PROJECT OVERVIEW

STEDI Human Balance Analytics

The Challenge

In this project you'll act as a data engineer for the STEDI team to build a Data Lakehouse solution for sensor data that trains a machine learning model. The STEDI team has been hard at work developing a hardware STEDI step trainer that trains the user to do a balance exercise

The Device

There are sensors on the device that collect data to train a machine learning algorithm to detect steps. It also has a companion mobile app that collects customer data and interacts with the device sensors. The step trainer is just a motion sensor that records the distance of the object detected.

The Project

I'll work to build data pipelines that use Apache Spark to store, filter, process, and transform data from STEDI users for data analytics and machine learning applications.

Project Instructions

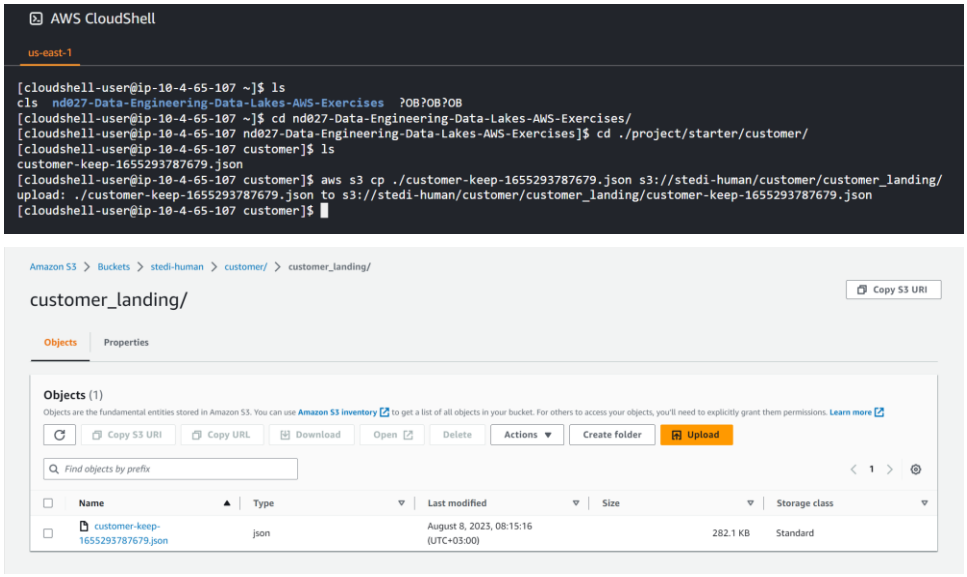
Using AWS Glue, AWS S3, Python, and Spark, create or generate Python scripts to build a lakehouse solution in AWS that satisfies these requirements from the STEDI data scientists.

Requirements

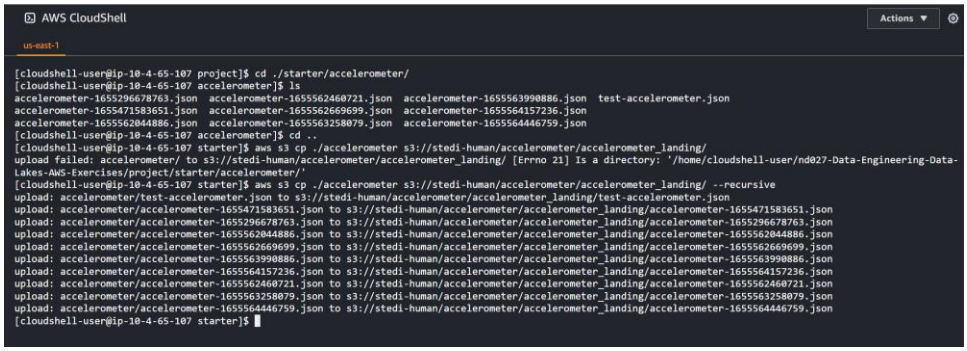
To simulate the data coming from the various sources, you will need to create your own S3 directories for customer_landing, step_trainer_landing, and accelerometer_landing zones, and copy the data there as a starting point.

