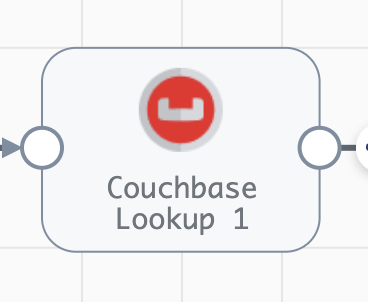
**PROCESSOR NAME: Couchbase Lookup**



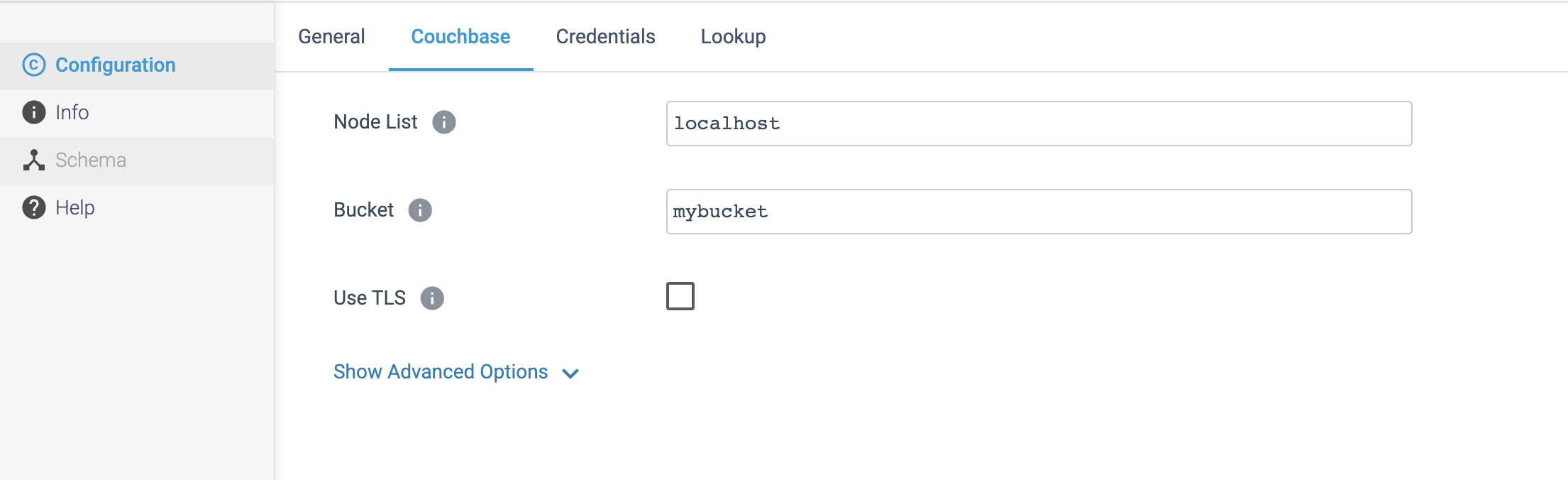
**Engine Type**: Data Collector

**Pipeline Name**: D\_MongoAtlas\_orgn\_cssndra\_Dest\_pipeline\_CredStr

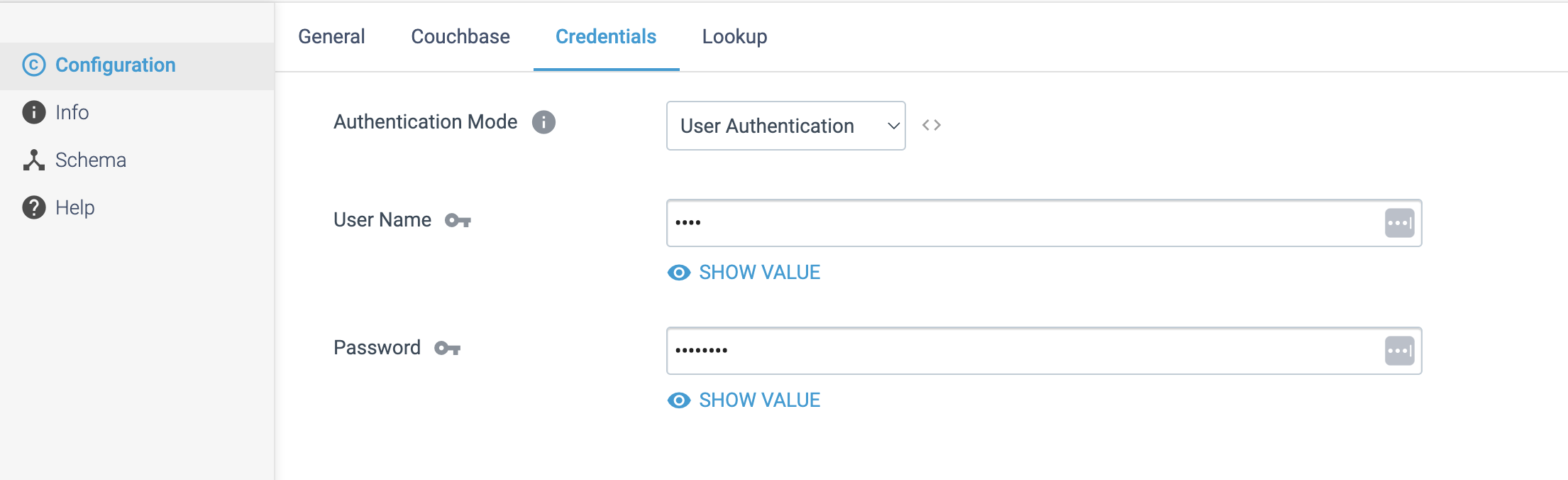
**Description**: The **Couchbase Lookup** processor looks up documents in Couchbase Server and returns values to fields in the record.

We need to configure the **Couchbase Lookup** processor with the following steps.

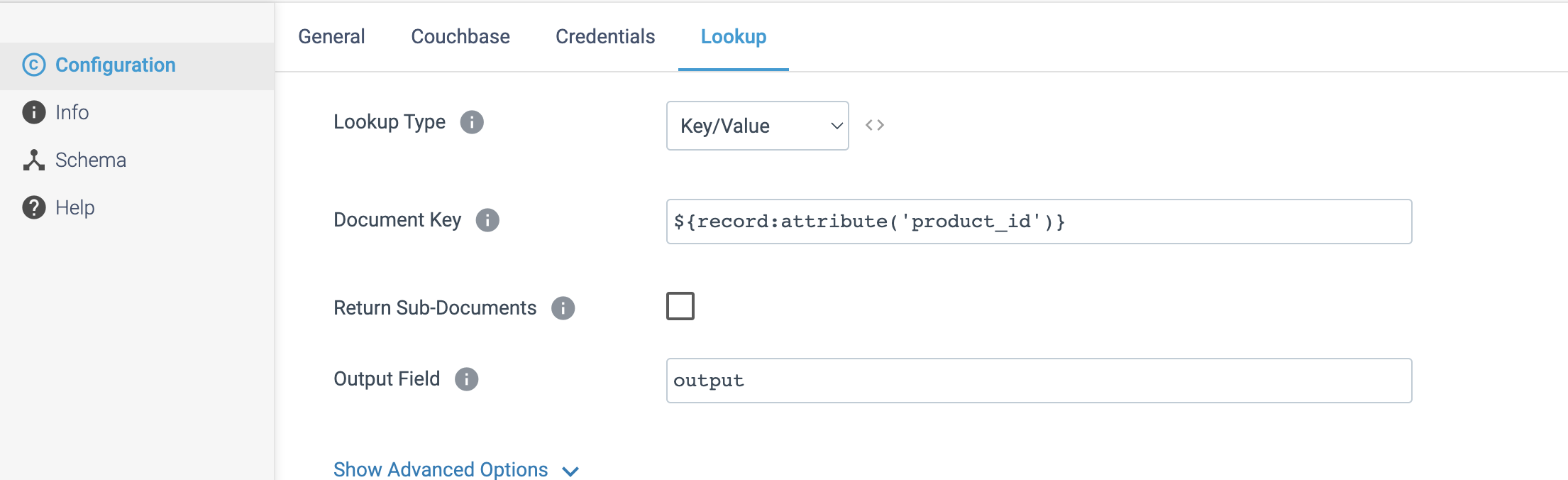
Step 1: Click on the **Couchbase Lookup** processor to enter connection information, such as the nodes and bucket to connect to, as well as timeout properties for the connection.



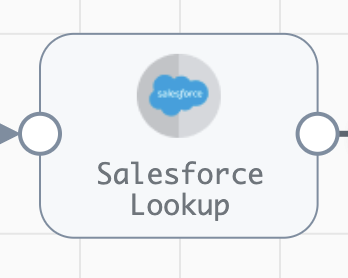
Step 2: Click on the **Credentials** tab to enter information to authenticate with Couchbase Server.



Step 3: Click on the **Lookup** tab to configure the following properties.



