

**SeUng Jung**

**Data Life Cycle on AWS**

*Lab2-3. Data Transforming with Glue*

**Table of Contents**

Lab 설명 4

Lab Architecture 5

Data Transforming with Glue 6

# Lab 설명

이번 Lab은 Tier 1 S3 bucket에 저장된 Raw 데이터를 다음 단계의 Tier 2 S3 Analytics버킷으로 이동시키고, 그 과정에서 분석에 활용하기 유용한 형태로 파티셔닝 하고, 파일 포멧을 변환(JSON -> parquet)하여 Fact 테이블을 생성하는 작업을 수행합니다.

기본적으로 서로 다른 Source(RDS, DDB, Kinesis)로 부터 S3 Raw버킷에 저장된 서로 다른 데이터를 통합하고, 정리하는 과정입니다. 최종 목표 뷰는 S3 Analytics 버킷에 분석에 필요한 형태의 데이터가 저장되는 모습입니다.

# Lab Architecture



단계별 상세 Lab은 다음과 같습니다.

Lab 2-1. RDS to S3 with Glue

Lab 2-2. DDB to S3

**Lab 2-3. Data Transforming with Glue**

Lab 2-4. ETL to Redshift with Glue

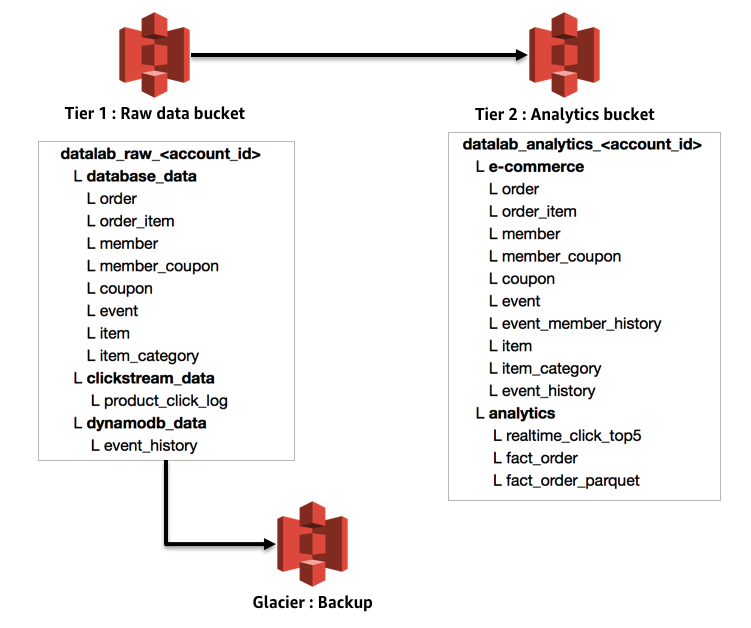
Lab 2-5. Glue Data Catalog for Analytics

Lab 2-6. Query with Athena and Redshift spectrum

# Data Transforming with Glue

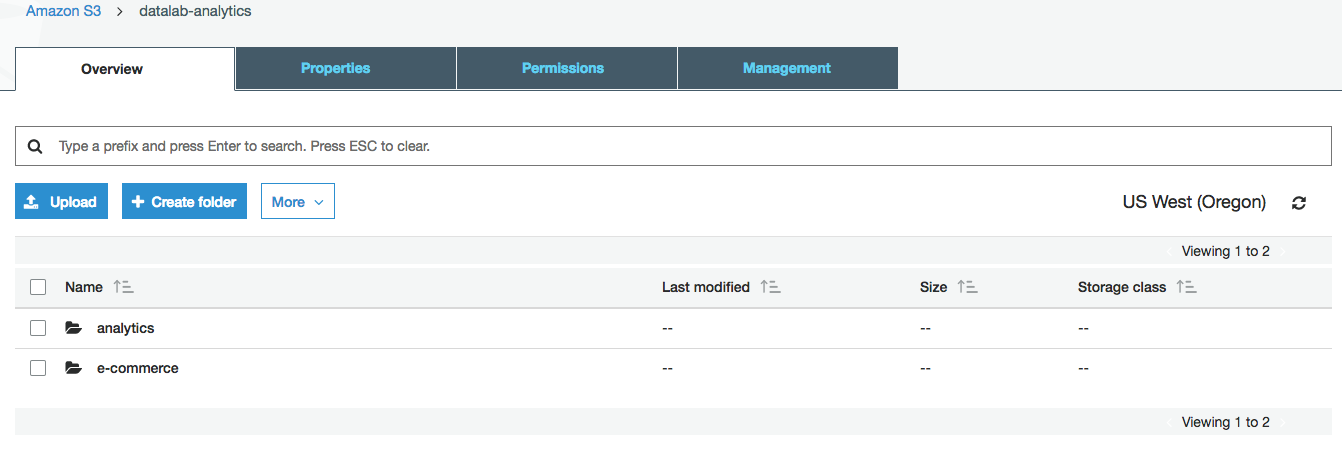
이전 Lab을 통해 데이터는 Aurora, DDB로 부터 S3 버킷에 저장이 되었습니다. 해당 데이터를 분석을 위해 다음과 같이 변환 / 이동 하는 작업을 수행하도록 하겠습니다.

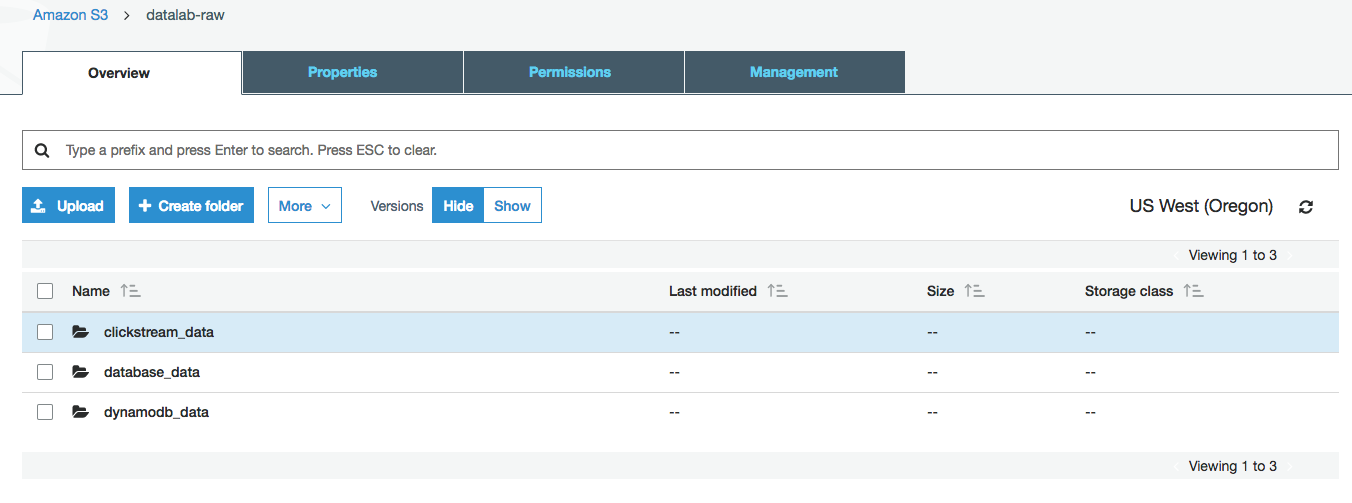
그림에서 보는 것과 같이 테이블의 이름은 소스버킷에서는 데이터의 출처에 따라, 분석 버킷에서는 비지니스 관점에 따라 변경되는 것을 알 수 있습니다.



