratecode	int	
5	passenger_count	int	
6	trip_distance	double	
7	fare_amount	double	
8	total_amount	double	
9	payment_type	int	
10	year	string	Partition (0)
11	month	string	Partition (1)
12	type	string	Partition (2)

- 테이블 추출이 정상적으로 수행되었음을 확인하고, Redshift 쿼리에디터로 돌아갑니다. Glue Catalog Database spectrumdb 를 지정하여 External 스키마 `adb305` 를 생성합니다.

```
CREATE external SCHEMA adb305
FROM data catalog DATABASE 'spectrumdb'
IAM_ROLE default
CREATE external DATABASE if not exists;
```

- DAS(direct-attached storage) 대신 External 테이블을 사용하여 아래 쿼리를 실행합니다.

```
SELECT TO_CHAR(pickup_datetime, 'YYYY-MM-DD'),
COUNT(*)
FROM adb305.ny_pub
WHERE YEAR = 2016 and Month = 01
GROUP BY 1
ORDER BY 2;
```

통합 분석

- 이번에는 Spectrum을 활용한 S3와 Redshift DAS를 통합하여 View를 생성해 보겠습니다.

VIEW 생성

데이터과학자를 위하여 2016년 1월 기준 Green사의 S3 기반 데이터와 DAS 테이블을 활용하여 단일 테이블로 분석할 수 있도록 View를 생성합니다. CTAS 구문을 사용하여 테이블을 생성하고 COPY 명령어를 통해 실행했을 경우와 실행시간을 비교해 봅니다.

```
CREATE TABLE workshop_das.taxi_201601 AS
SELECT * FROM adb305.ny_pub
WHERE year = 2016 AND month = 1 AND type = 'green';
```

Note: Redshift에서는 CTAS 활용 시 아래 규칙에 따라서 압축과 암호화가 적용됩니다.

- 정렬키(sort key)로 정의된 컬럼은 RAW 압축 적용
- BOOLEAN, REAL, DOUBLE PRECISION, GEOMETRY 타입으로 정의된 컬럼은 RAW 압축 적용
- SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIMESTAMP, TIMESTAMPTZ로 정의된 컬럼은 AZ64 압축 적용
- CHAR, VARCHAR로 정의된 컬럼은 LZO 압축 적용

https://docs.aws.amazon.com/redshift/latest/dg/r_CTAS_usage_notes.html

```
ANALYZE COMPRESSION workshop_das.taxi_201601
```

Aa Column	Encoding	# Est_reduction_pct
<u>vendorid</u>	zstd	76.85
<u>pickup_datetime</u>	az64	0
<u>dropoff_datetime</u>	az64	0
<u>ratecode</u>	zstd	74.49
<u>passenger_count</u>	zstd	45.69
<u>trip_distance</u>	zstd	73.49
<u>fare_amount</u>	bytedict	85.9
<u>total_amount</u>	zstd	75.38
<u>payment_type</u>	az64	0
<u>year</u>	zstd	90.51
<u>month</u>	zstd	90.83
<u>type</u>	zstd	89.94

테이블에 추가 데이터 입력하기

INSERT/SELECT 구문을 활용하여 Green 컴퍼니 이외의 택시회사들의 2016년 1월 데이터를 입력하세요.

```
INSERT INTO workshop_das.taxi_201601 (
    SELECT *
    FROM adb305.ny_pub
    WHERE year = 2016 AND month = 1 AND type != 'green');
```

Spectrum 테이블의 중복된 데이터 제거하기

현재 모든 2016년 1월 데이터를 로딩 되었으므로 Spectrum 테이블과 DAS 테이블이 중복되지 않도록 Spectrum 테이블의 파티션을 삭제합니다.

```
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=1, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=1, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=1, type='yellow');
```

스키마 지정없이 VIEW 생성

DAS와 Spectrum 데이터를 동시에 활용하기 위해 VIEW를 생성해줍니다.

```
CREATE VIEW adb305_view_NYTaxiRides AS
    SELECT * FROM workshop_das.taxi_201601
    UNION ALL
    SELECT * FROM adb305.ny_pub
    WITH NO SCHEMA BINDING;
```

쿼리 유효성 검증

- SELECT/WHERE 절에서 파티션 컬럼이 사용되는 것을 확인합니다.
- 쿼리의 Spectrum 영역은 파티션 또는 파일 수준에서 필터가 적용된다는 점을 유의하세요.
- 실제 쿼리를 실행하여 수행시간을 확인해 보세요.

```
EXPLAIN
SELECT year, month, type, COUNT(*)
FROM adb305_view_NYTaxiRides
WHERE year = 2016 AND month IN (1) AND passenger_count = 4
GROUP BY 1,2,3 ORDER BY 1,2,3;
```

```
XN Merge (cost=1000250814498.70..1000250814498.70 rows=2 width=104)
Merge Key: derived_col1, derived_col2, derived_col3
-> XN Network (cost=1000250814498.70..1000250814498.70 rows=2 width=104)
    Send to leader
    -> XN Sort (cost=1000250814498.70..1000250814498.70 rows=2 width=104)
        Sort Key: derived_col1, derived_col2, derived_col3
        -> XN HashAggregate (cost=250814498.68..250814498.68 rows=2 width=104)
            -> XN Subquery Scan adb305_view_nytaxirides (cost=564453.61..250814498.66 rows=2 width=104)
                -> XN Append (cost=564453.61..250814498.64 rows=2 width=1556)
                    -> XN Subquery Scan "*SELECT* 1" (cost=564453.61..564453.62 rows=1 width=22)
                        -> XN HashAggregate (cost=564453.61..564453.61 rows=1 width=22)
                            -> XN Seq Scan on taxi_201601 (cost=0.00..560275.24 rows=417837 width=22)
                                Filter: ((passenger_count = 4) AND ((month)::text = '1'::text) AND ((year)::text = '2016'::text))
                            -> XN Subquery Scan "*SELECT* 2" (cost=250250045.00..250250045.02 rows=1 width=1556)
                                -> XN HashAggregate (cost=250250045.00..250250045.01 rows=1 width=1556)
                                    -> XN Partition Loop (cost=250250000.00..250250035.00 rows=1000 width=1556)
                                        -> XN Seq Scan PartitionInfo of adb305.ny_pub (cost=0.00..15.00 rows=1 width=1548)
                                            Filter: (((month)::text = '1'::text) AND ((year)::text = '2016'::text))
                                        -> XN S3 Query Scan ny_pub (cost=125125000.00..125125010.00 rows=1000 width=8)
                                            -> S3 Aggregate (cost=125125000.00..125125000.00 rows=1000 width=0)
                                                -> S3 Seq Scan adb305.ny_pub location:"s3://us-west-2.serverless-analyt
                                                    Filter: (passenger_count = 4)
```

- 이제 데이터가 Spectrum에 있는 달을 추가하여 실행해 봅니다.

```

EXPLAIN
SELECT year, month, type, COUNT(*)
FROM adb305_view_NYTaxiRides
WHERE year = 2016 AND month IN (1,2) AND passenger_count = 4
GROUP BY 1,2,3 ORDER BY 1,2,3;

```

```

XN Merge  (cost=1000250894540.52..1000250894540.52 rows=2 width=104)
Merge Key: derived_col1, derived_col2, derived_col3
-> XN Network  (cost=1000250894540.52..1000250894540.52 rows=2 width=104)
    Send to leader
    -> XN Sort  (cost=1000250894540.52..1000250894540.52 rows=2 width=104)
        Sort Key: derived_col1, derived_col2, derived_col3
        -> XN HashAggregate  (cost=250894540.50..250894540.50 rows=2 width=104)
            -> XN Subquery Scan adb305_view_nytaxirides  (cost=644492.93..250894540.48 rows=2 width=104)
                -> XN Append  (cost=644492.93..250894540.46 rows=2 width=1556)
                    -> XN Subquery Scan "*SELECT* 1"  (cost=644492.93..644492.94 rows=1 width=22)
                        -> XN HashAggregate  (cost=644492.93..644492.93 rows=1 width=22)
                            -> XN Seq Scan on taxi_201601  (cost=0.00..640314.56 rows=417837 width=22)
                                Filter: ((passenger_count = 4) AND ((year)::text = '2016'::text) AND ((month)::text = '1'))
                            -> XN Subquery Scan "*SELECT* 2"  (cost=250250047.50..250250047.52 rows=1 width=1556)
                                -> XN HashAggregate  (cost=250250047.50..250250047.51 rows=1 width=1556)
                                    -> XN Partition Loop  (cost=250250000.00..250250037.50 rows=1000 width=1556)
                                        -> XN Seq Scan PartitionInfo of adb305.ny_pub  (cost=0.00..17.50 rows=1 width=1548)
                                            Filter: ((year)::text = '2016'::text) AND ((month)::text = '1'::text) OR ((month)::text = '2')
                                        -> XN S3 Query Scan ny_pub  (cost=125125000.00..125125010.00 rows=1000 width=8)
                                            -> S3 Aggregate  (cost=125125000.00..125125000.00 rows=1000 width=0)
                                                -> S3 Seq Scan adb305.ny_pub location:"s3://us-west-2.serverless-analytics/ny-taxi-rides"
                                                    Filter: (passenger_count = 4)

```

```

EXPLAIN
SELECT passenger_count, COUNT(*)
FROM adb305.ny_pub
WHERE year = 2016 AND month IN (1,2)
GROUP BY 1 ORDER BY 1;

