Wave Analytics Data Integration

Salesforce, Spring '16

@salesforcedocs

Last updated: April 28, 2016

© Copyright 2000–2016 salesforce.com, inc. All rights reserved. Salesforce is a registered trademark of salesforce.com, inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

CONTENTS

DATA INTEGRATION 1
Datasets 2
Dataflow JSON 5
Dataset Builder 6
Wave Connector for Excel Data 6
Upload User Interface for External Data 6
External Data API 7

CREATE DATASETS WITH A DATAFLOW 8
Design the Dataflow 8
Configure the Dataflow 9
Start the Dataflow 12
Monitor a Dataflow Job 13
Reschedule the Dataflow 14

DATAFLOW TRANSFORMATION REFERENCE 16
Transformations 16
Overriding Metadata Generated by a Transformation 76

CREATE A DATASET WITH THE DATASET BUILDER 80

CREATE A DATASET WITH EXTERNAL DATA 86
Create a Dataset with External Data 86
Monitor an External Data Upload 90

EDIT A DATASET 93

DELETE A DATASET 96

ROW-LEVEL SECURITY FOR DATASETS 97
Considerations when Defining a Predicate for a Dataset 98
Row-Level Security Example based on Record Ownership 98
Row-Level Security Example based on Opportunity Teams 103
Row-Level Security Example based on Role Hierarchy and Record Ownership 111

SECURITY PREDICATE REFERENCE 122
Predicate Expression Syntax 122
Sample Predicate Expressions 127

DATA INTEGRATION

You can integrate Salesforce data and external data into Wave Analytics to enable users to explore and visualize the data with explorer and designer. *External data* is data that resides outside of Salesforce, such as data from outside applications and spreadsheets.

When you load data into Wave Analytics, you load it into datasets. A dataset is a collection of related data that is stored in a denormalized, yet highly compressed form.

You can use the following methods to create datasets in Wave Analytics.

	Dataflow JSON	Dataset Builder	Upload User Interface	External Data API	Wave Connector
Data source	Salesforce objects; existing datasets	Salesforce objects	External data	External data	Microsoft Excel
Can join external and Salesforce data?	Yes	No	No	No	No
Graphical user interface?	No (JSON)	Yes	Yes	No (programmatic access)	Yes
Can create multiple datasets at once?	Yes	No	No	No	No
Supports incremental extraction?	Yes	No	No	Yes	No
Data refresh method	Automatic	Automatic	Manual	Manual	Manual
Can filter records?	Yes	No	No	No	No
Can generate new columns when creating datasets?	Yes (delta and dimension columns)	No	No	No	No
Can override metadata?	Yes	No	Yes	Yes	No

IN THIS SECTION:

Datasets

A dataset is a collection of related data that is stored in a denormalized, yet highly compressed form. For each platform license, your organization can have a maximum of 250 million rows of data stored for all registered datasets combined.

Dataflow JSON

You can use the dataflow to create one or more datasets based on data from Salesforce objects or existing datasets. A *dataflow* is a set of instructions that specifies what data to extract from Salesforce objects or datasets, how to transform the datasets, and which datasets to make available for querying. With a dataflow, you can manipulate the extracted data and override the metadata before you load it into a dataset. The dataflow runs on a daily schedule to continually refresh the data.

Dataset Builder

Use the dataset builder to create a single dataset based on data from one or more related Salesforce objects. With the dataset builder you simply point and click to identify and select related Salesforce objects.

Wave Connector for Excel Data

The Salesforce Wave Connector makes it easy to import data from Microsoft Excel 2013 to Wave Analytics.

Upload User Interface for External Data

You can use the upload user interface to create a single dataset based on external data. You can upload an external data file in a .csv, .gz, or .zip format. To refresh the data, you can overwrite the data in the dataset by uploading a new external data file.

External Data API

You can use the External Data API to create a single dataset based on external data in the .csv format. You can also use the API to edit the dataset by uploading a new .csv file. When you edit the dataset, you can choose to overwrite all records, append records, update records, or delete records.

Datasets

A dataset is a collection of related data that is stored in a denormalized, yet highly compressed form. For each platform license, your organization can have a maximum of 250 million rows of data stored for all registered datasets combined.

Wave Analytics applies one of the following types to each dataset field:

Date

A *date* can be represented as a day, month, year, and, optionally, time. You can group, filter, and perform math on dates.

Dimension

A *dimension* is a qualitative value, like region, product name, and model number. Dimensions are handy for grouping and filtering your data. Unlike measures, you can't perform math on dimensions. To increase query performance, Wave Analytics indexes all dimension fields in datasets.

Measure

A *measure* is a quantitative value, like revenue and exchange rate. You can do math on measures, such as calculating the total revenue and minimum exchange rate.

For each dataset that you create, you can apply row-level security to restrict access to records in the dataset.

Attention: Before you create a dataset, verify that the source data contains at least one value in each column. Columns with all null values won't be created in datasets and can't be referenced in dataflows, lenses, or dashboards. Consider providing a default value for null values, like "n/a" or "empty."

IN THIS SECTION:

Numeric-Value Handling in Datasets

Wave Analytics internally stores numeric values in datasets as long values. For example, it stores the number "3,200.99" with a scale of "2" as "320099". The user interface converts the stored value back to decimal notation to display the number as "3200.99."

Date Handling in Datasets

When Wave Analytics loads dates into a dataset, it breaks up each date into multiple fields, such as day, week, month, quarter, and year, based on the calendar year. For example, if you extract dates from a `CreateDate` field, Wave Analytics generates date fields such as `CreateDate_Day` and `CreateDate_Week`. If your fiscal year differs from the calendar year, you can enable Wave Analytics to generate fiscal date fields as well.

Numeric-Value Handling in Datasets

Wave Analytics internally stores numeric values in datasets as long values. For example, it stores the number "3,200.99" with a scale of "2" as "320099". The user interface converts the stored value back to decimal notation to display the number as "3200.99."

The maximum numeric value that can be stored in a dataset is 36,028,797,018,963,967 and the minimum numeric value is -36,028,797,018,963,968.

Warning: If a numeric value is not within this range, you might receive unexpected results. For example, if you try to load the value 3.7E-16 with a scale of 16 into a dataset, Wave Analytics tries to store the value as 37000000000000000. However, because this value exceeds the maximum, Wave Analytics fails to load the entire record. In addition, if you perform a query that aggregates measures—like sum or group by—and the resulting value exceeds the maximum, the value overflows and Wave Analytics returns an incorrect result.

Date Handling in Datasets

When Wave Analytics loads dates into a dataset, it breaks up each date into multiple fields, such as day, week, month, quarter, and year, based on the calendar year. For example, if you extract dates from a `CreateDate` field, Wave Analytics generates date fields such as `CreateDate_Day` and `CreateDate_Week`. If your fiscal year differs from the calendar year, you can enable Wave Analytics to generate fiscal date fields as well.

Wave Analytics generates the following date fields.

Field Name	Field Type	Description
<date field name>_Second	Text	Number of seconds. If the date contains no seconds, value is '0.'
<date field name>_Minute	Text	Number of minutes. If the date contains no minutes, value is '0.'
<date field name>_Hour	Text	Number of hours. If the date contains no hours, value is '0.'
<date field name>_Day	Text	Day of the month.
<date field name>_Week	Text	Week number in calendar year.
<date field name>_Month	Text	Month number in calendar year.
<date field name>_Quarter	Text	Quarter number in calendar year.
<date field name>_Year	Text	Calendar year.
<date field name>_Week_Fiscal	Text	Week number in fiscal year.
<date field name>_Month_Fiscal	Text	Month number in fiscal year.
<date field name>_Quarter_Fiscal	Text	Quarter number in fiscal year.

Field Name	Field Type	Description
<date field name>_Year_Fiscal	Text	Fiscal year.
<date field name>_sec_epoch	Numeric	Number of seconds that have elapsed since January 1, 1970 (midnight UTC).
<date field name>_day_epoch	Numeric	Number of days that have elapsed since January 1, 1970 (midnight UTC).

You can set metadata attributes to control how dates are loaded into datasets and to enable Wave Analytics to generate fiscal date fields. You set the metadata attributes in the sfdcDigest transformation parameters for Salesforce data or in the metadata file for external data.

Note: Before loading dates from an external data file, ensure that you review the date format requirements here. Also, ensure that the column names in the external data file do not conflict with the generated date field names. For example, if you load a CSV with column `Create_Date`, Wave Analytics generates the `Create_Date_Year` field in the dataset. If the CSV also had a field named `Create_Date_Year`, Wave Analytics would throw an error because the names conflict.

Fiscal Periods in Wave Analytics

If the calendar and fiscal year differ, you can enable Wave Analytics to generate the fiscal date fields in the dataset in addition to calendar date fields. To enable Wave Analytics to generate fiscal date fields, set the `fiscalMonthOffset` attribute to a value other than '0'. You set this attribute for each date column for which you want to generate fiscal date fields. If you set the offset to '0' or you do not specify a value, Wave Analytics does not generate any fiscal date fields.

Additionally, to configure the fiscal periods, set the following metadata attributes for each date column:

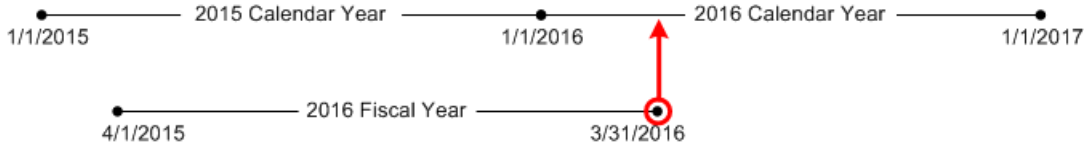
`fiscalMonthOffset`

In addition to enabling the generation of fiscal date fields, this attribute also determines the first month of the fiscal year. You specify the difference between the first month of the fiscal year and first month of the calendar year (January) in `fiscalMonthOffset`. For example, if your fiscal year begins in April, set `fiscalMonthOffset` to '3'.

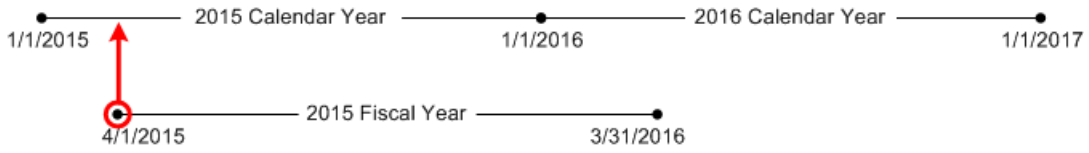
`isYearEndFiscalYear`

Because the fiscal year can start in one calendar year and end in another, you must specify which year to use for the fiscal year. The `isYearEndFiscalYear` attribute indicates whether the fiscal year is the year in which the fiscal year ends or begins.

To see how this works, let's look at a couple of examples. If `isYearEndFiscalYear` = true (or you do not specify this attribute), then the fiscal year is the year in which the fiscal year ends. As shown in the following diagram, any dates between 4/1/2015 and 3/31/2016 are part of the fiscal year 2016 because the fiscal year ends in 2016.



If `isYearEndFiscalYear` = false, then the fiscal year is the year in which the fiscal year begins. As shown in the following diagram, any dates between 4/1/2015 and 3/31/2016 are part of the fiscal year 2015 because the fiscal year begins in 2015.



Week Numbering in Wave Analytics

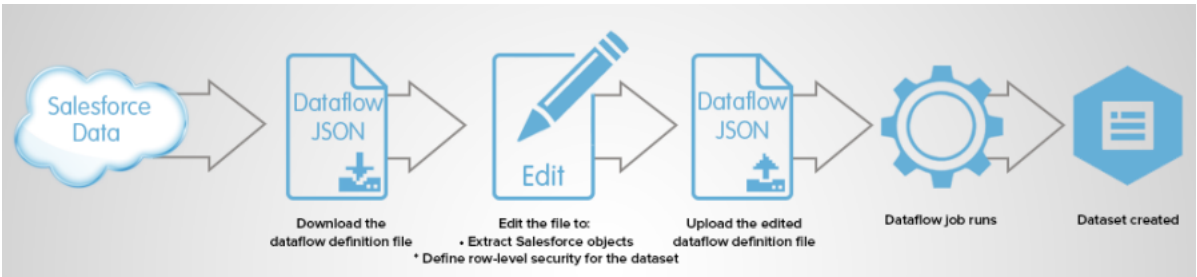
For each date loaded into a dataset, Wave Analytics generates the corresponding week number for the calendar year and, if applicable, fiscal year. Similar to the SOQL function `WEEK_IN_YEAR`, week 1 in Wave Analytics is January 1 - January 7. (This is different from the UTC `week()` calculation.)

If needed, you can configure the week to start on a particular day of the week by setting the `firstDayOfWeek` attribute. For example, if January 1 is a Saturday and you configure the week to start on a Monday, then week 1 is January 1 - 2. Week 2 starts on Monday, January 3. Week 3 starts January 10, the following Monday. Notice that week 1 can be a short week to ensure that the subsequent weeks start on the specified day of the week.

Dataflow JSON

You can use the dataflow to create one or more datasets based on data from Salesforce objects or existing datasets. A *dataflow* is a set of instructions that specifies what data to extract from Salesforce objects or datasets, how to transform the datasets, and which datasets to make available for querying. With a dataflow, you can manipulate the extracted data and override the metadata before you load it into a dataset. The dataflow runs on a daily schedule to continually refresh the data.

Wave Analytics provides a default dataflow that contains some sample transformation logic. This dataflow is just a sample that you must configure before running it.



To configure the dataflow, you add transformations to the dataflow definition file. A *dataflow definition file* is a JSON file that contains transformations that represent the dataflow logic. You can add transformations to determine what data to extract, how to transform datasets, and which datasets to register to make available for queries.

After you configure the dataflow, you upload the new dataflow definition file to Wave Analytics.

By default, the dataflow doesn't run automatically. To start running the dataflow on the schedule, you must manually start the dataflow first. After the first job runs, the dataflow job runs on the daily schedule. The dataflow runs on a daily schedule to capture the latest changes to Salesforce data and changes in the dataflow logic.

You can also stop, reschedule, and monitor dataflow jobs.

SEE ALSO:

Create Datasets with a Dataflow

Dataset Builder

Use the dataset builder to create a single dataset based on data from one or more related Salesforce objects. With the dataset builder you simply point and click to identify and select related Salesforce objects.

After you select the data to include in the dataset, the dataset builder generates and appends the associated JSON to the dataflow definition file. The dataset is created the next time the dataflow runs. The data in the dataset refreshes each time the dataflow runs.

SEE ALSO:

Create a Dataset with the Dataset Builder

Wave Connector for Excel Data

The Salesforce Wave Connector makes it easy to import data from Microsoft Excel 2013 to Wave Analytics.

The Wave Connector is available as an app for Excel 2013 on the desktop and Excel Online in Office 365. The Connector is available as an app from the Microsoft Apps for Office store or your organization's private app catalog. After you install the Connector just point and click to import data from Excel to Salesforce.

SEE ALSO:

Install the Wave Connector Excel App

Upload User Interface for External Data

You can use the upload user interface to create a single dataset based on external data. You can upload an external data file in a .csv, .gz, or .zip format. To refresh the data, you can overwrite the data in the dataset by uploading a new external data file.

When Wave Analytics loads any data into a dataset, it also adds metadata about each column of data. For example, metadata can include the field type, precision, scale, and default value.

For external data, Wave Analytics infers metadata about each column of data in the external data file unless you specify different metadata attributes in a metadata file. A *metadata file* is a JSON file that describes the structure of an external data file. For example, you can use a metadata file to explicitly set the field type and default value for a specific column of external data. If no metadata file is provided when you upload external data, Wave Analytics treats every column as a dimension and sets the field type to 'Text.' This impacts the type of queries that can be placed on the dataset because you can't perform mathematical calculations on dataset columns with a Text field type. You can only perform mathematical calculations on dataset columns with a Numeric field type.

After you create a dataset based on an external data file, you can edit the dataset to apply a new metadata file. This enables you to change the metadata attributes of each column.

Note: Wave temporarily stores the uploaded CSV and metadata files for processing only. After a dataset is created, Wave purges the files.

SEE ALSO:

Create a Dataset with External Data

External Data API

You can use the External Data API to create a single dataset based on external data in the .csv format. You can also use the API to edit the dataset by uploading a new .csv file. When you edit the dataset, you can choose to overwrite all records, append records, update records, or delete records.

For more information about the External Data API, see the *Wave Analytics External Data API Developer Guide*.

CREATE DATASETS WITH A DATAFLOW

You can use a dataflow to create one or more datasets based on data from Salesforce objects or existing datasets.

IN THIS SECTION:

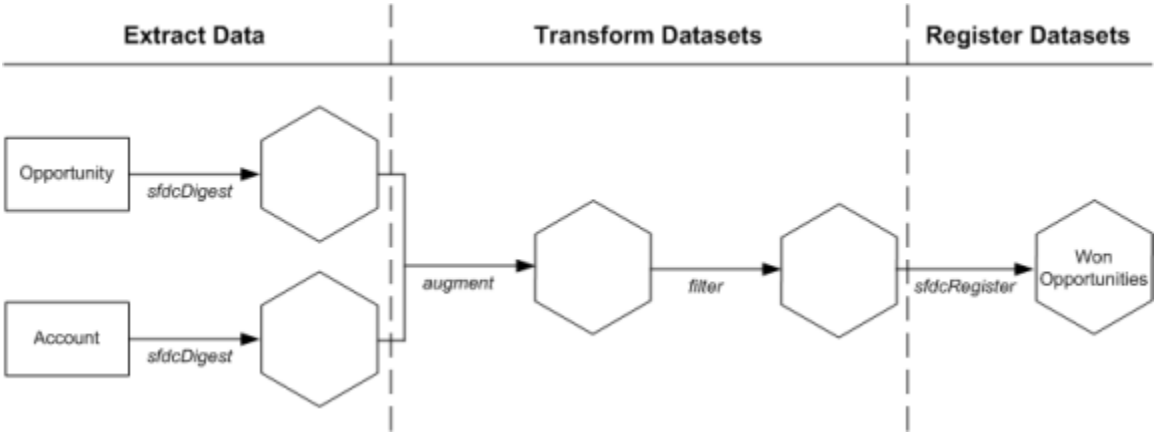
1. Design the Dataflow
   Before you start creating the dataflow definition file in the .json format, think about the dataflow design. Consider what data to make available for queries, where to extract the data from, and whether you need to transform the extracted data to get the data you want.
2. Configure the Dataflow
   Configure the dataflow based on your dataflow design. You can configure the dataflow to extract data, transform datasets based on your business requirements, and register datasets that you want to make available for queries. To configure the dataflow, add transformations to the dataflow definition file.
3. Start the Dataflow
   You can manually start a dataflow job to load the data into datasets immediately. You can also stop the job while it's running. You can run a maximum of 24 dataflow jobs during a rolling 24-hour period.
4. Monitor a Dataflow Job
   Use the data monitor to monitor dataflow jobs to ensure that they complete successfully or to troubleshoot them if they fail.
5. Reschedule the Dataflow
   The dataflow job runs on a daily schedule. You can change the time that the dataflow job runs. You might change the time to ensure that the data is available by a particular time or to run the job during non-business hours.

Design the Dataflow

Before you start creating the dataflow definition file in the .json format, think about the dataflow design. Consider what data to make available for queries, where to extract the data from, and whether you need to transform the extracted data to get the data you want.

To illustrate some key design decisions, let's consider an example. In this example, the goal is to create a dataset called "Won Opportunities." The dataset will contain opportunity details, including the account name for each opportunity.

To create this dataset, you design the following dataflow:



The dataflow extracts opportunity data from the Opportunity object and extracts the account name from the Account object. For each extracted object, the dataflow creates a new dataset.

The dataflow then transforms the datasets created from the extracted data. First, the dataflow joins the opportunity and account data into a new dataset. Next, the dataflow filters the records based on the opportunity stage so that the dataset contains only won opportunities. Each time the dataflow transforms a dataset, it creates a new dataset.

Finally, because you want users to be able to query won opportunities only, you configure the dataflow to register the final dataset only. However, if you wanted, you could register any dataset created by the dataflow and register as many datasets as you like.

Carefully choose which datasets to register because:

- The total number of rows in all registered datasets cannot exceed 250 million per platform license.
- Users that have access to registered datasets can query their data. Although, you can apply row-level security on a dataset to restrict access to records.

Configure the Dataflow

Configure the dataflow based on your dataflow design. You can configure the dataflow to extract data, transform datasets based on your business requirements, and register datasets that you want to make available for queries. To configure the dataflow, add transformations to the dataflow definition file.

A *dataflow definition file* is a JSON file that contains transformations that represent the dataflow logic. The dataflow definition file must be saved with UTF-8 encoding.

Before you can configure a dataflow to process external data, you must upload the external data to Wave Analytics.

1. In Wave Analytics, click the gear icon ( ) and then click **Data Monitor** to open the data monitor.
   The Jobs view of the data monitor appears by default.
2. Select **Dataflow View**.
3. To download the dataflow definition file, click **Download** in the actions list (1).

EDITIONS

Available in: Salesforce Classic and Lightning Experience

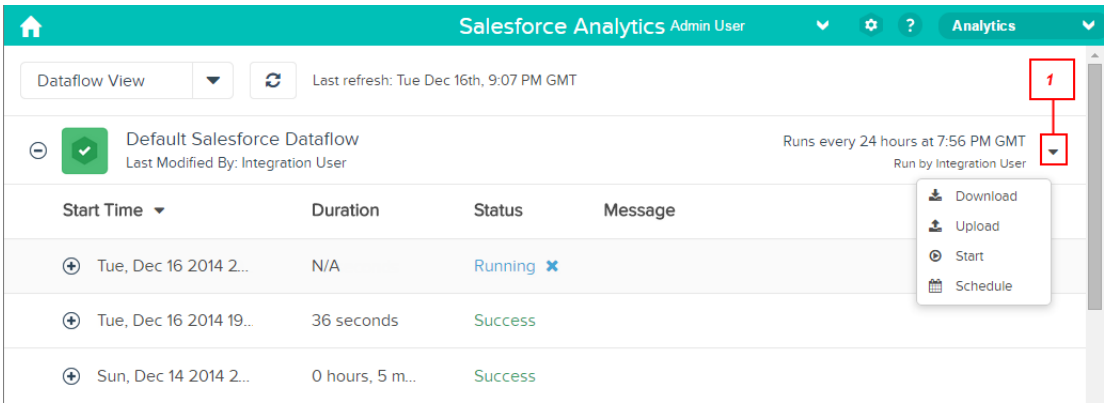
Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To edit the dataflow definition file:

- "Edit Wave Analytics Dataflows"



4. Make a backup copy of the dataflow definition file before you modify it.

   Wave Analytics doesn't retain previous versions of the file. If you make a mistake, you can upload the previous version to roll back your changes.

5. Add transformations to the dataflow definition file.

   For example, based on the design in the previous step, you can add the following transformations:

```
{
   "Extract_Opportunities": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "Amount" },
            { "name": "StageName" },
            { "name": "CloseDate" },
            { "name": "AccountId" },
            { "name": "OwnerId" }
         ]
      }
   },
   "Extract_AccountDetails": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Account",
         "fields": [
            { "name": "Id" },
            { "name": "Name" }
         ]
      }
   },
   "Transform_Augment_OpportunitiesWithAccountDetails": {
      "action": "augment",
      "parameters": {
         "left": "Extract_Opportunities",
         "left_key": [ "AccountId" ],
         "relationship": "OpptyAcct",
```

```
            "right": "Extract_AccountDetails",
            "right_key": [ "Id" ],
            "right_select": [
               "Name"
            ]
         }
      },
      "Transform_Filter_Opportunities": {
         "action": "filter",
         "parameters": {
            "filter": "StageName:EQ:Closed Won",
            "source": "Transform_Augment_OpportunitiesWithAccountDetails"
         }
      },
      "Register_Dataset_WonOpportunities": {
         "action": "sfdcRegister",
         "parameters": {
            "alias": "WonOpportunities",
            "name": "WonOpportunities",
            "source": "Transform_Filter_Opportunities"
         }
      }
}
```

Note: The JSON keys and values are case-sensitive. Each bolded key in the previous example JSON contains one action, which identifies the transformation type. The order in which you add the transformations to the dataflow definition file doesn't matter. Wave Analytics determines the order in which to process the transformations by traversing the dataflow to determine the dependencies among them.

6. Before you save the dataflow definition file, use a JSON validation tool to verify that the JSON is valid.

   An error occurs if you try to upload the dataflow definition file with invalid JSON. You can find JSON validation tool on the internet.

7. Save the dataflow definition file with UTF-8 encoding, and then close the file.
8. In the Dataflow view of the data monitor, click **Upload** from the action list (1) to upload the updated dataflow definition file.

   Note: Uploading the dataflow definition file does not affect any running dataflow jobs and does not automatically start the dataflow job.

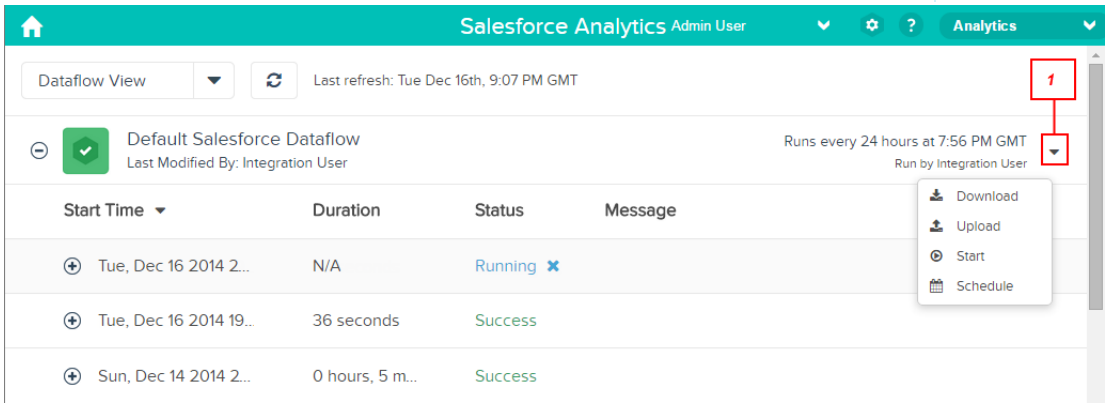
You can now start the dataflow on demand or wait for it to run on the schedule. Users cannot query the registered datasets until the dataflow runs.

Start the Dataflow

You can manually start a dataflow job to load the data into datasets immediately. You can also stop the job while it's running. You can run a maximum of 24 dataflow jobs during a rolling 24-hour period.

Note: By default, the dataflow doesn't run automatically. To start running the dataflow on the schedule, you must manually start the dataflow first. After the first job runs, the dataflow job runs on the daily schedule.

1. In Wave Analytics, click the gear icon ( ) and then click **Data Monitor** to open the data monitor.
   The Jobs view of the data monitor appears by default.
2. Select **Dataflow View**.



3. Click **Start** in the actions list (1) to start the dataflow job.
   The dataflow job is added to the job queue. The **Start** button is greyed out while the dataflow job runs.
4. After the job completes, Wave Analytics sends an email notification to the user who last modified the dataflow definition file.
   The email notification indicates whether the job completed successfully. It also shows job details like start time, end time, duration, and number of processed rows. If the job failed, the notification shows the reason for the failure.

You can monitor the dataflow job in the data monitor to determine when dataflow completes. After the dataflow completes successfully, refresh the Home page to view the registered datasets.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To start a dataflow job:

- "Edit Wave Analytics Dataflows"

Monitor a Dataflow Job

Use the data monitor to monitor dataflow jobs to ensure that they complete successfully or to troubleshoot them if they fail.