**PROCESSOR NAME: Salesforce Lookup**



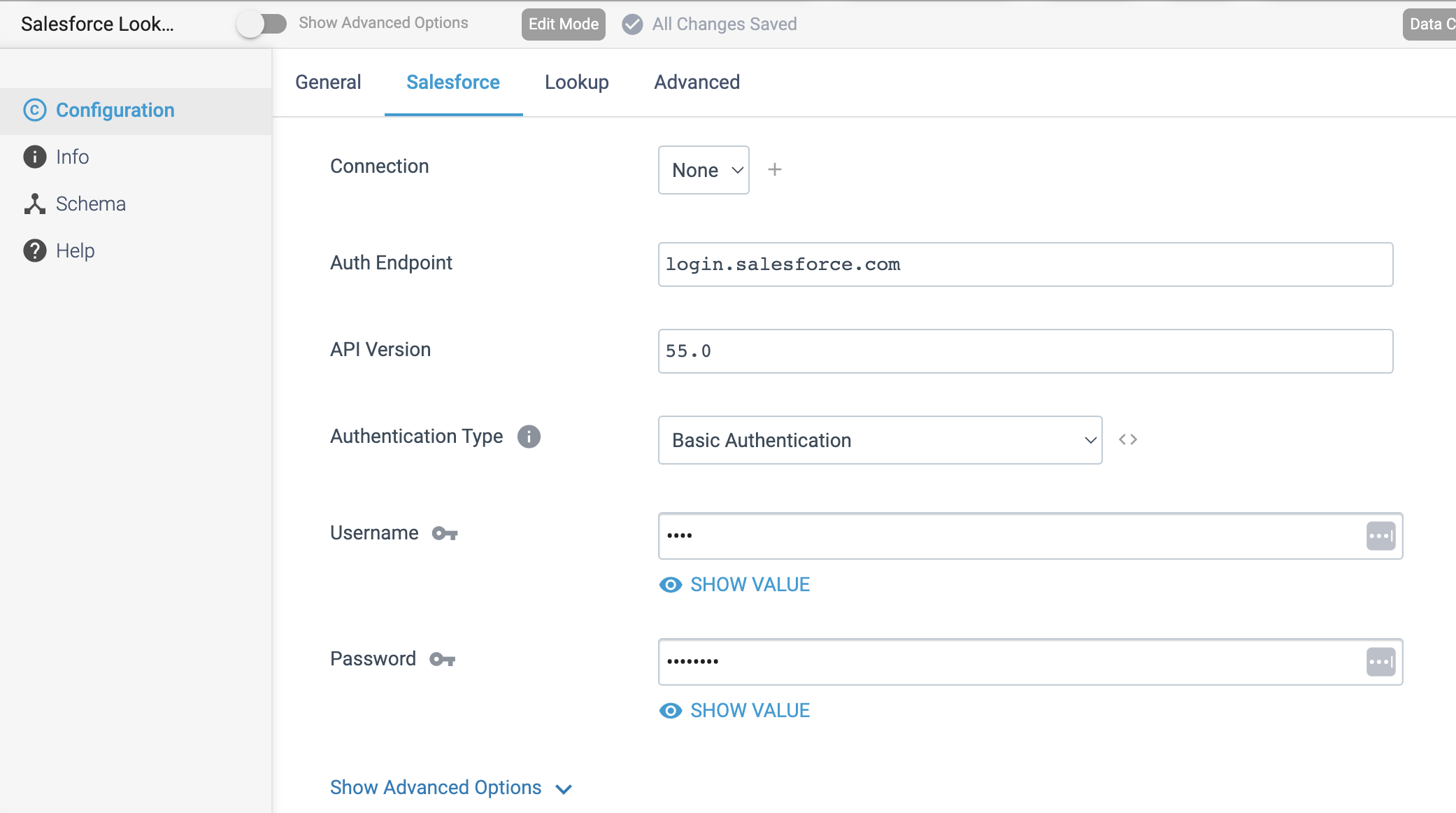
**Engine Type**: Data Collector

**Pipeline Name**: D\_Salesfrcebulk\_orgn\_Dest\_Pipeline\_Cred

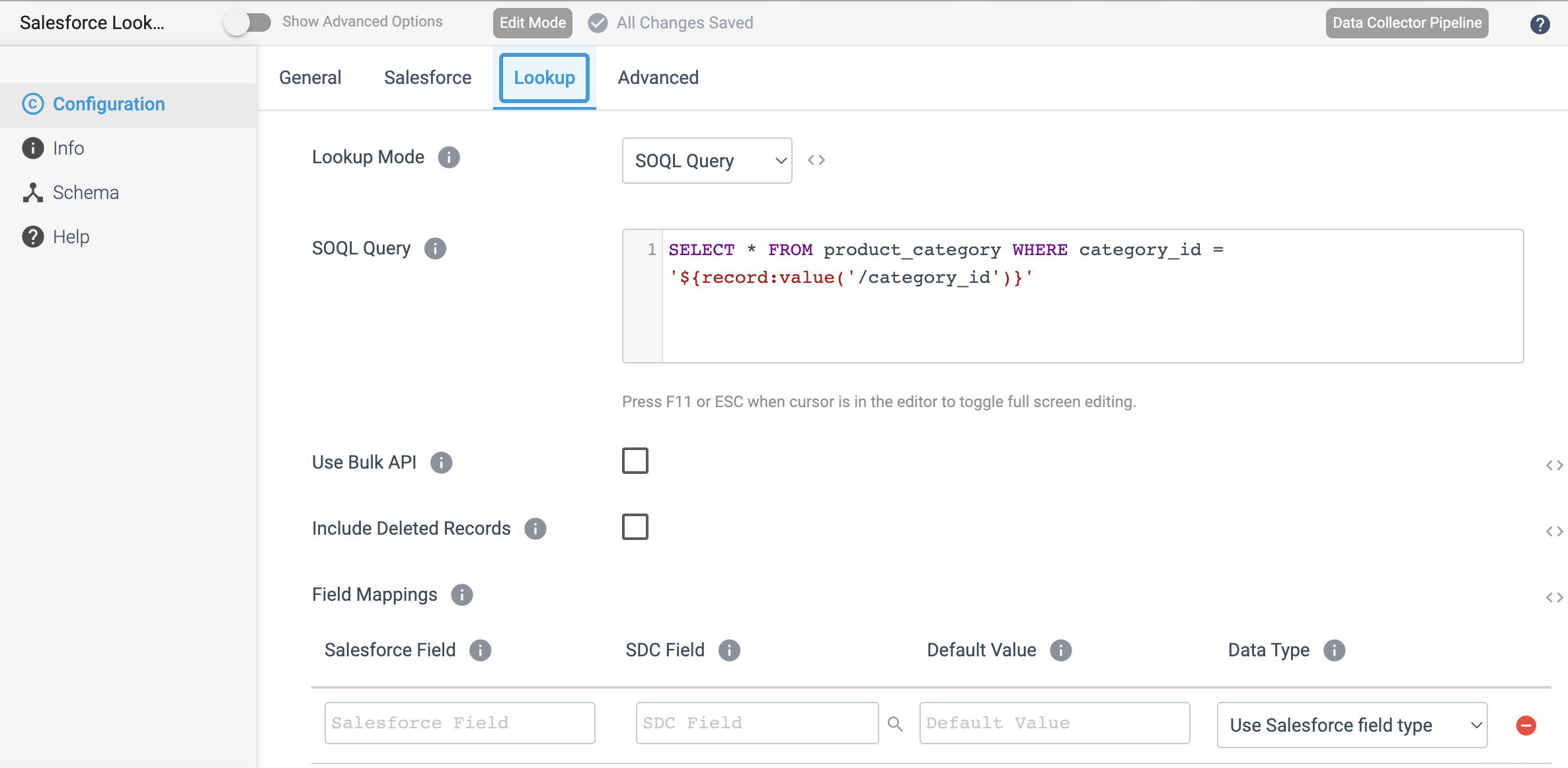
**Description**: The **Salesforce Lookup** processor performs lookups in a Salesforce object and passes the lookup values to fields. The processor can perform lookups with the **SOAP** or **Bulk API**.

We need to configure the **Salesforce Lookup** processor with the following steps.

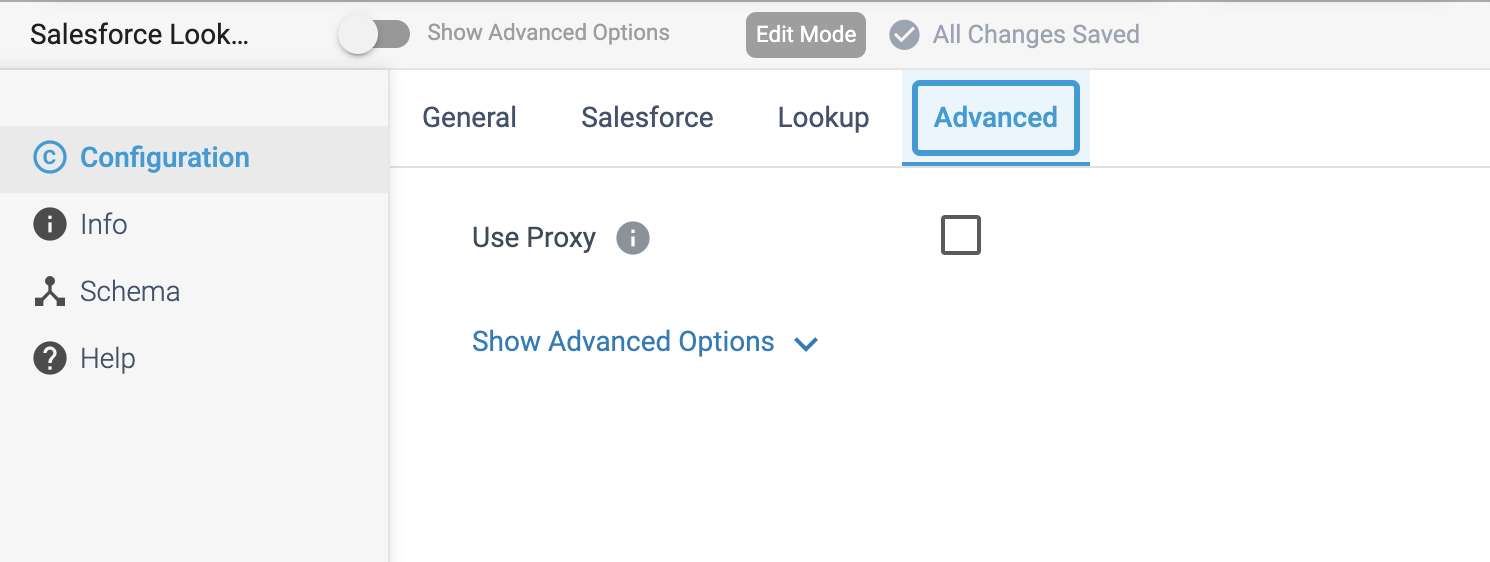
Step 1: Click on the **Salesforce Lookup** processor to define connection information that the processor uses to connect to Salesforce, including the Salesforce API version and the authentication to use. We can also use a connection to configure the processor.



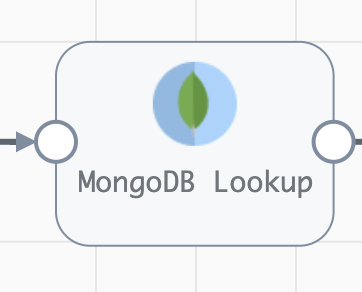
Step 2: Click on the **Lookup** tab to provide the SOQL Query details.



Step 3: Click on the **Advanced** tab to optionally use an HTTP proxy to connect to Salesforce.



**PROCESSOR NAME: MongoDB Lookup**



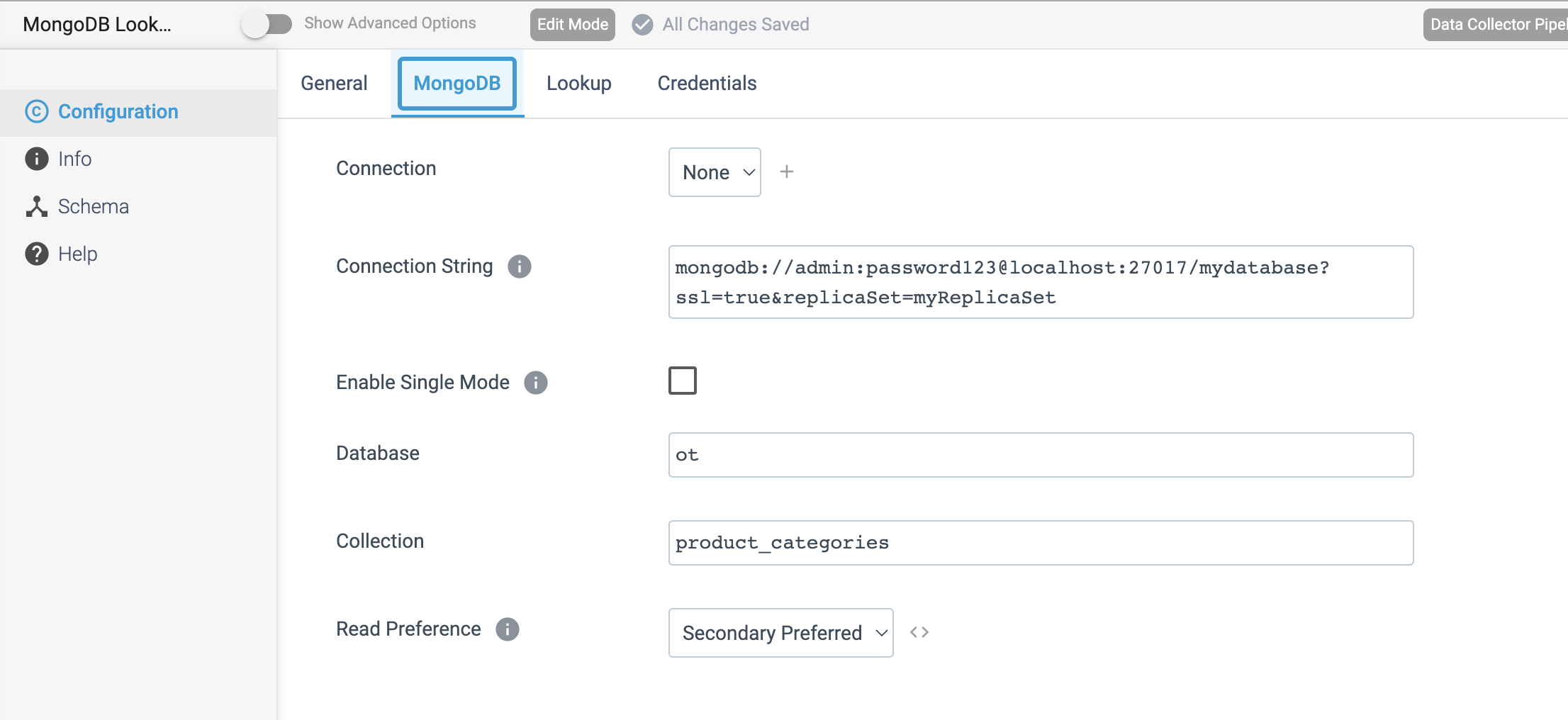
**Engine Type**: Data Collector

**Pipeline Name**: D\_MongoDB\_Orgn\_Dest\_Pipeline\_Cred

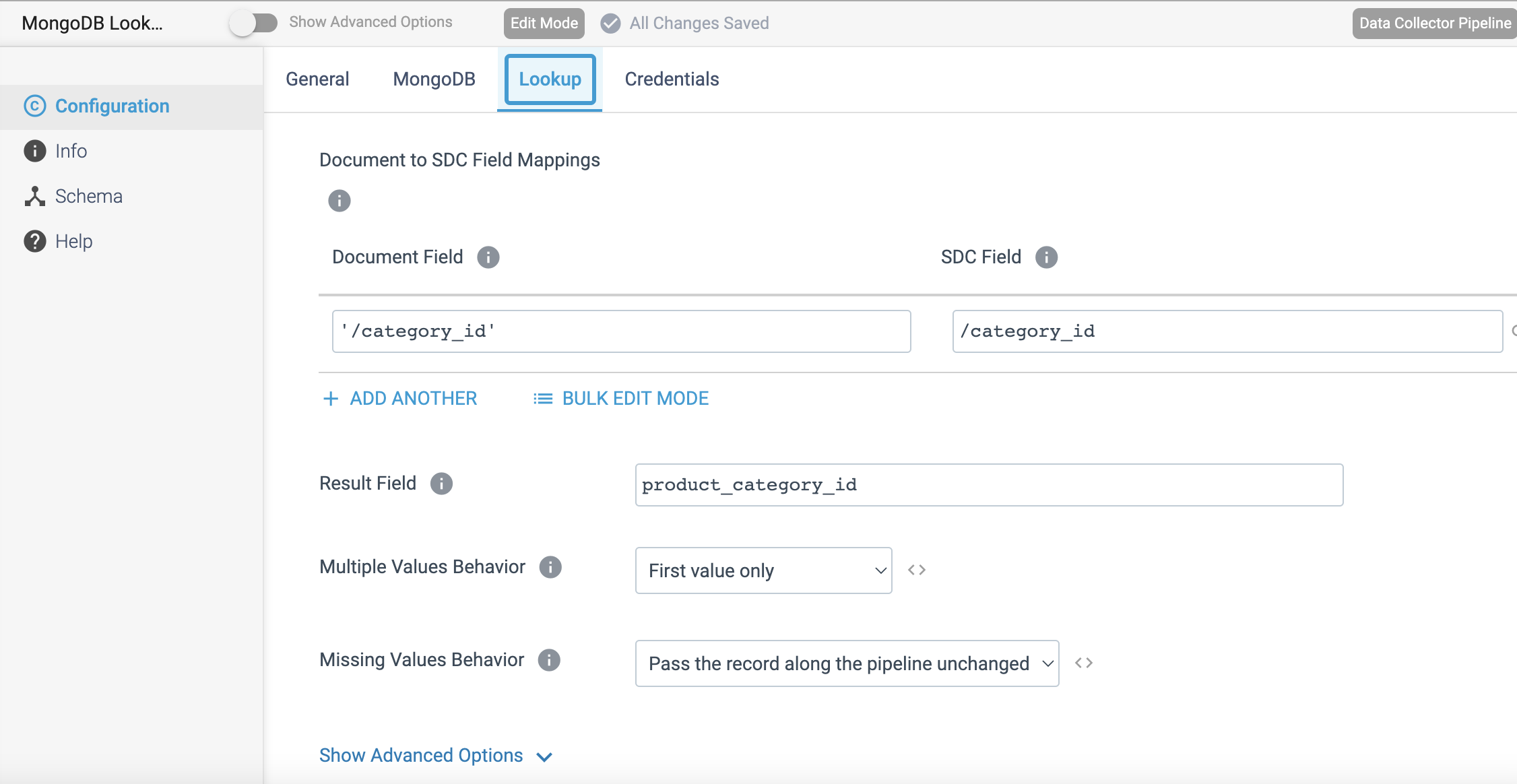
**Description**: The **MongoDB Lookup** processor performs lookups in MongoDB and passes all values from the returned document to a new list-map field in the record.

We need to configure the MongoDB Lookup processor in a following way.

