**TRANSFORMER PIPELINE :**

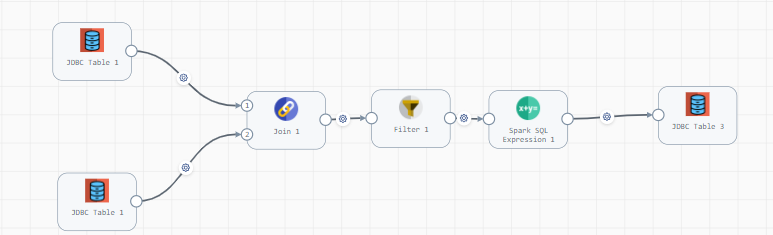
**Calculate Geographic Distance: UDF**

**Prerequisites for ETL Pipeline Implementation**

Before embarking on the implementation of the ETL pipeline, ensure that the following prerequisites are in place:

1. **Stable Internet Connection:**
   * A reliable and high-speed internet connection is essential for seamless data extraction, transformation, and loading processes.
2. **StreamSets Account:**
   * Create a StreamSets account to access the platform for ETL operations. If you don't have an account, sign up for the 30-day free trial to explore its features.
3. **Docker Desktop:**
   * Install Docker Desktop, which provides a convenient and consistent environment for running containerized applications, ensuring portability and ease of deployment.
4. **Windows Operating System:**
   * The ETL pipeline is configured for Windows OS, ensuring compatibility and optimal performance. Ensure that your system is running a supported version of Windows.In My case I Am Using Windows 10 Pro.
5. **MySQL for Read and Write:**
   * Set up MySQL as both the source and destination for your ETL pipeline.
   * This relational database will facilitate the extraction of data from the source and the loading of transformed data into the target system.
6. **Login Credentials:**
   * Collect and securely store the necessary login credentials for StreamSets and MySQL.
   * These credentials will be utilized during the configuration of the ETL pipeline components.
7. **Environment Setup:**
   * Familiarize yourself with the Docker environment and ensure that it is up and running.
   * Verify that StreamSets and MySQL are accessible within the Docker containers.
8. **System Requirements:**
   * Check and meet the minimum system requirements for running StreamSets and Docker on your Windows OS.
   * Ensure that your system resources align with the specified recommendations.
9. **Data Connection Details:**
   * Gather details about the source data (in MySQL) and understand the connection parameters required for StreamSets to establish a secure and efficient data transfer.
10. **Access Permissions:**
    * Confirm that the necessary permissions are granted for StreamSets to interact with MySQL for both reading and writing data. This includes database access and user privileges.
11. **Testing Environment:**
    * Set up a testing environment within Docker to validate the functionality of the ETL pipeline before deploying it in a production environment.
12. **Documentation Review:**
    * Review and keep this document handy for quick reference during the implementation phase. Ensure that all prerequisites are met to avoid potential roadblocks.

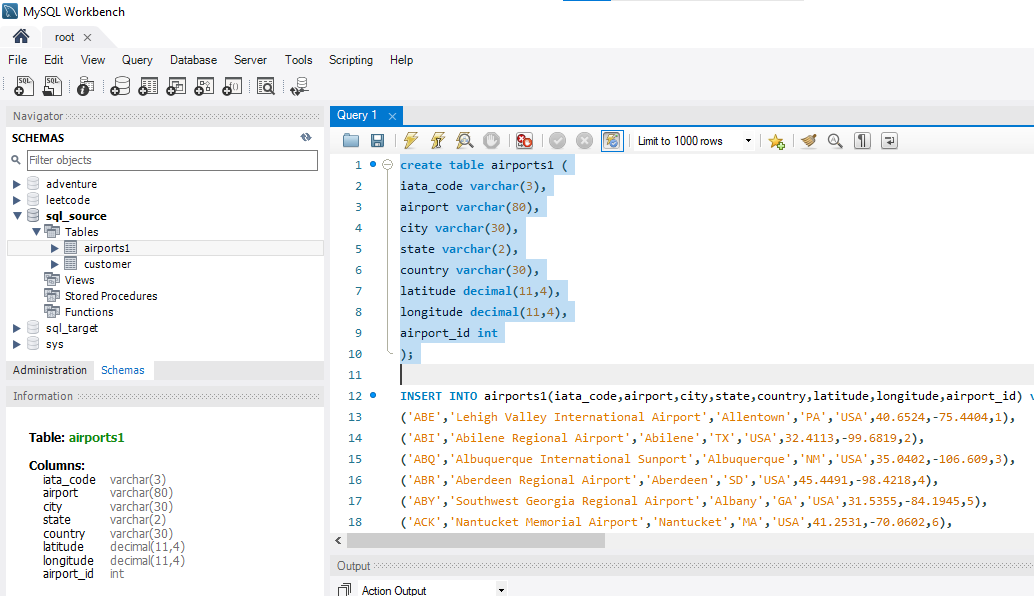
PIPELINE Architectures :