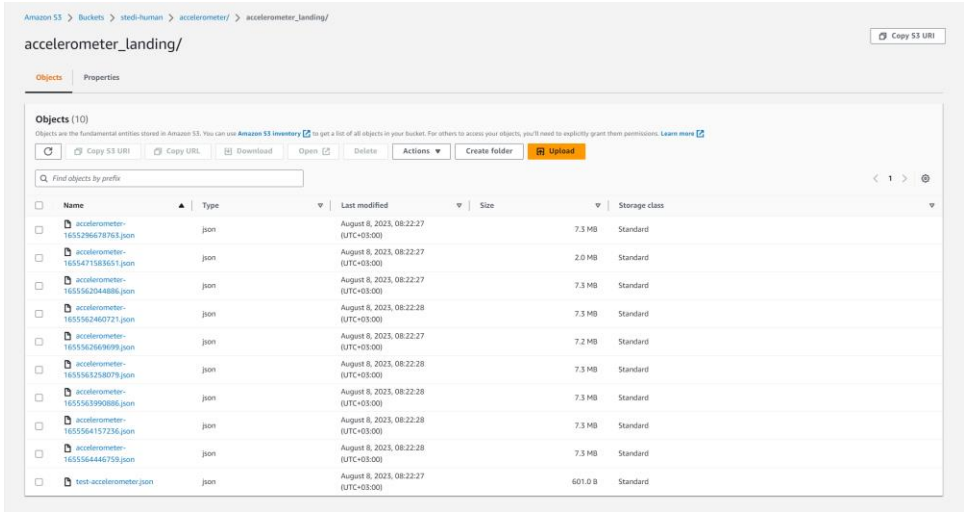
Clone data to the landing zone

- Customer Data

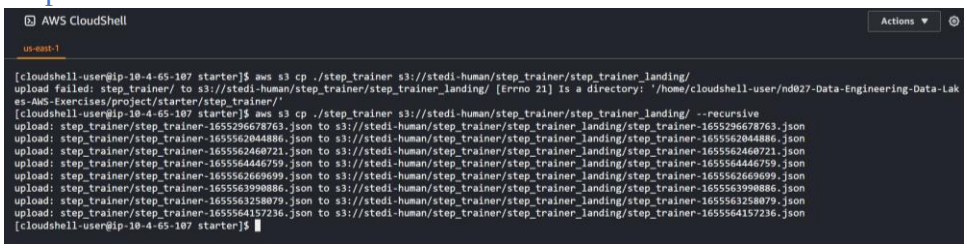


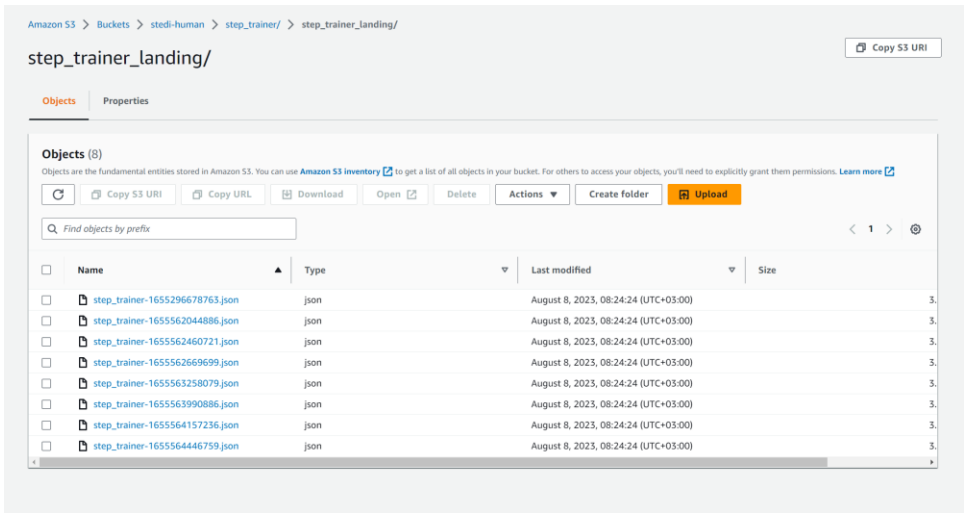
- Accelerometer Data





- Step Trainer Data





You have decided you want to get a feel for the data you are dealing with in a semi-structured format, so you decide to create **two Glue tables** for the two landing zones. Share your customer_landing.sql and your accelerometer_landing.sql script in git.