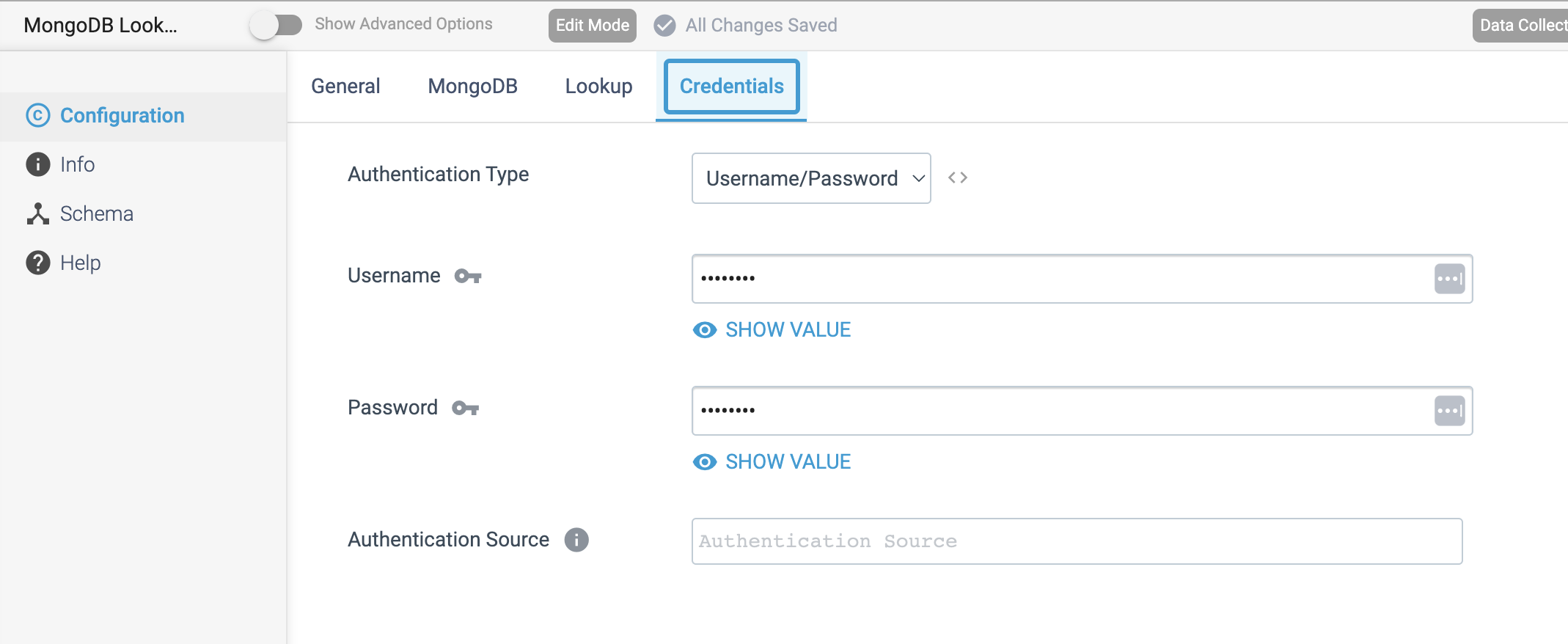
Step 1: Click on the **MongoDB Lookup** processor to define connection information, such as the **connection string**, **database** name and **collection** name.



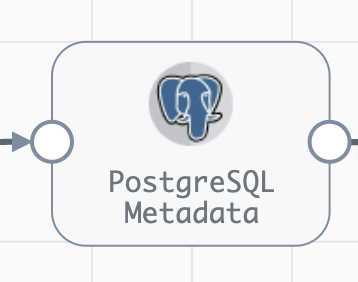
Step 2: Click on the **Lookup** tab to configure the fields to look up and the field for the return values.



Step 3: Click on the **Credentials** tab to provide the credential details.



**PROCESSOR NAME: PostgreSQL Metadata**



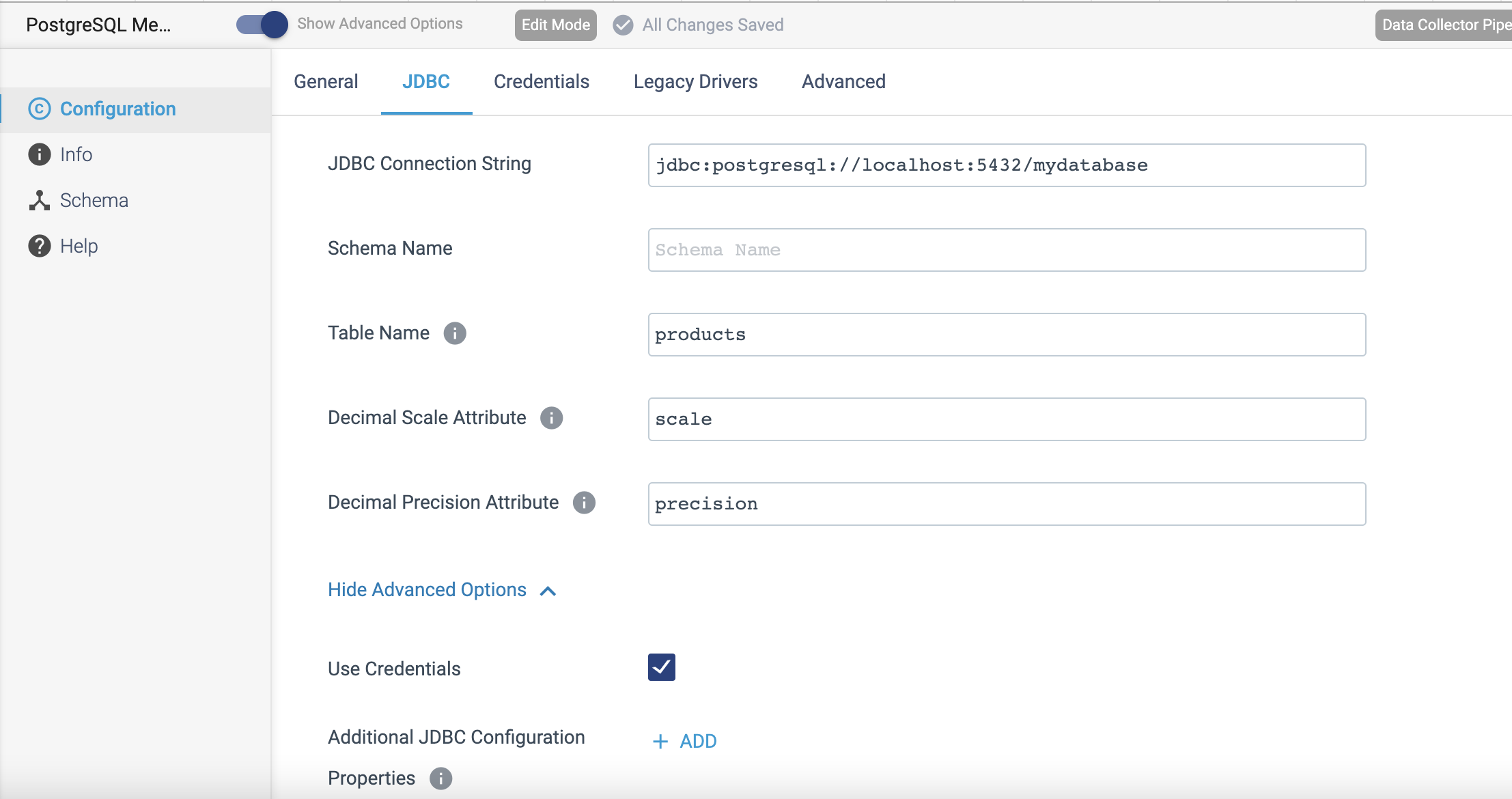
**Engine Type**: Data Collector

**Pipeline Name**: D\_Postgrs\_Orgn\_Azre\_LkeGen2\_Dest\_Pipelne\_CredStr

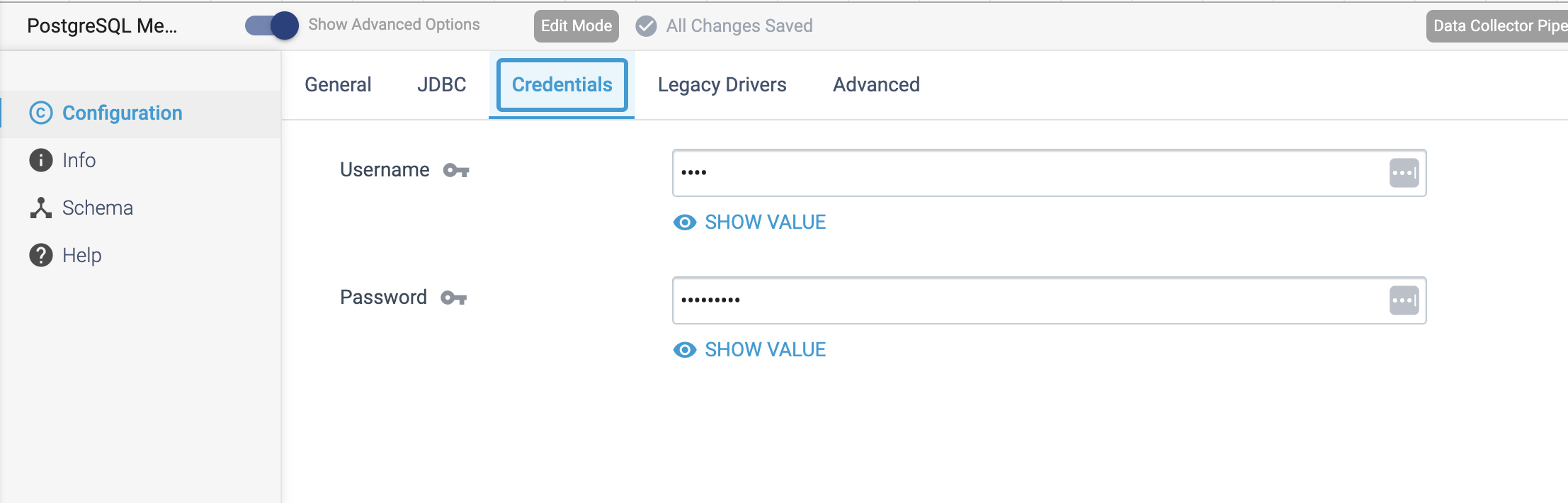
**Description**: The **PostgreSQL Metadata** processor determines the PostgreSQL table where each record should be written, compares the record structure against the table structure, then creates or alters the table as needed.

We need to configure the **PostgreSQL Metadata** processor in a following way.

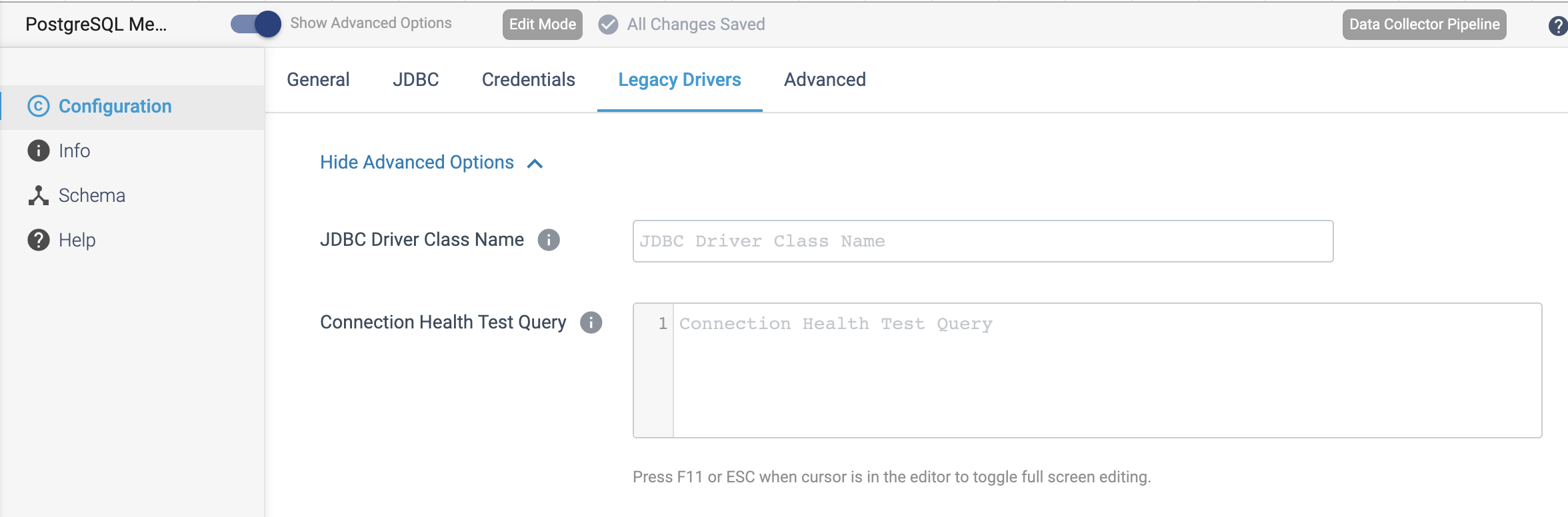
Step 1: Click on the **PostgreSQL Metadata** processor to configure the connection details, schema name and table name.



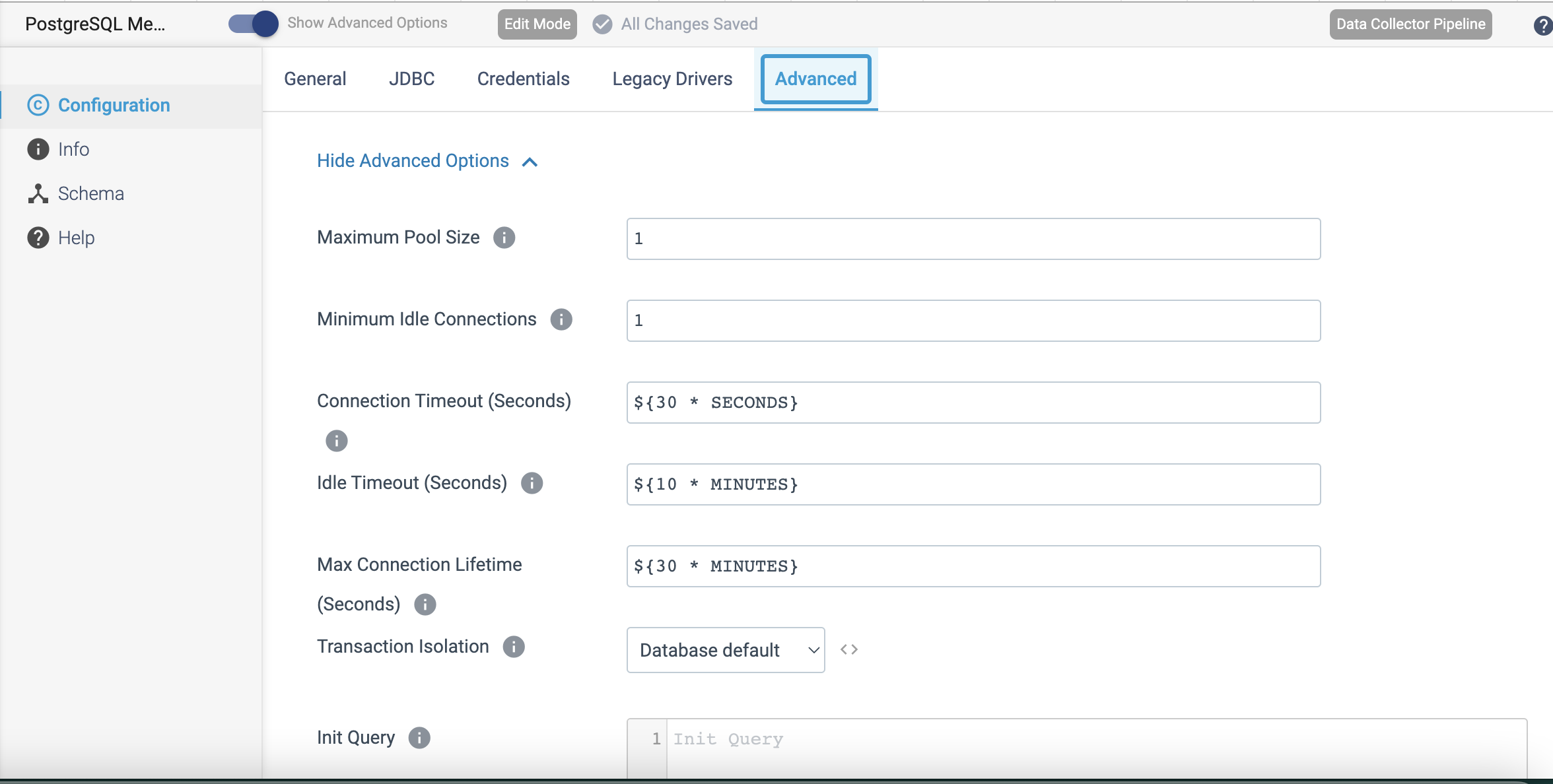
Step 2: Click on the **credentials** tab to provide the credential details.