```

```

XN Merge  (cost=1000300005026.64..1000300005027.14 rows=200 width=12)
Merge Key: ny_pub.derived_col1
-> XN Network  (cost=1000300005026.64..1000300005027.14 rows=200 width=12)
    Send to leader
    -> XN Sort  (cost=1000300005026.64..1000300005027.14 rows=200 width=12)
        Sort Key: ny_pub.derived_col1
        -> XN HashAggregate  (cost=300005018.50..300005019.00 rows=200 width=12)
            -> XN Partition Loop  (cost=300000000.00..300004018.50 rows=200000 width=12)
                -> XN Seq Scan PartitionInfo of adb305.ny_pub  (cost=0.00..17.50 rows=1 width=0)
                    Filter: ((year)::text = '2016'::text) AND ((month)::text = '1'::text) OR ((month)::text = '2')
                -> XN S3 Query Scan ny_pub  (cost=150000000.00..150002000.50 rows=200000 width=12)
                    -> S3 HashAggregate  (cost=150000000.00..150000000.50 rows=200000 width=4)
                        -> S3 Seq Scan adb305.ny_pub location:"s3://us-west-2.serverless-analytics/canonical/NY-Pub/" f

```

```

EXPLAIN
SELECT type, COUNT(*)
FROM adb305.ny_pub
WHERE year = 2016 AND month IN (1,2)
GROUP BY 1 ORDER BY 1 ;

```

```

XN Merge  (cost=1000250000042.52..1000250000042.52 rows=1 width=524)
Merge Key: ny_pub."type"
-> XN Network  (cost=1000250000042.52..1000250000042.52 rows=1 width=524)
    Send to leader
    -> XN Sort  (cost=1000250000042.52..1000250000042.52 rows=1 width=524)
        Sort Key: ny_pub."type"
        -> XN HashAggregate  (cost=250000042.50..250000042.51 rows=1 width=524)
            -> XN Partition Loop  (cost=250000000.00..250000037.50 rows=1000 width=524)
                -> XN Seq Scan PartitionInfo of adb305.ny_pub  (cost=0.00..17.50 rows=1 width=516)
                    Filter: ((year)::text = '2016'::text) AND ((month)::text = '1'::text) OR ((month)::text = '2')
                -> XN S3 Query Scan ny_pub  (cost=125000000.00..125000010.00 rows=1000 width=8)
                    -> S3 Aggregate  (cost=125000000.00..125000000.00 rows=1000 width=0)
                        -> S3 Seq Scan adb305.ny_pub location:"s3://us-west-2.serverless-analytics/canonical/NY-Pub/" f

```

데이터 생애주기 관리

이번 Lab의 마지막은 HOT/COLD 데이터에 따른 전략을 실습하겠습니다. 최근 사용된 HOT 데이터는 Redshift DAS(direct-attached storage)에서 관리하고, 오래된 COLD 데이터는 S3에 관리하는 방법을 적용하도록 하겠습니다.

- 2015년 4분기 데이터를 Redshift DAS에 추가하여 이어지는 5개 분기의 리포팅 데이터를 대상으로 합니다.
 - 3개월 단위로 가장 오래된 분기 데이터를 “age-off” 대상으로 정의하여 DAS 테이블을 설계합니다.
 - 2015년 4분기를 제외되도록 Redshift Spectrum 테이블을 조정합니다.
- 2015년 4분기 데이터를 S3로 이동하여 실행될 수 있도록 구현합니다.
 - 각 단계를 정의하고, Redshift 추가 기능을 확인합니다.
 - 기존 Parquet 데이터를 활용하여 Redshift 각 단계를 시뮬레이션합니다. Redshift DAS에서 2015년 4분기 데이터를 “age-off”하고, 통합 분석(Single version of the truth)이 가능하도록 구성합니다.
- 해당 기능을 적용하기 위해서는 다양한 방법이 있습니다. 3개월 단위로 오래된 분기를 “age-off”하기 위해서 다음과 같이 쉽게 DAS 테이블을 설계할 수 있습니다.

```
CREATE OR REPLACE VIEW adb305_view_NYTaxiRides AS
    SELECT * FROM workshop_das.taxi_201504
UNION ALL
    SELECT * FROM workshop_das.taxi_201601
UNION ALL
    SELECT * FROM workshop_das.taxi_201602
UNION ALL
    SELECT * FROM workshop_das.taxi_201603
UNION ALL
    SELECT * FROM workshop_das.taxi_201604
UNION ALL
    SELECT * FROM adb305.ny_pub
WITH NO SCHEMA BINDING;
```

- 또는 아래처럼 구현할 수도 있습니다. Redshift의 Bulk Delete 기능은 굉장히 빠르기 때문에 활용가능한 방법입니다.(VACUUM까지 몇 분이내로 처리 가능)

```
CREATE OR REPLACE VIEW adb305_view_NYTaxiRides AS
    SELECT * FROM workshop_das.taxi_current
UNION ALL
    SELECT * FROM adb305.ny_pub
WITH NO SCHEMA BINDING;
```

- 필요시 COPY 명령어를 이용하여 Parquet 데이터를 DAS 테이블로 직접 입력할 수도 있습니다.

| 참고: Parquet 데이터를 COPY 명령어로 입력하는 경우 파티션 컬럼을 이용할 수 없습니다.

- 그럼 이런 경우 어떻게 적용하는지 살펴보겠습니다. 단일 테이블을 기준으로 설계해 보겠습니다.

```
CREATE TABLE workshop_das.taxi_current
DISTSTYLE EVEN
SORTKEY(year, month, type) AS
SELECT * FROM adb305.ny_pub WHERE 1 = 0;
```

- 그리고 Spectrum 테이블로 부터 파티션 컬럼이 포함되지 않은 Helper 테이블을 만들어 줍니다.

```
CREATE TABLE workshop_das.taxi_loader AS
SELECT vendorid, pickup_datetime, dropoff_datetime, ratecode, passenger_count,
       trip_distance, fare_amount, total_amount, payment_type
FROM workshop_das.taxi_current
WHERE 1 = 0;
```

Parquet Copy

- 아래 스크립트는 `type=green` 인 각각의 파티션을 구분하여 COPY 커맨드로 구성하였습니다. 스크립트 실행이 끝나면 다른 `type` 파티션에 대해서도 스크립트를 구성해야 합니다.

```
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analytics/canonical/NY-Pub/year=2015/month=10/type=green' IAM_ROLE defau
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analytics/canonical/NY-Pub/year=2015/month=11/type=green' IAM_ROLE defau
```

```

COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2015/month=12/type=green' IAM_ROLE defau
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2016/month=1/type=green' IAM_ROLE defaul
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2016/month=2/type=green' IAM_ROLE defaul
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2016/month=3/type=green' IAM_ROLE defaul
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2016/month=4/type=green' IAM_ROLE defaul
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2016/month=5/type=green' IAM_ROLE defaul
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2016/month=6/type=green' IAM_ROLE defaul
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2016/month=7/type=green' IAM_ROLE defaul
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2016/month=8/type=green' IAM_ROLE defaul
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2016/month=9/type=green' IAM_ROLE defaul
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2016/month=10/type=green' IAM_ROLE defau
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2016/month=11/type=green' IAM_ROLE defau
COPY workshop_das.taxi_loader FROM 's3://us-west-2.serverless-analyticscanonical/NY-Pub/year=2016/month=12/type=green' IAM_ROLE defau

```

```

INSERT INTO workshop_das.taxi_current
SELECT *, DATE_PART(year,pickup_datetime), DATE_PART(month,pickup_datetime), 'green'
FROM workshop_das.taxi_loader;

```

```
TRUNCATE workshop_das.taxi_loader;
```

- Spectrum 테이블을 활용하여 데이터를 입력할 수도 있습니다.

```

DROP TABLE IF EXISTS workshop_das.taxi_201601;
CREATE TABLE workshop_das.taxi_201601 AS SELECT * FROM adb305.ny_pub WHERE year = 2016 AND month IN (1,2,3);
CREATE TABLE workshop_das.taxi_201602 AS SELECT * FROM adb305.ny_pub WHERE year = 2016 AND month IN (4,5,6);
CREATE TABLE workshop_das.taxi_201603 AS SELECT * FROM adb305.ny_pub WHERE year = 2016 AND month IN (7,8,9);
CREATE TABLE workshop_das.taxi_201604 AS SELECT * FROM adb305.ny_pub WHERE year = 2016 AND month IN (10,11,12);

```

- 아래는 2015년 4분기가 제외되도록 Spectrum 테이블을 조정한 것입니다.

```

ALTER TABLE adb305.ny_pub DROP PARTITION(year=2015, month=10, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2015, month=10, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2015, month=10, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2015, month=11, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2015, month=11, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2015, month=11, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2015, month=12, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2015, month=12, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2015, month=12, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=1, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=1, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=1, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=2, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=2, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=2, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=3, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=3, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=3, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=4, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=4, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=4, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=5, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=5, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=5, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=6, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=6, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=6, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=7, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=7, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=7, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=8, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=8, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=8, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=9, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=9, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=10, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=10, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=11, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=11, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=11, type='green');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=12, type='yellow');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=12, type='fhv');
ALTER TABLE adb305.ny_pub DROP PARTITION(year=2016, month=12, type='green');

```

- DAS에서는 5분기를 Spectrum에서는 모든 데이터를 통합하여 사용이 가능합니다. 그럼 2015년 4분기 데이터를 “age-off” 하는 절차는 어떻게 될까요?
 - Redshift DAS 테이블의 데이터를 S3에 저장
 - UNLOAD
 - https://docs.aws.amazon.com/ko_kr/redshift/latest/dg/r_UNLOAD.html
 - Spectrum 테이블이 2015년 4분기를 커버할 수 있도록 확장
 - ADD Partition
 - Redshift DAS 테이블 제거
 - 구현 방법에 따라서 DELETE / DROP TABLE

5. Spectrum 쿼리 튜닝

이번 실습에서는 Redshift Spectrum 쿼리 성능을 진단하여 파티션 활용, 스토리지 최적화, 조건절 튜닝을 통하여 성능을 최적화합니다.

