Salesforce Analytics Cloud Implementation and Data Integration Guide

Salesforce.com, Winter '15

@salesforcedocs

Last updated: September 23, 2014

CONTENTS

Introduction to Analytics Cloud 1

Analytics Cloud Setup 1
- Analytics Cloud Setup Process 1
- Permission Set Licenses for Analytics Cloud 4
- User Permissions for Analytics Cloud 5

Data Integration 6
- Data Integration Overview 6
- Data Integration Process 6
- Before You Integrate the Data 6

External Data 7
- Load External Data into a Dataset 7
- External Data API Reference (PILOT) 17

Dataflow 23
- Dataflow Definition File 24
- Dataflow Definition Example 25
- Configure the Dataflow 28
- Run, Stop, and Reschedule Dataflow Jobs 29
- Monitoring Dataflow Jobs 30

Transformations 32
- Augment Transformation 32
- Flatten Transformation 35
- Ngram Transformation 39
- Register Transformation 40
- Salesforce Extract Transformation 42

Dashboard JSON Overview 44
- View or Modify a Dashboard JSON File 45
- Dashboard JSON File 45
- Steps 47
- Widgets 48
- Query 48
- Query Example 51
- Static Steps 54
- Bindings 56
- Selection Binding in a Static Step 57
- Bind a Static Filter and Group Selector to a Query 62
- Relative Dates in a Static Filter Selector 64

SAQL Overview 65
- Operators 68
- Statements 71

Functions 76
Analytics Cloud Limits 79

Index 81

INTRODUCTION TO ANALYTICS CLOUD

EDITIONS

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

Analytics Cloud is a cloud-based platform for connecting data from multiple sources, creating exploration views of that data, and creating dashboards. These insights can be distributed to business users so that they can collaborate and take action to determine how their business is changing and what's behind those shifts.

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

IN THIS SECTION:

External Data

You can integrate external data into Analytics Cloud to make it available for queries from explorer and builder.

Dataflow

A dataflow is a reusable set of instructions that defines how to extract data from Salesforce, load it into datasets, and transform datasets. Use a dataflow to repeatedly extract, load, and transform the latest data to make it available for Analytics Cloud users.

Transformations

A transformation is an operation that Analytics Cloud performs on data or a dataset. You can add transformations to a dataflow to extract Salesforce data and transform datasets.

SAQL Overview

The SAQL language is a real-time query language that uses data flow as a means of aligning results. It enables ad hoc analysis of data that's stored in datasets.

Analytics Cloud Setup

Analytics Cloud Setup Process

EDITIONS

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

You must set up Analytics Cloud before users can access it. You must enable Analytics Cloud and assign user permissions to users.

You must complete the following tasks to set up Analytics Cloud.

1. Enable Analytics Cloud.
2. Assign an Analytics Cloud permission set license to each user.
3. Create permission sets to group related Analytics Cloud user permissions.
4. Assign the permission sets to Analytics Cloud users.

After you enable Analytics Cloud, you can access it. You can assign user permissions to other users to enable them to access Analytics Cloud as well.

You assign a permission set license to each user to identify which Analytics Cloud user permissions can be assigned to the user. If a user permission is not included in the permission set license assigned to the user, you cannot assign that user permission to the user.

Each permission set license can be assigned to one user—multiple users cannot share the same permission set license. Also, the user license associated with the user profile must support the Analytics Cloud permission set license that you assign to the user. Not all user licenses support Analytics Cloud permission set licenses.

Create permission sets to group related user permissions. You can then assign the permission set to users to quickly assign multiple user permissions to users. For example, you can group user permissions in two groups, one for loading data and another for using dashboards. Each Analytics Cloud permission set can include user permissions from one Analytics Cloud permission set license.

After you create the Analytics Cloud permission sets, assign them to users. You can assign a permission set to an individual user or a group of users. It's faster to assign a permission set to a group of users. You can assign multiple permission sets to a user.

Note: If you disable Analytics Cloud, user permissions are removed from each defined permisssion set. If you re-enable Analytics Cloud, you must define the permission sets again.

IN THIS SECTION:

Enable Analytics Cloud
Before users can access Analytics Cloud, you must enable it for your organization.

Assign a Permission Set License to a User
Some permissions require that you assign a permission set license to the user and then add the permissions to permission sets.

Creating Permission Sets
You can either clone an existing permission set or create a new one. A cloned permission set starts with the same user license and enabled permissions as the permission set it is cloned from, while a new permission set starts with no user license selected and no permissions enabled.

Assign a Permission Set to Multiple Users
From any permission set page, you can assign the permission set to one or more users.

Enable Analytics Cloud

Before users can access Analytics Cloud, you must enable it for your organization.

1. From Setup, click **Analytics** > **Settings**.
2. Select **Enable Analytics Cloud**.
3. Click **Save**.

Note: No data will be available for queries until you load data into Analytics Cloud.

EDITIONS

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

Assign a Permission Set License to a User

Some permissions require that you assign a permission set license to the user and then add the permissions to permission sets.

1. From Setup, click **Manage Users** > **Users**.
2. Click the name of the user to whom you want to assign the permission set license.
3. In the Permission Set License Assignments related list, click **Edit Assignments**.
4. Select the permission set license you want to assign to that user, and click **Save**.

Once you've assigned a permission set license to a user, add the related permission to a permission set and assign that permission set to the user.

EDITIONS

Available in:
- Enterprise
- Performance
- Unlimited
- Developer

USER PERMISSIONS

To assign a permission set license:
- "Manage Internal Users"

Creating Permission Sets

You can either clone an existing permission set or create a new one. A cloned permission set starts with the same user license and enabled permissions as the permission set it is cloned from, while a new permission set starts with no user license selected and no permissions enabled.

You can create up to 1,000 permission sets.

1. From Setup, click **Manage Users** > **Permission Sets**.
2. Do one of the following:
   - To create a permission set with no permissions enabled, click **New**.
   - To create a permission set based on an existing set, click **Clone** next to the set you want to copy. You can also select the permission set and click **Clone** in the overview page or one of the settings pages.

     Note: Clone a permission set only if the new one should have the same user license as the original. In a cloned permission set, you can't select a different license.

EDITIONS

Available in:
- Enterprise
- Performance
- Unlimited
- Developer
- Database.com

USER PERMISSIONS

To create permission sets:
- "Manage Profiles and Permission Sets"

3. Enter a label, API name, and description.

   The API name is a unique name used by the Force.com API and managed packages. It must begin with a letter and use only alphanumeric characters and underscores. It can't include spaces, end with an underscore, or have two consecutive underscores.

4. If this is a new permission set, select a user license option. If you plan to assign this permission set to multiple users with different licenses, select `--None--`. If only users with one type of license will use this permission set, select the user license that's associated with them.

   If you're cloning a permission set, you can't select a user license. If the User License field is blank, no user license is associated with the permission set.

5. Click **Save**.

The permission set overview page appears. From here, you can navigate to the permissions you want to add or change.

Assign a Permission Set to Multiple Users

From any permission set page, you can assign the permission set to one or more users.

1. From Setup, click **Manage Users** > **Permission Sets**.
2. Select a permission set.
3. In the permission set toolbar, click **Manage Assignments**.
4. Click **Add Assignments**.

   If any users are selected, this button isn't available.
5. Select the users to assign to this permission set.

   You can assign up to 1000 users at a time.

   Tip: Use the selected list view, select another list view, or to narrow the list of users using a different filter criteria, create a new list view.
6. Click **Assign**.
7. Review the messages on the Assignment Summary page. If any users weren't assigned, the Message column lists the reasons.
8. To return to a list of all users assigned to the permission set, click **Done**.

EDITIONS

Available in:
- Enterprise
- Performance
- Unlimited
- Developer
- Database.com

USER PERMISSIONS

To assign a permission set to users:
- "Assign Permission Sets"

Permission Set Licenses for Analytics Cloud

You assign Analytics Cloud permission set licenses to users to incrementally add Analytics Cloud features, which aren't included with their user licenses. You must assign an Analytics Cloud permission set license to a user before you can assign Analytics Cloud user permissions to the user. For example, before you can assign a permission set that includes the "Create and Edit Analytics Cloud Dashboards" permission to a user, you must assign the Builder permission set license to that user.

EDITIONS

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

Analytics Cloud contains the following permission set licenses:

Builder permission set license
Create, deploy, and manage data sets.

Explorer permission set license
View, discover, and personalize data insights.

The following table lists the Analytics Cloud user permissions included with each Analytics Cloud permission set license:

User Permission	Explorer Permission Set License	Builder Permission Set License
"Create and Edit Analytics Cloud Dashboards"		X
"Create Analytics Cloud Apps"		X
"Edit Analytics Cloud Dataflows"		X
"Manage Analytics Cloud"		X
"Upload External Data to Analytics Cloud"	X	X
"Use Analytics Cloud"	X	X

You can assign an Analytics Cloud permission set license on top of any of the following user licenses:

- Force.com - App Subscription
- Force.com - Free
- Force.com - One App
- Full CRM
- License Manager (INTERNAL)
- Salesforce Platform
- Salesforce Platform Light
- Salesforce Platform One
- Insights Integration User
- Service Cloud
- Sfdc Admin
- Work.com Only User

User Permissions for Analytics Cloud

EDITIONS

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

You can assign Analytics Cloud user permissions to specify what tasks users can perform and what features they can access. You can assign only user permissions that are included in the permission set license assigned to the user's profile.

The following table describes all Analytics Cloud user permissions:

User Permission	Description
"Create and Edit Analytics Cloud Dashboards"	Create and edit Analytics Cloud dashboards. Requires the "Use Analytics Cloud" user permission.
"Create Analytics Cloud Apps"	Create and share Analytics Cloud applications. Requires the "Use Analytics Cloud" user permission.
"Edit Analytics Cloud Dataflows"	Download, upload, start, stop, and reschedule the dataflow and view dataflow and system jobs in the dataflow manager. Requires the "Use Analytics Cloud" user permission.
"Manage Analytics Cloud"	Access all Analytics Cloud features. Requires the "Use Analytics Cloud" user permission.
"Upload External Data to Analytics Cloud"	Upload external data to Analytics Cloud to create a dataset and view dataflow and system jobs in the dataflow manager. Requires the "Use Analytics Cloud" user permission.
"Use Analytics Cloud"	Use Analytics Cloud, and view the datasets, lenses, and dashboards that the user has permission to view.

Data Integration

Data Integration Overview

You can integrate Salesforce and external data in Analytics Cloud to make the data available for queries from explorer and builder.

EDITIONS

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

Data Integration Process

The data integration process defines how to extract, load, and transform data, and make the resulting data available in Analytics Cloud.

Analytics Cloud provides a dataflow that you can configure to load Salesforce data into Analytics Cloud. A dataflow is a set of instructions that defines how to extract and load Salesforce data.

You can also load external data into Analytics Cloud. External data is data that resides outside of Salesforce—it's third-party data. You can use the Analytics Cloud user interface or the External Data API (Pilot) to upload external data.

EDITIONS

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

When you load data into Analytics Cloud, you load it into datasets. A dataset is a collection of related data. Analytics Cloud stores data in a denormalized form in datasets. To increase query performance, Analytics Cloud also indexes all dimension columns in datasets.

You can transform datasets using the dataflow. For example, you can use the dataflow to join data from two datasets.

SEE ALSO:

Dataflow

External Data

Before You Integrate the Data

Review the following guidelines before you start a data integration project.

- Your organization can have a maximum of 250 million rows of data stored in all queryable datasets. This limit excludes rows in datasets that aren't registered.
- You must register datasets to enable users to query them. Users cannot view or run queries against unregistered datasets.

  Note: Analytics Cloud automatically registers datasets that are built from external data files.

EDITIONS

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

External Data

You can integrate external data into Analytics Cloud to make it available for queries from explorer and builder.

The External Data API (Pilot) enables you to upload external data files to Analytics Cloud. The External Data API (Pilot) can upload .csv and .ebin files, and you can optionally specify the structure of your data by defining metadata in JSON format.

Note: For information about the proprietary Salesforce .ebin format, contact salesforce.com.

EDITIONS

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

The External Data API (Pilot) is available in API version 31 or later.

The high-level steps for uploading external data using the API are:

1. Prepare your data in .csv or .ebin format and create a metadata file to specify the structure of the data.
2. Connect programmatically to your Salesforce organization.
3. Configure the upload by inserting a row into the `InsightsExternalData` object, and set input values such as the name of the dataset, the format of the data, and the operation to perform on the data.
4. Split your data into 10-MB chunks, and upload the chunks to `InsightsExternalDataPart` objects.
5. Start the upload by updating the `Action` field in the `InsightsExternalData` object.
6. Monitor the `InsightsExternalData` object for status updates and confirm the successful uploading of files.

Load External Data into a Dataset

External Data API (Pilot) Limits

When working with the External Data API, consider the following limits, in addition to the general Analytics Cloud limits.

EDITIONS

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

Limit	Value
Maximum file size per external data upload	2 GB; if you upload multiple external data files simultaneously, the total file size of all files cannot exceed this limit
Maximum number of external data files uploaded per day	50 files
Maximum number of characters in a field	255 characters
Maximum number of fields in a record	5,000 fields
Maximum number of characters for all fields in a record	400,000 characters

Prepare Data Files

Metadata Overview

To upload external data from .csv or .ebin files into a dataset, you need to prepare your data files.

External data can be loaded into a dataset by preparing two files:

- A file that contains the external data, in comma-separated value (.csv) or .ebin format (data file)
- An optional JSON file that describes the structure of the data file (metadata file)

The data and metadata files are used to populate a dataset with the external data.

Note: For information about the .ebin format, contact salesforce.com.

EDITIONS

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

CSV Example

The following .csv example contains data that conforms to the JSON metadata file described below.

```
Name,Amount,CloseDate
opportunityA,100.99,6/30/2014
opportunityB,99.01,1/31/2012
```

The first row in the CSV file lists the field names for your object. Each subsequent row corresponds to a record of data. A record consists of a series of fields that are delimited by commas.

Note: Field names in the header of the .csv file can use only alpha-numeric or "_" characters.

The External Data API uses a strict format for field values to optimize processing for large sets of data. Note the following when generating CSV files:

- The delimiter for field values in a row must be a comma.
- If a field value contains a comma, a new line, or a double quote, the field value must be contained within double quotes: for example, `"Director of Operations, Western Region"`.
- If a field value contains a double quote, the double quote must be escaped by preceding it with another double quote: for example, `"This is the ""gold"" standard"`.
- Field values aren't trimmed. A space before or after a delimiting comma is included in the field value. A space before or after a double quote generates an error for the row. For example, `John,Smith` is valid; `John, Smith` is valid, but the second value is `" Smith"`;.`"John", "Smith"` is not valid.
- Empty field values are ignored when you update records. To set a field value to `null`, use a field value of `#N/A`.

JSON Example

The following JSON example represents a SalesData object with three fields: Name, Amount, and CloseDate.

```
{
    "fileFormat": {
    "charsetName": "UTF-8",
    "fieldsEnclosedBy": "\"",
    "fieldsDelimitedBy": ",",
    "linesTerminatedBy": "\n",
    "numberOfLinesToIgnore": 1
  },
  "objects": [
  {
        "connector": "AcmeCSVConnector",
        "description": "",
        "fullyQualifiedName": "SalesData",
        "label": "Sales Data",
        "name": "SalesData",
        "fields": [
        {
            "description": "",
            "fullyQualifiedName": "SalesData.Name",
            "label": "Account Name",
            "name": "Name",
            "isSystemField": false,
            "defaultValue": "",
            "isUniqueId": false,
            "isMultiValue": false,
            "multiValueSeparator": "null",
            "type": "Text",
            "precision": null,
            "scale": null,
            "format": ""
        },
        {
            "description": "",
            "fullyQualifiedName": "SalesData.Amount",
            "label": "Opportunity Amount",
            "name": "Amount",
            "isSystemField": false,
            "defaultValue": "",
            "isUniqueId": false,
            "isMultiValue": false,
            "multiValueSeparator": "null",
            "type": "Numeric",
            "precision": 10,
            "scale": 2,
            "format": "$#,##0.00"
        },
        {
            "description": "",
            "fullyQualifiedName": "SalesData.CloseDate",
            "label": "Opportunity Close Date",
            "name": "CloseDate",
            "isSystemField": false,
            "defaultValue": "",
            "isUniqueId": false,
            "isMultiValue": false,
            "multiValueSeparator": "null",
            "type": "Date",
            "precision": null,
            "scale": null,
            "format": "MM/dd/yyyy",
            "fiscalMonthOffset": 0
        }
     ]
```

```
  }
    ]
}
```

Metadata Reference

The metadata describes the structure of external data files. The metadata file is in JSON format. The JSON file consists of three main sections: file format, object information, and field information. You must include all required fields when you create a record, but you can leave out optional fields, if you don't need them.

