

Wave Analytics Data Integration

Salesforce, Spring '17





© Copyright 2000–2017 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
Datasets
Dataflow JSON
Dataset Builder
Wave Connector for Excel Data 6
Evternal Data API
CREATE DATASETS WITH A DATAFLOW
Design the Dataflow
Configure the Dataflow
Start and Stop a Dataflow
Monitor a Dataflow Job
Reschedule and Unschedule a Dataflow
DATAFLOW TRANSFORMATION REFERENCE
Transformations for Wave Dataflows
Overriding Metadata Generated by a Transformation
,
CREATE A DATASET WITH THE DATASET BUILDER
CREATE A DATASET WITH EXTERNAL DATA
Create d Dataset With External Data
EDIT A DATASET
DELETE A DATASET
Considerations when Defining a Predicate for a Dataset
Considerations when Defining a Predicate for a Dataset
Considerations when Defining a Predicate for a Dataset 97 Row-Level Security Example based on Record Ownership 97 Row-Level Security Example based on Opportunity Teams 102 Row-Level Security Example based on Role Hierarchy and Record Ownership 110
Considerations when Defining a Predicate for a Dataset 97 Row-Level Security Example based on Record Ownership 97 Row-Level Security Example based on Opportunity Teams 102 Row-Level Security Example based on Role Hierarchy and Record Ownership 110 Row-Level Security Example based on Territory Management 120
Considerations when Defining a Predicate for a Dataset97Row-Level Security Example based on Record Ownership97Row-Level Security Example based on Opportunity Teams102Row-Level Security Example based on Role Hierarchy and Record Ownership110Row-Level Security Example based on Territory Management120
Considerations when Defining a Predicate for a Dataset97Row-Level Security Example based on Record Ownership97Row-Level Security Example based on Opportunity Teams102Row-Level Security Example based on Role Hierarchy and Record Ownership110Row-Level Security Example based on Territory Management120SECURITY PREDICATE REFERENCE127
Considerations when Defining a Predicate for a Dataset97Row-Level Security Example based on Record Ownership97Row-Level Security Example based on Opportunity Teams102Row-Level Security Example based on Role Hierarchy and Record Ownership110Row-Level Security Example based on Territory Management120SECURITY PREDICATE REFERENCE127Predicate Expression Syntax for Datasets127

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.

	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

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

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.

Data Integration

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 External Data from the User Interface

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 3700000000000000. 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</date>	Text	Number of seconds. If the date contains no seconds, value is '0.'
<date field="" name="">_Minute</date>	Text	Number of minutes. If the date contains no minutes, value is '0.'
<date field="" name="">_Hour</date>	Text	Number of hours. If the date contains no hours, value is '0.'
<date field="" name="">_Day</date>	Text	Day of the month.
<date field="" name="">_Week</date>	Text	Week number in calendar year.
<date field="" name="">_Month</date>	Text	Month number in calendar year.
<date field="" name="">_Quarter</date>	Text	Quarter number in calendar year.
<date field="" name="">_Year</date>	Text	Calendar year.
<date field="" name="">_Week_Fiscal</date>	Text	Week number in fiscal year.
<date field="" name="">_Month_Fiscal</date>	Text	Month number in fiscal year.
<date field="" name="">_Quarter_Fiscal</date>	Text	Quarter number in fiscal year.

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

Important: 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 External Data from the User Interface

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 and Stop a 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 monitor in the data manager to monitor dataflow jobs to ensure that they complete successfully or to troubleshoot them if they fail.

5. Reschedule and Unschedule a Dataflow

After you run a dataflow job for the first time, it runs on a daily schedule, by default. You can change the time of the daily schedule or schedule it to run on an hourly interval. You might change the schedule to ensure that the data is available by a particular time or to run the job during non-business hours. You can also unschedule a dataflow.

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.

- In Wave Analytics, click the gear icon () and then click Data Manager. The data manager opens on the Monitor tab, with the Jobs view selected 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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

USER PERMISSIONS

To edit the dataflow definition file:

 "Edit Wave Analytics Dataflows"

Dataflow View 🔻			Last refresh: Today at 10:41 AM
Default Salesforce Dataflow Last Modified By: Admin User			Runs every 24 hours at 7:46 AM SalesEdgeEltWorkflow - Run by Integration User
	STATUS	MESSACE	🛃 Download
START HIVE V DURATION	STATUS	MESSAGE	📩 Upload
✤ Today at 10:41 AM N/A	Running ×		Start
			🔂 Schedule
Yesterday at 2:46 PM 0 hours, 2 minutes	Successful		Ø Unschedule

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 each transformation as a node in the dataflow definition file.