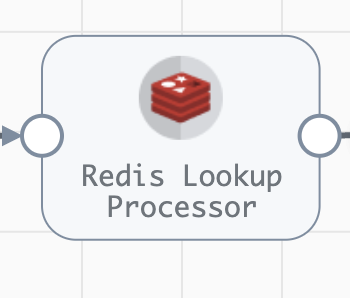
Step 3: When using JDBC versions older than 4.0, on the **Legacy Drivers** tab, optionally configure the following properties.



Step 4: Click on the **Advanced** tab to configure optionally.



PROCESSOR NAME: Redis Lookup Processor



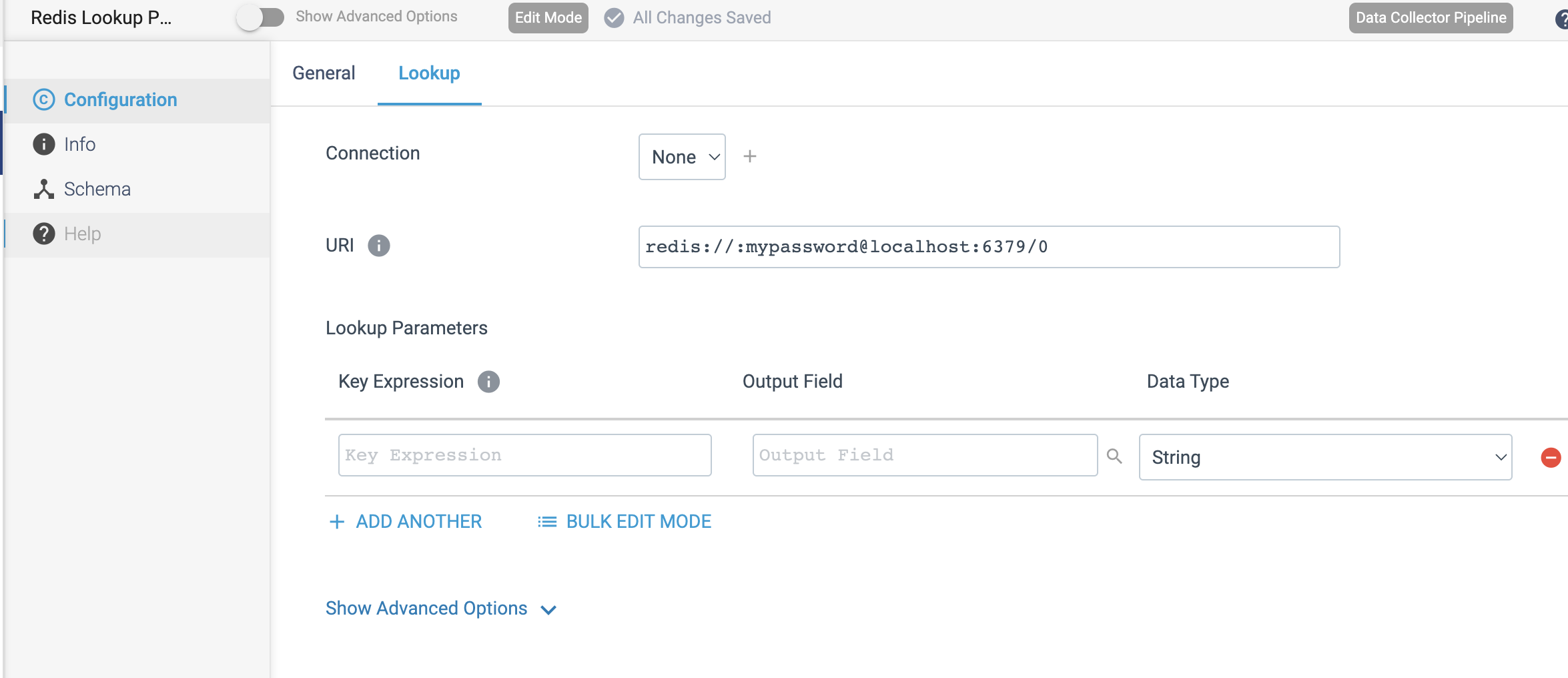
**Engine Type**: Data Collector

**Pipeline Name**: D\_Redis\_orgn\_Dest\_Pipeline\_URI

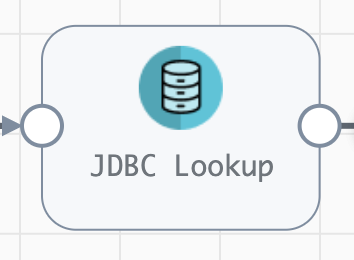
**Description**: The **Redis Lookup** processor performs key-value lookups in Redis and passes the lookup values to fields.

We need to configure the **Redis Lookup** processor in a following way.

Step 1: Click on the **Redis Lookup** processor to specify whether the processor performs a bulk lookup of all keys in a batch, or performs an individual lookup of each key in a record.



PROCESSOR NAME: JDBC Lookup



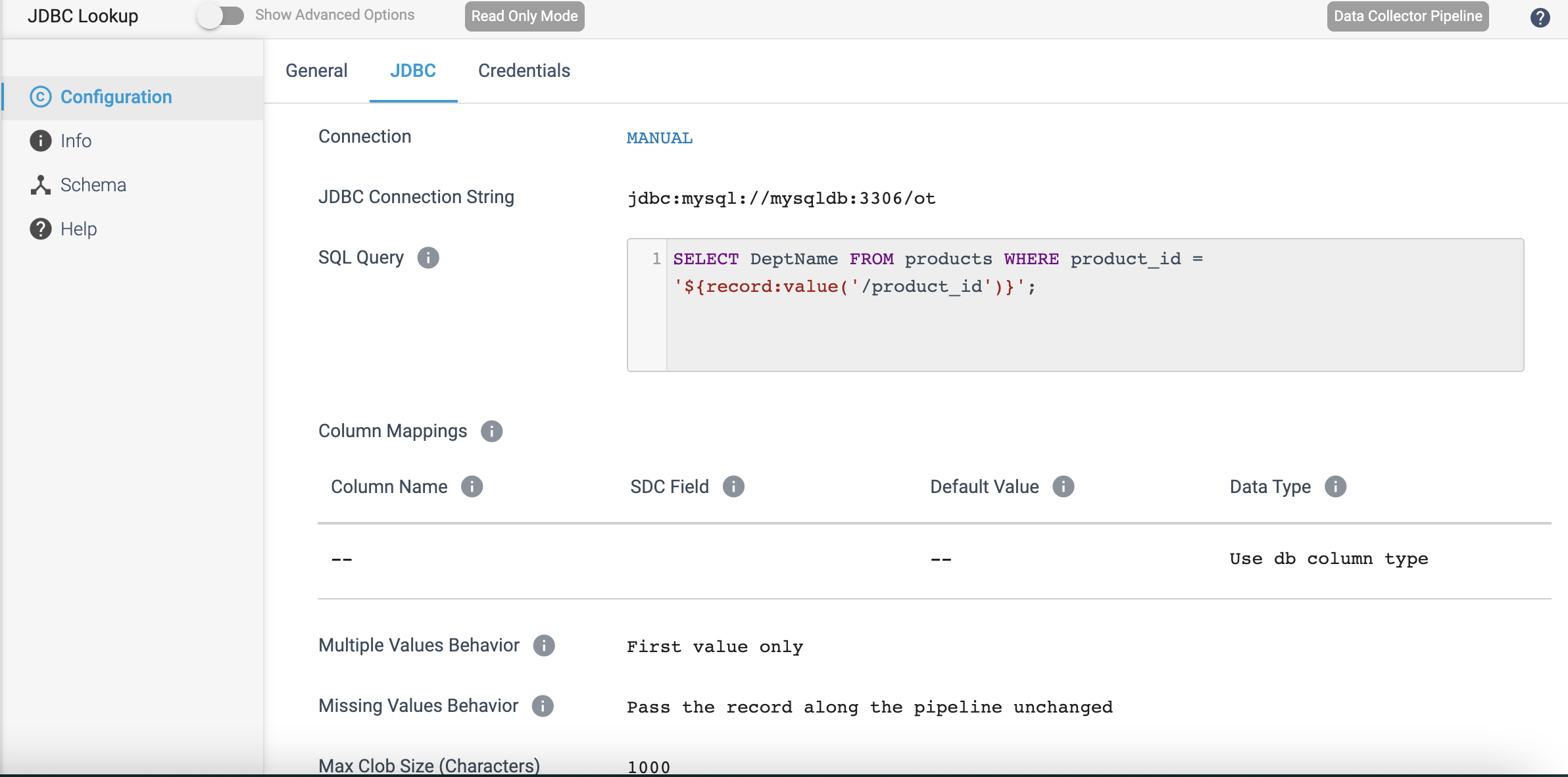
**Engine Type**: Data Collector

**Pipeline Name**: D\_AmazonS3\_Origin\_S3\_Dest\_Pipeline\_CredStr

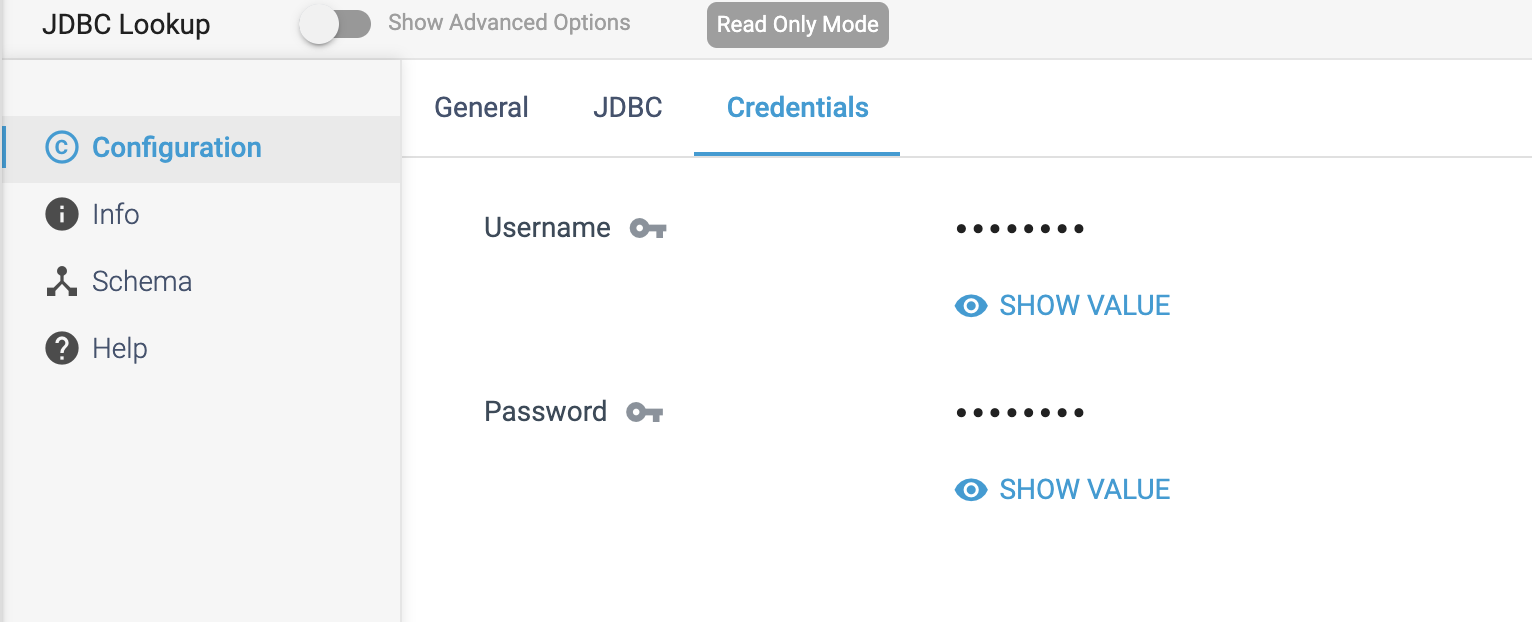
**Description**: The **JDBC Lookup** processor uses a JDBC connection to perform lookups in a database table and pass the lookup values to fields.

We need to configure the **JDBC Lookup** processor in a following way.

Step 1: Click **JDBC Lookup** processor to specify connection information and custom JDBC configuration properties to determine how the processor connects to the database.



Step 2: Click on the **Credentials** tab to provide credentials details.