File Format Section

The file format section of the metadata file specifies information about the format of the data file, including the character set and delimiter character.

Field Name	Type	Required?	Description
`charsetName`	String	Yes	Character set of the CSV file. Must be UTF-8. Example: `"charsetName": "UTF-8"`
`fieldsEnclosedBy`	String	Yes	Character that can be used to enclose fields in the CSV file. Only double-quotes are supported. If a double quote is used within a field, it must be escaped by preceding it with another double-quote. Must be "\"". Example: `"fieldsEnclosedBy": "\""`
`fieldsDelimitedBy`	String	Yes	Character that separates field values in the CSV file. Must be ",". Example: `"fieldsDelimitedBy": ","`
`linesTerminatedBy`	String	No	Character indicating the end of a line. Must be "\n". Example: `"linesTerminatedBy": "\n"`
`numberOfLinesToIgnore`	Number	Yes	Number of lines for the parser to ignore (allows you to specify a header). Must be at least 1. Example: `"numberOfLinesToIgnore": 1`

Objects Section

The objects section of the metadata file specifies information about the top-level database object, including object-level security information, display name, and API name.

Note: There can only be a single object definition within the metadata file.

Field Name	Type	Required?	Description
`rowLevelSecurityFilter`	String	No	The predicate used to apply row-level security on the dataset. For more information about creating the predicate, see the *Security Implementation Guide for Analytics Cloud*.
`connector`	String	Yes	String that uniquely identifies the client application. Example: `"connector": "AcmeCSVConnector"`
`description`	String	No	Description of the object. Example: `"description": "The SalesData object tracks basic sales data."`
`fullyQualifiedName`	String	Yes	Full path that uniquely identifies the record. Example: `"fullyQualifiedName": "CRM.SalesData"` Note: Names can use only alpha-numeric or "_" characters, and components of the path must be separated by a "." character.
`label`	String	Yes	Display name for the object. Example: `"label": "Sales Data"`
`name`	String	Yes	Unique API name for object. Example: `"name": "SalesData"` Note: Names can use only alpha-numeric or "_" characters.
`fields`	Array	Yes	Array of fields for this object.

Fields Section

The fields section of the metadata file specifies information about each field in the record, including data type and formatting information.

Field Name	Type	Required?	Description
`description`	String	No	Description of the field.

Field Name	Type	Required?	Description
			Example: `"description": "The Amount field contains the opportunity amount."`
`fullyQualifiedName`	String	Yes	Full path that uniquely identifies the field. (object.field) Example: `"fullyQualifiedName": "SalesData.Amount"` Note: Names can use only alpha-numeric or "_" characters, and components of the path must be separated by a "." character.
`label`	String	Yes	Display name for the field. Example: `"label": "Opportunity Amount"`
`name`	String	Yes	Unique API name for the field. Example: `"name": "Amount"` Note: Names can use only alpha-numeric or "_" characters.
`isSystemField`	Boolean	No	Indicates whether this is a system field and should be excluded from query results. Example: `"isSystemField": false`
`defaultValue`	String	No	Default value of the field, if any. All numeric types must have a default value.
`isUniqueId`	Boolean	No	Indicates whether this field is the primary key for the record. This field is required for incremental extract. Reserved for future use, and currently not supported. Example: `"isUniqueId": false`
`isMultiValue`	Boolean	No	Indicates whether the field has multiple values. Only applies to Text fields. Example: `"isMultiValue": false`
`multiValueSeparator`	String	No	Character used to separate multiple values. This must be specified if isMultiValue equals true. Should be null if isMultiValue equals false.

Field Name	Type	Required?	Description
			Example: `"multiValueSeparator": ";"`
`type`	String	Yes	Type of the field. Can be Text, Numeric, or Date. Example: `"type": "Numeric"`
`precision`	Number	Yes (for Numeric values)	Maximum number of digits in a numeric value, including all numbers to the left and to the right of the decimal point (but excluding the decimal point character). Example: `"precision": 10`
`scale`	Number	Yes (for Numeric values)	Number of digits to the right of the decimal point in a numeric value. Example: `"scale": 2`
`format`	String	Yes (for Numeric or Date values)	Format of the numeric or date value. For more information, see Numeric Formats and Date Formats. Example: `"format": "$#,##0.00" (Numeric)"format": " MM/DD/YYYY " (Date)`
`fiscalMonthOffset`	Number	Yes (for Date values)	Offset, in months, that the fiscal year differs from the calendar year. Represented as the month number - 1. For example, if the fiscal year starts in January, the offset is 0 (1 - 1). If the fiscal year starts in October, the offset is 9 (10 - 1). Example: `"fiscalMonthOffset": 1`

Numeric Formats

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

Symbol	Meaning
0	One digit
#	Zero or one digits
.	Decimal separator (only "." is supported as a decimal separator)
-	Minus sign
,	Grouping separator

Symbol	Meaning
$	Currency sign

Date Formats

For Date fields, you must specify the format of the date using one of the following formats. The timestamp portion of each date format is optional.

Format	Sample Value
yyyy-MM-dd'T'HH:mm:ss.SSSZ	2014-04-29T16:53:34.000Z
yy-MM-dd'T'HH:mm:ss.SSSZ	14-04-29T16:53:34.000Z
yyyy-MM-dd'T'HH:mm:ssZ	2014-04-29T16:53:34Z
yy-MM-dd'T'HH:mm:ssZ	14-04-29T16:53:34Z
yyyy-MM-dd HH:mm:ss	2014-06-03 11:31:45
yy-MM-dd HH:mm:ss	14-06-03 11:31:45
dd.MM.yyyy HH:mm:ss	03.06.2014 11:31:45
dd.MM.yy HH:mm:ss	03.06.14 11:31:45
dd/MM/yyyy HH:mm:ss	03/06/2014 11:31:45
dd/MM/yy HH:mm:ss	03/06/14 11:31:45
dd/MM/yyyy hh:mm:ss a	03/06/2014 11:31:45 AM
dd/MM/yy hh:mm:ss a	03/06/14 11:31:45 AM
dd-MM-yyyy HH:mm:ss	03-06-2014 11:31:45
dd-MM-yy HH:mm:ss	03-06-14 11:31:45
dd-MM-yyyy hh:mm:ss a	03-06-2014 11:31:45 AM
dd-MM-yy hh:mm:ss a	03-06-14 11:31:45 AM
MM/dd/yyyy hh:mm:ss a	06/03/2014 11:31:45 AM
MM/dd/yy hh:mm:ss a	06/03/14 11:31:45 AM
MM-dd-yyyy hh:mm:ss a	06-03-2014 11:31:45 AM
MM-dd-yy hh:mm:ss a	06-03-14 11:31:45 AM
HH:mm:ss dd/MM/yyyy	11:31:45 03/06/2014
HH:mm:ss dd/MM/yy	11:31:45 03/06/14

Note: Only the formats listed above are supported.

These formats use the following symbols:

Symbol	Meaning
yyyy or yy	Four-digit year (yyyy) or two-digit year (yy)
MM	Two-digit month (01-12)
dd	Two-digit day (01-31)
'T'	Separator indicating that time-of-day follows
HH	Two-digit hour (00-23)
mm	Two-digit minute (00-59)
ss	Two-digit seconds (00-59)
SSS	Optional three-digit milliseconds (000-999)
'Z'	Reference UTC timezone

Note: All dates must be in GMT.

Connect to Salesforce

EDITIONS

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

After preparing your data files, the next step in loading external data into Analytics Cloud is to connect to your Salesforce organization using standard Salesforce APIs.

Note: The following examples use the SOAP API, but you can use any of the Salesforce APIs. The examples assume that you are using the Force.com Web Services Connector.

To load external data into Analytics Cloud, you must first connect to your Salesforce organization. Use the `PartnerConnection` object to log into your organization, as shown in the following example. You'll need to supply a username, password, and endpoint.

```
ConnectorConfig config = new ConnectorConfig();
config.setUsername(username);
config.setPassword(password);
config.setAuthEndpoint(endpoint);

PartnerConnection partnerConnection = new PartnerConnection(config);
```

For more information about the Web Services Connector, see *Introduction to the Force.com Web Services Connector* on the Salesforce Developers website. For more information about user authentication, see "Security and the API" in the *SOAP API Developer's Guide*.

Configure the Upload

EDITIONS

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

Configure the external data upload by inserting a row into the `InsightsExternalData` object and setting configuration values.

After establishing a connection with Salesforce, insert a row into the `InsightsExternalData` object to configure and control the upload. The `InsightsExternalData` object provides a "header" that contains information about the upload, such as the name of the dataset, the format of the data, and the operation to perform on the data. You can also provide the metadata file. The following example inserts a row into the `InsightsExternalData` object, and sets configuration values.

```
SObject sobj = new SObject();
sobj.setType("InsightsExternalData");
sobj.setField("Format","CSV");
sobj.setField("EdgemartAlias", EdgemartName);
sobj.setField("MetadataJson",metadataJson);
sobj.setField("Operation","Overwrite");
sobj.setField("Action","None");

SaveResult[] results = partnerConnection.create(new SObject[] { sobj });

for(SaveResult sv:results)
    if(sv.isSuccess())
        parentID = sv.getId();
```

Note: The Web Services Connector (WSC) converts the metadata JSON file to a Base64-encoded string, but if you are using the REST API, you'll need to make this conversion yourself.

For detailed information about the `InsightsExternalData` object, see InsightsExternalData.

Add the Data

EDITIONS

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

When uploading external data files, you can use the `InsightsExternalDataPart` object to load the data in smaller chunks.

After inserting a row into the `InsightsExternalData` (header) object, you can split your data into 10-MB chunks and upload the chunks to `InsightsExternalDataPart` objects. The part objects are associated with the header object by setting the `InsightsExternalDataId` field on the part objects to the ID of the header object. The part objects contain the actual bytes of data, and must be assigned part numbers in a contiguous sequence, starting with 1.

The chunks of data should be smaller than 10 MB. If the data is compressed, it must be compressed first and then split into 10-MB chunks. Only the gzip format is supported.

The following example splits a file into 10-MB chunks, and uploads the chunks to `InsightsExternalDataPart` objects.

```
List<File> fileParts = chunkBinary(dataFile); //Split the file

for(int i = 0;i<fileParts.size();i++)
{
    SObject sobj = new SObject();
    sobj.setType("InsightsExternalDataPart");
    sobj.setField("DataFile", FileUtils.readFileToByteArray(fileParts.get(i)));
    sobj.setField("InsightsExternalDataId", parentID);
    obj.setField("PartNumber",i+1); //Part numbers should start at 1
    SaveResult[] results = partnerConnection.create(new SObject[] { sobj });
    for(SaveResult sv:results)
        if(sv.isSuccess())
            rowId = sv.getId();
}
```

For detailed information about the `InsightsExternalDataPart` object, see InsightsExternalDataPart.

Manage the Upload

EDITIONS

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

Start the external data processing by updating the `Action` field on the header object. Once the processing is underway, you can monitor its progress by checking the `Status` field on the header object.

Once you've created a header and uploaded the data parts using the `InsightsExternalData` and `InsightsExternalDataPart` objects, update the `Action` field on the header object to `Process` to start processing the data.

The following example sets the `Action` field and updates the row in the `InsightsExternalData` object.

```
SObject sobj = new SObject();
sobj.setType("InsightsExternalData");
sobj.setField("Action","Process");
sobj.setId(parentID);
SaveResult[] results = partnerConnection.update(new SObject[] { sobj });

for(SaveResult sv:results)
    if(sv.isSuccess())
        rowId = sv.getId();
```

Once the `Action` is set to `Process`, a dataflow job is created and marked active. You can monitor the `Status` field of the header object to determine when the file upload is completed.

External Data API Reference (PILOT)

InsightsExternalData Object (Pilot)

The `InsightsExternalData` object enables you to configure and control external data uploads. You can use it to provide metadata, trigger the start of the upload process, check status, and request cancellation and cleanup.

The `InsightsExternalData` object is used in conjunction with the `InsightsExternalDataPart` object, which holds the parts of the data to be uploaded. Together, they provide a programmatic way to upload a large file in parts and trigger a dataflow into a dataset. The first step is to insert a row into the `InsightsExternalData` object. Data parts are then uploaded to `InsightsExternalData` objects. The `Action` field of the `InsightsExternalData` object is updated to start processing and request cancellations. Once the `Action` field is updated to request processing, no user edits are allowed on the objects, except to request cancellation.

Note: The standard system fields (`CreatedById`, `CreatedDate`, `LastModifiedById`, `LastModifiedDate`, and `SystemModstamp`) are documented in "System Fields" in the *SOAP API Developer's Guide*.

The `InsightsExternalData` object is available in API version 31 or later.

Fields

Field	Details
`Id`	**Type** String

Field	Details
	Properties Defaulted on create, Filter, Group, Sort **Description** Unique ID generated by the system for new jobs.
`EdgemartAlias`	**Type** String **Properties** Create, Filter, Group, Sort, Update **Description** Alias of a dataset, which must be unique across an organization.
`EdgemartContainer`	**Type** String **Properties** Create, Filter, Group, Nillable, Sort, Update **Description** The folder name. If not specified when creating a new dataset, the `EdgemartContainer` will be the user's private folder. If not specified for an existing dataset, the `EdgemartContainer` will be the folder that currently contains the dataset. If `EdgemartContainer` is specified for an existing dataset, it must match the current folder that contains the dataset.
`MetadataJson`	**Type** Blob (Base64-encoded string) **Properties** Create, Nillable, Update **Description** Metadata in JSON format, describing the structure of the uploaded file. Required for upsert or delete operations.
`Format`	**Type** Enum **Properties** Create, Filter, Group, Sort, Update **Description** Format of the uploaded data. Can be CSV or Binary. For more information, see InsightsExternalDataFormat.
`Operation`	**Type** Enum

Field	Details
	Properties Create, Filter, Group, Sort, Update **Description** Indicates which database operation to use when loading data into the dataset. Can be Append, Upsert, Overwrite, or Delete. For more information, see InsightsExternalDataOperation.
`Status`	**Type** Enum **Properties** Create, Filter, Group, Sort, Update **Description** The current status of this data upload. Initial value is null. Can be New, Queued, In progress, Completed, Not processed, or Failed. For more information, see InsightsExternalDataStatus.
`Action`	**Type** Enum **Properties** Create, Filter, Group, Sort, Update **Description** The action to perform on this data. Can be None, Process, Abort, or Delete. For more information, see InsightsExternalDataAction.
`isIndependentParts`	**Type** Boolean **Properties** Create, Defaulted on create, Filter, Group, Sort, Update **Description** When `true`, indicates that file parts have been divided on row boundaries and can be processed independently of each other. The default is `false`.
`isDependentOnLastUpload`	**Type** Boolean **Properties** Create, Defaulted on create, Filter, Group, Sort, Update **Description** When `false`, indicates that this upload depends on the previous upload to the same dataset name.
`MetaDataLength`	**Type** Int

Field	Details
	Properties Create, Filter, Group, Nillable, Sort, Update **Description** Length of the metadata JSON file. This field is overwritten when data is uploaded.
`CompressedMetadataLength`	**Type** Int **Properties** Create, Filter, Group, Nillable, Sort, Update **Description** Length of the compressed metadata JSON file. This field is overwritten when data is uploaded.
`isDeleted`	**Type** Boolean **Properties** Defaulted on Create, Filter, Group, Sort **Description** Indicates whether the object has been moved to the Recycle Bin (`true`) or not (`false`).

InsightsExternalDataPart Object (Pilot)

The `InsightsExternalDataPart` object enables you to upload an external data file that has been split into parts.

The `InsightsExternalDataPart` object works in conjunction with the `InsightsExternalData` object. After you insert a row into the `InsightsExternalData` object, you can create part objects to split up your data into parts, if needed. You should split your file into parts that are smaller than 10 MB, if your initial data is larger than 10 MB.