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

```
{
  "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 example JSON is the node name for a transformation. Each node contains an action value, 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.
- Important: Node names must be unique in a dataflow definition file, and can't contain space or tab characters. Consider that node names are **not** treated as case sensitive, so names such as "Extract_Opportunities" and "extract_opportunities" are not unique in the same definition file.
- 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 Monitor tab, 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.

If you have enabled replication, you can create multiple dataflows, in addition to the default dataflow, and configure them in the same way. Look out for the **Create** button in the Monitor tab of the data manager after you enable replication. See **#bi_integrate_replication__bi_integrate_replication**.

, Wave Analytics	
Dataflow View 🔻	Last refresh: Today at 11:09 AM C Create
Default Salesforce Dataflow Last Modified By: Admin User	Runs every 24 hours at 6:00 AM SalesEdgeEltWorkflow - Run by Integration User
Marketing Dataflow Last Modified By: Admin User	Runs every 24 hours at 6:00 AM
Product Support Dataflow Last Modified By: Admin User	Runs every 24 ho Product_Support_Dataflow - Run by Cupload
	Start
	O Unschedule
	💼 Delete

Note: Your org can have up to 10 dataflows. This includes the default dataflow, and Wave app dataflows such as Sales Wave and Service Wave. Keep in mind that each dataflow run counts towards your limit of 24 dataflow runs per day.

Start and Stop a 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.

 In Wave Analytics, click the gear icon () and then click Data Manager. The data manager opens on the Monitor tab, with the Jobs view selected by default.

2. Select Dataflow View.

EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

USER PERMISSIONS

To start a dataflow job:

 "Edit Wave Analytics Dataflows"

Dataflow View 🔻				Last refresh: Today at 10:41 AM
C C	rce Dataflow Imin User			Runs every 24 hours at 7:46 AM SalesEdgeEltWorkflow - Run by Integration User
START TIME	DURATION	STATUS	MESSAGE	⊥ Upload
 Today at 10:41 AM 	N/A	Running ×		▶ Start
				🔂 Schedule
 Yesterday at 2:46 PM 	0 hours, 2 minutes	Successful		Ø Unschedule

- 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 grayed out while the dataflow job runs.
- 4. After the job completes, Wave Analytics sends an email notification to the user who created the dataflow.

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.

Note: If the dataflow creator is not an active user, the notification is sent to the user who last modified the dataflow schedule or definition file.

To stop a dataflow job that is currently running, click next to the job status.
 If you click Start to restart a stopped dataflow, the job starts over—the dataflow job does not resume from the point at which it was stopped.

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.

Monitor a Dataflow Job

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

The Dataflow view of the 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, view the run-time details about every transformation that is processed, and download error logs where available.

EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

USER PERMISSIONS

To access the data monitor:

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

To download an error log:

 "Edit Wave Analytics Dataflows" and "View All Data"

Dataflow View 🔻	1				Last refresh	: Today at 1:04 PM	G
Default Salesford Last Modified By: Jon (Datham				Runs every 8 hours - nex SalesEdgeEltWorkflow - Run	t run at 5:00 PM by Integration User	•
START TIME 🔻	DURATION	STATUS	MESSAGE				
 Today at 9:00 AM 	16 seconds	Successful					
⊖ Today at 1:00 AM	27 seconds	Warning	The job completed sucessfu monitor to view the failed re	illy, but some rows faile ows.	d. Download the error log fron	n the data	
2							
NODE NAME	START TIME	I	DURATION	STATUS	NODE TYPE	INPUT ROWS	
salesData	Today at 1:0	0 AM	22 seconds	Successful	sfdcDigest	706	
Derived_Fields	Today at 1:0	0 AM	0 seconds	Warning	computeExpression	706	
optimize-Register_Catego	Today at 1:0	0 AM	0 seconds	Successful	optimizer	N/A	
Register_CategorizedSales	Today at 1:0	0 AM	3 seconds	Successful	sfdcRegister	N/A	
4						Þ	

1. In Wave Analytics, click the gear icon (🔯) and then click **Data Manager**.

The data manager opens on the Monitor tab, with the Jobs view selected by default.

2. Select Dataflow View (1).

3. Click cli

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.
Warning	The job completed successfully, but some rows failed.

- 4. If the dataflow job fails, expand the job node (2) and review the run-time details for every transformation that was processed.
- 5. If an error log is available for a node, click the download log button (3) to download a CSV file containing the failed rows.
 - Note: Error logs display the data from rows that have failed to load. To maintain data security and prevent unauthorized access to this data, only users with the "View All Data" permission can download error logs.
- 6. If there's a problem with the dataflow logic, edit the dataflow definition file, upload it, and then run the dataflow again.

Reschedule and Unschedule a Dataflow

After you run a dataflow job for the first time, it runs on a daily schedule, by default. You can change the time of the daily schedule or schedule it to run on an hourly interval. You might change the schedule to ensure that the data is available by a particular time or to run the job during non-business hours. You can also unschedule a dataflow.

- 1. In Wave Analytics, click the gear icon (2) and then click **Data Manager**. The data manager opens on the Monitor tab, with the Jobs view selected by default.
- 2. Select Dataflow View.

EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

USER PERMISSIONS

To schedule or unschedule a dataflow job:

 "Edit Wave Analytics Dataflows"

Datanow View	•				Last refresh: Tot	1 1 10.41 AW
⊖ 🔗 Default S Last Modifie	alesfor d By: Adn	ce Dataflow hin User			Runs every 24 H SalesEdgeEltWorkflow - Run	by Integration User
START TIME	•	DURATION	STATUS	MESSAGE		1 Upload
€ Today at 10:41 AM		N/A	Running ×			Start
						📅 Schedule
Yesterday at 2:46 PM	Λ	0 hours, 2 minutes	Successful			Ø Unschedul

- **3.** Click **Schedule** in the actions list (1) to reschedule the dataflow job. The default daily schedule and start time appears.
- 4. To continue running it once per day, select the time that you'd like to start the dataflow job.
- 5. To run it on an hourly basis, select **Hour(s)** and then specify the interval details. Minute offset is the number of minutes past the hour to run the dataflow.

×				Dataflow Schedule	Da
) ~	Hour(s)	~	1	Run every	
	~	\ \	0:10	Minute offset	
Course					
Save			0:10	Minute offset	

Hourly scheduled dataflows run relative to a fixed base time (00:00 UTC). For example, if you schedule a dataflow to run every 6 hours with a 15-minute offset, the dataflow is scheduled to run at 00:15 UTC, 06:15 UTC, 12:15 UTC, and 18:15 UTC, each day. The first dataflow run occurs at the next scheduled time. So, if you change the schedule at 04:00 UTC, the dataflow runs at the next scheduled time (06:15 UTC), and then every 6 hours thereafter. To keep things simple, the scheduler specifies the time of the first dataflow run (in your time zone), after you save the schedule.

6. Click Save.

The Dataflow Schedule pop-up box specifies the next time, in your local time zone, that the dataflow runs.

- 7. Click Done.
- 8. To stop a dataflow from running on a schedule, select Unschedule in the action list (1).

DATAFLOW TRANSFORMATION REFERENCE

Transformations for Wave Dataflows

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 its 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.

Opportunity_Account	
Id — Name Amount StageName CloseDate AccountId OwnerId —	Opportunity Columns
OpptyAcct.Id	Account Columns

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		Price			
d	Name	rodid	me	Pricebook	UnitP
1	Table	od1	le	Standard	1
	Chair	od2	air -	Standard	
	Bench	ıll	nch	Custom	
2		od3		Standard	

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": <i>null</i> ,	"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	
	Name
Tabl	e
Cha	ir
Ber	ıch

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				Price	
N	ame	P	rodid	Pricebook	UnitPrice
Tab	le	P	rod1	Standard	1000
Cha	ir	P	rod2	Standard	450
Cha	air	P	rod3	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", "Nam	e":"Table", "Price.Pri	cebook":"Standard", "Pr	rice.UnitPrice":1000 },
{ "id":"Prod2", "Nam	e":"Chair", "Price.Pri	cebook":"Standard", "Pr	rice.UnitPrice":450 },
{ "id":"Prod2", "Nam	e":"Chair", "Price.Pri	cebook":"Standard", "Pr	rice.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.

ю	duct			Price	
N	ame		Prodid	Pricebook	UnitPrice
Tab	le		Prod4	Standard	1000
Chai	r		Prod5	Standard	450
Bend	ch				

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": <i>null</i> ,	"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			Price		
	Name		Prodid	Pricebook	UnitPrice
rod1	Table		Prod1	Standard	100
od2	Chair		Prod2	Standard	45
rod3	Bench		Prod3	Custom	80
			Prod3	Standard	70

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
rod2	Chair
Prod3	Bench

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	<u>}</u>	
{ "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 augment the datasets. If you use a composite key, the left and right key have the same number of columns same order. For a composite key, us 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",
               "saglExpression": "case when FinalPrice < 1000 then \"Low\" when
FinalPrice >= 1000 and FinalPrice < 2000 then \"Medium\" else \"High\" end"}
         1
      }
   },
   "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	
		"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 a source 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.



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 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": "computeTrending"
      }
   }
}
```

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	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 partition-by field only.
orderBy	Yes	<pre>Field used to sort the records within each partition and the sort order: ascending (asc) or descending (desc). Specify one sort field only. Example: "orderBy": [{</pre>
computedFields	Yes	<pre>A list of definitions for derived fields. Example:</pre>

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.