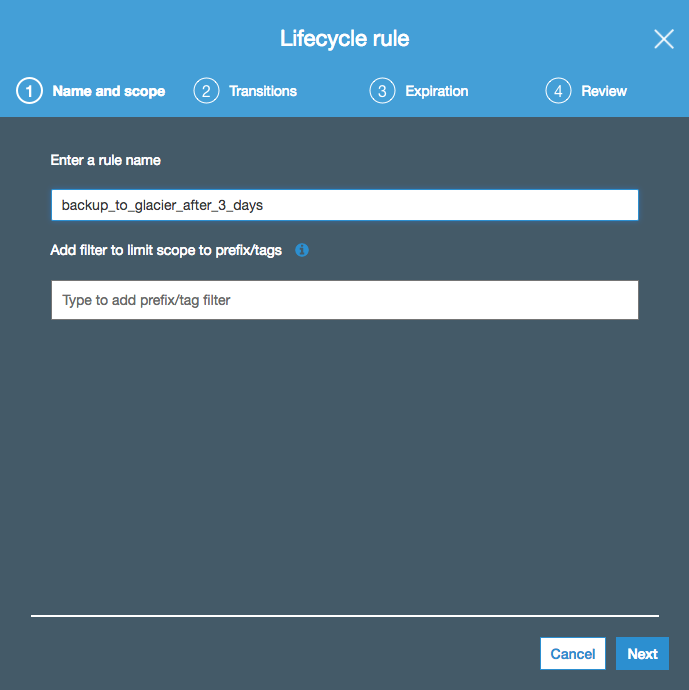
1. 우측 상단의 리전을 **Oregon (US-WEST)**으로 선택합니다.
2. AWS Management Console에서 S3 화면으로 이동하여 데이터 이동의 타겟 버킷 **‘datalab-analytics-[account\_id]’**이 생성되어 있는지 확인합니다 .

앞쪽에서 적절히 폴더를 생성하였으면 다음과 같이 서브 폴더가 확인 됩니다.

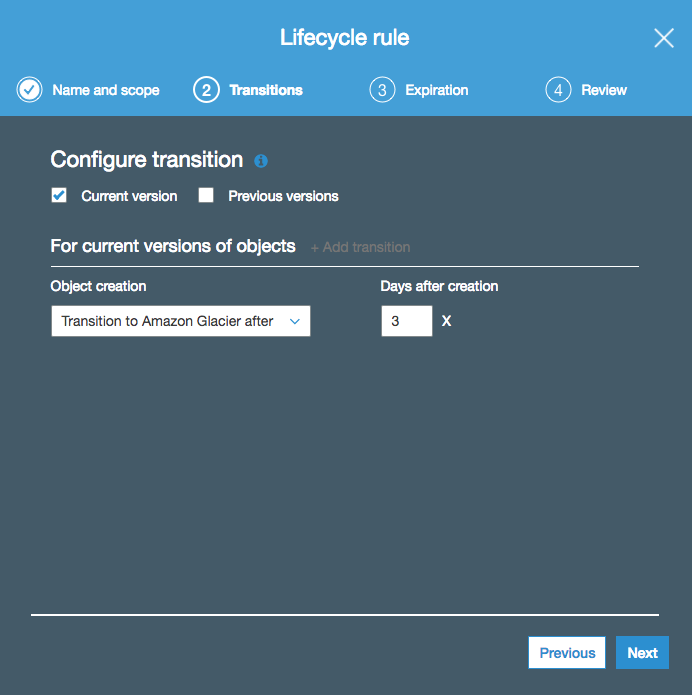




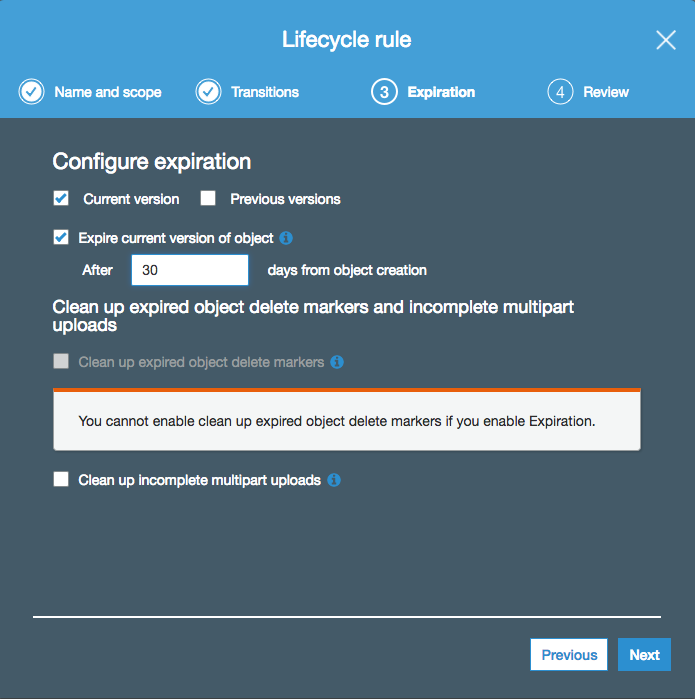
1. 다음으로는 **‘datalab-raw-[account-id]’** 버킷(원본데이터가 있는 버킷)에 Lifecycle를 설정하여 데이터를 주기적으로 Glacier로 이동시키도록 하겠습니다.
2. 해당 버킷을 선택하고**[Management]** 탭으로 이동합니다. **[Add lifecycle]** rule를 선택하고 이름과 설정을 입력합니다. (만약 같은 이름이 존재하면 삭제하고 다시 만들거나 다른 이름으로 만들어도 좋습니다.)



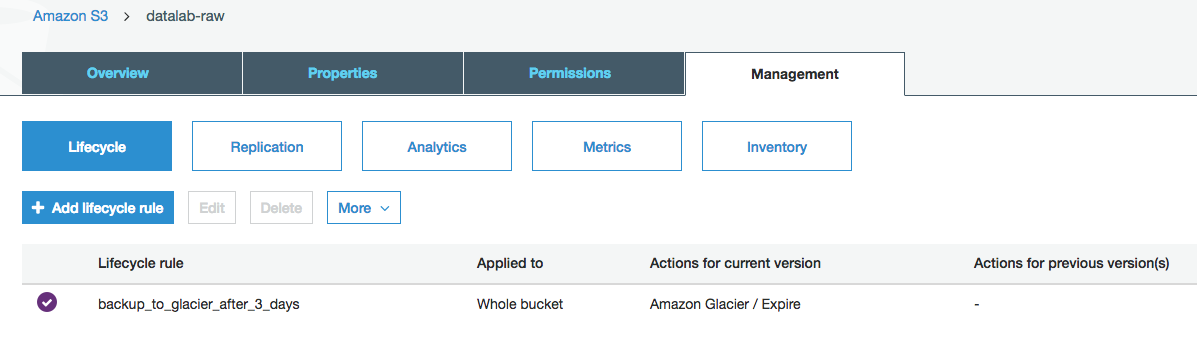
1. **[Current version]** 을 체크하고, 아래쪽에 **[+Add transition]**을 선택합니다.



1. 추가로 파일의 만료기간도 설정할 수 있습니다.

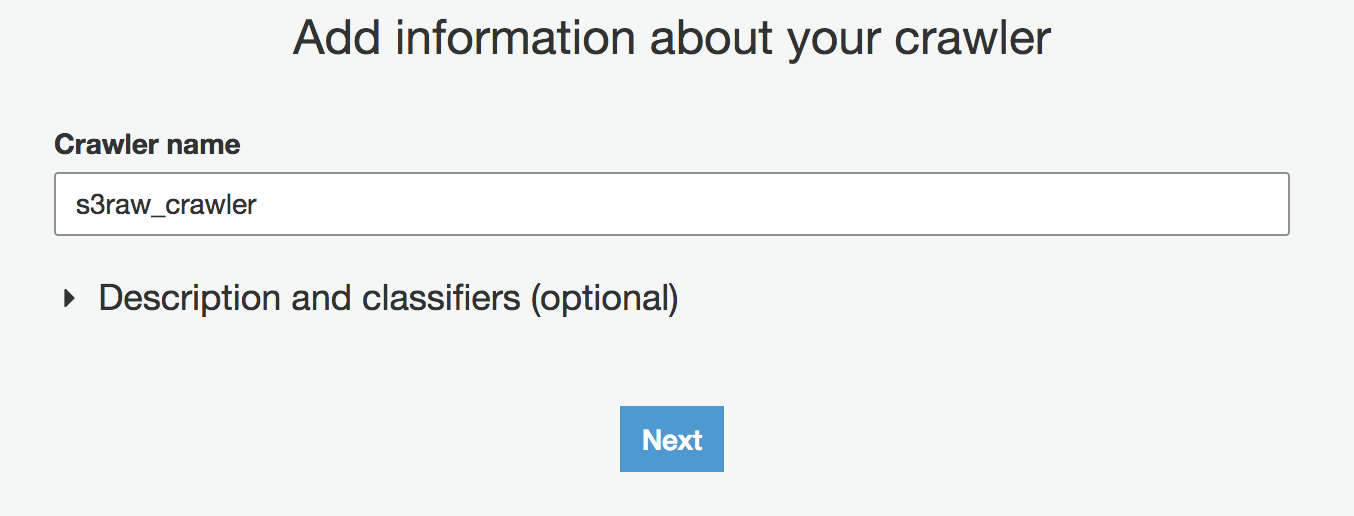


1. 완료하면 다음처럼 Lifecycle 설정이 완료됩니다.

