

**SeUng Jung**

**Data Life Cycle on AWS**

*Lab2-4. ETL to Redshift with Glue*

**Table of Contents**

Lab 설명 3

Lab Architecture 4

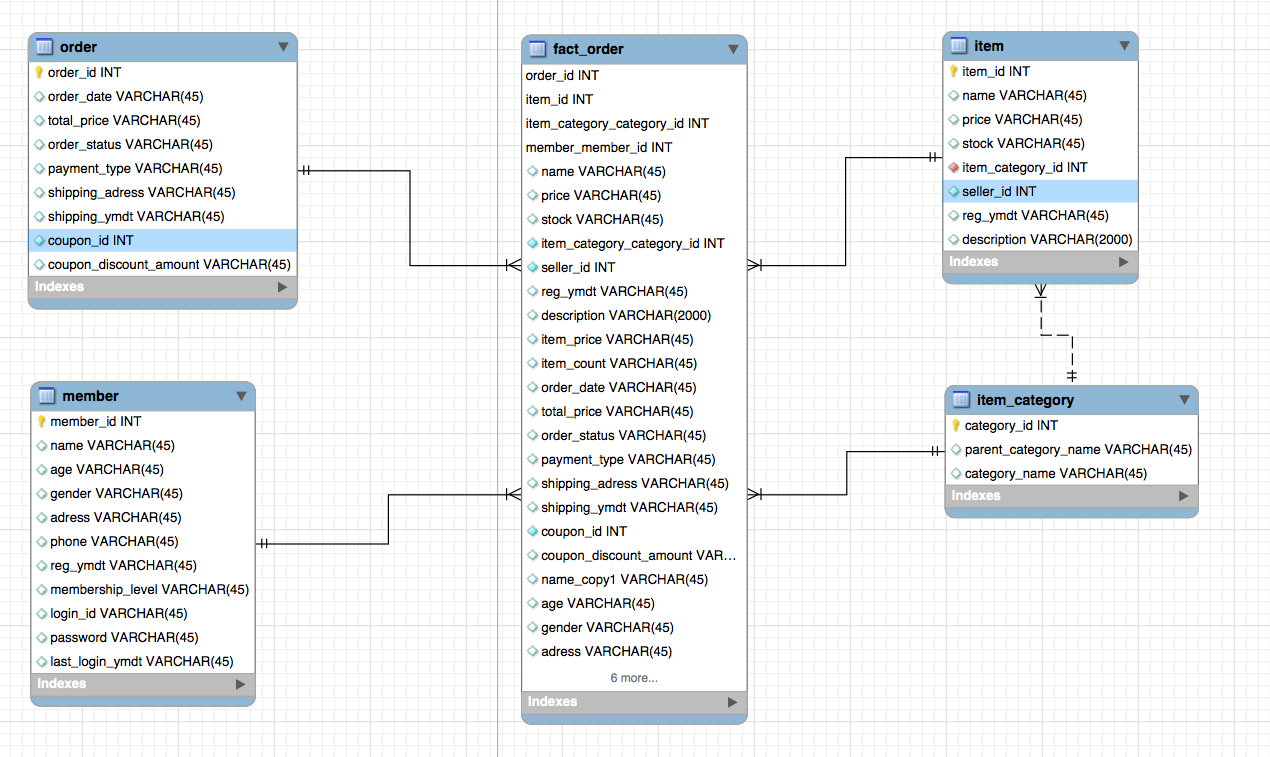
ETL to Redshift with Glue 5

# Lab 설명

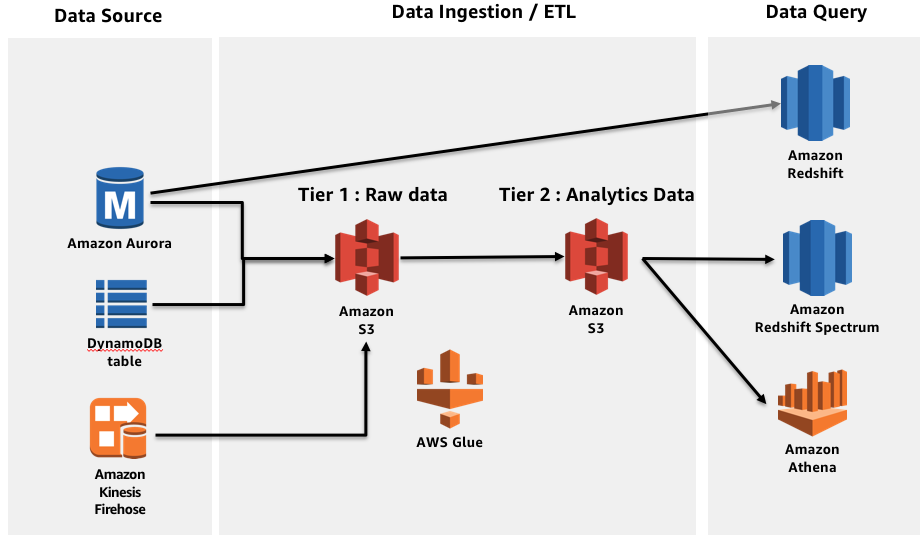
이번 Lab은 Aurora DB에 저장된 데이터와Tier 1 S3 bucket에 저장된 데이터를 Redshift로 변환 적재 하는 작업을 진행합니다.