Parameter	Required?	Value
		Note: If the name is not unique, the dataflow fails to run.
expression	sion Yes Expression attributes used to calculate the value expression can be based on input fields or other transformation. Example:	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:
		<pre>"expression": { "sourceField": "CloseDate", "offset": "previous()", "default": "01/01/1970" See expression.</pre>

expression

The following table describes the attributes in the expression section.

Parameter	Required?	Value	
sourceField	Yes	API name of the input field from the source node 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 ()	
		for an opportunity.	
		Example:	
		"offset": "first()"	
		next()	
		Gets the value from the next record.	
		Example:	
		"offset": "next()"	
		previous()	
		Gets the value from the previous record.	
		Example:	
		"offset": "previous()"	

Parameter	Required?	Value
		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"
		<pre>"default": "current()"</pre>

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</date_column_name>	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).
<date_column_name>_day_epoch</date_column_name>	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 (Opportunityld) 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

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

After the records are sorted, for each dimension (Opportunityld), 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.

Parameter	Required?	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"
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 its 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

Table 1. FinalSales Dataset

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.

Filter Expression Syntax	Description
	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.
	<pre>Example: "filter": "LastName:R::Levy"</pre>
dim:N:val	True if the dimension and value are not equal.
	<pre>Example: "filter": "RoleName:N:Manager"</pre>
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.
	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.

Dataflow Transformation Reference

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 hierarchy node, which we refer to as the "self ID." Each record contains two generated columns that store the hierarchy for each self ID node. One column contains a comma-separated list of all ancestors for each node in the hierarchy. The other column contains the hierarchy path.

Role ID (Self 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

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

You can also configure the flatten transformation to include the self ID node in the generated hierarchy columns. The following dataset shows the self ID in bold.

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

Example: Let's look at an 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, 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",
      "include self field":false
   }
},
"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

ParentRoleld 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
include_self_id	No	Indicates whether to include the self ID node in the generated multi_field and path_field columns. Valid values are false (default) and true.
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"
   }
```

}

Considerations When Using the sfdcDigest Transformation

- Consider dataset storage limits when extracting data. For example, a dataset can contain a maximum of 5,000 fields, so be selective when choosing fields. See Wave Analytics Limits.
- The sfdcDigest transformation runs a SOQL query to extract data from a Salesforce object, and so is subject to SOQL limits. If the query exceeds any of these limits, it may return no results or cause the dataflow job to fail. For example, The length of the SOQL query cannot exceed 20,000 characters. To reduce the SOQL query length, consider breaking up the extract into two or more sfdcDigest transformations and then use the augment transformation to combine the results. For example, you might create one sfdcDigest transformation to extract half of the fields and create another sfdcDigest transformation to extract the remaining fields. See SOQL and SOSL Limits.

IN THIS SECTION:

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 in Wave

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.

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:	
	<pre>"filterConditions": [{ "field": "OwnerId", "operator": "=", "value": "a07B00000012HYu" }]</pre>	

Operator	Comment
!=	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):
	<pre>"filterConditions": [{ "field": "Nickname_c", "operator": "!=", "value": "\"Sammy\"" }]</pre>
>	Filter condition is true if the value in the field is greater than 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 the specified value.
	Example (using a date literal):
	<pre>"filterConditions": [{ "field": "CloseDate", "operator": "<", "value": "THIS_MONTH", "isQuoted": false }]</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>

Operator	Comment	
	}	
<=	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>	
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:	
	"filterConditions": [{ "field": "StageName".	

```
"field": "StageName",
"operator": "IN",
```

Operator	Comment	
	<pre>"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.	
	<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", "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.

{

```
{ "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": "Id" },
            { "name": "Name" },
            { "name": "AccountId" },
            { "name": "StageName" },
            { "name": "CloseDate" },
            { "name": "OwnerId" },
            { "name": "FiscalYear" },
            { "name": "FiscalYear" },
            { "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