Note: The standard system fields (`CreatedById`, `CreatedDate`, `LastModifiedById`, `LastModifiedDate`, and `SystemModstamp`) are documented in “System Fields” in the *SOAP API Developer's Guide*.

The `InsightsExternalDataPart` object is available in API version 31 or later.

Fields

Field	Details
`Id`	**Type** String **Properties** Defaulted on create, Filter, Group, Sort **Description** Unique ID of the part.

Field	Details
`PartNumber`	**Type** Int **Properties** Create, Filter, Group, Sort, Update **Description** The part number. Part numbers must be in a contiguous sequence, starting with 1. (1, 2, 3, etc.)
`InsightsExternalDataId`	**Type** String **Properties** Create, Filter, Group, Sort **Description** ID of the `InsightsExternalData` object that this part belongs to.
`DataFile`	**Type** Blob (Base64-encoded string) **Properties** Create, Nillable, Update **Description** The data bytes. May be zipped after the part has been separated from the full file.
`DataLength`	**Type** Int **Properties** Create, Filter, Group, Nillable, Sort, Update **Description** Length of the data. This field is overwritten when data is uploaded.
`CompressedDataLength`	**Type** Int **Properties** Create, Filter, Group, Nillable, Sort, Update **Description** Length of the compressed data. This field is overwritten when data is uploaded.
`isDeleted`	**Type** Boolean **Properties** Defaulted on Create, Filter, Group, Sort

Field	Details
	Description Indicates whether the object has been moved to the Recycle Bin (`true`) or not (`false`).

InsightsExternalDataAction Enum

The `InsightsExternalDataAction` enum describes the action to be taken when processing the data.

Enum Values

The following are the values of the `InsightsExternalDataAction` enum.

Value	Description
`None`	The user has not yet completed the data upload. This is the default value at creation.
`Process`	The user has completed the data upload, and is requesting that the data be processed.
`Abort`	The user no longer wants to upload the data, and the system should stop processing if possible. Reserved for future use.
`Delete`	The user wants to remove uploaded data parts as soon as possible. Implies that an Abort status is queued. Reserved for future use.

InsightsExternalDataFormat Enum

The `InsightsExternalDataFormat` enum describes the format of the data to upload.

Enum Values

The following are the values of the `InsightsExternalDataFormat` enum.

Value	Description
`CSV`	The data is in .csv format.
`Binary`	The data is in .ebin format.

InsightsExternalDataOperation Enum

The `InsightsExternalDataOperation` enum describes the operation to be performed when adding the data to the dataset.

Enum Values

The following are the values of the `InsightsExternalDataOperation` enum.

Value	Description
`Append`	Add all data to the dataset. Reserved for future use.
`Upsert`	Insert or update rows in the dataset. Reserved for future use.
`Overwrite`	Create a new dataset with the given data, and replace dataset if it already exists.
`Delete`	Delete the rows from the dataset. Reserved for future use.

InsightsExternalDataStatus Enum

The `InsightsExternalDataStatus` enum describes the current status of the data upload.

Enum Values

The following are the values of the `InsightsExternalDataStatus` enum.

Value	Description
`New`	The data upload job has been created.
`Queued`	The data upload job has been scheduled.
`In progress`	The data upload job is in progress.
`Failed`	The data upload job failed. Data parts are retained for XX days after failure.
`Completed`	The data upload job completed successfully. Data parts are retained for XX days after completion.
`Not processed`	The data upload job was aborted on user request. Data parts have been removed.

Dataflow

EDITIONS

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

A dataflow is a reusable set of instructions that defines how to extract data from Salesforce, load it into datasets, and transform datasets. Use a dataflow to repeatedly extract, load, and transform the latest data to make it available for Analytics Cloud users.

Analytics Cloud provides a default dataflow that contains some sample transformation logic. This dataflow is just a sample that will not run until you edit the logic. You must redefine the logic in the dataflow to meet your business requirements.

You can add transformations to the dataflow to extract different Salesforce data and to perform additional transformations on datasets. For example, you can add an Augment transformation to the dataflow to join data from two different datasets.

After you configure the dataflow, you upload the new dataflow to Analytics Cloud. Analytics Cloud validates access to Salesforce objects and fields based on the user profile of the user who uploads the dataflow. If the user profile does not have read access to a field or object, the upload fails.

The dataflow runs on a daily schedule to capture the latest changes to Salesforce data and changes in the dataflow logic. You can also start a dataflow on demand from the dataflow manager. When the dataflow runs, Analytics Cloud processes each transformation sequentially and runs the dataflow under the Integration User. At runtime, Analytics Cloud validates access to Salesforce objects and fields based on the user profile of the Integration User. For example, if you run the dataflow and it is configured to extract data from a custom field, the dataflow fails because the Integration User does not have read access on the custom fields by default.

You can also stop, reschedule, and monitor dataflow jobs in the dataflow manager.

IN THIS SECTION:

Dataflow Definition File

A dataflow definition file is a JSON file that contains the extract, load, and transformation logic for the dataflow. This file specifies the Salesforce objects to extract and load. It also specifies how to transform datasets that contain Salesforce or external data.

Dataflow Definition Example

The following dataflow definition is a sample that extracts data from Salesforce objects, loads the data into Analytics Cloud, and transforms and registers datasets. Use this sample to learn how to structure and add transformations to the dataflow definition file.

Configure the Dataflow

You can configure the dataflow to extract Salesforce data and transform datasets. To configure the dataflow, add transformations to the dataflow definition file.

Run, Stop, and Reschedule Dataflow Jobs

You can run, stop, and reschedule dataflow jobs in the dataflow manager. You cannot run more than one instance of a dataflow at a time.

Monitoring Dataflow Jobs

The Dataflow view of the dataflow manager shows the last 10 dataflow jobs and retains the last 7 days of history.

Dataflow Definition File

EDITIONS

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

A dataflow definition file is a JSON file that contains the extract, load, and transformation logic for the dataflow. This file specifies the Salesforce objects to extract and load. It also specifies how to transform datasets that contain Salesforce or external data.

You can edit the dataflow definition file to change the logic that's used to process the data. To edit the dataflow definition, download the dataflow definition JSON file from the dataflow manager. You can add, update, or remove transformations in the dataflow definition and then upload the updated dataflow definition file through the dataflow manager.

Analytics Cloud performs the following validation tasks when you upload a dataflow definition file.

- Verifies that the file is a JSON file.
- Verifies that there are no orphan nodes in the JSON file.
- Validates the syntax of the dataflow definition.

If any validation fails, the dataflow manager cancels the upload and displays an error.

SEE ALSO:

Transformations

Dataflow Definition Example

The following dataflow definition is a sample that extracts data from Salesforce objects, loads the data into Analytics Cloud, and transforms and registers datasets. Use this sample to learn how to structure and add transformations to the dataflow definition file.

EDITIONS

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

```
{
    "102": {
        "action": "sfdcDigest",
        "parameters": {
            "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"
                }
            ],
            "object": "User"
        }
    },
    "103": {
        "action": "sfdcDigest",
        "parameters": {
            "fields": [
                {
                    "name": "Id"
                },
                {
                    "name": "Name"
                },
                {
                    "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"
                    }
                ],
                "object": "UserRole"
            }
        },
        "104": {
            "action": "flatten",
            "parameters": {
                "multi_field": "Roles",
                "parent_field": "ParentRoleId",
                "path_field": "RolePath",
                "self_field": "Id",
                "source": "103"
            }
        },
        "105": {
            "action": "augment",
            "parameters": {
                "left": "102",
                "left_key": [
                    "UserRoleId"
                ],
                "relationship": "Role",
                "right": "104",
                "right_key": [
                    "Id"
                ],
                "right_select": [
                    "Name",
                    "Roles",
                    "RolePath"
                ]
            }
        },
        "106": {
            "action": "sfdcRegister",
            "parameters": {
                "alias": "users",
                "name": "UsersWithRoles",
                "source": "105"
            }
        }
    }
```

Configure the Dataflow

You can configure the dataflow to extract Salesforce data and transform datasets. To configure the dataflow, add transformations to the dataflow definition file.

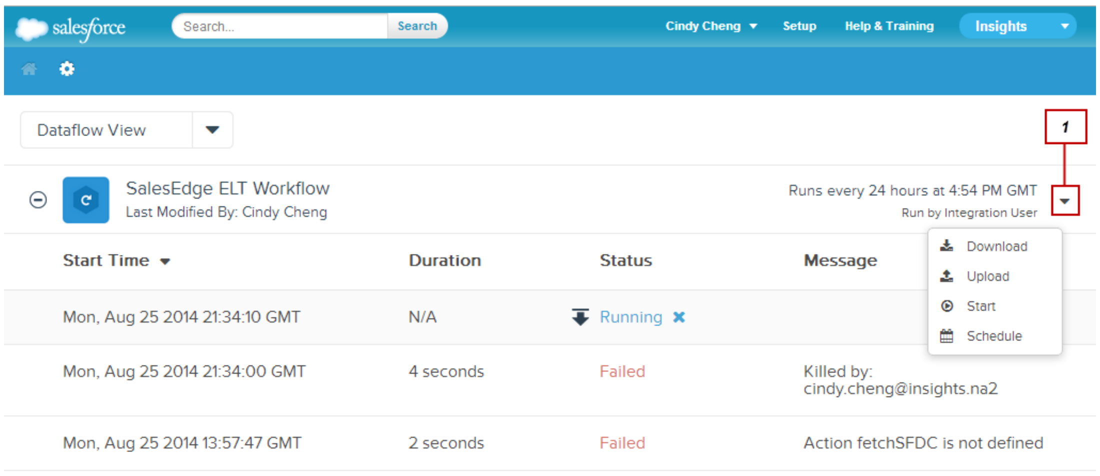
EDITIONS

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

USER PERMISSIONS

To edit the dataflow definition file:
- "Edit Analytics Cloud Dataflows"

1. In Analytics Cloud, click the gear icon (⚙) and then click **Dataflow Manager** to open the dataflow manager.
   The Dataflow view of the dataflow manager appears by default.
2. In the Dataflow view, click **Download** from the action list next to the dataflow to download the dataflow definition file.



Start Time	Duration	Status	Message
Mon, Aug 25 2014 21:34:10 GMT	N/A	Running	
Mon, Aug 25 2014 21:34:00 GMT	4 seconds	Failed	Killed by: cindy.cheng@insights.na2
Mon, Aug 25 2014 13:57:47 GMT	2 seconds	Failed	Action fetchSFDC is not defined

3. Make a backup copy of the dataflow definition file before you modify it.
   Analytics Cloud doesn't retain previous versions of the file. You may need to upload the previous version to roll back your changes.
4. Add transformations to the dataflow definition file.
   You can add the following transformations:

Transformation	Description
Augment	The Augment transformation joins data from two datasets to enable queries across both of them. For example, you can augment the User dataset with the Account dataset to enable the user to generate a query that displays all account details, including the names of the account owner and creator.
Flatten	The Flatten transformation flattens the Salesforce role hierarchy. Flatten the role hierarchy when you want to implement role-based security. With role-based security, a user can access

Transformation	Description
	records that are owned and shared by their subordinates in the role hierarchy.
Ngram	The Ngram transformation generates a case-sensitive, full-text search index based on data in a dataset. Analytics Cloud uses the search index to quickly retrieve stored data.
Register	The Register transformation registers a dataset to make it available for queries. Users cannot view or run queries against unregistered datasets.
Salesforce Extract	The Salesforce Extract transformation extracts data from fields of a Salesforce object. You specify the Salesforce object and fields from which to extract data. .

5. After you are done, save and close the dataflow definition file.
6. In the Dataflow view of the dataflow manager, click **Upload** from the action list next to the dataflow 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.

After the dataflow job runs, the data in registered datasets will be available for queries.

Run, Stop, and Reschedule Dataflow Jobs

You can run, stop, and reschedule dataflow jobs in the dataflow manager. You cannot run more than one instance of a dataflow at a time.

1. In Analytics Cloud, click the gear icon (⚙) and then click **Dataflow Manager** to open the dataflow manager.
   The dataflow manager has two views: Dataflow view and Job view. By default, the Dataflow view displays.
2. To perform tasks on the dataflow, click the actions list (1) for the dataflow.

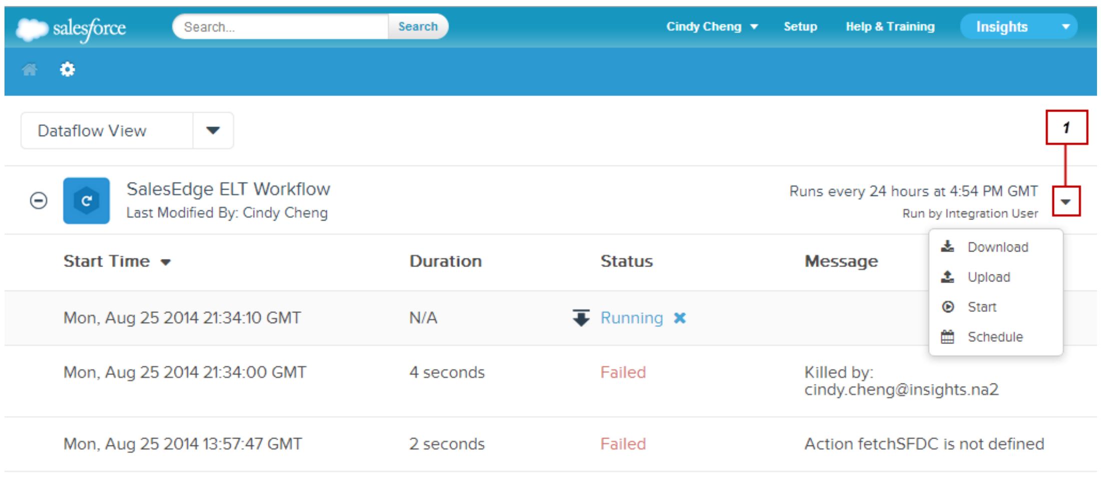
EDITIONS

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

USER PERMISSIONS

To access the dataflow manager:
- "Edit Analytics Cloud Dataflows" or "Manage Analytics Cloud"

To run a dataflow job on-demand:
- Edit Analytics Cloud Dataflows



SalesEdge ELT Workflow
Last Modified By: Cindy Cheng

Runs every 24 hours at 4:54 PM GMT
Run by Integration User

Start Time	Duration	Status	Message
Mon, Aug 25 2014 21:34:10 GMT	N/A	Running	
Mon, Aug 25 2014 21:34:00 GMT	4 seconds	Failed	Killed by: cindy.cheng@insights.na2
Mon, Aug 25 2014 13:57:47 GMT	2 seconds	Failed	Action fetchSFDC is not defined

The actions list displays the following options: **Download**, **Upload**, **Start**, and **Schedule**.

3. Click **Start** in the actions list to run the dataflow job now.
   The dataflow job is added to the job queue. The **Start** button is greyed out while the dataflow job is running.
4. To stop a dataflow job that is currently running, click next to the job.
   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.
5. Click **Schedule** in the actions list to change the time the dataflow is scheduled to run.

Monitoring Dataflow Jobs

The Dataflow view of the dataflow manager shows the last 10 dataflow jobs and retains the last 7 days of history.

The Dataflow view shows the status, start time, and duration of each job. To help you troubleshoot a failed job, it also displays error messages and run-time details for each transformation included in the dataflow.

1. In Analytics Cloud, click the gear button ( ) and then click **Dataflow Manager** to open the dataflow manager.
   The Dataflow view shows by default.