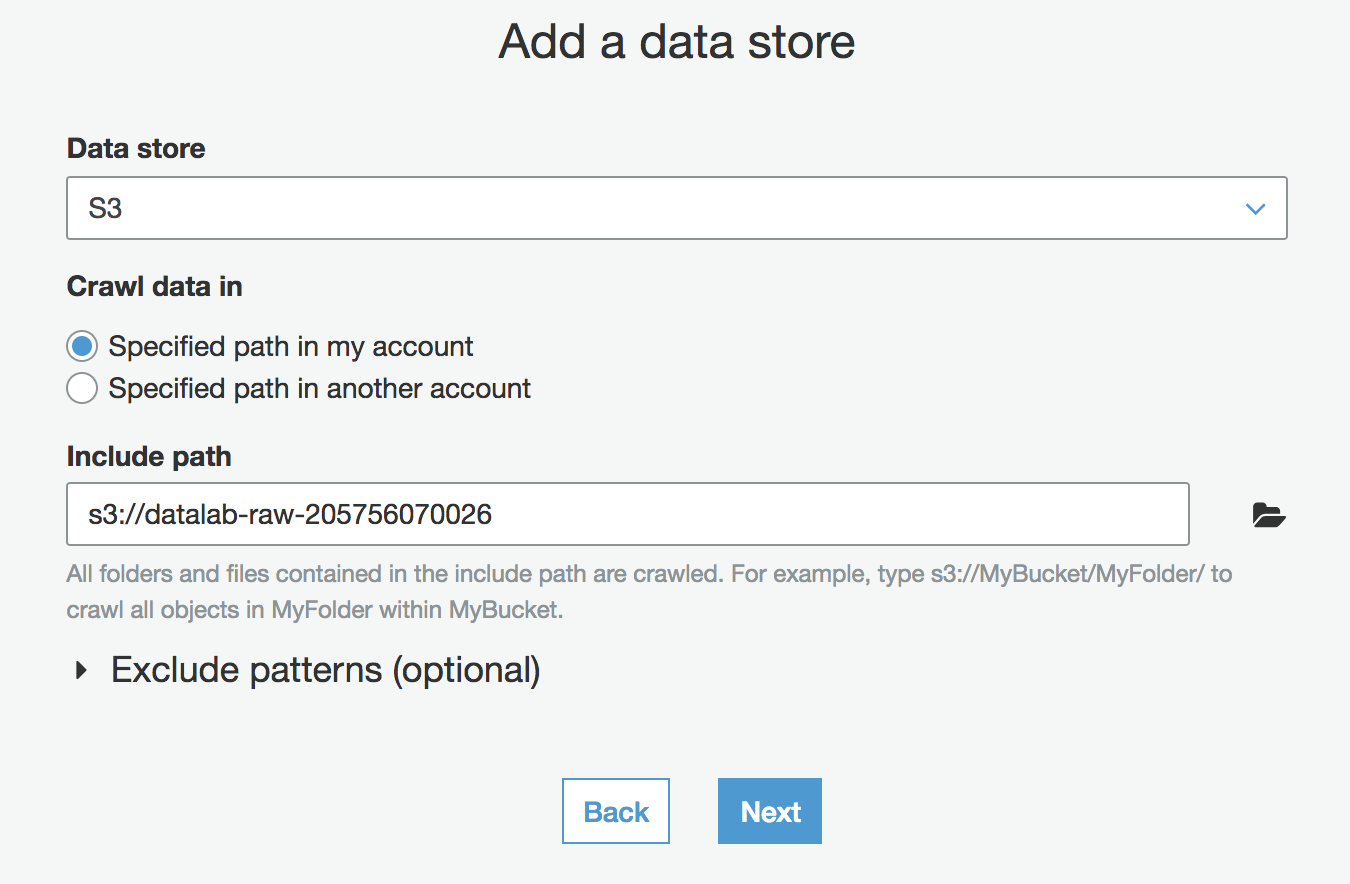


1. 이제 본격적으로 데이터 변환 작업을 시작합니다 .
2. **‘datalab-raw-[account\_id]’** 버킷에 저장된 데이터에 대한 카탈로그를 생성해야 합니다.
3. Console에서 Glue 서비스로 이동하여 Add Crawler을 선택하여 새로운 크롤러를 생성합니다.

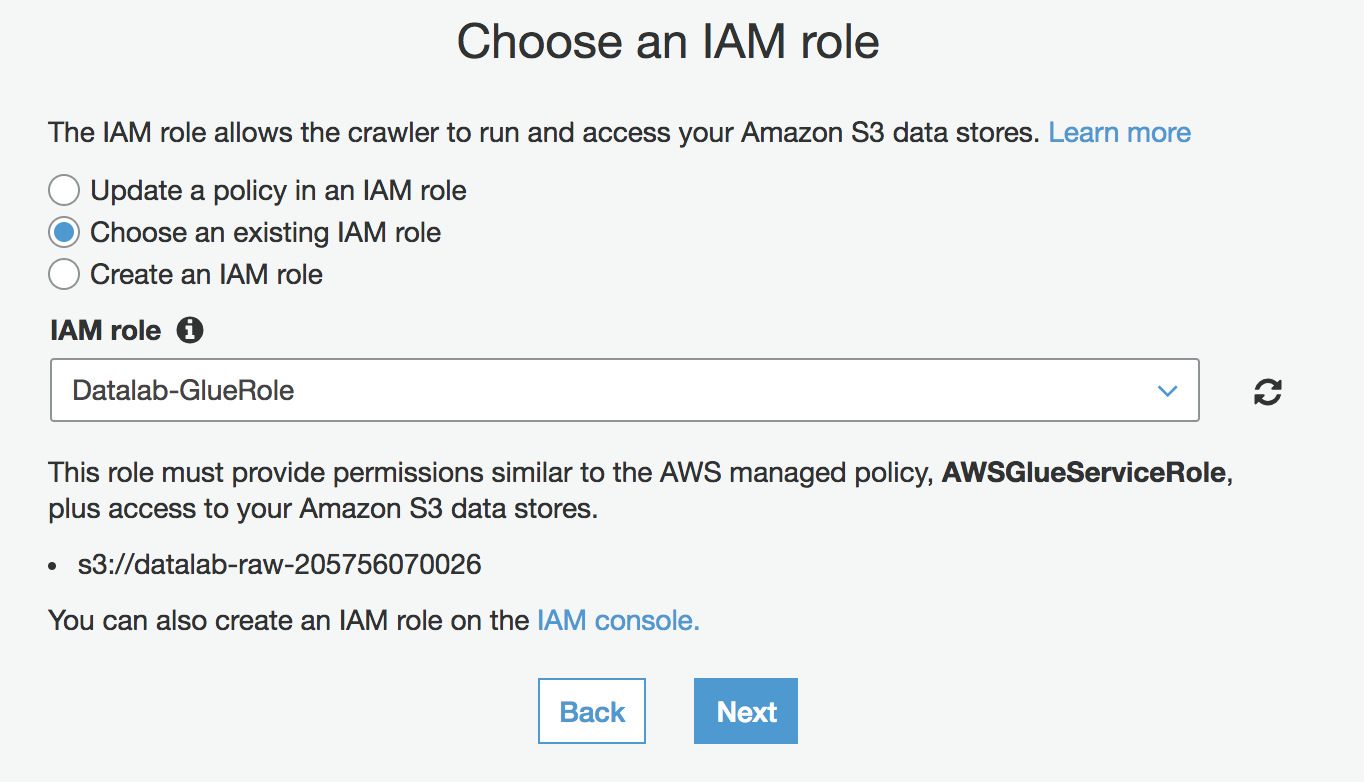
Crawler name : s3raw-crawler



1. 해당 폴더 아래에는 여러개의 폴더가 존재할 것이나 모두 데이터 소스로 사용하기 위해 최상위 버킷을 Inclued path로 지정합니다.
2. Data store 는 **S3**를 선택하고 Include path 에 자신의 raw data 버킷을 선택합니다.