  
  
The Above Pipeline is Dived into 3 categories   
Source =Reads data from mysql table   
Processors=Usually it was called Transformation part like Cleaning or Enrich the Data into this Part.

Destination = Writes Data into the MYsql .

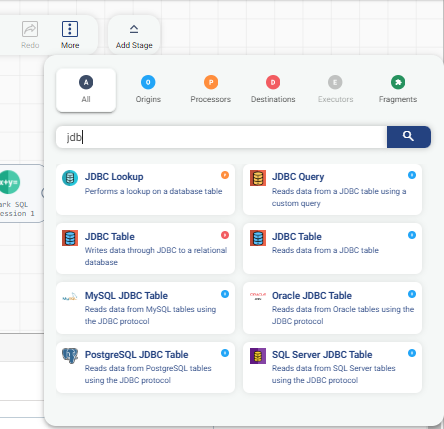
Step 1: ORIGIN SETUP  
Part 1: Setting up MySQL Database

Open MySQL and execute the provided SQL script available in the GitHub repository. You can download the SQL script from the following Github Link [Download Sql Script](https://github.com/lakshmi-narayanant/ETL_StremsetsUsecse/blob/main/Transformer/Calculate%20Geographic%20Distance%20-%20UDF/airport.sql) .



**Part 2: Configuring StreamSets for JDBC Origin**

1. Open the StreamSets UI.
2. Add a new JDBC Origin to your pipeline to facilitate data extraction from the MySQL database. Configure the JDBC Origin with the necessary details.



**Connection Information:**

* + - Database URL or JDBC Connection String:

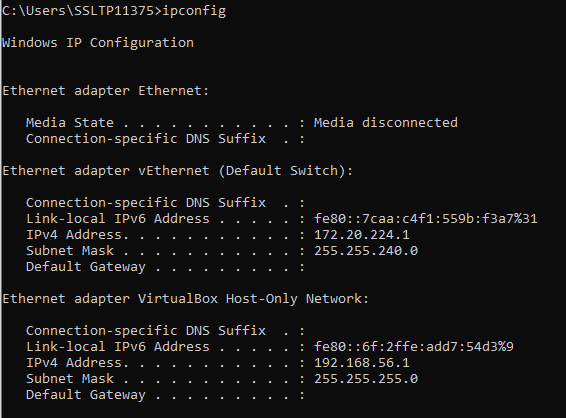
jdbc:mysql://172.20.224.1:3306

Certainly! Here's a refined explanation suitable for a professional document:

**Database Connection URL Explanation:**

The provided JDBC connection URL **jdbc:mysql://172.20.224.1:3306** is instrumental for establishing a connection to a MySQL database. Let's dissect its components:

* **jdbc:** This denotes the use of the Java Database Connectivity (JDBC) API, a Java-based technology facilitating database connections.
* **mysql:** Specifies the database management system in use, which, in this case, is MySQL.
* **//172.20.224.1:** Represents the IP address (or hostname) of the server where the MySQL database is hosted. In this context, the IP address is 172.20.224.1, indicating the destination for the JDBC connection.
* **Open command prompt and type ipconfig to copy and paste the IPv4 Address**

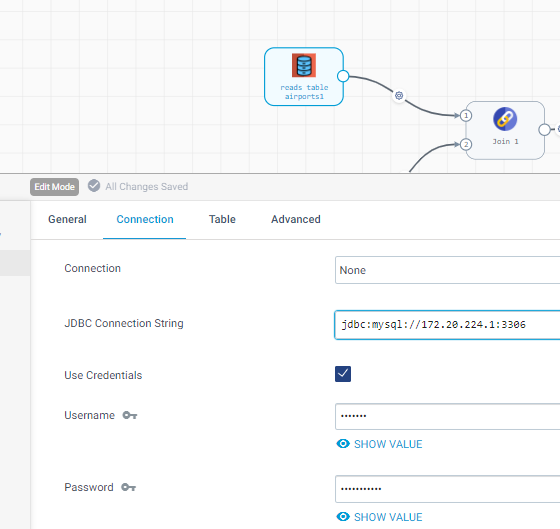


* **:3306:** Signifies the communication port on which the MySQL database server is listening. The default port for MySQL is 3306.