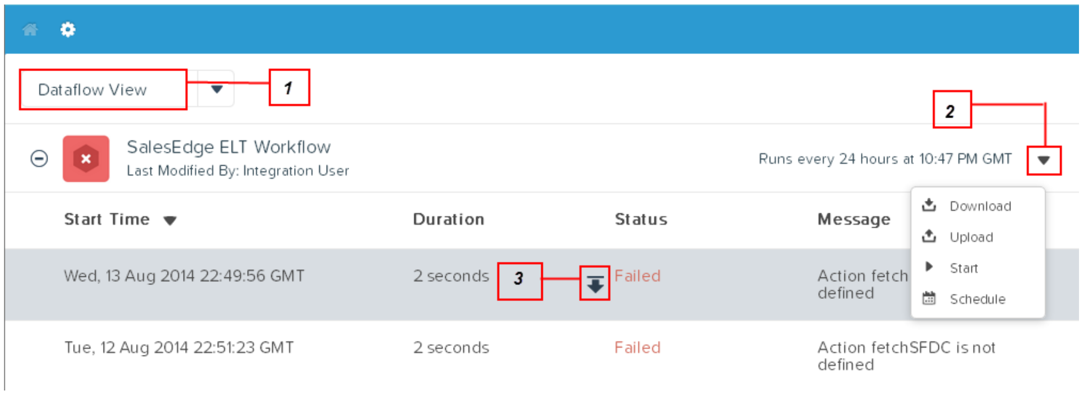
EDITIONS

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

USER PERMISSIONS

To access the dataflow manager:

- "Edit Analytics Cloud Dataflows" or "Manage Analytics Cloud"



SalesEdge ELT Workflow
Last Modified By: Integration User
Runs every 24 hours at 10:47 PM GMT

Start Time	Duration	Status	Message
Wed, 13 Aug 2014 22:49:56 GMT	2 seconds	Failed	Action fetch defined
Tue, 12 Aug 2014 22:51:23 GMT	2 seconds	Failed	Action fetchSFDC is not defined

2. Review the status of the dataflow job.

   The Dataflow view shows one of the following statuses for the latest dataflow job.

Status Icon	Description
✓	The dataflow job completed successfully.
↻	The dataflow job is currently running.
✕	The dataflow job failed.

3. Click the **Refresh Jobs** button (↻) to see the latest status of a running job.
4. Click the downward arrow (3) to the left of the status icon to view the run-time details for each transformation that was processed as part of the dataflow job.
5. If the dataflow job fails, click the actions list button (2) and then click **Download** to download the dataflow definition file.
6. Make a backup copy of the dataflow definition file before you edit it to fix the errors.

   Analytics Cloud doesn't retain previous versions of the file. You may need to upload the previous version to roll back your changes.
7. After you fix the file, click the actions list button and then click **Upload** to upload the updated file.
8. Click the actions list button (2) and then click **Start** to start the dataflow job again to verify your fix.

Transformations

A transformation is an operation that Analytics Cloud performs on data or a dataset. You can add transformations to a dataflow to extract Salesforce data and transform datasets.

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

To add a transformation to the dataflow, add the transformation operation to the dataflow definition file. A transformation operation is a procedure that you insert into the dataflow definition file. During run time, Analytics Cloud calls the procedures in the order that's specified in the dataflow definition file.

EDITIONS

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

IN THIS SECTION:

Augment Transformation
The Augment transformation joins data from two datasets to enable queries across both of them. For example, you can augment the User dataset with the Account dataset to enable the user to generate a query that displays all account details, including the names of the account owner and creator.

Flatten Transformation
The Flatten transformation flattens the Salesforce role hierarchy. Flatten the role hierarchy when you want to implement role-based security. With role-based security, a user can access records that are owned and shared by their subordinates in the role hierarchy.

Ngram Transformation
The Ngram transformation generates a case-sensitive, full-text search index based on data in a dataset. Analytics Cloud uses the search index to quickly retrieve stored data.

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

Salesforce Extract Transformation
The Salesforce Extract transformation extracts data from fields of a Salesforce object. You specify the Salesforce object and fields from which to extract data.

SEE ALSO:

Configure the Dataflow

Augment Transformation

The Augment transformation joins data from two datasets to enable queries across both of them. For example, you can augment the User dataset with the Account dataset to enable the user to generate a query that displays all account details, including the names of the account owner and creator.

The Augment transformation creates a new dataset based on data from two input datasets. You identify each input dataset as the left or right dataset. The new dataset contains all the columns of the left dataset and appends only the specified columns from the right dataset. The Augment transformation performs a left outer join, where the new dataset contains all rows from the left dataset and only matched rows from the right dataset.

EDITIONS

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

To enable queries that span 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.

Note: Although performance might not be as fast, instead of augmenting datasets, you can also join data from different datasets within a SAQL query.

IN THIS SECTION:

Join Condition for the Augment Transformation

You must specify a join condition in the Augment transformation to join the datasets. The join condition determines how to match rows in the right dataset with those in the left.

Augment Operation Reference

Add the `augment` operation to the dataflow definition file to add an Augment transformation to a dataflow. You must use the required JSON syntax and include all required JSON name-value pairs for the operation.

Join Condition for the Augment Transformation

EDITIONS

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

You must specify a join condition in the Augment transformation to join the datasets. The join condition determines how to match rows in the right dataset with those in the left.

Similar to creating an SQL join, you augment two datasets based on a key from each dataset. The key must uniquely identify rows in each dataset. A key can be a single-column key or a composite key.

To augment datasets based on a composite key, the keys from both datasets must have the same number of columns and the corresponding columns must be specified in the same order. For example, the keys might be:

```
"left_key": [ " AcctID, QuotaID " ],"right_key": [ " Account_ID, Quota_ID " ]
```

IN THIS SECTION:

Join Condition Example

When you create an Augment transformation, you specify a join condition to match rows in the right dataset with those in the left. The following example shows how to use a single-column join condition.

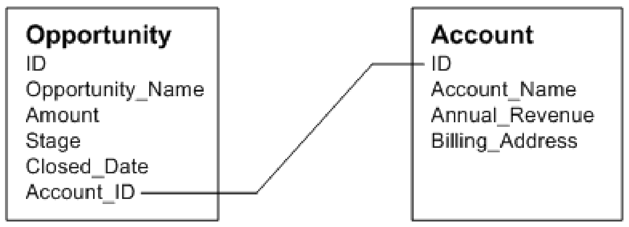
Join Condition Example

EDITIONS

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

When you create an Augment transformation, you specify a join condition to match rows in the right dataset with those in the left. The following example shows how to use a single-column join condition.

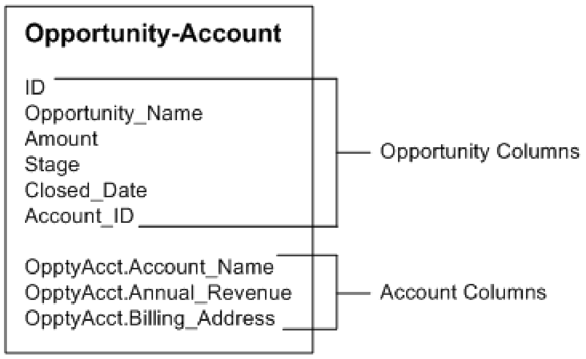
You want to augment the following datasets based on a single-column key:



You assign Opportunity as the left dataset and Account as the right dataset. You also specify "OpptyAcct" as the relationship between them. When you run the dataflow, it adds the "OpptyAcct" prefix to all Account columns and joins the datasets based on the following keys:

```
Opportunity.Account_ID=Account.ID
```

After you run the dataflow to augment the two input datasets, the resulting dataset contains the following columns:



Augment Operation Reference

Add the `augment` operation to the dataflow definition file to add an Augment transformation to a dataflow. You must use the required JSON syntax and include all required JSON name-value pairs for the operation.

EDITIONS

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

Syntax

Use the following syntax as shown in the sample `augment` operation:

```
"105_AugmentUserAndRoles" : {
    "action" : "augment" ,
    "parameters" : {
        "left" : "102_User" ,
        "left_key" : ["UserRoleId"],
        "right" : "104_Roles" ,
        "relationship" : "Owner" ,
        "right_select": [
                    "Name",
                    "Roles",
                    "RolePath"
            ],
        "right_key" : ["Id"]
}
}
```

The syntax is case-sensitive.

Name-Value Pairs

The following table describes the name-value pairs for the `augment` operation:

Name	Required?	Value
action	True	Operation name for the Augment transformation. Set to "augment."
parameters	True	An array of parameters for the operation.
left	True	Node in the dataflow definition file that identifies the left dataset.
left_key	True	Key column that uniquely identifies a record in the left dataset. If you use a composite key, the left and right keys must have the same number of columns in the same order. For a composite key, use the following syntax: `[ "Key Column1", "Key Column2", …, "Key ColumnN" ]`
right	True	Node in the dataflow definition file that identifies the right dataset.
relationship	True	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	True	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	True	Key column that uniquely identifies a record in the right dataset. If you use a composite key, the left and right keys must have the same number of columns in the same order.

Flatten Transformation

The Flatten transformation flattens the Salesforce role hierarchy. Flatten the role hierarchy when you want to implement role-based security. With role-based security, a user can access records that are owned and shared by their subordinates in the role hierarchy.

To view the list of objects that support access based on role hierarchies, see the organization-wide default settings in Salesforce.

EDITIONS

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

IN THIS SECTION:

Flatten Transformation Example

You want to implement role-based access on account records to enable Analytics Cloud users to view account records that their subordinates own or share.

Flatten Operation Reference

Add the `flatten` operation to the dataflow definition file to add a Flatten transformation to a dataflow. You must use the required JSON syntax and include all required JSON name-value pairs for the operation.

Flatten Transformation Example

EDITIONS

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

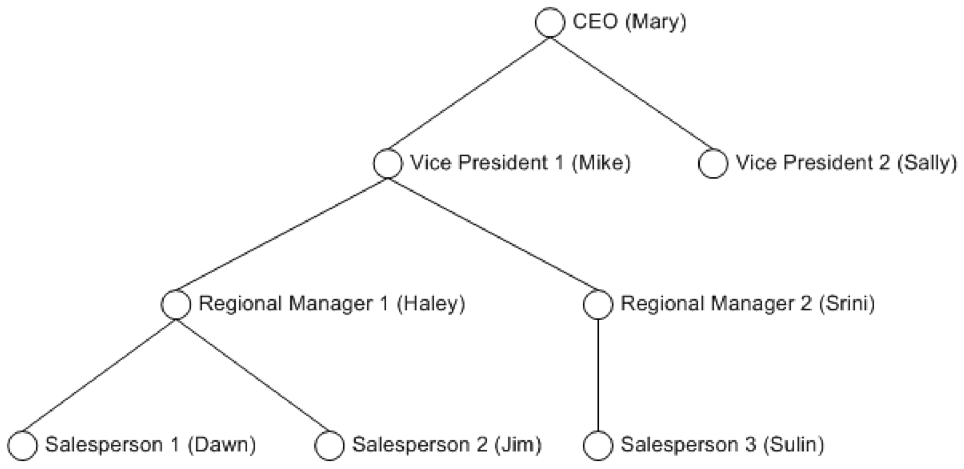
You want to implement role-based access on account records to enable Analytics Cloud users to view account records that their subordinates own or share.

Your high-level plan is to gather a list of parents for each user based on the Salesforce role hierarchy. Then, you append the applicable parents of the record owners to each account record.

Your organization has the following role hierarchy.

```
CEO > Vice President > Regional Manager > Salesperson
```

The organization contains the following users:



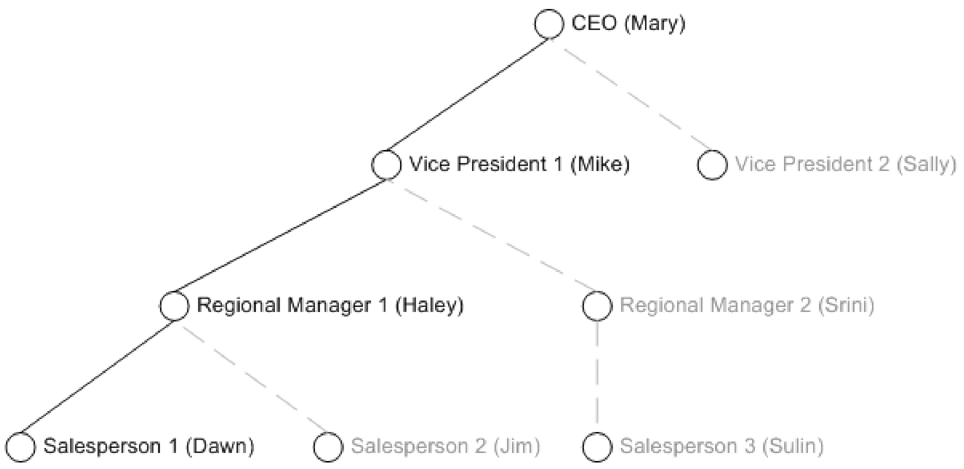
To implement role-based access on accounts in Analytics Cloud, you perform the following steps.

1. Load the users into a dataset. Add a Salesforce Extract transformation to the dataflow to extract users from the User object and load the data into a dataset that's called Users.
2. Load the user roles into a dataset. Add a Salesforce Extract transformation to the dataflow to extract user roles from the UserRole object and load the data into a dataset that's called UserRoles.
3. Load accounts into a dataset. Add a Salesforce Extract transformation to the dataflow to extract accounts from the Account object and load the data into a dataset that's called Accounts.
4. Generate a list of parents for each user role. Add a Flatten transformation to the dataflow to generate a list of parents for each user role based on the UserRole dataset. The Flatten transformation generates a new dataset that's called FlatUserRoles.

   The following table shows some sample data after you flatten the role data in the UserRole dataset:

User Role ID	Role Name	Parent Role ID	Hierarchy Nodes	Hierarchy Path
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

The Flatten transformation adds the Hierarchy Nodes and Hierarchy Path columns to the dataset to trace the parents in the role hierarchy for each user. Analytics Cloud uses the parent lineage to determine the users that have access to records that are owned or shared by subordinates. For example, Mary, Mike, and Haley have access to Dawn's records based on the following role hierarchy:



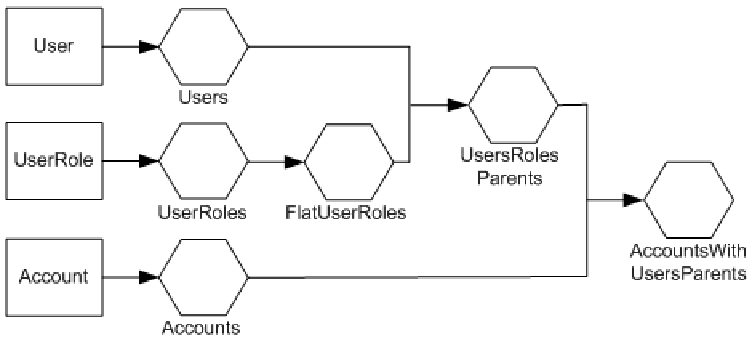
5. Combine the flattened role and user data into a new dataset. Add an Augment transformation to the dataflow to combine the user data from the Users dataset and flattened role data from the FlatUserRoles dataset into a single dataset that's called UsersRolesParents.
6. Combine the user, role, and parent data with the account data in a new dataset. Add an Augment transformation to the dataflow to combine the data from the UsersRolesParents dataset and Accounts dataset into a single dataset that's called AccountsWithUsersParents. The Augment transformation adds the parents of users to each account to enable parents to view the account records.

   The following table shows sample data in the new AccountsWithUsersParents dataset:

Account ID	Account Owner ID	Role Name	Parent Role ID	Hierarchy Nodes	Hierarchy Path
500	1	Salesperson 1	10	10, 20, 30	\10\20\30
501	2	Salesperson 2	10	10, 20, 30	\10\20\30

Account ID	Account Owner ID	Role Name	Parent Role ID	Hierarchy Nodes	Hierarchy Path
502	3	Salesperson 3	11	11, 20, 30	\11\20\30

The following diagram summarizes the steps to implement role-based access on accounts:



Flatten Operation Reference

Add the `flatten` operation to the dataflow definition file to add a Flatten transformation to a dataflow. You must use the required JSON syntax and include all required JSON name-value pairs for the operation.

EDITIONS

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

Syntax

Use the following syntax as shown in the sample `flatten` operation:

```
"104_FlattenRoles" : {
    "action" : "flatten" ,
    "parameters" : {
        "multi_field" : "Roles" ,
        "parent_field" : "ParentRoleId" ,
        "path_field" : "RolePath" ,
        "self_field" : "Id" ,
        "source" : "103_ExtractRoles"
        }
    }
```

The syntax is case-sensitive.

Name-Value Pairs

The following table describes the name-value pairs for the `flatten` operation:

Name	Required?	Value
action	True	Operation name for the Flatten transformation. Set to "flatten."
parameters	True	An array of parameters for the operation.
multi_field	True	Multi-value field that contains a comma-separated list of all parent roles in the role hierarchy, in order from the lowest to the highest level. For example, for a salesperson, the value is: `Regional Manager, Vice President, CEO`.
parent_field	True	Parent node of the node in the hierarchy.
path_field	True	The hierarchical path of all parent roles in the role hierarchy, in order from the lowest to the highest level. For example, for a salesperson, the value is: `Regional Manager\Vice President\CEO`.
self_field	True	ID of the node in the hierarchy.
source	True	Node in the dataflow definition file that identifies the dataflow that contains the hierarchy that you want to flatten.