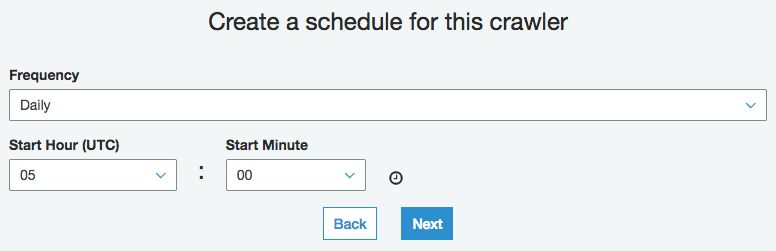


1. IAM Role에서 [Choose an existing IAM role]을 고르,고 **[Datalab-GlueRole]**을 선택하고 [Next] 를 클릭합니다.

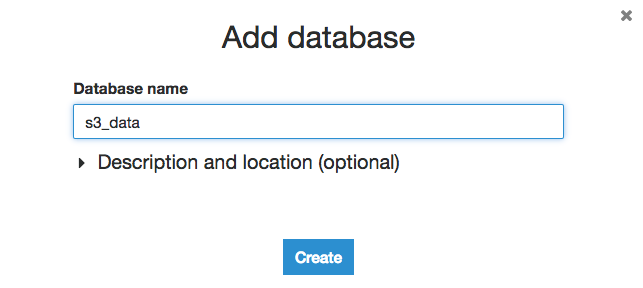


1. 해당 Crawler 부터는 데이터의 변동을 고려하여 일단위 스케줄링을 설정합니다.

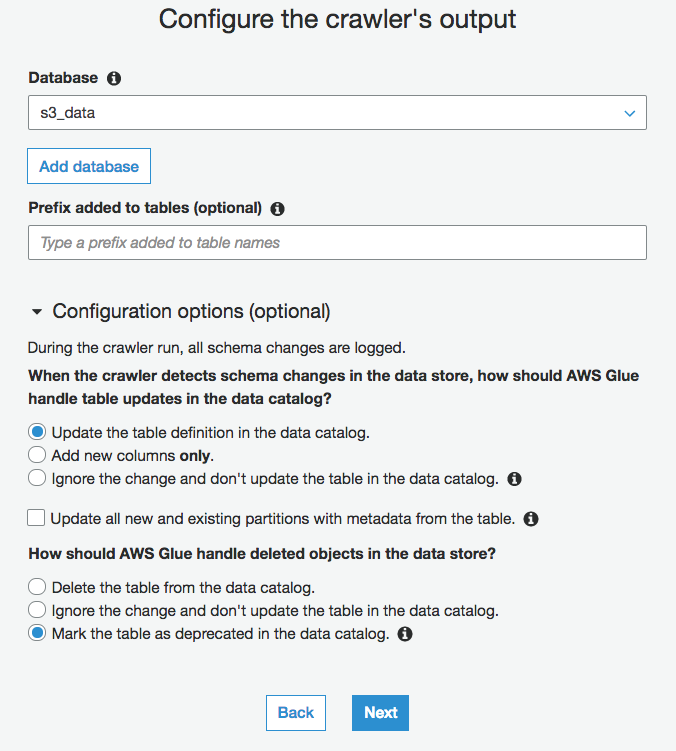
Frequency는 **[Daily]**로 선택하고 **[Next]** 클릭



1. 타겟 Database는 이후 코드와의 호환을 위해서 반드시 “s3\_data”로 입력하시기 바랍니다. **[Add database]** 버튼을 클릭하여 생성합니다.



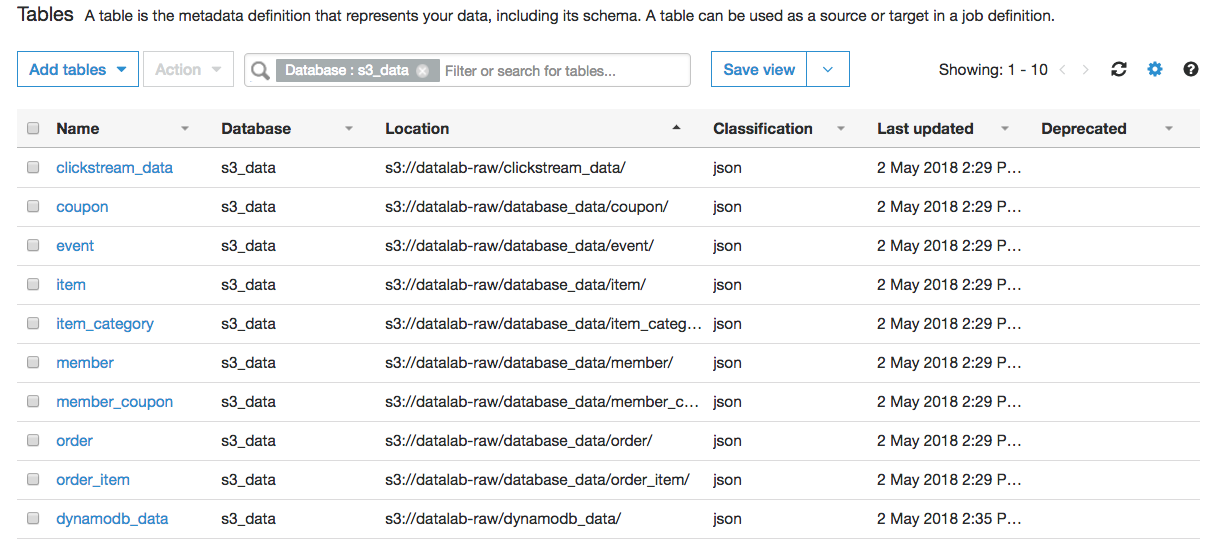
1. 아래쪽의 Configuration options는 필요에 따라 수정할 수 있으나 디폴트 값을 사용합니다. 데이터 스키마가 지속적으로 변경되는 경우에는 아래 값을 세부적으로 설정하셔야 합니다.



1. 만들어진 크롤러를 실행 시킵니다.



1. 몇분이 지나고, 크롤러 실행이 완료되면 해당 데이터베이스(s3\_data)의 테이블 리스트를 확인합니다.



1. 이로서 Aurora, DDB로 각각 수집된 데이터의 원본에 대한 데이터 카탈로그가 완성되었습니다.
2. 이제부터 해당 데이터를 가공하여 S3 Tier 2 Analytics 버킷으로 이동시켜 보겠습니다.
3. 새로운 Job을 생성합니다. 해당 작업은 하루 단위로 추가된 데이터만 변경하는 형태로 Job을 실행하기 위해서 Job Bookmark 기능을 사용합니다.

아래쪽 **Advanced Properties**에서 Job Bookmark를 **Enable**로 설정합니다. Role는 **Datalab-GlueRole**를 선택합니다.