시작에 앞서

- 이번 Lab을 수행하기 위해서 Redshift 클러스터와 클라이언트 도구 설정이 필요합니다. [Lab 1 - 클러스터 생성 및 연결](#) 이 먼저 선행되어야 합니다.
- 이번 Lab을 위해서는 [US-WEST-2\(Oregon\)](#) 지역에 클러스터를 구성해야 합니다.

Redshift Spectrum 쿼리 구성

Redshift 클러스터에 Dimension 테이블을 생성하고, S3에는 Fact 테이블을 생성하여 Star 스키마 모델을 구성합니다.

- 아래 스크립트를 수행하여 Dimension 테이블을 생성합니다.

```

DROP TABLE IF EXISTS customer;
CREATE TABLE customer (
    c_custkey      integer      not null sortkey,
    c_name         varchar(25)   not null,
    c_address      varchar(25)   not null,
    c_city         varchar(10)    not null,
    c_nation       varchar(15)    not null,
    c_region       varchar(12)    not null,
    c_phone        varchar(15)    not null,
    c_mktsegment   varchar(10)    not null)
diststyle all;

DROP TABLE IF EXISTS dwdate;
CREATE TABLE dwdate (
    d_datekey      integer      not null sortkey,
    d_date         varchar(19)   not null,
    d_dayofweek    varchar(10)   not null,
    d_month        varchar(10)   not null,
    d_year         integer      not null,
    d_yearmonthnum integer      not null,
    d_yearmonth    varchar(8)    not null,
    d_daynuminweek integer      not null,
    d_daynuminmonth integer      not null,
    d_daynuminyear integer      not null,
    d_monthnuminyear integer      not null,
    d_weeknuminyear integer      not null,
    d_sellingseason varchar(13)   not null,
    d_lastdayinweekfl varchar(1)  not null,
    d_lastdayinmonthfl varchar(1)  not null,
    d_holidayfl    varchar(1)  not null,
    d_weekdayfl    varchar(1)  not null)
diststyle all;
  
```

2. 다음 스크립트를 실행하여 Dimension 테이블에 데이터를 로딩합니다. 클러스터에 COPY 명령어를 실행시키기 위해 IAM 역할을 입력해야 합니다. S3로부터 Redshift 클러스터에 데이터 셋이 적재됩니다. 수행에 몇 분정도가 소용되고, Customer는 약 3만개, Dwddate는 2556개의 데이터로 구성되어 있습니다.

```
copy customer from 's3://awssampledbuswest2/ssbgz/customer'
iam_role default
gzip region 'us-west-2';

copy dwddate from 's3://awssampledbuswest2/ssbgz/dwddate'
iam_role default
gzip region 'us-west-2';
```

3. 다음은 Redshift 클러스터 외부에 존재하는 참조 데이터셋을 위한 External Schema를 생성합니다. 아래 명령어를 실행하여 스키마를 정의합니다. S3 데이터를 읽기 위해서 적합한 권한의 IAM 역할이 필요합니다. AWS Glue data catalog에 External DB / Schema 가 정의된 메타데이터를 정의합니다. 생성이 마무리되면 Glue나 Athena를 통해 해당 스키마를 확인할 수 있습니다.

```
CREATE EXTERNAL SCHEMA clickstream
from data catalog database 'clickstream'
iam_role default
CREATE EXTERNAL DATABASE IF NOT EXISTS;
```

The screenshot shows the AWS Glue Data Catalog interface. On the left, there's a sidebar with options like '데이터 카탈로그', '데이터베이스', '테이블', '연결', and '크롤러'. The main area has tabs for '데이터베이스 추가', '데이터베이스 보기', and '작업'. A search bar at the top says '이름' and has a dropdown with 'clickstream'. Below the search bar is a '설명' field. At the bottom right, there's a pagination indicator '표시: 1-2 < > 🔍 🔍'.

4. AWS Glue Crawler를 사용하여 `s3://redshift-spectrum-bigdata-blog-datasets/clickstream-csv10`, `s3://redshift-spectrum-bigdata-blog-datasets/clickstream-parquet1` External 테이블을 생성합니다.

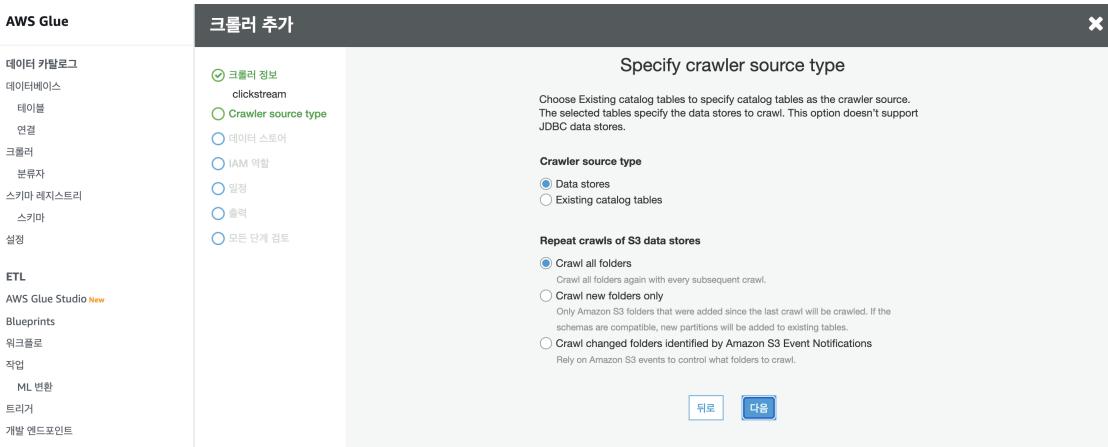
- `s3://redshift-spectrum-bigdata-blog-datasets/clickstream-csv10`
 - `s3://redshift-spectrum-bigdata-blog-datasets/clickstream-parquet1`
- i. [Glue Crawler Page](#) 접속

The screenshot shows the AWS Glue Crawler page. On the left, there's a sidebar with 'Data catalog', 'Tables', 'Connections', 'Crawlers' (which is selected), 'Classifiers', and 'Settings'. The main area has a heading 'Crawlers' with a sub-instruction: 'A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata to describe the data in the data catalog.' Below this are buttons for 'Add crawler', 'Run crawler', and 'Action'. There's also a search bar 'Filter by tags and attributes'. A message 'You don't have any crawlers yet.' is displayed above a 'Add crawler' button. The table below has columns: Name, Schedule, Catalog type, Status, Logs, Last runtime, Median runtime, and Tables up.

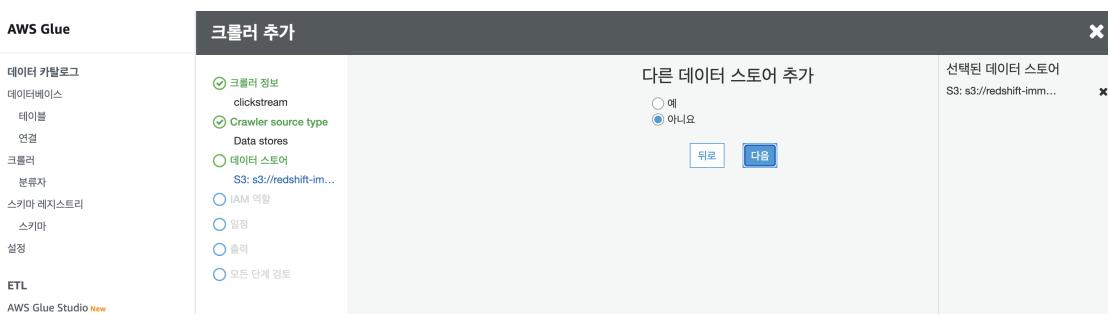
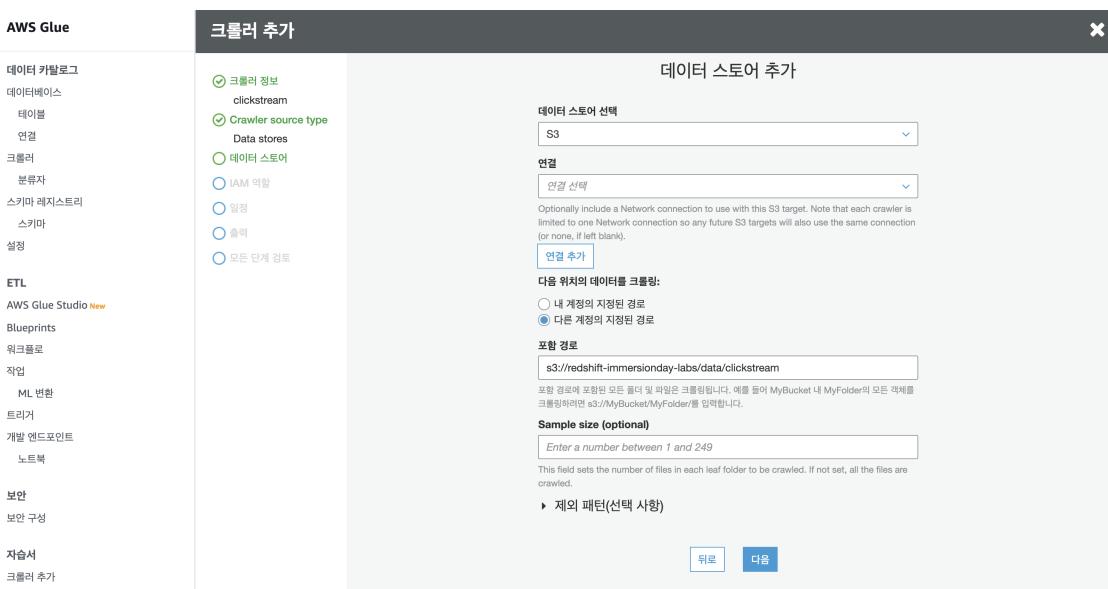
ii. [Add Crawler](#) 클릭, Crawler name에 `clickstream` 입력 후 NEXT 클릭

The screenshot shows the 'Add Crawler' dialog box. On the left, there's a sidebar with the same options as the previous screenshot. The main area has a title '크롤러 추가' and a sub-section '크롤러 정보 추가'. It shows a list of configuration options: '크롤러 정보' (selected), 'Crawler source type' (radio button), '데이터 스토어' (radio button), 'IAM 역할' (radio button), '일정' (radio button), '출력' (radio button), and '모든 단계 검토' (radio button). To the right, there's a '크롤러 이름' input field containing 'clickstream'. Below it is a note: '▶ 태그, 설명, 보안 구성, 분류자(선택 항목)'. At the bottom right is a '다음' (Next) button.