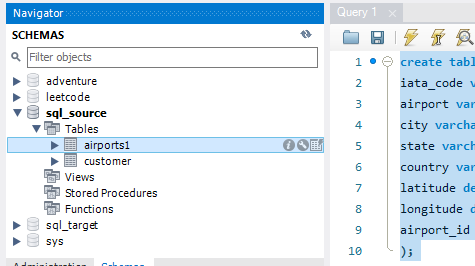
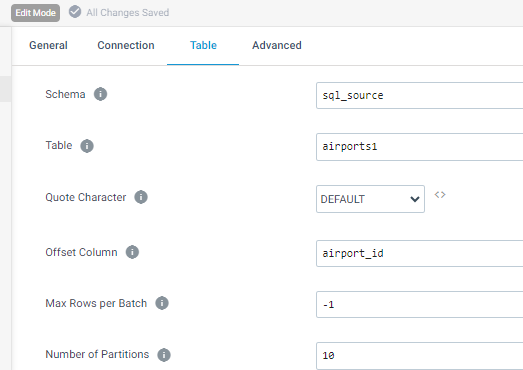
In summary, the entire connection string **jdbc:mysql://172.20.224.1:3306** is a JDBC URL directing a connection to a MySQL database situated at the specified IP address (172.20.224.1) and communicating over port 3306.

* + Username: XXXXXX
  + Password: YYYYYY
  + **Other Configuration:**
  + [Include any additional configuration parameters]

Tab1:



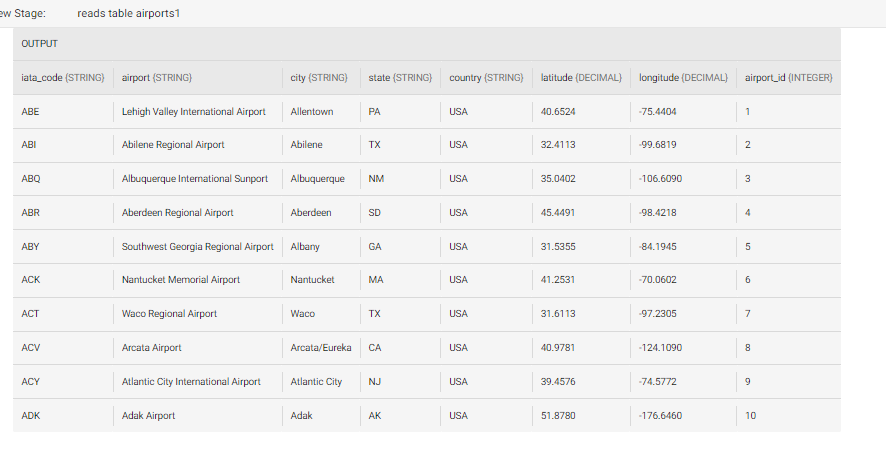
Tab2:



1. Once configured, validate the connection settings by running a connection test.
2. Ensure that the connection test is successful before proceeding to further pipeline development.

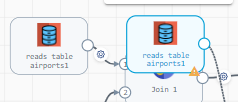
By following these steps, you will have successfully set up MySQL, created a table using the provided SQL script, and configured StreamSets to extract data from the MySQL database using the JDBC Origin. This process ensures a seamless and efficient data integration workflow.

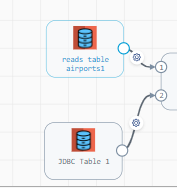
**Output:**

****

**ORIGIN2:**

Copy the Origin 1 AND paste to Corresponding location



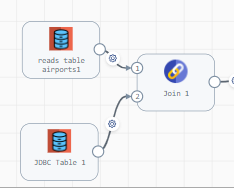
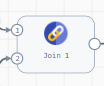