Ngram Transformation

The Ngram transformation generates a case-sensitive, full-text search index based on data in a dataset. Analytics Cloud uses the search index to quickly retrieve stored data.

The Ngram transformation generates the search index based on ngrams. The ngram index enables Analytics Cloud to automatically complete search phrases when you start entering a search string. For example, when you enter "sal" as the beginning of your search string, Analytics Cloud suggests possible search strings such as "Sales," "Sally," and "Bansal" based on the data that contains "sal."

From a technical standpoint, an ngram is a contiguous sequence of n items from a given sequence of text. Although you can configure the size of the ngram for the Ngram transformation, salesforce.com recommends that you use bigrams, which are ngrams of size 2.

EDITIONS

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

IN THIS SECTION:

Ngram Operation Reference

Add the `ngram` operation to the dataflow definition file to add an Ngram transformation to a dataflow. You must use the required JSON syntax and include all required JSON name-value pairs for the operation.

Ngram Operation Reference

Add the `ngram` operation to the dataflow definition file to add an Ngram transformation to a dataflow. You must use the required JSON syntax and include all required JSON name-value pairs for the operation.

EDITIONS

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

Syntax

Use the following syntax as shown in the sample `ngram` operation:

```
"107_Ngram_User": {
    "action": "ngram",
    "parameters": {
        "source": "106_Index_RoleID",
        "width": "2"
        }
    }
```

The syntax is case-sensitive.

Name-Value Pairs

The following table describes the name-value pairs for the `ngram` operation:

Name	Required?	Value
action	True	Operation name for the Ngram transformation. Set to 'ngram.'
parameters	True	An array of parameters for the operation.
source	True	Node in the dataflow definition file that identifies the dataflow for which you want to generate bigrams.
width	True	Width of the ngram. Set to '2' to create bigrams.

Register Transformation

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

Note: 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 load external data because Analytics Cloud automatically registers these datasets for you.

EDITIONS

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

IN THIS SECTION:

Register Operation Reference

Add the `sfdcRegister` operation to the dataflow definition file to add a Register transformation to a dataflow. You must use the required JSON syntax and include all required JSON name-value pairs for the operation.

Register Operation Reference

Add the `sfdcRegister` operation to the dataflow definition file to add a Register transformation to a dataflow. You must use the required JSON syntax and include all required JSON name-value pairs for the operation.

EDITIONS

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

Syntax

Use the following syntax as shown in the sample `sfdcRegister` operation:

```
"108_Register": {
    "action": "sfdcRegister",
    "parameters": {
        "alias": "User",
        "name": "User",
        "source": "107_Ngram_UserAndFlatRoles"
}
}
```

The syntax is case-sensitive.

Name-Value Pairs

The following table describes the name-value pairs for the `sfdcRegister` operation:

Name	Required?	Value
action	True	Operation name for the Register transformation. Set to “sfdcRegister.”
parameters	True	An array of parameters for the operation.
alias	True	Display name of the registered dataset.
name	True	Internal name of the registered dataset. The internal name must be unique among all datasets in the organization.
source	True	Node in the dataflow definition file that identifies the dataset that you want to register.

Salesforce Extract Transformation

The Salesforce Extract transformation extracts data from fields of 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.

The Salesforce Extract transformation generates a dataset for the Salesforce object based on the specified fields. You must register the dataset to make it available for queries.

For more information about Salesforce objects, see http://www.salesforce.com/us/developer/docs/object_reference/.

EDITIONS

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

IN THIS SECTION:

sfdcDigest Operation Reference

Add the `sfdcDigest` operation to the dataflow definition file to add a Salesforce Extract transformation to a dataflow. You must use the required JSON syntax and include all required JSON name-value pairs for the operation.

sfdcDigest Operation Reference

Add the `sfdcDigest` operation to the dataflow definition file to add a Salesforce Extract transformation to a dataflow. You must use the required JSON syntax and include all required JSON name-value pairs for the operation.

EDITIONS

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

Syntax

Use the following syntax as shown in the sample `sfdcDigest` operation:

```
{"102_User" : {
    "action" : "sfdcDigest" ,
    "parameters" : {
        "fields" : [
            {"name" : "Id"},
            {"name" : "Username"},
            {"name" : "LastName"},
            {"name" : "FirstName"},
            {"name" : "Name"},
            {"name" : "CompanyName"},
            {"name" : "Division"},
            {"name" : "Department"},
            {"name" : "Title"},
            {"name" : "Street"},
            {"name" : "City"},
            {"name" : "State"},
            {"name" : "PostalCode"},
            {"name" : "Country"},
            {"name" : "Email"},
            {"name" : "Phone"},
            {"name" : "Fax"},
            {"name" : "MobilePhone"},
            {"name" : "Alias"},
            {"name" : "CommunityNickname"},
            {"name" : "IsActive"},
            {"name" : "TimeZoneSidKey"},
            {"name" : "UserRoleId"}],
        "object" : "User"
        }
    }
```

The syntax is case-sensitive.

Name-Value Pair Names

The following table describes the name-value pairs for the `sfdcDigest` operation:

Name	Required?	Value
action	True	Operation name for the Salesforce Extract transformation. Set to "sfdcDigest."
parameters	True	An array of parameters for the operation.
fields	True	An array of names of all fields from which you want to extract data from the specified Salesforce object. The Salesforce Extract transformation cannot extract data from fields with the following field types: • address • base64 • calculated • DataCategoryGroupReference • encryptedstring • location • masterrecord • textarea If you include a field with an unsupported field type in the Salesforce Extract transformation, the dataflow ignores the field.
name	True	Name of the field in the Salesforce object that you want to include in the dataset. You can specify multiple fields.
object	True	Name of the Salesforce object from which you want to extract data.

Name	Required?	Value
maxAllowedOffset	False	System field that only applies when extracting data from teh User object. For salesforce.com's use only.

Dashboard JSON Overview

To create advanced dashboards, it might be necessary to directly modify the JSON that defines a dashboard.

The easiest way to design dashboards is to use the builder. However, to complete the following tasks, you must modify the dashboard's JSON file.

- Specify a SAQL query, and specify relationships between the query and other steps.
- Populate a selector with a specified list of values instead of from a query.
- Use manual bindings to override the default faceting and manually specify the relationships between the steps.
- Set query limits.
- Specify columns for a values table.

EDITIONS

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

IN THIS SECTION:

View or Modify a Dashboard JSON File
To create advanced dashboards, it might be necessary to modify the JSON file that defines a dashboard.

Dashboard JSON File
A dashboard JSON file defines the components that a dashboard contains and describes how they're connected together.

Steps
The `steps` section contains all of the queries that you've clipped from the explorer.

Widgets
The `widgets` section defines all of the widgets that appear in the dashboard. Each widget has a name.

Query
The `query` section defines the query for that step.

Query Example
This example shows a dashboard that contains two queries.

Static Steps
You can also populate a selector from a specified list of values instead of from a query.

Bindings
After you define steps, you bind them to the widgets.

Selection Binding in a Static Step
Almost all parts of a step can include a selection binding to the results of a prior query.

Bind a Static Filter and Group Selector to a Query
Static filters or group selectors can be bound to a query written in SAQL.

Relative Dates in a Static Filter Selector

View or Modify a Dashboard JSON File

To create advanced dashboards, it might be necessary to modify the JSON file that defines a dashboard.

USER PERMISSIONS

To modify the JSON file that defines a dashboard:

- "Create and Edit Analytics Cloud Dashboards"

1. In your browser's address bar, type the URL of the Create Lens page. For example, if your Salesforce.com instance is `na3.salesforce.com`, type `https://na3.salesforce.com/insights/web/lens.apexp` in your browser's address bar.
2. In the list of lenses, click the lens to modify it.
   The JSON that defines that lens is displayed in the Lens text box. To increase the size of the text box, click and drag the resizing handle in its bottom right corner.
3. Modify the JSON in the Lens text box. Optionally, cut and paste the text into a text editor or JSON editor, make your changes, and then paste it back into the text box.
4. Click **Update Lens**.
   The changes are saved.

Dashboard JSON File

A dashboard JSON file defines the components that a dashboard contains and describes how they're connected together.

EDITIONS

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

This sample JSON file defines a simple dashboard that uses a number widget to display the count of rows in a dataset. This sample JSON file defines one step, called `"step_1"`, and one widget, called `"number_1"`. The `"edgemarts"` section lists all of the datasets that the dashboard uses.

```
{
    "name_lc": "simple example dashboard",
    "state": {
        "widgets": {
            "number_1": {
                "params": {
                    "title": "",
                    "textColor": "#000",
                    "measureField": "count",
                    "fontSize": 36,
                    "step": "step_1"
                },
                "type": "NumberWidget",
                "pos": {
                    "w": 300,
                    "y": 40,
                    "h": "auto",
                    "x": 40
                }
            }
        },
        "steps": {
            "step_1": {
                "isFacet": true,
                "start": null,
                "query": {
                    "values": [],
                    "order": [],
                    "pigql": null,
                    "dimensions": [],
                    "measures": [
                        [
                            "count",
                            "*"
                        ]
                    ],
                    "aggregateFilters": [],
                    "groups": [],
                    "filters": [],
                    "formula": null
                },
                "extra": {
                    "chartType": "hbar"
                },
                "selectMode": "single",
                "useGlobal": true,
                "em": "0Fb400000004CH2CAM",
                "type": "aggregate",
                "isGlobal": false
            }
        },
        "cards": {}
    },
    "_uid": "0FK400000004CGOGA2",
    "_createdBy": {
        "_type": "user",
        "profilePhotoUrl": "https://myorg/profilephoto/005/T",
        "name": "Insights DashEditor",
        "_uid": "00540000000Hew7AAC"
    },
    "folder": {
        "_type": "folder",
        "_uid": "00540000000Hew7AAC"
    },
    "_container": {
        "_container": "0FK400000004CGOGA2",
        "_type": "container"
    },
    "_type": "dashboard",
    "edgemarts": {
        "emName": {
            "_type": "edgemart",
            "_uid": "0Fb400000004CH2CAM"
        }
    },
    "_createdDateTime": 1406060540,
    "_permissions": {
        "modify": true,
        "view": true
    },
    "description": "",
    "_url": "/insights/internal_api/v1.0/esObject/lens/0FK400000004CGOGA2/json",
    "name": "Simple example dashboard",
    "_lastAccessed": 1406060541,
    "_files": {}
}
```

Steps

The *steps* section contains all of the queries that you've clipped from the explorer.

Each step has a name that's used to link it to a widget that's defined elsewhere in the JSON file.

The properties of the steps section of a dashboard JSON file are:

EDITIONS

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

Field Name	Description
`em`	The alias of the dataset that this step uses.
`extra`	Extra information about the step.
`isFacet`	Indicates whether the step will be connected to other steps used in the dashboard (`true`) or not (`false`), that reference the same dataset.
`isGlobal`	Indicates whether the filter that's specified in the query will be used as a global filter (`true`) or not (`false`). A global filter will filter all other steps in the dashboard that have their `useGlobal` property set to `true` , and reference the same dataset.
`query`	The query that the step uses. It can be in SAQL or compact form.
`selectMode`	Determines the selection interaction for charts and selectors. The options for charts are: `none`, `single`, and `single_required`. The options for selectors are: `single`, `single_required`, `multi`, and `multi_required`.
`start`	The default start value or values for a step. This value will be used when a dashboard is initialized or refreshed.
`type`	The type can be set to `grain`, `aggregate`, `multi`, or `static`.
`useGlobal`	Indicates whether the step should use the dashboard's global filter (`true`) or not (`false`).

Widgets

The *widgets* section defines all of the widgets that appear in the dashboard. Each widget has a name.

The properties of the widgets section of a dashboard JSON file are:

Field Name	Description
`params`	Widget parameters vary depending on the type of widget. The step that a widget is attached to is defined by its `step` element.
`pos`	The top left corner of the widget is specified by `x` and `y`. Width is `w`, and height is `h`. Measurements are in pixels.
`type`	The widget type specifies one of the other supported widget types such as `NumberWidget`, `ChartWidget`, `ValuesTable`, `CompareTable`, `PillBox`, `ListSelector`, or `TextWidget`.

EDITIONS

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

Query

The *query* section defines the query for that step.

The properties of the *query* section of a dashboard JSON file are:

Field Name	Description
`pigql`	The SAQL query to use. The SAQL language is a real-time query language that uses data flow as a means of aligning results. It enables ad hoc analysis of data that's stored in datasets.
`dimensions`	The dimensions to use are specified like this: `"dimensions": [ "Department" ]`
`measures`	The measures to use are specified like this: `"measures": [["count", "*"]]`
`values`	Values are used with the `grain` step type in a step for a raw data table widget. Values list all of the columns to include in a grain or raw data table. For example: `"step_grain": { "type": "grain", "em": "opp", "query": { "values": ["Amount", "Owner-Name", "Name", "Account-Name", "StageName", "ForecastCategory", "Current Age", "Time to Win"], } }`

```
"step_grain": {
  "type": "grain",
  "em": "opp",
  "query": {
    "values": ["Amount", "Owner-Name", "Name",
"Account-Name", "StageName", "ForecastCategory",
"Current Age", "Time to Win"],
    }
}
```

EDITIONS

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

Field Name	Description
`filters`	The filter conditions to apply to the data. Here is an example of a simple filter condition to include only rows that have the destination "SFO", "LAX", "ORD", or "DFW": `"filters": [["dest", ["SFO", "LAX", "ORD", "DFW"]]]`
`groups`	The dimension to group by. For example, `"groups": ["carrier"]`.
`order`	The sort order is specified like this: `"order": [[ -1, { "ascending": false } ]]` The value, -1, indicates that the ordering will be done for the first measure. To order the results in ascending order, set ascending to `true`. To order the results in descending order, set ascending to `false`. If you don't want to impose a specific order, specify empty brackets like this: `"order": []`.
`limit`	The number of results to return. For example, `"limit": 10`. The results that are returned by the limit statement aren't automatically ordered, so you must use this statement only with data that has been ordered.
`formula`	Formula is used with the *`multi`* step type in a step for a compare table. In a *`multi`* type step, there is more than one subquery. You can use the basic mathematical operators, `*`, `/`, `-`, `+`, `(`, and `)`, to create a formula to reference other subqueries in the step. To reference other subqueries, use the automatically assigned names: "A" is the first query, "B" is the second query, and so on. (example below)

```
 "step_comptable": {
      "type": "multi",
      "em": "opp",
      "isFacet": true,
      "useGlobal": true,
      "query": {
        "columns": [
          {
            "header": "Opptys Won",
            "query": {
              "pigql": null,
              "filters": [["StageName", ["5 - Closed-Won"]], ["Close
Date", [[["year", -1], ["year", 0]]]]],
              "measures": [["count", "*"]],
              "values": [],
              "groups": ["Owner-Name"],
              "formula": null,
              "order": []
            }
          }, {
            "header": "Opptys Won ($)",
            "query": {
              "pigql": null,
              "filters": [["StageName", ["5 - Closed-Won"]]],
              "measures": [["sum", "Amount"]],
              "values": [],
              "groups": ["Owner-Name"],
              "formula": null,
              "order": []
            }
```

Field Name	Description

```
          }, {
            "sort": {
              "asc": false,
              "inner": false
            },
            "header": "Opptys Won ($)",
            "showBars": true,
            "query": {
              "pigql": null,
              "filters": [["StageName", ["5 - Closed-Won"]]],
              "measures": [["sum", "Amount"]],
              "values": [],
              "groups": ["Owner-Name"],
              "formula": null,
              "order": []
            }
          }, {
            "header": "Opptys Lost ($)",
            "query": {
              "pigql": null,
              "filters": [["StageName", ["5 - Closed-Lost"]]],
              "measures": [["sum", "Amount"]],
              "values": [],
              "groups": ["Owner-Name"],
              "formula": null,
              "order": []
            }
          }, {
            "header": "Opptys Lost ($)",
            "showBars": true,
            "query": {
              "pigql": null,
              "filters": [["StageName", ["5 - Closed-Lost"]]],
              "measures": [["sum", "Amount"]],
              "values": [],
              "groups": ["Owner-Name"],
              "formula": null,
              "order": []
            }
          }, {
            "header": "Win-Loss (%)",
            "query": {
              "groups": ["Owner-Name"],
              "filters": [["StageName", ["5 - Closed-Lost"]]],
              "measures": [["sum", "Amount"]],
              "values": [],
              "pigql": null,
              "formula": "B/(B+D)*100",
              "order": []
            }
          }
        ]
```

Field Name	Description
	`} } },`
`aggregateFilters`	Automatically generated. Do not modify.
`facet_filters`	Automatically generated. Do not modify.