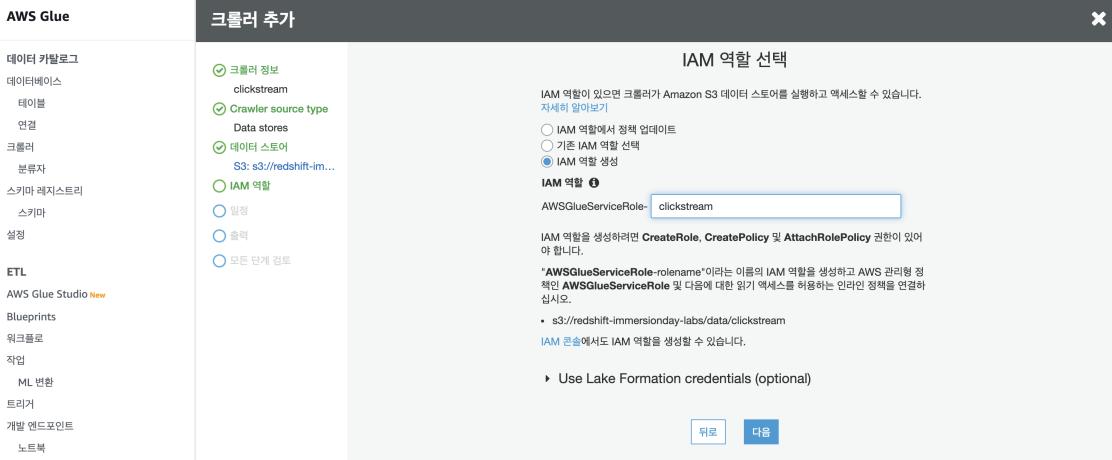
iii. [Data stores](#) 선택 후 Next 클릭



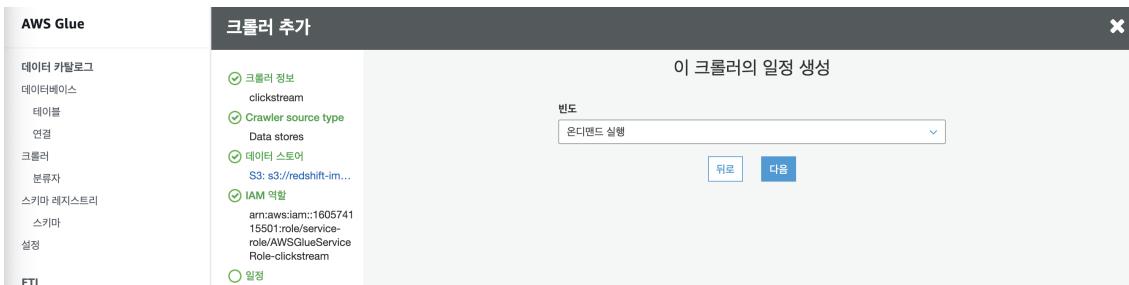
iv. Choose a data store `s3` 선택, Specified path in another account 선택, Include path `s3://redshift-immersionday-labs/data/clickstream` 입력



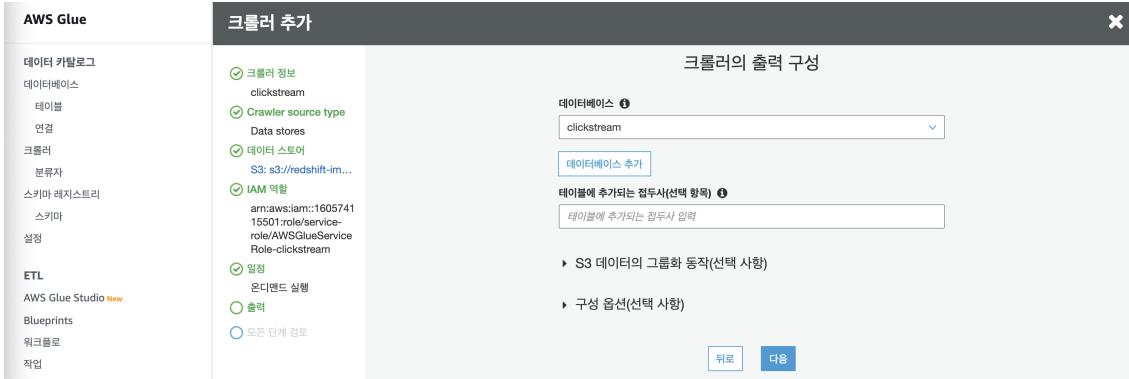
v. IAM 역할 생성(Create an IAM Role)을 선택하고, AWSGlueServiceRole-`clickstream`을 입력합니다.

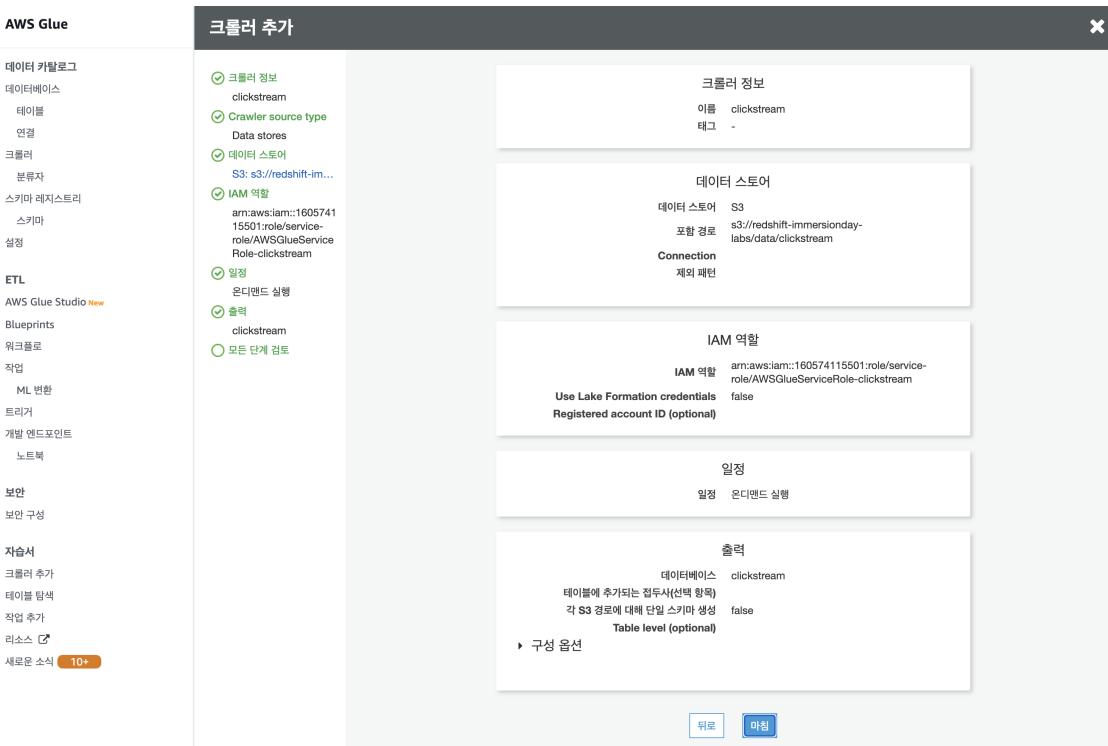


vi. Run on demand 선택 후 Next 클릭



vii. 데이터베이스 리스트에서 clickstream 선택합니다.





viii. Crawler가 생성되면 크롤러 실행 버튼을 클릭합니다.

이름	상태	로그	마지막 런타임	중간 런타임	테이블 업데이트 릴	테이블 추가됨
NYTaxiCrawler	Ready	로그	1분	1분	0	1
clickstream	Ready		0초	0초	0	0

ix. 크롤링이 완료되면 Glue Catalog에서 새로운 테이블이 생성된 것을 확인할 수 있습니다.

- <https://us-west-2.console.aws.amazon.com/glue/home?region=us-west-2#catalog:tab=tables>

이름	데이터베이스	위치	분류	최종 업데이트 날짜	사용 중단
uservisits_parquet1	clickstream	s3://redshift-immersionday-labs/data/clickstream/parquet	parquet	4 2월 2022 3:10 오전 UTC+9	
uservisits_csv10	clickstream	s3://redshift-immersionday-labs/data/clickstream/csv	csv	4 2월 2022 3:10 오전 UTC+9	

x. `uservisits_parquet1` 테이블을 클릭하고 37억건의 데이터가 로딩되었는지 확인합니다.