Name : s3raw\_s3analytics

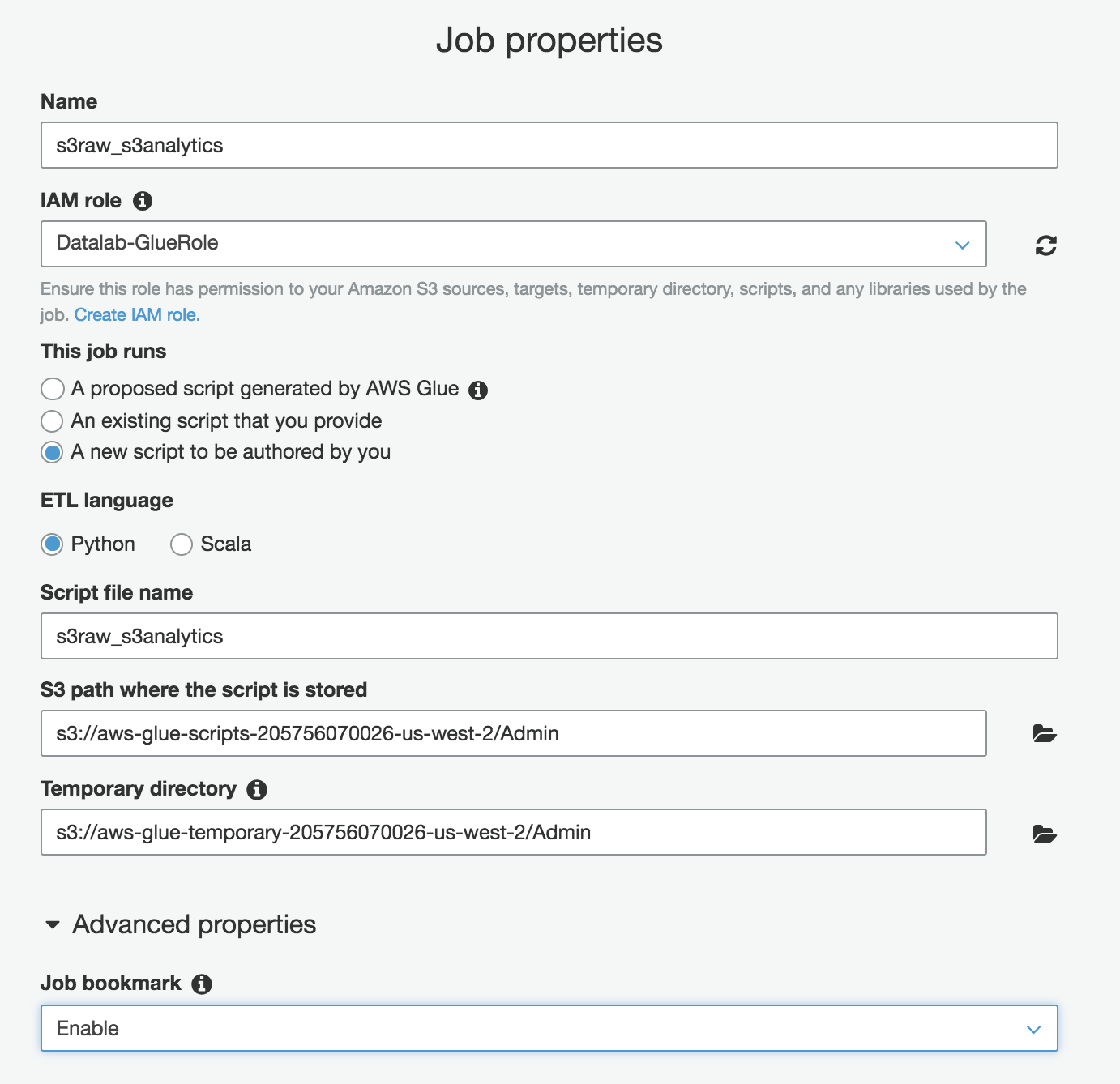
IAM role : Datalab-GlueRole

Type : Spark

This job runs : A new script to be authored by you

Advanced properties메뉴를 확장하여

Job bookmark : Enable

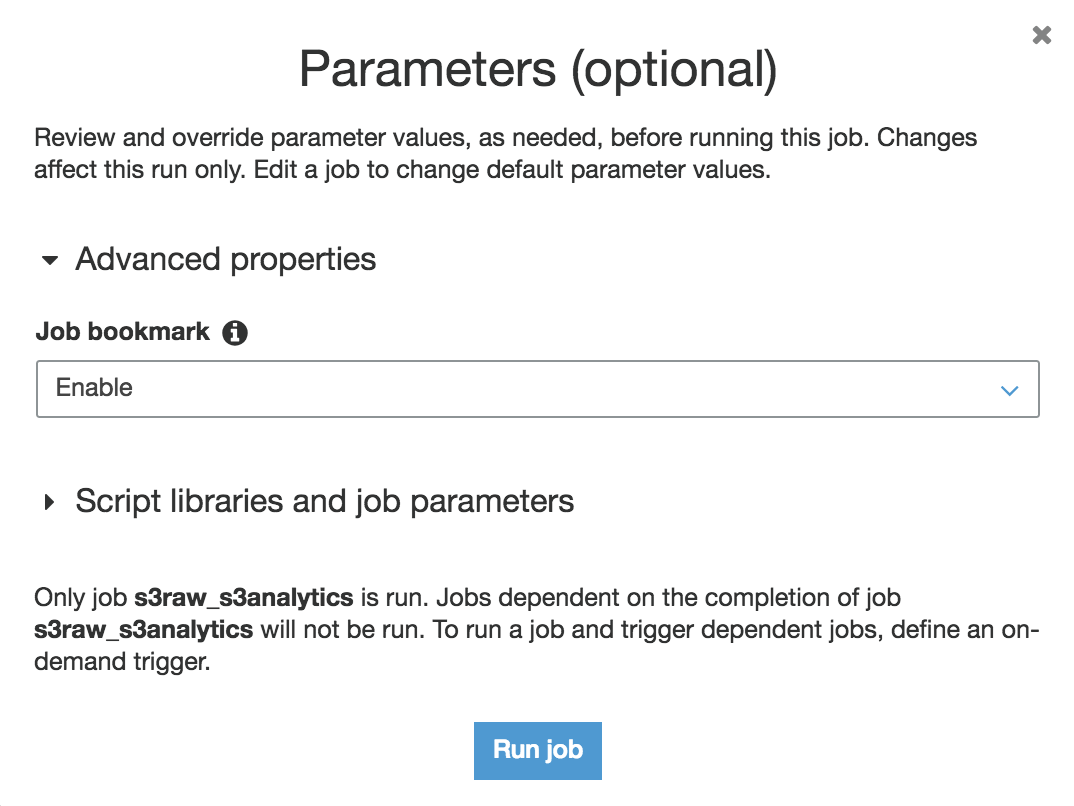


1. 아래 코드를 붙여넣기 합니다.

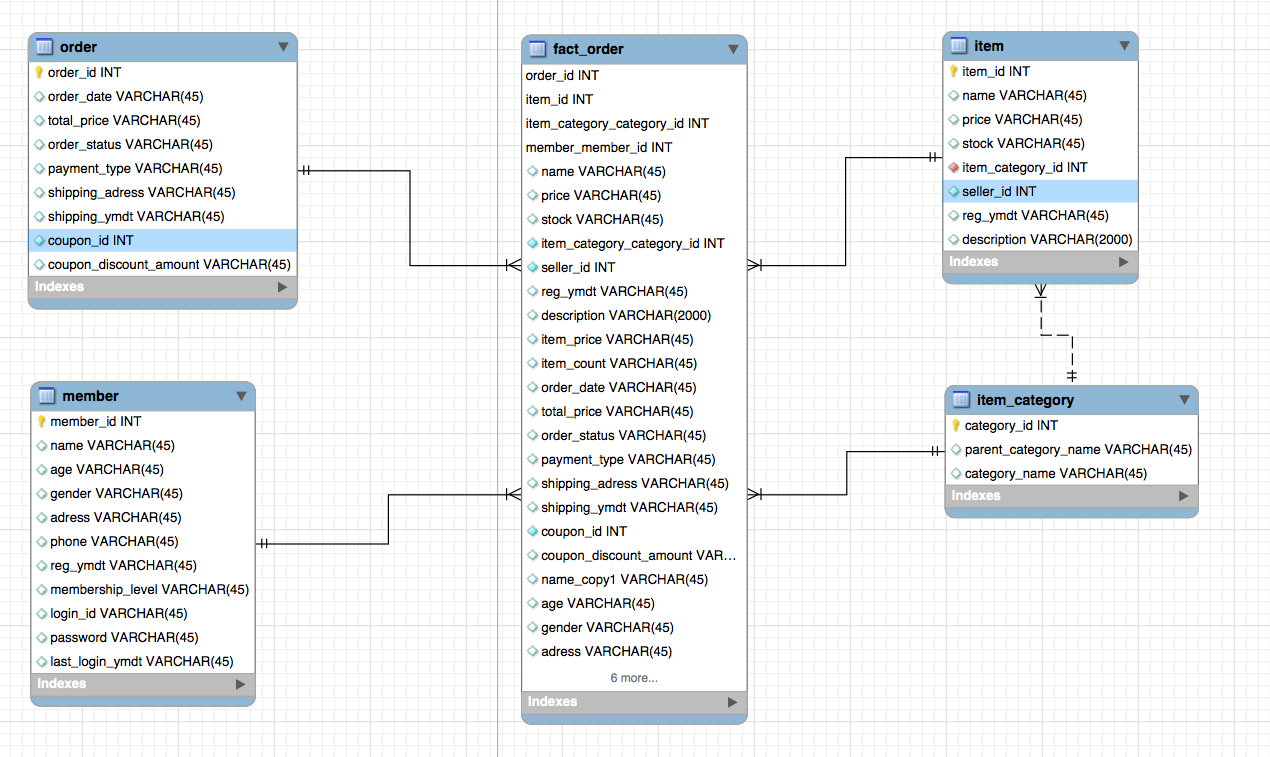
중요 : 반드시 최상단 **‘S3bucket’** 명칭을 자신의 버킷명과 일치 시켜야 합니다.

