How-To Library

Informatica

Implementing the upsert operation using Amazon Redshift V2 Connector

Abstract

This article discusses the solutions to use the upsert operation with Amazon Redshift V2 Connector from Cloud Data Integration.

Supported Versions

- Informatica® Cloud Data Integration Amazon Redshift V2 Connector
- Informatica® Cloud Data Integration

Table of Contents

Overview

You can configure the insert, update, and delete operations using Amazon Redshift V2 Connector to add, update, and remove data rows in Amazon Redshift. However, if you want to perform an upsert operation in Amazon Redshift from Cloud Data Integration, you need to include additional configurations based on your requirement.

You also need to consider the following factors before you can upsert data to Amazon Redshift:

- The data row is already available in the Amazon Redshift target table.
- More than one data row for a key field or entity is available for a specific batch of commit to the Amazon Redshift data warehouse table.

The first scenario is already addressed using Amazon Redshift v2 Connector. Use the solutions described in the article to address the second scenario.

Use case

You want to perform an upsert operation when there are multiple data rows for a key field or entity.

Consider a dataset that has the following entities: `customer` and `credithistory`

The `customer` dataset is a flat file with the following fields: customer ID and customer name. The customer ID uniquely identifies a customer.

The following image shows a list of customer IDs and customers:

customer.csv

```
1  id,name
2  101,"Eva"
3  102,"Megan"
4  103,"Oscar"
5  104,"Rene"
6  105,"Vickie"
```

The `credithistory` dataset denotes the credit score for a customer at intervals of three months.

The following image shows the credit history for the customers:

credithistory.csv

```
1  customerid,month,year,creditscore
2  101,"Jan",2020,687
3  101,"Apr",2020,630
4  101,"Jul",2020,605
5  102,"Mar",2020,658
6  102,"Jun",2020,673
7  103,"Feb",2020,724
8  103,"May",2020,749
9  104,"Mar",2020,685
10 104,"Jun",2020,723
11 105,"Feb",2020,760
12 105,"May",2020,742
```

Problem statement

For the data set, you must get the latest credit score for all the customers. The final target data output must have the name of customer, the latest credit score, and the month and year in which the score was updated.

Solutions

You can perform a join between the `customer` and `credithistory` data set to get the desired output. However, a join creates duplicate data rows for all the customers in the data set. For example, if a customer with ID `101` has credit scores `687`, `630`, and `605`, a join creates three data rows for the credit scores of the customer.

Implement one of the following solutions to resolve this issue.

Solution 1. Use a custom query to read data

You can filter the data set based on the latest score using a custom SQL query in the Source transformation.

Use the following transformations in a mapping:

- Source transformation: Use Amazon Redshift V2 Connector with the source type as query.
- Target transformation: Use the required connection to write to Amazon Redshift.

Use the source query override in the following format:

```
SELECT B.customerid, B1.month, B1.year, B1.creditscore  FROM PUBLIC.CREDITHISTORY B1, (SELECT
B.customerid, MAX(B.month) as maxmonth FROM PUBLIC.CREDITHISTORY AS B GROUP BY B.customerid) AS
B WHERE B1.customerid = B.customerid and B1.month = B.maxmonth;
```

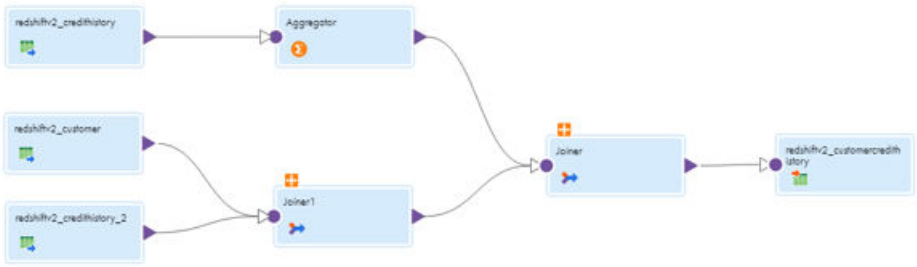
This approach improves the performance by processing the data set at the beginning of the integration pipeline.

Solution 2. Use an Aggregator transformation in a mapping

Use the following transformations in a mapping:

1. Source transformation: Use three Source transformations that read the credit card history and the customer.
2. Aggregator transformation. The Aggregator transformation aggregates the data based on the customer ID and defines the output field `MAX(month)`.
3. Joiner transformation: Use two joiners.
4. Target transformation: Use Amazon Redshift V2 Connector with the upsert operation.

The following image shows the mapping solution:



The first source is from the `credithistory` table.

The second and third sources are `customer` and `credithistory` tables that are joined in the joiner transformation using condition `customerid`.

Another normal join is defined between the upstream data using the `customerid` condition and the month.

The following image shows the join condition:

Join Conditions

Master	Operator	Detail
agg_customerid	=	id
agg_maxmonth	=	month

The fields with prefix `agg_` are added from upstream in the Aggregator transformation.

Note: The prefix `agg_` is added as a part of the naming convention in the incoming field rules.

The Target transformation must point to the object, for example, `customercredithistory` and implement an upsert operation.

The advantage of this approach is that it is a complete mapping solution. Also, if you parameterize the source and target connections and objects, you can use this solution for any upsert operation for objects of a similar use case.

Solution 3. Use a post-SQL query in the Target transformation

You can write duplicate data from the source as is and then configure a post-SQL query with Amazon Redshift v2 Connector to clean the written data set.

Use the following transformations in the mapping:

1. Source transformation: Use Amazon Redshift V2 Connector with the source type as single object or multiple objects.
2. Target transformation: Use Amazon Redshift V2 Connector and select the upsert operation.

Configure a post-SQL query for the Target transformation in the advanced properties and use the following command to include only the latest updated credit card scores using the following SQL query:

```
DELETE from PUBLIC.CUSTOMERCREDITHISTORY WHERE idenid NOT IN (SELECT MAX(B.idenid) FROM
PUBLIC.CUSTOMERCREDITHISTORY AS B GROUP BY customerid);
```

The advantage of this solution is that you do not need to make changes to the mapping solution. The data is cleaned in the Amazon Redshift target table based on the configured post-SQL statement after the write operation completes.

Author

Yoganayagam S