실습 구성에서는 Dimension 테이블인 order, item, item\_category, member 테이블은 Aurora DB에서 직접 Redshift에 적재하고 Fact table인 fact\_order 테이블은 S3 데이터를 활용하여 생성하고 Redshift 와 S3에 각각 저장합니다.

이후에Redshift Spectrum을 활용하여 External Table로 쿼리 성능을 확인해 보도록 하겠습니다.



# 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

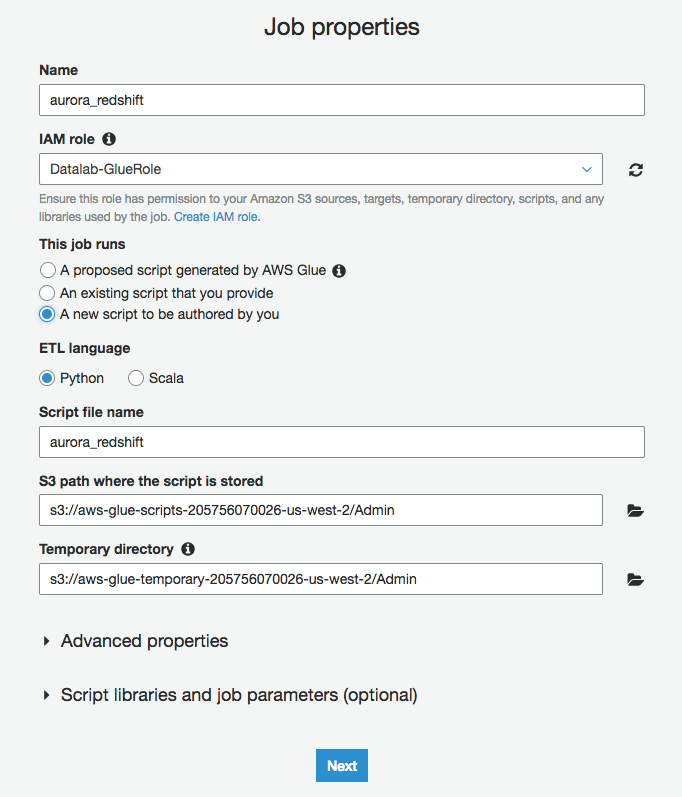
Lab 2-6. Query with Athena and Redshift spectrum