|  |
| --- |
| import sys  from awsglue.transforms import \*  from awsglue.utils import getResolvedOptions  from pyspark.context import SparkContext  from awsglue.context import GlueContext  from awsglue.job import Job  ## @params: [JOB\_NAME]  args = getResolvedOptions(sys.argv, ['JOB\_NAME'])  sc = SparkContext()  glueContext = GlueContext(sc)  spark = glueContext.spark\_session  job = Job(glueContext)  job.init(args['JOB\_NAME'], args)  s3Bucket = "s3://datalab-analytics-205756070026"  s3Folder ="/e-commerce/"  ## order  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "order", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("member\_id", "string", "member\_id", "string"), ("order\_date", "string", "order\_date", "string"), ("order\_status", "string", "order\_status", "string"), ("country", "string", "country", "string"), ("shipping\_date", "date", "shipping\_date", "date"), ("total\_price", "int", "total\_price", "int"), ("city", "string", "city", "string"), ("order\_time", "string", "order\_time", "string"), ("state", "string", "state", "string"), ("postal\_code", "string", "postal\_code", "string"), ("region", "string", "region", "string"), ("order\_id", "string", "order\_id", "string")], transformation\_ctx = "applymapping1")  datasink2 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping1, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "order", "partitionKeys" : ["order\_date"]}, format = "parquet", transformation\_ctx = "datasink2")  ## order\_item  datasource1 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "order\_item", transformation\_ctx = "datasource1")  applymapping2 = ApplyMapping.apply(frame = datasource1, mappings = [("item\_count", "int", "item\_count", "int"), ("order\_date", "string", "order\_date", "string"), ("item\_id", "string", "item\_id", "string"), ("item\_price", "int", "item\_price", "int"), ("order\_time", "string", "order\_time", "string"), ("order\_id", "string", "order\_id", "string")], transformation\_ctx = "applymapping2")  datasink3 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping2, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "order\_item", "partitionKeys" : ["order\_date"]}, format = "parquet", transformation\_ctx = "datasink3")  ## member  datasource2 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "member", transformation\_ctx = "datasource2")  applymapping3 = ApplyMapping.apply(frame = datasource2, mappings = [("member\_id", "string", "member\_id", "string"), ("country", "string", "country", "string"), ("login\_id", "string", "login\_id", "string"), ("gender", "string", "gender", "string"), ("city", "string", "city", "string"), ("last\_login\_ymdt", "string", "last\_login\_ymdt", "string"), ("membership\_level", "string", "membership\_level", "string"), ("login\_password", "string", "login\_password", "string"), ("name", "string", "name", "string"), ("state", "string", "state", "string"), ("postal\_code", "string", "postal\_code", "string"), ("region", "string", "region", "string"), ("age", "string", "age", "string"), ("reg\_ymdt", "string", "reg\_ymdt", "string")], transformation\_ctx = "applymapping3")  datasink4 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping3, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "member"}, format = "parquet", transformation\_ctx = "datasink4")  ## member\_coupon  datasource3 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "member\_coupon", transformation\_ctx = "datasource3")  applymapping4 = ApplyMapping.apply(frame = datasource3, mappings = [("member\_id", "string", "member\_id", "string"), ("coupon\_id", "int", "coupon\_id", "int")], transformation\_ctx = "applymapping4")  datasink5 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping4, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "member\_coupon"}, format = "parquet", transformation\_ctx = "datasink5")  ## event  datasource4 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "event", transformation\_ctx = "datasource4")  applymapping5 = ApplyMapping.apply(frame = datasource4, mappings = [("event\_id", "int", "event\_id", "int"), ("available\_count", "int", "available\_count", "int"), ("end\_ymdt", "string", "end\_ymdt", "string"), ("discount\_amount", "int", "discount\_amount", "int"), ("name", "string", "name", "string"), ("available\_yn", "string", "available\_yn", "string"), ("start\_ymdt", "string", "start\_ymdt", "string")], transformation\_ctx = "applymapping5")  datasink6 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping5, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "event"}, format = "parquet", transformation\_ctx = "datasink6")  ## item  datasource5 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "item", transformation\_ctx = "datasource5")  applymapping6 = ApplyMapping.apply(frame = datasource5, mappings = [("item\_id", "string", "item\_id", "string"), ("price", "int", "price", "int"), ("name", "string", "name", "string"), ("description", "string", "description", "string"), ("reg\_ymdt", "string", "reg\_ymdt", "string"), ("item\_category\_id", "int", "item\_category\_id", "int")], transformation\_ctx = "applymapping6")  datasink7 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping6, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "item"}, format = "parquet", transformation\_ctx = "datasink7")  ## item\_category  datasource6 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "item\_category", transformation\_ctx = "datasource6")  applymapping7 = ApplyMapping.apply(frame = datasource6, mappings = [("category\_name", "string", "category\_name", "string"), ("category\_id", "int", "category\_id", "int"), ("parent\_category\_name", "string", "parent\_category\_name", "string")], transformation\_ctx = "applymapping6")  datasink8 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping7, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "item\_category"}, format = "parquet", transformation\_ctx = "datasink8")  ## coupon  datasource7 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "coupon", transformation\_ctx = "datasource7")  applymapping8 = ApplyMapping.apply(frame = datasource7, mappings = [("coupon\_id", "int", "coupon\_id", "int"), ("end\_ymdt", "string", "end\_ymdt", "string"), ("discount\_amount", "int", "discount\_amount", "int"), ("name", "string", "name", "string"), ("start\_ymdt", "string", "start\_ymdt", "string")], transformation\_ctx = "applymapping8")  datasink9 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping8, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "coupon"}, format = "parquet", transformation\_ctx = "datasink9")  ## event\_history from ddb data  datasource10 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "dynamodb\_data", transformation\_ctx = "datasource10")  applymapping11 = ApplyMapping.apply(frame = datasource10, mappings = [("reg\_date", "string", "reg\_date", "string"), ("member\_event\_id", "string", "member\_event\_id", "string"), ("history\_seq", "string", "history\_seq", "string")], transformation\_ctx = "applymapping11")  datasink12 = glueContext.write\_dynamic\_frame.from\_options(frame = applymapping11, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "event\_history"}, format = "parquet", transformation\_ctx = "datasink12")  job.commit() |

1. 작업을 실행할때 북마크가 **Enable** 되어 있음을 확인합니다.



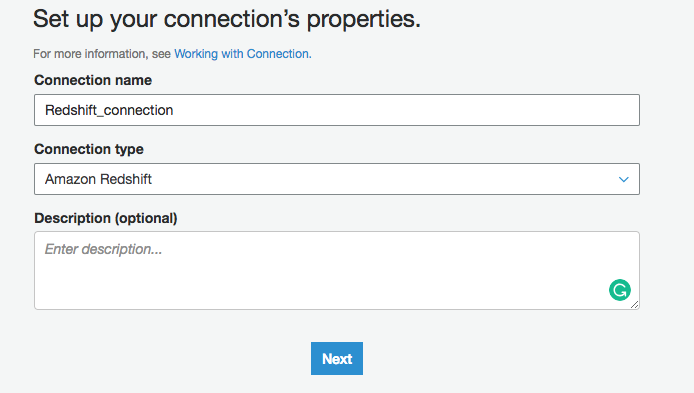
1. 다음은 분석작업을 위해서 사용될 Fact 테이블을 만듭니다. **order\_item** 테이블을 기준으로 **fact\_order** 테이블을 만듭니다. Dimension 테이블로 order, item, item\_category, member 테이블을 사용합니다.



1. 다음 작업을 위해서 새로운 Job을 생성합니다. 작업을 생성하기에 앞서 Redshift 데이터 적재를 위한 Connection을 먼저 만들고 진행합니다.
2. 좌측 메뉴의 **Database – Connections** 로 이동하여 **[Add connection]** 선택

Connection name : Redshift\_connection

Connection type : Amazon Redshift



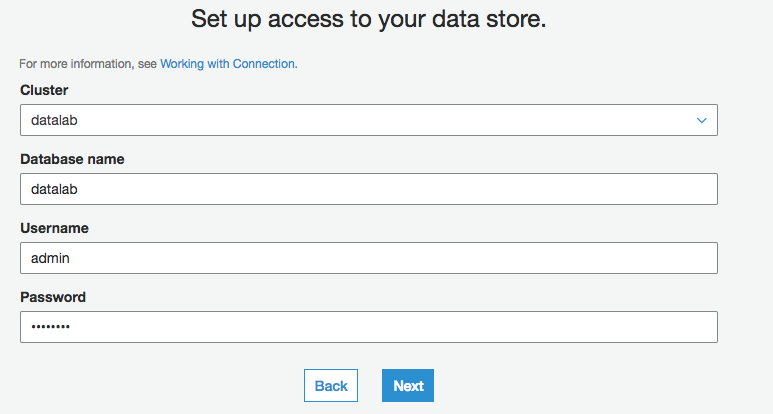
1. 다음과 같이 Redshift 접속정보를 적어줍니다.

