# 1 | Prism Analytics

## 1.1 | Steps: Set Up Tenant for Prism Analytics

### Prerequisites

You need to create a Tenant Management case with Workday and create a Tenant - Setup Feature request to provision Prism Analytics hardware for your tenant.

Security: *Security Configuration* domain in the System functional area.

### Context

Set up Prism Analytics to enable data administrators and data analysts to create tables and datasets for blending Workday and non-Workday data that data analysts can analyze inside Workday.

### Steps

1. Create User-Based Security Groups.
   Create security groups for your data administrators and data analysts, and assign users. Use these security group types:
   - *Unconstrained groups*
   - *Role-based (constrained) groups*

   Note: When you create a role-based (constrained) security group for any of the dataset-related roles, you must select Role has access to the positions they support for the Access Rights to Multiple Job Workers option.

   Example: You can define these security groups (administered by the Security Configurator group):

| Group Name | Group Type |
|---|---|
| Prism Data Writer | User-Based |
| Prism Data Administrator | User-Based |

2. Access the Maintain Functional Areas task.

   Select the Enabled check box for the Prism Analytics functional area.

3. Edit Domain Security Policies.

   Create or edit a security policy for these domains:

   - *Prism Datasets: Create*
   - *Prism Datasets: Manage*
   - *Prism Datasets: Owner Manage*
   - *Prism Datasets: Publish*
   - *Prism: Manage Data Source*
   - *Prism: Manage Relax Sharing*
   - *Prism: Tables Create*
   - *Prism: Tables Manage*
   - *Prism: Tables Owner Manage*

   Workday suggests that you use the security groups you created along with the Security Configurator group.

| Domain | Security Groups | Task Permissions |
|---|---|---|
| *Prism Datasets: Create* | Prism Data Administrator, Prism Data Writer<br>System Auditor | View and Modify<br>View only |
| *Prism Datasets: Manage* | Prism Data Administrator<br>Security Configurator, Security Administrator, System Auditor | View and Modify<br>View only |
| *Prism Datasets: Owner Manage* | Prism Data Administrator, Security Administrator<br>System Auditor | View and Modify<br>View only |
| *Prism Datasets: Publish* | Prism Data Administrator, Security Configurator<br>System Auditor | View and Modify<br>View only |
| *Prism: Manage Data Source* | Security Configurator<br>System Auditor | View and Modify<br>View only |

| Domain | Security Groups | Task Permissions |
|---|---|---|
| *Prism: Manage Relax Sharing* | Prism Data Administrator<br>System Auditor | View and Modify<br>View only |
| *Prism: Tables Create* | Prism Data Administrator, Prism Data Writer<br>System Auditor | View and Modify<br>View only |
| *Prism: Tables Manage* | Prism Data Administrator<br>Security Configurator, Security Administrator, System Auditor | View and Modify<br>View only |
| *Prism: Tables Owner Manage* | Prism Data Administrator, Security Administrator<br>System Auditor | View and Modify<br>View only |

4. Activate Pending Security Policy Changes.
5. Set Up Dataset Sharing.
6. Set Up Table Sharing.
7. (Optional) Enable Contextual Publishing for Datasets.
8. (Optional) Customize notification settings.
    a. Access the Edit Tenant Setup - Notifications task.
    b. Select the System notification group in the Notification Delivery Settings section.
    c. Customize notification settings for Prism Data Acquisition Notification and Prism Wbucket Complete Notification.

See: Reference: Edit Tenant Setup - Notifications.

**Related Information**

**Concepts**
Concept: Security in Prism Analytics
**Reference**
Community article: How to enable tenants for Prism Analytics

## 1.2 | Concept: Prism Analytics Data Management Workflow

With Workday Prism Analytics, you can analyze your Workday and non-Workday data together without having to export it into a separate data warehouse and BI (business intelligence) application. This makes analysis faster, easier, more secure, and performed in a location where users can take action on it.

What are the steps involved in going from raw data (both internal and external to Workday) to visualizations and reports in Workday? What skills do you need to perform each step? This section explains each phase of the data workflow from transforming data to analyzing it.