PROCESSOR NAME: JDBC Tee



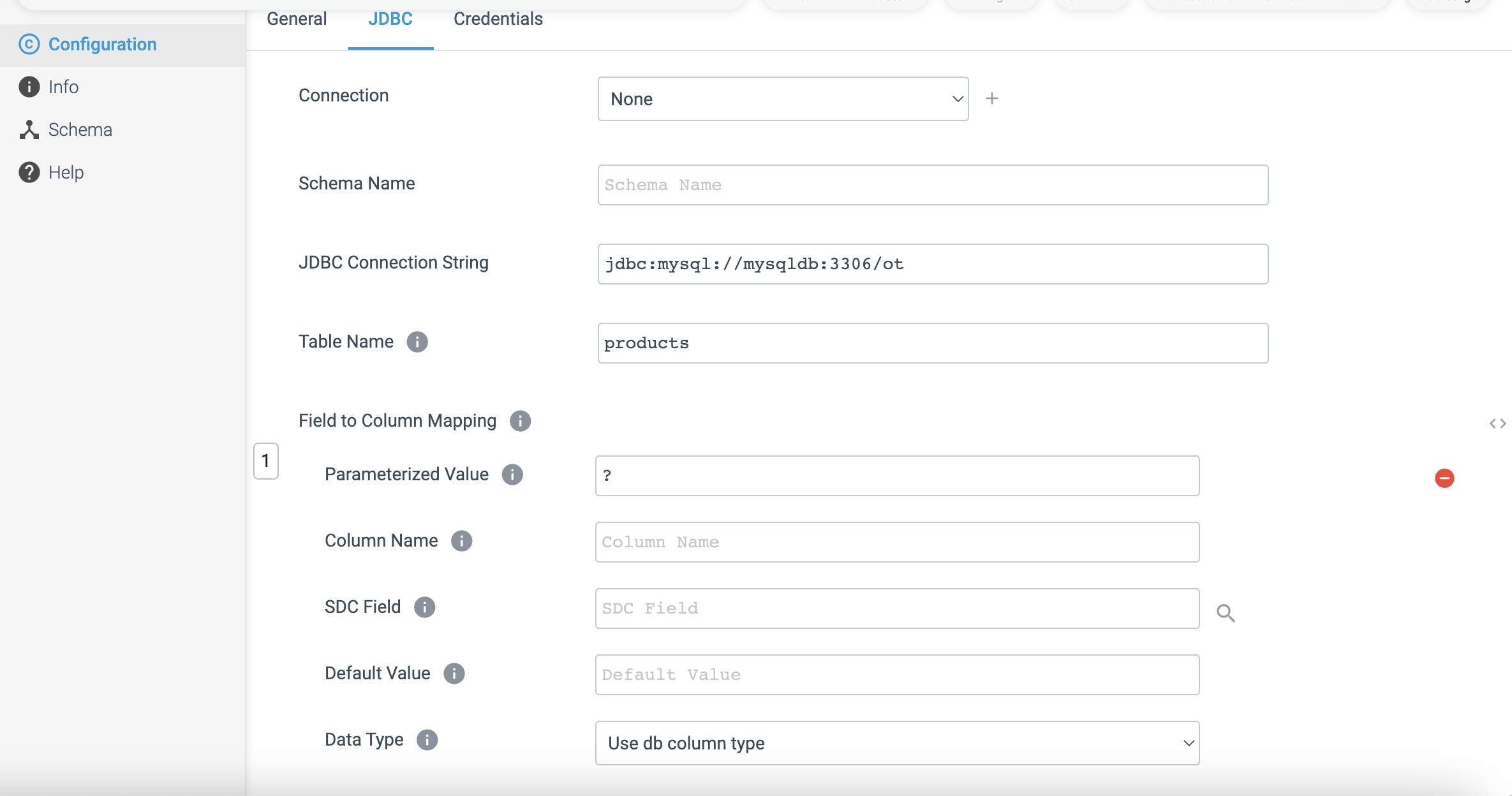
**Engine Type**: Data Collector

**Pipeline Name**: D\_AmazonSQS\_Origin\_JDBC\_Dest\_Pipeline\_CredStr

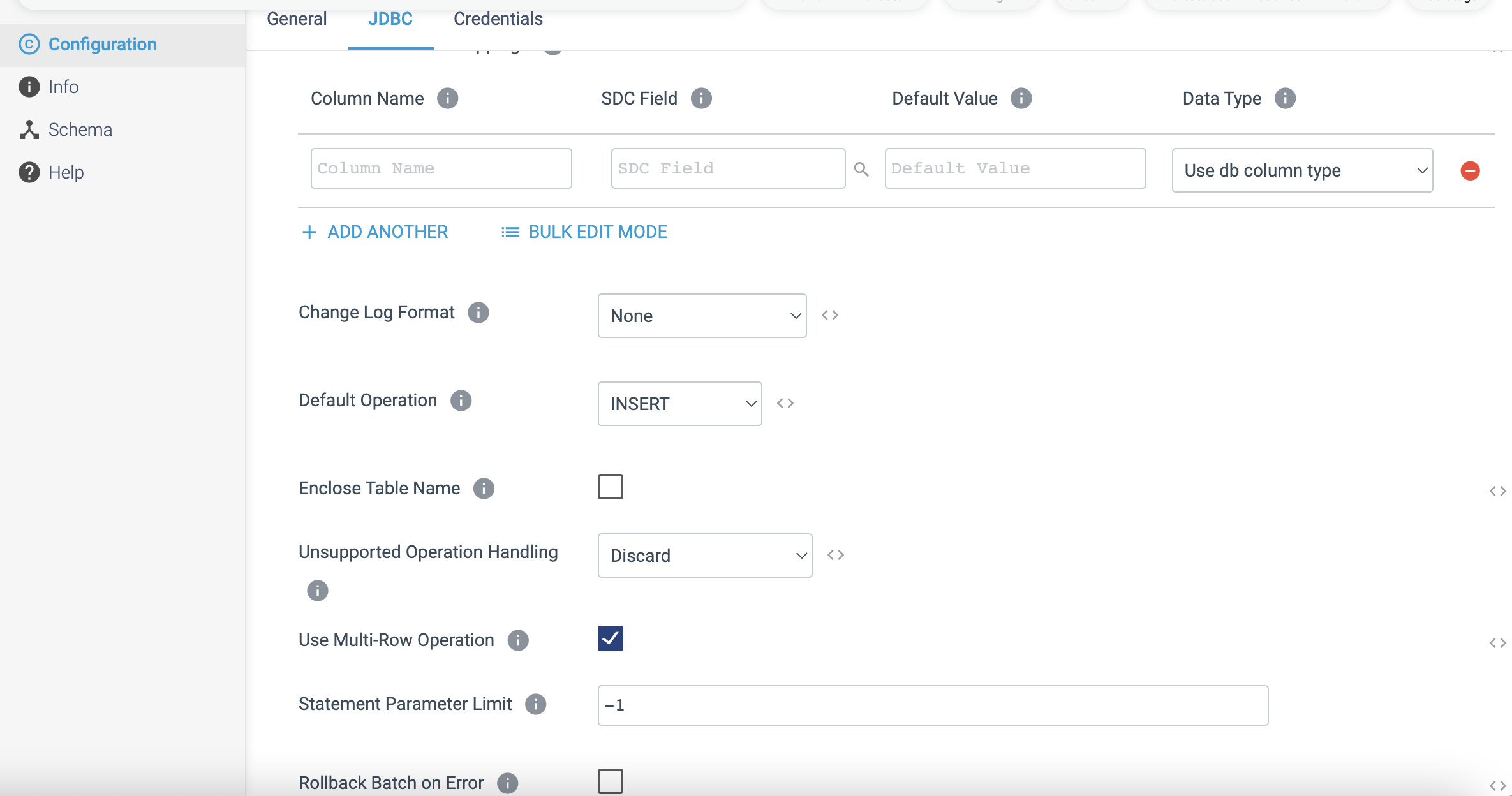
**Description**: The **JDBC Tee** processor uses a JDBC connection to write data to a MySQL or PostgreSQL database table, and then pass generated database column values to fields.

We need to configure the **JDBC Tee** processor in a following way.

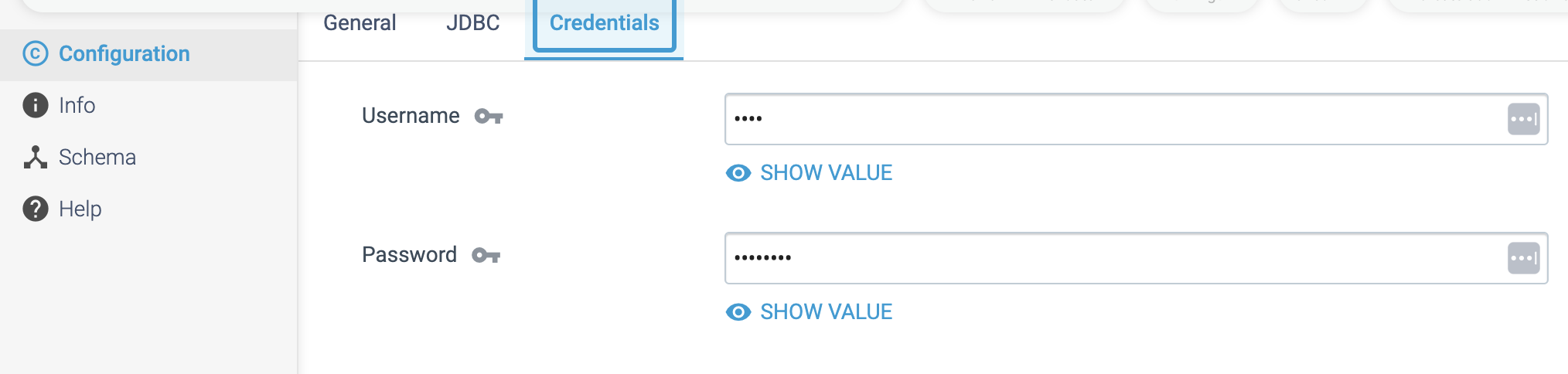
Step 1: Click on the **JDBC Tee** processor to specify **connection** information to the MySQL or PostgreSQL database, table name, and optionally define field mappings.



By default, the processor writes data to the table based on the matching field names. We can override the default field mappings by defining specific mappings.



Step 2: Click on the **Credentials** tab to provide the credential details.



PROCESSOR NAME: Http Client



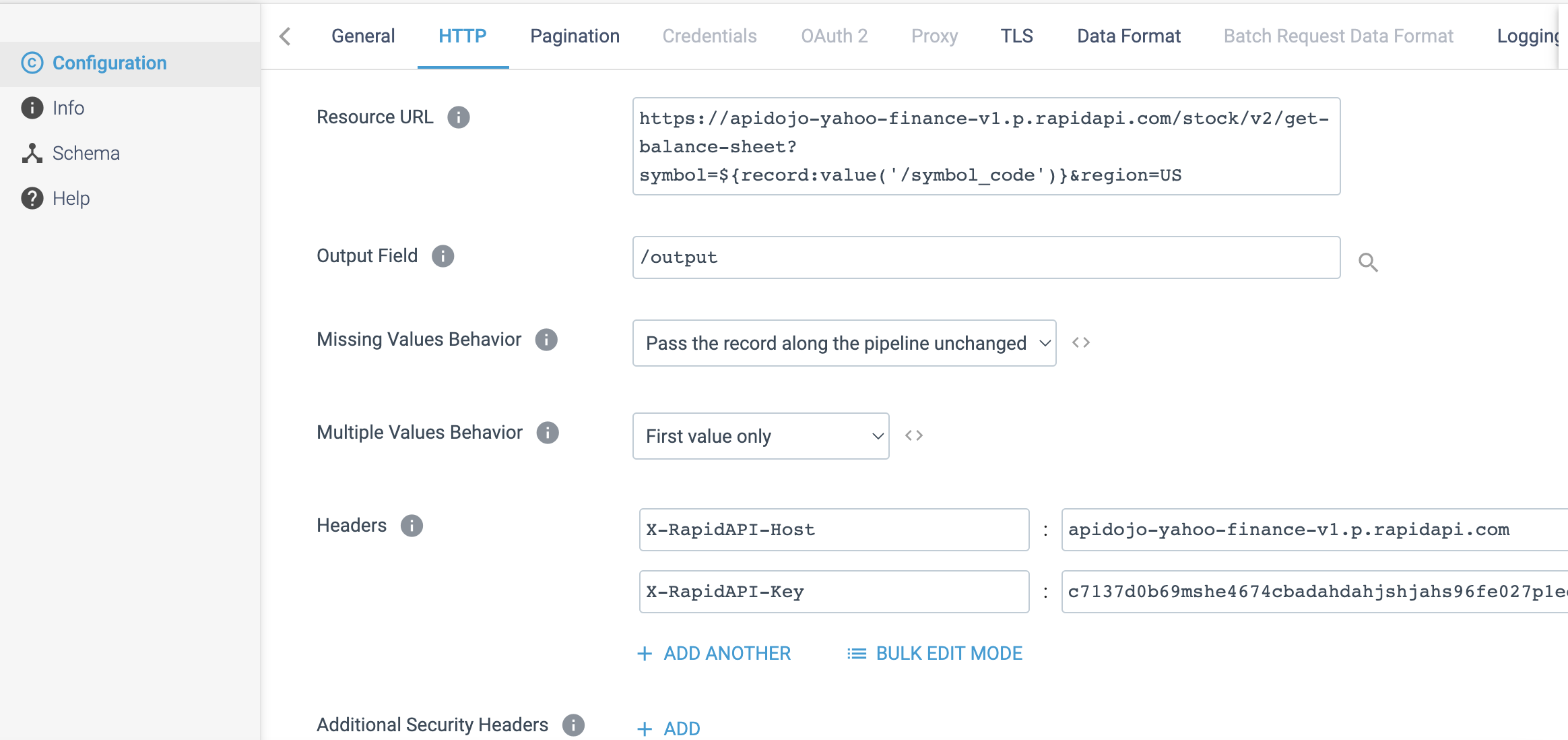
**Engine Type**: Data Collector

**Pipeline Name**: D\_JDBC\_orgn\_Dest\_Snglestre\_Pipeline\_CredStr\_URL

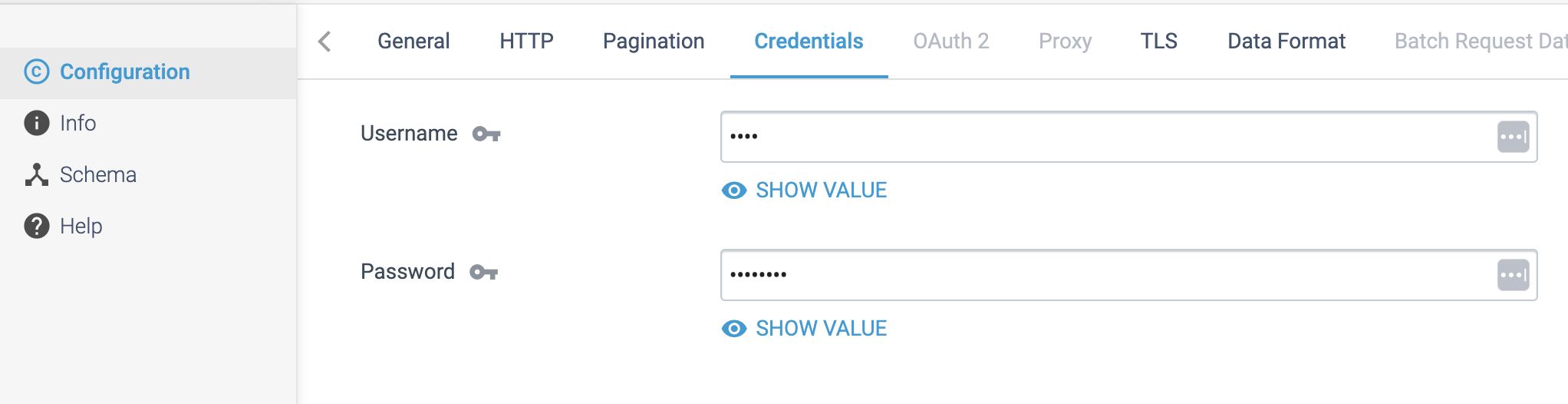
**Description**: The **HTTP Client** processor sends requests to an HTTP resource URL and writes responses to records.