0

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
            }
        1
      }
   },
   "Register Opportunities Dataset":{
     "action":"sfdcRegister",
      "parameters":{
         "alias":"Opportunities",
         "name": "Opportunities",
         "source":"Extract Opportunities"
     }
   }
}
```

SEE ALSO: sfdcDigest Transformation

Unsupported Salesforce Objects and Fields in Wave

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.

- ActivityMetric
- ApexEmailNotification
- AuthProvider
- BrandTemplate
- ChatterConversation
- ChatterConversationMember
- ChatterMessage
- ConnectedApplication
- ContentFolderLink
- ContentWorkspace
- ContentWorkspaceDoc
- ContentWorkspaceMember
- ContentWorkspacePermission
- CopyExport
- CorsWhitelistEntry
- DataAssessmentFieldMetric
- DataAssessmentMetric
- DataAssessmentValueMetric
- DataHubSetupData
- DataHubSetupDefinition
- DirectMessage
- DirectMessageFeed
- DirectMessageMember
- EmailCapture
- EmailDomainKey
- EmailServicesAddress
- EmailServicesFunction
- EmailTemplate
- EnvironmentHub
- EnvironmentHubInvitation
- EnvironmentHubMemberRel
- EventType
- EventTypeParameter
- FeedPollChoice
- FeedPollVote
- KnowledgeArticleVersion
- LeadChangeEvent
- LoginGeo

Dataflow Transformation Reference

- LoginHistory
- ModelFactor
- NetworkActivityAudit
- NetworkModeration
- OrganizationProperty
- OrgWideEmailAddress
- PackageLicense
- PartnerNetworkSyncLog
- ReputationLevel
- ReputationLevelLocalization
- ReputationPointsRule
- SalesforcelqUser
- SampledEntity
- SandOmInfo
- SandOmInfoDetail
- SandOmObserver
- ScoreIntelligence
- SearchPromotionRule
- SecurityCustomBaseline
- SelfServiceUser
- SsoUserMapping
- TenantSecret
- TwoFactorInfo
- TwoFactorTempCode
- UserPackageLicense
- UserProvAccount
- UserProvAccountStaging
- UserProvisioningConfig
- UserProvisioningLog
- UserProvisioningRequest
- UserProvisioningRequestOwnerSharingRule
- UserProvisioningRequestShare
- UserProvMockTarget
- UserRecordAccess
- VerificationHistory
- VoiceUserLine
- VoiceUserLineOwnerSharingRule
- VoiceUserLineShare
- VoiceVendorLine
- VoiceVendorLineOwnerSharingRule

- VoiceVendorLineShare
- WebLink
- WebLinkLocalization

Note: The sfdcDigest transformation doesn't support extracts from BigObjects.

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 can't extract data from these fields.

Object	Unsupported Fields	
Account	CleanStatus	
ActionPlanItem	ItemId	
AuthSession	LoginGeoldLoginHistoryId	
CaseArticle	KnowledgeArticleId	
Contact	CanAllowPortalSelfRegCleanStatus	
ContentDocument	ParentId	
CustomPersonp	Title	
DocumentAttachmentMap	Parentld	
EmailMessage	ActivityId	
EmailRoutingAddress	EmailServicesAddressId	
EnvironmentHubMember	EnvironmentHubld	
ExternalEventMapping	EventId	
InstalledMobileApp	ConnectedApplicationId	
Lead	CleanStatus	
KnowledgeArticle	MasterLanguage	
Network	 CaseCommentEmailTemplateId ChangePasswordEmailTemplateId ForgotPasswordEmailTemplateId WelcomeEmailTemplateId 	

Object	Unsupported Fields	
Organization	 SelfServiceEmailUserOnCaseCreationTemplateld SelfServiceNewCommentTemplateld SelfServiceNewPassTemplateld SelfServiceNewUserTemplateld WebToCaseAssignedEmailTemplateld WebToCaseCreatedEmailTemplateld WebToCaseEmailTemplateld WebToLeadEmailTemplateld 	
PermissionSet	PermissionsEditEventPermissionsEditTask	
PermissionSetLicense	MaximumPermissionsEditEventMaximumPermissionsEditTask	
Profile	PermissionsEditEventPermissionsEditTask	
ThirdPartyAccountLink	SsoProviderId	
User	LastPasswordChangeDateUserPreferencesEnableVoicePilot	
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.
		Note: Incremental extraction is available if you have enabled replication.
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.
		Note: Incremental extraction is available if you have enabled replication.

Parameter	Required?	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. 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.

Attribute	Required?	Value
		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 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

Attribute	Required?	Value
		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
		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:
		"nrecision" 10
		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.

Attribute	Required?	Value
		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:	
		<pre>{ "field": "<field name="">", "operator": "<operator>", "value": "<value>", "isQuoted": true false}</value></operator></field></pre>
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.	
		<pre>"filterConditions": [{ "operator": "OR", "conditions": [{ "field": "StageName", "operator": "=", "value": "Closed Won" },</pre>
		{
Parameter	Required?	Value
------------	-----------	---
		<pre>"field": "Probability",</pre>
value	No	The value used in a filter condition.
isQuoted	No	Indicates whether you quoted the string value in a filter condition. Example with quoted values:
		<pre>"filterConditions": [{ "field": "StageName", "operator": "IN", "value": "('Closed Won', 'Closed Lost')", "isQuoted": true }]</pre>
		Example with non-quoted values:
		<pre>"filterConditions": [{ "field": "StageName", "operator": "IN", "value": ["Closed Won", "Closed Lost"], "isQuoted": false }]</pre>
		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

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 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.
		"'Ownerld' == "\$User.ld""
		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.

Accounts		UpdatedAccountNames
iu	Keys	Accountib
Name 🚽	Replace	 NewAccountName
AnnualRevenue	Replace	

You create the following dataflow definition file.

```
{
   "Extract AccountDetails": {
      "action": "sfdcDigest",
      "parameters": {
         "object": "Account",
         "fields": [
            { "name": "Id" },
            { "name": "Name" },
            { "name": "AnnualRevenue" }
         1
      }
   },
   "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
		If you do not specify this parameter, the transformation updates the left dataset