### Phase 1: Create Tables and Base Datasets to Bring in Data

The first phase in the data management workflow is to bring data into the Prism Analytics Data Catalog. You can bring in external data or Workday transactional data. The Data Catalog is where data analysts can see what data is available to them.

You bring in data by creating a table or a base dataset. A table is a Prism Analytics object that stores (materializes) data and represents it in a tabular format. A table has a schema and contains data that's valid against the schema. A base dataset is a Prism Analytics object that controls some underlying data stored in 1 or more files, and describes processing logic to read the data in the files. A base dataset contains all of the information about the data in the files, plus a subset of example rows to help you understand the data values.

Example: You might create a table and load external operational data into it using the REST API.

Example: You might create a base dataset that brings in data from an external server on a recurring basis.

If you're familiar with ETL workflows (extract, transform, and load), tables and base datasets encompass the extract logic.

**Who does this phase?** Data Administrators or Data Analysts.

### Phase 2: Create Derived Datasets to Transform the Data

After you've brought in data into the Data Catalog, you create derived datasets to transform, enrich, and join the data. Create a derived dataset to describe the processing logic that you need to prepare the data for analysis.

Derived datasets contain information on how to process, blend, and transform the data you import into them. Example: You might create a derived dataset that describes how to join a table and a base dataset together, or how to calculate point of sales revenue per store.

From the perspective of an ETL workflow, the derived dataset encompasses the transform logic.

**Who does this phase?** Data Administrators or Data Analysts.

### Phase 3: Apply Security to the Data

Workday offers a strong, flexible, configurable security model that applies to all Workday objects. Data outside of Workday doesn't have any security applied to it. But by bringing external data into the Data Catalog, you can take advantage of the strong security model that Workday offers.

The next phase in the data workflow is to apply security to the data in a dataset using the existing Workday security domains. You apply security to the data by editing the data source security on the dataset that has all the data you need to analyze.

When you first create a dataset, no security is configured in the data source security. However, Workday implicitly applies security to the data when no data source security is configured.

Workday applies the security restrictions to the published data in the form of a Prism data source (see next phase).

**Who does this phase?** Security Administrators or Data Administrators.

### Phase 4: Make the Data Available for Analysis (Publish)

The way you make data available for analysis is by publishing a dataset. When you publish a dataset, Workday creates the Prism data source, loads it with the transformed data, and applies the appropriate security restrictions to the data.

Workday applies the security domains configured in the data source security on the dataset, or it applies the *Prism: Default to Dataset Access* security domain if no data source security was configured. The *Prism: Default to Dataset Access* domain provides contextual access to a Prism data source based on your access to the underlying dataset.

You can use Prism data sources in visualizations and reports like any Workday delivered data source.

From the perspective of an ETL workflow, publishing a dataset is the load part of the process.

**Who does this phase?** Data Administrators or Data Analysts.

### Phase 5: Analyze and Visualize the Data

After a Prism data source is available, you can use it to create visualizations and reports.

**Who does this phase?** Report Writers.

#### Related Information

**Concepts**
Concept: Security in Prism Analytics
**Tasks**
Steps: Create a Table Manually
Steps: Create a Dataset with External Data (SFTP Server)
Steps: Create a Dataset with External Data (Upload a File)
Steps: Create a Dataset Using Workday Data
Steps: Create a Derived Dataset
Edit Prism Data Source Security in a Dataset
Publish a Dataset as a Prism Data Source Manually
Create Dataset Publish Schedules
**Reference**
The Next Level: Prism Analytics Community Guide

## 1.3 | Concept: Creating Reports to Import into Datasets

To create a dataset from a Workday report, the report must:

- Be an advanced report. Datasets don't support importing other report types, such as matrix or composite.
- Be enabled as a web service.
- Be enabled for Prism Analytics. Select the Enable for Prism check box in the Advanced tab of the custom report.
- Not include any fields from a related business object that have a many to 1 relationship with the primary business object.