We need to configure the **HTTP Client** processor in a following way.

Step 1: Click on the **HTTP Client** processor to define the resource URL, optional headers, and method to use. For some methods, we need to specify the request body and default content type.



Step 2: Click on the **Credential** tab to provide the credential details based on the Authentication Type selected in Http tab.



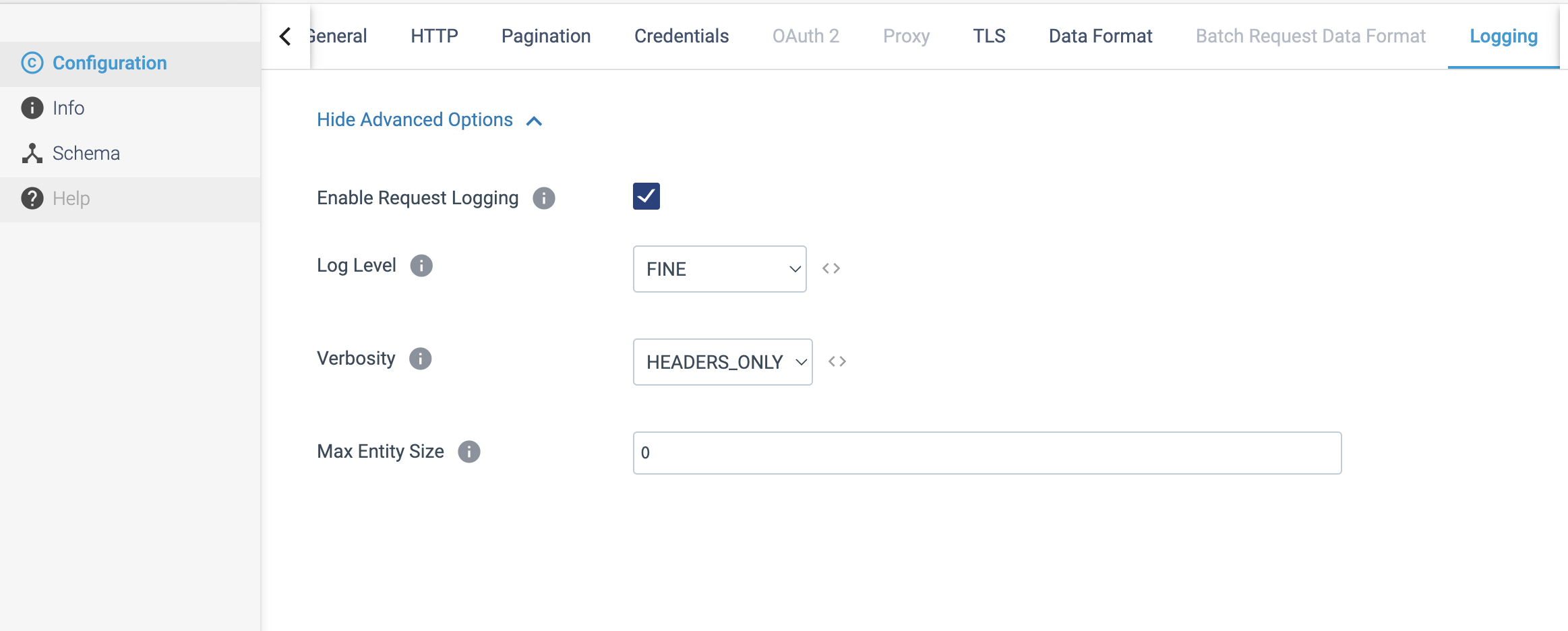
We can also configure the processor to use the **OAuth 2** protocol to connect to an **HTTP** service.

We can optionally use an **HTTP proxy** and configure SSL/TLS properties.

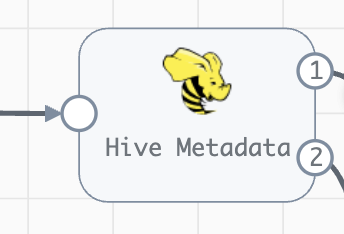
Step 3: Click on the **Data Format** tab to define the output data format.



Step 4: On the **Logging** tab, configure the following properties to log request and response data.



PROCESSOR NAME: Hive Metadata



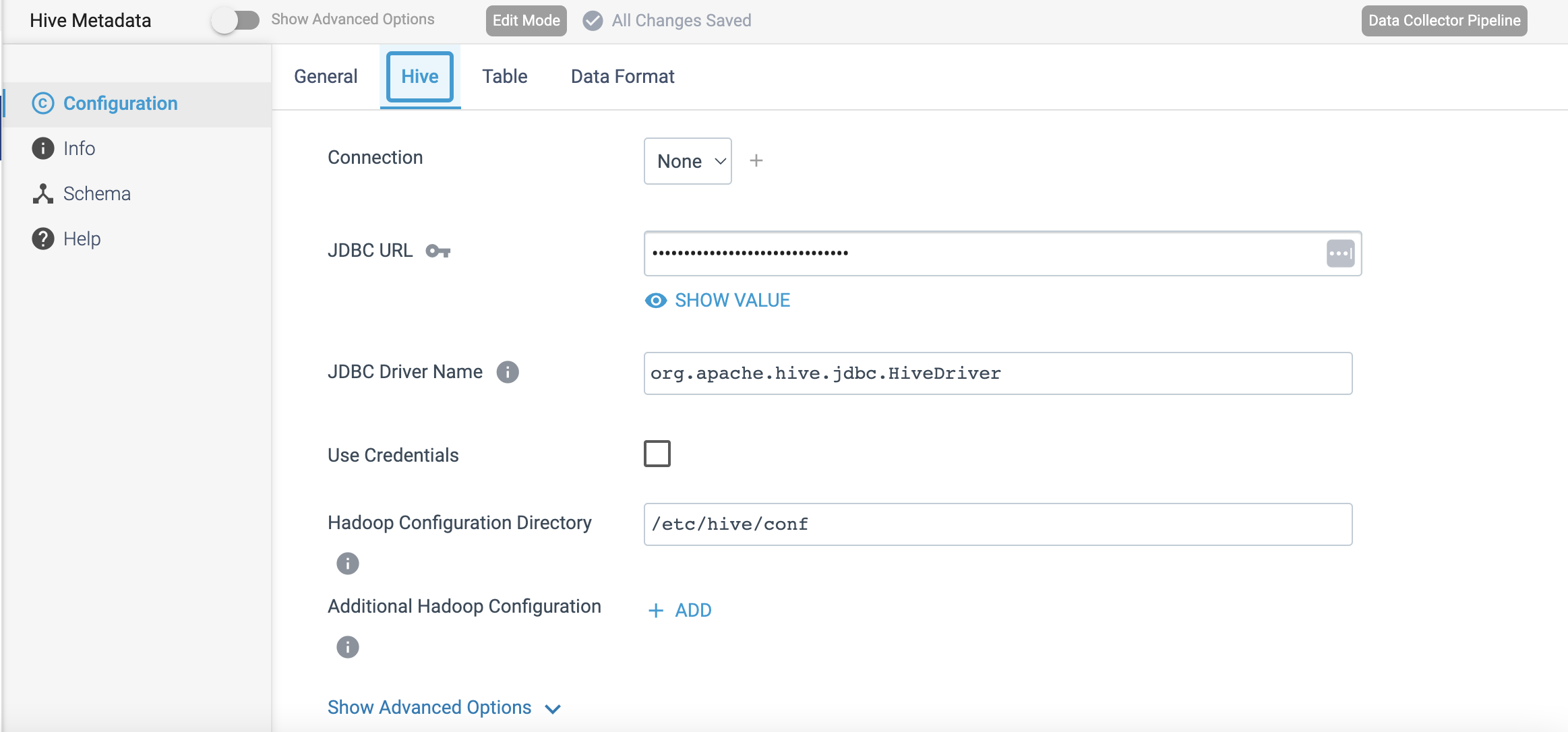
**Engine Type**: Data Collector

**Pipeline Name**: D\_JDBC\_Orgn\_Dest\_Pipeline\_Cred\_All

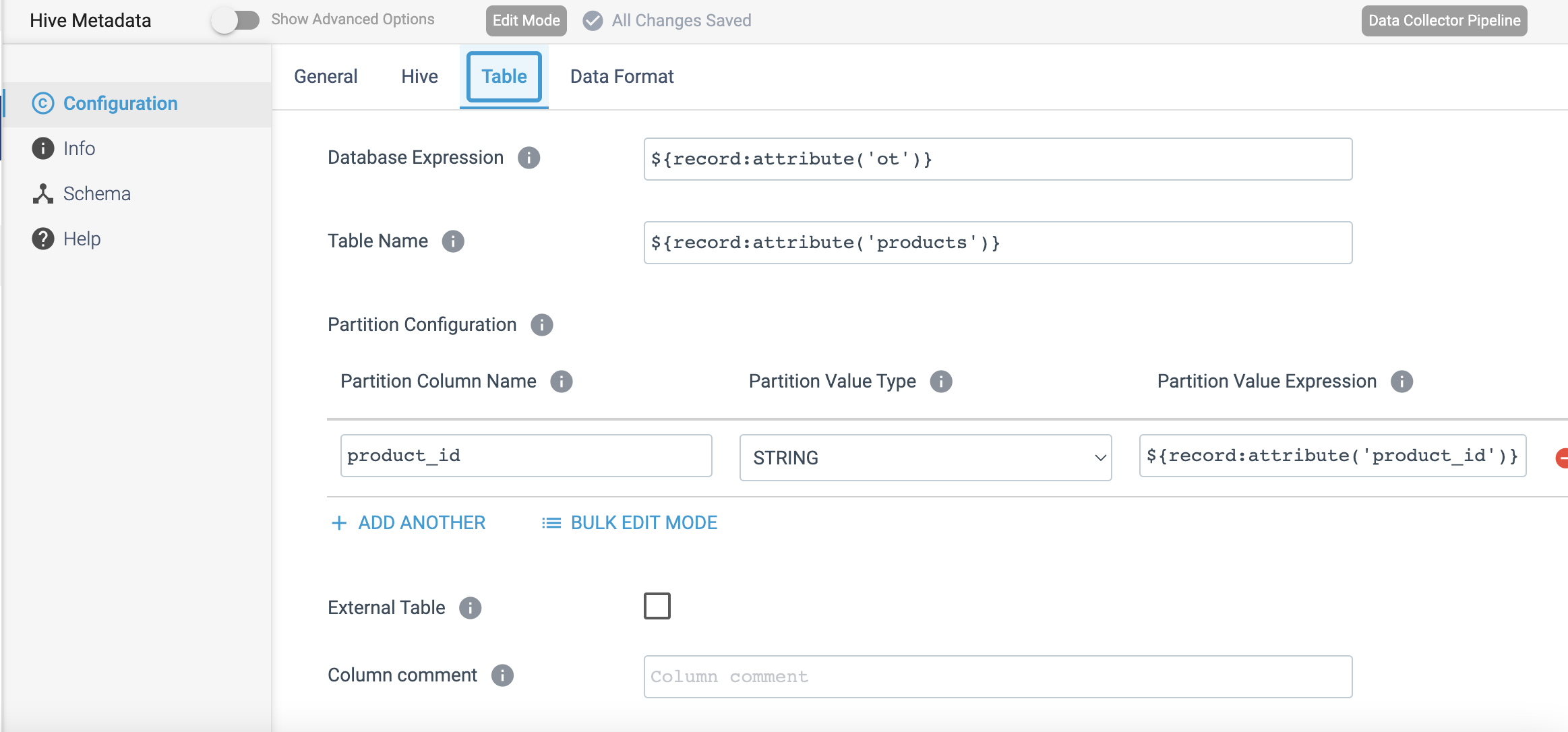
**Description**: The **Hive Metadata** processor are used for records to be written to HDFS or MapR FS when you want the Hive Metastore destination to create and update tables as needed.

We need to configure the **Hive Metadata** in a following way.