Parameter	Required?	Value
		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 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": [
                ł
                   "name": "Amount",
                   "label": "Opportunity Amount"
                }
            1
         }
      1
   }
}
```

Object Attributes

You can override the following object attributes.

Object Attribute	Туре	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	Туре	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:

Field Attribute	Туре	Description	
		"isSystemField": false	
format	String	The display format of the numeric value.	
		Examples:	
		"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 1 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
ld 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.

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.

📀 Dataset Builder 🗙	
Q. Pick an sObject to start.	×
Name 🔺	Type
Accepted Event Relation	Std
Account	Std
Account Contact Role	Std
Account Feed	Std
Account History	Std
Account Partner	Std
Account Share	Std
Additional Directory Number	Std
Announcement	Std
Apex Class	Std
Apex Debug Log	Std
Apex Job	Std
Apex Test Queue Item	Std
Apex Test Result	Std 🗸

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"

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.

Create a Dataset with the Dataset Builder

Or Ro	pportunity ot	× ×
Se	lect Fields	FIELDS RELATIONSHIPS
Q	Search by name or metadata.	ж
~	Name ∧	Туре
~	Amount	currency
~	Close Date	date
~	Closed	boolean
×	Created Date	datetime
~	Deleted	boolean
~	Description	textarea
~	Division ID	picklist
~	Fiscal Period	string
~	Fiscal Quarter	int
~	Fiscal Year	int
~	Forecast Category	picklist
~	Forecast Category	picklist
~	Has Line Item	boolean
~	Is Locked	boolean 🗸

Mote: 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.

				Account Account ID
Opportuni loot	ity (12)	~	×	User Created By ID
elect Rel	lated Objects	FIELDS (9) REL	ATIONSHIPS (3)	
Q Searc	h by name or m	etadata.	×	User Owner ID
Q Searc	h by name or me	etadata. Related Object	× Type	User Owner ID
Q Searc	th by name or more the second se	etadata. Related Object Account	X Type Std	User Owner ID
Q Searc Action R ELETE A ELETE C	th by name or me Related Field A	Related Object Account User	X Type Std Std	User Owner ID
Q Searc	th by name or material field Account ID Created By ID ast Modified By	Related Object Account User User	¥ Type Std Std Std	User Owner ID
Q Searc Action R ELETE A ELETE C JOIN Li ELETE C	th by name or mo telated Field A Account ID Created By ID ast Modified By Dwner ID	Related Object Account User User User	X Type Std Std Std Std	User Owner ID

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.

^	Salesforce Analytics Cindy Cheng	Analytics	~
🔁 Dataset Builder 🗙			
		Create Dataset	
/	Account (1) Account ID	User Owner ID	v
Case (3) Root	Contact Contact ID		
\ \	Group Owner ID		

- 8. For each related object, select the fields from which you want to extract data.
- 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.

🜔 Dataset Builder 🛛 🗙		
(t)		Create Dataset
Selected Objects		
Account (1) (Account ID) User (Owner ID)		
Contact (Contact ID) Group (Owner ID)	Account (1) Account ID	User owner iD
Case (3) Root	Contact Contact ID	
	Group	
	Owner ID	

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 Wav Wave Ar	The Wave Connector app gives users a fast, easy way to import data from Excel 2013 into Salesforce USER PERMISSIO Wave Analytics						SSIONS						
	,											To import date 2013 to Wave Upload Ex Wave And	a from Excel Analytics : tternal Data to Ilytics
PivetChart *	Line Column Sparkline	Win/ Loss	Slicer Timeline Filters	Hyperlink	Test Box	Header W 8. Footer	A FordArt Sig	gnature Objec Line *	τ Equatio Sym	Ω n Symbol nboli		•	~
													~
N	0	P	0	R	s	т	U	v	w	X			- *
ectF Oppor	tunityStatus										Salesforce Wa	ve Connector to	* *
mer Open (Oppty												
ect Open (Oppty										Wave	connector	
ect Prospe	act										sunitiku	emetech.com *	
ect Open (Oppty												
mer Open (Oppty										Cat Started	0	^
mer Custon	mer No Oppty										Get Started	9	
mer Open (Oppty												
mer Open (Oppty										The Wave Connector	makes it easy to create	
mer Custor	mer No Oppty										datasets for Salesford	e Analytics Cloud using data select data in Excel, name	
mer Open (ορρτγ										your dataset, and clic	k Submit Data. The data you	
ect Prospe	oct Operty										selected will be instar	ntly imported to Salesforce.	
ert Open (Oppty										Name Your Dataset		
ect Open (Oppty										EventMarketingLead	rackin	
mer Open (Oppty												
mer Custor	ner No Oppty												
ect Open (Oppty										Data Selection		
ect Open (Oppty										Columns: 15 Rows: 1048576		
ect Open (Oppty										101131 20100110		
ect Prospe	ect										Title	Data Tana	
mer Open (Oppty										The	Cona Type	
mer Custon	ner No Oppty										Land Count	Numeric A	
mer Open (Oppty										Lean Count	Numeric 24	
mer Custor	ner No Oppty										Compaign ID18	Text	
mer Custon	ner No Oppty										Companying	1644	
mer Onen (Oppty Oppty										Sheet Date	Date: A	
mer Custon	ner No Conty										attar Lorena	Cate 23	
mer Custon	ner No Oppty										Product	Text	
mer Custon	ner No Oppty										FIGURE	- EM	
mer Custon	ner No Oppty										Location	Text	
mer Open (Oppty										CONTRACT.	1644	_
mer Custor	mer No Oppty										Load Strategy	Text	
mer Custor	mer No Oppty										CHARLES CHARCE	1845	
mer Open (Oppty										Frond Address	Text	
mer Custor	ner No Oppty										Citra Peruaness	1EM	
mer Open (Oppty										Account ID	Text	
ect Open (Oppty										ACCOUNTED	1626	
mer Custor	ner No Oppty										6.000	Test	
mer Custor	ner No Oppty										Customer	Lext	~
ect Prospe	ect No Onetri												
mer Once /	Onety											Submit Data	
and open	- thui												

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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

USER PERMISSIONS

To upload external data:

• "Upload External Data to Wave Analytics"

↑				Salesforce Analytics Admin User	× 0	? Analytics	~
Dataset	×						
				Add External Data File (CSV) (Max: 500 MB)		
1				ExternalData_Targets.csv		×	
				Add Metadata File (JSON)			
				ExternalData_Targets.json		*	
				📩 Select file or drag fil	le here		
Data	aset Name			Pre	view Data	Create Dataset	
Sale	es Targets 20)15					
Арр							
Sha	red App		~				

- 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.

ft	Salesfo	rce Analytics Admin User	✓ 🗢 ? Analytics ✓
Dataset 🗙			
CSV File: ExternalData_Tar	gets.csv Metadata File: ExternalData_T	argets.json	Cancel
A AccountOwner	A Region	I Target	🛗 TargetDate
Fony Santos	Midwest	10000	1/1/2011
ucy Timmer	Northeast	50000	1/1/2011
ucy 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.



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



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

Region	∧ Target	▲ TargetDate
Data type	Data type	Data type
	DATE MEASURE DIMENSION	DATE MEASURE DIMENSION
	Scale O 0 Precision O 18 Default Value 0 Apply to All Measure Columns	Format dd/MM/yyyy 🖍

- 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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

USER PERMISSIONS

To access the data monitor:

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