Cluster : datalab

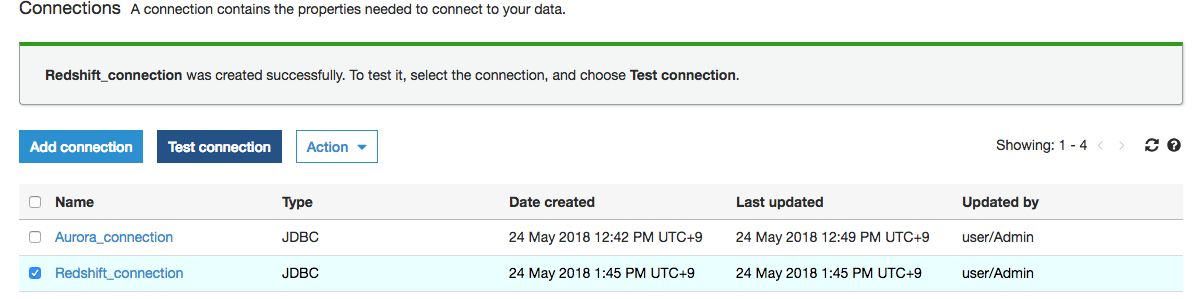
Database name : datalab

Username : admin

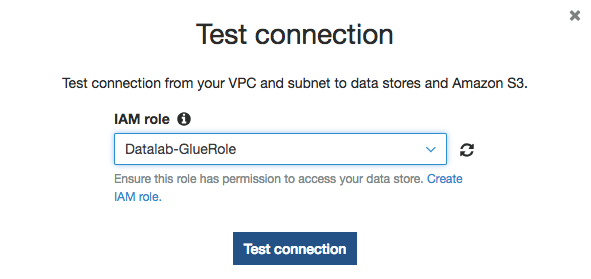
Password : passW0rd (W는 대문자 그다음 0은 숫자입니다.)



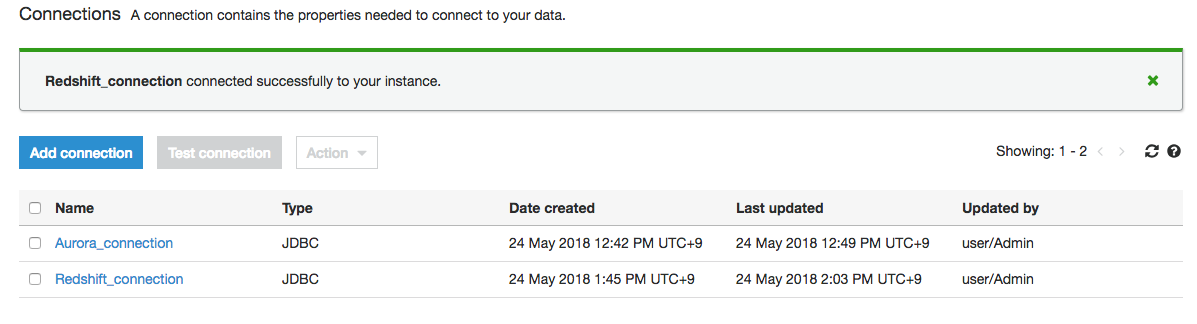
1. Connection 리스트에서 **[Redshift\_connection]** 을 선택하고 **[Test connection]** 누릅니다.



1. 작업을 위한 Role에서 **[Datalab-GlueRole]**를 선택합니다.



1. 접속 테스트가 성공하면 다음 단계로 넘어갈 수 있습니다.



1. 역시 일단위로 Fact 테이블을 증분 데이터에 대해 생성해 줘야하므로, Job Bookmark를 Enable 합니다.