customer_landing.sql

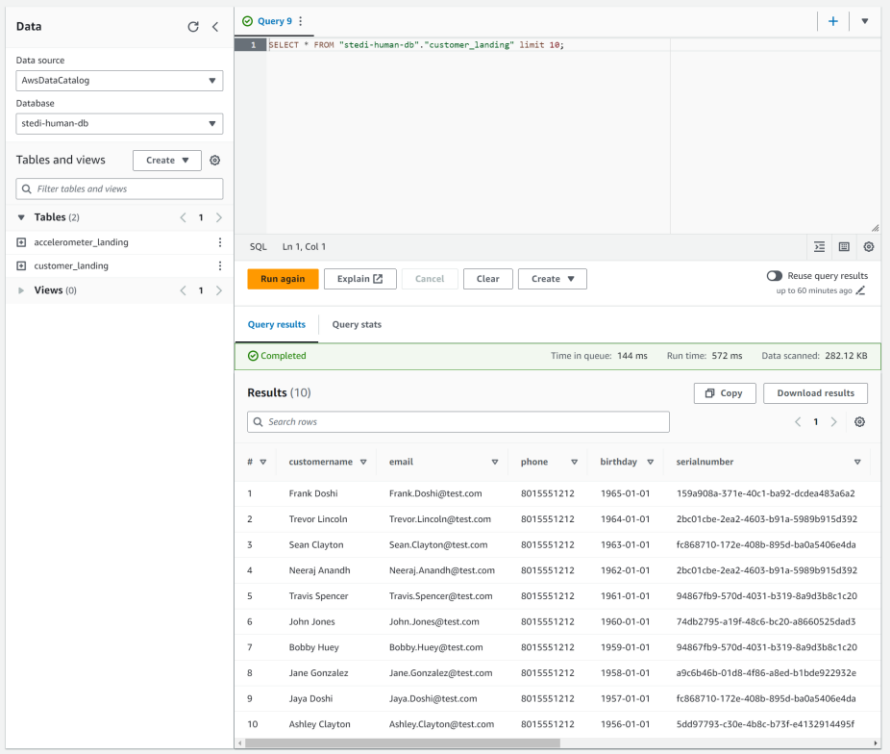
```
CREATE EXTERNAL TABLE IF NOT EXISTS `stedi-human-db`.`customer_landing` (
  `customerName` string,
  `email` string,
  `phone` string,
  `birthDay` string,
  `serialNumber` string,
  `registrationDate` bigint,
  `lastUpdateDate` bigint,
  `shareWithResearchAsOfDate` bigint,
  `shareWithPublicAsOfDate` bigint,
  `shareWithFriendsAsOfDate` bigint
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (
  'ignore.malformed.json' = 'FALSE',
  'dots.in.keys' = 'FALSE',
  'case.insensitive' = 'TRUE',
  'mapping' = 'TRUE'
)
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://stedi-human/customer/customer_landing/'
TBLPROPERTIES ('classification' = 'json');
```

accelerometer_landing.sql

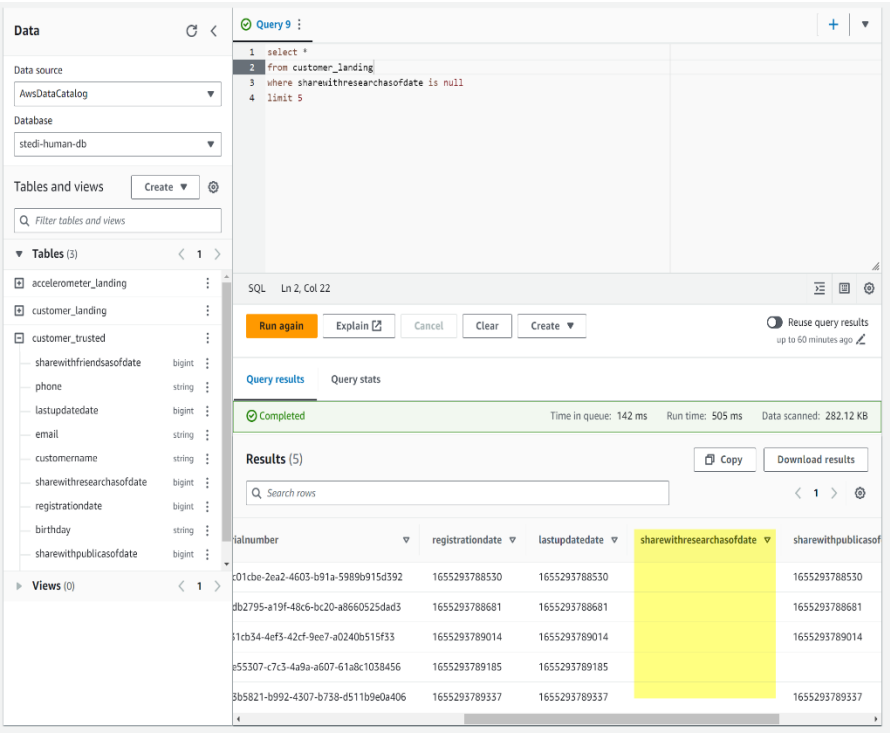
```
CREATE EXTERNAL TABLE IF NOT EXISTS `stedi-human-db`.`accelerometer_landing` (
  `user` string,
  `timeStamp` bigint,
  `x` float,
  `y` float,
  `z` float
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (
  'ignore.malformed.json' = 'FALSE',
  'dots.in.keys' = 'FALSE',
  'case.insensitive' = 'TRUE',
  'mapping' = 'TRUE'
)
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://stedi-human/accelerometer/accelerometer_landing/'
TBLPROPERTIES ('classification' = 'json');
```