Note: If your custom report includes a Currency field, you must select the Show Currency Column check box in the **Field Options** column for the Currency field, located in the Columns tab of the report.

To optimize performance, consider these options:

- Use indexed data sources whenever possible.
- Use data sources that provide the smallest possible set of data that meets your needs. Example: If you're interested in compensation-related transactions, use employee compensation events instead of all business process transactions.

## 1.4 | Concept: Prism Data Sources

A Prism data source is a type of Workday data source that gets created when you publish a dataset. A dataset is a user-defined object in the Prism Analytics Data Catalog that describes transformed data.

When you publish a dataset, Workday creates a Prism data source and loads it with the transformed data as configured in the dataset. Typically, datasets blend together Workday and non-Workday data.

Prism data sources are different than Workday-delivered data sources in several ways. Prism data sources:

- Are created by a colleague at your organization.
- Have their own primary business object that isn't linked to any other business object in Workday.
- Can have their data removed. When someone removes the data in a Prism data source, all reports that use it will be broken until a Prism data analyst populates the Prism data source again.
- Can be deleted from your tenant by unpublishing the corresponding dataset.
- Only support some report types.
- Only support some field types.
- Only support some calculated fields. Some of the supported calculated fields provide limited functionality.

Prism data sources also have their own security. Security group access controls who can see a Prism data source.

With the appropriate permissions, you can:

- Create a visualization in a discovery board using the Prism data source.
- Create a custom report using the Prism data source.
- View a Prism data source by accessing the View Prism Data Source report.
- Delete the data (rows) in a Prism data source from the View Prism Data Source report by selecting Actions > Prism Data Source > Delete Published Rows.
- Remove the Prism data source from Workday by unpublishing the dataset that created the Prism data source.

### Supported Custom Reports on Prism Data Sources

You can create these types of custom reports using a Prism data source:

- **Advanced.** Some features aren't supported, such as Subfilters. Workday only displays features that it supports. Also, only add a field from a related business object when necessary. Fields from related business objects might impact report performance.
- **Composite.** The Prism data source must include at least 1 Instance field.
- **Matrix.** Most features are supported. Lookup Prior Value isn't supported.
- **Simple.** All features are supported.
- **Transposed.** All features are supported.

### Supported Calculated Fields on Prism Data Sources

You can create these calculated fields using Prism data sources:

- Arithmetic Calculation
- Build Date
- Concatenate Text
- Convert Text To Number
- Date Constant
- Date Difference
- Evaluate Expression
- Format Date
- Format Number
- Format Text
- Increment or Decrement Date
- Lookup Date Rollup
- Numeric Constant
- Prompt For Value
- Substring Text
- Text Constant
- Text Length
- True/False Condition

Workday only makes available calculated fields that are supported for Prism data sources.

There are some limitations when creating some calculated fields:

- Calculated fields only work on supported field types. Example: date-related functions work only on fields of type Date.
- You can't use Instance or Multi-Instance fields in some calculated fields, such as Format Text and Substring Text. This is because Instance and Multi-Instance fields in Prism data sources only include the unique identifier information (also known as a WID), not the display name.

#### Related Information

**Concepts**
Concept: Prism Analytics Data Management Workflow
Concept: Deleting Prism Data
**Tasks**
Unpublish a Dataset
Delete Rows from a Prism Data Source
**Reference**
The Next Level: Prism Analytics Community Guide

## 1.5 | Table and Dataset Concepts

### 1.5.1 | Concept: Tables

A table is a Workday Prism Analytics object that stores (materializes) data and represents it in a tabular format. A table has a user-defined schema and only contains data that's valid against the schema. The data in tables is backed by a distributed columnar data store.

You create tables to bring in data from multiple sources and store it in a central location, the Data Catalog (similar to a data warehouse). You can then join, transform, blend, and enrich table data using derived datasets based on the table. Use derived datasets to prepare data for analysis.