Name : etl\_fact\_order

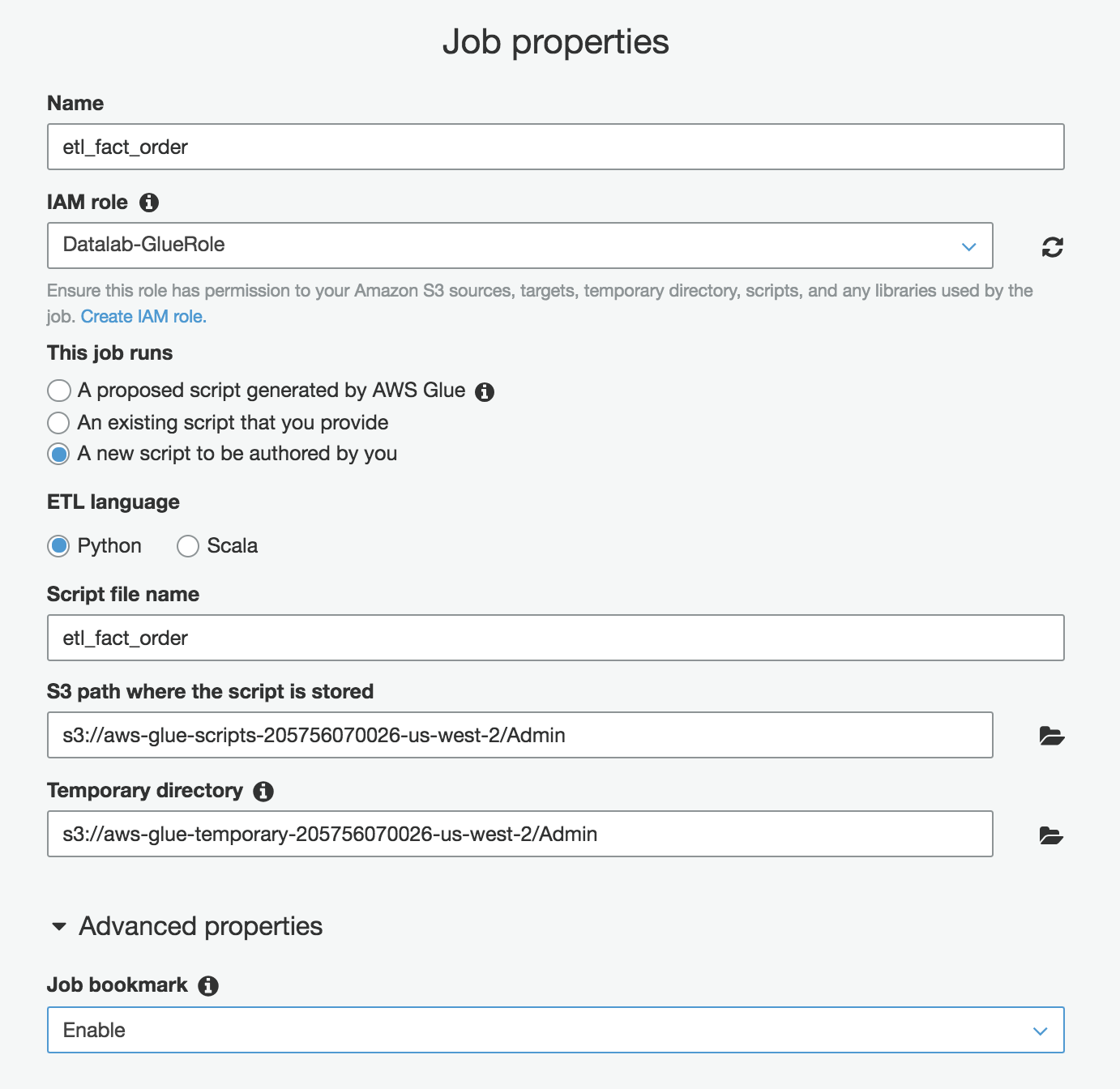
IAM Role : Datalab-GlueRole

Type : Spark

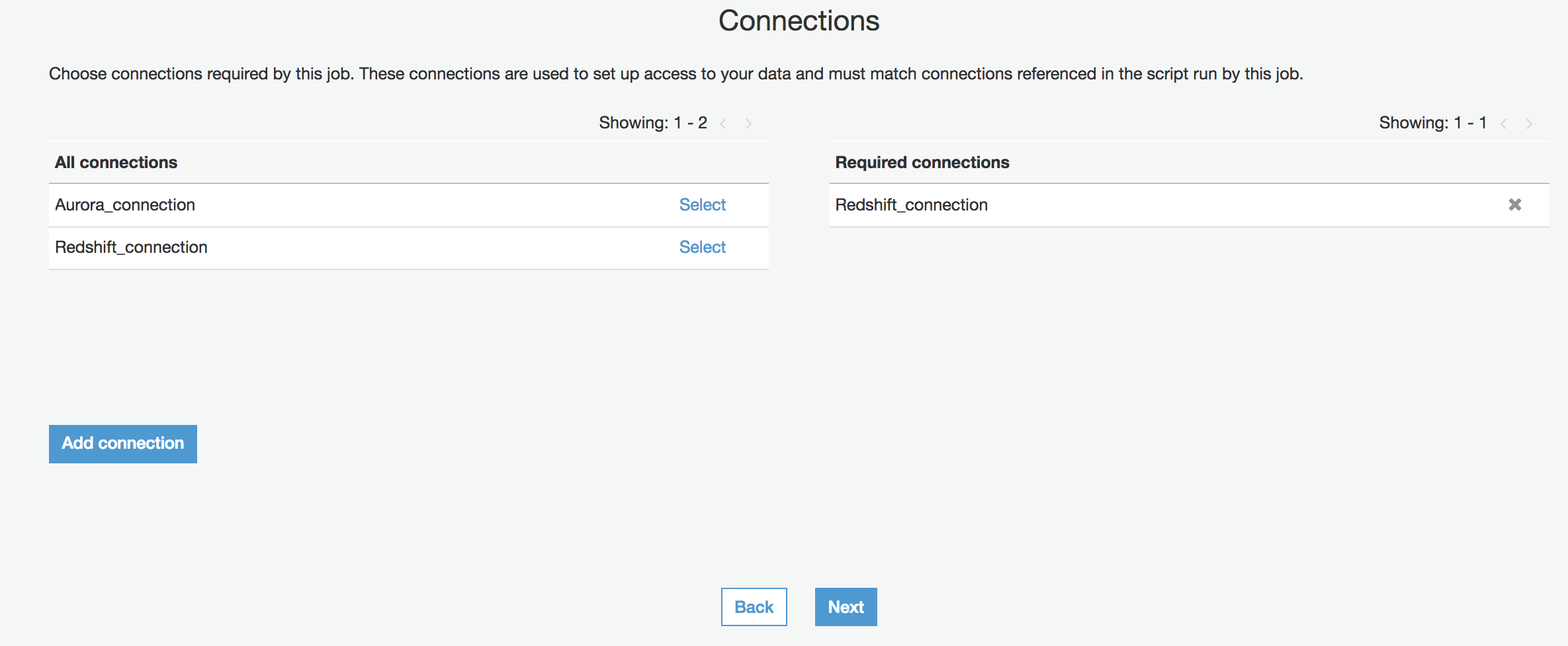
This job runs : A new script to be authored by you

**Advanced properties 메뉴를 확장하여**

Job bookmarks : Enable



1. Fact 테이블을 만들어서 S3와 Redshift에 동시에 데이터를 로딩시킬 것이므로 Connection에서 **[Redshift\_connection]**를 선택합니다.



1. 코드 수정 창에 아래 코드를 붙여넣기 합니다.

중요 : 반드시 최상단 S3bucket 명칭을 자신의 버킷명과 일치 시켜야 합니다.

|  |
| --- |
| import sys  import datetime  from awsglue.transforms import \*  from awsglue.dynamicframe import DynamicFrame  from awsglue.utils import getResolvedOptions  from pyspark.context import SparkContext  from awsglue.context import GlueContext  from awsglue.job import Job  args = getResolvedOptions(sys.argv, ['JOB\_NAME'])  sc = SparkContext()  glueContext = GlueContext(sc)  spark = glueContext.spark\_session  job = Job(glueContext)  job.init(args['JOB\_NAME'], args)  **s3Bucket = "s3://datalab-analytics-205756070026"**  s3Folder ="/analytics/"  # Set Source as s3\_data tables  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "order")  datasource1 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "order\_item")  datasource2 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "item")  datasource3 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "item\_category")  datasource4 = glueContext.create\_dynamic\_frame.from\_catalog(database = "s3\_data", table\_name = "member")  # Query to join  sql\_select = 'SELECT \* FROM orderTABLE, orderItemTable, itemTable, itemCategoryTable, memberTable WHERE orderTable.order\_id = orderItemTable.order\_id and orderItemTable.item\_id = itemTable.item\_id and itemTable.item\_category\_id = itemCategoryTable.category\_id and orderTable.member\_id = memberTable.member\_id order by orderTable.order\_id'  test\_df = datasource0.toDF()  test\_df.createOrReplaceTempView("orderTable")  test\_df2 = datasource1.toDF()  test\_df2.createOrReplaceTempView("orderItemTable")  test\_df3 = datasource2.toDF()  test\_df3.createOrReplaceTempView("itemTable")  test\_df4 = datasource3.toDF()  test\_df4.createOrReplaceTempView("itemCategoryTable")  test\_df5 = datasource4.toDF()  test\_df5.createOrReplaceTempView("memberTable")  test\_sql = spark.sql(sql\_select)  test\_sql\_dyf = DynamicFrame.fromDF(test\_sql, glueContext, "test\_sql\_dyf")  # Set Target as S3 into two types, json and parquet  datasink2 = glueContext.write\_dynamic\_frame.from\_options(frame = test\_sql\_dyf, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "fact\_order\_json", "partitionKeys" : ["order\_date"]}, format = "json", transformation\_ctx = "datasink2")  datasink3 = glueContext.write\_dynamic\_frame.from\_options(frame = test\_sql\_dyf, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "fact\_order\_parquet", "partitionKeys" : ["order\_date"]}, format = "parquet", transformation\_ctx = "datasink3")  datasink5 = glueContext.write\_dynamic\_frame.from\_options(frame = test\_sql\_dyf, connection\_type = "s3", connection\_options = {"path": s3Bucket + s3Folder + "fact\_order"}, format = "json", transformation\_ctx = "datasink5")  # Set Target as Redshift  applymapping1 = ApplyMapping.apply(frame = test\_sql\_dyf, mappings = [("member\_id", "string", "member\_id", "string"), ("order\_status", "string", "order\_status", "string"), ("country", "string", "country", "string"), ("shipping\_date", "string", "shipping\_date", "string"), ("total\_price", "int", "total\_price", "int"), ("city", "string", "city", "string"), ("order\_time", "string", "order\_time", "string"), ("state", "string", "state", "string"), ("postal\_code", "string", "postal\_code", "string"), ("region", "string", "region", "string"), ("order\_id", "string", "order\_id", "string"), ("item\_count", "int", "item\_count", "int"), ("item\_id", "string", "item\_id", "string"), ("item\_price", "int", "item\_price", "int"), ("price", "int", "price", "int"), ("name", "string", "name", "string"), ("description", "string", "description", "string"), ("reg\_ymdt", "string", "reg\_ymdt", "string"), ("item\_category\_id", "int", "item\_category\_id", "int"), ("category\_name", "string", "category\_name", "string"), ("category\_id", "int", "category\_id", "int"), ("parent\_category\_name", "string", "parent\_category\_name", "string"), ("login\_id", "string", "login\_id", "string"), ("gender", "string", "gender", "string"), ("last\_login\_ymdt", "string", "last\_login\_ymdt", "string"), ("login\_password", "string", "login\_password", "string"), ("age", "string", "age", "string"), ("order\_date", "string", "order\_date", "string")], transformation\_ctx = "applymapping1")  resolvechoice2 = ResolveChoice.apply(frame = applymapping1, choice = "make\_cols", transformation\_ctx = "resolvechoice2")  dropnullfields3 = DropNullFields.apply(frame = resolvechoice2, transformation\_ctx = "dropnullfields3")  datasink4 = glueContext.write\_dynamic\_frame.from\_jdbc\_conf(frame = dropnullfields3, catalog\_connection = "Redshift\_connection", connection\_options = {"dbtable": "fact\_order", "database": "datalab"}, redshift\_tmp\_dir = args["TempDir"], transformation\_ctx = "datasink4")  job.commit() |

1. 작업이 완료되면 S3 Analytics 버킷에 결과 폴더가 생긴 것을 확인 가능합니다.

