# AWS GLUE ETL TRANSFORMATIONS

In this article, we explain how to do ETL transformations in Amazon's Glue. For background material please consult How To Join Tables in AWS Glue. You first need to set up the crawlers in order to create some data.

By this point you should have created a **titles** DynamicFrame using this code below. Now we can show some ETL transformations.

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import *


glueContext = GlueContext(SparkContext.getOrCreate())

titles =
glueContext.create_dynamic_frame.from_catalog(database="moviesandratings",
table_name="movieswalker")
```

## Select fields

This ETL transformation creates a new DynamicFrame by taking the fields in the **paths** list. We use **toDF().show()** to turn it into Spark Dataframe and print the results.

```
titles.select_fields(paths=).toDF().show()
```

# Map

The **map** function iterates over every record (called a **DynamicRecord**) in the DynamicFrame and runs a function over it.

First create a function that takes a DynamicRecord as an argument and returns the DynamicRecord. Here we take one column and make it uppercase:

```
def upper(rec):
    rec=rec.upper()
    return rec
```

Then call that function on the DynamicFrame **titles**.

```
Map.apply(frame=titles,f=upper).toDF().show()
```

# Apply mapping

```
This method changes column names and types. mappings is an array of tuples
("oldName", "oldType", "newName", "newType").
```

# DynamicFrameCollection

A Dynamic Frame collection is a dictionary of Dynamic Frames. We can create one using the **split_fields** function. Then you can run the same map, flatmap, and other functions on the collection object. Glue provides methods for the collection so that you don't need to loop through the dictionary keys to do that individually.

Here we create a DynamicFrame Collection named **dfc**. The first DynamicFrame **splitoff** has the columns **tconst** and **primaryTitle**. The second DynamicFrame **remaining** holds the remaining columns.

```
dfc=titles.split_fields(paths=,name1="splitoff",name2="remaining")
>>> dfc.keys()
dict_keys()
```

We show the DynamicFrame **splitoff** below

```
dfc.toDF().show()
+----------+------------------+
|    tconst|       primaryTitle|
+----------+------------------+
| tt0276132|      The Fetishist|
| tt0279481|        Travel Daze|
| tt0305295|        Bich bozhiy|
```

# Create a Dynamic DataFrame from a Spark DataFrame

As we can turn DynamicFrames into Spark Dataframes, we can go the other way around. We can create data by first creating a Spark Dataframe and then using the **fromDF** function.

We use the Apache Spark SQL **Row** object.

```
from pyspark.sql import *

walker = Row(name='Walker',age=59)
stephen = Row(name='Stephen', age=40)
students=

dfc=spark.createDataFrame(students).fromDF
```

# Additional resources

Explore these resources:

- BMC Machine Learning & Big Data Blog
- Apache Spark Guide
- AWS Guide