Within the query section of a step, you can manually insert bindings. To do this, use templates, which are expressions that are embedded in double braces ({{ }}) and that get replaced with the current state of the step that they are attached to. For example:

```
"filters": [["carrier", "{{ selection(step1) }}"], ["dest", "{{ filter(step1, 'dest') }}"],
 ["origin", "{{ filter(step1, 'origin') }}"]]
```

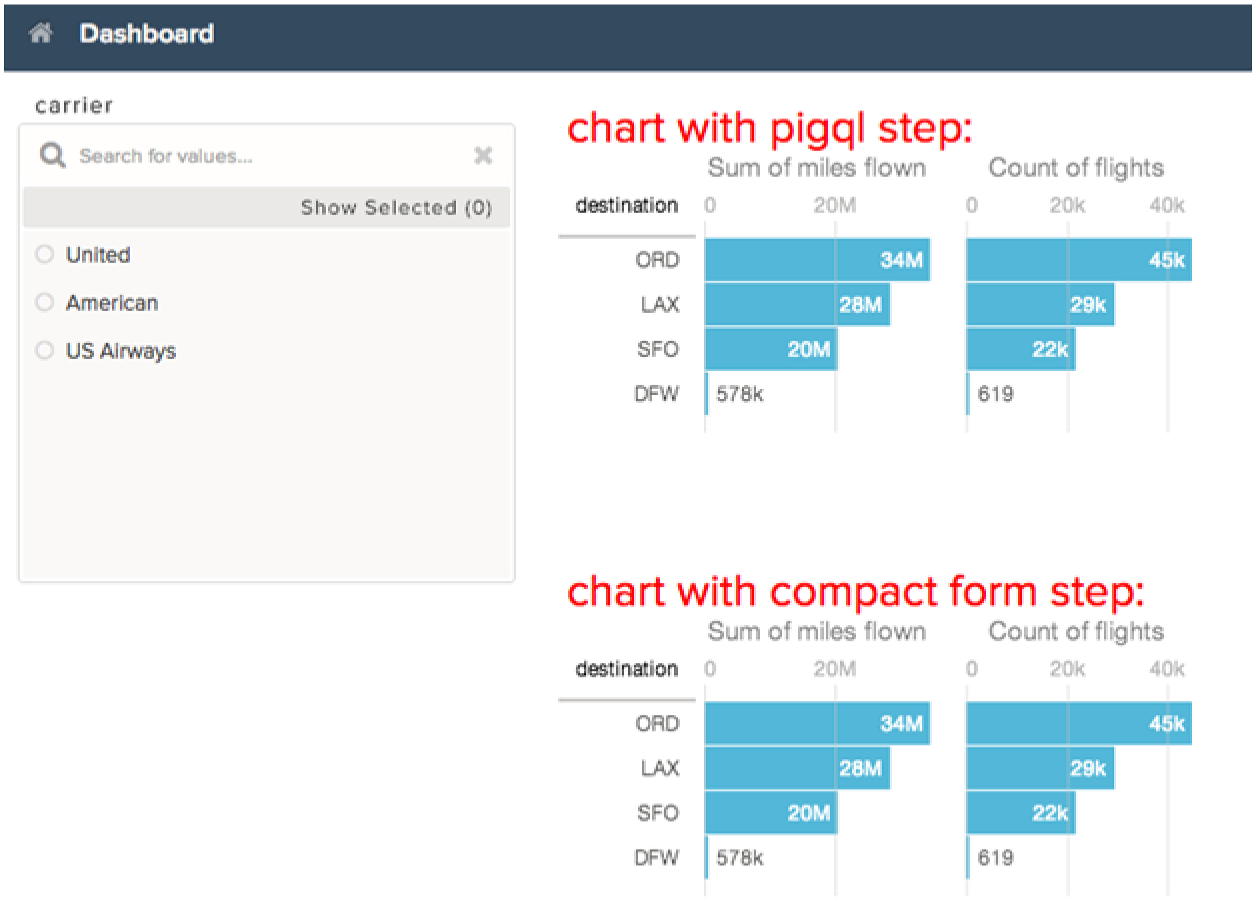
Query Example

This example shows a dashboard that contains two queries.

The first bar chart is connected to a step that contains a query that uses SAQL. The second bar chart is connected to a step that contains a compact form query.

EDITIONS

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



```
{
  "steps": {
    "step1": {
      "type": "aggregate",
      "em": "airline",
      "query": {
        "groups": ["carrier"],
        "filters": [["dest", ["SFO", "LAX", "ORD", "DFW"]]],
        "measures": [["count", "*"]],
        "order": [
          [
            -1, {
              "ascending": false
            }
          ]
        ],
        "limit": 3
      }
    },
    "step2": {
      "type": "aggregate",
      "em": "airline",
      "query": {
        "groups": ["dest"],
        "filters": [["carrier", "{{ selection(step1) }}"], ["dest", "{{ filter(step1,
'dest') }}"], ["origin", "{{ filter(step1, 'origin') }}"]],
        "measures": [["sum", "miles"], ["count", "*"]],
        "order": [
          [
            -1, {
              "ascending": false
            }
          ]
        ]
      }
    },
    "step3": {
      "type": "aggregate",
      "em": "airline",
      "query": {
        "pigql": "q = load \"airline\";\nq = filter q by 'carrier' in {{ selection(step1)
 }};\nq = filter q by 'dest' in {{ filter(step1, 'dest') }};\nq = filter q by 'origin' in
 {{ filter(step1, 'origin') }};\nq = group q by 'dest';\nq = foreach q generate 'dest' as
 'dest', sum('miles') as 'sum_miles', count() as 'count';\nq = order q by 'count' desc;",

        "groups": ["dest"],
        "measures": [["sum", "miles"], ["count", "*"]]
      }
    }
  },
  "widgets": {
    "barchart1": {
      "type": "ListSelector",
      "pos": {
        "x": 10,
        "y": 10,
        "w": 270,
        "h": 180
      },
      "params": {
        "step": "step1"
      }
    },
    "text2": {
      "type": "TextWidget",
      "pos": {
        "x": 310,
        "y": 10
      },
      "params": {
        "text": "chart with pigql step:",
        "textColor": "#f00"
      }
    },
    "barchart2": {
      "type": "ChartWidget",
      "pos": {
        "x": 310,
        "y": 30,
        "w": 400,
        "h": 280
      },
      "params": {
        "step": "step2",
        "chartType": "hbar"
      }
    },
    "text3": {
      "type": "TextWidget",
      "pos": {
        "x": 310,
        "y": 280
      },
      "params": {
        "text": "chart with compact form step:",
        "textColor": "#f00"
      }
    },
    "barchart3": {
      "type": "ChartWidget",
      "pos": {
        "x": 310,
        "y": 300,
        "w": 400,
        "h": 280
      },
      "params": {
        "step": "step3",
        "chartType": "hbar"
      }
    }
  }
}
```

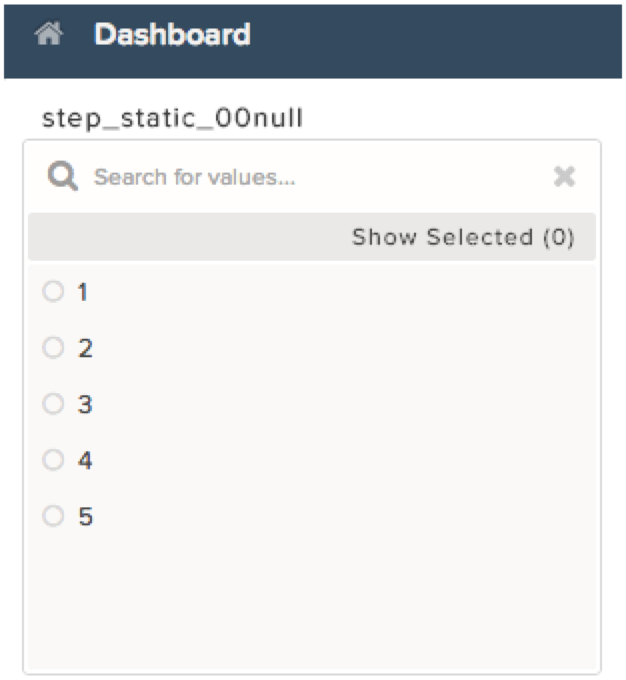
Static Steps

You can also populate a selector from a specified list of values instead of from a query.

A static step is shown in this example:

EDITIONS

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



```
  "steps": {
    "step_static_00null": {
      "type": "static",
      "values": [
        {
          "display": "1",
          "value": "1",
          "measure": 100000
        }, {
          "display": "2",
          "value": "2",
          "measure": 200000
        }, {
          "display": "3",
          "value": "3",
          "measure": 300000
        }, {
          "display": "4",
          "value": "4",
          "measure": 400000
        }, {
          "display": "5",
          "value": "5",
          "measure": 500000
        }
      ],
      "selectMode": "single"
    },
```

Bindings

After you define steps, you bind them to the widgets.

The kinds of bindings are:

- Selection binding
- Results binding
- Filter binding

EDITIONS

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

Selection Binding

When a user makes a selection in a dashboard, that selection values can be used to update other steps and widgets to make the dashboard interactive.

When you build a dashboard with the dashboard builder UI, by default, everything is faceted. The "isFaceted" option for each step takes care of bidirectional selection bindings between steps of the same dataset. However, you can modify a dashboard JSON file directly to manually specify the relationships between the various step to achieve:

- Selection bindings between steps of different datasets.
- Unidirectional selection binding.
- Selection binding for a static step.

Results Binding

Results binding is used to filter a step using the values resulting from another step. It's typically used across multiple datasets. An example of when results binding is useful is when you want to filter opportunities by top-selling products.

```
step_all_salesreps:
 type: "aggregate"
 em: "opp"
 query:
  groups: ["Owner-Name"]
  filters: [
   ["StageName", ["5 - Closed-Won"]]
   ["Products", "{{ results(step_top5_products) }}"]
  ]
  measures: [ ["sum", "Amount"] ]
```

Filter Binding

If a step is constrained by a particular set of filters, filter binding is used to make another step be constrained by the same filter values. Filter binding also works on steps that reference different datasets. The following step is constrained by a *`CloseDate Year`* of *`2012`*:

```
step_owner_by_role:
 type: "aggregate"
 em: "opp"
 query:
  groups: ["Owner-UserRole-Name", "CloseDate Year"]
  filters: [
   ["CloseDate Year",  ["2012"]]
  ]
  measures: [ ["sum", "Amount"] ]
```

To constrain subsequent steps by the same filter dimension and values, the appropriate dimension, step, and dimension name is referenced within filters:

```
step_quota_filtered_by_role:
  type: "aggregate"
  em: "quota"
  query:
   filters: [
    ["Closed Year", "{{ filter(step_opp_owner_role, 'CloseDate Year') }}"]
   ]
   measures: [ ["sum", "Amount"] ]
```

Selection Binding in a Static Step

Almost all parts of a step can include a selection binding to the results of a prior query.

In an aggregate query, the fields that can be included in a selection binding are:

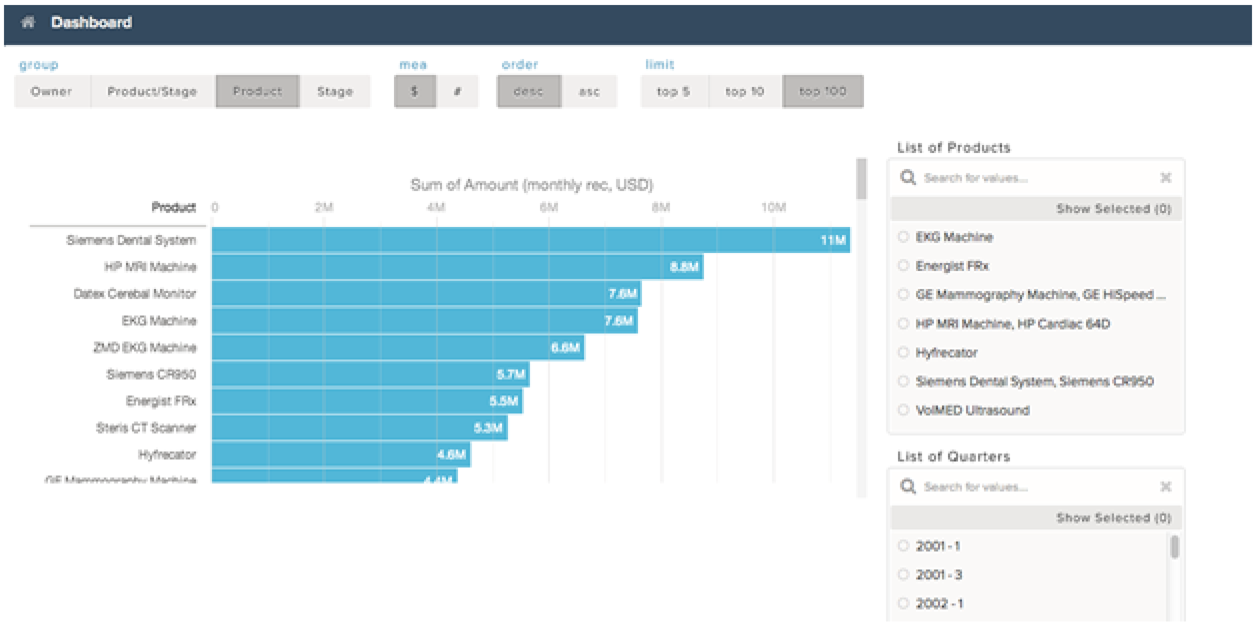
- Group
- Measure
- Filters
- Sort
- Limit

EDITIONS

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

Use Static Steps for Binding Any Part of a Query

This example shows a dashboard with static steps and selection bindings in multiple parts of a query.



```
{
  "steps": {
    "step_filter_dim": {
      "type": "static",
      "dim": "Product",
      "em": "opp",
      "selectMode": "single",
      "values": [
        {
          "value": ["EKG Machine"]
        }, {
          "value": ["Energist FRx"]
        }, {
          "value": ["GE Mammography Machine", "GE HiSpeed DXi", "GE Stress System"]
        }, {
          "value": ["HP MRI Machine", "HP Cardiac 64D"]
        }, {
          "value": ["Hyfrecator"]
        }, {
          "value": ["Siemens Dental System", "Siemens CR950"]
        }, {
          "value": ["VolMED Ultrasound"]
        }
      ],
      "isFacet": true
    },
    "step_group": {
      "type": "static",
      "values": [
        {
          "display": "Owner",
          "value": ["Owner-Name"]
        }, {
          "display": "Product/Stage",
          "value": ["Product", "StageName"]
        }, {
          "display": "Product",
          "value": ["Product"]
        }, {
          "display": "Stage",
          "value": ["StageName"]
        }
      ],
      "start": [["Product"]],
      "selectMode": "single"
    },
    "step_measure": {
      "type": "static",
      "values": [
        {
          "display": "$",
          "value": [["sum", "Amount"]]
        }, {
```

```
        "display": "#",
        "value": [["count", "*"]]
      }
    ],
    "start": [[["sum", "Amount"]]],
    "selectMode": "single_required"
  },
  "step_order": {
    "type": "static",
    "values": [
      {
        "display": "desc",
        "value": false
      }, {
        "display": "asc",
        "value": true
      }
    ],
    "selectMode": "single_required"
  },
  "step_limit": {
    "type": "static",
    "values": [
      {
        "display": "top 5",
        "value": 5
      }, {
        "display": "top 10",
        "value": 10
      }, {
        "display": "top 100",
        "value": 100
      }
    ],
    "start": [100],
    "selectMode": "single_required"
  },
  "step_quarterly_bookings": {
    "type": "aggregate",
    "em": "opp",
    "query": {
      "groups": [["CloseDate_Year", "CloseDate_Quarter"]],
      "measures": [["sum", "Amount"]]
    },
    "isFacet": true,
    "useGlobal": true
  },
  "step_top_10": {
    "type": "aggregate",
    "em": "opp",
    "query": {
      "groups": "{{ selection(step_group) }}",
      "measures": "{{ selection(step_measure) }}",
      "order": [
        [
          -1, {
            "ascending": "{{ value(selection(step_order)) }}"
          }
        ]
      ],
      "limit": "{{ value(selection(step_limit)) }}"
    },
    "isFacet": true
  }
},
"widgets": {
  "sel_list_filter_dim": {
    "type": "ListSelector",
    "pos": {
      "x": 860,
      "y": 90,
      "w": 290,
      "h": 288
    },
    "params": {
      "step": "step_filter_dim",
      "title": "List of Products",
      "expanded": true,
      "instant": true
    }
  },
  "sel_list_filter_compound_dim": {
    "type": "ListSelector",
    "pos": {
      "x": 860,
      "y": 390,
      "w": 290,
      "h": 288
    },
    "params": {
      "step": "step_quarterly_bookings",
      "title": "List of Quarters",
      "expanded": true,
      "instant": true
    }
  },
  "sel_group": {
    "type": "PillBox",
    "pos": {
      "x": 10,
      "y": 10
    },
    "params": {
      "title": "group",
      "step": "step_group"
    }
  },
  "sel_measure": {
      "type": "PillBox",
      "pos": {
        "x": 380,
        "y": 10
      },
      "params": {
        "title": "mea",
        "step": "step_measure"
      }
    },
    "sel_order": {
      "type": "PillBox",
      "pos": {
        "x": 480,
        "y": 10
      },
      "params": {
        "title": "order",
        "step": "step_order",
        "start": true
      }
    },
    "sel_limit": {
      "type": "PillBox",
      "pos": {
        "x": 620,
        "y": 10
      },
      "params": {
        "title": "limit",
        "step": "step_limit"
      }
    },
    "widget1": {
      "type": "ChartWidget",
      "pos": {
        "x": 10,
        "y": 110,
        "w": 830,
        "h": 330
      },
      "params": {
        "chartType": "hbar",
        "step": "step_top_10"
      }
    }
  }
}
```