The Dataflow view of the data monitor shows the status, start time, and duration of the last 10 dataflow jobs and retains the last 7 days of history. To help you troubleshoot a failed job, you can view error messages about the job and view the run-time details about every transformation that is processed.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

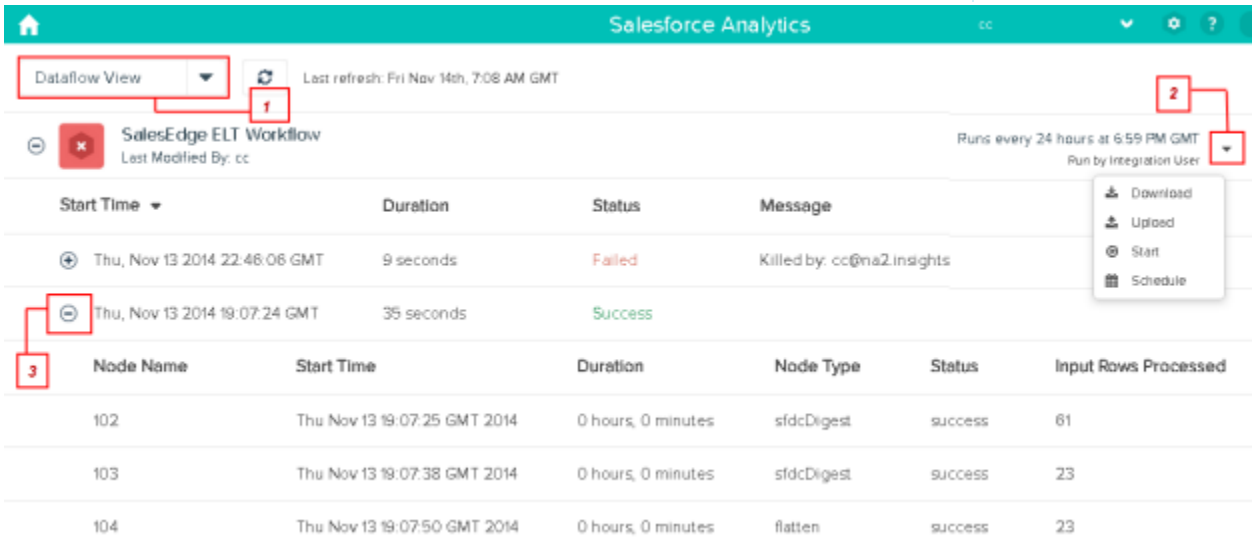
Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To access the data monitor:

- "Edit Wave Analytics Dataflows," "Upload External Data to Wave Analytics," or "Manage Wave Analytics"



1. In Wave Analytics, click the gear button ( ), and then click **Data Monitor** to open the data monitor. The Jobs view shows by default.
2. Select **Dataflow View** (1).
3. Click the Refresh Jobs button ( ) to see the latest status of a job.

   Each job can have one of the following statuses:.

Status	Description
Running	The job is running.
Failed	The job failed.
Successful	The job completed successfully.

4. If the dataflow job fails, expand the job node (3) and review the run-time details for every transformation that was processed.
5. If there's a problem with the dataflow logic, edit the dataflow definition file, upload it, and then run the dataflow again.

Reschedule the Dataflow

The dataflow job runs on a daily schedule. You can change the time that the dataflow job runs. You might change the time to ensure that the data is available by a particular time or to run the job during non-business hours.

1. In Wave Analytics, click the gear icon ( ) and then click **Data Monitor** to open the data monitor.
   The Jobs view shows by default.
2. Select **Dataflow View**.

EDITIONS

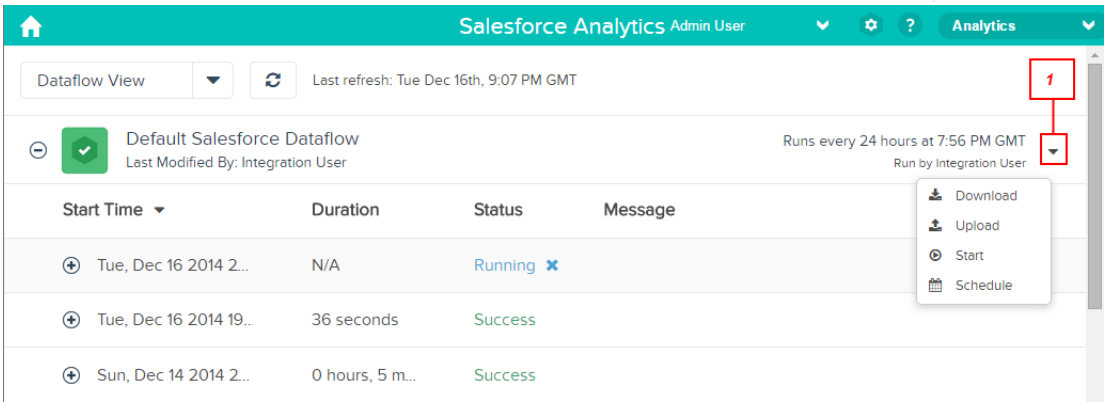
Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To start a dataflow job:
- "Edit Wave Analytics Dataflows"



3. Click **Schedule** in the actions list (1) to reschedule the dataflow job.
   The Dataflow Schedule dialog box appears.

4. Select the time that you want the dataflow to run.
5. Click **Save**.
6. Click **Done**.

DATAFLOW TRANSFORMATION REFERENCE

Transformations

A *transformation* refers to the manipulation of data. You can add transformations to a dataflow to extract data from Salesforce objects or datasets, transform datasets that contain Salesforce or external data, and register datasets.

For example, you can use transformations to join data from two related datasets and then register the resulting dataset to make it available for queries.

IN THIS SECTION:

append Transformation

The append transformation combines records from multiple datasets into a single dataset.

augment Transformation

The augment transformation augments an input dataset by combining columns from another related dataset. The resulting, augmented dataset enables queries across both related input datasets. For example, you can augment the Account dataset with the User dataset to enable a query to return account records and the full names of the account owners.

computeExpression Transformation

The computeExpression transformation enables you to add derived fields to a dataset. The values for derived fields aren't extracted from the input data source. Instead, Wave generates the values using a SAQL expression, which can be based on one or more fields from the input data or other derived fields. For example, you can use an expression to assign a value to a field, concatenate text fields, or perform mathematical calculations on numeric fields.

computeRelative Transformation

You can use the computeRelative transformation to analyze trends in your data by adding derived fields to a dataset based on values in other rows. For example, to analyze sales pipeline trends, create derived fields that calculate the number of days an opportunity remains in each stage. You can also calculate the changes to the opportunity amount throughout the stages of the opportunity.

delta Transformation

The delta transformation calculates changes in the value of a measure column in a dataset over a period of time. The delta transformation generates an output column in the dataset to store the delta for each record. Create deltas to make it easier for business analysts to include them in queries.

dim2mea Transformation

The dim2mea transformation creates a new measure based on a dimension. The transformation adds the new measure column to the dataset. The transformation also preserves the dimension to ensure that existing lenses and dashboards don't break if they use the dimension.

edgemart Transformation

The edgemart transformation gives the dataflow access to an existing, registered dataset, which can contain Salesforce data, external data, or a combination of the two. Use this transformation to reference a dataset so that it's data can be used in subsequent transformations in the dataflow. You can use this transformation and the Augment transformation together to join an existing dataset with a new dataset.

filter Transformation

The filter transformation removes records from an existing dataset. You define a filter condition that specifies which records to retain in the dataset.

flatten Transformation

The flatten transformation flattens hierarchical data. For example, you can flatten the Salesforce role hierarchy to implement row-level security on a dataset based on the role hierarchy.

sfdcDigest Transformation

The sfdcDigest transformation generates a dataset based on data that it extracts from a Salesforce object. You specify the Salesforce object and fields from which to extract data. You might choose to exclude particular fields that contain sensitive information or that aren't relevant for analysis.

sfdcRegister Transformation

The sfdcRegister transformation registers a dataset to make it available for queries. Users cannot view or run queries against unregistered datasets.

update Transformation

The update transformation updates the specified field values in an existing dataset based on data from another dataset, which we'll call the lookup dataset. The transformation looks up the new values from corresponding fields in the lookup dataset. The transformation stores the results in a new dataset.

append Transformation

The append transformation combines records from multiple datasets into a single dataset.

Consider the following rules when using this transformation.

- This transformation does not remove duplicate records.
- All input datasets must have the same structure—the corresponding columns must be in the same order and have the same name and field type.

Example: Let's look at an example. Each month, you create a dataset that contains the month's sales targets. Now, you want a holistic view of sales targets for all months. To do that, you create the following dataflow definition file to merge the existing datasets into a single dataset.

```
{
   "Extract_SalesTargets_2014Jan": {
      "action": "edgemart",
      "parameters": { "alias": "SalesTargets_2014Jan" }
   },
   "Extract_SalesTargets_2014Feb": {
      "action": "edgemart",
      "parameters": { "alias": "SalesTargets_2014Feb" }
   },
   "Extract_SalesTargets_2014Mar": {
      "action": "edgemart",
      "parameters": { "alias": "SalesTargets_2014Mar" }
   },
   "Append_SalesTargets_2014Quarter1": {
      "action": "append",
      "parameters": {
         "sources": [
            "Extract_SalesTargets_2014Jan",
```

```
            "Extract_SalesTargets_2014Feb",
            "Extract_SalesTargets_2014Mar"
         ]
      }
   },
   "Register_AllSalesTargets": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "AllSalesTargets",
         "name": "AllSalesTargets",
         "source": "Append_SalesTargets_2014Quarter1"
      }
   }
}
```

After you create the single dataset, you can use date filters to analyze the sales targets by month, quarter, or year.

IN THIS SECTION:

append Parameters
When you define an append transformation, you set the action attribute to `append` and specify the parameters.

append Parameters

When you define an append transformation, you set the action attribute to `append` and specify the parameters.

The following table describes the input parameters.

Parameter	Required?	Value
sources	Yes	Nodes in the dataflow definition file that identify the datasets that you want to merge.

SEE ALSO:

append Transformation

augment Transformation

The augment transformation augments an input dataset by combining columns from another related dataset. The resulting, augmented dataset enables queries across both related input datasets. For example, you can augment the Account dataset with the User dataset to enable a query to return account records and the full names of the account owners.

When you create the transformation, you identify each input dataset as the left or right dataset and specify the relationship between them. Wave Analytics combines all the columns of the left dataset with only the specified columns from the right dataset. (Keep in mind that each dataset can't have more than 5,000 columns.) Wave Analytics adds the relationship to column names from the right dataset, which is useful when the left and right datasets have columns with the same names.

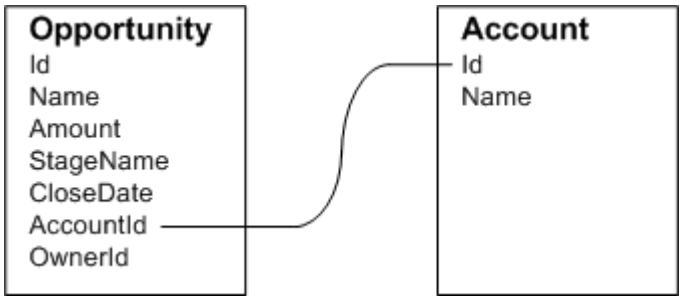
For each record in the left dataset, the augment transformation performs a lookup to find a matching record in the right dataset. To match related records, the augment transformation uses a match condition. You specify the match condition based on a key from each dataset. For example:

```
"left_key": [ "Id" ],"right_key": [ "Prod_ID" ]
```

A key can be a single-column key or a composite key. For a match condition based on a composite key, the keys for both datasets must have the same number of columns, specified in the same order.

Tip: To augment more than two datasets, augment two datasets at a time. For example, to augment three datasets, augment the first two datasets, and then augment the resulting dataset with the third dataset.

Example: Let's look at an example of the augment transformation. In this example, you want to extract data from the Opportunity and Accounts objects, and then match the data based on the account ID field.

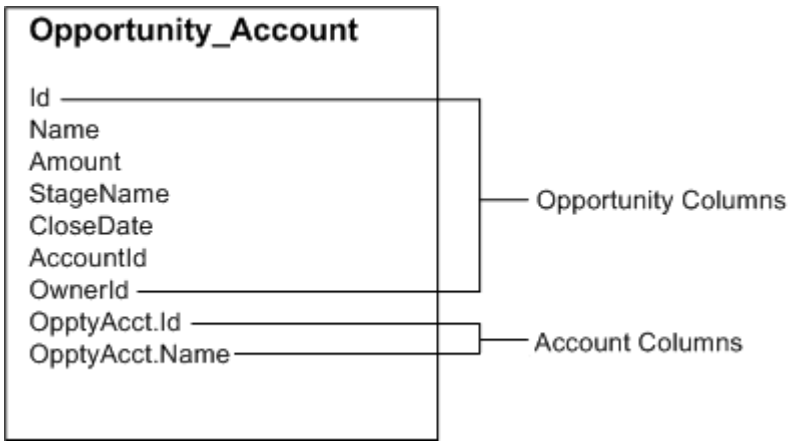


You create the following dataflow definition file.

```
{
    "Extract_Opportunity": {
        "action": "sfdcDigest",
        "parameters": {
            "object": "Opportunity",
            "fields": [
                { "name": "Id" },
                { "name": "Name" },
                { "name": "Amount" },
                { "name": "StageName" },
                { "name": "CloseDate" },
                { "name": "AccountId" },
                { "name": "OwnerId" }
            ]
        }
    },
    "Extract_AccountDetails": {
          "action": "sfdcDigest",
          "parameters": {
              "object": "Account",
              "fields": [
                  { "name": "Id" },
                  { "name": "Name" }
              ]
          }
    },
    "Augment_OpportunitiesWithAccountDetails": {
```

```
        "action": "augment",
        "parameters": {
            "operation": "LookupSingleValue",
            "left": "Extract_Opportunity",
            "left_key": [
                "AccountId"
            ],
            "relationship": "OpptyAcct",
            "right": "Extract_AccountDetails",
            "right_key": [
                "Id"
            ],
            "right_select": [ "Name" ]
        }
    },
    "Register_OpportunityDetails": {
        "action": "sfdcRegister",
        "parameters": {
            "alias": "Opportunity_Account",
            "name": "Opportunity_Account",
            "source": "Augment_OpportunitiesWithAccountDetails" }
    }
}
```

After you run the dataflow, Wave Analytics creates and registers the Opportunity_Account dataset. It also adds the relationship as a prefix to all columns from the right dataset.



IN THIS SECTION:

Special Cases for Matching Records with the augment Transformation

For each record in the left dataset, the augment transformation performs a lookup to find a matching record in the right dataset. However, it's critical that you understand how the augment transformation handles special cases when matching records.

augment Parameters

When you define an augment transformation, you set the action attribute to `augment` and specify the parameters.

Special Cases for Matching Records with the augment Transformation

For each record in the left dataset, the augment transformation performs a lookup to find a matching record in the right dataset. However, it's critical that you understand how the augment transformation handles special cases when matching records.

Let's look at some examples that illustrate some special cases.

Handling Null Keys

When a record in the left dataset contains a null key, Wave Analytics doesn't perform a lookup to match the record. Instead, Wave Analytics appends the right columns and inserts null for dimensions (including dates) and '0' for measures.

Let's look at an example. You apply the augment transformation on the following datasets, set the relationship to "Price,", and match the records based on the `Id` and `ProdId` fields.

Product	
Id	**Name**
Prod1	Table
Prod2	Chair
null	Bench

Price		
ProdId	**Pricebook**	**UnitPrice**
Prod1	Standard	1000
Prod2	Standard	450
null	Custom	800
Prod3	Standard	700

Wave Analytics doesn't match the last record because the product ID is null. Instead, Wave Analytics inserts a null for the `Price.Pricebook` dimension and '0' for the `Price.UnitPrice` measure. Here's the resulting dataset after the augment.

```
{ "id":"Prod1", "Name":"Table",   "Price.Pricebook":"Standard", "Price.UnitPrice":1000 },
{ "id":"Prod2", "Name":"Chair",   "Price.Pricebook":"Standard", "Price.UnitPrice":450  },
{ "id":null,    "Name":"Bench",   "Price.Pricebook":null,       "Price.UnitPrice":0    }
```

Handling Empty Keys

Wave Analytics matches empty-value left keys and empty-value right keys.

Let's look at an example. You apply the augment transformation on the following datasets, set the relationship to "Price,", and match the records based on the `Id` and `ProdId` fields.

Product	
Id	**Name**
Prod1	Table
Prod2	Chair
""	Bench

Price		
ProdId	**Pricebook**	**UnitPrice**
Prod1	Standard	1000
Prod2	Standard	450
""	Custom	800
null	Standard	700

Wave Analytics matches the last record in the Product dataset with the third record in the Price dataset because they both have empty values (""). Here's the resulting dataset after the augment.

{ "id":"Prod1", "Name":"Table", "Price.Pricebook":"Standard", "Price.UnitPrice":1000 },
{ "id":"Prod2", "Name":"Chair", "Price.Pricebook":"Standard", "Price.UnitPrice":450 },
{ "id":"", "Name":"Bench", "Price.Pricebook":"Custom", "Price.UnitPrice":800 }

Handling Non-Unique Keys

Although it's recommended, the left key doesn't have to be unique. If multiple records have the same left key, Wave Analytics creates the same values for the appended columns.

Let's look at an example. You apply the augment transformation on the following datasets, set the relationship to "Price,", and match the records based on the `Id` and `ProdId` fields.

Product	
Id	**Name**
Prod1	Table
Prod2	Chair
Prod2	Chair

Price		
ProdId	**Pricebook**	**UnitPrice**
Prod1	Standard	1000
Prod2	Standard	450
Prod3	Standard	700

Wave Analytics matches the records in the Product dataset with records in the Price dataset. Here's the resulting dataset after the augment.

{ "id":"Prod1", "Name":"Table", "Price.Pricebook":"Standard", "Price.UnitPrice":1000 },
{ "id":"Prod2", "Name":"Chair", "Price.Pricebook":"Standard", "Price.UnitPrice":450 },
{ "id":"Prod2", "Name":"Chair", "Price.Pricebook":"Standard", "Price.UnitPrice":450 }

Handling No Match

If left key doesn't have a match in the right dataset, Wave Analytics appends the right columns and inserts null for dimensions (including dates) and '0' for measures.

Let's look at an example. You apply the augment transformation on the following datasets, set the relationship to "Price,", and match the records based on the `Id` and `ProdId` fields.

Product	
Id	**Name**
Prod1	Table
Prod2	Chair
Prod3	Bench

Price		
ProdId	**Pricebook**	**UnitPrice**
Prod4	Standard	1000
Prod5	Standard	450

Because no keys match, Wave Analytics doesn't match any records in the Product dataset with records in the Price dataset. Here's the resulting dataset after the augment.

{ "id":"Prod1", "Name":"Table", "Price.Pricebook": *null*, "Price.UnitPrice":0 },
{ "id":"Prod2", "Name":"Chair", "Price.Pricebook": *null*, "Price.UnitPrice":0 },
{ "id":"Prod3", "Name":"Bench", "Price.Pricebook": *null*, "Price.UnitPrice":0 }

Handling Multiple Matches

If the left dataset has a one-to-many relationship with the right dataset, Wave Analytics might find multiple matches for a left record. What Wave Analytics does with multiple matches depends on the specified augment operation. You can specify one of the following operations to handle the multiple-match case:

`LookupSingleValue`

The augment transformation returns results from a single row. Wave Analytics randomly selects one row from the list of matched rows.

Note: Each time you run the dataflow, Wave Analytics can return different results depending on the returned row.

Let's look at an example. You apply the augment transformation on the following datasets, set the relationship to "Price," set the operation to `LookupSingleValue`, and match the records based on the `Id` and `ProdId` fields.

Product	
Id	**Name**
Prod1	Table
Prod2	Chair
Prod3	Bench

Price		
ProdId	**Pricebook**	**UnitPrice**
Prod1	Standard	1000
Prod2	Standard	450
Prod3	Custom	800
Prod3	Standard	700

Although there are multiple rows for Prod3 in the Price dataset, Wave Analytics randomly chooses one matching row and returns values based on that row. Here's the resulting dataset after the augment if Wave Analytics chooses the first Prod3 row.

{ "id":"Prod1", "Name":"Table", "Price.Pricebook":"Standard", "Price.UnitPrice":1000 },
{ "id":"Prod2", "Name":"Chair", "Price.Pricebook":"Standard", "Price.UnitPrice":450 },
{ "id":"Prod3", "Name":"Bench", "Price.Pricebook":"Custom", "Price.UnitPrice":800 }

`LookupMultiValue`

Wave Analytics returns results from all matched rows.

Let's look at an example. You apply the augment transformation on the following datasets, set the relationship to "Price," set the operation to `LookupMultiValue`, and match the records based on the `Id` and `ProdId` fields.

Product	
Id	**Name**
Prod1	Table
Prod2	Chair
Prod3	Bench

Price		
ProdId	**Pricebook**	**UnitPrice**
Prod1	Standard	1000
Prod2	Standard	450
Prod3	Custom	800
Prod3	Standard	700

Because the lookup returns multiple rows for Prod3, the dimension `Price.Pricebook` field in the resulting dataset becomes a multi-value field, showing all dimension values. The measure field `Price.UnitPrice` contains 1500, which is the sum of 800 and 700. Here's the resulting dataset after the augment.

```
{ "id":"Prod1", "Name":"Table",   "Price.Pricebook":"Standard",   "Price.UnitPrice":1000          },
{ "id":"Prod2", "Name":"Chair",   "Price.Pricebook":"Standard",   "Price.UnitPrice":450           },
{ "id":"Prod3", "Name":"Bench", "Price.Pricebook":"Custom",     "Price.Pricebook":"Standard", "Price.UnitPrice":1500 }
```

augment Parameters

When you define an augment transformation, you set the action attribute to `augment` and specify the parameters.

The following table describes the input parameters.

Parameter	Required?	Value
operation	No	Indicates what the transformation does if it matches multiple rows in the right dataset with a row in the left. Valid values: • `LookupSingleValue`. Returns values from one of the matched rows. If you don't specify the `operation` parameter, the transformation uses this operation. • `LookupMultiValue`. Returns values from all matched rows. For more information about each operation, see Special Cases for Matching Records with the augment Transformation.
left	Yes	Node in the dataflow definition file that identifies the left dataset. This is one of two input sources for this transformation.
left_key	Yes	Key column in the left dataset used to augment the datasets. If you use a composite key, the left and right keys must have the same number of columns in the same order. For a composite key, use the following syntax: `[ "Key Column1", "Key Column2", …, "Key ColumnN" ]` Note: The left or right key can't be a multi-value field.
right	Yes	Node in the dataflow definition file that identifies the right dataset. This is one of two input sources for this transformation.

Parameter	Required?	Value
relationship	Yes	Relationship between the left and right datasets. The dataflow adds the relationship to the beginning of the right column names in the output dataset to make the column names unique and descriptive.
right_select	Yes	An array of column names from the right dataset that you want to include in the output dataset. The dataflow adds the relationship as a prefix to the column name to determine the name of the right column in the output dataset.
right_key	Yes	Key column in the right dataset used to augment the datasets. If you use a composite key, the left and right keys must have the same number of columns in the same order.

SEE ALSO:

augment Transformation

computeExpression Transformation

The computeExpression transformation enables you to add derived fields to a dataset. The values for derived fields aren't extracted from the input data source. Instead, Wave generates the values using a SAQL expression, which can be based on one or more fields from the input data or other derived fields. For example, you can use an expression to assign a value to a field, concatenate text fields, or perform mathematical calculations on numeric fields.

Note: The computeExpression and computeRelative transformations are similar, but they have a key difference. The computeExpression transformation performs calculations based on other fields within the same row. The computeRelative transformation performs calculations based on the previous and next values of the same field in other rows.

Consider the following guidelines when creating a computeExpression transformation:

- You can include only the following SAQL operators and functions in the expression:
  - Arithmetic operators
  - Case operator
  - String operator
  - Date functions
- The values of the derived field must match its specified type. For example, set the type of the derived field to `Text` if the values are strings.

- Wave calculates the values of derived fields in the order in which they are listed in the JSON. Thus, if you create a derived field based on other derived fields in the same computeExpression transformation, the input derived fields must be listed first. For example, Derived_A must be listed before Derived_B in the following computeExpression transformation JSON snippet:

```
"CreateDerivedFields": {
   "action": "computeExpression",
   "parameters": {
      "source": "sourceNode",
      "mergeWithSource": false,
      "computedFields": [
         {
            "name": "Derived_A",
            "type": "Text",
            "label": "Derived Field A",
            "saqlExpression": "\"hello \""},
         {
            "name": "Derived_B",
            "type": "Text",
            "label": "Derived Field B Dependent on Field A",
            "saqlExpression": "Derived_A + \"world\""}
         ]
      }
   }
```

- You can choose whether the resulting dataset includes only the derived fields, or includes the input and derived fields.

Example: Let's look at an example. You want to create a dataset based on Salesforce opportunity data. You create a dataflow that extracts the Id and Amount fields from the Opportunity object. In addition, you also want to add the following derived fields to the dataset: ModifiedId, SalesTax, FinalPrice, and ValueCategory. For the derived fields, you will:

- Append "SFDC" to each opportunity Id to get a new modified Id.
- Calculate the sales tax based on an 8% tax rate.
- Calculate the final price by adding the amount and sales tax.
- Categorize opportunities into low-, medium-, and high-value buckets based on the calculated final price.

You create the following dataflow definition.

```
{
   "salesData": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            {"name": "Amount"},
            {"name": "Id"}]}},
   "Derived_Fields": {
      "action": "computeExpression",
      "parameters": {
         "source": "salesData",
         "mergeWithSource": true,
         "computedFields": [
            {
               "name": "ModifiedId",
               "type": "Text",
               "saqlExpression": "\"SFDC\" + Id"},
```

```
            {
               "name": "SalesTax",
               "type": "Numeric",
               "precision": 18,
               "defaultValue": "0",
               "scale": 5,
               "saqlExpression": "Amount * 0.08"},
            {
               "name": "FinalPrice",
               "type": "Numeric",
               "precision": 18,
               "defaultValue": "0",
               "scale": 5,
               "saqlExpression": "Amount + SalesTax"},
            {
               "name": "ValueCategory",
               "type": "Text",
               "saqlExpression": "
                  case
                     when FinalPrice < 1000 then \"Low\"
                     when FinalPrice >= 1000 and FinalPrice < 2000 then \"Medium\"
                     else \"High\"
                  end"}
         ]
      }
   },
   "Register_CategorizedSales": {
       "action": "sfdcRegister",
       "parameters": {
           "alias": "Categorized_Sales",
           "name": "Categorized_Sales",
           "source": "Derived_Fields" }
   }
}
```

IN THIS SECTION:

computeExpression Parameters
When you define a computeExpression transformation, you set the action attribute to `computeExpression`. You also specify the parameters for the input source and the expression used to generate the values.

computeExpression Parameters

When you define a computeExpression transformation, you set the action attribute to `computeExpression`. You also specify the parameters for the input source and the expression used to generate the values.

You can specify parameters in the following sections of the computeExpression node: parameters and computedFields.

Parameters

The following table describes the parameters in the `parameters` section.

Parameter	Required?	Value
source	Yes	Node in the dataflow definition file that identifies the input source for this transformation.
mergeWithSource	No	Indicates whether the input fields are included with the derived fields in the resulting dataset. When true, the resulting dataset contains all input fields from the source and the newly generated derived fields. When false, the resulting dataset contains the derived fields only. Default is true.
computedFields	Yes	Attributes and expression used to generate derived fields in the dataset. See computedFields.

computedFields

The following table describes the attributes in the `computedFields` section. It also describes optional attributes that you can provide to override the field metadata to make the data appear differently in a dataset. For example, Wave can replace null values in a field with a default value.

Parameter	Required?	Value
name	Yes	API name of the generated field. Note: The API names must be unique. Otherwise, the dataflow fails to run.
type	Yes	Wave Analytics field type associated with the field. Valid types are Text, Numeric, or Date. Example: `"type": "Text"`
label	No	The display name of the generated field that appears in the Wave user interface. Can be up to 255 characters. Defaults to input field name if not specified.
saqlExpression	Yes	SAQL expression used to calculate the value for the derived field. The expression can be based on input fields or other derived fields in the transformation.

Parameter	Required?	Value
		Example: `"saqlExpression":"toDate(birth_day, \"yyyy-M-d\")"`
format	Yes (for Date fields only)	Format of the derived date field. For information about formats, see the External Data Format Reference.
precision	Yes (for Numeric fields only)	The maximum number of digits in a numeric value, or the length of a text value. For numeric values: Includes all numbers to the left and to the right of the decimal point (but excludes the decimal point character). Value must be between 1 and 18, inclusive. For text values: Value defaults to 255 characters, and must be between 1 and 32,000 characters, inclusive. Example: `"precision": 10`
scale	Yes (for Numeric fields only)	The number of digits to the right of the decimal point in a numeric value. Must be less than the precision value. Value must be between 0 and 17 characters, inclusive. Example: `"scale": 2`
defaultValue	No	For text and numeric fields that can be null. Default value that replaces a null value for the specified field.

SEE ALSO:

computeExpression Transformation

computeRelative Transformation

You can use the computeRelative transformation to analyze trends in your data by adding derived fields to a dataset based on values in other rows. For example, to analyze sales pipeline trends, create derived fields that calculate the number of days an opportunity remains in each stage. You can also calculate the changes to the opportunity amount throughout the stages of the opportunity.

Note: The computeExpression and computeRelative transformations are similar, but the computeExpression transformation performs calculations based on fields within the same row. The computeRelative transformation performs calculations based on the same field in other rows—particularly the current, first, previous, or next rows.