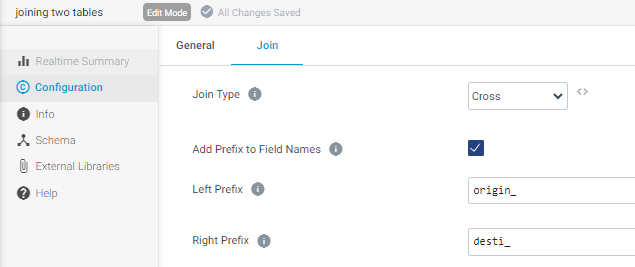
**Step2:**

**PROCESSORS**

**ADD Join Procesor In pipeline**

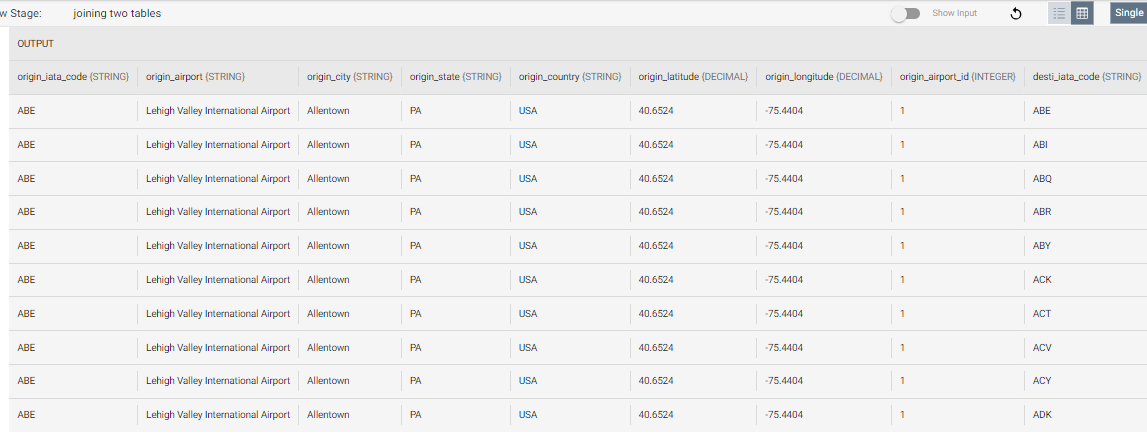
**Join Processor:**

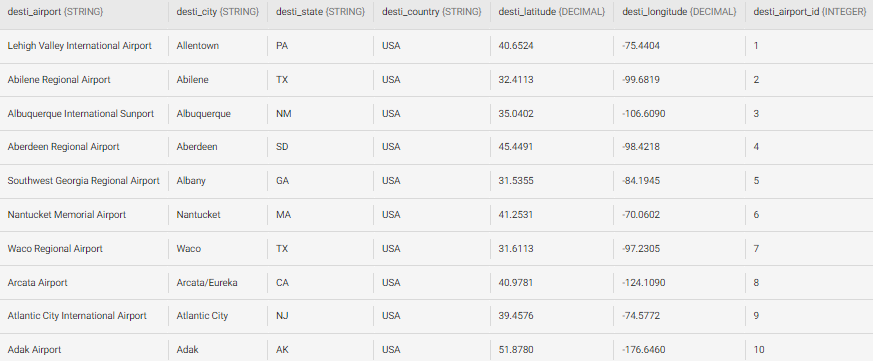
**  
configure Join Processor :-**

****

OUTPUT:

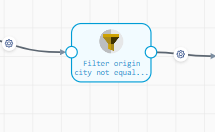
When you see the joined output from red border Indicates Origin table from JDBC table 1 and green border indicates table2 from origin of JDBC 2.





**FILTER PROCESSOR:**

**Add filter processor to the next of pipeline**

****

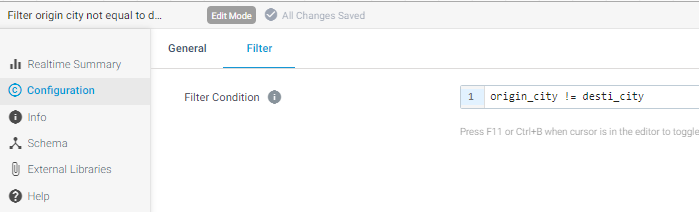
**Condition IS:  
origin city is not equal to destination city  
eg:origin =India at the same time not equal to Destination=India   
so when calculating destination between two country so not means same country.**