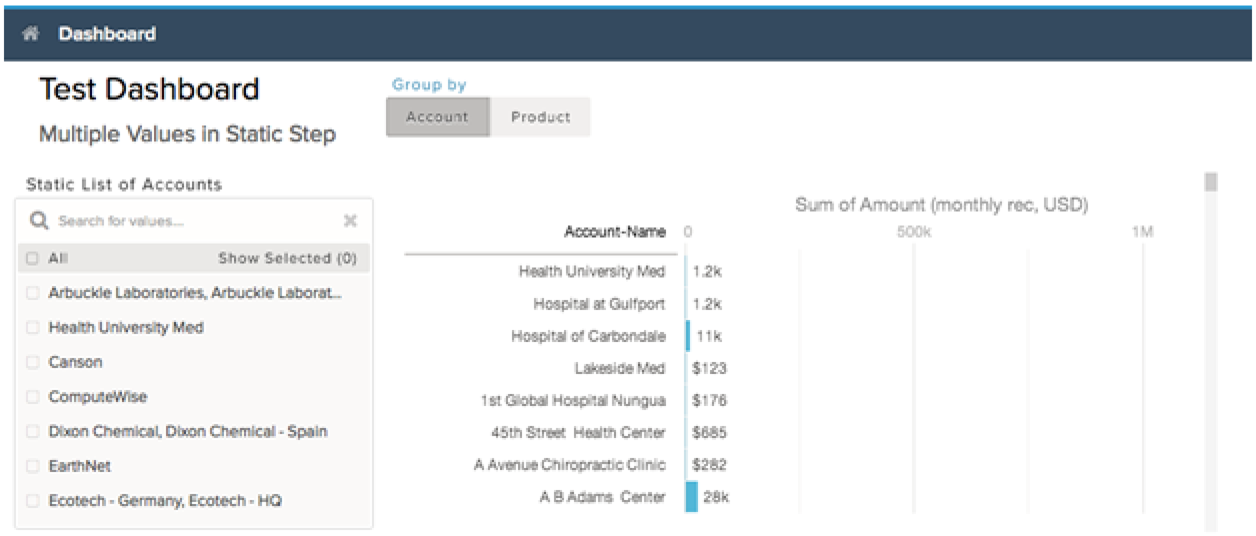
Bind a Static Filter and Group Selector to a Query

Static filters or group selectors can be bound to a query written in SAQL.

Templates are expressions, embedded in double braces ({{ }}), that get replaced with the current state of the step that they're attached to.

EDITIONS

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



For example, this dashboard contains a static filter widget that contains a list of accounts. The dashboard also contains a group selector widget that lets users indicate whether they want to group by account or product. When a user makes a selection, the chart is updated accordingly. The part of the query that controls the filtering is:

```
q = filter q by 'Account-Name' in {{ selection(step_Account_Owner_Name_2) }};
```

The step that's named `step_Account_Owner_Name_2` is configured as a selection binding so that it will pick up the current selection state. Because it's within the double braces, the value of that selection will be substituted and used in the query.

The part of the query that controls the grouping is:

```
q = group q by {{ single_quote(value(selection(step_StageName_3))) }};
q = foreach q generate {{ single_quote(value(selection(step_StageName_3))) }} as {{
value(selection(step_StageName_3)) }}, sum('Amount') as 'sum_Amount', count() as 'count'";
```

If a user selects Product in the group selector widget, the actual query that will be passed to the query engine contains:

```
q = group q by 'Product';
q = foreach q generate 'Product' as "Product", sum('Amount') as 'sum_Amount', count() as
'count';
```

Note: To view the query that's used to update the chart, open your browser's JavaScript console and type *`edge.log.query=true`*. On the dashboard, select a different group. The new query will appear in the console unless the query has been cached.

```
  "steps": {
    "step_Account_Name_1": {
      "isFacet": false,
      "query": {
        "pigql": "q = load \"opp\";\nq = filter q by 'Account-Name' in {{
selection(step_Account_Owner_Name_2) }};\nq = group q by {{
single_quote(value(selection(step_StageName_3))) }};\nq = foreach q generate {{
single_quote(value(selection(step_StageName_3))) }} as {{ value(selection(step_StageName_3))
 }}, sum('Amount') as 'sum_Amount', count() as 'count'",
        "groups": "{{ selection(step_StageName_3) }}",
        "measures": [["sum", "Amount"]]
      },
      "extra": {
        "chartType": "hbar"
      },
      "selectMode": "none",
      "useGlobal": true,
      "em": "opp",
      "type": "aggregate",
      "isGlobal": false
    },
    "step_Account_Owner_Name_2": {
      "dim": "Account-Name",
      "isFacet": false,
      "values": [
        {
          "value": ["Lakeside Med", "Hospital at Gulfport", "Hospital at Carbondale"],
          "display": "Arbuckle Laboratories, Arbuckle Laboratories - Austria, Arbuckle
Laboratories - France"
        }, {
          "value": ["Health University Med"],
          "display": "Health University Med"
        }, {
          "value": ["Canson"],
          "display": "Canson"
        }, {
          "value": ["ComputeWise"],
          "display": "ComputeWise"
        }, {
          "value": ["Dixon Chemical", "Dixon Chemical - Spain"],
          "display": "Dixon Chemical, Dixon Chemical - Spain"
        }, {
          "value": ["EarthNet"],
          "display": "EarthNet"
        }, {
          "value": ["Ecotech - Germany", "Ecotech - HQ"],
          "display": "Ecotech - Germany, Ecotech - HQ"
        }
      ],
      "selectMode": "multi",
      "useGlobal": true,
      "em": "opp",
      "type": "static",
      "isGlobal": false
    },
    "step_StageName_3": {
      "isFacet": false,
      "values": [
        {
          "value": ["Account-Name"],
          "display": "Account"
        }, {
          "value": ["Product"],
          "display": "Product"
        }
      ],
      "useGlobal": true,
      "em": "opp",
      "type": "static",
      "selectMode": "single_required",
      "isGlobal": false
    }
  }
```

Relative Dates in a Static Filter Selector

This example demonstrates how to create a static step that uses relative dates to filter another query.

EDITIONS

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

```
    "step_compare_year2": {
      "type": "static",
      "values": [
        {
          "display": "This Month",
          "value": [[["month", 0], ["month", 1]]]
        }, {
          "display": "Last Month",
          "value": [[["month", -1], ["month", 0]]]
        }, {
          "display": "2 Month Ago",
          "value": [[["month", -2], ["month", -1]]]
        }, {
          "display": "Last 2 Months",
          "value": [[["month", -2], ["month", 0]]]
        }, {
          "display": "Last Quarter",
          "value": [[["quarter", -1], ["quarter", 0]]]
        }, {
          "display": "This Quarter",
          "value": [[["quarter", 0], ["quarter", 0]]]
        }, {
          "display": "Next Quarter",
          "value": [[["quarter", 0], ["quarter", 1]]]
        }
      ],
      "start": [[[["quarter", 0], ["quarter", 0]]]],
      "selectMode": "single_required"
    },
    "step_CloseDate_Year_CloseDate_Month_CloseDate_Day_2": {
      "type": "aggregate",
      "em": "opp",
      "isFacet": true,
      "useGlobal": true,
      "query": {
        "measures": [["sum", "Amount"]],
        "groups": [["CloseDate_Year", "CloseDate_Month", "CloseDate_Day"]],
        "filters": [["Close Date", ["{{ value(selection(step_compare_year2)) }}"]]]
      }
    },
```

SAQL Overview

The SAQL language is a real-time query language that uses data flow as a means of aligning results. It enables ad hoc analysis of data that's stored in datasets.

A SAQL script consists of a sequence of statements that are made up of keywords (such as `filter`, `group`, and `order`), identifiers, literals, or special characters. Statements can span multiple lines and must end with a semicolon. SAQL is declarative, which means that you describe what you want to get from your query. Then, the query engine will decide how to efficiently serve it. SAQL is compositional. Every statement has a result, and you can chain statements together. SAQL is influenced by the Pig Latin programming language, but their implementations differ.

EDITIONS

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

IN THIS SECTION:

Keywords
Keywords are case-sensitive.

Identifiers
Identifiers are case-sensitive. They can be unquoted or quoted.

Number Literals
A number literal represents a number in your script.

String Literals
A string is a set of characters inside double quotes (").

Quoted String Escape Sequences
Strings can be escaped with the backslash character.

Special Characters

Comments
Two sequential hyphens (--) indicate the beginning of a single-line comment.

Operators

Statements

Functions

Keywords

Keywords are case-sensitive.

Keywords must be lowercase.

EDITIONS

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

Identifiers

Identifiers are case-sensitive. They can be unquoted or quoted.

Unquoted identifiers cannot be one of the reserved words and must start with a letter (A to Z or a to z) or an underscore. Subsequent characters can be letters, numbers, or underscores.

Quoted identifiers are wrapped in single quotes (') and can contain any character that a string can contain.

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

EDITIONS

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

Number Literals

A number literal represents a number in your script.

Some examples of number literals are 16 and 3.14159. You can't explicitly assign a type (for example, integer or floating point) to a number literal. Scientific E notation is not supported.

The responses to queries are in JSON, therefore the returned numeric field is a "number" class.

EDITIONS

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

String Literals

A string is a set of characters inside double quotes (").

Example:

```
"This is a string."
```

EDITIONS

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

Quoted String Escape Sequences

Strings can be escaped with the backslash character.

You can use the following string escape sequences:

Sequence	Meaning
`\b`	One backspace character
`\n`	New line
`\r`	Carriage return
`\t`	Tab
`\Z`	CTRL+Z (ASCII 26)
`\'`	One single-quote character
`\"`	One double-quote character
`\\`	One backslash character
`\0`	One ASCII null character

EDITIONS

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

Special Characters

Each of these characters has a special meaning:

Character	Name	Description
`;`	Semicolon	Used to terminate statements.
`'`	Single quote	Used to quote identifiers.
`"`	Double quote	Used to quote strings.
`()`	Parentheses	Used for function calls, to enforce precedence, for order clauses, and to group expressions. Parentheses are mandatory when defining more than one group or order field.
`[]`	Brackets	Used to denote arrays. For example, this is an array of strings: `[ "this", "is", "a", "string", "array" ]` Also used for referencing a particular member of an object. For example, `em['miles']`, which is the same as `em.miles`.
`.`	Period	Used for referencing a particular member of an object. For example, `em.miles`, which is the same as `em['miles']`.
`::`	Two colons	Used to explicitly specify the dataset that a measure or dimension belongs to, by placing it between a dataset name

EDITIONS

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

Character	Name	Description
		and a column name. It is the same as using a period (.) between names. For example: `data = foreach data generate left::airline as airline`
`..`	Two periods	Used to separate a range of values. For example: `c = filter b by "the_date" in ["2011-01-01".."2011-01-31"];`

Comments

Two sequential hyphens (--) indicate the beginning of a single-line comment.

Example:

```
a = load "myData"; --This is a comment
```

EDITIONS

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

Operators

These types of operators are available:

- Arithmetic Operators
- Comparison Operators
- String Operators
- Logical Operators

EDITIONS

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

IN THIS SECTION:

Arithmetic Operators
On the client-side query engine, if any of the operands are NULL, an arithmetic operation returns a NULL. On the server-side query engine, if any of the operands are NULL, they are treated as if they were zero.

Comparison Operators

String Operators
Use the plus sign (+) to concatenate strings.

Logical Operators

Arithmetic Operators

On the client-side query engine, if any of the operands are NULL, an arithmetic operation returns a NULL. On the server-side query engine, if any of the operands are NULL, they are treated as if they were zero.

Operator	Description
+	Plus

EDITIONS

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

Operator	Description
`-`	Minus
`*`	Multiplication
`/`	Division
`%`	Modulo

Comparison Operators

Comparisons are defined for values of the same type only. For example, strings can be compared with strings and numbers compared with numbers.

On the server-side query engine, a comparison operation with a numeric operand that is NULL returns false. A comparison operation with a string operand that is NULL treats that operand as an empty string. A boolean value is always returned.

On the client-side query engine, if any of the operands are NULL, a comparison operation returns NULL.

EDITIONS

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

Operator	Name	Description
`==`	Equals	True if the operands are equal. String comparisons that use the equals operator are case-sensitive.
`!=`	Not equals	True if the operands are not equal.
`<`	Less than	True if the left operand is less than the right operand.
`<=`	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.
`matches`	Matches	True if the left operand contains the string on the right. Wildcards and regular expressions aren't supported. For performance reasons, we recommend that you create a bigram dataset for data that you're looking for substring matches in. For example, the following query matches airport codes such as LAX, LAS, ALA, and BLA: `my_matches = filter a by origin matches "LA";`
`in`	In	If the left operand is a dimension, true if the left operand has one or more of the values in the array on the right. For example: `a1 = filter a by origin in ["ORD", "LAX", "LGA"];` If the left operand is a measure, true if the left operand is in the array on the right. You can use the `date()` function to filter by date ranges.
`in*`	In star	Used with multivalued dimensions. True for every row that has every value in an array on the right. It is a type of intersection query.

Operator	Name	Description
`not in`	Not in	True if the left operand is not equal to all of the values in an array on the right. It will include rows in which the origin key doesn't exist. For example: `a1 = filter a by origin not in ["ORD", "LAX", "LGA"];` Note: Only the client-side query engine currently supports the `not in` operator.
`not in*`	Not in star	Used with multivalued dimensions. True if the left operand doesn't contain every value in an array on the right. Note: Only the client-side query engine currently supports the `not in*` operator.

Example: Given a row for a flight with the origin "SFO" and the destination "LAX" and weather of rain and snow, here are the results for each type of "in" operator: `weather in ["rain", "wind"] = true`

```
weather in* ["rain", "wind"] = false

weather in* ["rain", "snow"] = true

weather not in ["rain", "wind"] = false

weather not in* ["rain", "wind"] = true

weather not in* ["rain", "snow"] = false
```

String Operators

Use the plus sign (+) to concatenate strings.

EDITIONS

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

Operator	Description
+	Concatenate

Example: To combine the year, month, and day into a value called `CreatedDate`: `q = foreach q generate "Id" as "Id", "Year"+"-"+"Month"+"-"+"Day" as "CreatedDate";`

Logical Operators

Logical operators can return true or false.