# ETL to Redshift with Glue

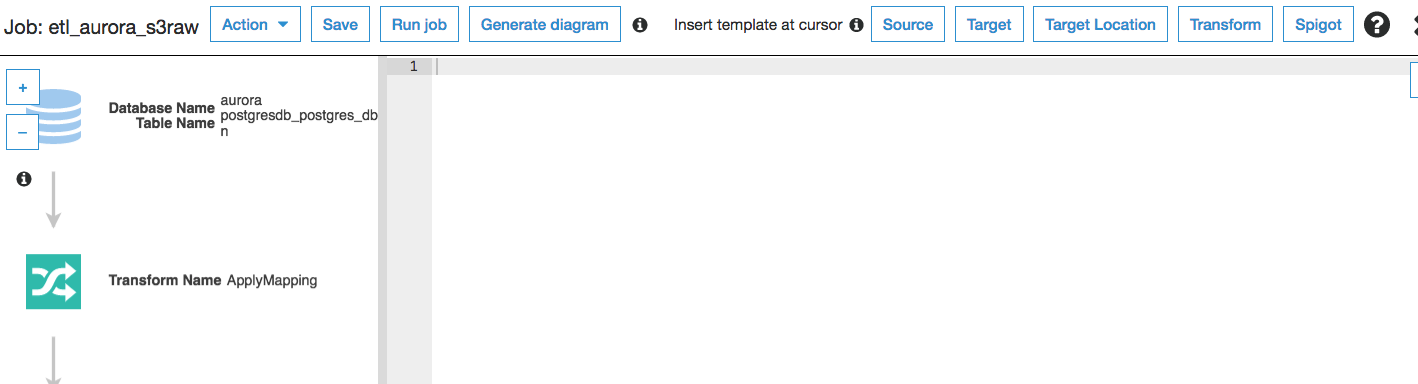
1. 우측 상단의 리전을 **Oregon (US-WEST)**으로 선택합니다.
2. AWS Management Console에서 AWS Glue 서비스 화면으로 이동
3. 먼저 Aurora에 저장되어 있는 테이블 중 일부를 Redshift로 이동시키는 Job을 생성합니다.
4. **[Create Job]**을 누르고 다음과 같이 입력합니다.

Name : aurora\_redshift

IAM role : Datalab-GlueRole

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

1. Connections에서 필요한 **[Aurora\_connection]**, **[Redshift connection]**을 선택해줍니다.
2. 코드 수정화면이 나오면 전체 코드를 삭제하고



1. 아래 코드를 붙여넣기 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: [TempDir, JOB\_NAME]  args = getResolvedOptions(sys.argv, ['TempDir','JOB\_NAME'])  sc = SparkContext()  glueContext = GlueContext(sc)  spark = glueContext.spark\_session  job = Job(glueContext)  job.init(args['JOB\_NAME'], args)  ## order table  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_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")  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": "order", "database": "datalab", "partitionKeys" : ["order\_date"]}, redshift\_tmp\_dir = args["TempDir"], transformation\_ctx = "datasink4")  ## order\_item table  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_order\_item", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, 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 = "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": "order\_item", "database": "datalab", "partitionKeys" : ["order\_date"]}, redshift\_tmp\_dir = args["TempDir"], transformation\_ctx = "datasink4")  ## member table  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_member", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, 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 = "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": "member", "database": "datalab"}, redshift\_tmp\_dir = args["TempDir"], transformation\_ctx = "datasink4")  ## item table  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_item", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, 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 = "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": "item", "database": "datalab"}, redshift\_tmp\_dir = args["TempDir"], transformation\_ctx = "datasink4")  ## item category table  datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "aurora", table\_name = "postgresdb\_postgres\_dba\_item\_category", transformation\_ctx = "datasource0")  applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("category\_name", "string", "category\_name", "string"), ("category\_id", "int", "category\_id", "int"), ("parent\_category\_name", "string", "parent\_category\_name", "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": "item\_category", "database": "datalab"}, redshift\_tmp\_dir = args["TempDir"], transformation\_ctx = "datasink4")  job.commit() |

1. 작업이 완료되면 Redshift에서 해당 테이블을 확인해볼 수 있습니다. 해당 내용은 이후 lab에서 진행합니다.