## Tables Compared to Base Datasets

A table is similar to a base dataset with important differences.

| Concept | Tables | Base Datasets |
|---|---|---|
| Schema | You define the table schema before you load data into the table. If you're familiar with databases, this is commonly referred to as schema-on-write.<br><br>You can still change the table schema, but there are requirements and limitations. Example: You can add/remove fields at any time, but if you want to change the field type, the table must be empty (zero rows). Schema changes are destructive. Example: if you remove a field, you lose all data contained in the field. | Base datasets control underlying data stored in files. Dataset schemas describe how to read (and later transform) the data stored in those files. If you're familiar with databases, this is commonly referred to as schema-on-read.<br><br>Although you can change the dataset schema at any time, you must make sure that the schema matches the data in the underlying files so that it recognizes the data correctly. If the schema doesn't match the file, then Prism sets value as null. |
| Source type | Tables can accept data from any type of source at any time, such as file upload or REST API. Currently, you can only load data into a table using the REST API.<br><br>Example: You create a table and load data into it using file upload, and then later you load data into it using the REST API. | A base dataset only accepts data from the same source type you used when you created the dataset. That is, if you create a dataset using SFTP, it will only accept data from an SFTP server. |
| Data validation | When you load data into a table, Workday validates the data against the defined schema.<br><br>If the value for a field doesn't match the field type or other field parameters (such as date format), then Workday marks the entire row as invalid and doesn't include the row in the table. Instead, the row is sent to an error file that you can download. | When you publish a dataset, Workday reads the data stored on disk and validates it against the current schema of the base dataset.<br><br>If the value for a field doesn't match the field type or other field parameters (such as date format), then Workday marks that field value as NULL and includes the row in the published Prism data source. |
| NULL handling | Every field allows NULL values.<br><br>Workday distinguishes between NULL values and empty string values in Text fields when reading a delimited source file to load data into a table.<br><br>If a delimited file contains:<br>• 2 consecutive field delimiters only, then Workday treats the field value as NULL. Example: `"Smith",,"Tom"`<br>• 2 consecutive field delimiters with the 2 consecutive quote characters in between, then Workday treats the field value as an empty string. Example: `"Smith","","Tom"` | Every field allows NULL values.<br><br>Workday doesn't distinguish between NULL values and empty string values in delimited files. |
| Name restrictions | Tables have 2 names:<br>• Name. This is a display name that displays in the Data Catalog. You can change the display name at any time.<br>• API Name. This is a unique name used to reference the table in the API. You can't change the table API name after you create the table. API names must be unique and conform to the name validation rules.<br><br>Table fields also have a display name and an API Name. You can change the display name at any time. However, you can't change the API name after you create the field and save the table. | Base (and derived) datasets only allow 1 name and it must be unique. You can't change the dataset name.<br><br>Dataset fields only allow 1 name and you can change it at any time. But you must ensure to fix any downstream errors that might result from changing the field name. |

| Concept | Tables | Base Datasets |
|---|---|---|
| Deleting data | You can remove all rows (truncate) or some rows (delete) from a table.<br><br>To delete some rows from a table, you delete all rows from a single load from the table. You might want to do this when you append data to the table and you need to remove the rows added from a particular load. | You can remove all rows (truncate) from a base dataset. You can't delete a subset of rows. |
| Publishing | You can't publish a table. If you need to analyze the table data, create a derived dataset using the table as the source and then publish the dataset. | You can publish any base or derived dataset. |
| Row count | Workday knows exactly how many rows of data exist in a table, and it displays the number of rows in the Data Catalog and on the View Table report. | Workday doesn't know how many rows of data exist in a base dataset. |

### Related Information

**Concepts**
Concept: Datasets
**Reference**
2020R1 What's New Post: Prism Analytics Tables

### 1.5.2 | Concept: Table Error File

When you load data into a table, Workday validates the data against the defined schema.

If the value for a field doesn't match the field type or other field attributes (such as date format), then Workday marks the entire row as invalid and doesn't include the row in the table. Instead, Workday sends the row to an error file that you can download.

Use the error file to get a list of all rows that failed to load into the table. You can fix the errors in the data, remove the extra fields that Workday adds, and load the fixed data into the table.