테이블 > uservisits_parquet1

서비스

테이블 편집 테이블 삭제

최종 업데이트 날짜 4 2월 2022 03:10 오후 테이블 버전 (현재 버전) ▾

Partitions and indices 파티션 보기 버전 비교 스키마 편집

이름	uservisits_parquet1					
설명						
데이터베이스	clickstream					
분류	parquet					
위치	s3://redshift-immersionday-labs/data/clickstream/uservisits_parquet1/					
연결						
사용 중단	아니요					
최종 업데이트 날짜	Fri Feb 04 15:10:53 GMT+900 2022					
입력 형식	org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat					
출력 형식	org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat					
Serde	직렬화 라이브러리	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe				
Serde 파라미터	serialization.format	1				
sizeKey	124762272367	objectCount	503	UPDATED_BY_CRAWLER	clickstream	
테이블 속성	CrawlerSchemaSerializerVersion	1.0	recordCount	3751427282	averageRecordSize	87
	CrawlerSchemaDeserializerVersion	1.0	compressionType	none	typeOfData	file

스키마

표시: 1~13/13 < >

	열 이름	데이터 형식	파티션 키	설명
1	custkey	int		
2	yearmonthkey	int		
3	visitdate	int		
4	adrevenue	double		
5	countrycode	string		
6	desturl	string		
7	duration	int		
8	languagecode	string		
9	searchword	string		
10	sourceip	string		
11	useragent	string		
12	customer	string	Partition (0)	
13	visityearmonth	string	Partition (1)	

xi. Glue Catalog에서 돌아가서 *uservisits_csv10* 클릭합니다. 아래와 같이 컬럼명이 임의로 생성된것을 볼수 있으며, col0의 데이터입은 String으로 설정되었습니다. 이 필드의 경우는 *adRevenue* 항목을 의미하며 double 데이터 타입이 되어야 합니다.

테이블 > uservisits_csv10

최종 업데이트 날짜 4 2월 2022 03:10 오후 테이블 버전 (현재 버전) ▾

[테이블 편집](#) [테이블 삭제](#)

[Partitions and Indices](#) [파티션 보기](#) [버전 비교](#) [스키마 편집](#)

이름	uservisits_csv10				
설명					
데이터베이스	clickstream				
분류	csv				
위치	s3://redshift-immersionday-labs/data/clickstream/uservisits_csv10/				
연결					
사용 중단	아니요				
최종 업데이트 날짜	Fri Feb 04 15:10:53 GMT+900 2022				
입력 형식	org.apache.hadoop.mapred.TextInputFormat				
출력 형식	org.apache.hadoop.hive.io.HiveIgnoreKeyTextOutputFormat				
Serde	직렬화 라이브러리: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe				
Serde 파라미터	field.delim : ,				
sizeKey	660065374805	objectCount	5030	UPDATED_BY_CRAWLER	clickstream
CrawlerSchemaSerializerVersion	1.0	recordCount	3807645431	averageRecordSize	155
테이블 속성	CrawlerSchemaDeserializerVersion: 1.0	compressionType: none	columnsOrdered: true	areColumnsQuoted: false	delimiter: , typeOfData: file

스키마

표시: 1~13/13 < >

	열 이름	데이터 형식	파티션 키	설명
1	col0	string		
2	col1	string		
3	col2	bigint		
4	col3	string		
5	col4	bigint		
6	col5	string		
7	col6	string		
8	col7	string		
9	col8	string		
10	col9	bigint		
11	col10	bigint		
12	customer	string	Partition (0)	
13	visityearmonth	string	Partition (1)	

xii. *Edit Schema* 클릭하고 컬럼명과 데이터타입을 아래와 같이 변경 후 저장합니다.

열 이름	데이터 형식	키
1 adrevenue	double	
2 countrycode	string	
3 custkey	bigint	
4 desturl	string	
5 duration	bigint	
6 languagecode	string	
7 searchword	string	
8 sourceip	string	
9 useragent	string	
10 visitdate	bigint	
11 yearmonthkey	bigint	
12 customer	string	Partition (0)
13 visityearmonth	string	Partition (1)



컬럼명

adrevenue ## double
 countrycode
 custkey
 desturl
 duration
 languagecode
 searchword
 sourceip
 useragent
 visitdate
 yearmonthkey
 customer
 visityearmonth

5. 쿼리에디터에서 아래 쿼리를 실행합니다. Redshift의 Dimension 테이블과 S3의 clickstream Fact 테이블을 병합하여 조인처리를 합니다. 해당 쿼리는 customer 1-3에 대하여 최근 3개월간의 광고수익(ad revenue) 결과를 조회합니다. 광고 수익(adrevenue) 데이터는 S3로 부터 추출하였고, 마켓 세그먼트 데이터와 같은 시간 속성은 Redshift의 Dimension 테이블에서 추출하였습니다.

```

SELECT c.c_name, c.c_mktsegment, t.prettyMonthYear, SUM(uv.adRevenue)
FROM clickstream.uservisits_csv10 as uv
RIGHT OUTER JOIN customer as c ON c.c_custkey = uv.custKey
INNER JOIN (
  SELECT DISTINCT d_yearmonthnum, (d_month||','||d_year) as prettyMonthYear
  FROM dwdate
  WHERE d_yearmonthnum >= 199810) as t ON uv.yearMonthKey = t.d_yearmonthnum
WHERE c.c_custkey <= 3
GROUP BY c.c_name, c.c_mktsegment, t.prettyMonthYear, uv.yearMonthKey
ORDER BY c.c_name, c.c_mktsegment, uv.yearMonthKey ASC
  
```

해당 쿼리는 37억개의 데이터를 검색하기 때문에 몇 분의 수행시간(약 10분)이 걸리며 결과는 아래와 같습니다.

c_name	c_mktsegment	prettymonthyear	sum
Customer#000000001	BUILDING	October,1998	3596847.838855554
Customer#000000001	BUILDING	November,1998	3776957.0441623246
Customer#000000001	BUILDING	December,1998	3674480.427097163
Customer#000000002	AUTOMOBILE	October,1998	3593281.2807776257
Customer#000000002	AUTOMOBILE	November,1998	3777930.644197874
Customer#000000002	AUTOMOBILE	December,1998	3671834.14359701
Customer#000000003	AUTOMOBILE	October,1998	3596234.3073224304
Customer#000000003	AUTOMOBILE	November,1998	3776715.017401728
Customer#000000003	AUTOMOBILE	December,1998	3674360.280659755

성능진단

아래 도구를 이용하여 Redshift Spectrum 성능을 진단합니다.

- EXPLAIN - 쿼리 실행 계획을 제공합니다. 접두사에 S3가 있을 경우 Spectrum에서 실행됩니다.
- SVL_S3QUERY_SUMMARY - 테이블에 저장되어 있는 Spectrum 쿼리의 통계정보를 제공합니다. 실행계획은 예상 비용을 제공하지만, 해당 테이블은 과거 실행된 쿼리를 기반으로 실제 통계정보를 저장하고 있습니다.
- SVL_S3PARTITION - 세그먼트와 노드 레벨에서 Spectrum 파티션 관련 정보를 제공합니다.

1. 아래 쿼리를 수행하여 SVL_S3QUERY_SUMMARY 테이블 정보를 확인합니다.

```
select elapsed, s3_scanned_rows, s3_scanned_bytes,
       s3query_returned_rows, s3query_returned_bytes, files, avg_request_parallelism
  from svl_s3query_summary
 where query = pg_last_query_id()
  order by query,segment;
```

elapsed	s3_scanned_rows	s3_scanned_bytes	s3query_returned_rows	s3query_returned_bytes	files	avg_request_parallelism
555245768	3751427282	660401574485	66270117	494740098	5030	10

- 이 진단에서 왜 쿼리가 오래 걸렸는지 확인할 수 있습니다. s3_scanned_row 결과를 보면 쿼리가 38억건의 데이터를 모두 스캔하고 있습니다.
- 아래 쿼리를 이용하여 Spectrum이 모든 파티션을 스캔하는 것을 볼 수 있습니다.

```
SELECT query, segment,
       MIN(starttime) AS starttime,
       MAX(endtime) AS endtime,
       datediff(ms,MIN(starttime),MAX(endtime)) AS dur_ms,
       MAX(total_partitions) AS total_partitions,
       MAX(qualified_partitions) AS qualified_partitions,
       MAX(assignment) as assignment_type
  FROM svl_s3partition
 WHERE query=pg_last_query_id()
 GROUP BY query, segment;
```

2. EXPLAIN 명령어와 함께 Redshift Spectrum 쿼리를 재수행 합니다.

```

EXPLAIN
SELECT c.c_name, c.c_mktsegment, t.prettyMonthYear, SUM(uv.adRevenue)
FROM clickstream.uservisits_csv10 as uv
RIGHT OUTER JOIN customer as c ON c.c_custkey = uv.custKey
INNER JOIN (
    SELECT DISTINCT d_yeарmonthnum, (d_month||','||d_year) as prettyMonthYear
    FROM dwdate WHERE d_yeарmonthnum >= 199810) as t ON uv.yearMonthKey = t.d_yeарmonthnum
WHERE c.c_custkey <= 3
GROUP BY c.c_name, c.c_mktsegment, t.prettyMonthYear, uv.yearMonthKey
ORDER BY c.c_name, c.c_mktsegment, uv.yearMonthKey ASC