When you define a computeRelative transformation, you specify an sfdcDigest transformation as the input, partition the records, and sort the records within each partition. For example, you can use sfdcDigest to extract opportunity history records, and then use computeRelative to calculate changes in each opportunity over time. You can partition opportunity records by opportunity ID, and then chronologically sort records within each partition to correctly identify the previous and next values.

Wave processes the computeRelative transformation differently than other transformations. Wave merges the logic from this transformation into the associated sfdcDigest transformation. As a result, you don't have to register the computeRelative transformation to add the derived fields to the final dataset. Instead, register only sfdcDigest—the registered dataset has fields from both transformations. For example, the sfdcDigest transformation extracts data from three fields (Id, CloseDate, and Amount). The computeRelative transformation generates two derived fields (PreviousCloseDate and PreviousAmount). The resulting dataset contains all five columns.

Consider the following guidelines when creating a computeRelative transformation.

- This transformation supports only the sfdcDigest node as input—you can't use any other transformation node as input.
- Specify this transformation directly after the associated input sfdcDigest transformation in the dataflow JSON.
- You can use this transformation to analyze trends in Salesforce data, not external data.
- You can use this transformation in a replication dataflow. For more information about replication, see Getting Started with Wave Replication Tipsheet.

Example: Let's look at an example. To perform trending analysis on the sales pipeline, create a dataflow that contains the following transformations.

sfdcDigest transformation

Extracts the following data from the OpportunityHistory object.

OpportunityHistory Object

OpportunityId	CreatedDate	StageName	Amount	CloseDate
00630000003JrH6AAK	2005-11-23T20:24:09.000Z	Closed Won	36938	11/15/2005
00630000003SYADAA4	2005-11-08T18:47:18.000Z	Present/Demo/Quote	4200	12/3/2005
00630000003Yx9BAAS	2005-11-23T01:06:57.000Z	Present/Demo/Quote	71500	11/30/2005
00630000003JsMcAAK	2005-10-28T18:12:06.000Z	Negotiation/Review	18750	11/15/2005
00630000003Yx9BAAS	2005-11-25T18:03:46.000Z	Closed Won	5832	11/23/2005
00630000003Yx9BAAS	2005-11-23T01:07:46.000Z	Present/Demo/Quote	7128	11/30/2005
00630000003JrH6AAK	2005-11-03T21:53:36.000Z	Negotiation/Review	20148	11/15/2005
00630000003SZAZAA4	2005-11-08T19:44:27.000Z	Present/Demo/Quote	141120	12/16/2005
00630000003JrH6AAK	2005-10-17T22:28:33.000Z	Present/Demo/Quote	42900	11/15/2005
00630000003JsMcAAK	2005-10-17T23:30:20.000Z	Negotiation/Review	18750	10/31/2005
00630000003OG95AAG	2005-10-28T17:45:30.000Z	Present/Demo/Quote	52000	11/30/2005

computeRelative transformation

Performs the following tasks:

- Partitions the extracted records by opportunity ID.
- Within each partition, sorts the extracted records by CreatedDate in ascending order. Sorting by CreatedDate ensures that the changes that occur for each opportunity are listed in chronological order.
- Adds the following derived fields to the final dataset.

  OpportunityCreatedDate

  Determines the date that the opportunity was first created. You can use this date with the actual close date to determine the number of days required to close the sale. The goal is to shorten the sales cycle to recognize revenue.

  ValidToDate

  Determines the date to which the state of the opportunity is valid. You can use this field to determine the time period for which the opportunity details are valid in each record.

AmountPrev
Determines the previous amount of the opportunity. You can use this field to determine if the values of opportunities are increasing or decreasing, which can affect whether you hit your sales targets.

CloseDatePrev
Determines the previous expected close date for the opportunity. You can use this field to analyze how the expected close date changes over the sales cycle of the opportunity. If the expected close date keeps getting pushed out, identify the issues that are causing the longer sales cycle.

sfdcRegister transformation
Registers the final dataset that contains the extracted fields from the sfdcDigest transformation and the derived fields from computeRelative transformation. You apply sfdcRegister on sfdcDigest to get the fields from both transformations.

You create the following dataflow definition.

```
{
    "extractOppHistory": {
        "action": "sfdcDigest",
        "parameters": {
            "object": "OpportunityHistory",
            "fields": [
                {"name": "OpportunityId"},
                {"name": "CreatedDate"},
                {"name": "StageName"},
                {"name": "Amount"},
                {"name": "CloseDate"}
            ]
        }
    },
    "computeTrending": {
        "action": "computeRelative",
        "parameters": {
            "partitionBy": ["OpportunityId"],
            "orderBy": [
                {
                    "name":"CreatedDate",
                    "direction":"asc"
                }
            ],
            "computedFields": [
                {
                    "name": "OpportunityCreatedDate",
                    "expression": {
                        "sourceField": "CreatedDate",
                        "offset": "first()",
                        "default": "current()"
                    }
                },
                {
                    "name": "ValidToDate",
                    "expression": {
                        "sourceField": "CreatedDate",
                        "offset": "next()",
                        "default": "3000-01-01T00:00:00.000Z"
                    }
```

```
                },
                {
                    "name": "AmountPrev",
                    "expression": {
                        "sourceField": "Amount",
                        "offset": "previous()",
                        "default": "0"
                    }
                },
                {
                    "name": "CloseDatePrev",
                    "expression": {
                        "sourceField": "CloseDate",
                        "offset": "previous()",
                        "default": "01/01/1970"
                    }
                }
            ],
            "source": "extractOppHistory"
        }
    },
    "Register_OppportunityHistory_Dataset": {
        "action": "sfdcRegister",
        "parameters": {
            "alias": "SalesPipelineTrending",
            "name": "Sales Pipeline Trending",
            "source": "extractOppHistory"
      }
   }
}
```

The dataflow runs and creates the following dataset.

Sales Pipeline Trending Dataset

OpportunityId	CreatedDate	StageName	Amount	CloseDate	OpportunityCreatedDate	ValidToDate	AmountPrev	CloseDatePrev
00630000003JrH6AAK	2005-10-17T22:28:33.000Z	Present/Demo/Quote	429000	11/15/2005	2005-10-17T22:28:33.000Z	2005-11-03T21:53:36.000Z	0	1/1/1970
00630000003JrH6AAK	2005-11-03T21:53:36.000Z	Negotiation/Review	201480	11/15/2005	2005-10-17T22:28:33.000Z	2005-11-23T20:24:09.000Z	429000	11/15/2005
00630000003JrH6AAK	2005-11-23T20:24:09.000Z	Closed Won	369380	11/15/2005	2005-10-17T22:28:33.000Z	3000-01-01T00:00:00.000Z	201480	11/15/2005
00630000003JsMcAAK	2005-10-17T23:30:20.000Z	Negotiation/Review	187500	10/31/2005	2005-10-17T23:30:20.000Z	2005-10-28T18:12:06.000Z	0	1/1/1970
00630000003JsMcAAK	2005-10-28T18:12:06.000Z	Negotiation/Review	187500	11/15/2005	2005-10-17T23:30:20.000Z	3000-01-01T00:00:00.000Z	187500	10/31/2005
00630000003OG95AAG	2005-10-28T17:45:30.000Z	Present/Demo/Quote	520000	11/30/2005	2005-10-28T17:45:30.000Z	3000-01-01T00:00:00.000Z	0	1/1/1970
00630000003SYADAA4	2005-11-08T18:47:18.000Z	Present/Demo/Quote	42000	12/3/2005	2005-11-08T18:47:18.000Z	3000-01-01T00:00:00.000Z	0	1/1/1970
00630000003SZAZAA4	2005-11-08T19:44:27.000Z	Present/Demo/Quote	1411200	12/16/2005	2005-11-08T19:44:27.000Z	3000-01-01T00:00:00.000Z	0	1/1/1970
00630000003Yx9BAAS	2005-11-23T01:06:57.000Z	Present/Demo/Quote	715000	11/30/2005	2005-11-23T01:06:57.000Z	2005-11-23T01:07:46.000Z	0	1/1/1970
00630000003Yx9BAAS	2005-11-23T01:07:46.000Z	Present/Demo/Quote	71280	11/30/2005	2005-11-23T01:06:57.000Z	2005-11-25T18:03:46.000Z	715000	11/30/2005
00630000003Yx9BAAS	2005-11-25T18:03:46.000Z	Closed Won	58320	11/23/2005	2005-11-23T01:06:57.000Z	3000-01-01T00:00:00.000Z	71280	11/30/2005

Notice that Wave partitions the records by opportunity ID and then sorts the records in ascending order based on the CreatedDate field within each partition. Wave can now use the previous and next rows within each partition to determine changes in field values in the dataset.

IN THIS SECTION:

computeRelative Parameters

When you define a computeRelative transformation, you set the action attribute to `computeRelative`. You also specify the parameters for the input source, partition-by field, sort field, and derived field definitions.

computeRelative Parameters

When you define a computeRelative transformation, you set the action attribute to `computeRelative`. You also specify the parameters for the input source, partition-by field, sort field, and derived field definitions.

You can specify parameters in the following sections of the computeRelative node.

Parameters

The following table describes the parameters in the `parameters` section.

Parameter	Required?	Value
source	Yes	sfdcDigest node in the dataflow definition file that identifies the input source for this transformation.
partitionBy	Yes	API name of the field used to partition the records in the dataset. Specify one and only one partition-by field.
orderBy	Yes	Field used to sort the records within each partition and the sort order: ascending (`asc`) or descending (`dsc`). Specify one and only one sort field. Example: `"orderBy": [ { "name":"CreatedDate", "direction":"asc" } ]`
computedFields	Yes	A list of definitions for derived fields. Example: `"computedFields": [ { "name": "OpportunityCreatedDate", "expression": { "sourceField": "CreatedDate", "offset": "first()", "default": "current()" } }, { "name": "AmountPrev", "expression": { "sourceField": "Amount", "offset": "previous()", "default": "0" } } ]` See computedFields.

computedFields

The following table describes the attributes in the `computedFields` section.

Parameter	Required?	Value
name	Yes	API name of the derived field to add to the dataset. The name must be unique in the dataset. Note: If the name is not unique, the dataflow fails to run.
expression	Yes	Expression attributes used to calculate the value for the derived field. The expression can be based on input fields or other derived fields in the transformation. Example: `"expression": { "sourceField": "CloseDate", "offset": "previous()", "default": "01/01/1970"` See expression.

expression

The following table describes the attributes in the `expression` section.

Parameter	Required?	Value
sourceField	Yes	API name of the input field from the sfdcDigest transformation that's used in the expression.

offset	Yes	The function used in the expression. You can use the following functions:
		`current()`
		Gets the value from the current record.
		Example:
		`"offset": "current()"`
		`first()`
		Gets the value from the first record in the partition, like the first CreateDate for an opportunity.
		Example:
		`"offset": "first()"`
		`next()`
		Gets the value from the next record.

Parameter	Required?	Value

		Example:
		`"offset": "next()"`
		`previous()`
		Gets the value from the previous record.
		Example:
		`"offset": "previous()"`
		Note: Derived fields are computed in the order that they're defined. The calculation of a derived field can be based on the value from another derived field as long as it has already been defined. For example, `next()` can't access the value of a derived field in the next row.
		Tip: To get the correct results when using the `previous()` and `next()` functions, the computeRelative transformation requires you to sort the records.

default	Yes (for numeric fields only)	The default value if one can't be calculated. For example, you can specify a default value when no previous or next value exists. You can insert a constant value or `current()` as a default value.
		Examples:
		`"default": "3000-01-01T00:00:00.000Z"`
		`"default": "current()"`

delta Transformation

The delta transformation calculates changes in the value of a measure column in a dataset over a period of time. The delta transformation generates an output column in the dataset to store the delta for each record. Create deltas to make it easier for business analysts to include them in queries.

The delta transformation calculates each delta value by comparing the value in each record with the value in the previous record. Because records might not be sorted, the delta transformation orders the records before computing the delta values. To do this, the transformation sorts the data by the specified dimension, and then by the specified epoch date column.

Note: When Wave Analytics processes dates, it creates the following epoch date columns for each date processed:

Epoch Time Column	Description
<date_column_name>_sec_epoch	For example, if the date column is CloseDate, the generated epoch second column is CloseDate_sec_epoch. This column provides the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT).

Epoch Time Column	Description
<date_column_name>_day_epoch	For example, if the date column is CloseDate, the generated epoch day column is CloseDate_day_epoch. This column provides the number of days that have elapsed since January 1, 1970 (midnight UTC/GMT).

Example: Let's look at an example. You want to create an OppHistoryDelta dataset that contains opportunity history from the OpportunityHistory object and also calculates the deltas for opportunity amounts.

The OpportunityHistory object contains the following data.

OpportunityId	CloseDate	StageName	Amount
1	1/1/2014	New	100
2	1/1/2014	New	100
2	2/1/2014	ClosedWon	200
1	3/1/2014	ClosedWon	100

You create the following dataflow definition.

```
{
   "Extract_Opportunities": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "OpportunityHistory",
         "fields": [
            { "name": "OpportunityId" },
            { "name": "CloseDate" },
            { "name": "StageName" },
            { "name": "Amount" }
         ]
      }
   },
   "Calculate_Delta": {
      "action": "delta",
      "parameters": {
         "dimension": "OpportunityId",
         "epoch": "CloseDate_day_epoch",
         "inputMeasure": "Amount",
         "outputMeasure": "DeltaAmount",
         "source": "Extract_Opportunities"
      }
   },
   "Register_Dataset": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "OppHistoryDelta",
         "name": "OppHistoryDelta",
```

```
            "source": "Calculate_Delta"
        }
    }
}
```

To calculate the delta values for each opportunity amount, the delta transformation sorts the records by the dimension (OpportunityId) first, and then by time (CloseDate_day_epoch) as shown here.

OpportunityID	CloseDate	StageName	Amount
1	1/1/2014	New	100
1	3/1/2014	ClosedWon	100
2	1/1/2014	New	100
2	2/1/2014	ClosedWon	200

After the records are sorted, for each dimension (OpportunityId), the transformation compares the previous value to the next value to determine the delta for each record. The transformation creates the following dataset.

OpportunityId	CloseDate	StageName	Amount	DeltaAmount
1	1/1/2014	New	100	0
1	3/1/2014	ClosedWon	100	0
2	1/1/2014	New	100	0
2	2/1/2014	ClosedWon	200	100

For the first record of each dimension, the transformation inserts '0' for the delta value.

Note: If an opportunity contains multiple changes on the same day, you must sort the records on a shorter time interval. In this case, sort on CloseDate_sec_epoch column. Otherwise, records might not be sorted correctly, which means delta values will be incorrect.

IN THIS SECTION:

delta Parameters
When you define a delta transformation, you set the action attribute to `delta` and specify the parameters.

delta Parameters

When you define a delta transformation, you set the action attribute to `delta` and specify the parameters.

The following table describes the input parameters:

Parameter	Required?	Value
dimension	Yes	Dimension column in the dataset used to sort records when calculating the delta values.
epoch	Yes	Epoch date column in the dataset used to sort records within each dimension when calculating delta values.
inputMeasure	Yes	Measure column on which you want to calculate the delta.
outputMeasure	Yes	Name of the output column that contains the delta value.
source	Yes	Node in the dataflow definition file that contains the dataset to which you want to add the delta column.

SEE ALSO:

delta Transformation

dim2mea Transformation

The dim2mea transformation creates a new measure based on a dimension. The transformation adds the new measure column to the dataset. The transformation also preserves the dimension to ensure that existing lenses and dashboards don't break if they use the dimension.

If the transformation cannot create a measure from a dimension, the transformation populates the measure with the specified default value. If no default value is provided, the transformation inserts '0.'

Example: Let's look at an example. Your Opportunity object contains a custom text field called StageVal__c, which contains the opportunity amount at a particular stage. Because this is a text field, Wave Analytics loads this data as a dimension. However, you'd like to create a measure from this dimension to enable users to perform calculations on stage amount.

You create the following dataflow definition.

```
{
   "Extract_Opportunities": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "Amount" },
            { "name": "StageName" },
            { "name": "CloseDate" },
            { "name": "AccountId" },
            { "name": "StageVal__c" }
         ]
      }
```

```
    },
    "Create_Measure_From_Dimension": {
        "action": "dim2mea",
        "parameters": {
            "dimension": "StageVal__c",
            "measure": "StageValue",
            "measureDefault": "0",
            "measureType": "long",
            "source": "Extract_Opportunities"
        }
    },
    "Register_The_Dataset": {
        "action": "sfdcRegister",
        "parameters": {
            "alias": "OpportunitiesWithConvertedMeasure",
            "name": "OpportunitiesWithConvertedMeasure",
            "source": "Create_Measure_From_Dimension"
        }
    }
}
```

IN THIS SECTION:

dim2mea Parameters
When you define a dim2mea transformation, you set the action attribute to `dim2mea` and specify the parameters.

dim2mea Parameters

When you define a dim2mea transformation, you set the action attribute to `dim2mea` and specify the parameters.

The following table describes the input parameters:

Parameter	Required?	Value
dimension	Yes	Dimension column in the dataset from which you want to create the measure.
measure	Yes	Name of the output measure. This column name must be unique within the dataset. Do not use the same name as the dimension because the transformation preserves the dimension in the dataset.
measureDefault	Yes	Default value for the measure if the transformation is unable to create a measure from a dimension.
measureType	Yes	Type of measure. Valid value: “long”

Parameter	Required?	Value
source	Yes	Node in the dataflow definition file that contains the dataset to which you want to add the measure.

SEE ALSO:

dim2mea Transformation

edgemart Transformation

The edgemart transformation gives the dataflow access to an existing, registered dataset, which can contain Salesforce data, external data, or a combination of the two. Use this transformation to reference a dataset so that it's data can be used in subsequent transformations in the dataflow. You can use this transformation and the Augment transformation together to join an existing dataset with a new dataset.

Example: Let's look at an example. You would like to compare the final sales amount against the opportunity amount to determine if heavy discounts were offered to close deals. You previously created and registered the FinalSales dataset. The FinalSales dataset contains the final sale amount of each opportunity that was closed and won.

Table 1: FinalSales Dataset

OppID	UpdateDate	StageName	SaleAmount
1	1/1/2014	ClosedWon	100,000
2	11/1/2013	ClosedWon	150,000
3	2/1/2014	ClosedWon	200,000

You would now like to create a dataset that contains opportunity information from the Opportunity object. Then, you would like to join the data from the existing FinalSales dataset with the Opportunity dataset.

You create the following dataflow definition.

```
{
    "Extract_Opportunities": {
        "action": "sfdcDigest",
        "parameters": {
            "object": "Opportunity",
            "fields": [
                { "name": "Id" },
                { "name": "Name" },
                { "name": "Amount" }
            ]
        }
    },
    "Extract_Final_Sales_Data": {
        "action": "edgemart",
        "parameters": { "alias": "FinalSales" }
    },
    "Combine_Opportunities_FinalSales": {
        "action": "augment",
```

```
        "parameters": {
            "left": "Extract_Opportunities",
            "left_key": [ "Id" ],
            "relationship": "Opportunity",
            "right": "Extract_Final_Sales_Data",
            "right_key": [ "OppID" ],
            "right_select": [ "SaleAmount" ]
        }
    },
    "Register_Opportunity_FinalSales_Dataset": {
        "action": "sfdcRegister",
        "parameters": {
            "alias": "OpportunityVersusFinalSales",
            "name": "OpporunityVersusFinalSales",
            "source": "Combine_Opportunities_FinalSales"
        }
    }
}
```

IN THIS SECTION:

edgemart Parameters
When you define an edgemart transformation, you set the action attribute to `edgemart` and specify the parameters.

edgemart Parameters

When you define an edgemart transformation, you set the action attribute to `edgemart` and specify the parameters.

The following table describes the input parameter:

Parameter	Required?	Value
alias	Yes	API name of the dataset from which you want to extract data. To determine the API name of a dataset, edit the dataset and view the system name.

SEE ALSO:

edgemart Transformation

filter Transformation

The filter transformation removes records from an existing dataset. You define a filter condition that specifies which records to retain in the dataset.

Example: Let's look at an example. You would like to create a dataset that contains only opportunities that were Closed Won. First, you extract all opportunities from the Opportunity object. Next, you filter the records so that you only include opportunities with a Closed Won stage name.

You create the following dataflow definition.

```
{
   "Extract_Opportunities": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "Amount" },
            { "name": "StageName" },
            { "name": "CloseDate" },
            { "name": "AccountId" },
            { "name": "OwnerId" }
         ]
      }
   },
   "Filter_Opportunities": {
      "action": "filter",
      "parameters": {
         "filter": "StageName:EQ:Closed Won",
         "source": "Extract_Opportunities"
      }
   },
   "Register_My_Won_Oppportunities_Dataset": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "MyWonOpportunities",
         "name": "MyWonOpportunities",
         "source": "Filter_Opportunities"
      }
   }
}
```

IN THIS SECTION:

filter Parameters

When you define a filter transformation, you set the action attribute to `filter` and specify the parameters.

filter Expression Syntax

You create a filter expression in the filter transformation based on one or more dimensions in a dataset.

filter Parameters

When you define a filter transformation, you set the action attribute to `filter` and specify the parameters.

The following table describes the input parameters:

Parameter	Required?	Value
filter	Yes	Filter expression that specifies which records to include in the new dataset. See filter Expression Syntax.
source	Yes	Node in the dataflow definition file that contains the dataset that you want to filter.

SEE ALSO:

filter Transformation

filter Expression Syntax

You create a filter expression in the filter transformation based on one or more dimensions in a dataset.

Note: String comparisons in a filter expression are case-sensitive.

You can use the following types of filter expressions:

Filter Expression Syntax	Description
dim:EQ:value	True if the dimension and value are equal. Example: `"filter": "StageName:EQ:Closed Won"`
dim:R:val0:val1	True if the left dimension falls within the specified range between val0 and val1. Example: `"filter": "EmployeeId:R:100:1000"`
dim:R:val	True if the dimension is greater than or equal to the value based on binary sort order. For example, this is true when the dimension is 'City' and the value is 'Anderson' because ' 'City' > 'Anderson'). Example: `"filter": "LastName:R:Li"`
dim:R::val	True if the dimension is less than or equal to the value based on binary sort order. Example: `"filter": "LastName:R::Levy"`
dim:N:val	True if the dimension and value are not equal. Example: `"filter": "RoleName:N:Manager"`
dim:EQ:val1|val2	True if the dimension equals values val1 or val2. This filter expression uses the logical OR operator (|). You can compare the dimension value against more than two values. For example, to compare against three values, use the following syntax: dim1:EQ:val1|val2|val3.

Filter Expression Syntax	Description
	Example: `"filter": "Lead Status:EQ:Open|Contacted"`
dim1:EQ:val1,dim2:EQ:val2	True if dimension dim1 equals value val1 and dimension dim2 equals value val2. This filter expression uses the logical AND operator (,). You can compare more than two dimensions. For example, to compare three dimensions, use the following syntax: dim1:EQ:val1,dim2:EQ:val2,dim3:EQ:val3. Example: `"filter": "Lead Status:EQ:Qualified,Rating:EQ:Hot"`

SEE ALSO:

filter Transformation

flatten Transformation

The flatten transformation flattens hierarchical data. For example, you can flatten the Salesforce role hierarchy to implement row-level security on a dataset based on the role hierarchy.

When you configure the flatten transformation to flatten a hierarchy, you specify the field that contains every node in the hierarchy and the field that contains their corresponding parent based on the hierarchy. The flatten transformation generates one record for each node in the hierarchy. Each record contains all ancestors up the hierarchy chain for each node.

The flatten transformation stores the list of ancestors in two formats in the resulting dataset. One column contains a comma-separated list of all ancestors for each node in the hierarchy. The other column contains the hierarchy path.

See the Roles and RolePath columns in the following flattened dataset to see how ancestors are stored.

Role ID	Role Name	Parent Role ID	Roles	RolePath
1	Salesperson 1	10	10, 20, 30	\10\20\30
2	Salesperson 2	10	10, 20, 30	\10\20\30
3	Salesperson 3	11	11, 20, 30	\11\20\30
10	Regional Manager 1	20	20, 30	\20\30
11	Regional Manager 2	20	20, 30	\20\30
20	Vice President 1	30	30	\30
21	Vice President 2	30	30	\30
30	CEO	Not applicable	Not applicable	Not applicable

Example: Let's look at an example. In this example, you want to create a dataset that contains all opportunities. For each opportunity, you want to include user and role information about the opportunity owner. Also, to implement row-level security

based on the role hierarchy, each opportunity record must also contain a list of all ancestors above each opportunity owner based on the role hierarchy. To generate the list of ancestors, you will use the flatten transformation to flatten the role hierarchy.

You create the following dataflow definition file:

```
{
"Extract_Opportunity": {
   "action": "sfdcDigest",
   "parameters": {
      "object": "Opportunity",
      "fields": [
         { "name": "Id" },
         { "name": "Name" },
         { "name": "Amount" },
         { "name": "StageName" },
         { "name": "AccountId" },
         { "name": "OwnerId" }
      ]
   }
},
"Extract_User": {
   "action": "sfdcDigest",
   "parameters": {
      "object": "User",
      "fields": [
         { "name": "Id" },
         { "name": "Name" },
         { "name": "Department" },
         { "name": "UserRoleId" }
      ]
   }
},
"Extract_UserRole": {
   "action": "sfdcDigest",
   "parameters": {
      "object": "UserRole",
      "fields": [
         { "name": "Id" },
         { "name": "Name" },
         { "name": "ParentRoleId" }
      ]
   }
},
"Flatten_UserRole": {
   "action": "flatten",
   "parameters": {
      "source": "Extract_UserRole",
      "self_field": "Id",
      "parent_field": "ParentRoleId",
      "multi_field": "Roles",
      "path_field": "RolePath"
   }
},
"Augment_User_FlattenUserRole": {
```

```
   "action": "augment",
   "parameters": {
      "left": "Extract_User",
      "left_key": [ "UserRoleId" ],
      "relationship": "Role",
      "right": "Flatten_UserRole",
      "right_key": [ "Id" ],
      "right_select": [
         "Id",
         "Name",
         "Roles",
         "RolePath"
      ]
   }
},
"Augment_Opportunity_UserWithRoles": {
   "action": "augment",
   "parameters": {
      "left": "Extract_Opportunity",
      "left_key": [ "OwnerId" ],
      "right": "Augment_User_FlattenUserRole",
      "relationship": "Owner",
      "right_select": [
         "Name",
         "Department",
         "Role.Id",
         "Role.Name",
         "Role.Roles",
         "Role.RolePath"
      ],
      "right_key": [ "Id" ]
   }
},
"Register_OpportunityWithRoles_Dataset": {
   "action": "sfdcRegister",
   "parameters": {
      "alias": "OppRoles",
      "name": "OppRoles",
      "source": "Augment_Opportunity_UserWithRoles",
     "rowLevelSecurityFilter": "'Owner.Role.Roles' == \"$User.UserRoleId\" || 'OwnerId'
== \"$User.Id\""
   }
}
}
```

To flatten the Salesforce role hierarchy, the flatten transformation uses the following input fields from the UserRole object.

Id

Id identifies each node in the Salesforce role hierarchy.

ParentRoleId

ParentRoleId identifies the parent as defined in the role hierarchy.

After traversing through each parent-child relationship in the UserRole object, the flatten transformation generates one record for each role ID. Each record contains all ancestor roles for each role in the hierarchy. The flatten transformation generates two output columns—Roles and RolePath—to store all ancestor roles for each role.

IN THIS SECTION:

flatten Parameters
When you define a flatten transformation, you set the action attribute to `flatten` and specify the parameters.

flatten Parameters

When you define a flatten transformation, you set the action attribute to `flatten` and specify the parameters.

The following table describes the input parameters:

Parameter	Required?	Value
self_field	Yes	Name of the input field that identifies each node in the hierarchy.
parent_field	Yes	Name of the input field that identifies the direct parent of each node in the hierarchy. For example, the Regional Manager 1 role is the parent of the Salesperson 1 role in a role hierarchy.
multi_field	Yes	Name of the multi-value output field that contains a list of all ancestors in the hierarchy, in order from the lowest to the highest level. The flatten transformation creates this field and generates the list of ancestors for each node in the hierarchy. For example, for Salesperson 1 role, the hierarchy of ancestors is: `Sales Manager 1, Regional Manager 1, Vice President 1, CEO`.
path_field	Yes	A string representation of the multi-field field, separated by backslashes. This output field contains the hierarchical path of all ancestors in the hierarchy, in order from the lowest to the highest level. The flatten transformation creates this field and generates the ancestry path for each node in the hierarchy. For example, for a salesperson role in a role hierarchy, the value is: `Sales Manager 1\Regional Manager 1\Vice President 1\CEO`.
source	Yes	Node in the dataflow definition file that contains the hierarchical data that you want to flatten. This node is the input source for this transformation and it must contain the input fields mapped to self_field and parent_field.