- Query those tables using Athena, and take a screenshot of each one showing the resulting data. Name the screenshots customer_landing(.png,.jpeg, etc.) and accelerometer_landing(.png,.jpeg, etc.).

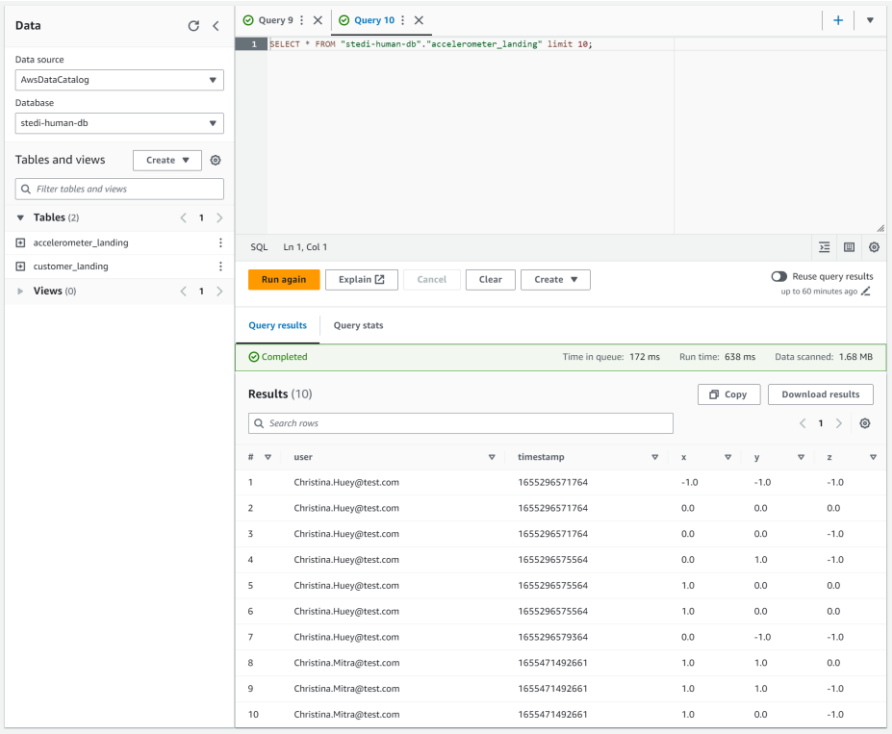
customer_landing.png



customer_landing_2.png



accelerometer_landing.png



The Data Science team has done some preliminary data analysis and determined that the Accelerometer Records each match one of the Customer Records.
They would like you to create 2 AWS Glue Jobs that do the following:

1. Sanitize the Customer data from the Website (Landing Zone) and only store the Customer Records who agreed to share their data for research purposes (Trusted Zone) - creating a Glue Table called customer_trusted.