^	Sale	sforce Analytic	SAdmin User	 • • • • 	Analytics 🗸 🗸
Jobs View	Last refresh: Today at 8:42	AM	2	File Uploads	Show Hide
Dataflow	Start Time 🔹	Duration	Status	Message	
⊕ Default Salesf	Today at 7:39 AM	24 seconds	Successful		
➔ Default Salesf	Yesterday at 7:39 AM	0 hours, 10	Failed	Queue wa	it time exceeded limi
Sales_Targets	2 Days Ago at 9:34 AM	0 hours, 1	Failed	Error exect	uting node load [roo [.]
3 Node Name	Start Time	Duratio	n	Node Type	Status
load	2 Days Ago at 9:34 AM	0 hours,	1 minute	sfdcFetch	error
digest	N/A	N/A		csvDigest	pending
optimize-register	r N/A	N/A		optimizer	pending
register	N/A	N/A		sfdcRegister	pending
4					•

In Wave Analytics, click the gear icon () and then click **Data Manager**. The data manager opens on the Monitor tab, with the Jobs view selected by default. The Jobs view displays dataflow and upload jobs. It displays each upload job name as <dataset name upload flow>. You can hover a job to view the entire name.



1.

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 (C). 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.

C Wave An	alytics	My Private App	×				
Jobs View	•	2 Last refresh: To	day at 2:29 PM			File Uploads	Show Hide
Dataflow	I	Start Time	•	Duration	Status	Message	
⊖ Temp	Upload	Today at 2:26 PI	N	0 hours, 2	Warning	The job com	pleted sucessfully, b
Status	Input F	Rows Processed	Input Row	s Failed	Output Rows Pro	cessed Ou	tput Rows Failed
success	N/A		N/A		N/A	N/#	A1
warning	N/A		N/A		11	1	₹
success	N/A		N/A		N/A	N/#	4
success	N/A		N/A		N/A	N/#	Ą
4							

The error log contains a list of failed rows.

1	ExternalFileWorkflow06V4000000005mEAA_digest.csv						
		А	В	С	D	E	F
	1	row	error	File_Name	Page_Views	View_Date	Authenticated
	2	7	(column: Page_Views) strconv.ParseFloat: parsing "Text": invalid syntax	about:blank	Text	5/1/2015	No
	3						
	4						

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

EDIT A DATASET

USER PERMISSIONS

To view a dataset edit page:	"Use Wave Analytics" AND Editor access to the dataset's app
To update a dataset name, app, and extended metadata:	"Use Wave Analytics" AND Editor access to the dataset's app
To delete a dataset:	"Use Wave Analytics" AND Editor access to the dataset's app
To upload and preview data:	"Upload External Data to Wave Analytics" AND Editor access to the dataset's app
To edit a dataset security predicate:	"Edit Wave Analytics Dataflows"

EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

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.
Арр	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 <i>Extended Metadata (XMD) Reference</i> 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.

Option	Description
	Refer to <i>External Data Format Reference</i> 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 <i>External Data Format Reference</i> for information about metadata files.
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.

ft i i i i i i i i i i i i i i i i i i i	Salesforce Analy	tics Admin User 🛛 👻	Analytics
Dataset 🗙			
CSV File: ExternalData_Targets.csv	Metadata File: ExternalData_Targets.json		Cancel Submit
A AccountOwner	A Region	III 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

- 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.

A Regi	on		∧ Targ	et		▲ Targ	jetDate	
Data type			Data type			Data type		
DATE	MEASURE	dimension		MEASURE			MEASURE	
			Scale @ 0 Precision @ 18			Format dd/MM/yyyy	/ pply to All Date Co	v olumns
			Default Vale	ue Iy to All Measure C	Columns			

- 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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

USER PERMISSIONS

To delete a dataset:

• "Use Wave Analytics" AND Editor access to the dataset's app

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

B 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.

Row-Level Security Example Based on Territory Management

Let's look at an example where you create a dataset based on Salesforce data and then implement row-level security based on your defined territories. In this example, you determine what model you use for territory management, so you can later review sample JSON for that dataset. To restrict access on each record in the dataset, you will create a security predicate where each user can view only data appropriate for the territory to which they belong.

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.

- 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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

USER PERMISSIONS

To upload a CSV and metadata file:

 "Upload External Data to Wave Analytics"

^	Salesforce Analytics Manuel Castro 🗸 🌻 ? Analytics	
Dataset 🗙		
	Add CSV or .gz or ZIP File (Max: 500 MB)	
	⊥ Select File	
	Add JSON Schema File	
	1 Select File	
	Back Create Dataset	
Dataset Name		
Enter Dataset Name		
Арр		
My Private App 🛛 🗸		

- 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:

Click the gear icon (i) 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.
- 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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

≜		Salesforce Analytics					Keith Laz 🗸	• ?	Analytics	
SalesTargets	×									
< > > C									0 9	*
bar length		Group	Color							
Count of Rows	Heasure	by Account Owne and	Region	+ Group	≣	¢	- 1			
T Filter										
			Co	ount of Rows						
Account Owner F	Region 0								1	Region
Keith Laz Sour	thwest								Southw	/est

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.



EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

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

(<u>L</u>	All Oppor	rtunities	▼ Edit Delete Cre	eate New View			📰 List 👍 Feed	₿ 🥹
	Ne	ew Opportunity	0		A B C D E	F G H I J K L N	1 N O P Q R S	T U V W X Y Z 0	Other All
C		Action	Opportunity Name ↑	Account Name	Amount	Close Date	Stage	Opportunity Owner Alias	
0		Edit Del 🕀	Acc - 1000 Widgets	Acc salesrep		9/4/2014	Prospecting	<u>Tony</u>	
0		Edit Del 🕀	Acme - 1,200 Widgets	Acme	\$140,000.00	6/14/2012	Value Proposition	<u>Keith</u>	
0		Edit Del 🕀	Acme - 200 Widgets	Acme	\$20,000.00	10/13/2012	Prospecting	Keith	
0		Edit Del 🕀	Acme - 600 Widgets	Acme	\$70,000.00	8/10/2012	Needs Analysis	Keith	
0		Edit Del 🕀	ESales 01	East Sales acc 01		9/4/2014	Prospecting	Bill	
0		Edit Del 🕀	Global Media - 400	Global Media	\$40,000.00	7/13/2012	Id. Decision Makers	Keith	
0		Edit Del 🕀	salesforce.com - 1,	salesforce.com	\$100,000.00	6/14/2012	Negotiation/Review	Keith	
0		Edit Del 🕀	salesforce.com - 2	salesforce.com	\$20,000.00	8/12/2012	Value Proposition	Keith	
0		Edit Del 🕀	salesforce.com - 50	Global Media	\$50,000.00	5/12/2012	Closed Won	Keith	
0		Edit Del 🕀	salesforce.com - 50	Global Media	\$500,000.00	5/12/2012	Closed Won	Keith	
0		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.

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.



EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

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

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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

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

This predicate compares the Userld column in the dataset against the ID of the user running a query against the dataset. The Userld 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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