Note: By default, the multi_field and path_field fields are created as system fields, which aren't visible in the user interface. To make the fields appear in the user interface, set the `IsSystemField` metadata attribute to *`false`* for each field in the flatten transformation. For more information about metadata attributes, see Overriding Metadata Generated by a Transformation.

SEE ALSO:

flatten Transformation

sfdcDigest Transformation

The sfdcDigest transformation generates a dataset based on data that it extracts from a Salesforce object. You specify the Salesforce object and fields from which to extract data. You might choose to exclude particular fields that contain sensitive information or that aren't relevant for analysis.

When you upload the dataflow definition file, Wave Analytics validates access to Salesforce objects and fields based on the user profile of the user who uploads the file. If the user profile does not have read access to a field or object, the upload fails.

At run time, Wave Analytics runs the dataflow as the Integration User. Wave Analytics validates access to Salesforce objects and fields based on the user profile of the Integration User. For example, if the dataflow tries to extract data from a custom field on which the Integration User does not have read access, the dataflow job fails.

Note: The Integration User is a preconfigured user that is created when Wave Analytics is enabled in your organization. If you or the Integration User need permission on a Salesforce object or field, ask the administrator to grant access.

For more information about preconfigured users in Wave Analytics, see the *Wave Analytics Security Implementation Guide*.

Example: Let's look at an example. You would like to create a dataset that contains all opportunities from the Opportunity object.

You create the following dataflow definition.

```
{
   "Extract_Opportunities": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "Amount" },
            { "name": "StageName" },
            { "name": "CloseDate" },
            { "name": "AccountId" },
            { "name": "OwnerId" },
            { "name": "OpportunitySupportTeamMembers__c" }
         ]
      }
   },
   "Register_Opportunities_Dataset": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "Opportunities",
         "name": "Opportunities",
         "source": "Extract_Opportunities"
      }
   }
}
```

Tips for Using the sfdcDigest Transformation

- Be selective when choosing fields because a dataset can contain a maximum of 5,000 fields.
- To extract data from a Salesforce object, the sfdcDigest transformation runs a SOQL query. The length of the SOQL query cannot exceed 20,000 characters. No results are returned if the SOQL query exceeds the maximum. To reduce the SOQL query length, break up the extract into two or more sfdcDigest transformations and then use theaugment transformation to combine the results. For

example, you might create one sfdcDigest transformation to extract the half of the fields and create another sfdcDigest transformation to extract the remaining fields.

IN THIS SECTION:

Extracting Incremental Changes to Data (Pilot)
You can configure the sfdcDigest transformation to extract only records that changed in a Salesforce object since the last dataflow run. Use incremental extraction to decrease the time required to extract the data.

Filtering Records Extracted from a Salesforce Object
Add a filter to the sfdcDigest transformation to extract a subset of all records from a Salesforce object. You can filter records to reduce the number of extracted and processed records, exclude records that contain irrelevant or sensitive data, and increase dataflow performance.

Overriding Salesforce Field Metadata
You can override the field metadata that the sfdcDigest transformation extracts from a Salesforce object to make the data appear differently in a dataset. For example, Wave Analytics can add a default value to records that have missing values for a field.

Unsupported Salesforce Objects and Fields
The sfdcDigest transformation can't extract data from all Salesforce objects and fields. Consider these limitations before configuring the extraction of Salesforce objects.

sfdcDigest Parameters
When you define an sfdcDigest transformation, you set the action attribute to `sfdcDigest` and specify the parameters for the object and fields that you want to extract. Optionally, you can also specify parameters to filter the records extracted from the Salesforce object.

Extracting Incremental Changes to Data (Pilot)

You can configure the sfdcDigest transformation to extract only records that changed in a Salesforce object since the last dataflow run. Use incremental extraction to decrease the time required to extract the data.

Note: Incremental extraction with the sfdcDigest transformation is currently available through a pilot program. Any unreleased services or features referenced in this or other press releases or public statements are not currently available and may not be delivered on time or at all. Customers who purchase our services should make their purchase decisions based upon features that are currently available.

There are two types of extractions: full and incremental. With a full extraction, the sfdcDigest transformation extracts all records from the Salesforce object. With incremental extraction, the transformation extracts only records that have changed since the last dataflow run. Unless configured otherwise, the sfdcDigest transformation runs full extractions.

When you enable incremental extraction, the sfdcDigest transformation performs different actions on the dataset based the types of record changes in the Salesforce object since the last dataflow run:

Record Change in Salesforce Object	Change to Dataset
Deleted record	Deletes corresponding record in the dataset.
New record	Inserts the new record into the dataset.
Updated record	Updates the corresponding record in the dataset.

Consider the following guidelines when enabling incremental extraction for an sfdcDigest transformation:

- Use incremental extraction for as many instances of the sfdcDigest transformation in the dataflow as possible.
- If necessary, when the sfdcDigest transformation is configured to run an incremental extraction, you can use the `fullRefreshToken` parameter to run a one-time, full extraction. You might use this option if the data in the Salesforce object and dataset become out of sync.
- The sfdcDigest transformation extracts all records from the object if you enable incremental extraction and it's the first time running the dataflow.
- You must use a full extraction in the following cases because incremental extraction is not supported:
  - The Salesforce object contains a formula field that references an external data object.
  - The Salesforce object fields or field types changed since the last dataflow run.
  - The filter in the sfdcDigest transformation changed since the last dataflow run.
  - The Salesforce object is one of the following objects:
    - CallCenter
    - CaseTeamMember
    - CategoryNode
    - CollaborationGroup
    - CollaborationGroupMember
    - CollaborationGroupMemberRequest
    - Division
    - Domain
    - DomainSite
    - Group
    - GroupMember
    - Profile
    - Site
    - Territory
    - Topic
    - User
    - UserRole
    - UserTerritory

Example: Let's consider an example. You'd like to incrementally extract data from the Opportunity object. As a result, you configure the following dataflow definition:

```
{
   "Extract_Opportunities": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "incremental": true,
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "StageName" },
            { "name": "CloseDate" },
```

```
            { "name": "AccountId" },
            { "name": "OwnerId" }
         ]
      }
   },
   "Register_Opportunities_Dataset": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "Opportunities",
         "name": "Opportunities",
         "source": "Extract_Opportunities"
      }
   }
}
```

Later, you realize that one of the opportunity records is missing. You decide to run a one-time full extraction by adding the `fullRefreshToken` parameter as shown in the following dataflow definition.

```
{
   "Extract_Opportunities": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "incremental": true,
         "fullRefreshToken": "bb",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "StageName" },
            { "name": "CloseDate" },
            { "name": "AccountId" },
            { "name": "OwnerId" }
         ]
      }
   },
   "Register_Opportunities_Dataset": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "Opportunities",
         "name": "Opportunities",
         "source": "Extract_Opportunities"
      }
   }
}
```

After the full extraction, the dataflow performs an incremental extraction each time thereafter even though the `fullRefreshToken` parameter is included in the dataflow definition. To run a full extraction again, change the value of the `fullRefreshToken` parameter to a different value.

SEE ALSO:

sfdcDigest Transformation

Filtering Records Extracted from a Salesforce Object

Add a filter to the sfdcDigest transformation to extract a subset of all records from a Salesforce object. You can filter records to reduce the number of extracted and processed records, exclude records that contain irrelevant or sensitive data, and increase dataflow performance.

A filter consists of one or more filter conditions, where each filter condition compares a field value to a value. For example, `Amount >= 1000000`. You can also apply SOQL functions on the field value in a filter condition, like `CALENDAR_YEAR(CreatedDate) = 2011`. You can add multiple filter conditions using logical operators AND, OR, and NOT. You can also use a backslash (\) to escape double quotes included in strings.

The sfdcDigest transformation extracts all records for which the filter is true. If you configured the sfdcDigest transformation for incremental extraction, the filter applies to data extracted during the incremental run only—Wave Analytics doesn't apply the filter to records that were previously loaded into the dataset. If you add an invalid filter, the dataflow fails at run time.

For each instance of sfdcDigest, you can use one of the following types of filters:

- Structured filter
- Advanced filter

Tip: Are you trying to decide whether to use a filter in the sfdcDigest transformation or use a filter transformation? Use a filter transformation to filter records at any point in the dataflow. For example, you can add it after the dataflow joins two datasets. However, to reduce the number of rows processed in the dataflow and optimize dataflow performance, add the filter closest to the point at which records are extracted—when possible, add the filter in the sfdcDigest transformation.

IN THIS SECTION:

Structured Filter in sfdcDigest Transformation

You define a structured filter using JSON syntax.

Advanced Filter in sfdcDigest Transformation

You define an advanced filter using a Salesforce Object Query Language (SOQL) WHERE clause expression. Use an advanced filter only if you are familiar with SOQL.

SEE ALSO:

sfdcDigest Transformation

Structured Filter in sfdcDigest Transformation

You define a structured filter using JSON syntax.

A structured filter uses the following JSON syntax for each filter condition.

```
{
"field": "<field name>",
"operator": "<operator>",
"value": "<value>"|"["<value 1>", "<value 2>"]",
"isQuoted": true|false}
```

The value can be a number, date, string, list of strings, or date literal. Wave Analytics automatically quotes strings unless you set `isQuoted` to true, which indicates that the string is already quoted.

You can use one of the following operators in a filter condition.

Operator	Comment
=	Filter condition is true if the value in the field equals the specified value. String comparisons using the equals operator are case-insensitive. Example: `"filterConditions": [ { "field": "OwnerId", "operator": "=", "value": "a07B00000012HYu" } ]`
!=	Filter condition is true if the value in the field does not equal the specified value. Example (using backslashes to escape double quotes in a string value): `"filterConditions": [ { "field": "Nickname__c", "operator": "!=", "value": "\"Sammy\"" } ]`
>	Filter condition is true if the value in the field is greater than the specified value. Example: `"filterConditions": [ { "field": "Amount", "operator": ">", "value": "100000" } ]`
<	Filter condition is true if the value in the field is less than the specified value. Example (using a date literal): `"filterConditions": [ { "field": "CloseDate", "operator": "<", "value": "THIS_MONTH", "isQuoted": false`

Operator	Comment
	<pre> }]</pre>

>=	Filter condition is true if the value in the field is greater than or equal to the specified value.
	Example:
	<pre>"filterConditions": [
	   {
	   "field": "Amount",
	   "operator": ">=",
	   "value": "100000"
	   }
	]</pre>

<=	Filter condition is true if the value in the field is less than or equal to the specified value.
	Example (using a SOQL function):
	<pre>"filterConditions": [
	   {
	   "field": "CALENDAR_YEAR (CreatedDate)",
	   "operator": "<=",
	   "value": "2015",
	   "isQuoted": true
	   }
	]</pre>

LIKE	Filter condition is true if the value in the field matches the specified value. The LIKE operator is similar to the LIKE operator in SQL; it provides a mechanism for matching partial text strings and supports wildcards.
	• The % and _ wildcards are supported for the LIKE operator.
	• The % wildcard matches zero or more characters.
	• The _ wildcard matches exactly one character.
	• The LIKE operator is supported for string fields only.
	• The LIKE operator performs a case-insensitive match.
	• The LIKE operator supports escaping of special characters % or _. Use a backslash (\) to escape special characters.
	Example:
	<pre>"filterConditions": [
	   {
	   "field": "FirstName",
	   "operator": "LIKE",
	   "value": "Chris%"</pre>

Operator	Comment
	<pre> }]</pre>

IN	Filter condition is true if the value in the field equals any one of the values in the specified list. You can specify a quoted or non-quoted list of values. If the list is quoted, set `isQuoted` to true.
	Example:
	<pre>"filterConditions": [
	   {
	   "field": "StageName",
	   "operator": "IN",
	   "value": ["Closed Won", "Closed Lost"]
	   }
	]</pre>

NOT IN	Filter condition is true if the value in the field does not equal any of the values in the specified list.
	Example:
	<pre>"filterConditions": [
	   {
	   "field": "BillingState",
	   "operator": "NOT IN",
	   "value": ["California", "New York"]
	   }
	]</pre>

INCLUDES	For picklist or multi-select picklist fields only. Filter condition is true if the value in the picklist field includes the specified value.
	Example:
	<pre>"filterConditions": [
	   {
	   "field": "BillingState",
	   "operator": "INCLUDES",
	   "value": ["California"]
	   }
	]</pre>

EXCLUDES	For picklist or multi-select picklist fields only. Filter condition is true if the value in the picklist field excludes the specified value.
	Example:
	<pre>"filterConditions": [
	   {
	   "field": "BillingState",</pre>

Operator	Comment

	<pre>"operator": "EXCLUDES",
	"value": ["California", "New York"]
	}
	]</pre>

Let's look at a few examples of structured filters.

Example: Let's look at an example with a basic structured filter. To perform pipeline analysis on opportunities in fiscal quarter 2 of fiscal year 2015, you create this dataflow definition file to create the relevant dataset.

```
{
   "Extract_Filtered_Opportunities": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "AccountId" },
            { "name": "Amount" },
            { "name": "StageName" },
            { "name": "CloseDate" },
            { "name": "OwnerId" },
            { "name": "FiscalYear" },
            { "name": "FiscalQuarter" },
            { "name": "SystemModstamp" }
         ],
         "filterConditions": [
               {
                  "field": "FiscalYear",
                  "operator": "=",
                  "value": "2015"
               },
               {
                  "field": "FiscalQuarter",
                  "operator": "=",
                  "value": "2"
               }
         ]
      }
   },
   "Register_Opportunities_Dataset": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "Opportunities_2015Q2",
         "name": "Opportunities_2015Q2",
         "source": "Extract_Filtered_Opportunities"
      }
   }
}
```

Note: If you do not specify a logical operator for multiple filter conditions—as is the case in this example—Wave Analytics applies AND between the conditions.

Example: Let's look at an example of a structured filter with a logical operator. To help forecast expected revenue, you create this dataflow to view all opportunities that have either closed or have greater than 90% probability of closing.

```
{
   "Extract_Opportunities": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "AccountId" },
            { "name": "Amount" },
            { "name": "StageName" },
            { "name": "CloseDate" },
            { "name": "OwnerId" },
            { "name": "Probability" },
            { "name": "FiscalYear" },
            { "name": "FiscalQuarter" }
         ],
         "filterConditions": [
            {
               "operator": "OR",
               "conditions": [
                  {
                     "field": "StageName",
                     "operator": "=",
                     "value": "Closed Won"
                  },
                  {
                     "field": "Probability",
                     "operator": ">=",
                     "value": "90"
                  }
               ]
            }
         ]
      }
   },
   "Register_Opportunities_Dataset": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "OpportunitiesExpectedToWin",
         "name": "OpportunitiesExpectedToWin",
         "source": "Extract_Opportunities"
      }
   }
}
```

Example: Finally, let's look at an example of a structured filter with nested logical operators. You create this dataflow to view all opportunities that closed in the current fiscal quarter and are owned by either one of your two direct reports.

```
{
    "Extract_Opportunities": {
        "action": "sfdcDigest",
        "parameters": {
            "object": "Opportunity",
            "fields": [
                { "name": "Id" },
                { "name": "Name" },
                { "name": "AccountId" },
                { "name": "Amount" },
                { "name": "StageName" },
                { "name": "CloseDate" },
                { "name": "OwnerId" },
                { "name": "FiscalYear" },
                { "name": "FiscalQuarter" }
            ],
            "filterConditions": [
                {
                    "operator": "AND",
                    "conditions": [
                        {
                            "field": "CloseDate",
                            "operator": "=",
                            "value": "THIS_FISCAL_QUARTER",
                            "isQuoted": false
                        },
                        {
                            "operator": "OR",
                            "conditions": [
                                {
                                    "field": "OwnerId",
                                    "operator": "=",
                                    "value": "00540000000HfUz"
                                },
                                {
                                    "field": "OwnerId",
                                    "operator": "=",
                                    "value": "00540000000HfV4"
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    },
    "Register_Opportunities_Dataset": {
        "action": "sfdcRegister",
        "parameters": {
            "alias": "DirectReport_Opportunities",
            "name": "DirectReport_Opportunities",
```

```
            "source": "Extract_Opportunities"
        }
    }
}
```

Advanced Filter in sfdcDigest Transformation

You define an advanced filter using a Salesforce Object Query Language (SOQL) WHERE clause expression. Use an advanced filter only if you are familiar with SOQL.

Example: Let's look at an example of an advanced filter. You want to extract only opportunity records that are owned by a specific user and that have either high value or a high probability of closing. You create this dataflow.

```
{
   "Extract_Filtered_Opportunities": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "AccountId" },
            { "name": "Amount" },
            { "name": "StageName" },
            { "name": "CloseDate" },
            { "name": "Probability" },
            { "name": "OwnerId" }
         ],
         "complexFilterConditions": "OwnerId = '00540000000HfUz' AND (Amount > 100000
 OR Probability > 75)"
        }
   },
   "Register_Opportunities_Dataset": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "FilteredOpportunities",
         "name": "FilteredOpportunities",
         "source": "Extract_Filtered_Opportunities"
      }
   }
}
```

Overriding Salesforce Field Metadata

You can override the field metadata that the sfdcDigest transformation extracts from a Salesforce object to make the data appear differently in a dataset. For example, Wave Analytics can add a default value to records that have missing values for a field.

You can add the following field parameters to the sfdcDigest transformation to override the field metadata:

- defaultValue
- type
- fiscalMonthOffset

- isYearEndFiscalYear
- firstDayOfWeek
- isMultiValue
- multiValueSeparator
- precision
- scale

For a description of each of these field parameters, see Field Parameters. For information about using metadata attributes to configure dates, see Date Handling in Datasets.

Example: Let's look at an example. You would like to override metadata extracted from the Opportunity object.

You add the bold text in the following dataflow definition to override field metadata from the Opportunity object.

```
{
   "Extract_Opportunities": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            {
               "name": "Amount",
               "defaultValue":0
               "precision":18
               "scale":2
            },
            { "name": "StageName" },
            {
               "name": "CloseDate",
               "fiscalMonthOffset":9,
               "firstDayOfWeek":"2",
               "isYearEndFiscalYear":true
            },
            { "name":"AccountId" },
            { "name":"OwnerId" },
            {
               "name": "OpportunitySupportTeamMembers__c",
               "type":"Text",
               "isMultiValue":true,
               "multiValueSeparator":","
               "precision":255
            }
         ]
      }
   },
   "Register_Opportunities_Dataset":{
      "action":"sfdcRegister",
      "parameters":{
         "alias":"Opportunities",
         "name":"Opportunities",
         "source":"Extract_Opportunities"
      }
```

```
    }
}
```

SEE ALSO:

sfdcDigest Transformation

Unsupported Salesforce Objects and Fields

The sfdcDigest transformation can't extract data from all Salesforce objects and fields. Consider these limitations before configuring the extraction of Salesforce objects.

For information about all Salesforce objects and fields, see the Object Reference for Salesforce and Force.com.

Unsupported Objects

The sfdcDigest transformation can't extract data from these Salesforce objects.

- AuthProvider
- BrandTemplate
- ChatterConversation
- ChatterConversationMember
- ChatterMessage
- ConnectedApplication
- ContentFolderLink
- ContentWorkspace
- ContentWorkspaceDoc
- CorsWhitelistEntry
- CustomNotDeployed__OwnerSharingRule
- EmailDomainKey
- EmailServicesAddress
- EmailServicesFunction
- EmailStreamMessage
- EmailStreamShareOption
- EmailStreamShareOptionHistory
- EmailStreamShareOptionOwnerSharingRule
- EmailStreamShareOptionShare
- EmailTemplate
- EventType
- EventTypeParameter
- EnvironmentHub
- EnvironmentHubInvitation
- EnvironmentHubMemberRel
- FeedPollChoice

- FeedPollVote
- KnowledgeArticleVersion
- LoginGeo
- LoginHistory
- NetworkActivityAudit
- NewsArticleFeed
- NetworkModeration
- OrganizationProperty
- OrgWideEmailAddress
- OutboundField
- PackageLicense
- PackageMemberOwnerSharingRule
- PartnerNetworkSyncLog
- PermissionSet
- PermissionSetLicense
- Profile
- ReputationLevel
- ReputationLevelLocalization
- ReputationPointsRule
- SearchPromotionRule
- SelfServiceUser
- SessionPermSetAssignment
- SetupAssistantAnswer
- SetupAssistantProgress
- SsoUserMapping
- TenantSecret
- Territory2ModelHistory
- TwoFactorInfo
- UserLogin
- UserPackageLicense
- UserProvAccount
- UserProvAccountStaging
- UserProvisioningConfig
- UserProvisioningLog
- UserProvisioningRequest
- UserProvisioningRequestOwnerSharingRule
- UserProvisioningRequestShare
- UserProvMockTarget
- UserRecordAccess
- VerificationHistory

- WebLink
- WebLinkLocalization

The sfdcDigest transformation cannot extract data from external objects created in Salesforce. External objects are similar to custom objects, except that they map to data located outside Salesforce.

If you include an unsupported or inaccessible object in the sfdcDigest transformation, the dataflow fails at run time with an error message.

Unsupported Fields

The sfdcDigest transformation cannot extract data from these fields:

Object	Unsupported Fields
Account	CleanStatus
ActionPlanItem	ItemId
AuthSession	• LoginGeoId • LoginHistoryId
CaseArticle	KnowledgeArticleId
Contact	• CanAllowPortalSelfReg • CleanStatus
ContentDocument	ParentId
ContentFolderLink	ParentEntityId
CustomPerson__p	Title
DocumentAttachmentMap	ParentId
EmailMessage	ActivityId
EmailRoutingAddress	EmailServicesAddressId
EnvironmentHubMember	EnvironmentHubId
ExternalEventMapping	EventId
InstalledMobileApp	ConnectedApplicationId
Lead	CleanStatus
KnowledgeArticle	MasterLanguage
KnowledgeArticleVersion	• IsOutOfDate • TranslationCompletedDate • TranslationExportedDate • TranslationImportedDate
Network	• CaseCommentEmailTemplateId

Object	Unsupported Fields
	• ChangePasswordEmailTemplateId • ForgotPasswordEmailTemplateId • WelcomeEmailTemplateId
Organization	• SelfServiceEmailUserOnCaseCreationTemplateId • SelfServiceNewCommentTemplateId • SelfServiceNewPassTemplateId • SelfServiceNewUserTemplateId • WebToCaseAssignedEmailTemplateId • WebToCaseCreatedEmailTemplateId • WebToCaseEmailTemplateId • WebToLeadEmailTemplateId
PermissionSet	• PermissionsEditEvent • PermissionsEditTask
PermissionSetLicense	• MaximumPermissionsEditEvent • MaximumPermissionsEditTask
Profile	• PermissionsEditEvent • PermissionsEditTask
ThirdPartyAccountLink	SsoProviderId
User	• LastPasswordChangeDate • UserPreferencesEnableVoicePilot
WorkBadge	RewardId
WorkBadgeDefinition	RewardFundId

If you include a field with an unsupported field in the sfdcDigest transformation, the dataflow ignores the field.

In addition, Salesforce recommends that you do not extract data from the MayEdit field of the Account object. Extracting data from this field significantly decreases performance and can cause the dataflow to fail.

Unsupported Field Types

The sfdcDigest transformation can't extract data from fields with these field types.

- base64
- composite (like address and location)
- data category group reference
- encrypted string

If you include a field with an unsupported field type in the sfdcDigest transformation, the dataflow ignores the field.

SEE ALSO:

sfdcDigest Transformation

sfdcDigest Parameters

When you define an sfdcDigest transformation, you set the action attribute to `sfdcDigest` and specify the parameters for the object and fields that you want to extract. Optionally, you can also specify parameters to filter the records extracted from the Salesforce object.

You can specify parameters in the following sections of the sfdcDigest node: parameters, fields, and filterConditions.

Parameters

The following table describes the parameters in the `parameters` section.

Parameter	Required?	Value
object	Yes	API name of the Salesforce object from which you want to extract data. This object is the input source for this transformation. The sfdcDigest transformation doesn't support extraction from all Salesforce objects.
incremental	No	Performs an incremental extraction, which extracts only changes to the Salesforce object since the last dataflow run. Valid values: true or false.
fullRefreshToken	No	Performs a one-time full extraction to synchronize the data in the dataset with data in the Salesforce object. Specify any value for this parameter. After the full extraction, the dataflow performs an incremental extraction each time thereafter even though the `fullRefreshToken` parameter is included in the dataflow definition. To run a full extraction again, change the value of the `fullRefreshToken` parameter to a different value.
fields	Yes	An array of names of all fields from which you want to extract data from the specified Salesforce object. The sfdcDigest transformation doesn't support extraction from all field types.

Parameter	Required?	Value
		See Field Attributes.
filterConditions	No	A filter that restricts the records extracted from the specified Salesforce object. The sfdcDigest transformation extracts all records from the Salesforce object for which the filter is true. You can specify a structured or advanced filter. See Filter Conditions Parameters.
complexFilterConditions	No	For advanced filters only. A SOQL WHERE clause used to filter records extracted from the specified Salesforce object.

Field Attributes

The following table describes the attributes in the `fields` section. It also describes optional attributes that you can provide to override the field metadata. You can override the metadata that the sfdcDigest transformation extracts from a Salesforce object to make the data appear differently in a dataset. For example, Wave Analytics can add a default value to records that have missing values for a field. If you don't override the values, Wave Analytics gets the values from Salesforce.

Attribute	Required?	Value
name	Yes	API name of the field in the Salesforce object that you want to include in the dataset. You can specify multiple fields.
defaultValue	No	For text and numeric fields that can be null. Default value that replaces a null value for the specified field.
type	No	Wave Analytics field type associated with the specified field. Valid types are Text, Numeric, or Date. Any value, including numeric values, can be Text. For example, by default, fiscal quarter from Salesforce objects is Number. However, you can change it to Text. Specify a type to override the type determined by Wave Analytics. Example: `"type": "Text"`
fiscalMonthOffset	No	For date fields only. The difference, in months, between the first month of the fiscal year and the first month of the

Attribute	Required?	Value
		calendar year (January). For example, if the fiscal year starts in January, the offset is 0. If the fiscal year starts in October, the offset is 9. Example: `"fiscalMonthOffset": 9` Note: This attribute also controls whether Wave Analytics generates fiscal date fields. To generate fiscal date fields, set `fiscalMonthOffset` to a value other than '0'. For more information, see Date Handling in Datasets.
isYearEndFiscalYear	No	For date fields only. Indicates whether the fiscal year is the year in which the fiscal year ends or begins. Because the fiscal year can start in one calendar year and end in another, you must specify which year to use for the fiscal year. • If true, then the fiscal year is the year in which the fiscal year ends. The default is true. • If false, then the fiscal year is the year in which the fiscal year begins. Example: `"isYearEndFiscalYear": true` This field is relevant only when `fiscalMonthOffset` is greater than 0. For more information, see Date Handling in Datasets.
firstDayOfWeek	No	For date fields only. The first day of the week for the calendar year and, if applicable, fiscal year. Use 0 to set the first day to be Sunday, 1 to set the first day to be Monday, and so on. Use -1 to set the first day to be on January 1. The default is -1. Example: `"firstDayOfWeek": "0"`

Attribute	Required?	Value
		For more information, see Date Handling in Datasets.
isMultiValue	No	For text fields only. Indicates whether the specified field has multiple values. Example: `"isMultiValue": false`
multiValueSeparator	No	For text fields only. Character used to separate multiple values in the specified field when isMultiValue equals true. This value defaults to a semicolon (;) if you do not specify a value and isMultiValue equals true. Set to null when isMultiValue equals false. Example: `"multiValueSeparator": ";"`
precision	No	The maximum number of digits in a numeric value, or the length of a text value. For numeric values: Includes all numbers to the left and to the right of the decimal point (but excludes the decimal point character). Value must be between 1 and 18, inclusive. For text values: Value defaults to 255 characters, and must be between 1 and 32,000 characters, inclusive. Example: `"precision": 10`
scale	No	The number of digits to the right of the decimal point in a numeric value. Must be less than the precision value. Value must be between 0 and 17 characters, inclusive. Example: `"scale": 2`

Filter Conditions Parameters

The following table describes the structured filter parameters in the `filterConditions` section. These parameters do not apply to advanced filters.