   customer_landing_to_trusted.py

```
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
import re

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node Customer Landing Zone
```

```
CustomerLandingZone_node1 =
glueContext.create_dynamic_frame.from_options(
    format_options={"multiline": False},
    connection_type="s3",
    format="json",
    connection_options={
        "paths": ["s3://stedi-human/customer/customer_landing/"],
        "recurse": True,
    },
    transformation_ctx="CustomerLandingZone_node1",
)

# Script generated for node Privacy filter
Privacyfilter_node1691480634830 = Filter.apply(
    frame=CustomerLandingZone_node1,
    f=lambda row: (not (row["shareWithResearchAsOfDate"] == 0)),
    transformation_ctx="Privacyfilter_node1691480634830",
)

# Script generated for node Customer Trusted Zone
CustomerTrustedZone_node3 =
glueContext.write_dynamic_frame.from_options(
    frame=Privacyfilter_node1691480634830,
    connection_type="s3",
    format="json",
    connection_options={
        "path": "s3://stedi-human/customer/customer_trusted/",
        "partitionKeys": [],
    },
    transformation_ctx="CustomerTrustedZone_node3",
)

job.commit()
```

2. Sanitize the Accelerometer data from the Mobile App (Landing Zone) - and only store Accelerometer Readings from customers who agreed to share their data for research purposes (Trusted Zone) - creating a Glue Table called accelerometer_trusted.
   Dsa

   accelerometer_landing_to _trusted.py

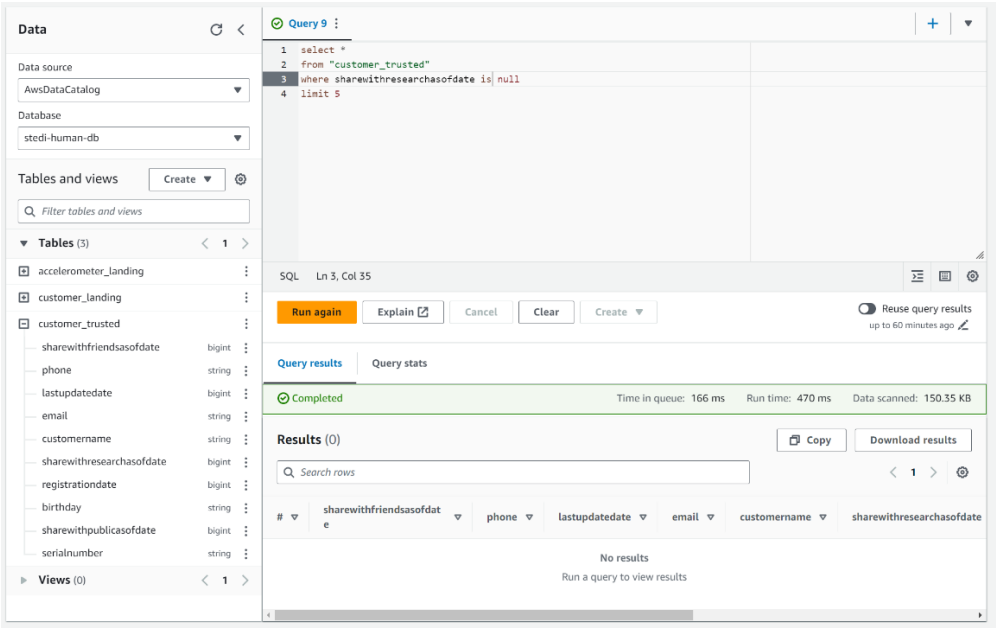
```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
```

```
args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node Customer Trusted Zone
CustomerTrustedZone_node1 = glueContext.create_dynamic_frame.from_options(
    format_options={"multiline": False},
    connection_type="s3",
    format="json",
    connection_options={
        "paths": ["s3://stedi-human/customer/customer_trusted/"],
        "recurse": True,
    },
    transformation_ctx="CustomerTrustedZone_node1",
)

# Script generated for node Accelerometer Landing
AccelerometerLanding_node1691481731514 = glueContext.create_dynamic_frame.from_options(
    format_options={"multiline": False},
    connection_type="s3",
    format="json",
    connection_options={
        "paths": ["s3://stedi-human/accelerometer/accelerometer_landing/"],
        "recurse": True,
    },
    transformation_ctx="AccelerometerLanding_node1691481731514",
)

# Script generated for node Join privacy filter
Joinprivacyfilter_node1691481845940 = Join.apply(
    frame1=CustomerTrustedZone_node1,
    frame2=AccelerometerLanding_node1691481731514,
    keys1=["email"],
    keys2=["user"],
    transformation_ctx="Joinprivacyfilter_node1691481845940",
)

# Script generated for node Drop Fields
DropFields_node1691481879689 = DropFields.apply(
    frame=Joinprivacyfilter_node1691481845940,
    paths=[
        "serialNumber",
        "shareWithPublicAsOfDate",
        "birthDay",
```

```
        "registrationDate",
        "shareWithResearchAsOfDate",
        "customerName",
        "email",
        "lastUpdateDate",
        "phone",
        "shareWithFriendsAsOfDate",
    ],
    transformation_ctx="DropFields_node1691481879689",
)

# Script generated for node Accelerometer Trusted
AccelerometerTrusted_node3 = glueContext.write_dynamic_frame.from_options(
    frame=DropFields_node1691481879689,
    connection_type="s3",
    format="json",
    connection_options={
        "path": "s3://stedi-human/accelerometer/accelerometer_trusted/",
        "partitionKeys": [],
    },
    transformation_ctx="AccelerometerTrusted_node3",
)

job.commit()
```