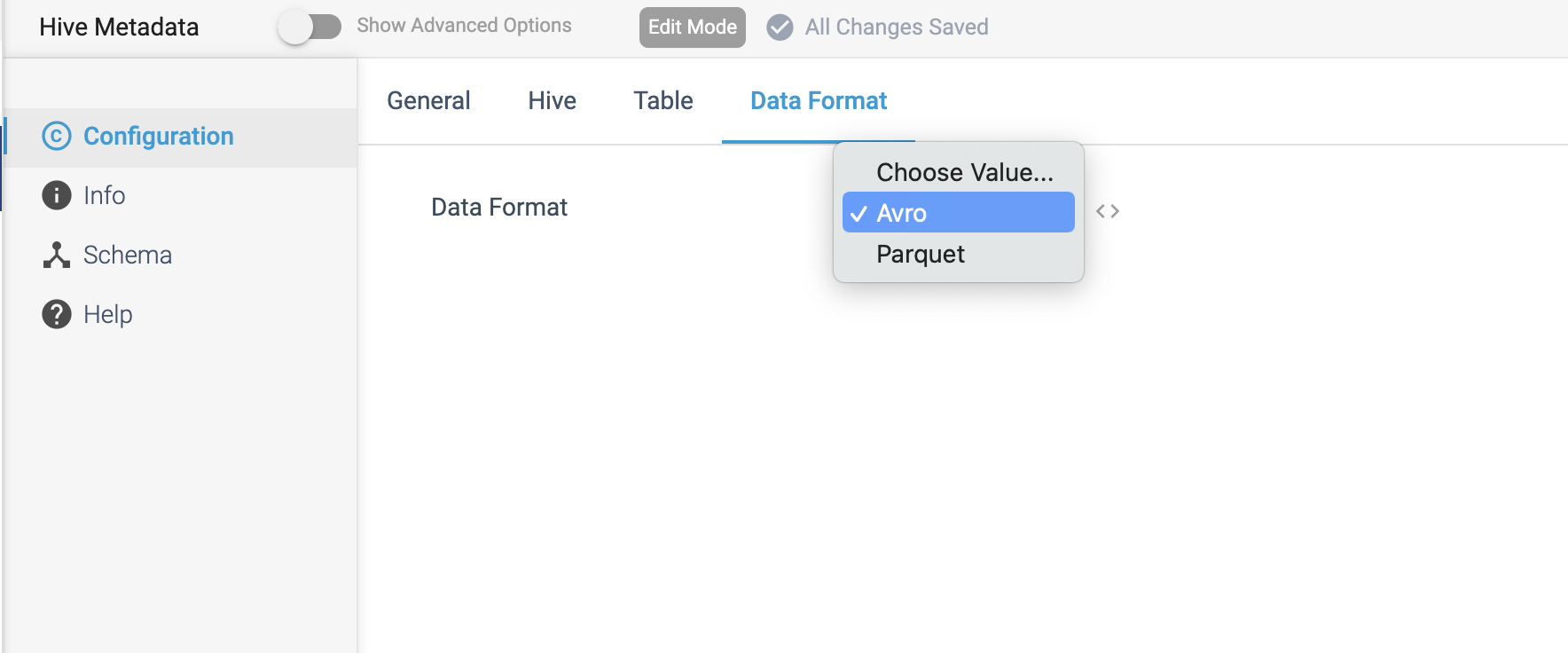
Step1: Click on the **Hive Metadata** processor to define the connection information for Hive and enable Use Credentials, if you have authentication details.



Step 2: Click on the **Table** tab to define the database, table, partitions, and decimal field expressions that the records require below details.

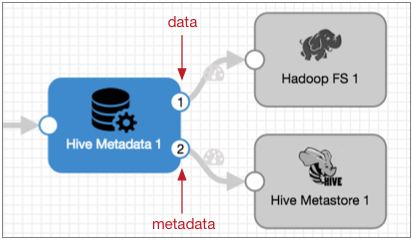


Step 3: Click on the **Data Format** tab to define the output data format such as Avro or parquet.



**Output Streams from Hive Metadata:**

The Hive Metadata processor includes a data and a metadata output stream. The following image shows the Hive Metadata processor output streams:



**Data output stream**

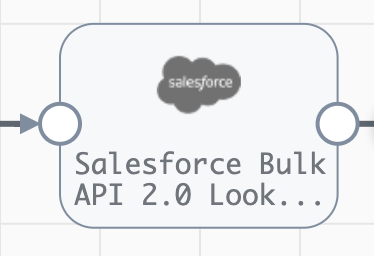
Passes records downstream to the Hadoop FS destination or the MapR FS destination. You can add additional stages between the Hive Metadata processor and the destinations when needed, but only the Hadoop FS and MapR FS destinations can use the generated record header attributes to write records.

**Metadata output stream**

Passes the metadata records downstream to the Hive Metastore destination. The metadata output stream does not pass record data of any kind.

You can add additional stages between the Hive Metadata processor and the Hive Metastore destination when needed, but only the Hive Metastore destination can use the metadata record to update the Hive Metastore.

PROCESSOR NAME: Salesforce Bulk API 2.0 Lookup



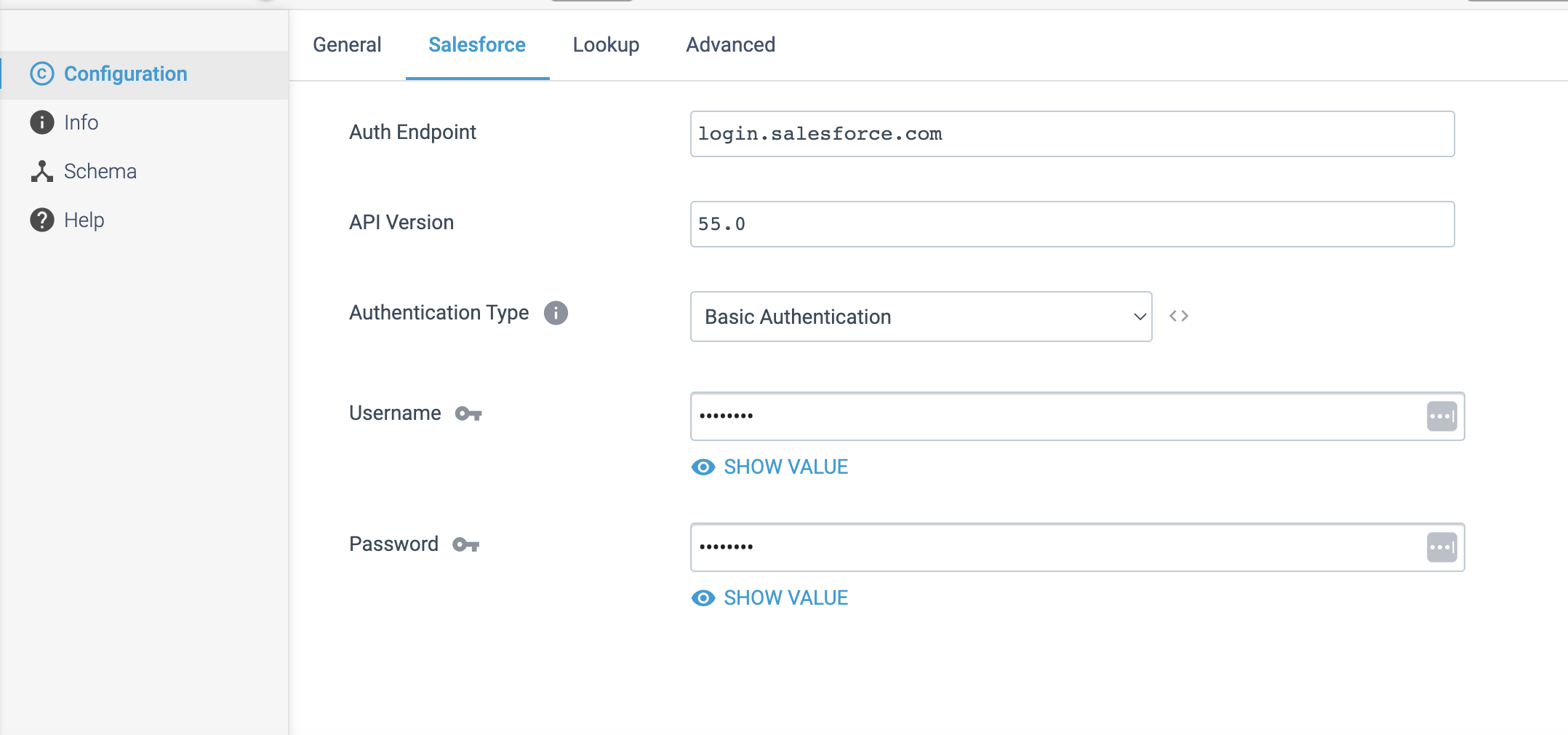
**Engine Type**: Data Collector

**Pipeline Name**: D\_Azure\_Blob\_Orgn\_AzureSQL\_Dest\_Pipeline\_Cred

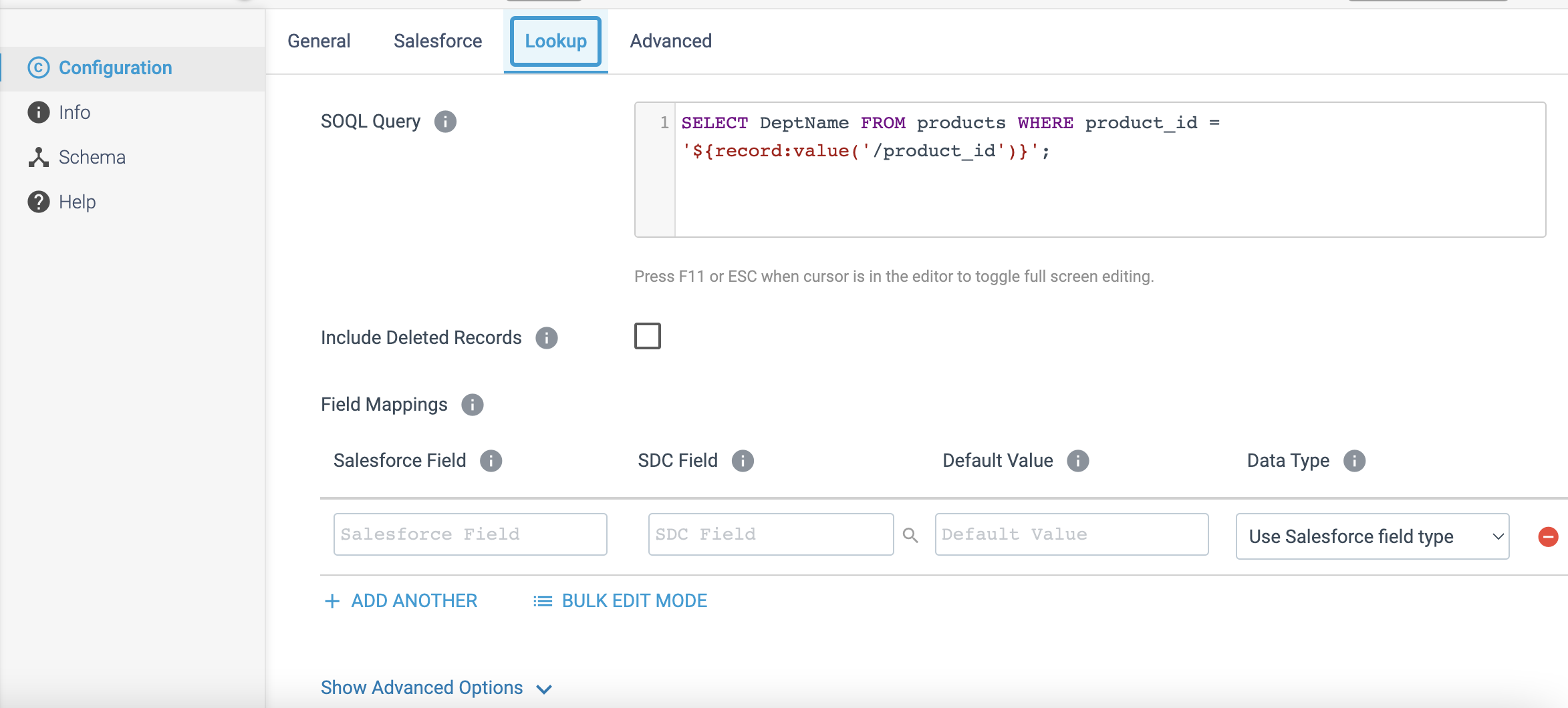
**Description**: The **Salesforce Bulk API 2.0 Lookup** processor performs lookups on a Salesforce object using Salesforce Bulk API 2.0.

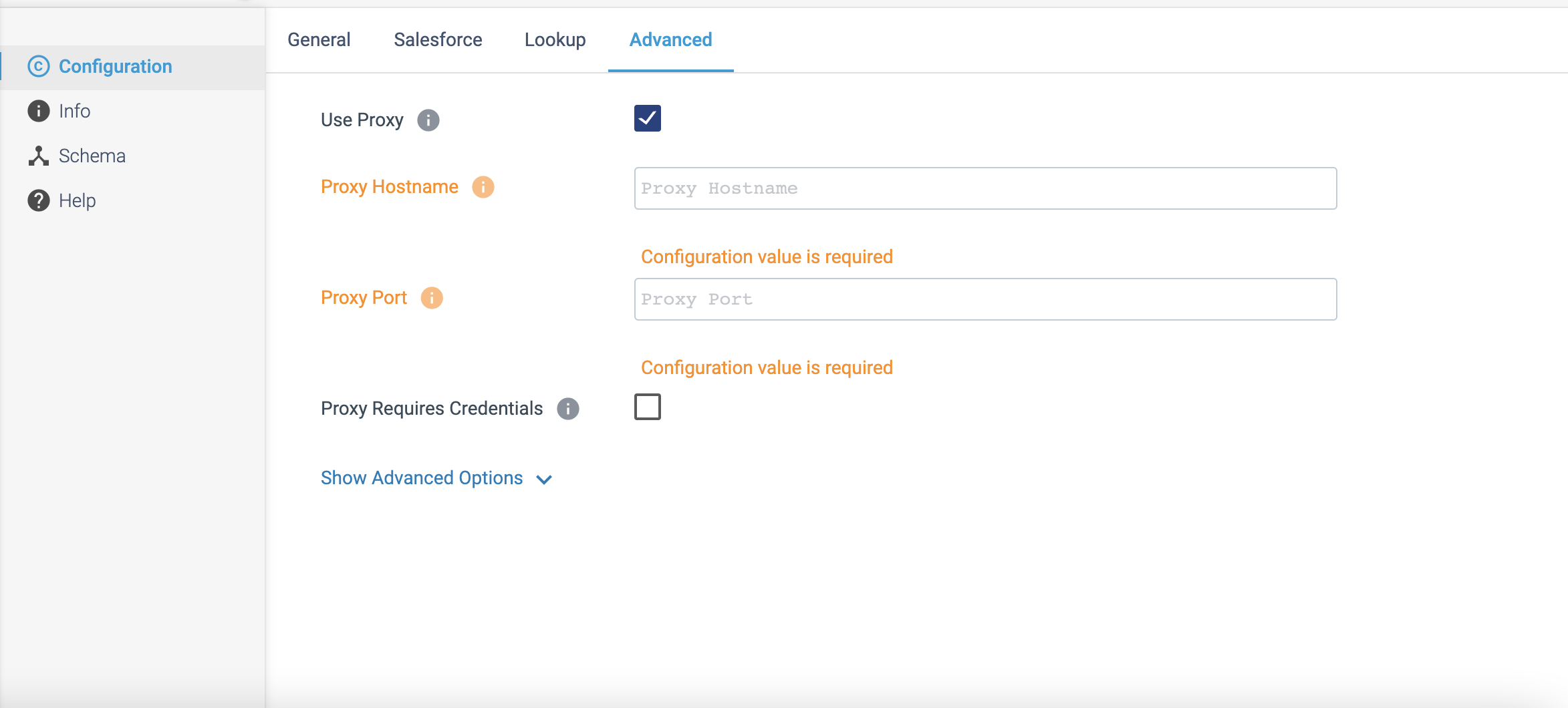
We need to configure the **Salesforce Bulk API 2.0 Lookup** processor in a following way.