```

- 결과값은 아래와 같습니다. 지금은 실행 계획에 대한 상세정보를 모두 이해하지 않아도 됩니다.

```

XN Merge (cost=1175163695866.12..1175163696909.42 rows=417322 width=82)
Merge Key: c.c_name, c.c_mktsegment, uv.yeарmonthkey
-> XN Network (cost=1175163695866.12..1175163696909.42 rows=417322 width=82)
    Send to leader
    -> XN Sort (cost=1175163695866.12..1175163696909.42 rows=417322 width=82)
        Sort Key: c.c_name, c.c_mktsegment, uv.yeарmonthkey
        -> XN HashAggregate (cost=175163655864.13..175163656907.43 rows=417322 width=82)
            -> XN Hash Join DS_DIST_ALL_NONE (cost=37.34..175163608057.37 rows=3824541 width=82)
                Hash Cond: ("outer".yeарmonthkey = ("inner".d_yeарmonthnum)::bigint)
                -> XN Hash Join DS_DIST_ALL_NONE (cost=4.50..175162203871.62 rows=109272600 width=50)
                    Hash Cond: ("outer".custkey = ("inner".c_custkey)::bigint)
                    -> XN Partition Loop (cost=0.00..161272222241.12 rows=111111112000 width=24)
                        -> XN Seq Scan PartitionInfo of clickstream.uservisits_csv10 uv (cost=0.00..10.00 rows=1000 width=24)
                        -> XN S3 Query Scan uv (cost=0.00..16111111.12 rows=111111112 width=24)
                            -> S3 Seq Scan clickstream.uservisits_csv10 uv location:"s3://redshift-immersionday-labs/clickstream/uservisits_csv10.csv" Filter: ((custkey <= 3) AND (yeарmonthkey >= 199810))
                        -> XN Hash (cost=3.75..3.75 rows=300 width=38)
                            -> XN Seq Scan on customer c (cost=0.00..3.75 rows=300 width=38)
                                Filter: (c.custkey <= 3)
                        -> XN Hash (cost=32.82..32.82 rows=7 width=36)
                            -> XN Subquery Scan t (cost=0.00..32.82 rows=7 width=36)
                                -> XN Unique (cost=0.00..32.75 rows=7 width=18)
                                    -> XN Seq Scan on dwdate (cost=0.00..32.43 rows=64 width=18)
                                        Filter: (d_yeарmonthnum >= 199810)

```

위 쿼리 실행계획의 핵심은 Redshift Spectrum이 어떻게 작동하는지에 있습니다. 아래 부분을 통해서 Redshift Spectrum은 스캔 작업을 위해서 쿼리의 일부분으로만 활용되는 것을 알 수 있습니다. 또한 파티션은 사용되지 않았으며 해당 부분은 다음 단락에서 더 자세히 알아보겠습니다.

```

-> S3 Seq Scan clickstream.uservisits_csv10 uv location:"s3://redshift-immersionday-labs/clickstream/uservisits_csv10.csv" format:T

```

파티션을 활용한 최적화

이번에는 파티션에 대해서 알아보고 Redshift Spectrum의 성능 향상을 위해서 어떻게 활용되는지 살펴보겠습니다. 파티셔닝은 스캔을 효율적으로 하는 방법입니다. 이전에 Glue Crawler를 수행하였을 때 External 테이블이 파티션을 기준으로 생성된 것을 확인하였습니다. Glue Catalog을 <https://console.aws.amazon.com/glue/home?#catalog:tab=tables> 접속하여 uservisits_csv10 테이블을 클릭합니다. *customer*와 *visityeарmonth*는 파티션키입니다.

열 이름	데이터 형식	키
1 adrevenue	double	
2 countrycode	string	
3 custkey	bigint	
4 desturl	string	
5 duration	bigint	
6 languagecode	string	
7 searchword	string	
8 sourceip	string	
9 useragent	string	
10 visitdate	bigint	
11 yearmonthkey	bigint	
12 customer	string	Partition (0)
13 visityearmonth	string	Partition (1)

다음 위치로 이동하여 Redshift Spectrum 데이터셋을 제공하는 S3 버킷을 확인할 수 있습니다

- https://s3.console.aws.amazon.com/s3/buckets/redshift-immersionday-labs/data/clickstream/uservisits_csv10/

전체 38억 행 데이터셋은 각 파일에 특정 고객 및 년월 데이터가 포함된 대용량 파일 모음으로 구성됩니다. 이렇게 하면 데이터를 고객 및 연도/월별로 논리적 하위 집합으로 분류할 수 있습니다. 파티션을 사용하면 쿼리 엔진이 파일의 서브셋을 타겟하여 스캔할 수 있습니다.

- 특정 고객만 대상
- 특정 월 기준
- 특정 고객 및 년/월 조합

파티션 전략은 워크로드 특성에 따라서 선택되어야 합니다. 최적화하려는 기본 쿼리와 데이터 프로필을 기반으로 파티션을 선택해야 합니다. 자체 클릭스트림 분석을 구현하는 경우 연도/월/지역과 같은 파티션이 의미있는 경우가 많습니다. 고객이 매우 많고 각 고객에 대한 데이터가 거의 없는 사용 사례에서 고객 파티션기는 도움이 되지 않습니다. 이 예제에서 사용된 데이터셋의 경우 멀티테넌트 광고 기술 플랫폼 또는 IoT 플랫폼과 같은 시나리오에 대해 유용할 것입니다. 이러한 경우 고객의 수는 적당하고 고객당 데이터는 많습니다.

1. 다음 쿼리를 실행하여 쿼리에서 파티셔닝을 적용했을 때 효과를 확인해 보겠습니다.

```

SELECT c.c_name, c.c_mktsegment, t.prettyMonthYear, SUM(uv.adRevenue)
FROM clickstream.uservisits_csv10 as uv
RIGHT OUTER JOIN customer as c ON c.c_custkey = uv.customer
INNER JOIN
  (SELECT DISTINCT d_yearmonthnum, (d_month||','||d_year) as prettyMonthYear
   FROM dwdate
   WHERE d_yearmonthnum >= 199810) as t ON uv.yearMonthKey = t.d_yearmonthnum
WHERE c.c_custkey <= 3
GROUP BY c.c_name, c.c_mktsegment, t.prettyMonthYear, uv.yearMonthKey
ORDER BY c.c_name, c.c_mktsegment, uv.yearMonthKey ASC

```

- 이전 쿼리에서 조인 조건을 수정하였습니다. `custKey`를 활용하여 조인하는 대신 데이터 모델링 프로세스의 일부로 생성한 파티션 키 `customer`를 사용합니다. 이 쿼리는 이전보다 약 2배 빠르게 실행됩니다.

- 동일한 Redshift Spectrum 쿼리를 EXPLAIN 명령어로 다시 실행합니다. 이전과 달리 PartitionInfo 스캔의 일부로 Filter 절이 적용되었습니다.

```
-> XN Seq Scan PartitionInfo of clickstream.uservisits_csv10 uv  (cost=0.00..10.00 rows=1000 width=516)
   Filter: (subplan 1: ((customer)::text = ($2)::text))
```

- SVL_S3QUERY_SUMMARY 쿼리를 재실행 합니다.

```
select elapsed, s3_scanned_rows, s3_scanned_bytes,
       s3query_returned_rows, s3query_returned_bytes, files, avg_request_parallelism
  from svl_s3query_summary
 where query = pg_last_query_id()
  order by query,segment;
```

elapsed	s3_scanned_rows	s3_scanned_bytes	s3query_returned_rows	s3query_returned_bytes	files	avg_request_parallelism
264330005	1898739653	334255910285	66270117	494673572	2520	9.9

- `s3_scanned_rows` 스캔된 행이 이전 쿼리와 비교할 때 절반으로 줄었습니다. 이로 인해 쿼리는 2배 이상 빠르게 수행됩니다.
- 결과적으로 데이터가 모든 고객에게 고르게 분포되어 있고 고객 파티션 키로 6명의 고객 중 3명을 쿼리하였기 때문에 데이터베이스 엔진이 고객 1, 2, 3을 포함하는 데이터의 집합에 대해서만 스캔하였습니다. 그러나 스캔은 여전히 매우 비효율적이어서 추가로 년/월 파티션 키를 활용해 보도록 하겠습니다.

- 아래 쿼리를 실행 합니다.

```
SELECT c.c_name, c.c_mktsegment, t.prettyMonthYear, SUM(uv.adRevenue)
  FROM clickstream.uservisits_csv10 as uv
  RIGHT OUTER JOIN customer as c ON c.c_custkey = uv.customer
 INNER JOIN (
    SELECT DISTINCT d_yeарmonthnum, (d_month||','||d_year) as prettyMonthYear
      FROM dwdate
     WHERE d_yeарmonthnum >= 199810) as t ON uv.visitYearMonth = t.d_yeарmonthnum
   WHERE c.c_custkey <= 3
 GROUP BY c.c_name, c.c_mktsegment, t.prettyMonthYear, uv.yearMonthKey
 ORDER BY c.c_name, c.c_mktsegment, uv.yearMonthKey ASC
```

- 이전 쿼리에서 조인 조건이 수정되었습니다. 합성 키인 `yearMonthKey`에 조인하는 대신 파티션 키인 `visitYearMonth`를 사용합니다. 이번 쿼리에서 고객 파티션과 시간 파티션을 모두 활용하며 기존 쿼리보다 20배 이상 빠릅니다.