3. You need to verify your Glue job is successful and only contains Customer Records from people who agreed to share their data. Query your Glue customer_trusted table with Athena and take a screenshot of the data. Name the screenshot customer_trusted(.png,.jpeg, etc.).

Table 1 : customer_trusted_query.png



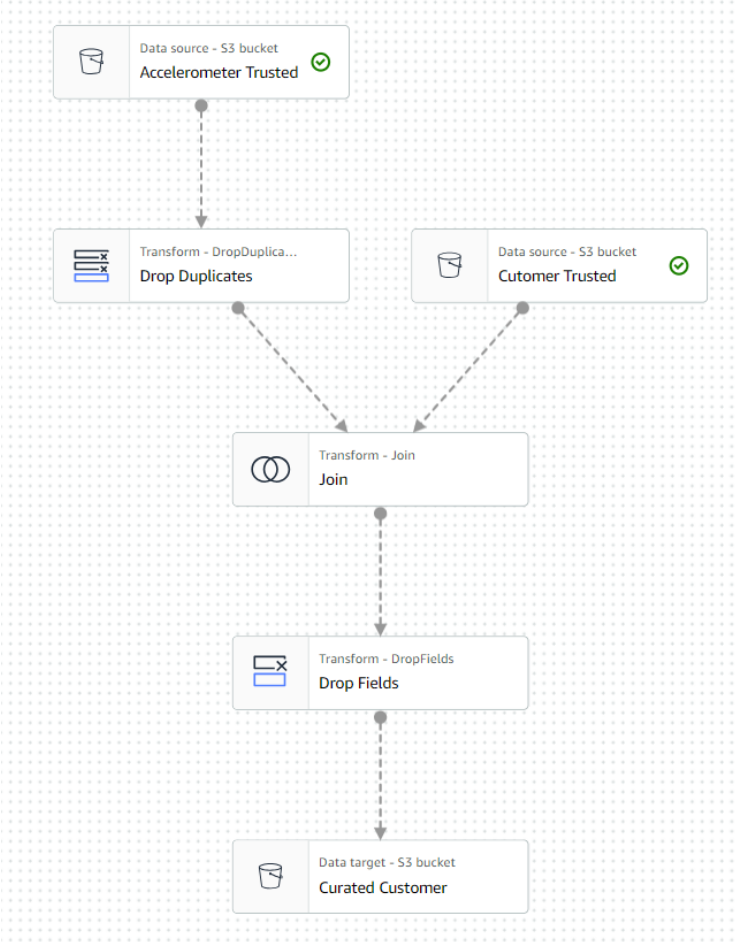
Data Scientists have discovered a data quality issue with the Customer Data. The serial number should be a unique identifier for the STEDI Step Trainer they purchased. However, there was a defect in the fulfillment website, and it used the same 30 serial numbers over and over again for millions of customers! Most customers have not received their Step Trainers yet, but those who have, are submitting Step Trainer data over the IoT network (Landing Zone). *The data from the Step Trainer Records has the correct serial numbers.*

The problem is that because of this serial number bug in the fulfillment data (Landing Zone), we don't know which customer the Step Trainer Records data belongs to.

The Data Science team would like you to write a Glue job that does the following:

1. Sanitize the Customer data (Trusted Zone) and create a Glue Table (Curated Zone) that only includes customers who have accelerometer data *and* have agreed to share their data for research called **customers_curated**.