```
{
    "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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

USER PERMISSIONS

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

 "Edit Wave Analytics Dataflows"

Dataflow View	•				Last refresh: Today at 10:41 AM
⊖ 💉 Default Sa Last Modified	lesford By: Adm	ce Dataflow hin User			Runs every 24 hours at 7:46 AM
START TIME	•	DURATION	STATUS	MESSAGE	± Download ↑ Upload
 Today at 10:41 AM 		N/A	Running ×		Start
					📅 Schedule
Yesterday at 2:46 PM 0 hours, 2 minutes		Successful		Ø Unschedule	

- 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 (C) 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.
- 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.



Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

^		S	alesforc	e Analy	tics	Lucy Tin	nmer 🗸	۲	?	Analytic	:s	
► OppTeamMember												
< v > C										Ō	Ģ	*
bar length		Group										
Count of Opportu	+ Measure by	Opportunity	Team	+ Group	≣	¢	-					
Filter												
						•						

No results meet those criteria. Try changing your filters.

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.

^		Salesfo	rce Analytics	Bill Rolley $$	• ?	Analytics	×
OppTeamMember	×						
< > > C						¢ 0	1
bar length		Group					
Count of Opportu	🕂 Measur	by Opportunity Team	+ Group	۰ 📥			
T Filter							
		Count	of Opportunity ID				
Opportunity Team Member Name	• 0				1		
Bill Rolley					1		

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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.



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

All Oppor	🥹 All Opportunities 🔻 Edit Delete Create New View 🗰 List 🗼 Feed 🖨 🥹										
New Opportunity	0	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 1	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.



EDITIONS

Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

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

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.

A			Salest	force A	Analyti	cs	Bill Rolley 🗸	• ?	Analytics	~
OppRoles	×									
< v > C									0 9	*
bar length		Group								
Count of Opportu	Measure	by Name	+ Group	:=	¢	-				
T Filter										
			Co	unt of Op	portunity	r ID				
Name	0								1	
Acme - 1,200 Widgets									1	
Acme - 200 Widgets									1	
Acme - 600 Widgets									1	
ESales_01									1	
Global Media - 400 Widgets									1	
West_Sales_01									1	
salesforce.com - 1,000 Widgets									1	
salesforce.com - 2,000 Widgets									1	
salesforce.com - 500 Widgets									1	
salesforce.com - 5000 Widgets									1	
salesrep										

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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

'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.UserRoleld refers to the UserRoleld column of the User object. Wave Analytics looks up the user role ID of the user who submits the query from the User object.
'Ownerld' == "\$User.ld"	• Ownerld 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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.



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"
      1
```

```
}
   },
   "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 Ownerld. 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 for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

USER PERMISSIONS

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

 "Edit Wave Analytics Dataflows"

Dataflow View	•				Last refresh: Today at 10:41 AM	I
⊖ 🚺 Default Sa Last Modified	lesford By: Adm	e Dataflow in User			Runs every 24 hours at 7:46 AM SalesEdgeEltWorkflow - Run by Integration User	•
START TIME	•	DURATION	STATUS	MESSAGE	🖞 Upload	
✤ Today at 10:41 AM		N/A	Running X		▶ Start	
_					📅 Schedule	
 Yesterday at 2:46 PM 		0 hours, 2 minutes	Successful		O Unschedule	

- 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.



Available in Salesforce Classic and Lightning Experience.

Available for an extra cost in Enterprise, Performance, and Unlimited Editions. Also available in Developer Edition.

USER PERMISSIONS

""

^			Salesfo	rce Analy	/tics	Bill Rolley 🗸	?	Analytics	
OppRoles	×								
< v > C								Q D	1
bar length		Group	Color						
Count of Oppo	Heasure	by Name ar	d Opportunity.Ow.	+ Group	1	۰ 📥			
T Filter									
				Count of O	oportunity ID				
Name Opp	portunity.Owner.	Full Name 0					1	Opportunity.Owne	ame
Acc - 1000 Widgets	То	ny Santos					1	Bill Rolle	ey 👘
ESales_01		Bill Rolley					1	Tony Santo	DS

3. Log out and now log in as Keith.

As expected, Keith can still see all opportunities.

A		Salesforce A	nalytics	Keith Laz 🗸	۰ ۲	Analytics ~
➡ OppRoles	×					
< v > C						Ō 🖓 🏲
bar length Count of Oppo + M	Group Leasure by Name and	Color Opportunity.Ow. + Gro	oup 🔳	۵ 📥		
T Filter						
		C	Count of Oppor	tunity ID		
Name	Opportunity.Owner.Full Name	0				Opportunity.Owneame
Acc - 1000 Widgets	Tony Santos				1	Bill Rolley
Acme - 1,200 Widgets	Keith Laz				1	Keith Laz
Acme - 200 Widgets	Keith Laz				1	Tony Santos
Acme - 600 Widgets	Keith Laz	-			1	
ESales_01	Bill Rolley				1	
Global Media - 400 Widgets	Keith Laz				1	
TestOpp	Keith Laz				1	
West_Sales_01	Lucy Timmer				1	
salesforce.com - 1,000 Widgets	Keith Laz				1	
salesforce.com - 2,000 Widgets	Keith Laz				1	
salesforce.com - 500 Widgets	Keith Laz				1	
salesforce.com - 5000 Widgets	Keith Laz				1	

Row-Level Security Example Based on Territory Management

Let's look at an example where you create a dataset based on Salesforce data and then implement row-level security based on your defined territories. In this example, you determine what model you use for territory management, so you can later review sample JSON for that dataset. To restrict access on each record in the dataset, you will create a security predicate where each user can view only data appropriate for the territory to which they belong.

Territory management is an account sharing system that grants access to accounts based on the characteristics of the accounts. It enables your company to structure your Salesforce data and users the same way you structure your sales territories.

If your organization has a private sharing model, you might have granted users access to accounts based on criteria such as postal code, industry, revenue, or a custom field that is relevant to your business. Perhaps you also need to generate forecasts for these diverse categories of accounts. Territory management solves these business needs and provides a powerful solution for structuring your users, accounts, and their associated contacts, opportunities, and cases.

IN THIS SECTION:

1. Determine How You Use Territory Management

When working with security related to territory management, it helps to know how your organization implements territory management. Usually, one of 2 methods are used. Either accounts are assigned to regions manually, following some organization-specific precedence, or the organization use's Salesforce's territory hierarchy feature.

2. Create the DataSet

Now we look at sample JSON code that describes territory management in a dataset.

3. Create the Security Predicate

Now we can apply a security predicate to filter the dataset.

Determine How You Use Territory Management

When working with security related to territory management, it helps to know how your organization implements territory management. Usually, one of 2 methods are used. Either accounts are assigned to regions manually, following some organization-specific precedence, or the organization use's Salesforce's territory hierarchy feature.

The manual process

Manually Assigned Accounts			Add Accounts	8			
Action	Account Name	Billing State/Province	e	Phone	Туре	Account Owner Alias	Owner Alias
Remove	Santa's Workshop	North Pole				AUser	

For this example, any account with a Billing State or Province that is North Pole is manually assigned to the Canada region.

Territory Management hierarchies



For this example, we have a user called North America VP who needs access to all accounts in the Canada, Mexico, and US territories. We also have a user called Rep1 Canada who should only have access to the accounts in the Canada territory, not Mexico or US, and nowhere above in the hierarchy.

Create the DataSet

Now we look at sample JSON code that describes territory management in a dataset.