Parameter	Required?	Value
field	No	The field in the Salesforce object on which you want to apply a filter condition. Each filter condition in a structured filter uses the following syntax: `{ "field": "<field name>", "operator": "<operator>", "value": "<value>", "isQuoted": true|false}`
operator	No	The purpose depends on the context. • `operator` can be used as a comparison operator–like =, <, and IN–that compares the field value against a constant value. • `operator` can also be used as a logical operator (AND, OR, or NOT) that links multiple filter conditions together. In the example below, the bold `operator` is the logical operator. The other instances of `operator` are comparison operators. `"filterConditions": [ { **"operator": "OR",** "conditions": [ { "field": "StageName", "operator": "=", "value": "Closed Won" }, { "field": "Probability", "operator": ">=", "value": "90" } ] } ]`
value	No	The value used in a filter condition.

Parameter	Required?	Value

isQuoted	No	Indicates whether you quoted the string value in a filter condition.
		Example with quoted values:
		`"filterConditions": [ { "field": "StageName", "operator": "IN", "value": "('Closed Won', 'Closed Lost')", "isQuoted": true } ]`
		Example with non-quoted values:
		`"filterConditions": [ { "field": "StageName", "operator": "IN", "value": ["Closed Won", "Closed Lost"], "isQuoted": false } ]`
		If you don't include isQuoted for a filter on a string value, Wave Analytics assumes that the string value is not quoted and adds the quotes for you.

conditions	No	Use to specify a logical operator to link multiple filter conditions together.

SEE ALSO:

- sfdcDigest Transformation
- Filtering Records Extracted from a Salesforce Object
- Extracting Incremental Changes to Data (Pilot)

sfdcRegister Transformation

The sfdcRegister transformation registers a dataset to make it available for queries. Users cannot view or run queries against unregistered datasets.

You don't need to register all datasets. For example, you don't need to register an intermediate dataset that is used to build another dataset and does not need to be queried. In addition, you don't need to register datasets that are created when you upload external data because Wave Analytics automatically registers these datasets for you.

Carefully choose which datasets to register because:

- The total number of rows in all registered datasets cannot exceed 250 million per platform license.
- Users that have access to registered datasets can query their data. Although, you can apply row-level security on a dataset to restrict access to records.

Example: Let's look at an example. You create a dataflow that extracts opportunities from the Opportunity object. To register the dataset, name it "Opportunities," and apply row-level security on it, you add the sfdcRegister transformation as shown in the following dataflow definition file.

```
{
   "Extract_Opportunities": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "Amount" },
            { "name": "StageName" },
            { "name": "CloseDate" },
            { "name": "AccountId" },
            { "name": "OwnerId" }
         ]
      }
   },
   "Register_Oppportunities_Dataset": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "Opportunities",
         "name": "Opportunities",
         "source": "Extract_Opportunities",
         "rowLevelSecurityFilter": "'OwnerId' == \"$User.Id\""
      }
   }
}
```

IN THIS SECTION:

sfdcRegister Parameters
When you define an sfdcRegister transformation, you set the action attribute to `sfdcRegister` and specify the parameters.

sfdcRegister Parameters

When you define an sfdcRegister transformation, you set the action attribute to `sfdcRegister` and specify the parameters.

The following table describes the input parameters:

Parameter	Required?	Value
alias	Yes	API name of the registered dataset. This name can contain only underscores and alphanumeric characters, and must be unique among other dataset aliases in your organization. It must begin with a letter, not

Parameter	Required?	Value
		include spaces, not end with an underscore, and not contain two consecutive underscores. It also cannot exceed 80 characters.
name	Yes	Display name of the registered dataset. The name cannot exceed 80 characters. Note: To change the name after you create the dataset, you must edit the dataset.
source	Yes	Node in the dataflow definition file that identifies the dataset that you want to register. This is the input source for this transformation.
rowLevelSecurityFilter	No	The predicate used to apply row-level security on the dataset when the dataset is first created. Example: "rowLevelSecurityFilter": "'OwnerId' == "$User.Id"" Note: To change the predicate after you create the dataset, you must edit the dataset. When entering the predicate in the Register transformation of the dataflow JSON, you must escape the double quotes around string values. After the dataset is created, Wave Analytics ignores it's security predicate setting in the dataflow. To change the security predicate for an existing dataset, edit the dataset in the user interface.

SEE ALSO:

sfdcRegister Transformation

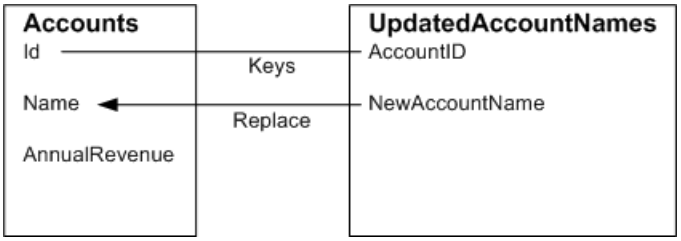
update Transformation

The update transformation updates the specified field values in an existing dataset based on data from another dataset, which we'll call the lookup dataset. The transformation looks up the new values from corresponding fields in the lookup dataset. The transformation stores the results in a new dataset.

When you create the transformation, you specify the keys that are used to match records between the two datasets. To dictate which field in the lookup dataset updates the field in the source dataset, you also map the corresponding fields from both datasets.

Example: Let's look at an example. You have an existing Accounts dataset that contains account information—Id, Name, and AnnualRevenue. Unfortunately, some of the account names in the dataset are now incorrect because of a series of mergers and acquisitions. To quickly update the account names in the dataset, you perform the following tasks.

1. Create a .csv file that contains the new account names and associated account IDs for accounts that have name changes.
2. Upload the .csv file to create a dataset called UpdatedAccountNames.
3. Create a dataflow definition file to update account names in the Accounts dataset by looking up the new account names in the UpdatedAccountNames dataset.



You create the following dataflow definition file.

```
{
   "Extract_AccountDetails": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Account",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "AnnualRevenue" }
         ]
      }
   },
   "Extract_UpdatedAccountNames": {
      "action": "edgemart",
      "parameters": { "alias": "UpdatedAccountNames" }
   },
   "Update_AccountRecords": {
      "action": "update",
      "parameters": {
         "left": "Extract_AccountDetails",
         "right": "Extract_UpdatedAccountNames",
         "left_key": [ "Id" ],
         "right_key": [ "AccountID" ],
         "update_columns": { "Name": "NewAccountName" }
      }
   },
   "Register_UpdatedAccountRecords": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "Accounts",
```

```
            "name": "Accounts",
            "source": "Update_AccountRecords"
        }
    }
}
```

Example: Let's look at another example, where a composite key is used to match records between both datasets. In this case, you match records using the account ID and account name fields.

You create the following dataflow definition file.

```
{
   "Extract_AccountDetails": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Account",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "AnnualRevenue" }
         ]
      }
   },
   "Extract_UpdatedAccountNames": {
      "action": "edgemart",
      "parameters": { "alias": "UpdatedAccountNames" }
   },
   "Update_AccountRecords": {
      "action": "update",
      "parameters": {
         "left": "Extract_AccountDetails",
         "right": "Extract_UpdatedAccountNames",
         "left_key": ["Id","Name"],
         "right_key": ["AccountId","NewAccountName"],
         "update_columns": {
            "Name": "NewAccountName",
            "CreatedDate":"NewCreatedDate",
            "AnnualRevenue":"NewAnnualRevenue"
      }
   },
   "Register_UpdatedAccountRecords": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "Accounts",
         "name": "Accounts",
         "source": "Update_AccountRecords"
      }
   }
}
```

IN THIS SECTION:

update Parameters

When you define an update transformation, you set the action attribute to `update` and specify the parameters.

update Parameters

When you define an update transformation, you set the action attribute to `update` and specify the parameters.

The following table describes the input parameters.

Parameter	Required?	Value
left	Yes	Node in the dataflow definition file that identifies the dataset that contains the records that you want to update.
right	Yes	Node in the dataflow definition file that identifies the lookup dataset that contains the new values.
left_key	Yes	Key column in the left dataset used to match records in the other dataset. If you use a composite key, the left and right keys must have the same number of columns in the same order. For an example, see update Transformation on page 72.
right_key	Yes	Key column in the right dataset used to match records in the other dataset. If you use a composite key, the left and right keys must have the same number of columns in the same order.
update_columns	No	An array of corresponding columns between the left and right datasets. Use the following syntax: `"update_columns": { "LeftColumn1": "RightColumn1", "LeftColumn2": "RightColumn2",... "LeftColumnN": "RightColumnN" }`. The value from right column replaces the value from the corresponding left column. The field types of the left and right column must match. Note: If you specify a column name that does not exist, the dataflow fails. If you do not specify this parameter, the transformation updates the left dataset by matching all columns in the right dataset with those in the left. In this case, the right column names must match exactly with the

Parameter	Required?	Value
		left column names. Otherwise, an error might occur.

SEE ALSO:

update Transformation

Overriding Metadata Generated by a Transformation

Optionally, you can override the metadata that is generated by a transformation. You can override object and field attributes. For example, you can change a field name that is extracted from a Salesforce object so that it appears differently in the dataset. To override the metadata, add the overrides to the Schema section of the transformation in the dataflow definition file.

In the Schema section, you can override the metadata attributes for one object only.

The Schema section in this sample sfdcDigest transformation contains metadata overrides:

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available for an additional cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

```
"Extract_Opportunities": {
   "action": "sfdcDigest",
   "parameters": {
      "object": "Opportunity",
      "fields": [
         { "name": "Name" },
         { "name": "Amount" }
      ]
   },
   "schema": {
      "objects": [
         {
            "label":"Sales Opportunities",
            "fields": [
               {
                  "name": "Amount",
                  "label": "Opportunity Amount"
               }
            ]
         }
      ]
   }
}
```

Object Attributes

You can override the following object attributes.

Object Attribute	Type	Description
`label`	String	The display name for the object. Can be up to 40 characters. Example: `"label": "Sales Data"`
`description`	String	The description of the object. Must be less than 1,000 characters. Add a description to annotate an object in the dataflow definition file. This description is not visible to users in the Wave Analytics user interface. Example: `"description": "The SalesData object tracks basic sales data."`
`fields`	Array	The array of fields for this object.

Field Attributes

You can override attributes of each specified dataset field.

Field Attribute	Type	Description

`name`	String	Name of the field in the dataset. Identifies the field that you want to override.
		Examples:
		`"name": "Amount"`
		`"name": "Role.Name"`

`label`	String	The display name for the field. Can be up to 255 characters. Example: `"label": "Opportunity Amount"`
`description`	String	The description of the field. Must be less than 1,000 characters. Add a description to annotate a field in the dataflow definition file. This description is not visible to users in the Wave Analytics user interface. Example: `"description": "The Amount field contains the opportunity amount."`
`isSystemField`	Boolean	Indicates whether this field is a system field to be excluded from query results. Example: `"isSystemField": false`
`format`	String	The display format of the numeric value. Examples:

Field Attribute	Type	Description
		`"format": "$#,##0.00" (Numeric)` For more information about valid formats, see Numeric Formats.

Numeric Formats

An example of a typical numeric value is $1,000,000.99, which is represented as $#,##0.00. You are required to specify the precision and scale of the number. The format is specified by using the following symbols:

Symbol	Meaning
0	One digit
#	Zero or one digit
.	This symbol is the default decimal separator. Use the `decimalSeparator` field to set the decimal separator to a different symbol.
-	Minus sign
,	Grouping separator
$	Currency sign

Note: The format for numeric values isn't used in data ingestion. It is used only to specify how numeric values are formatted when displayed in the UI. Also, you can't override date formats.

Example: Let's consider an example where you want to override the following object and field attributes that the sfdcDigest transformation extracts from the Opportunity object.

Object/Field	Attribute Changes
Opportunity object	• Change the object label to "Sales Opportunities" • Add an object description
Id field	• Change the field label to "Opportunity Id" • Hide the field from queries
Amount field	• Change the field label to "Opportunity Amount" • Change the format to $#,##0.00
CloseDate field	• Change the field label to "Closing Date"

To override the attributes, you add the Schema section with the override values to sfdcDigest in the dataflow definition file.

```
{
"Extract_Opportunities": {
   "action": "sfdcDigest",
   "parameters": {
      "object": "Opportunity",
      "fields": [
         { "name": "Id" },
         { "name": "Name" },
         { "name": "Amount" },
         { "name": "StageName" },
         { "name": "CloseDate" },
         { "name": "AccountId" },
         { "name": "OwnerId" }
      ]
   },
   "schema": {
      "objects": [
         {
            "label":"Sales Opportunities",
            "description": "These are all sales opportunities.",
            "fields": [
               {
                  "name": "Id",
                  "label": "Opportunity Id",
                  "isSystemField": true
               },
               {
                  "name": "Amount",
                  "label": "Opportunity Amount",
                  "format": "$#,##0.00"
               },
               {
                  "name": "CloseDate",
                  "label": "Closing Date"
               }
            ]
         }
      ]
   }
},
 "Register_Dataset_Opportunities": {
   "action": "sfdcRegister",
   "parameters": {
      "source": "Extract_Opportunities",
      "alias": "Opportunities",
      "name": "Opportunities"
      }
   }
}
```

CREATE A DATASET WITH THE DATASET BUILDER

You can use the dataset builder to create a single dataset based on data from one or more Salesforce objects. The dataset builder generates and appends the associated JSON to the dataflow definition file. The dataset is created the next time the dataflow runs. The data in the dataset refreshes each time the dataflow runs. You can also edit the dataflow definition file to add transformations that manipulate the dataset.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

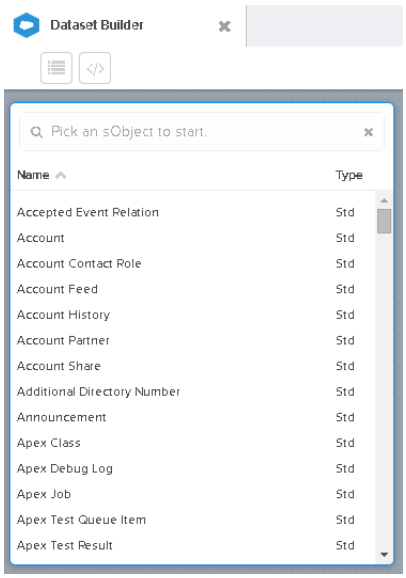
Available for an additional cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To access the dataset builder:

- "Edit Wave Analytics Dataflows"

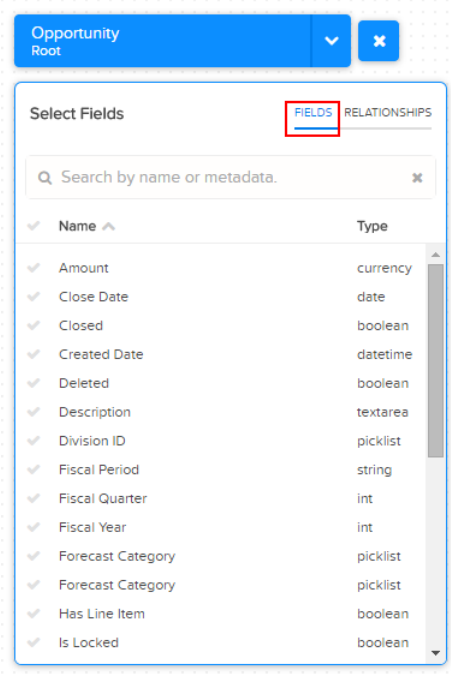
1. On the home page or on an app page, click **Create** > **Dataset**.
2. Click **Salesforce**.
   The dataset builder opens on the Dataset Builder tab.



3. Select the root object.

   The root object is the lowest level child object that you can add to the canvas. After you select the root object, you can add only parent objects of the root object—you can't add it's children objects. To change the root object, refresh the page and start over.

4. Hover over the root object, and then click .
   The Select Fields dialog box appears. By default, the Fields tab appears and shows all available object fields from which you can extract data.

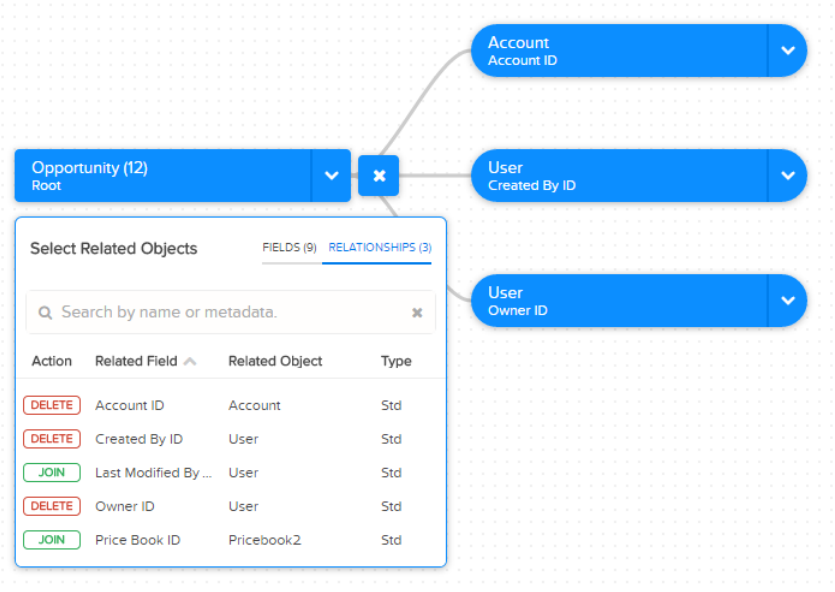


Note: You can view this dialog box for any object included in the canvas.

5. In the Fields tab, select the fields from which you want to extract data.
   To locate fields more quickly, you can search for them or sort them by name or type.

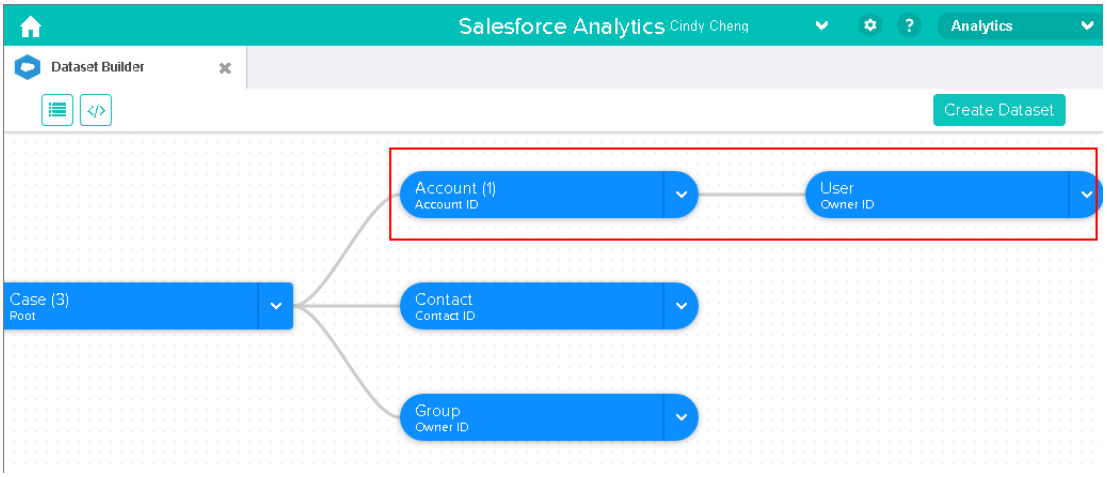
   Important: You must select at least one field for each object that you add to the canvas. If you add an object and don't add any of it's fields, the dataflow fails at run time.

6. In the Relationships tab, click **Join** to add the related objects to the canvas.
   When you add a related object, the related object appears in the canvas.



7. To remove a related object, click **Delete**.

   Warning: When you delete a related object, you also delete all objects that descend from the related object in the diagram. For example, if you delete Account shown below, you delete the branch that contains Account and User.

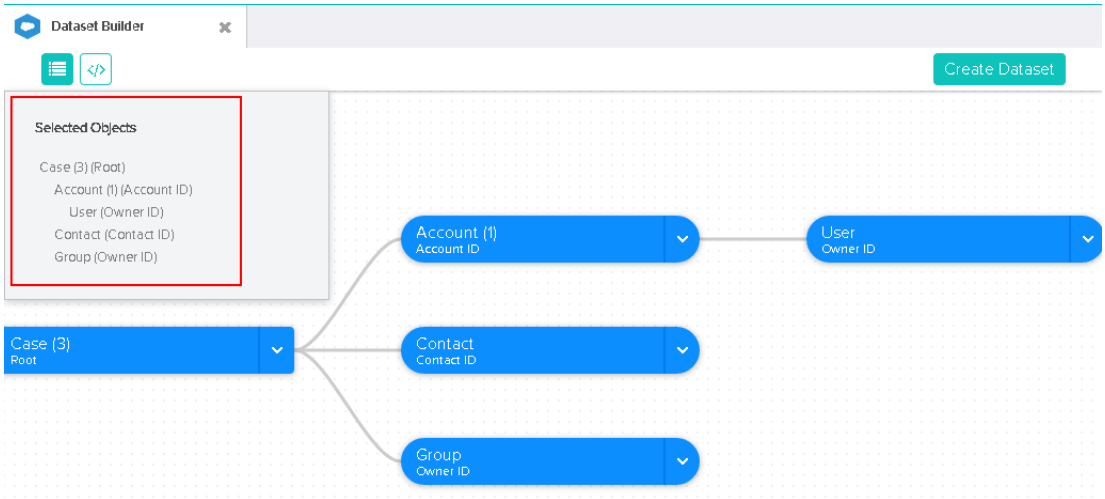


8. For each related object, select the fields from which you want to extract data.
9. To move the entire diagram, select a white space in the canvas and drag it.

   You might need to move the diagram to view a different section of the diagram.

10. To view all objects included in the canvas, click .

    The Selected Objects dialog box shows a tree structure of all objects included in the canvas. The root object appears at the top of the tree.



    If you select one of the objects, the dataset builder focuses on the object by placing the object in the center of the canvas.

11. To view the associated JSON, click .

    When you create the dataset, the dataset builder appends the JSON to the dataflow definition file.

12. Click **Create Dataset**.
13. Enter the name of the dataset, and select the app that will contain the dataset if it's not already selected.

    Note: If you enter a dataset name that is already used, when you create the dataset, the dataset builder appends a number to the dataset name. For example, if you entered MyOpportunities, the dataset builder creates MyOpportunities1. The dataset name cannot exceed 80 characters.

14. Click **Create**.
    The dataset builder appends the underlying JSON to the dataflow definition file. The dataset is created the next time the dataflow runs. You can manually run the dataflow to immediately create the dataset.

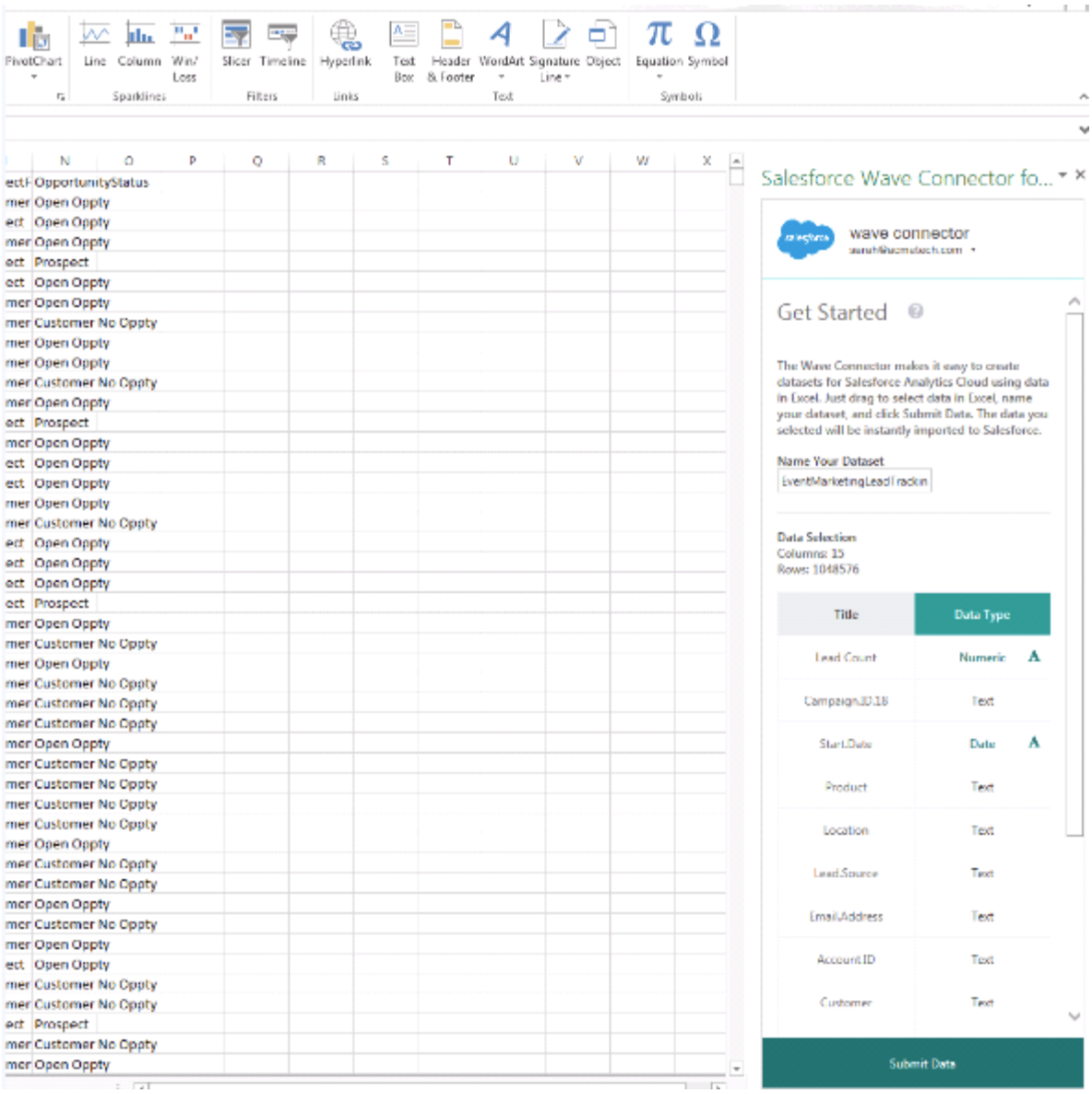
INSTALL THE WAVE CONNECTOR EXCEL APP

The Wave Connector app gives users a fast, easy way to import data from Excel 2013 into Salesforce Wave Analytics..

USER PERMISSIONS

To import data from Excel 2013 to Wave Analytics :

- Upload External Data to Wave Analytics



Install the Wave Connector Excel App

If you use Excel 2013 on the desktop or Office 365, the Office Online version of Excel, the Wave Connector gives you a great way to get your data into Salesforce Wave Analytics. After installing the Connector, you just select data from Excel, click Submit, and the Connector does the work for you, importing the data to Wave Analytics and creating a dataset.

Here's how to install the Connector:

1. Open Excel, either on your desktop or in Office Online.
2. Open the Insert tab.
3. Click Apps for Office.
4. Search for the Wave Connector, and click to install it.
5. Enter your Salesforce credentials to open the Connector.

Once you've installed the Connector, follow the instructions in the Connector window to create datasets based on Excel data. Opening the Connector automatically logs you in to Salesforce Wave Analytics. Click the Connector Help icon for complete information about using the app.

CREATE A DATASET WITH EXTERNAL DATA

Create a Dataset with External Data

You can either upload external data through the user interface or through the External Data API to create a dataset. When you upload an external data file (in .csv, .gz, or .zip format), you can also provide a metadata file. A metadata file contains metadata attributes that describe the structure of the data in the external data file. If you upload a .csv from the user interface, Wave Analytics automatically generates the metadata file, which you can preview and change. If you do not provide a metadata file, Wave Analytics imports all external data file columns as dimensions.

Tip: Wave temporarily stores the uploaded CSV and metadata files for processing only. After the datasets are created, Wave purges the files. If you want to use the files again later, keep a copy.

Before uploading external data files, review the format requirements and examples of the .csv and metadata files in the External Data Format Reference.

Note: You can also use the the External Data API to upload external data files. Use the API to take advantage of additional features, like performing incremental extracts and performing append, delete, and upsert operations. For more information about the External Data API, see the External Data API Developer's Guide.