   I'm not sure if my answer is correct, I drop duplicate user for accelerometer_trusted table to give the distinct user (email in customer_trusted) and the join the two tables, then drop fields that related to accelerometer_trusted.



customer_trusted_to_curated.py

```
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
from awsglue.dynamicframe import DynamicFrame
from pyspark.sql import functions as SqlFuncs

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node Cutomer Trusted
CutomerTrusted_node1 = glueContext.create_dynamic_frame.from_options(
    format_options={"multiline": False},
    connection_type="s3",
    format="json",
    connection_options={
        "paths": ["s3://stedi-human/customer/customer_trusted/"],
        "recurse": True,
    },
    transformation_ctx="CutomerTrusted_node1",
)

# Script generated for node Accelerometer Trusted
AccelerometerTrusted_node1691493201076 =
glueContext.create_dynamic_frame.from_options(
    format_options={"multiline": False},
    connection_type="s3",
    format="json",
    connection_options={
        "paths": ["s3://stedi-human/accelerometer/accelerometer_trusted/"],
        "recurse": True,
    },
    transformation_ctx="AccelerometerTrusted_node1691493201076",
)

# Script generated for node Drop Duplicates
DropDuplicates_node1691495792782 = DynamicFrame.fromDF(
    AccelerometerTrusted_node1691493201076.toDF().dropDuplicates(["user"]),
    glueContext,
    "DropDuplicates_node1691495792782",
```

```
)

# Script generated for node Join
Join_node1691495822535 = Join.apply(
    frame1=CutomerTrusted_node1,
    frame2=DropDuplicates_node1691495792782,
    keys1=["email"],
    keys2=["user"],
    transformation_ctx="Join_node1691495822535",
)

# Script generated for node Drop Fields
DropFields_node1691493448444 = DropFields.apply(
    frame=Join_node1691495822535,
    paths=["z", "timeStamp", "user", "y", "x"],
    transformation_ctx="DropFields_node1691493448444",
)

# Script generated for node Curated Customer
CuratedCustomer_node3 = glueContext.write_dynamic_frame.from_options(
    frame=DropFields_node1691493448444,
    connection_type="s3",
    format="json",
    connection_options={
        "path": "s3://stedi-human/customer/customer_curated/",
        "partitionKeys": [],
    },
    transformation_ctx="CuratedCustomer_node3",
)

job.commit()
```

Finally, you need to ***create two Glue Studio jobs*** that do the following tasks:

1. Read the Step Trainer IoT data stream (S3) and populate a Trusted Zone Glue Table called **step_trainer_trusted** that contains the Step Trainer Records data for customers who have accelerometer data and have agreed to share their data for research (customers_curated).

   step_trainer_landing_to_trusted.py

```
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

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node Step Trainer Landing
StepTrainerLanding_node1 =
glueContext.create_dynamic_frame.from_options(
    format_options={"multiline": False},
    connection_type="s3",
    format="json",
    connection_options={
        "paths": ["s3://stedi-
human/step_trainer/step_trainer_landing/"],
        "recurse": True,
    },
    transformation_ctx="StepTrainerLanding_node1",
)

# Script generated for node Customer Curated
CustomerCurated_node1691500664051 =
glueContext.create_dynamic_frame.from_options(
    format_options={"multiline": False},
    connection_type="s3",
    format="json",
    connection_options={
        "paths": ["s3://stedi-human/customer/customer_curated/"],
        "recurse": True,
    },
    transformation_ctx="CustomerCurated_node1691500664051",
)
```

```
# Script generated for node Join
Join_node1691500697750 = Join.apply(
    frame1=StepTrainerLanding_node1,
    frame2=CustomerCurated_node1691500664051,
    keys1=["serialNumber"],
    keys2=["serialNumber"],
    transformation_ctx="Join_node1691500697750",
)

# Script generated for node Drop Fields
DropFields_node1691500723878 = DropFields.apply(
    frame=Join_node1691500697750,
    paths=[
        "shareWithFriendsAsOfDate",
        "phone",
        "`.serialNumber`",
        "birthDay",
        "shareWithPublicAsOfDate",
        "shareWithResearchAsOfDate",
        "registrationDate",
        "customerName",
        "email",
        "lastUpdateDate",
    ],
    transformation_ctx="DropFields_node1691500723878",
)

# Script generated for node S3 bucket
S3bucket_node3 = glueContext.write_dynamic_frame.from_options(
    frame=DropFields_node1691500723878,
    connection_type="s3",
    format="json",
    connection_options={
        "path": "s3://stedi-human/step_trainer/step_trainer_trusted/",
        "partitionKeys": [],
    },
    transformation_ctx="S3bucket_node3",
)

job.commit()
```