- SVL_S3QUERY_SUMMARY 쿼리를 재실행 합니다.

```
select elapsed, s3_scanned_rows, s3_scanned_bytes,
       s3query_returned_rows, s3query_returned_bytes, files, avg_request_parallelism
  from svl_s3query_summary
 where query = pg_last_query_id()
  order by query,segment;
```

- 통계 정보를 활용해서 Redshift Spectrum이 쿼리를 계산하는 데 필요한 행 수만 검색하고 반환하는 것을 확인할 수 있습니다.

elapsed	s3_scanned_rows	s3_scanned_bytes	s3query_returned_rows	s3query_returned_bytes	files	avg_request_parallelism
12000125	66270117	11666574974	66270117	494673572	90	8

스토리지 최적화

Redshift Spectrum은 클러스터 외부의 대규모 인프라를 통해 처리합니다. 즉, S3에서 대규모 스캔 및 집계를 수행하는 데 최적화되어 있습니다. 최적화된 Redshift Spectrum은 특정 워크로드에서 작은 규모의 Redshift 클러스터보다 성능이 뛰어나기도 합니다. 대규모 스캔

및 집계를 최적화하기 위해 고려해야 할 두 가지 중요한 변수가 있습니다.

- **File size and count** - 일반적으로 분할되지 않는 파일을 기준으로 50-500MB 사이의 파일 크기를 사용합니다. 이는 Redshift Spectrum에 가장 적합합니다. 그러나 쿼리에서 작동하는 파일의 수는 쿼리에서 달성할 수 있는 병렬 처리 성능과 직접적인 상관 관계가 있습니다. 파일 크기와 개수 사이에는 반비례 관계에 있습니다. 동일한 데이터셋을 기준으로 파일이 커지면 파일 수는 적어지게 됩니다. 결과적으로 읽기 성능에 대한 최적화와 병렬 처리 기능에는 트레이드 오프 관계가 성립됩니다. 쿼리가 많은 수의 파일에서 작동할 가능성이 높기 때문에 대용량 파일은 대용량 스캔에 가장 적합합니다. 적은 수의 파일을 선택하여 작동하는 쿼리라면 파일이 작을 수록 병렬처리 성능이 높아질 것입니다.
- **Data format** - Redshift Spectrum은 다양한 데이터 형식을 지원합니다. Parquet과 같은 컬럼 타입은 특정 워크로드에서 압축과 효율적인 I/O를 제공하여 성능을 극대화 시킬 수 있습니다. 일반적으로 Parquet와 같은 유형은 대규모 스캔을 선택적으로 활용하는 쿼리 워크로드에서 사용해야 합니다. Parquet과 같은 형식은 일반 텍스트보다 처리하는 데 더 많은 컴퓨팅 성능을 필요로하는 트레이드오프 관계에 있습니다. 작은 데이터셋에 대한 쿼리의 경우 Parquet의 I/O 효율성이 줄어듭니다. 이런 경우 Parquet는 일반 텍스트와 같거나 느리게 수행될 수 있습니다. 대기 시간, 압축률, 사용자 경험, 처리비용을 고려하여야 하며, 대부분의 경우에서 Parquet 타입이 가장 효율이 좋습니다.

대규모 집계 워크로드를 대상으로 어떻게 Spectrum을 어떻게 활용하는지 알아보겠습니다. 전체 37억 개 데이터셋을 대상으로 집계하는 쿼리를 Redshift 클러스터와 비교하여 살펴보겠습니다.

이러한 대규모 집계 워크로드에서 Spectrum이 수행하는 방식을 설명하기 위해 Redshift Spectrum에서 전체 37억 개 이상의 레코드 데이터 세트를 집계하고 Redshift에서만 쿼리를 실행하는 것과 비교하는 기본 쿼리를 살펴보겠습니다.

```
-- Parquet
SELECT uv.custKey, COUNT(uv.custKey)
FROM clickstream.uservisits_parquet1 as uv
GROUP BY uv.custKey
ORDER BY uv.custKey asc
```



Redshift 전용 테스트 케이스는 시간관계상 생략

시간적인 제약으로 직접 실습은 하지 않고 결과를 가지고 기능을 이해하도록 하겠습니다.

- Redshift 전용 테스트 케이스의 경우 클릭스트림 데이터가 로드되고 Redshift의 ANALYZE 명령으로 규정된 최적의 컬럼 압축 인코딩을 사용하여 모든 노드에 고르게 배포됩니다.
- Redshift Spectrum 테스트의 경우 특정 고객을 기준으로 하나의 파일에 한 달의 데이터가 Parquet 형식으로 구성되어 있습니다. 대부분 220-280MB 범위의 파일로 구성되어 있습니다. 제공된 다른 데이터셋으로 테스트를 실행하면 이 데이터 형식과 크기가 최적임을 알수있고, 다른 데이터셋보다 60배 이상 성능이 우수함을 알 수 있습니다.

조건절 푸시다운(Predicate Pushdown)

이전 단락에서 Spectrum이 대규모 집계를 수행하는 데 탁월하다는 것을 배웠습니다. 이번에는 Redshift Spectrum으로 더 많은 작업을 푸시다운한 경우를 테스트합니다.

1. 다음 쿼리를 실행합니다. 이 쿼리를 몇 번 실행하고 4초 범위의 실행 시간이 경과되는 것을 확인합니다.

```
SELECT c.c_name, c.c_mktsegment, t.prettyMonthYear, uv.totalRevenue
FROM (
    SELECT customer, visitYearMonth, SUM(adRevenue) as totalRevenue
    FROM clickstream.uservisits_parquet1
    WHERE customer <= 3 and visitYearMonth >= 199810
    GROUP BY customer, visitYearMonth) as uv
RIGHT OUTER JOIN customer as c ON c.c_custkey = uv.customer
INNER JOIN (
    SELECT DISTINCT d.yearnmonthnum, (d.month||'-'||d.year) as prettyMonthYear
    FROM dwdate WHERE d.yearnmonthnum >= 199810) as t ON uv.visitYearMonth = t.d.yearnmonthnum
ORDER BY c.c_name, c.c_mktsegment, uv.visitYearMonth ASC;
```

위 쿼리는 두 가지 측면에서 이전 쿼리보다 성능이 개선되었습니다.

- *clickstream.uservisits_csv10* 대신 *clickstream.uservisits_parquet1* 테이블을 쿼리하고 있습니다. 이 두 테이블은 동일한 데이터 세트를 포함하지만 다른 방식으로 처리됩니다. *clickstream.uservisits_parquet1* 테이블에는 Parquet 형식의 데이터가 포함되어 있습니다. Parquet는 컬럼 형식이며 쿼리에서 선택한 속성의 압축과 효율적인 검색이 가능하도록하여 분석 워크로드에 대한 I/O 이점을

제공합니다. 또한 "1"과 "10" 접미사가 의미하는 것은 CSV 타입은 10개의 파일로 분할되어 있고 Parquet는 1개의 파일에 모든 데이터가 들어가 있다는 것을 의미합니다. 그래서 대규모 스캔 및 집계 처리에 대한 오버헤드가 적습니다.

Amazon S3 > redshift-immersionday-labs > data/ > clickstream/ > uservisits_csv10/ > customer=1/ > visitYearMonth=199201/

S3 URI 복사

객체 속성

객체 (10)

객체는 Amazon S3에 저장되어 있는 기본 엔터티입니다. Amazon S3 인벤토리 를 사용하여 버킷에 있는 모든 객체의 목록을 얻을 수 있습니다. 다른 사용자가 객체에 액세스할 수 있게 하려면 명시적으로 권한을 부여해야 합니다. 자세히 알아보기 [?]

C S3 URI 복사 URL 복사 다운로드 열기 삭제 작업 폴더 만들기 업로드

검색어로 객체 찾기 버전 표시 < 1 > ⌂

선택	이름	유형	마지막 수정	크기	스토리지 클래스
	part-00000-64803196-a9da-49c8-ad41-cd4838b9f56a.csv	csv	2019. 6. 26. pm 2:45:22 PM KST	200.0MB	Standard
	part-00001-64803196-a9da-49c8-ad41-cd4838b9f56a.csv	csv	2019. 6. 26. pm 2:45:25 PM KST	154.1MB	Standard
	part-00002-64803196-a9da-49c8-ad41-cd4838b9f56a.csv	csv	2019. 6. 26. pm 2:45:24 PM KST	194.0MB	Standard
	part-00003-64803196-a9da-49c8-ad41-cd4838b9f56a.csv	csv	2019. 6. 26. pm 2:45:23 PM KST	186.7MB	Standard
	part-00004-64803196-a9da-49c8-ad41-cd4838b9f56a.csv	csv	2019. 6. 26. pm 2:45:26 PM KST	193.3MB	Standard
	part-00005-64803196-a9da-49c8-ad41-cd4838b9f56a.csv	csv	2019. 6. 26. pm 2:45:27 PM KST	186.5MB	Standard
	part-00006-64803196-a9da-49c8-ad41-cd4838b9f56a.csv	csv	2019. 6. 26. pm 2:45:30 PM KST	193.2MB	Standard
	part-00007-64803196-a9da-49c8-ad41-cd4838b9f56a.csv	csv	2019. 6. 26. pm 2:45:31 PM KST	186.5MB	Standard
	part-00008-64803196-a9da-49c8-ad41-cd4838b9f56a.csv	csv	2019. 6. 26. pm 2:45:28 PM KST	193.2MB	Standard
	part-00009-64803196-a9da-49c8-ad41-cd4838b9f56a.csv	csv	2019. 6. 26. pm 2:45:32 PM KST	186.5MB	Standard

Amazon S3 > redshift-immersionday-labs > data/ > clickstream/ > uservisits_parquet1/ > customer=1/ > visitYearMonth=199201/

S3 URI 복사

객체 속성

객체 (1)

객체는 Amazon S3에 저장되어 있는 기본 엔터티입니다. Amazon S3 인벤토리 를 사용하여 버킷에 있는 모든 객체의 목록을 얻을 수 있습니다. 다른 사용자가 객체에 액세스할 수 있게 하려면 명시적으로 권한을 부여해야 합니다. 자세히 알아보기 [?]