In this example, territory management data is stored on the following objects and fields.



***Territories can be nested, so you will need to flatten Territory before joining it to the Group and UserTerritory objects

```
Here is an example JSON file for this dataset.
```

```
{
  "Extract_AccountShare": {
   "action": "sfdcDigest",
   "parameters": {
      "object": "AccountShare",
      "fields": [
       { "name": "Id"},
        { "name": "RowCause"},
        { "name": "UserOrGroupId"},
        { "name": "AccountId"}
      ]
    }
  },
  "Extract Group": {
    "action": "sfdcDigest",
    "parameters": {
      "object": "Group",
      "fields": [
        { "name": "Name"},
        { "name": "Type"},
        { "name": "Id"},
        { "name": "RelatedId"}
      ]
    }
  },
  "Extract_Territory": {
   "action": "sfdcDigest",
    "parameters": {
      "object": "Territory",
      "fields": [
        { "name": "Id"},
        { "name": "Name"},
        { "name": "ParentTerritoryId"}
      ]
    }
 },
  "Extract User Territory": {
   "action": "sfdcDigest",
    "parameters": {
      "object": "UserTerritory",
      "fields": [
```

```
{ "name": "UserId"}
    ]
  }
},
"Extract_User": {
 "action": "sfdcDigest",
  "parameters": {
    "object": "User",
    "fields": [
     { "name": "Id"},
      { "name": "Name"}
    ]
  }
},
"Extract Account": {
  "action": "sfdcDigest",
  "parameters": {
    "object": "Account",
    "fields": [
     { "name": "Id"},
      { "name": "Name"},
      { "name": "BillingCountry"}
    ]
  }
},
"Augment TerritoryUsers": {
  "action": "augment",
  "parameters": {
    "left": "Extract Territory",
    "left_key": [
     "Id"
    ],
    "relationship": "TerritoryId",
    "right": "Extract User Territory",
    "right_key": [
     "TerritoryId"
    ],
    "right select": [
     "UserId"
    ],
    "operation": "LookupMultiValue"
  }
},
"Augment AccountShare To Territory Groups": {
  "action": "augment",
  "parameters": {
    "left": "Augment AccountShare To Account",
    "left key": [
     "UserOrGroupId"
    ],
    "relationship": "UserOrGroupId",
    "right": "Extract Group",
    "right key": [
```

{ "name": "TerritoryId"},

```
"Id"
     ],
     "right_select": [
      "Name",
       "RelatedId"
     ]
   }
 },
 "Augment AccountShare To Territory": {
   "action": "augment",
   "parameters": {
     "left": "Augment_AccountShare_To_Territory_Groups",
     "left key": [
      "UserOrGroupId.RelatedId"
     ],
     "relationship": "Territory",
     "right": "Augment_TerritoryUsers",
     "right_key": [
      "Id"
     ],
     "right select": [
      "TerritoryId.UserId"
     ],
     "operation": "LookupMultiValue"
   }
 },
 "Augment_AccountShare_To_Account": {
   "action": "augment",
   "parameters": {
     "left": "Extract AccountShare",
     "left key": [
      "AccountId"
     ],
     "relationship": "AccountId",
     "right": "Extract Account",
     "right_key": [
      "Id"
     ],
     "right select": [
      "Name"
     ]
   }
 },
 "Register Territory GroupUsers": {
   "action": "sfdcRegister",
    "parameters": {
      "alias": "Register Territory GroupUsers",
      "name": "Register Territory GroupUsers",
      "source": "Augment AccountShare To Territory"
    }
  }
}
```

When run, this JSON file results in a list of accounts. In this example, a list of 5:

Row-Level Security for Datasets

bar length		Group						
Count of Rows	+ Measure	by AccountId.Name	+ Group	≔ ‡	- F			
		,						
₽ Filter								
					Count of	Rows		
AccountId.Name 0		1			2			4
Acme							3	
Global Media								4
Porto Rico Beach					2			
Santa's Workshop								
salesforce.com							3	

Create the Security Predicate

Now we can apply a security predicate to filter the dataset.

Using this example, the following security predicate on the dataset enforces the territory management security rules.

'Territory.TerritoryId.UserId' == "\$User.Id" || 'UserOrGroupId' == "\$User.Id"

🗹 Note: Update the dataset, and then log out of and back in to the org so you see the changes.

Now you see only 2 accounts - Global Media because it is in the Canada territory, and Santa's Workshop because of the manual rule.

	<u>୪</u> ୦୦ ୪
bar length Group	
Count of Rows + Measure by AccountId.Name + Group 🗉 🕸 📻	
4 [™] Filter	
AccountId Name 0	1
Global Media	1
Santa's Workshop	1

SECURITY PREDICATE REFERENCE

Predicate Expression Syntax for Datasets

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 of type string, number, or multi-value picklist. 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. 	 'Owner.Role' == "\$User.UserRoleId" 'GroupID' == "\$User.UserGroupId_c" Note: Supported User object field value types are string, number, and multi-value picklist. Other types (for example, boolean) are not supported.

Value Type	Requirements	Predicate Expression Examples
	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 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
١Z	CTRL+Z (ASCII 26)
\"	One double-quote character
W	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
1	Single quote	Encloses a dataset column name in a predicate expression.
		Example predicate expression:
		'Expected_Revenue' >= 2000.00
u	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	Logical AND	True if both operands are true.
		Example predicate expression:
		'Stage Name' == "Closed Won" && 'isDeleted' != "False"
	Logical OR	True if either operand is true.
		Example predicate expression:
		'Expected_Revenue' > 4000 'Stage Name' == "Closed Won"

Sample Predicate Expressions for Datasets

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
ОррА	2000.00	Bill	20	Prospecting	True
ОррВ	3000.00	Joe	22	Closed Won	False
ОррС	1000.00	可爱的花	36	Closed Won	False
OppD	5000.00	O'Fallon	18	Prospecting	True
ОррЕ		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.
<pre>'Expected_Rev' > 1000 && 'Expected_Rev' <= 3000</pre>	
'Owner' = "Joe" 'Owner' = "Bill"	
<pre>('Expected_Rev' > 4000 'Stage Name' == "Closed Won") && 'isDeleted' != "False"</pre>	Parentheses specify the order of operations.
<pre>'Stage Name' == "Closed Won" && 'Expected_Rev' > 70000</pre>	
'Owner' == "可爱的花"	String contains Unicode characters.
'Owner' == "O\'Fallon"	Single quote in a string requires the escape character.

Predicate Expression	Details
'Stage Name' == ""	Checks for an empty string.