2. Create an aggregated table that has each of the Step Trainer Readings, and the associated accelerometer reading data for the same timestamp, but only for customers who have agreed to share their data, and make a glue table called **machine_learning_curated.**

machine_learning_curated.py

```
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

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node Accelerometer Trusted
AccelerometerTrusted_node1 =
glueContext.create_dynamic_frame.from_options(
    format_options={"multiline": False},
    connection_type="s3",
    format="json",
    connection_options={
        "paths": ["s3://stedi-
human/accelerometer/accelerometer_trusted/"],
        "recurse": True,
    },
    transformation_ctx="AccelerometerTrusted_node1",
)

# Script generated for node Step Trainer Trusted
StepTrainerTrusted_node1691500952137 =
glueContext.create_dynamic_frame.from_options(
    format_options={"multiline": False},
    connection_type="s3",
    format="json",
    connection_options={
        "paths": ["s3://stedi-
human/step_trainer/step_trainer_trusted/"],
        "recurse": True,
    },
    transformation_ctx="StepTrainerTrusted_node1691500952137",
)
```

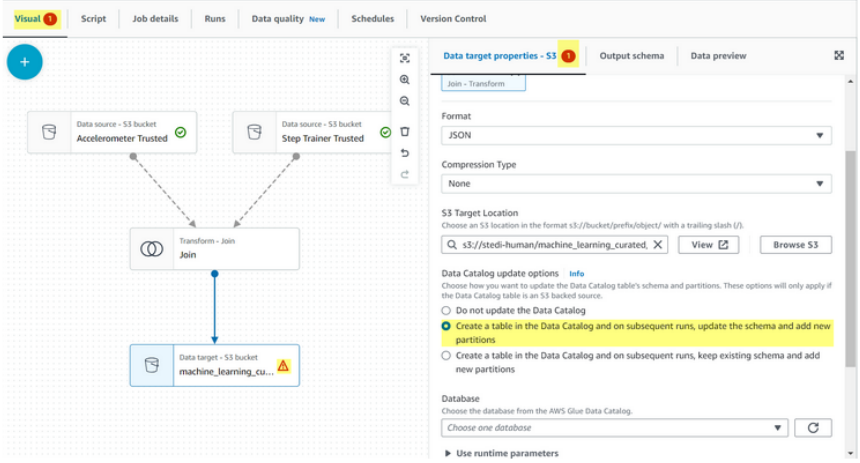
```
# Script generated for node Join
Join_node1691500955442 = Join.apply(
    frame1=StepTrainerTrusted_node1691500952137,
    frame2=AccelerometerTrusted_node1,
    keys1=["sensorReadingTime"],
    keys2=["timeStamp"],
    transformation_ctx="Join_node1691500955442",
)

# Script generated for node machine_learning_curated
machine_learning_curated_node3 =
glueContext.write_dynamic_frame.from_options(
    frame=Join_node1691500955442,
    connection_type="s3",
    format="json",
    connection_options={
        "path": "s3://stedi-human/machine_learning_curated/",
        "partitionKeys": [],
    },
    transformation_ctx="machine_learning_curated_node3",
)

job.commit()
```

NOTE: For specifications the request dynamically infer and update schema is enabled. I try it and I face a warring and I can’t make a job.

Trusted Zone

Success Criteria	Specifications
Configure Glue Studio to dynamically update a Glue Table schema from JSON data	Glue Job Python code shows that the option to dynamically infer and update schema is enabled.
Use Athena to query Trusted Glue Tables	A screenshot that shows a select * statement from Athena showing the customer landing data, where the resulting customer trusted data has no rows where shareWithResearchAsOfDate is blank.
Join Privacy tables with Glue Jobs	Glue jobs have inner joins that join up with the customer_landing table on the serialnumber field. (customer_landing_to_trusted.py and accelerometer_landing_to_trusted_zone.py)
Filter protected PII with Spark in Glue Jobs	Glue jobs drop data that doesn’t have data in the sharedWithResearchAsOfDate column (customer_landing_to_trusted.py and accelerometer_landing_to_trusted_zone.py)

When update schema option is enable.



When do not update the data catalog enable.