**Condition: Ensuring Distinct Origin and Destination= origin\_city !=desti\_city**

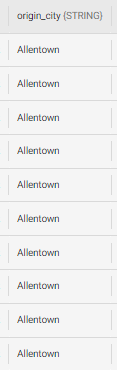
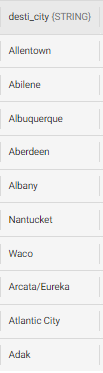
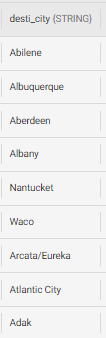
To maintain data integrity and accuracy, implement the following condition:

* Requirement: The origin city must not be equal to the destination city.
* Example:
  + If the origin is set to "India," the destination must not be "India" simultaneously.
  + This condition ensures that when calculating destinations between two countries, the origin and destination cannot be within the same country.

By enforcing this condition, you enhance the reliability of your data and prevent inconsistencies that may arise from having identical origin and destination locations.

****

**Output:  
inputdata : output :**

**** **** origin\_city!=desti\_city  

**Haversine Distance Calculation Function**

In the code snippet provided, a Scala function named Haversine is implemented to calculate the distance between two geographical points on the Earth's surface using the Haversine formula. The Haversine formula is commonly employed for measuring distances between two points specified by their latitude and longitude coordinates.

=====================================================================================

**SCALA Code**

=====================================================================================

**/\*\* Equatorial radius (WGS84) in meters \*/**

**//val EARTH\_RADIUS = 6378137d // km**

**val EARTH\_RADIUS = 3963.19d // miles**

**/\*\***

**\* Calculates distance, in R units**

**\***

**\* @param startLon - the longitude of the starting point**

**\* @param startLat - the latitude of the starting point**

**\* @param endLon - the longitude of the ending point**

**\* @param endLat - the latitude of the ending point**

**\* @return - the calculated distance in the specified units**

**\*/**

**def Haversine(startLon: Double, startLat: Double, endLon: Double, endLat: Double): Double = {**

**val dLat = math.toRadians(endLat - startLat)**

**val dLon = math.toRadians(endLon - startLon)**

**val lat1 = math.toRadians(startLat)**

**val lat2 = math.toRadians(endLat)**

**val a =**

**math.sin(dLat / 2) \* math.sin(dLat / 2) +**

**math.sin(dLon / 2) \* math.sin(dLon / 2) \* math.cos(lat1) \* math.cos(lat2)**

**val c = 2 \* math.atan2(math.sqrt(a), math.sqrt(1 - a))**

**EARTH\_RADIUS \* c**

**}**

**// Register the UDF for usage as a Spark SQL function**

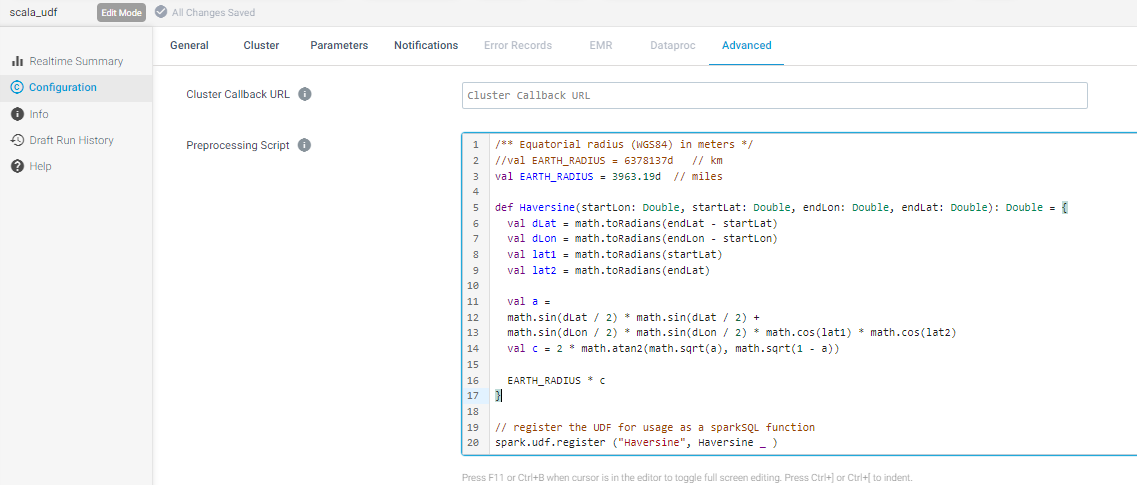
**spark.udf.register("Haversine", Haversine \_)**