C S3 URI 복사 URL 복사 다운로드 열기 삭제 작업 폴더 만들기 업로드

검색어로 객체 찾기 버전 표시 < 1 > ⌂

선택	이름	유형	마지막 수정	크기	스토리지 클래스
	part-00000-87b0f6d1-5b32-4c5f-9c8b-302125b7984f.snappy.parquet	parquet	2019. 6. 26. pm 3:53:05 PM KST	354.2MB	Standard

- 집계 작업이 Redshift Spectrum으로 푸시다운되었습니다. 이전에 쿼리 계획을 분석했을 때 Spectrum이 스캐닝에 사용되는 것을 확인했습니다. 위의 쿼리를 분석하면 Spectrum 레이어에서도 집계가 수행되는 것을 볼 수 있습니다.

2. SVL_S3QUERY_SUMMARY 쿼리를 재실행합니다.

```
select elapsed, s3_scanned_rows, s3_scanned_bytes,
       s3query_returned_rows, s3query_returned_bytes, files, avg_request_parallelism
  from svl_s3query_summary
 where query = pg_last_query_id()
  order by query, segment;
```

elapsed	s3_scanned_rows	s3_scanned_bytes	s3query_returned_rows	s3query_returned_bytes	files	avg_request_parallelism
1030079	66270117	532080630	18	144	9	1.5

통계 데이터는 성능 향상의 원인을 나타냅니다.

- 압축한 결과 같은 수의 행을 스캔하더라도 스캔된 바이트는 줄어듭니다.
- 반환되는 행 수는 66.3M에서 18로 줄어듭니다. 이 결과 494MB가 아닌 Spectrum 레이어에서 144 Byte만 반환됩니다. 집계 연산을 Spectrum 레이어로 푸시한 결과입니다. 데이터는 일 수준 단위로 저장되고 쿼리는 월 단위로 롤업합니다. 집계 연산을 Spectrum 플랫으로 푸시다운하면 제시된 Dimension 속성과 결합할 수 있도록 최대 월 수준까지 광고 수익을 집계하는데 18개의 레코드만 반환하면 됩니다.

3. EXPLAIN을 사용하여 쿼리를 다시 실행합니다.

실행계획에는 Spectrum 계층이 집계 처리하고 있음을 나타내는 S3 Aggregate 단계가 포함되어 있습니다.

```
XN Merge (cost=1000250091325.76..1000250091588.26 rows=105000 width=590)
  Merge Key: c.c_name, c.c_mktsegment, uv.visityearmonth
    -> XN Network (cost=1000250091325.76..1000250091588.26 rows=105000 width=590)
      Send to leader
    -> XN Sort (cost=1000250091325.76..1000250091588.26 rows=105000 width=590)
      Sort Key: c.c_name, c.c_mktsegment, uv.visityearmonth
        -> XN Hash Join DS_DIST_ALL_NONE (cost=250039517.84..250002568.75 rows=105000 width=590)
          Hash Cond: ("outer".customer)::text = ("inner".c_custkey)::text
        -> XN Hash Join DS_DIST_ALL_NONE (cost=250002017.84..250002018.72 rows=7 width=1072)
          Hash Cond: ("outer".visityearmonth)::text = ("inner".d_yearmonthnum)::text
            -> XN Subquery Scan uv (cost=250001985.00..250001985.29 rows=23 width=1040)
              -> XN HashAggregate (cost=250001985.00..250001985.06 rows=23 width=1040)
                -> XN Partition Loop (cost=25000000.00..250001145.00 rows=112000 width=1040)
                  -> XN Seq Scan PartitionInfo of clickstream.uservisits_parquet1 (cost=0.00..15.00 rows=1
                    Filter: (((customer)::text < '3'::text) AND ((visityearmonth)::text >= '199810'::te
                  -> XN S3 Query Scan uservisits_parquet1 (cost=125000000.00..125000010.00 rows=1000 width
                    -> S3 Aggregate (cost=125000000.00..125000000.00 rows=1000 width=8)
                      -> S3 Seq Scan clickstream.uservisits_parquet1 location:"s3://redshift-immers
                -> XN Hash (cost=32.82..32.82 rows=7 width=36)
                  -> XN Subquery Scan t (cost=0.00..32.82 rows=7 width=36)
                    -> XN Unique (cost=0.00..32.75 rows=7 width=18)
                      -> XN Seq Scan on dwdate (cost=0.00..32.43 rows=64 width=18)
                        Filter: (d_yearmonthnum >= 199810)
                -> XN Hash (cost=30000.00..30000.00 rows=3000000 width=38)
                  -> XN Seq Scan on customer c (cost=0.00..30000.00 rows=3000000 width=38)
```

Redshift Spectrum Request Accelerator

Redshift Spectrum Request Accelerator(SRA)는 Amazon S3의 데이터에 대한 쿼리 성능을 크게 향상시킵니다.

1. 1992년 데이터에 대해 다음 쿼리를 실행합니다. 이 쿼리는 10초 이내에 실행되어야 합니다. 1993년에 대해 필터를 수정하고 다시 실행합니다.

```
SELECT c.c_name, c.c_mktsegment, uv.visitYearMonth, uv.totalRevenue
FROM (
  SELECT customer, visitYearMonth, SUM(adRevenue) as totalRevenue
  FROM clickstream.uservisits_parquet1
  WHERE customer <= 3 and visitYearMonth between 199201 and 199212
  GROUP BY customer, visitYearMonth) as uv
RIGHT OUTER JOIN customer as c ON c.c_custkey = uv.customer
ORDER BY c.c_name, c.c_mktsegment, uv.visitYearMonth ASC;
```

2. 이제 1992년부터 1994년까지 3년 동안의 데이터에 대해 쿼리를 실행해 보겠습니다. 이 쿼리는 약 3배 더 많은 데이터에 액세스하지만 시간은 3로 증가하지 않았습니다. 이는 SRA가 1992년 및 1993년에 대한 스펙트럼 결과를 캐시하기 때문입니다. 이 쿼리는 다시 처리하는 대신 이를 재사용합니다.
3. 쿼리를 실행하여 SVL_S3QUERY_SUMMARY를 다시 수행합니다.

```
select query, starttime, elapsed, s3_scanned_rows, s3_scanned_bytes,
       s3query_returned_rows, s3query_returned_bytes, files, avg_request_parallelism
from svl_s3query_summary
order by query desc, segment;
```

결과적으로 11억건의 레코드가 스캔되고 있음을 확인할 수 있습니다.

query	starttime	elapsed	s3_scanned_rows	s3_scanned_bytes	s3query_returned_rows	s3query_returned_bytes	files	avg_request_parallelism
4361	2022-02-07 10:33:53.997701	6051539	835600294	6709009150	227	31327	108	8.1
4346	2022-02-07 10:33:04.396457	10770651	265426602	2131090671	72	1224	36	5
4335	2022-02-07 10:31:56.150515	9130688	304772150	2447026849	83	1411	36	5.6

처리할 데이터는 많지만 쿼리는 10초 이내에 실행되는 것을 볼 수 있습니다. 이는 동일한 데이터셋을 대상으로 처음에 실행했던 쿼리보다 훨씬 빠릅니다. 쿼리 성능은 데이터 형식, 크기를 개선하고 집계 작업을 Spectrum 레이어로 푸시한 결과 크게 향상되었습니다.

Native Redshift vs Redshift with Spectrum

그럼 무조건 Spectrum을 사용해야 할까요? 하지만 Native Redshift를 독립적으로 실행할 때 더 빠른 결과를 내는 경우도 많이 있습니다. 시간 관계상 마지막 수행했던 쿼리를 대상으로 비교 테스트 결과를 검토해 보겠습니다.

일반적으로 I/O 수행이 적고 다양한 조인 워크로드가 있을 경우 Native Redshift가 더 성능이 좋습니다. 또한 Native Redshift의 경우 일정하게 쿼리 성능을 유지합니다. 그러므로 쿼리 결과에 대해서 엄격한 SLA가 적용되는 사례에서는 독립적으로 Redshift를 사용하는 것이 좋습니다.

반면에 대규모 데이터셋을 대상으로 스캔을 수행해야 할 경우 두 장점을 모두 활용할 수 있습니다. 즉, 낮은 비용으로 최적의 성능을 얻을 수 있습니다. 예를 들면, 비즈니스 분석자가 방대한 양의 과거 데이터를 활용하여 즉각적으로 인사이트를 도출하는 경우가 대상 사례가 될 것입니다.

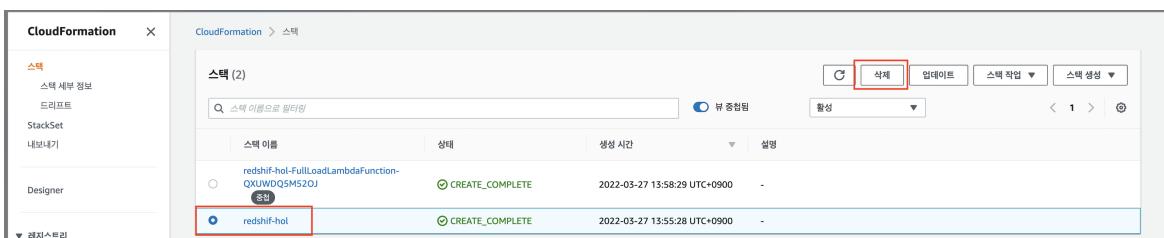
#. 실습 리소스 정리

CloudFormation 스택 삭제

1. CloudFormation 이동

- console.aws.amazon.com/cloudformation

2. 스택선택 후 삭제 버튼 클릭



3. 모든 스택 리소스 삭제 확인