1. On the home or app page, click **Create** > **Dataset**.

2. Type the name of your dataset in the Dataset Name field.
   The name cannot exceed 80 characters.
3. If you want to create the dataset in a different app, change the app in the App drop-down list.
4. Click **CSV**.
5. Add the .csv file.
   After you add the .csv file, Wave Analytics automatically generates and adds the corresponding metadata file.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

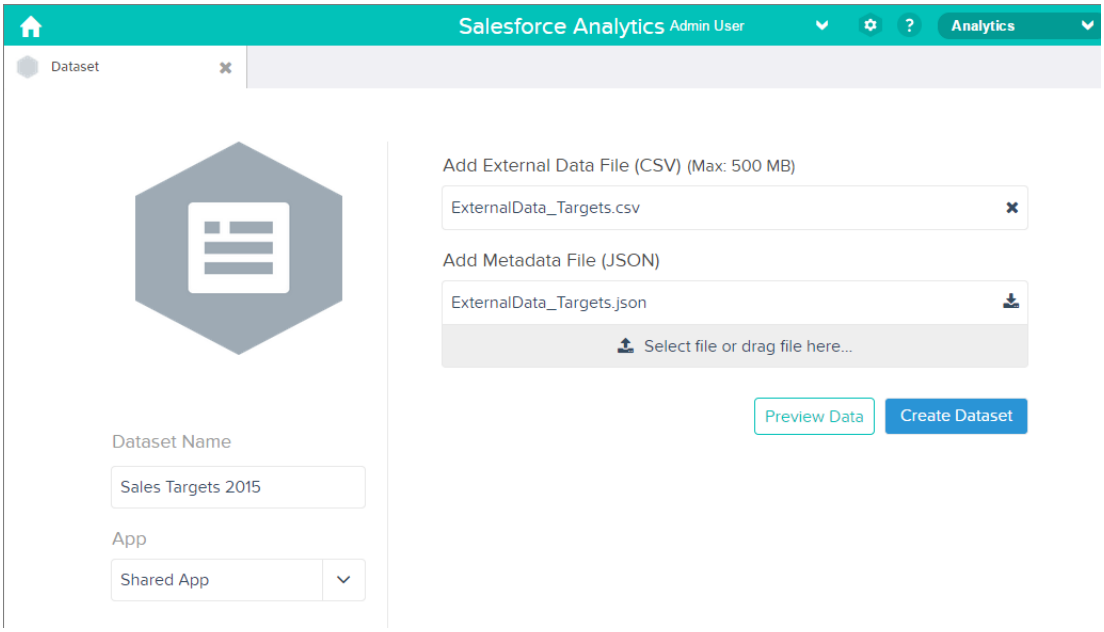
Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

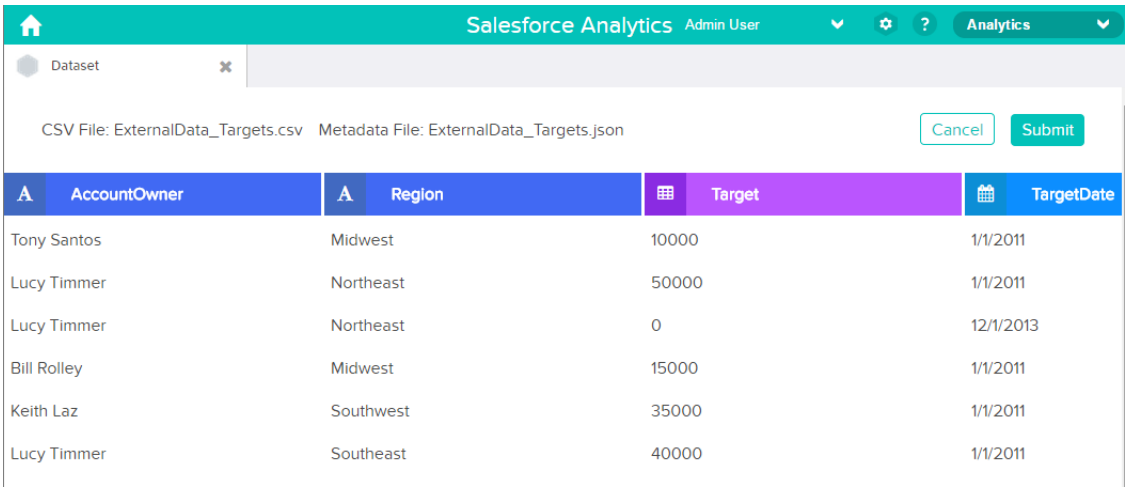
To upload external data:
- "Upload External Data to Wave Analytics"

Create a Dataset with External Data



Note: Instead of using the generated metadata file, if you want, you can upload a different metadata file that you created from scratch. If you upload your own metadata file, the **Preview Data** button is disabled.

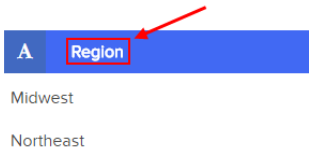
6. Perform the following tasks to change the metadata attributes in the generated metadata file.
   a. Click **Preview Data** to view and change the required metadata attributes.



AccountOwner	Region	Target	TargetDate
Tony Santos	Midwest	10000	1/1/2011
Lucy Timmer	Northeast	50000	1/1/2011
Lucy Timmer	Northeast	0	12/1/2013
Bill Rolley	Midwest	15000	1/1/2011
Keith Laz	Southwest	35000	1/1/2011
Lucy Timmer	Southeast	40000	1/1/2011

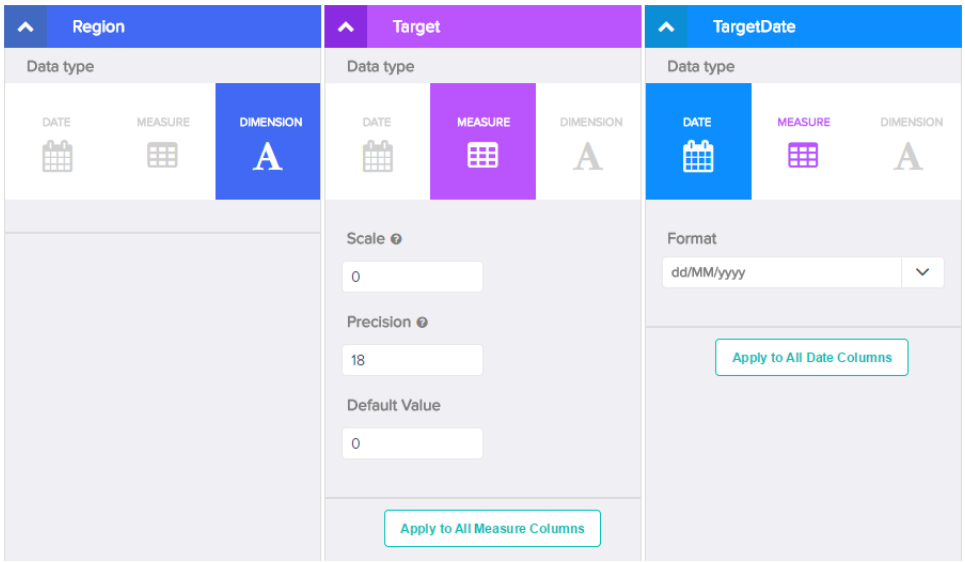
   b. Click a column name to change it. The column name is the display name in the dataset. The column name cannot exceed 40 characters.

Create a Dataset with External Data



c. Click the column header to change other attributes for the column.

You can change the attributes for measure and date columns only.



d. To apply the changes to all other columns of the same data type, click **Apply to All <data type> Columns**.

e. Click **Submit** to save the metadata changes to the metadata file.

Note: If there are errors, the **Submit** button is grayed out.

f. Click **OK** to close the confirmation message.

g. To change optional metadata attributes, click to download the metadata file, edit the file, and then upload it.

7. Click **Create Dataset**.
Your data files are scheduled for upload. It might take some time to process the data upload job; you can monitor its status in the data monitor. If upload is successful, the new dataset is available from the home or app page.

8. Click **Continue** to dismiss the confirmation message.

IN THIS SECTION:

Rules for Automatic Generation of a Metadata File
When you upload a CSV file from the user interface, Wave Analytics automatically generates the metadata file as long as the CSV file meets certain requirements.

Rules for Automatic Generation of a Metadata File

When you upload a CSV file from the user interface, Wave Analytics automatically generates the metadata file as long as the CSV file meets certain requirements.

To enable Wave Analytics to generate the metadata file, a CSV file must meet the following requirements.

- The file type must be .csv, not .gz or .zip.
- The file must contain one row for the column header and at least one record.
- The CSV file must meet all Wave Analytics requirements as mentioned in the External Data Format Reference.

Wave Analytics generates the metadata attributes for each CSV column based on the first 100 rows in the CSV file. Wave Analytics uses the following rules to convert the CSV column names to field labels.

- Replaces special characters and spaces with underscores. For example, "Stage Name" becomes "Stage_Name."
- Replaces consecutive underscores with one underscore, except when column name ends with "__c." For example, "stage*&name" becomes "stage_name."
- Prefixes the field label with "X" when the first character of the column name is numeric. For example, "30Day" becomes "X30Day."
- Replaces the field name with "Column" + column number when all characters in the column name are not alphanumeric. For example, the fourth column name "*&^*(&*(%" becomes "Column4."
- Deletes underscores at the beginning and end of the field label to ensure that it doesn't start or end with an underscore.
- Increments the derived field label if the label is the same as an existing label. For example, if "X2" already exists, uses "X21," "X22," "X23."

Tip: You can download the generated metadata file to change the metadata settings, and then upload it to apply the changes. You can download the metadata file when you create or edit a dataset.

SEE ALSO:

Create a Dataset with External Data

Monitor an External Data Upload

When you upload an external data file, Wave Analytics kicks off a job that uploads the data into the specified dataset. You can use the data monitor to monitor and troubleshoot the upload job.

The Jobs view (1) of the data monitor shows the status, start time, and duration of each dataflow job and external data upload job. It shows jobs for the last 7 days and keeps the logs for 30 days.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

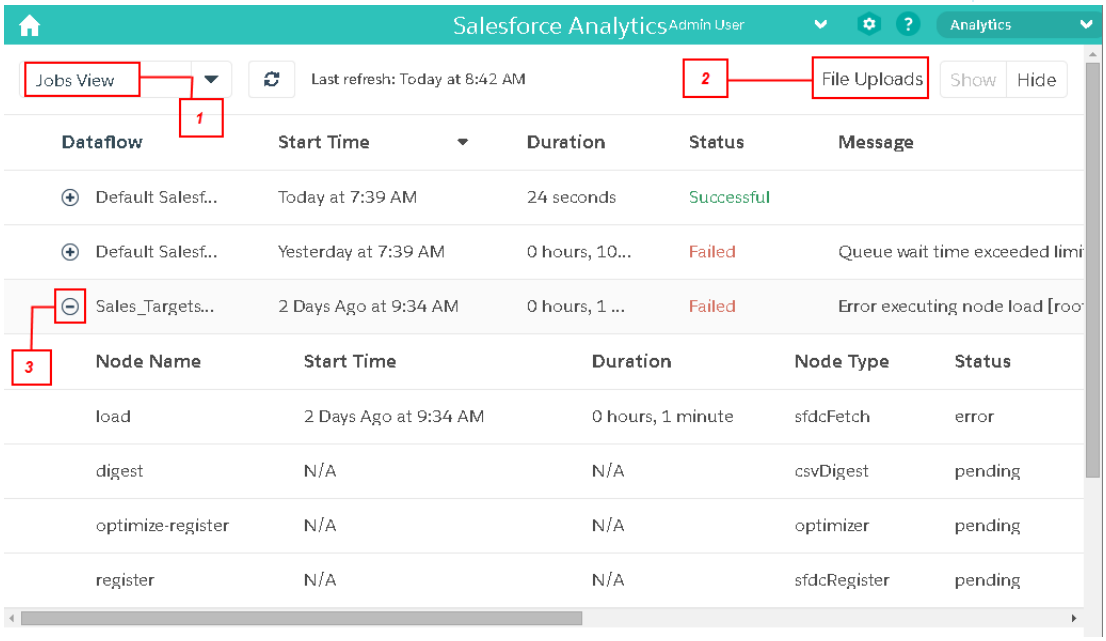
Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To access the data monitor:

- "Edit Wave Analytics Dataflows," "Upload External Data to Wave Analytics," or "Manage Wave Analytics"



1. In Wave Analytics, click the gear button ( ) and then click **Data Monitor** to open the data monitor.
   The Jobs view appears by default. The Jobs view displays dataflow and upload jobs. The Jobs view displays each upload job name as `<dataset_name upload flow>`. You can hover a job to view the entire name.

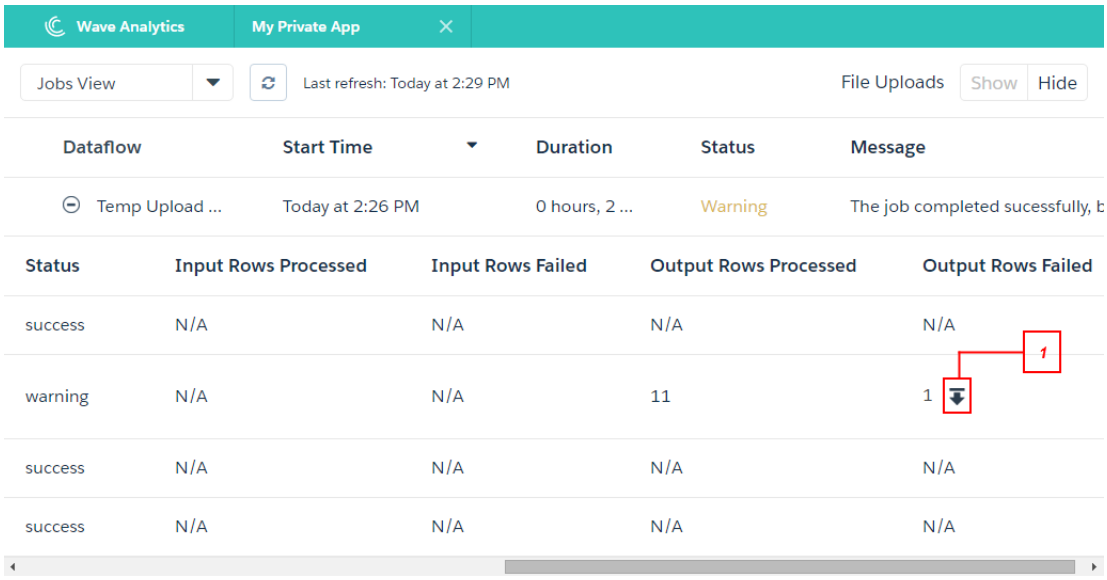
   Note: To view external data upload jobs in the Jobs view, **Show** in the File Uploads field (2) must be selected. It's selected by default.

2. To see the latest status of a job, click the Refresh Jobs button ( ).
   Each job can have one of the following statuses.

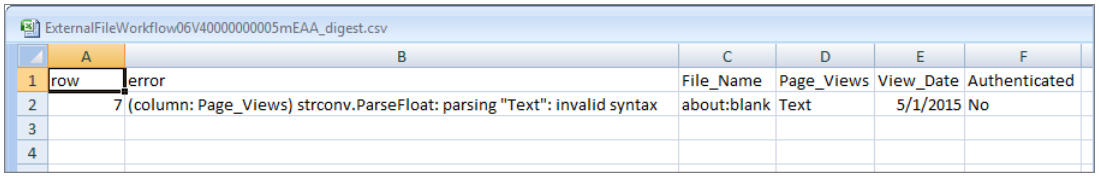
Status	Description
Queued	The job is in queue to start.
Running	The job is running.
Failed	The job failed.
Successful	The job completed successfully.
Warning	The job completed successfully, but some rows failed.

3. To view the run-time details for a job, expand the job node (3).
   The run-time details display under the job. In the run-time details section, scroll to the right to view information about the rows that were processed.
4. To troubleshoot a job that has failed rows, view the error message. Also, click the download button (1) in the run-time details section to download the error log.

   Note: Only the user who uploaded the external data file can see the download button.



The error log contains a list of failed rows.



5. To troubleshoot a failed job, view the error message and the run-time details.

EDIT A DATASET

You can edit a dataset to change the dataset name, app, security predicate, or extended metadata (XMD) file associated with the dataset. For datasets created from an external data file, you can also upload a new external data file or metadata file to update the data or metadata.

If you add an external data file, Wave Analytics generates and adds the corresponding metadata file. To make further changes to the metadata, you can click **Preview Data** or download and edit the generated metadata file. You can also upload your own metadata file to overwrite the generated file.

1. On the home or app page, click the **Datasets** tab.
2. Hover over the dataset that you want to edit, and then click **Edit**.
3. Configure the following options if applicable.

Option	Description
Dataset Name	Enter a new name if you'd like to change the name of the dataset. The name cannot exceed 80 characters.
App	Select a new app if you'd like to move the dataset to a different app.
Add Extended Metadata File (JSON)	Specify an extended metadata file if you'd like to customize the formatting of dashboards associated with the dataset. Refer to *Extended Metadata (XMD) Reference* for information about extended metadata files.
Add External Data File (CSV)	Specify an external data file if you'd like to replace the existing data in the dataset with data from the external data file. Maximum file size is 500 MB. You can upload a .CSV, .GZ, or .ZIP file. Refer to *External Data Format Reference* for information about external data files and metadata files.
Add Metadata File (JSON)	Specify a metadata file if you'd like to redefine the structure of the external data file. If you upload a new metadata file, you must also upload the corresponding external data file. Refer to *External Data Format Reference* for information about metadata files.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To edit a dataset:

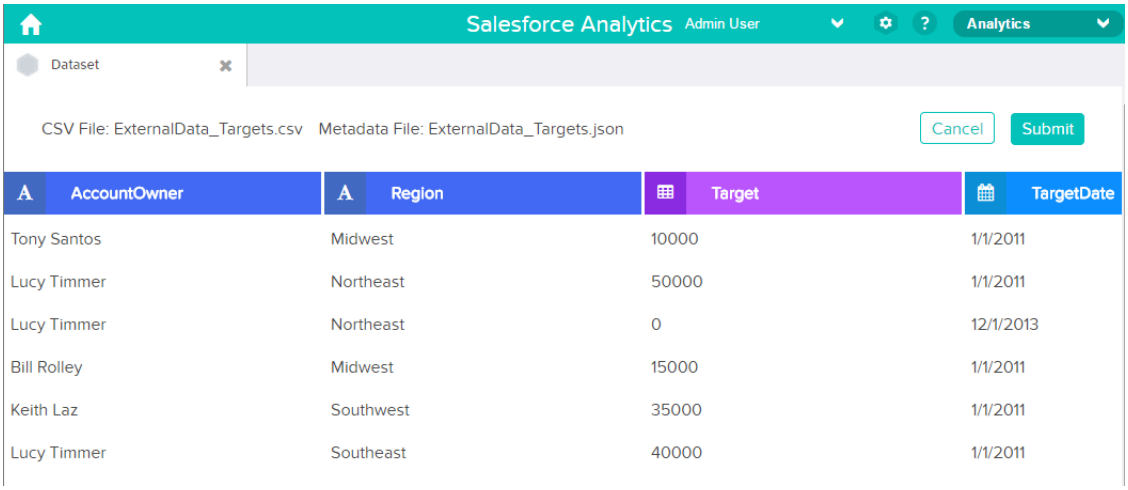
- "Edit Wave Analytics Dataflows"

Option	Description
Security Predicate	Add a security predicate if you'd like to apply row-level security on the dataset. For information about predicates, see Row-Level Security for Datasets.

4. If you uploaded a new .csv file, click **Preview Data** to view and change the required metadata attributes.
   You can change the optional metadata later.

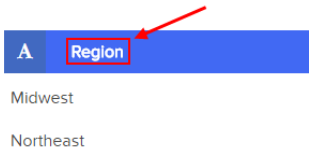
   Note: The **Preview Data** button is disabled if you uploaded your own metadata file.

   After you click **Preview Data**, the preview page appears.

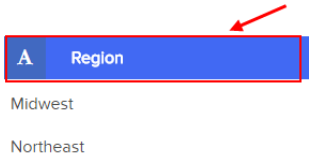


5. For each column:

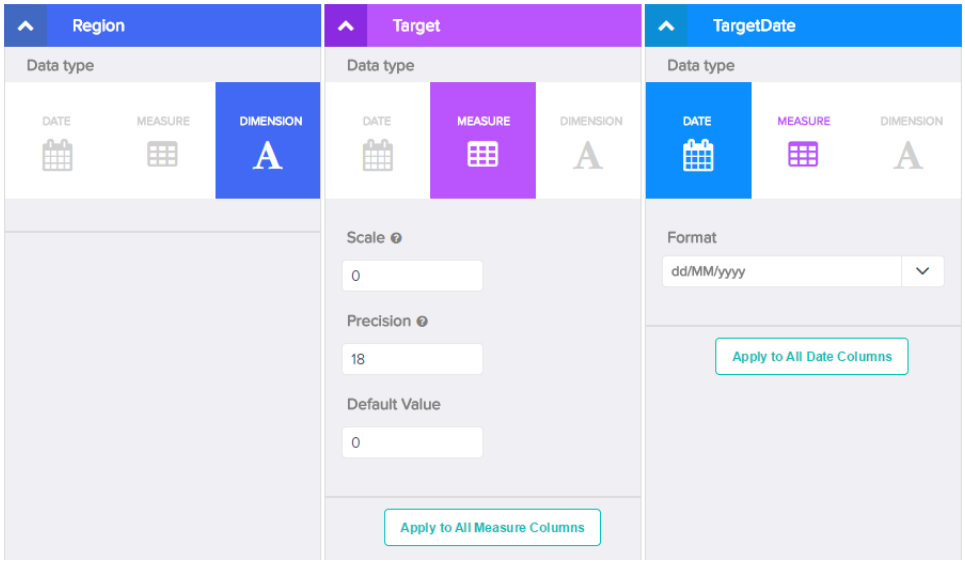
   a. Click a column name to change it. The column name is the display name in the dataset. The column name cannot exceed 40 characters.



   b. Click the column header to change other required attributes for the column.



You can change the attributes for measure and date columns only.



c. To apply the changes to all other columns of the same data type, click **Apply to All <data type> Columns**.

6. Click **Submit** to save the metadata changes in the preview page to the metadata file.

   Note: The **Submit** button is grayed out if there are errors.

7. Click **OK** to close the confirmation message.
8. To change optional metadata attributes—which are not visible in the preview page—click to download the metadata file, edit the file, and then upload it.
9. Click **Update Dataset**.
10. Click **Continue** to dismiss the confirmation message.

DELETE A DATASET

Delete unnecessary datasets from your My Private App or in shared apps on which you have at least Editor access. Removing datasets reduces clutter and helps you avoid reaching your org's limit for rows across registered datasets.

When you delete a dataset, Wave Analytics permanently deletes the dataset and doesn't delete the corresponding lenses or dashboards that reference the dataset. Lenses and dashboards that reference a deleted dataset will no longer be available. As a result, Salesforce.com recommends that you remove the associated lenses and dashboards before you delete a dataset.

If a dataflow transformation —like edgemart or sfdcRegister— references the dataset, you must remove the reference before you can delete the dataset. For example, to delete the "Opportunities" dataset, you must remove the sfdcRegister transformation from the dataflow snippet shown below.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To delete a dataset:

- "Edit Wave Analytics Dataflows"

```
{
...
   "Register_Dataset": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "Opportunities",
         "name": "Opportunities",
         "source": "Extract_Opportunities"
      }
   },
...}
```

Warning: You can't recover a deleted dataset.

1. On the home or app page, click the **Datasets** tab.
2. Hover over the dataset that you want to delete, and then click **Edit**.
3. Click **Delete Dataset**.
   If applicable, Wave Analytics shows a list of all lenses and dashboards that reference the dataset and that you have access to view. After you delete the dataset, any lens or dashboard that reference the dataset will become unusable.
4. Click **Delete Permanently** and confirm.

ROW-LEVEL SECURITY FOR DATASETS

If a Wave Analytics user has access to a dataset, they have access to all records in the dataset, by default. However, you can implement row-level security on a dataset to restrict access to records. Some records might contain sensitive data that shouldn't be accessible by everyone.

To implement row-level security, define a predicate for each dataset on which you want to restrict access to records. A *predicate* is a filter condition that defines row-level access to records in a dataset.

When a user submits a query against a dataset that has a predicate, Wave Analytics checks the predicate to determine which records the user has access to. If the user doesn't have access to a record, Wave Analytics does not return that record.

The predicate is flexible and can model different types of security policies. For example, you can create predicates based on:

- Record ownership. Enables each user to view only records that they own.
- Management visibility. Enables each user to view records owned or shared by their subordinates based on a role hierarchy.
- Team or account collaboration. Enables all members of a team, like an opportunity team, to view records shared with the team.
- Combination of different security requirements. For example, you might need to define a predicate based on the Salesforce role hierarchy, teams, and record ownership.

The type of security policy you implement depends on how you want to restrict access to records in the dataset.

Warning: If row-level security isn't applied to a dataset, any user that has access to the dataset can view all records in the dataset.

IN THIS SECTION:

Considerations when Defining a Predicate for a Dataset

Applying a predicate to a dataset is more than just defining the predicate expression. You also need to consider how the predicate is dependent on the information in the dataset and where to define the predicate expression.

Row-Level Security Example based on Record Ownership

Let's look at an example where you create a dataset based on a CSV file and then implement row-level security based on record ownership. In this example, you will create a dataset that contains sales targets for account owners. To restrict access on each record in the dataset, you will create a security policy where each user can view only sales targets for accounts that they own. This process requires multiple steps that are described in the sections that follow.

Row-Level Security Example based on Opportunity Teams

Let's look at an example where you create a dataset based on Salesforce data and then implement row-level security based on an opportunity team. In this example, you will create a dataset that contains only opportunities associated with an opportunity team. To restrict access on each record in the dataset, you will create a security policy where only opportunity members can view their opportunity. This process requires multiple steps that are described in the sections that follow.

Row-Level Security Example based on Role Hierarchy and Record Ownership

Let's look at an example where you create a dataset based on Salesforce data and then implement row-level security based on the Salesforce role hierarchy and record ownership. In this example, you will create a dataset that contains all opportunities. To restrict access on each record in the dataset, you will create a security policy where each user can view only opportunities that they own or that are owned by their subordinates based on the Salesforce role hierarchy. This process requires multiple steps that are described in the sections that follow.

SEE ALSO:

sfdcRegister Transformation

sfdcRegister Parameters

Considerations when Defining a Predicate for a Dataset

Applying a predicate to a dataset is more than just defining the predicate expression. You also need to consider how the predicate is dependent on the information in the dataset and where to define the predicate expression.

You can create a predicate expression based on information in the dataset. For example, to enable each user to view only dataset records that they own, you can create a predicate based on a dataset column that contains the owner for each record. If needed, you can load additional data into a dataset required by the predicate.

The location where you define the predicate varies.

- To apply a predicate on a dataset created from a dataflow, add the predicate in the **rowLevelSecurityFilter** field of the Register transformation. The next time the dataflow runs, Wave Analytics will apply the predicate.
- To apply a predicate on a dataset created from an external data file, define the predicate in the **rowLevelSecurityFilter** field in the metadata file associated with the external data file. Wave Analytics applies the predicate when you upload the metadata file and external data file. If you already created the dataset from a external data file, you can edit the dataset to apply or change the predicate.

Row-Level Security Example based on Record Ownership

Let's look at an example where you create a dataset based on a CSV file and then implement row-level security based on record ownership. In this example, you will create a dataset that contains sales targets for account owners. To restrict access on each record in the dataset, you will create a security policy where each user can view only sales targets for accounts that they own. This process requires multiple steps that are described in the sections that follow.

Note: Although this example is about applying a predicate to a dataset created from a CSV file, this procedure can also be applied to a dataset that is created from Salesforce data.

IN THIS SECTION:

1. Determine Which Data to Include in the Dataset
   First, determine what data you want to include in the dataset. For this example, you will create a Targets dataset that contains all sales targets.
2. Determine Row-Level Security for Dataset
   Now it's time to think about row-level security. How will you restrict access to each record in this dataset?
3. Add the Predicate to the Metadata File
   For a dataset created from a CSV file, you can specify the predicate in the metadata file associated with the CSV file or when you edit the dataset.

4. Create the Dataset
   Now that you updated the metadata file with the predicate, you can create the dataset.
5. Test Row-Level Security for the Dataset
   You must verify that the predicate is applied properly and that each user can see their own sales targets.