**=====================================================================================**

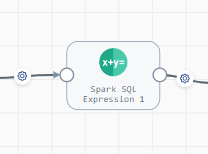
* **Equatorial Radius:** The equatorial radius of the Earth, defined by the WGS84 model, is set as **EARTH\_RADIUS**. In this context, it is specified in miles.
* **Haversine Function:** The **Haversine** function takes the longitude and latitude coordinates of two points and calculates the distance between them using the Haversine formula.
* **Spark SQL UDF Registration:** The function is registered as a User-Defined Function (UDF) in Spark SQL, allowing it to be seamlessly integrated into Spark SQL queries for distance calculations.

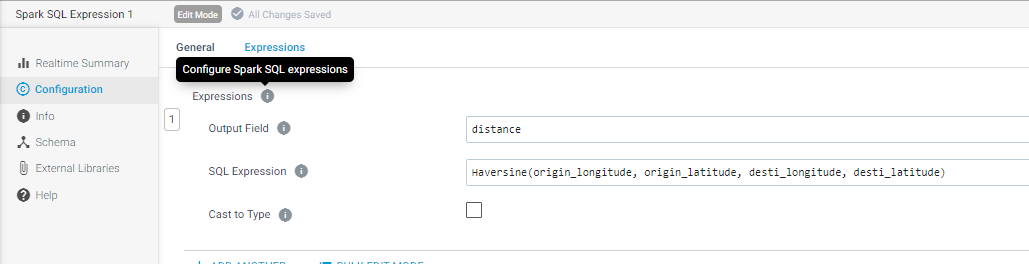
This code serves as a valuable tool for calculating distances between geographical points, especially in applications involving Spark SQL processing.

**Copy the Above Scala Code And paste into Stremsets Control Hub :-  
  
Click the blank canva and configuration go to Advanced Tab :-**

****

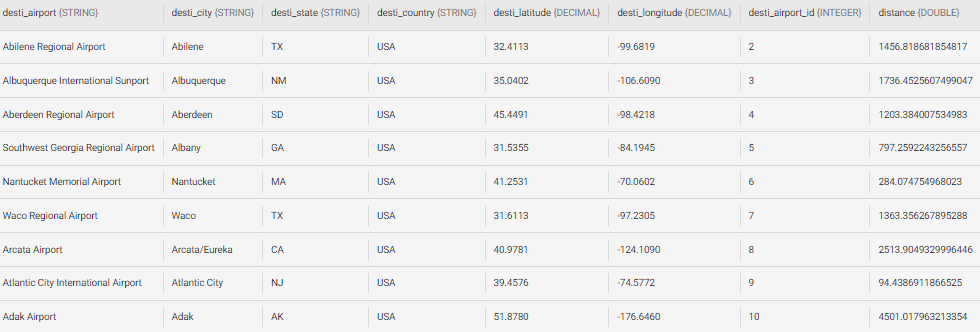
**Add Another Spark Sql Expression in streamsets control Hub :**

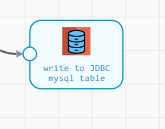
****

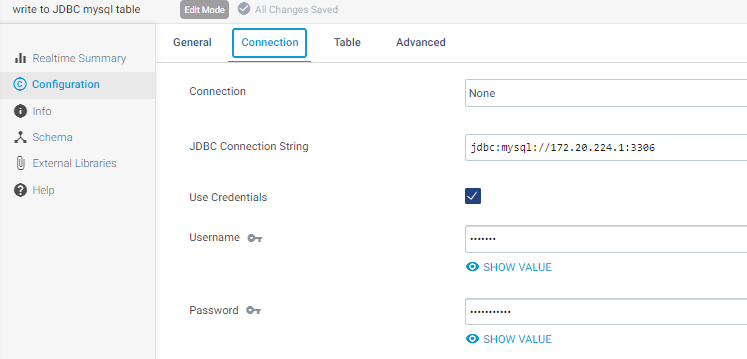
****

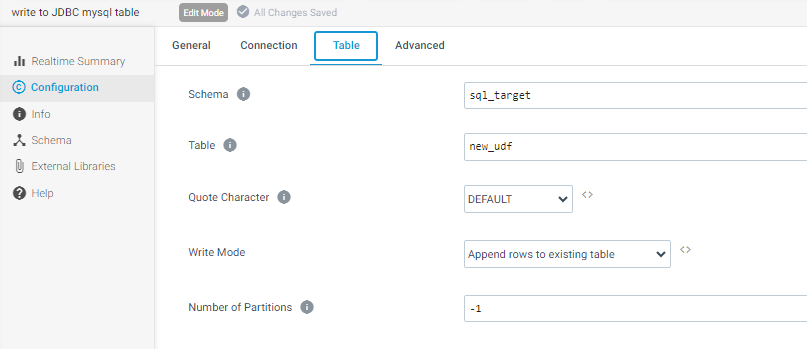
Output :