The error file:

- Is a CSV file.
- Includes all fields defined in the table schema plus fields for troubleshooting:
  - Error Code
  - Error Message
- Includes all failed rows up to a maximum of 10,000 rows. If there are more than 10,000 error rows, then Workday rejects the load with a status of Failed.

To download the error file, access the View Table report for the table, and click the Activities tab. Click the download icon for a data load that included some errors.

### Related Information

**Reference**
Reference: Table Error File Error Codes

### 1.5.3 | Concept: Datasets

A dataset is a Prism Analytics object that controls some underlying data and describes some processing logic to manipulate the data. A dataset is a description of the data, otherwise known as metadata. It contains all of the data about the data, plus a subset of example rows to help you understand the data values. You create datasets to prepare data for analysis.

A dataset can describe either Workday or non-Workday (external) data. You might want to create a dataset using external data to blend, transform, and enrich it with Workday data. This enables you to analyze your Workday and non-Workday data together without having to export it into a separate electronic data warehouse and business intelligence (BI) application.

A dataset can also describe data that's output from another table or dataset by deriving the dataset from an existing table or dataset.

You can create a:

- Base dataset. A base dataset is a dataset that is based on 1 or more source files. The source files can come from external sources (non-Workday data) or from the output of a Workday custom report. You create a base dataset when you create a new dataset using these options:
  - from File. This option creates a dataset with external data that you upload in the browser.
  - from SFTP. This option creates a dataset with external data that Workday retrieves from a SFTP server using an integration.
  - from Custom Report. This option creates a dataset with Workday data.
- Derived dataset. A derived dataset is a dataset that is based on 1 or more existing tables or datasets. The source data of a derived dataset comes from the output of existing tables and datasets. You use derived datasets to blend data together from different sources, such as Workday data and non-Workday data. Some stage types, such as Join and Union, are only available to derived datasets. You create a derived dataset when you create a new dataset using the Derived Dataset option.

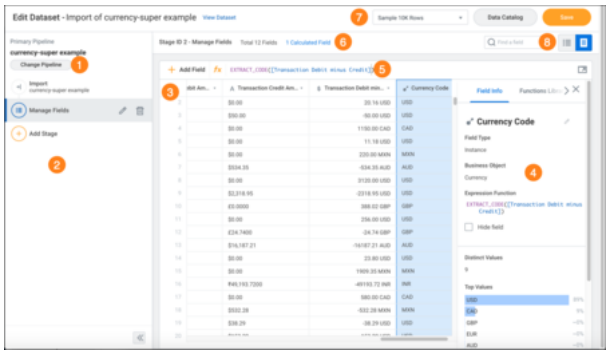
### 1.5.4 | Concept: Dataset Workspace

The Edit Dataset task is where you make changes to the dataset definition to manipulate data. This task acts like a single page application. This means you can interact with elements on the task dynamically as if it were its own application. You can also make all the changes you want, such as adding Prism calculated fields and stages. When you're done, click Save to save all changes to the dataset.

If you have permission to edit a dataset, you can access the Edit Dataset task using these methods:

- Right-click the dataset name on the Data Catalog report and select Edit.
- Click Edit on the View Dataset report.
- Access the Edit Dataset task and select the dataset name you want to edit.
- When creating a dataset for the first time, the workflow leads you to the Edit Dataset task.

When you view the Edit Dataset task, you see these components:




1. Pipeline list. (Derived datasets only.) Click Change Pipeline to view a collapsible panel that lists all tables and datasets that you've imported into the derived dataset. Importing a table or dataset creates a new pipeline. You can also use this panel to add a new pipeline by importing an additional table or dataset. When you select an item in the pipeline list, the pipeline details panel displays the details for that pipeline.
2. Pipeline details panel. This panel displays every stage in the dataset pipeline, starting with the first stage that created the pipeline. For base datasets, that can be an Import or Parse stage depending on where the source data comes from. For derived datasets, it's an Import stage. Use this panel to add, edit, and delete stages. You can also manage dataset fields from this panel. You can collapse this panel to increase the available space in the example data table.
3. Example data table. The example data table takes up most of the space on the Edit Dataset task. It displays the current view of the data (records and fields) for the output of the currently selected pipeline stage.
4. Inspector panel. This panel displays when you select a field in the example data table. You can hide this panel to increase the available space in the example data table. This panel has these tabs:
    - Field Info. This tab displays detailed information about the selected field, including statistics on the values in the field. All statistics are based on the data currently shown in the example data table. Therefore, to get more precise numbers, increase the number of example rows.
    - Functions Library. This tab displays the functions you can use in a Prism calculated field expression, including description, syntax, and an example. You can search for a specific function. You can click the + icon next to the function name to insert the function at the current location in the Prism calculated field expression.
5. Prism calculated field expression bar. Click Add field to add a new Prism calculated field, and then enter the field expression. You can:
    - Use the expression bar later to edit the field expression.
    - Expand the expression builder to create and view multiline expressions.
    - Edit the field name in the inspector panel.
6. Stage statistics and search bar. For a selected stage, you can see the number of:
    - Fields
    - Prism calculated fields
    - Field-related errors

    You can see the ID for stages. The stage ID is a unique ID that Workday assigns to the stage based on the order you added the stage to the dataset. For each stage you add, the stage ID increases.
7. Example data controls. Use this menu to change the number of example rows to display in the dataset by selecting a new value.
8. Table and list view. You can view the example data table as a table or a list. The field list view navigator enables you to see distinct, null, median, and top values at a glance. When you edit a new dataset, the default view is table view. Each time you change the view, the last view you select becomes the default view.

### Related Information

**Concepts**
Concept: Dataset Stages
Concept: Dataset Pipelines
Concept: Prism Expression Language
**Tasks**
Change the Dataset Example Rows
**Reference**
W33 What's New Post: Field List Navigator

## 1.5.5 | Concept: Data Catalog

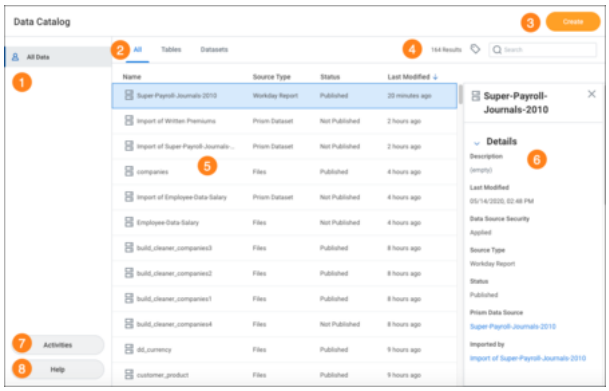
The Data Catalog report is your starting point for using Workday Prism Analytics. The report displays the data available to you. If you have permission to create datasets or tables, you can:

- Bring in and store data from multiple sources.
- Create derived datasets to transform the data.

You can also:

- Customize your view of datasets by filtering them.
- See details, such as a description, tags, and fields associated with a dataset or table while remaining in the Data Catalog.
- Access details about publishing and Prism usage.
- Access documentation that supports your use of Prism Analytics.

When you view the Data Catalog report, you see these components:




1. Data Catalog details panel. This panel displays how you view your data. The current selection enables you to view all your data in a tabular format in the main Data Catalog view.
2. Object type view. In the main Data Catalog view, select the tab for datasets, tables, or both (All). In the Datasets tab, you can filter your view of each column. Example: View only published datasets.
3. Data import options. Click Create to select how to bring in your data. You can create a table, base dataset, or derived dataset.
4. Tag filter. Click the tag icon to filter your datasets and tables by tags.
   You can add or edit tags to organize your data if you have editor permission when you:
   - Create a dataset by file upload.
   - Create a derived dataset.
   - Create a table.
   - View a dataset or table.
   - Edit a table.

   Note: Tags you create are visible to all.

   If you remove a tag from a dataset or table, the tag still displays in the tag menu if another dataset or table uses the tag.