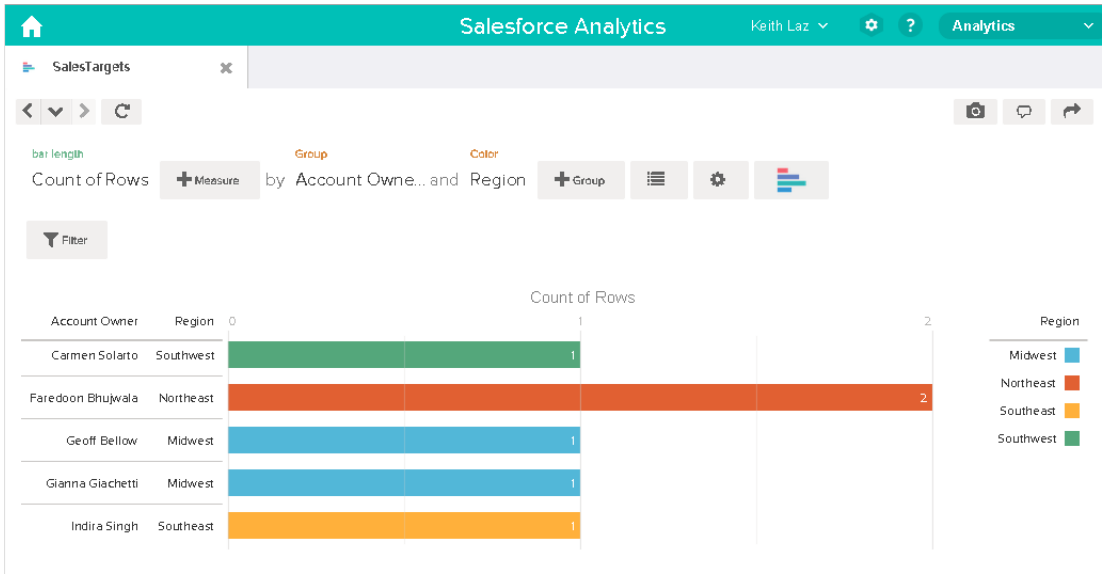
Determine Which Data to Include in the Dataset

First, determine what data you want to include in the dataset. For this example, you will create a Targets dataset that contains all sales targets.

You will obtain sales targets from the CSV file shown below.

AccountOwner	Region	Target	TargetDate
Tony Santos	Midwest	10000	1/1/2011
Lucy Timmer	Northeast	50000	1/1/2011
Lucy Timmer	Northeast	0	12/1/2013
Bill Rolley	Midwest	15000	1/1/2011
Keith Laz	Southwest	35000	1/1/2011
Lucy Timmer	Southeast	40000	1/1/2011

If you were to create the dataset without implementing row-level security, any user that had access to the dataset would be able to see the sales targets for all account owners. For example, as shown below, Keith would be able to view the sales targets for all account owners.



You need to apply row-level security to restrict access to records in this dataset.

Determine Row-Level Security for Dataset

Now it's time to think about row-level security. How will you restrict access to each record in this dataset?

You decide to implement the following predicate on the dataset.

```
'AccountOwner' == "$User.Name"
```

Note: All predicate examples in this document escape the double quotes because it's required when you enter the predicate in the Register transformation or metadata file.This predicate implements row-level security based on record ownership. Based on the predicate, Wave Analytics returns a sales target record when the user who submits the query on the dataset is the account owner.

Let's take a deeper look into the predicate expression:

- AccountOwner refers to the dataset column that stores the full name of the account owner for each sales target.
- $User.Name refers to the Name column of the User object that stores the full name of each user. Wave Analytics performs a lookup to get the full name of the user who submits each query.

Note: The lookup returns a match when the names in AccountOwner and $User.Name match exactly—they must have the same case.

Add the Predicate to the Metadata File

For a dataset created from a CSV file, you can specify the predicate in the metadata file associated with the CSV file or when you edit the dataset.

You must escape the double quotes around string values when entering a predicate in the metadata file.

In this example, you add the predicate to the metadata file shown below.

```
{
   "fileFormat": {
   "charsetName": "UTF-8",
   "fieldsDelimitedBy": ",",
   "fieldsEnclosedBy": "\"",
   "numberOfLinesToIgnore": 1 },
   "objects": [
      {
         "name": "Targets",
         "fullyQualifiedName": "Targets",
         "label": "Targets",
         "rowLevelSecurityFilter": "'AccountOwner' == \"$User.Name\"",
         "fields": [
      {
         "name": "AccountOwner",
         "fullyQualifiedName": "Targets.AccountOwner",
         "label": "Account Owner",
         "type": "Text"
      },
      {
         "name": "Region",
         "fullyQualifiedName": "Targets.Region",
         "label": "Region",
         "type": "Text"
      },
```

```
      {
         "name": "Target",
         "fullyQualifiedName": "Targets.Target",
         "label": "Target",
         "type": "Numeric",
         "precision": 16,
         "scale": 0,
         "defaultValue": "0",
         "format": null
      },
      {
         "name": "TargetDate",
         "fullyQualifiedName": "Targets.TargetDate",
         "label": "TargetDate",
         "description": "",
         "type": "Date",
         "format": "dd/MM/yy HH:mm:ss",
         "isSystemField": false,
         "fiscalMonthOffset": 0
      }
      ]
      }
   ]
}
```

Create the Dataset

Now that you updated the metadata file with the predicate, you can create the dataset.

Warning: If you wish to perform the steps in this sample implementation, perform the steps in a non-production environment. Ensure that these changes do not impact other datasets that you already created.

To create the dataset, perform the following steps.

1. In Wave Analytics, go to the home page.
2. Click **Create** > **Dataset**
3. Click **CSV**.

   The following screen appears.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

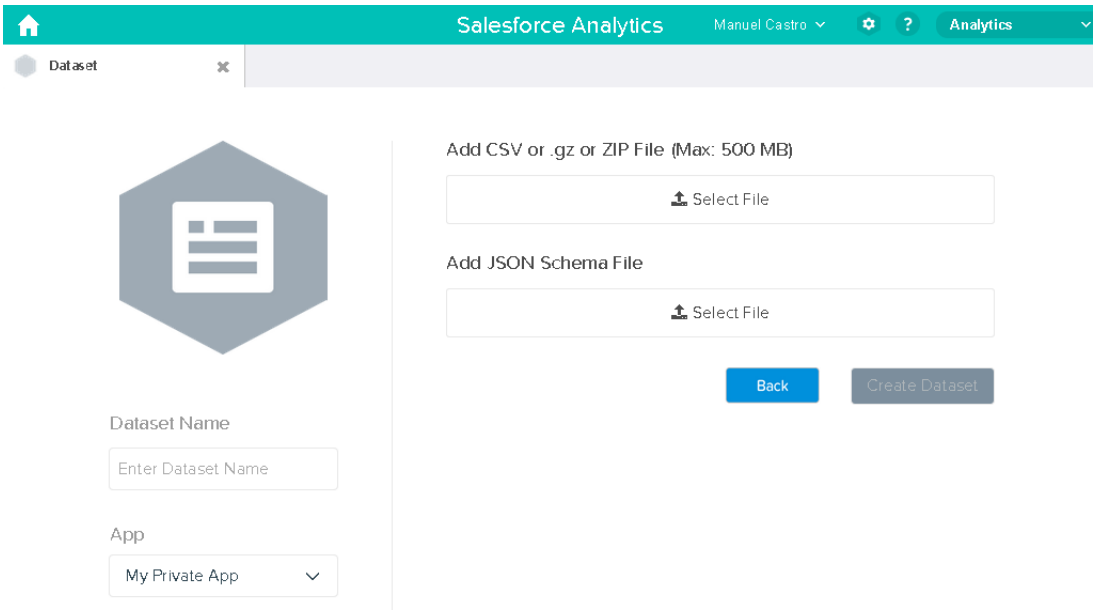
Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To upload a CSV and metadata file:

- "Upload External Data to Wave Analytics"



4. Select the CSV file and metadata (schema) file.
5. In the **Dataset Name** field, enter "SalesTarget" as the name of the dataset.
6. Optionally, choose a different app where you want to store the dataset.
7. Click **Create Dataset**.

   Wave Analytics confirms that the upload is successful and then creates a job to create the dataset. You can view the SalesTarget dataset after the job completes successfully.
8. To verify that the job completes successfully, perform the following steps:
   a. Click the gear icon ( ) and then select **Data Monitor** to open the data monitor.

      By default, the Jobs View of the data monitor appears. It shows the statuses of dataflow and external data upload jobs.
   b. Click the Refresh Jobs button ( ) to view the latest statuses of the jobs.

Test Row-Level Security for the Dataset

You must verify that the predicate is applied properly and that each user can see their own sales targets.

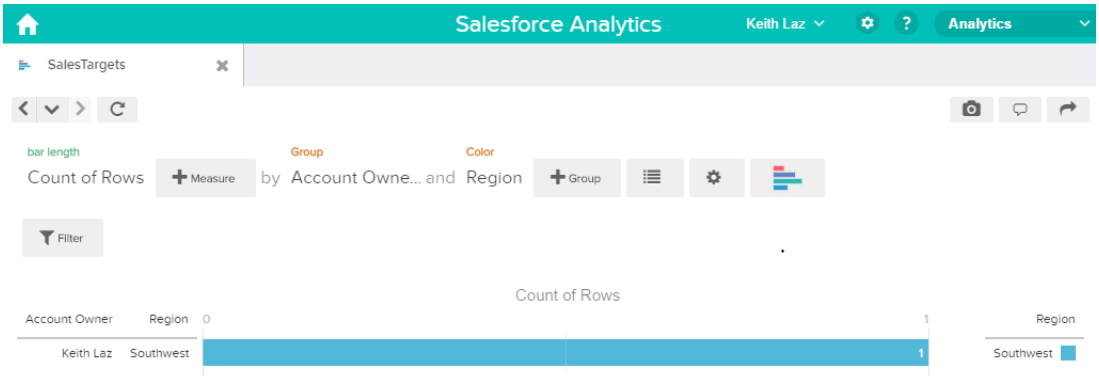
1. Log in to Wave Analytics as Keith.
2. Open the SalesTargets dataset.
   As shown in the following lens, notice that Keith can see only his sales target.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions



Row-Level Security Example based on Opportunity Teams

Let's look at an example where you create a dataset based on Salesforce data and then implement row-level security based on an opportunity team. In this example, you will create a dataset that contains only opportunities associated with an opportunity team. To restrict access on each record in the dataset, you will create a security policy where only opportunity members can view their opportunity. This process requires multiple steps that are described in the sections that follow.

IN THIS SECTION:

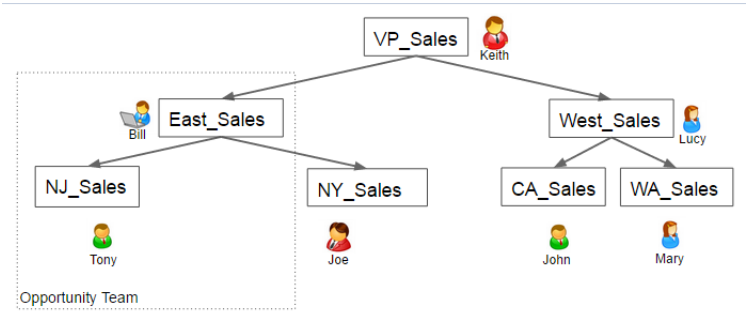
1. Determine Which Data to Include in the Dataset
   First, determine what data you want to include in the dataset. For this example, you will create an OppTeamMember dataset that contains only opportunities associated with an opportunity team.
2. Design the Dataflow to Load the Data
   Now it's time to figure out how the dataflow will extract the Salesforce data and load it into a dataset. You start by creating this high-level design for the dataflow.
3. Determine Row-Level Security for the Dataset
   Now it's time to think about row-level security. How will you restrict access to each record in this dataset?
4. Modify the Dataflow Based on Row-Level Security
   It's now time to add the predicate in the dataflow definition file.
5. Create the Dataset
   Now that you have the final dataflow definition file, you can create the dataset.
6. Test Row-Level Security for the Dataset
   You must verify that the predicate is applied properly and that each user can see the appropriate opportunities.

Determine Which Data to Include in the Dataset

First, determine what data you want to include in the dataset. For this example, you will create an OppTeamMember dataset that contains only opportunities associated with an opportunity team.

You will obtain opportunities from the Opportunity object and the opportunity teams from the OpportunityTeamMember object. Both are Salesforce objects.

In this example, your Salesforce organization has the following opportunity team and users.



Your organization also contains the following opportunities, most of which are owned by Keith.

All Opportunities ▾ Edit | Delete | Create New View — List | Feed

New Opportunity

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Other | All

Action	Opportunity Name ↑	Account Name	Amount	Close Date	Stage	Opportunity Owner Alias
Edit | Del	Acc - 1000 Widgets	Acc_salesrep		9/4/2014	Prospecting	Tony
Edit | Del	Acme - 1,200 Widgets	Acme	$140,000.00	6/14/2012	Value Proposition	Keith
Edit | Del	Acme - 200 Widgets	Acme	$20,000.00	10/13/2012	Prospecting	Keith
Edit | Del	Acme - 600 Widgets	Acme	$70,000.00	8/10/2012	Needs Analysis	Keith
Edit | Del	ESales_01	East_Sales_acc_01		9/4/2014	Prospecting	Bill
Edit | Del	Global Media - 400…	Global Media	$40,000.00	7/13/2012	Id. Decision Makers	Keith
Edit | Del	salesforce.com - 1,…	salesforce.com	$100,000.00	6/14/2012	Negotiation/Review	Keith
Edit | Del	salesforce.com - 2,…	salesforce.com	$20,000.00	8/12/2012	Value Proposition	Keith
Edit | Del	salesforce.com - 50…	Global Media	$50,000.00	5/12/2012	Closed Won	Keith
Edit | Del	salesforce.com - 50…	Global Media	$500,000.00	5/12/2012	Closed Won	Keith
Edit | Del	West_Sales_01	West_Sales_Acc_01		9/4/2014	Prospecting	Lucy

Acc - 1000 Widgets is the only opportunity shared by an opportunity team. Bill is the Sales Manager for this opportunity. Tony is the opportunity owner.

EDITIONS

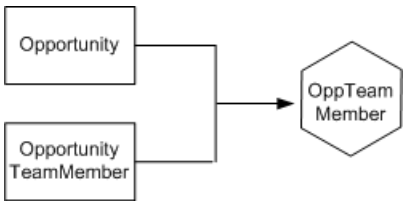
Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

Design the Dataflow to Load the Data

Now it's time to figure out how the dataflow will extract the Salesforce data and load it into a dataset. You start by creating this high-level design for the dataflow.



The dataflow will extract data from the Opportunity and OpportunityTeamMember objects, join the data, and then load it into the OppTeamMember dataset.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

Now let's implement that design in JSON, which is the format of the dataflow definition file. A dataflow definition file contains transformations that extract, transform, and load data into a dataset.

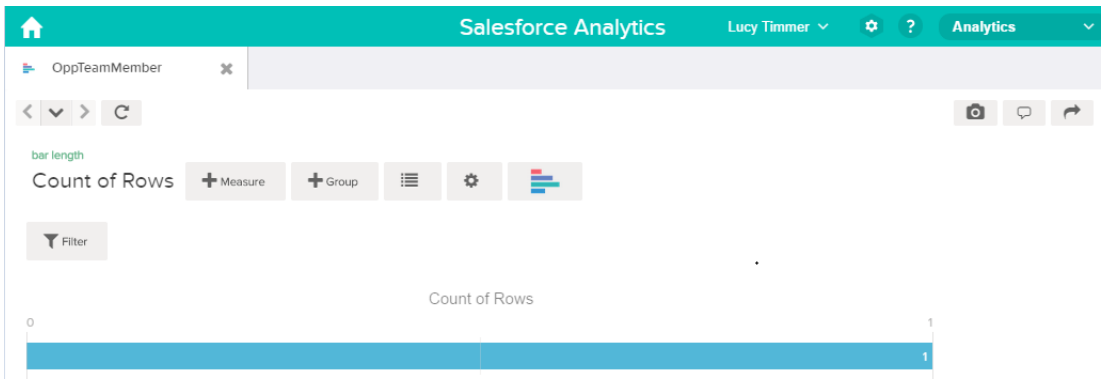
Based on the design, you create the JSON shown below.

```
{
   "Extract_OpportunityTeamMember": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "OpportunityTeamMember",
         "fields": [
            { "name": "Name" },
            { "name": "OpportunityId" },
            { "name": "UserId" }
         ]
      }
   },
   "Extract_Opportunity": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "Amount" },
            { "name": "StageName" },
            { "name": "AccountId" },
            { "name": "OwnerId" }
         ]
      }
   },
   "Augment_OpportunityTeamMember_Opportunity": {
      "action": "augment",
      "parameters": {
         "left": "Extract_OpportunityTeamMember",
         "left_key": [
            "OpportunityId"
         ],
         "relationship": "TeamMember",
```

```
        "right": "Extract_Opportunity",
        "right_key": [
           "Id"
        ],
        "right_select": [
           "Name","Amount"
        ]
     }
  },
  "Register_Dataset": {
     "action": "sfdcRegister",
     "parameters": {
        "alias": "OppTeamMember",
        "name": "OppTeamMember",
        "source": "Augment_OpportunityTeamMember_Opportunity",
        "rowLevelSecurityFilter": ""
     }
  }
}
```

If you were to run this dataflow, Wave Analytics would generate a dataset with no row-level security. As a result, any user that has access to the dataset would be able to see the opportunity shared by the opportunity team.

For example, as shown below, Lucy would be able to view the opportunity that belongs to an opportunity team of which she is not a member.



You need to apply row-level security to restrict access to records in this dataset.

Determine Row-Level Security for the Dataset

Now it's time to think about row-level security. How will you restrict access to each record in this dataset?

You decide to implement the following predicate on the dataset.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

```
'UserId' == "$User.Id"
```

This predicate compares the UserId column in the dataset against the ID of the user running a query against the dataset. The UserId column in the dataset contains the user ID of the team member associated with each opportunity. To determine the ID of the user running the query, Wave Analytics looks up the ID of the user making the query in the User object.

For each match, Wave Analytics returns the record to the user.

Modify the Dataflow Based on Row-Level Security

It's now time to add the predicate in the dataflow definition file.

You add the predicate to the Register transformation that registers the OppTeamMember dataset as shown below.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

```
{
   "Extract_OpportunityTeamMember": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "OpportunityTeamMember",
         "fields": [
            { "name": "Name" },
            { "name": "OpportunityId" },
            { "name": "UserId" }
         ]
      }
   },
   "Extract_Opportunity": {
```

```
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "Amount" },
            { "name": "StageName" },
            { "name": "AccountId" },
            { "name": "OwnerId" }
         ]
      }
   },
   "Augment_OpportunityTeamMember_Opportunity": {
      "action": "augment",
      "parameters": {
         "left": "Extract_OpportunityTeamMember",
         "left_key": [
            "OpportunityId"
         ],
         "relationship": "TeamMember",
         "right": "Extract_Opportunity",
         "right_key": [
            "Id"
         ],
         "right_select": [
            "Name","Amount"
         ]
      }
   },
   "Register_Dataset": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "OppTeamMember",
         "name": "OppTeamMember",
         "source": "105_Augment_OpportunityTeamMember_Opportunity",
         "rowLevelSecurityFilter": "'UserId' == \"$User.Id\""
      }
   }
}
```

Create the Dataset

Now that you have the final dataflow definition file, you can create the dataset.

Warning: If you wish to perform the steps in this sample implementation, verify that you have all required Salesforce objects and fields, and perform the steps in a non-production environment. Ensure that these changes do not impact other datasets that you already created. Also, always make a backup of the existing dataflow definition file before you make changes because you cannot retrieve old versions of the file.

To create the dataset, perform the following steps.

1. In Wave Analytics, click the gear icon ( ) and then select **Data Monitor** to open the data monitor.
   The Jobs view of the data monitor appears by default.
2. Select **Dataflow View**.
3. Click the actions list (1) for the dataflow and then select **Download** to download the existing dataflow definition file.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

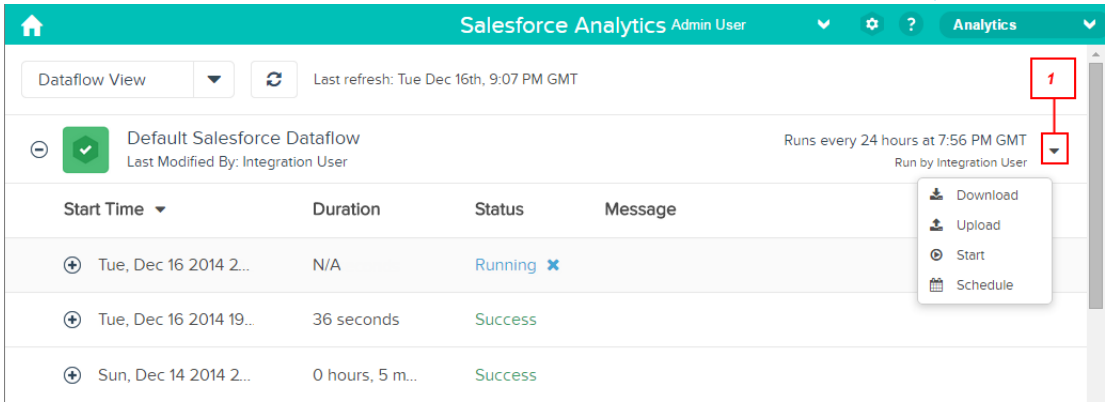
Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To download, upload, run, and monitor a dataflow:

- "Edit Wave Analytics Dataflows"



4. Open the dataflow definition file in a JSON or text editor.
5. Add the JSON determined in the previous step.
6. Before you save the dataflow definition file, use a JSON validation tool to verify that the JSON is valid.
   An error occurs if you try to upload the dataflow definition file with invalid JSON. You can find JSON validation tool on the internet.
7. Save and close the dataflow definition file.
8. In the Dataflow View of the data monitor, click the actions list for the dataflow and then select **Upload**.
9. Select the updated dataflow definition file and click **Upload**.
10. In the Dataflow View of the data monitor, click the actions list for the dataflow and then select **Run** to run the dataflow job.
11. Click the **Refresh Jobs** button ( ) to view the latest status of the dataflow job.
    You can view the OppTeamMember dataset after the dataflow job completes successfully.

Note: If you are adding a predicate to a dataset that was previously created, each user must log out and log back in for the predicate to take effect.

Test Row-Level Security for the Dataset

You must verify that the predicate is applied properly and that each user can see the appropriate opportunities.

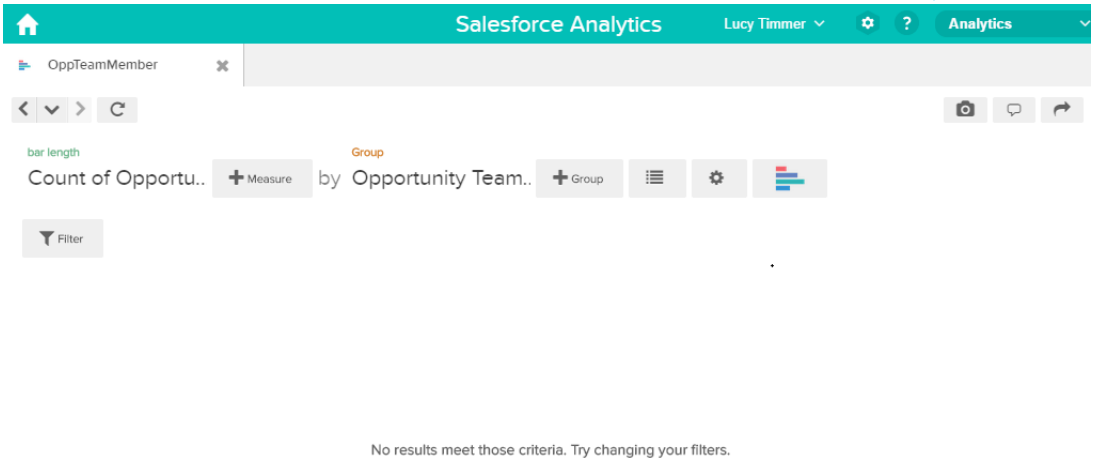
1. Log in to Wave Analytics as Lucy.
2. Open the OppTeamMember opportunity.
   Notice that Lucy can't view the opportunity associated with the opportunity team anymore because she is not a member of the team.

EDITIONS

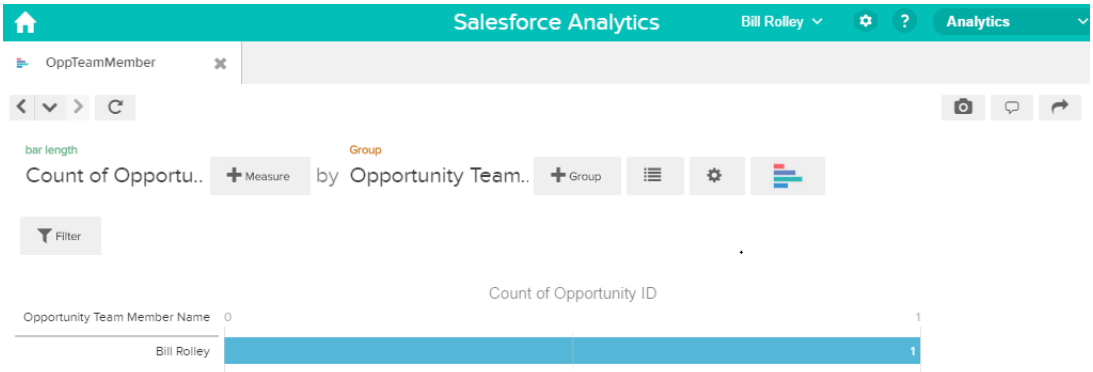
Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions



3. Log out and now log in as Bill.
   Bill can view the opportunity that is shared by the opportunity team of which he is a member.



Row-Level Security Example based on Role Hierarchy and Record Ownership

Let's look at an example where you create a dataset based on Salesforce data and then implement row-level security based on the Salesforce role hierarchy and record ownership. In this example, you will create a dataset that contains all opportunities. To restrict access on each record in the dataset, you will create a security policy where each user can view only opportunities that they own or that are owned by their subordinates based on the Salesforce role hierarchy. This process requires multiple steps that are described in the sections that follow.

IN THIS SECTION:

1. Determine Which Data to Include in the Dataset
   First, determine what data you want to include in the dataset. For this example, you will create the OppRoles dataset that contains all opportunities as well as user details about each opportunity owner, such as their full name, division, and title.
2. Design the Dataflow to Load the Data
   Now it's time to figure out how the dataflow will extract the data and load it into a dataset. You start by creating this high-level design for the dataflow.
3. Determine Row-Level Security for the Dataset
   Now it's time to think about row-level security. How will you restrict access to each record in this dataset?
4. Modify the Dataflow Based on Row-Level Security
   Now it's time to modify the dataflow definition file to account for the predicate.
5. Create the Dataset
   Now that you have the final dataflow definition file, you can create the dataset.
6. Test Row-Level Security for the Dataset
   You must verify that the predicate is applied properly and that each user can see the appropriate opportunities.

SEE ALSO:

flatten Parameters

Determine Which Data to Include in the Dataset

First, determine what data you want to include in the dataset. For this example, you will create the OppRoles dataset that contains all opportunities as well as user details about each opportunity owner, such as their full name, division, and title.

You will obtain opportunities from the Opportunity object and user details from the User object. Both are objects in Salesforce.

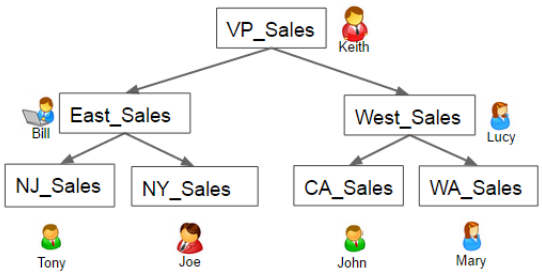
In this example, your Salesforce organization has the following role hierarchy and users.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions



Also, your organization contains the following opportunities, most of which are owned by Keith.

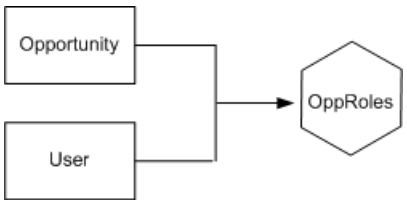
All Opportunities | Edit | Delete | Create New View

New Opportunity

Action	Opportunity Name	Account Name	Amount	Close Date	Stage	Opportunity Owner Alias
Edit | Del	Acc - 1000 Widgets	Acc_salesrep		9/4/2014	Prospecting	Tony
Edit | Del	Acme - 1,200 Widgets	Acme	$140,000.00	6/14/2012	Value Proposition	Keith
Edit | Del	Acme - 200 Widgets	Acme	$20,000.00	10/13/2012	Prospecting	Keith
Edit | Del	Acme - 600 Widgets	Acme	$70,000.00	8/10/2012	Needs Analysis	Keith
Edit | Del	ESales_01	East_Sales_acc_01		9/4/2014	Prospecting	Bill
Edit | Del	Global Media - 400	Global Media	$40,000.00	7/13/2012	Id. Decision Makers	Keith
Edit | Del	salesforce.com - 1	salesforce.com	$100,000.00	6/14/2012	Negotiation/Review	Keith
Edit | Del	salesforce.com - 2	salesforce.com	$20,000.00	8/12/2012	Value Proposition	Keith
Edit | Del	salesforce.com - 50	Global Media	$50,000.00	5/12/2012	Closed Won	Keith
Edit | Del	salesforce.com - 50	Global Media	$500,000.00	5/12/2012	Closed Won	Keith
Edit | Del	West_Sales_01	West_Sales_Acc_01		9/4/2014	Prospecting	Lucy

Design the Dataflow to Load the Data

Now it's time to figure out how the dataflow will extract the data and load it into a dataset. You start by creating this high-level design for the dataflow.