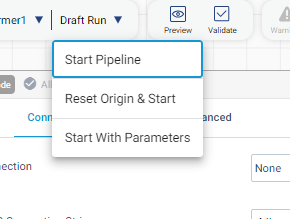


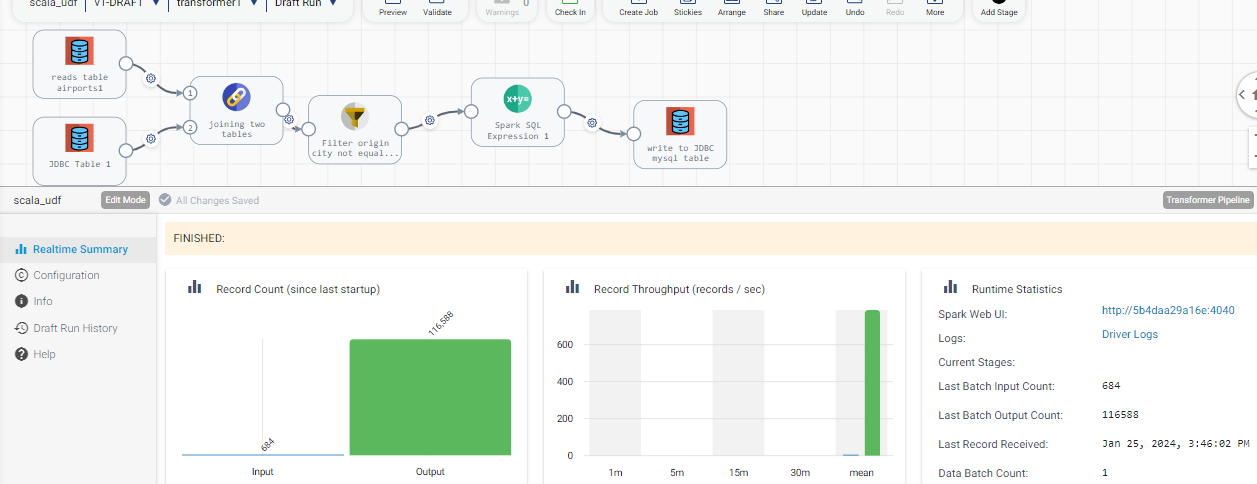
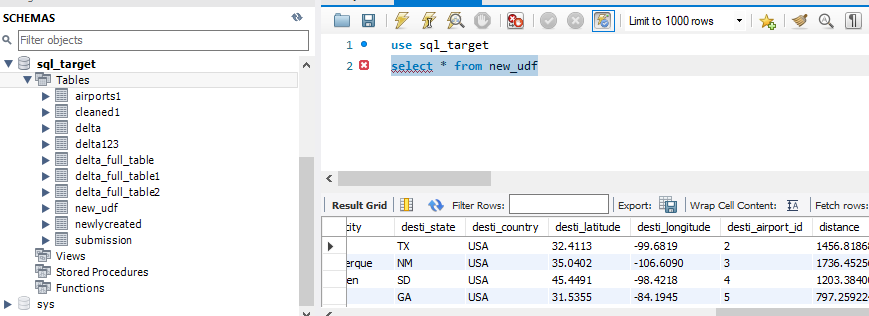


**Final add JDBC DESTINATION :**  
  






**Now Run the Pipeline :-**  
  


  
  
**Go to MySQL and Check Moved Are are not :**

---------------------\*\*\*\*\*\*\*\*\*\*\*\*------------------\*\*\*\*\*\*\*\*-------------------\*\*\*\*\*\*\*--------------\*\*\*\*\*\*\*\*\*--------

You can verified all data written in Mysql and New Distance column is generated means you Sucessfully Completed the Pipeline .  
  
 ================================THANKYOU============================================