Step 1: Click on the **Salesforce Bulk API 2.0 Lookup** processor to define connection information that the processor uses to connect to Salesforce, including the Salesforce **API version** and the **authentication** to use.

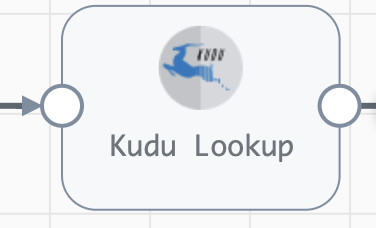


Step 2: Click on the **Lookup** tab to specify a **SOQL query**, the output fields for returned values, and the multiple match behaviour. You can optionally define a **default value** to use for fields with missing values and set the behaviour for fields with missing values and no default value.

  
Step 3: Click on the **Advanced** tab to optionally use an HTTP proxy to connect to Salesforce. When enabled in Salesforce, you can also configure the processor to use mutual authentication.



PROCESSOR NAME: Kudu Lookup



**Engine Type**: Data Collector

**Pipeline Name**: D\_Mongolog\_orgn\_Atlas\_Dest\_pipeline\_CredStr

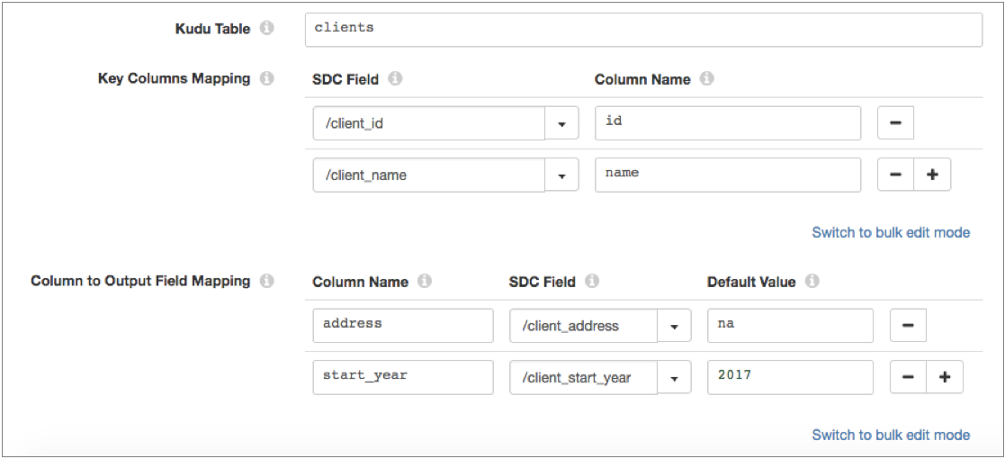
**Description**: The **Kudu Lookup** processor performs lookups in a Kudu table and passes the lookup values to fields.

We need to configure the **Kudu Lookup** processor in a following way.

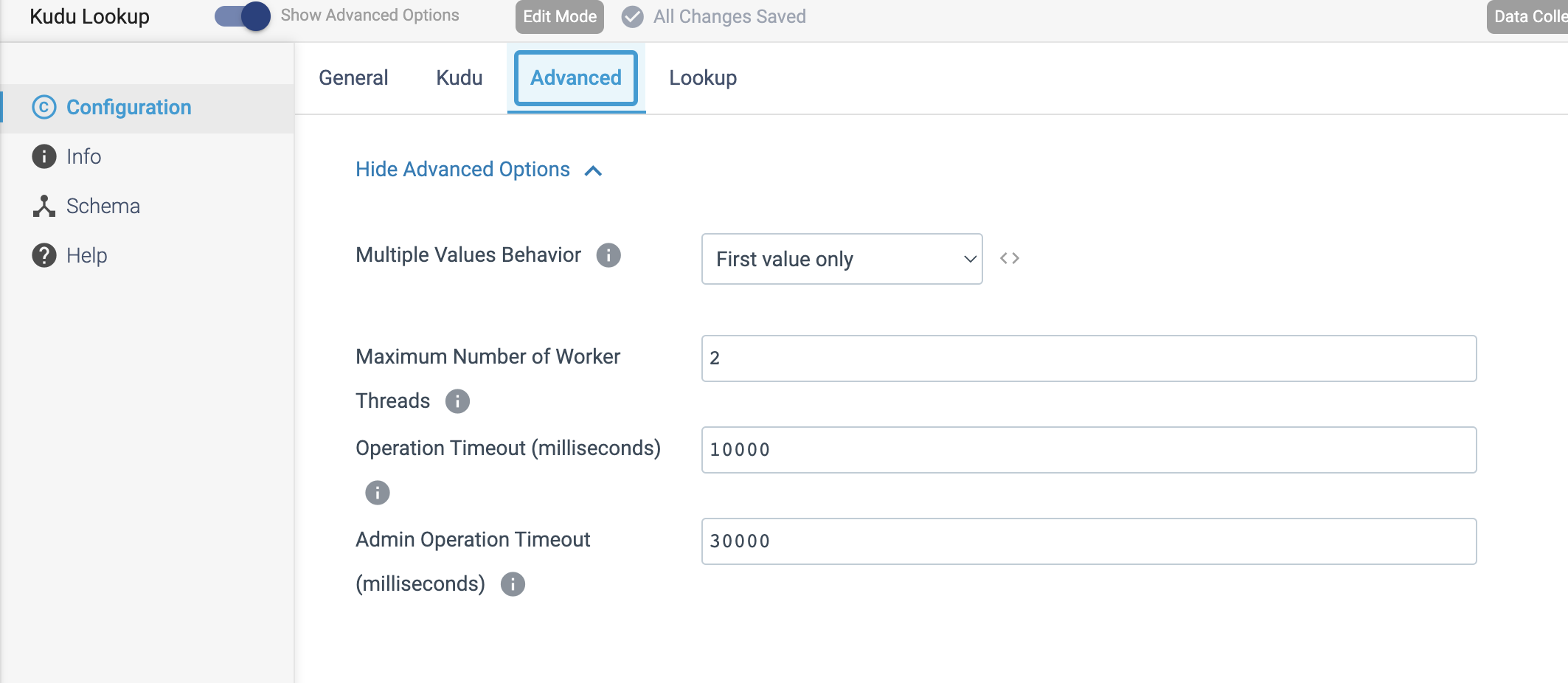
Step 1: Click on the **Kudu Lookup** processor to specify the connection information for one or more Kudu masters and define the table to use. You define the key columns to look up and define the output fields to write the lookup values to. You can also enable Kerberos authentication.



**Columns to Output Fields Mapping:**

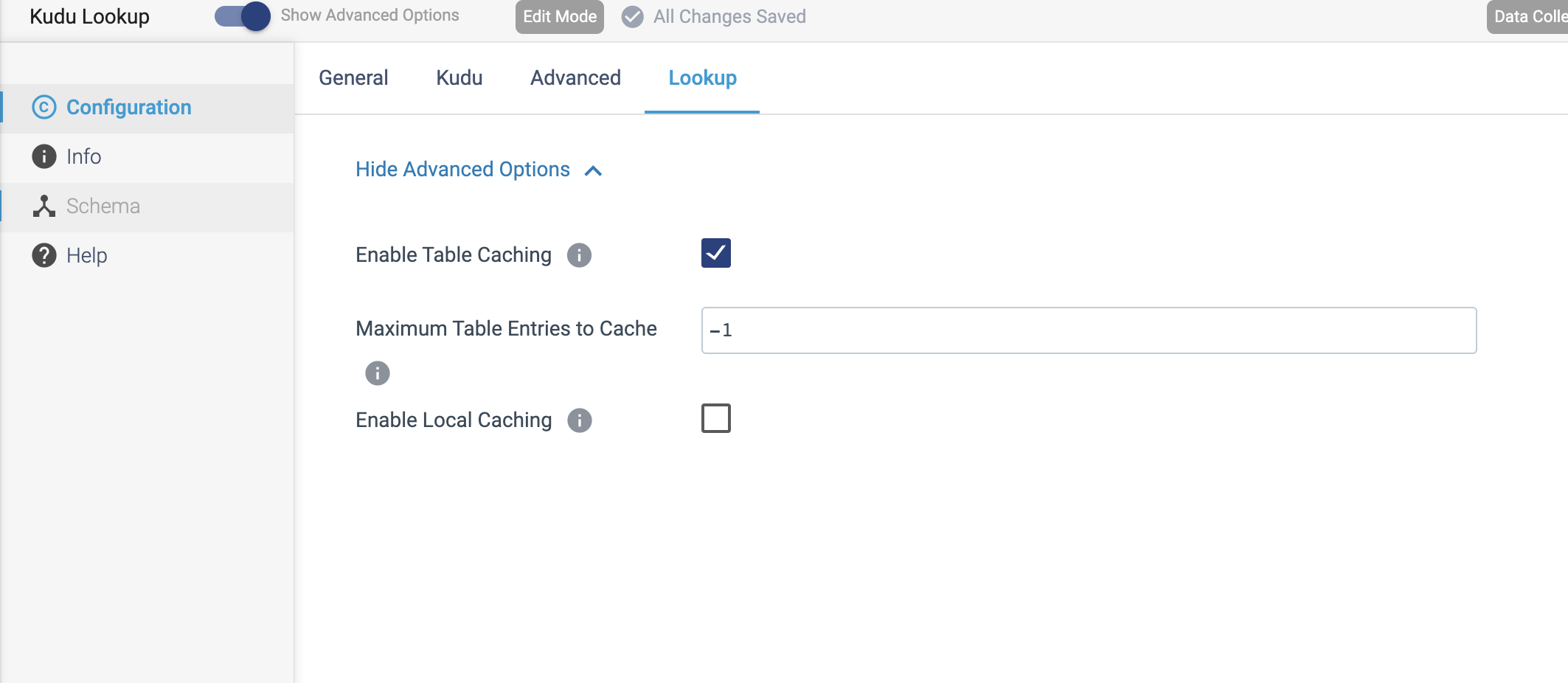


Step 2: Click on the **Advanced** tab to optionally configure operation timeouts and the maximum number of worker threads to use.

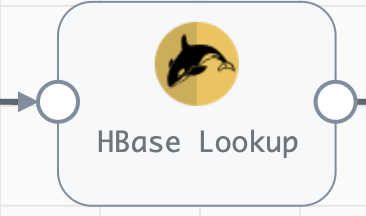


Step 3: Click on the **Lookup** tab to configure the Kudu Lookup processor to locally cache Kudu table information and the lookup values returned from a Kudu table. It will improve pipeline performance.

When you stop the pipeline, the processor clears both caches.



PROCESSOR NAME: HBase Lookup



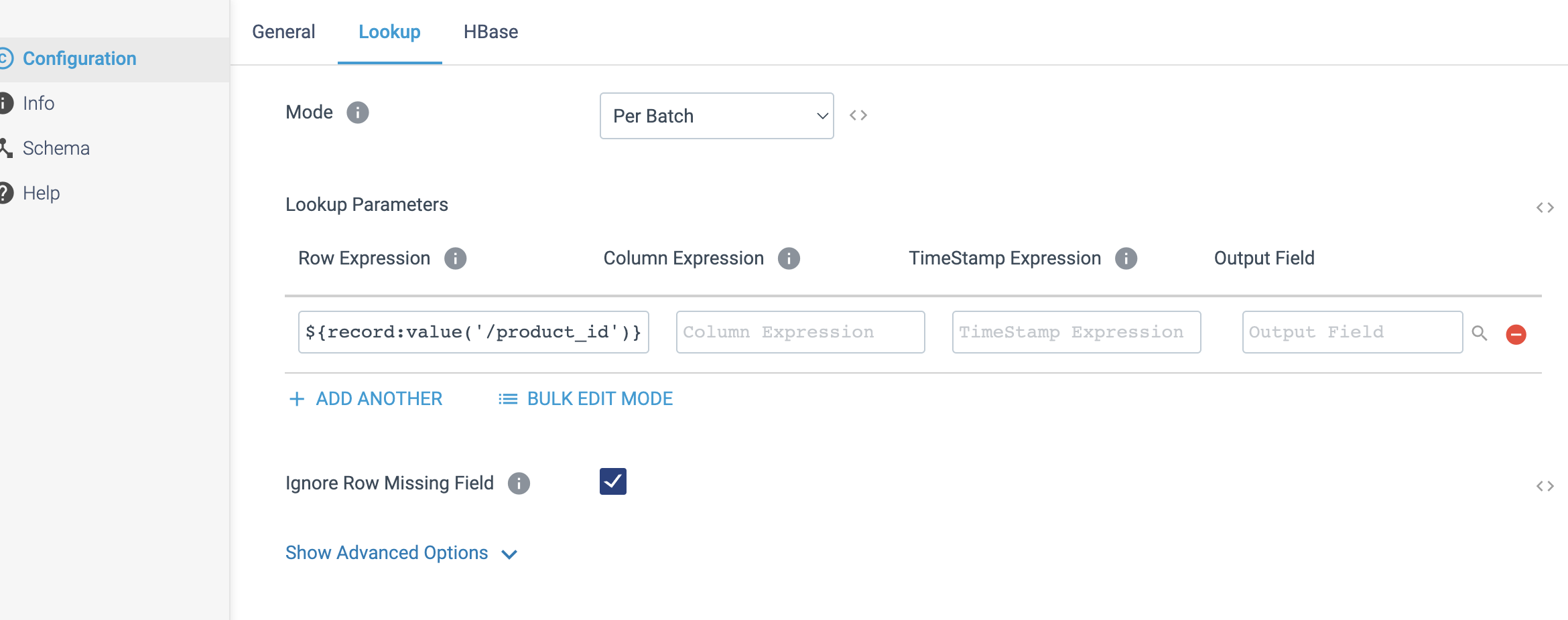
**Engine Type**: Data Collector

**Pipeline Name**: D\_JDBC\_Orgn\_HBase\_Dest\_Pipeline\_CredStr

**Description**: The **HBase Lookup** processor performs key-value lookups in HBase and passes the lookup values to fields.

We need to configure the **HBase Lookup** in a following way.

Step 1: Click on the **HBase Lookup** processor to specify whether the processor performs a bulk lookup of all keys in a batch or performs an individual lookup of each key in a record. You define the key to look up in **HBase** and specify the output field to write the lookup values to.



Step 2: Click on the **HBase** tab to specify the HBase configuration properties, including the **Zookeeper Quorum**, **parent znode**, and **table name**. When necessary, you can enable **Kerberos authentication**, specify an **HBase user**, and add additional **HBase configuration** properties.