The dataflow will extract data from the Opportunity and User objects, join the data, and then load it into the OppRoles dataset.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

Now let's implement that design in JSON, which is the format of the dataflow definition file. A dataflow definition file contains transformations that extract, transform, and load data into a dataset.

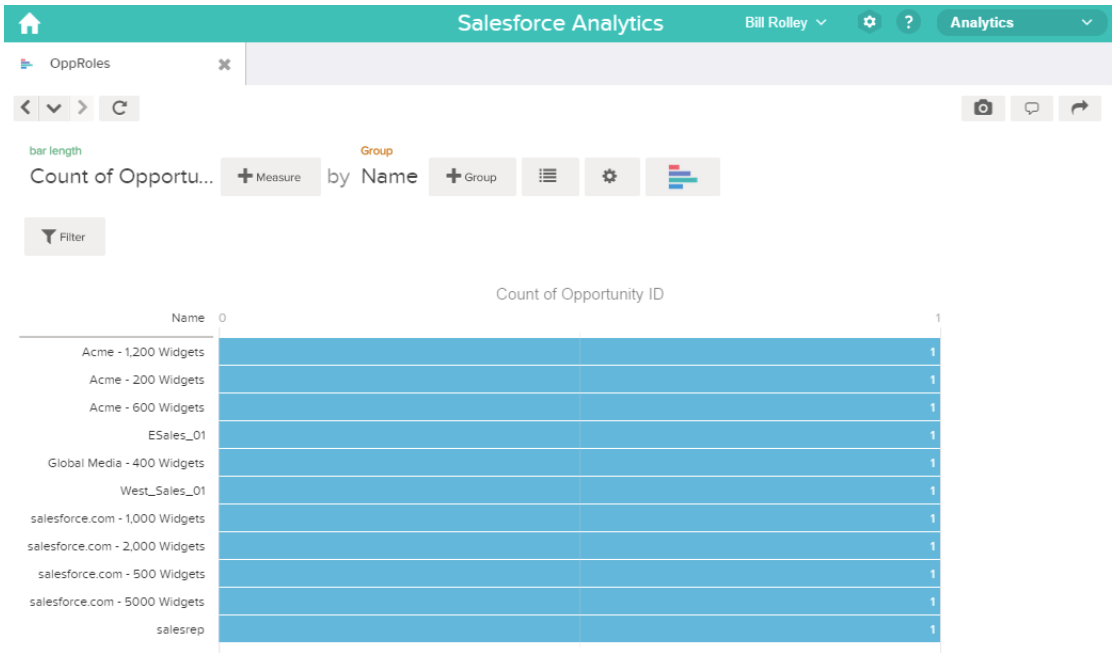
Based on the design, you create the JSON shown below.

```
{
   "Extract_Opportunity": {
      "action": "sfdcDigest",
```

```
    "parameters": {
    "object": "Opportunity",
    "fields": [
       { "name": "Id" },
       { "name": "Name" },
       { "name": "Amount" },
       { "name": "StageName" },
       { "name": "AccountId" },
       { "name": "OwnerId" }
      ]
    }
},
"Extract_User": {
    "action": "sfdcDigest",
    "parameters": {
    "object": "User",
    "fields": [
       { "name": "Id" },
       { "name": "Username" },
       { "name": "LastName" },
       { "name": "FirstName" },
       { "name": "Name" },
       { "name": "CompanyName" },
       { "name": "Division" },
       { "name": "Department" },
       { "name": "Title" },
       { "name": "Alias" },
       { "name": "CommunityNickname" },
       { "name": "UserType" },
       { "name": "UserRoleId" }
       ]
    }
},
"Augment_Opportunity_User": {
    "action": "augment",
    "parameters": {
       "left": "Extract_Opportunity",
       "left_key": [
          "OwnerId"
       ],
       "right": "Extract_User",
       "relationship": "Owner",
       "right_select": [
          "Name"
       ],
       "right_key": [
          "Id"
       ]
    }
},
"Register": {
    "action": "sfdcRegister",
    "parameters": {
       "alias": "OppRoles",
```

```
        "name": "OppRoles",
        "source": "Augment_Opportunity_User",
        "rowLevelSecurityFilter": ""
      }
   }
}
```

If you were to run this dataflow, Wave Analytics would generate a dataset with no row-level security. As a result, any user that has access to the dataset would be able to view all opportunities. For example, as shown below, Bill would be able to view all opportunities, including those owned by his manager Keith.



You need to apply row-level security to restrict access to records in this dataset.

Determine Row-Level Security for the Dataset

Now it's time to think about row-level security. How will you restrict access to each record in this dataset?

You decide to implement the following predicate on the dataset.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

```
'ParentRoleIDs' == "$User.UserRoleId" || 'OwnerId' == "$User.Id"
```

Note: The current dataflow doesn't contain logic to create a dataset column named "ParentRoleIDs." ParentRoleIDs is a placeholder for the name of a column that will contain this information. In the next step, you will modify the dataflow to add this column to the dataset. This column name will change based on how you configure the dataflow.

Based on the predicate, Wave Analytics returns an opportunity record if:

- The user who submits the query is a parent of the opportunity owner based on the Salesforce role hierarchy. Wave Analytics determines this based on their role IDs and the role hierarchy.
- Or, the user who submits the query on the dataset is the opportunity owner.

Let's examine both parts of this predicate.

Predicate Part	Description
'ParentRoleIDs' == "$User.UserRoleId"	• ParentRoleIDs refers to a dataset column that contains a comma-separated list of role IDs of all users above the opportunity owner based on the role hierarchy. You will create this dataset column in the next section. • $User.UserRoleId refers to the UserRoleId column of the User object. Wave Analytics looks up the user role ID of the user who submits the query from the User object.
'OwnerId' == "$User.Id"	• OwnerId refers to the dataset column that contains the user ID of the owner of each opportunity. • $User.Id refers to the Id column of the User object. Wave Analytics looks up the user ID of the user who submits the query from the User object.

Modify the Dataflow Based on Row-Level Security

Now it's time to modify the dataflow definition file to account for the predicate.

In this scenario, you have to make changes to the dataflow based on the predicate.

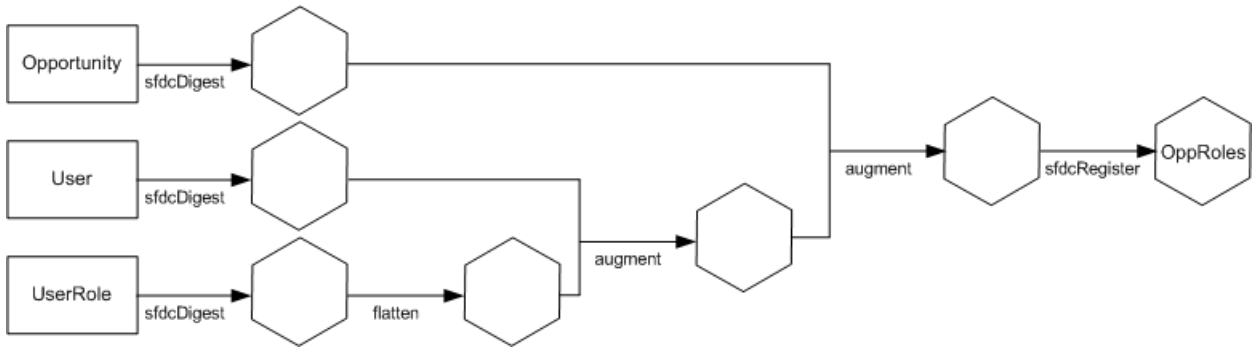
- Add a column in the dataset that stores a comma-separated list of the role IDs of all parents for each opportunity owner. When you defined the predicate in the previous step, you temporarily referred to this column as "ParentRoleIDs." To add the column, you redesign the dataflow as shown in the following diagram:

EDITIONS

Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions



The new dataflow design contains the following changes:

- Extracts the role IDs from the UserRole object.
- Uses the Flatten transformation to generate a column that stores a comma-separated list of the role IDs of all parents of each user. When you determined the predicate in the previous step, you temporarily referred to this column as "ParentRoleIDs."
- Link the new column to the OppRoles dataset.

- Add the predicate to the Register transformation that registers the OppRoles dataset.

You modify the dataflow as shown below.

```
{
   "Extract_Opportunity": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Opportunity",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "Amount" },
            { "name": "StageName" },
            { "name": "AccountId" },
            { "name": "OwnerId" }
         ]
      }
   },
   "Extract_User": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "User",
         "fields": [
            { "name": "Id" },
            { "name": "Username" },
            { "name": "LastName" },
            { "name": "FirstName" },
            { "name": "Name" },
            { "name": "CompanyName" },
            { "name": "Division" },
            { "name": "Department" },
            { "name": "Title" },
            { "name": "Alias" },
            { "name": "CommunityNickname" },
```

```
            { "name": "UserType" },
            { "name": "UserRoleId" }
         ]
      }
   },
   "Extract_UserRole": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "UserRole",
         "fields": [
            { "name": "Id" },
            { "name": "ParentRoleId" },
            { "name": "RollupDescription" },
            { "name": "OpportunityAccessForAccountOwner" },
            { "name": "CaseAccessForAccountOwner" },
            { "name": "ContactAccessForAccountOwner" },
            { "name": "ForecastUserId" },
            { "name": "MayForecastManagerShare" },
            { "name": "LastModifiedDate" },
            { "name": "LastModifiedById" },
            { "name": "SystemModstamp" },
            { "name": "DeveloperName" },
            { "name": "PortalAccountId" },
            { "name": "PortalType" },
            { "name": "PortalAccountOwnerId" }
         ]
      }
   },
   "Flatten_UserRole": {
      "action": "flatten",
      "parameters": {
         "multi_field": "Roles",
         "parent_field": "ParentRoleId",
         "path_field": "RolePath",
         "self_field": "Id",
         "source": "Extract_UserRole"
      }
   },
   "Augment_User_FlattenUserRole": {
      "action": "augment",
      "parameters": {
         "left": "Extract_User",
         "left_key": [
            "UserRoleId"
         ],
         "relationship": "Role",
         "right": "Flatten_UserRole",
         "right_key": [
            "Id"
         ],
         "right_select": [
            "Roles",
            "RolePath"
         ]
```

```
      }
   },
   "Augment_Opportunity_UserWithRoles": {
      "action": "augment",
      "parameters": {
         "left": "Extract_Opportunity",
         "left_key": [
            "OwnerId"
         ],
         "right": "Augment_User_FlattenUserRole",
         "relationship": "Owner",
         "right_select": [
            "Name",
            "Role.Roles",
            "Role.RolePath"
         ],
         "right_key": [
            "Id"
         ]
      }
   },
   "Register": {
      "action": "sfdcRegister",
      "parameters": {
         "alias": "OppRoles",
         "name": "OppRoles",
         "source": "Augment_Opportunity_UserWithRoles",
       "rowLevelSecurityFilter": "'Owner.Role.Roles' == \"$User.UserRoleId\" || 'OwnerId'
 == \"$User.Id\""
      }
   }
}
```

Note: In this example, the dataset has columns Owner.Role.Roles and OwnerId. A user can view the values of these columns for each record to which they have access.

Create the Dataset

Now that you have the final dataflow definition file, you can create the dataset.

Warning: If you wish to perform the steps in this sample implementation, verify that you have all required Salesforce objects and fields, and perform the steps in a non-production environment. Ensure that these changes do not impact other datasets that you already created. Also, always make a backup of the existing dataflow definition file before you make changes because you cannot retrieve old versions of the file.

To create the dataset, perform the following steps.

1. In Wave Analytics, click the gear icon ( ) and then select **Data Monitor** to open the data monitor.
   The Jobs View of the data monitor appears by default.
2. Select **Dataflow View**.
3. Click the actions list (1) for the dataflow and then select **Download** to download the existing dataflow definition file.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

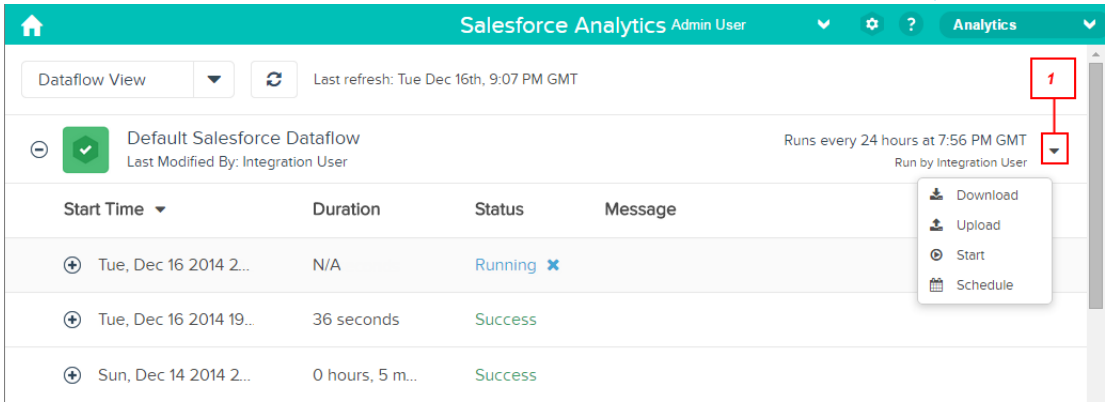
Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

To download, upload, run, and monitor a dataflow:

- "Edit Wave Analytics Dataflows"



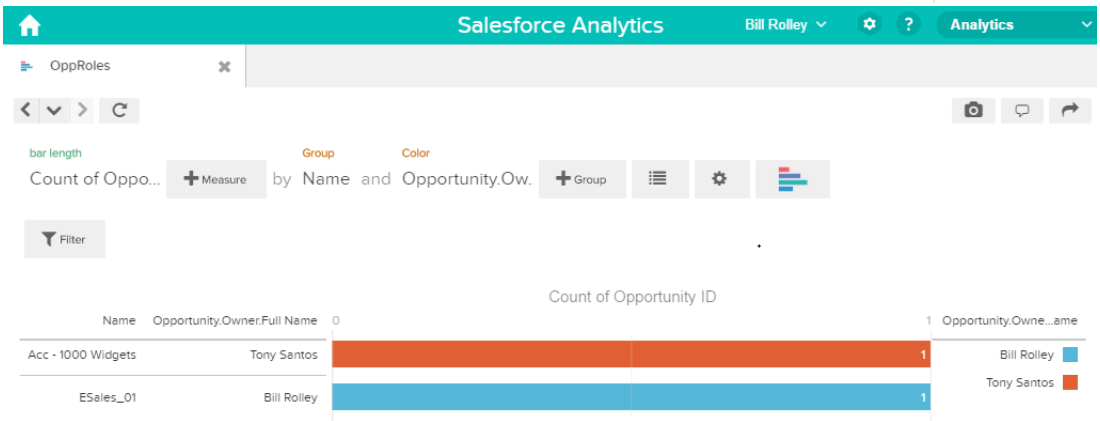
4. Open the dataflow definition file in a JSON or text editor.
5. Add the JSON determined in the previous step.
6. Before you save the dataflow definition file, use a JSON validation tool to verify that the JSON is valid.
   An error occurs if you try to upload the dataflow definition file with invalid JSON. You can find JSON validation tool on the internet.
7. Save and close the dataflow definition file.
8. In the Dataflow View of the data monitor, click the actions list for the dataflow and then select **Upload**.
9. Select the updated dataflow definition file and click **Upload**.
10. In the Dataflow View of the data monitor, click the actions list for the dataflow and then select **Run** to run the dataflow job.
11. Click the **Refresh Jobs** button ( ) to view the latest status of the dataflow job.
    You can view the OppRoles dataset after the dataflow job completes successfully.

Note: If you are adding a predicate to a dataset that was previously created, each user must log out and log back in for the predicate to take effect.

Test Row-Level Security for the Dataset

You must verify that the predicate is applied properly and that each user can see the appropriate opportunities.

1. Log in to Wave Analytics as Bill.
2. Open the OppRoles opportunity.
   Notice that Bill can't see his manager Keith's opportunities anymore. Now, he can see only his opportunity and his subordinate Tony's opportunity.



3. Log out and now log in as Keith.

   As expected, Keith can still see all opportunities.

EDITIONS

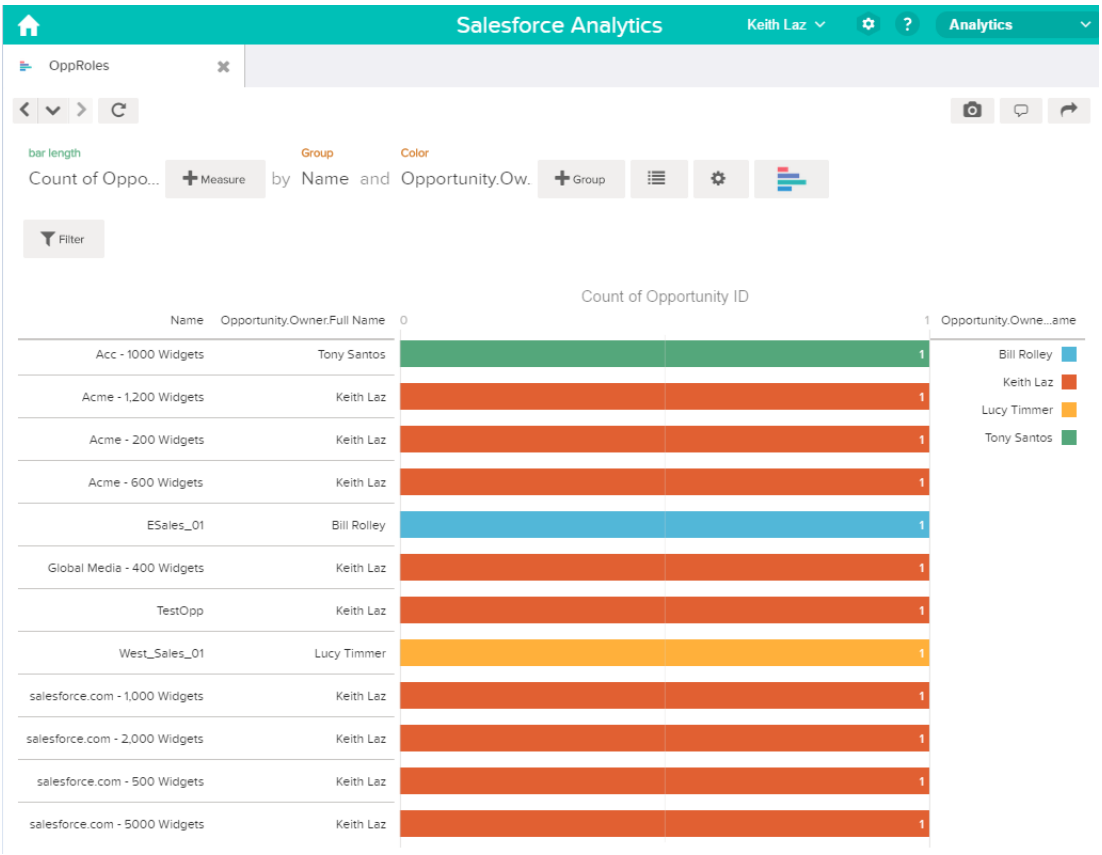
Available in: Salesforce Classic and Lightning Experience

Available in: **Developer** Edition

Available for an extra cost in: **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS

- ""
- ""


Salesforce Analytics
Keith Laz
Analytics
OppRoles
bar length
Count of Oppo...
Measure
by
Name
and
Group
Color
Opportunity.Ow.
Group
Filter
Count of Opportunity ID
Name
Opportunity.Owner.Full Name
0
1
Opportunity.Owne...ame
Acc - 1000 Widgets
Tony Santos
Acme - 1,200 Widgets
Keith Laz
Acme - 200 Widgets
Keith Laz
Acme - 600 Widgets
Keith Laz
ESales_01
Bill Rolley
Global Media - 400 Widgets
Keith Laz
TestOpp
Keith Laz
West_Sales_01
Lucy Timmer
salesforce.com - 1,000 Widgets
Keith Laz
salesforce.com - 2,000 Widgets
Keith Laz
salesforce.com - 500 Widgets
Keith Laz
salesforce.com - 5000 Widgets
Keith Laz
Bill Rolley
Keith Laz
Lucy Timmer
Tony Santos

SECURITY PREDICATE REFERENCE

Predicate Expression Syntax

You must use valid syntax when defining the predicate expression.

The predicate expression must have the following syntax:

```
<dataset column> <operator> <value>
```

For example, you can define the following predicate expression for a dataset:

```
'UserId' == "$User.Id"
```

You can create more complex predicate expressions such as:

```
(‘Expected_Revenue’ > 4000 || ‘Stage Name’ == "Closed Won") && ‘isDeleted’ != "False"
```

Consider the following requirements for the predicate expression:

- The expression is case-sensitive.
- The expression cannot exceed 1,000 characters.
- There must be at least one space between the dataset column and the operator, between the operator and the value, and before and after logical operators. This expression is not valid: `‘Revenue’>100`. It must have spaces like this: `‘Revenue’ > 100`.

If you try to apply a predicate to a dataset and the predicate is not valid, an error appears when any user tries to query the dataset.

IN THIS SECTION:

Dataset Columns in a Predicate Expression
You include at least one dataset column as part of the predicate expression.

Values in a Predicate Expression
The value in the predicate expression can be a string literal or number literal. It can also be a field value from the User object in Salesforce.

Escape Sequences
You can use the backslash character (\) to escape characters in column names and string values in a predicate expression.

Character Set Support
Wave Analytics supports UTF-8 characters in dataset column names and values in a predicate expression. Wave Analytics replaces non-UTF-8 characters with the UTF-8 symbol ( ). If Wave Analytics has to replace a non-UTF-8 character in a predicate expression, users may experience unexpected query results.

Special Characters
Certain characters have a special meaning in Wave Analytics.

Operators
You can use comparison operators and logical operators in predicate expressions.

Dataset Columns in a Predicate Expression

You include at least one dataset column as part of the predicate expression.

Consider the following requirements for dataset columns in a predicate expression:

- Column names are case-sensitive.
- Column names must be enclosed in single quotes ('). For example, `'Region' == "South"`

  Note: A set of characters in double quotes is treated as a string rather than a column name.

- Single quotes in column names must be escaped. For example, `'Team\'s Name' == "West Region Accounts"`

Values in a Predicate Expression

The value in the predicate expression can be a string literal or number literal. It can also be a field value from the User object in Salesforce.

Consider the following requirements for each value type.

Value Type	Requirements	Predicate Expression Examples
string literal	Enclose in double quotes and escape the double quotes.	• `'Owner' == "Amber"` • `'Stage Name' == "Closed Won"`
number literal	Can be a float or long datatype. Do not enclose in quotes.	• `'Expected_Revenue' >= 2000.00` • `'NetLoss' < -10000`
field value	When referencing a field from the User object, use the $User.[field] syntax. Use the API name for the field. You can specify standard or custom fields. When you define a predicate for a dataset, you must have read access on all User object fields used to create the predicate expression. However, when a user queries a dataset that has a predicate based on the User object, Wave Analytics uses the access permissions of the Insights Security User to evaluate the predicate expression based on the User object. Note: By default, the Security User does not have access permission on custom fields of the User object. To grant the Security User read access on a field, set field-level	• `'Owner.Role' == "$User.UserRoleId"` • `'GroupID' == "$User.UserGroupId__c"`

Value Type	Requirements	Predicate Expression Examples
	security on the field in the user profile of the Security User.	

Escape Sequences

You can use the backslash character (\) to escape characters in column names and string values in a predicate expression.

You can use the \' escape sequence to escape a single quote in a column name. For example:

```
'Team\'s Name' == "West Region Accounts"
```

You can use the following escape sequences for special characters in string values.

Sequence	Meaning
\b	One backspace character
\n	New line
\r	Carriage return
\t	Tab
\Z	CTRL+Z (ASCII 26)
\"	One double-quote character
\\	One backslash character
\0	One ASCII null character

Character Set Support

Wave Analytics supports UTF-8 characters in dataset column names and values in a predicate expression. Wave Analytics replaces non-UTF-8 characters with the UTF-8 symbol ( ). If Wave Analytics has to replace a non-UTF-8 character in a predicate expression, users may experience unexpected query results.

Special Characters

Certain characters have a special meaning in Wave Analytics.

Character	Name	Description
'	Single quote	Encloses a dataset column name in a predicate expression. Example predicate expression: `'Expected_Revenue' >= 2000.00`

Character	Name	Description
"	Double quote	Encloses a string value or field value in a predicate expression. Example predicate expression: `'OpportunityOwner' == "Michael Vesti"`
( )	Parentheses	Enforces the order in which to evaluate a predicate expression. Example predicate expression: `('Expected_Revenue' > 4000 || 'Stage Name' == "Closed Won") && 'isDeleted' != "False"`
$	Dollar sign	Identifies the Salesforce object in a predicate expression. Note: You can only use the User object in a predicate expression. Example predicate expression: `'Owner.Role' == "$User.UserRoleId"`
.	Period	Separates the object name and field name in a predicate expression. Example predicate expression: `'Owner' == "$User.UserId"`

Operators

You can use comparison operators and logical operators in predicate expressions.

IN THIS SECTION:

Comparison Operators
Comparison operators return true or false.
Logical Operators
Logical operators return true or false.

Comparison Operators

Comparison operators return true or false.

Wave Analytics supports the following comparison operators.

Operator	Name	Description
==	Equals	True if the operands are equal. String comparisons that use the equals operator are case-sensitive. Example predicate expressions: `'Stage Name' == "Closed Won"`
!=	Not equals	True if the operands are not equal. String comparisons that use the not equals operator are case-sensitive. Example predicate expression: `'isDeleted' != "False"`
<	Less than	True if the left operand is less than the right operand. Example predicate expression: `'Revenue' < 100`
<=	Less or equal	True if the left operand is less than or equal to the right operand.
>	Greater than	True if the left operand is greater than the right operand.
>=	Greater or equal	True if the left operand is greater than or equal to the right operand.
in	Multi-value list filter	True if the left operand exists in the list of strings substituted for a multi-value picklist (field value). Example predicate expression: `'Demog' in ["$User.Demographic__c"]` In this example, `Demographic__c` is of type `multiPicklistField`. During evaluation, the multi-value picklist field is substituted by a list of strings, with 1 string per user-selected item. Note: Comma-separated lists are not supported within the square-bracket construct.

You can use the <, <=, >, and >= operators with measure columns only.

Logical Operators

Logical operators return true or false.

Wave Analytics supports the following logical operators.

Operator	Name	Description
&&	Logical AND	True if both operands are true. Example predicate expression: 'Stage Name' == "Closed Won" && 'isDeleted' != "False"

Operator	Name	Description
||	Logical OR	True if either operand is true. Example predicate expression: 'Expected_Revenue' > 4000 || 'Stage Name' == "Closed Won"

Sample Predicate Expressions

Review the samples to see how to structure a predicate expression.

The samples are based on the following Opportunity dataset.

Opportunity	Expected_Rev	Owner	OwnerRoleID	Stage_Name	IsDeleted
OppA	2000.00	Bill	20	Prospecting	True
OppB	3000.00	Joe	22	Closed Won	False
OppC	1000.00	可爱的花	36	Closed Won	False
OppD	5000.00	O'Fallon	18	Prospecting	True
OppE		Joe	22	Closed Won	True

Let's take a look at some examples to understand how to construct a predicate expression.

Predicate Expression	Details
`'OwnerRoleID' == "$User.UserRoleId"`	Checks column values in the User object.
`'Expected_Rev' > 1000 && 'Expected_Rev' <= 3000`	
`'Owner' = "Joe" || 'Owner' = "Bill"`	
`('Expected_Rev' > 4000 || 'Stage Name' == "Closed Won") && 'isDeleted' != "False"`	Parentheses specify the order of operations.
`'Stage Name' == "Closed Won" && 'Expected_Rev' > 70000`	
`'Owner' == "可爱的花"`	String contains Unicode characters.
`'Owner' == "O\'Fallon"`	Single quote in a string requires the escape character.
`'Stage Name' == ""`	Checks for an empty string.