   If you change browsers or laptops, the tag filters you select in the Data Catalog report won't persist.

   Workday doesn't store tag names but rather a randomly generated identifier for each tag. This is consistent with how Workday stores other filters in Data Catalog.
5. Main Data Catalog view. This view displays all the data that you have permission to view. Right-clicking a dataset displays a list of actions you can perform, such as viewing the dataset in a new tab.
6. Inspector panel. This panel displays when you select a dataset or table. You can hide this panel to increase the available space in the main Data Catalog view. This panel displays detailed information about the selected dataset or table, such as the datasets that imported the dataset or table.

   You can also view tags assigned to a dataset or table from the inspector panel. Clicking a tag in the panel displays in the Data Catalog the datasets or tables filtered by the tag.
7. Activities button. Click Activities to view publishing-related activities and how much Prism data your organization has used in your tenant.
8. Help button. Click Help to download a PDF of the Prism Analytics Data Management User Guide (Administrator Guide). The guide is available in an online format in the Workday Community if you have access to Community.

### 1.5.6 | Concept: Dataset Schema Changes

Schemas are the structures behind how your data is organized. A dataset might consist of multiple schemas that you define when creating the dataset. Each of these dataset components has its own schema:

- The files containing external data that you upload into a dataset.
- The custom report you create to bring Workday data into a dataset.
- The input to a pipeline stage. For the first stage in a base dataset, this schema is determined by the dataset source.
- The output from a pipeline stage. For the last stage in the Primary Pipeline, this schema determines the output schema of the entire dataset.

Schema changes can happen anywhere in a dataset. Schemas change when you add new fields or remove existing fields, such as when you:

- Add a Prism calculated field.
- Hide or expose a field.
- Add a stage that adds or deletes fields in that stage.

Schemas can also change when you:

- Upload an external file.
- Import a file from an SFTP server.
- Import data from a custom report.

Where changes occur in a dataset can affect other dataset components. Changes to dataset schemas can break components that were based on the original schema. Where changes occur can also affect how you manage changes.

Note: When you change the schema and Workday imports the new source file into the dataset, you must open the dataset and save the changes. If you don't save the dataset, Workday continues to use the old schema definition. The next time you publish the dataset, you might get inconsistent data in the Prism data source.

When you import data that changes the schema of your dataset from an SFTP server or custom report, you must manually edit the dataset to incorporate the changes. If new fields are added to the dataset, Workday hides them. Expose the new fields and save the dataset to include the fields in the dataset schema.

Workday recommends that you ensure that the dataset is up to date and includes the fields you want. In most cases, you use the Manage Fields stage to manage dataset schema changes.

## Manage Fields

The Manage Fields stage uses the output of the previous stage as a baseline from which to monitor changes. When this baseline changes, Workday warns you in the pipeline details panel. If the baseline changes, in the Manage Fields stage, Workday:

- Displays all new and removed fields.
- Doesn't yet include all changes in the dataset schema.
- Displays no data in the example table.

Workday recommends that you:

- Add a Manage Fields stage at the beginning of the Primary Pipeline of a derived dataset when you want to monitor the schema of the table or dataset from which it's derived.
- Add a Manage Fields stage at the end of the Primary Pipeline of a dataset that you intend to publish. This enables you to detect any schema changes that might break reports that use the Prism data source of this published dataset.
- Add a Manage Fields stage at the end of a pipeline when you need to hide fields or expose new fields.
- Add a Manage Fields stage in a base dataset to ensure that no future integration can unintentionally remove an existing field.
- Include no more than 2 Manage Fields stages in a single pipeline.