EDITIONS

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

Operator	Name	Description
`&& (and)`	Logical AND	True if both operands are true.
`|| (or)`	Logical OR	True if either operand is true.
`!`	Logical NOT	True if the operand is false.

Statements

Statements control the flow of data-processing tasks.

EDITIONS

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

IN THIS SECTION:

Load
Loads a dataset.

Filter
Selects rows from a dataset based on a filter condition, also called a predicate.

Foreach
Applies a set of expressions to every row in a dataset. This is often referred to as projection.

Group
Groups matched records.

Union
Combines multiple result sets into one result set.

Order
Sorts by one or more attributes.

Limit
Limits the number of results that are returned.

Offset

Load

Loads a dataset.

EDITIONS

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

After being loaded, the data is in ungrouped form. The columns are the columns of the loaded dataset.

The `load` statement uses the following syntax:

```
result = load "ContainerID/VersionID";
```

Example: The following example loads the dataset with ContainerID “0Fbxx000000002qCAA” and VersionID “0Fcxx000000002WCAQ” and assigns it to “b”: `b = load "0Fbxx000000002qCAA/0Fcxx000000002WCAQ";`

Filter

Selects rows from a dataset based on a filter condition, also called a predicate.

EDITIONS

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

A predicate is a Boolean expression that uses the available comparison operators. The filter condition is evaluated for every row. If the condition is true, the row is included in the result. Comparisons on dimensions are lexicographic, and comparisons on measures are numerical.

When a filter is applied to grouped data, the filter is applied to the rows in the group. If all member rows are filtered out, groups are completely eliminated. You can run a `filter` statement before or after `group` to filter out members of the groups.

The `filter` statement uses the following syntax:

```
result = filter rows by predicate;
```

Example: The following example returns only rows where the origin is ORD, LAX, or LGA: `a1 = filter a by origin in ["ORD", "LAX", "LGA"];`

Example: The following example returns only rows where the destination is LAX or the number of miles is greater than 1,500: `y = filter x by dest == "LAX" || miles > 1500;`

Foreach

Applies a set of expressions to every row in a dataset. This is often referred to as projection.

The `foreach` statement uses the following syntax:

EDITIONS

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

```
q = foreach q generate expression as alias[, expression as alias ...];
```

The output column names are specified with the `as` keyword. The output data is ungrouped.

Using Foreach with Ungrouped Data

When used with ungrouped data, the `foreach` statement maps the input rows to output rows. The number of rows remains the same.

Example: This example generates all carriers and the corresponding count as "flights": `a2 = foreach a1 generate carrier as carrier, miles as miles;`

Using Foreach with Grouped Data

When used with grouped data, the `foreach` statement behaves differently than it does with ungrouped data.

Fields can only be directly accessed when the value is the same for all group members, such as the fields that were used as the grouping keys. Otherwise, the members of a group can only be accessed by using aggregate functions, rather than accessing them directly. The type of the column determines which aggregate functions can be used. For example, numeric aggregates such as `count()` and `sum()` only make sense for numeric columns (measures).

Example: This example demonstrates the foreach statement being used with grouped data: `z = foreach y generate day as Day, unique(origin) as uorg, count() as n;`

Group

Groups matched records.

EDITIONS

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

Simple Grouping

The result of grouping is that one or more columns are added to the group. If data is grouped by a value that's NULL for a certain row, that whole row is removed from the result.

Syntax:

```
result = group rows by field;
```

or

```
result = group rows by (field1, field2, ...);
```

For example, to group rows by the same key:

```
a = group a by carrier;
```

You can group by multiple dimensions:

```
a = group a by (month,carrier);
```

Note: The order of the fields that you group by doesn't matter. In other words, `group a by (month,carrier)` is the same as `group a by (carrier,month)`.

Here's an alternative way to group by multiple dimensions, in separate steps:

```
a = group a by month;
```

```
a = group a by carrier;
```

Inner Cogrouping

Cogrouping means that the left and the right input are grouped independently and that the groups from the left and right are arranged side by side. Only groups that exist on both sides will appear in the results.

`inner` and `outer` are optional modifiers. If you don't specify a modifier, `inner` is used.

Syntax:

```
result = group rows by expression[, rows by expression ...];
```

You can cogroup by using multiple group clauses. The result is joined by matching the group keys:

```
a = group a by carrier, b by carrier;
```

or

```
z = group x by (day,origin), y by (day,airport);
```

Note: Cogrouping differs from joining and then grouping the result of the join.

Several grouping and cogrouping operations can be done in sequence. The groups that result from the first cogrouping are refined by the second cogrouping operation.

Example:

```
x1 = group x by destination;
```

or

```
z = group x1 by (day,origin), y by (day,airport);
```

Groups are not hierarchical. In other words, there are no groups inside groups. Therefore, repeated grouping only splits the existing groups into smaller groups, and the smaller groups appear on the same level.

If you use aggregate functions when cogrouping, you need to specify which input side to use in the aggregate function. For example, if you have an "a" side and a "b" side that both contain a particular measure, you can use syntax that resembles the following sample:

```
sum(b['myMeasure'])
```

or

```
sum(b::myMeasure)
```

If you don't specify a side, the left side will be used.

Inner cogrouping can be applied across more than two sets of data, as shown in this example:

```
result = group a by keya, b by keyb, c by keyc;
```

Outer Cogrouping

Outer cogrouping combines the groups as an outer join. For the half-matches, NULL rows are added. The grouping keys are taken from the input that provides the value.

Syntax:

```
result = group rows by expression [left | right | full], rows by expression;
```

Example:

```
z = group x by (day,origin) left, y by (day,airport);
```

Outer cogrouping can be applied across more than two sets of data. For example, to do a left outer join from a to b, with a right join to c, you can use syntax that resembles the following sample:

```
result = group a by keya left, b by keyb right, c by keyc;
```

Union

Combines multiple result sets into one result set.

The `union` statement uses the following syntax:

EDITIONS

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

```
result = union resultSetA, resultSetB [, resultSetC ...];
```

Order

Sorts by one or more attributes.

EDITIONS

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

When you use the `order` statement, it isn't applied to the whole set. Instead, the rows are operated upon individually. You can specify one or more attributes to order by. The first attribute that you specify will be used for the primary ordering and the subsequent attributes that you specify will be used to sort the groups of equal values within each previous ordering.

You can use the `order` statement with ungrouped data. You can also use the `order` statement to specify order within a group or to sort grouped data by an aggregated value.

The `order` statement uses the following syntax:

```
result = order rows by attribute [ asc | desc ][, attribute [ asc | desc ]];
```

`asc` or `desc` specifies whether the results are ordered in ascending (`asc`) or descending (`desc`) order. The default order is ascending.

Example: `q = order q by 'count' desc;`

Limit

Limits the number of results that are returned.

EDITIONS

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

Only use this statement on data that has been ordered with the order statement. The results that are returned by the `limit` statement aren't automatically ordered, and their order might change each time that statement is called. This is not a `top` or `sample` function.

You can use the `limit` statement with ungrouped data. You can also use the `limit` statement to limit grouped data by an aggregated value. For example, to find the top ten regions by revenue, you could group by region, aggregate the data using sum(revenue), order by sum(revenue) in descending order, and limit the number of results returned to the first ten results.

The expression cannot contain any columns from the input.

The `limit` statement uses the following syntax:

```
result = limit rows number;
```

Example: This example limits the number of results that are returned to 10: `b = limit a 10;`

Offset

Used to paginate values from query results. This statement must be used with data that has been ordered with the `order` statement.

EDITIONS

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

The `offset` statement uses the following syntax:

```
result = offset rows number;
```

Example: This example loads a dataset, puts the rows in descending order, and returns rows 400 to 800:

```
a = load "0Fbxx000000002qCAA/0Fcxx000000002WCAQ";
b = foreach a generate 'carrier' as 'carrier', count() as 'count';
c = order b by 'count' desc;
d = limit c 400;
e = offset d 400;
```

Functions

These types of functions are available:

- Aggregate Functions
- Date Function

EDITIONS

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

IN THIS SECTION:

Aggregate Functions

Date Function
Use the date() function to specify to the query engine that a date is represented by the three dimensions that are specified.

Aggregate Functions

Using an aggregate function on an empty set returns null. For example, if you use an aggregate function with a nonmatching column of an outer cogrouping, you might have an empty set.

This table lists the aggregate functions that are supported:

EDITIONS

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

Aggregate Function	Description
`avg()` or `average()`	Returns the average value of a numeric field. For example, to calculate the average number of miles: `a1 = group a by (origin, dest); a2 = foreach a1 generate origin as origin, dest as destination, average(miles) as miles;`
`count()`	Returns the number of rows that match the query criteria. For example, to calculate the number of carriers: `q = foreach q generate 'carrier' as 'carrier', count() as 'count';`

Aggregate Function	Description

`first()`	Returns the value for the first tuple. To work as expected, you must be aware of the sort order or know that the values of that measure are the same for all tuples in the set.
	For example, you can use these statements to compute the distance between each combination of origin and destination:
	`a1 = group a by (origin, dest);`
	`a2 = foreach a1 generate origin as origin, dest as destination, first(miles) as miles;`

`last()`	Returns the value for the last tuple.
	For example, to compute the distance between each combination of origin and destination:
	`a1 = group a by (origin, dest);`
	`a2 = foreach a1 generate origin as origin, dest as destination, last(miles) as miles;`

`min()`	Returns the minimum value of a field.
`max()`	Returns the maximum value of a field.

`sum()`	Returns the sum of a numeric field.
	`a = load "0Fbxx000000002qCAA/0Fcxx000000002WCAQ";`
	`a = filter a by dest in ["ORD", "LAX", "ATL", "DFW", "PHX", "DEN", "LGA"];`
	`a = group a by carrier;`
	`b = foreach a generate carrier as airline, sum(miles) as miles;`

`unique()`	Returns the count of unique values.
	For example, to find how many origins and destinations a carrier flies from:
	`a1 = group a by carrier;`
	`a2 = foreach a1 generate carrier as carrier, unique(origin) as origins, unique(dest) as destinations;`

Date Function

Use the date() function to specify to the query engine that a date is represented by the three dimensions that are specified.

EDITIONS

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

date()

The dimensions must be specified in the order: year, month, day, for example, `date('year', 'month', 'day')`.

Alternatively, you can use a date column name as a parameter to the date() function, for example, `date(CreatedDate)`. In this example, during digestion, the CreatedDate is broken down into CreatedDate Year, CreatedDate Month, and CreatedDate Day. For this reason, when using the date() function, you can specify one dimension as a prefix, and the engine will automatically append the suffixes: year, month, and day.

Specify a Date Range

When filtering using the `in` operator, you can specify an array that contains a date range. For example:

- `a = filter a by date('year', 'month', 'day') in ["1970/1/1".."1970/1/11"];`

Previously, there was a dateRange() function that took two parameters. The first parameter is an array that specifies the first date in the range. The second parameter specifies the end of the range. The dates must be specified in the order: year, month, day. For example:

- `a = filter a by date('year', 'month', 'day') in [dateRange([1970, 1, 1], [1970, 1, 11])];`

Specify a Relative Date Range

When filtering using the `in` operator, you can specify an array that uses relative date keywords to define a range. For example:

- `a = filter a by date('year', 'month', 'day') in ["1 year ago".."current year"];`
- `a = filter a by date('year', 'month', 'day') in ["2 quarters ago".."2 quarters ahead"];`
- `a = filter a by date('year', 'month', 'day') in ["4 months ago".."1 year ahead"];`

The relative date keywords are:

- current day
- n day(s) ago
- n day(s) ahead
- current week
- n week(s) ago
- n week(s) ahead
- current month
- n month(s) ago
- n month(s) ahead
- current quarter
- n quarter(s) ago
- n quarter(s) ahead
- current year

- n year(s) ago
- n year(s) ahead

Analytics Cloud Limits

EDITIONS

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

This section describes Analytics Cloud limits.

The following limits apply to all supported editions. Contact salesforce.com to extend the limits.

API Call Limits

Limit	Value
Maximum concurrent Analytics Cloud API calls per organization	100
Maximum Analytics Cloud API calls per user per hour	5,000

Data Storage Limits

Limit	Value
Maximum rows in all datasets	250 million. This limit excludes rows in datasets that aren't registered.

Dataflow Job Limits

Limit	Value
Maximum dataflow runs per day	24
Maximum concurrent dataflow runs	1

External Data Limits

Limit	Value
Maximum file size per external data upload	512 MB. If you upload multiple external data files simultaneously, the total file size of all files cannot exceed this limit.
Maximum external data files uploaded per day	20 files

Query Limits

Limit	Value
Maximum concurrent queries per organization	50
Maximum concurrent queries per user	10

Limit	Value
Query timeout	2 minutes

INDEX

! 70
&& 70
|| 70

A

adding data 16
Aggregate functions 76
and 70
API
  limits 79
Arithmetic operators 68
augment operation
  name-value pairs 34
  syntax 34
Augment transformation
  description 32
  join condition 33
  operation 34
Average 76
avg 76

B

Bindings 56
Builder permission set license
  description 4

C

Comments 68
Comparison operators 69
configuring the upload 15
connecting to Salesforce 15
count 76

D

Dashboard JSON 44–45
data integration
  extract phase 6
  guidelines 6
  load phase 6
  process 6
  transformation phase 6
data storage
  limits 79
dataflow
  configuring 28
  definition file 24
dataflow *(continued)*
  description 23
  job management 29
  sample 25
dataflow definition
  sample 25
dataflow jobs
  description 30
  limits 79
  monitoring 30
  rescheduling 29
  running 29
  stopping 29
dataflow manager
  dataflow jobs 30
datasets
  joining 33
  optimizing search 39
  registering 40
Date functions 78
date() 78
dateRange() 78
dateRelative() 78
distinct 76

E

Enabling 2
Enums
  InsightsExternalDataAction 22
  InsightsExternalDataFormat 22
  InsightsExternalDataOperation 22
  InsightsExternalDataStatus 23
Explorer permission set license
  description 4
external data
  limits 79

F

filter 71
first 76
flatten operation
  name-value pairs 38
  syntax 38
Flatten transformation
  description 35
  example 36

Flatten transformation *(continued)*
  operation 38
foreach 72
Functions 76

G
group 73
Group selector 62, 64

I
Identifiers 66
in 69
InsightsExternal Data object 17
InsightsExternalDataAction enum 22
InsightsExternalDataFormat enum 22
InsightsExternalDataOperation enum 22
InsightsExternalDataPart object 20
InsightsExternalDataStatus enum 23

J
join condition
  composite key 33
  single-column key 33
joins
  datasets 33

K
Keywords 66

L
last 76
Licenses
  permission set licenses 3
limit 75
limits
  API calls 79
  data storage 79
  dataflow jobs 79
  external data 79
  queries 79
load 71
Logical operators 70

M
managing the upload 17
matches 69
max 76
metadata file
  overview 8

metadata file *(continued)*
  reference 10
min 76

N
ngram operation
  name-value pairs 40
  syntax 40
Ngram transformation
  description 39
  operation 40
not in 69
Number literals 66

O
Objects
  InsightsExternalData object 17
  InsightsExternalDataPart object 20
offset 75
operations
  augment operation 34
  flatten operation 38
  ngram operation 40
  sfdcDigest operation 42
  sfdcRegister operation 41
Operators 68
or 70
order 75
Overview 1

P
permission set licenses
  for Analytics Cloud 4
Permission sets
  assigning to multiple users 4
  cloning 3
  creating 3

Q
queries
  limits 79
Query 48
Query example 51

R
Register transformation
  description 40
  operation 41
Relative dates 64

S

Salesforce Extract transformation
- description 42
- operation 42

SAQL 65
Selection binding 57
Setup
- process 1

sfdcDigest operation
- name-value pairs 42
- syntax 42

sfdcRegister operation
- name-value pairs 41
- syntax 41

Special characters 67
Statements 71
Static filters 62
Static steps 54
Steps 47
String concatenation 70
String escaping 67
String literals 66
sum 76

T

transformations
- Augment transformation 32
- description 32
- Flatten transformation 35
- Ngram transformation 39
- Register transformation 40
- run-time monitoring 30
- Salesforce Extract transformation 42

U

union 74
unique 76
user permissions
- list 5

User permissions 45
Users
- permission set licenses 3
- permission sets, assigning to multiple users 4

W

Widgets 48