In some cases, Workday handles schema changes without using a Manage Fields stage. Example: When uploading a new version of a source file into a dataset, Workday handles schema changes based on the source file header row:

| Source File Contains Header Row | Result |
|---|---|
| Yes | Workday uses the field names in the header row of the dataset to determine which fields are in the new source file.<br><br>You can add or delete fields anywhere in the source file. If the field names in the source file don't change, Workday updates the schema of the dataset into which you import the new source file. |
| No | Workday handles changes that occur only at the end of the source file.<br><br>You can only add new fields at the end of the source file. If you do so, Workday updates the schema of the dataset into which you're importing the new source file. |

### Related Information

**Tasks**
Manage Dataset Fields
Steps: Create a Dataset with External Data (SFTP Server)
Steps: Create a Dataset Using Workday Data

### 1.5.7 | Concept: Dataset Pipelines

Datasets contain 1 or more pipelines. A pipeline is a container of stages that models the flow of how data should be transformed. It consists of an ordered list of stages, each of which define how to modify the data at that point in the pipeline. Pipelines can contain 1 or more stages.

Base datasets contain 1 pipeline, and derived datasets can contain 1 or more pipelines. Every dataset has a Primary Pipeline.

The first stage in a pipeline brings in data from the dataset source. Stages listed after the first stage in a pipeline take the output of the previous pipeline as the input to the current stage. If you're familiar with ETL workflows (extract, transform, and load), each stage is 1 step in a development pipeline.

The last stage of any pipeline is the output for that pipeline. The output of the Primary Pipeline is the output of the entire dataset. Therefore, when you publish a dataset, the output of the Primary Pipeline will be materialized as the data in the Prism data source.

### Related Information

**Tasks**
Add a Stage to a Dataset

### 1.5.8 | Concept: Dataset Stages

A stage is a pipeline object that takes in data, transforms it in some way, and outputs modified data. A stage performs a single computational function, such as joining data with a Join stage or parsing data from the dataset source with a Parse stage.

Prism Analytics supports these stage types: Import, Parse, Manage Fields, Filter, Group By, Join, and Union.

Adding a stage can change the number of records and fields in the dataset.

Deleting a Union or Join stage disconnects but keeps the stage's pipelines. If you don't want to keep the disconnected pipelines, delete or use them in another Join or Union stage in the Primary Pipeline.

You can't delete or add an Import stage or a Parse stage.

#### Related Information

**Tasks**
Add a Stage to a Dataset
**Reference**
Reference: Dataset Stages

### 1.5.9 | Concept: Table and Dataset Field Types

Each table and dataset field has a field type attribute. The field type is often referred to as a data type in other data applications.

The field type defines what kind of values the field can hold. The field type also determines which functions can use the field as an argument. Example: The CONCAT function only accepts Text fields as arguments.

When you create a table, you define the field type of each field. When you load data into a table, Workday validates the data against the defined schema. If the value for a field doesn't match the field type or other field parameters (such as date format), then Workday marks the entire row as invalid and doesn't include the row in the table. Instead, the row is sent to an error file that you can download.

When you create a dataset using external data, Workday attempts to guess the field type of each field by examining some of the data. However, you can change the field type if Workday assigned the wrong field type or if you want to apply a different field type.

When you create a dataset from a custom report, Workday maps the Workday field types to dataset field types.

For Prism calculated fields, the expression result determines the field type for that field. If a Prism calculated field expression is `TO_INT(zipcode)`, then the field type for that field is Integer.

Note: Workday recommends using the Numeric field type in datasets where possible. When you use a different numeric field type, such as Double or Integer, you risk losing precision and getting erroneous results depending on the data and calculations in the dataset. In base datasets, set the field type as Numeric(x,y). You can also change the field type in a derived dataset using the CAST function. Example: `CAST([salary] AS decimal(12,4))`

## Dataset Field Types

Datasets use these field types:

| Dataset Field Type | Description | Range of Values |
|---|---|---|
| Boolean | A Workday specific field type that contains boolean values. | `True`, `False` |
| Currency | A variable length decimal value that supports a maximum of 20 digits before the decimal point and a maximum of 6 digits after the decimal point, combined with a Workday-recognized, 3-digit currency code. | Currency values can represent any valid positive or negative value given the specified number of digits before and after the decimal point. |
| Date | Date combined with a time of day with fractional seconds based on a 24-hour clock. | Date range: January 1, 1753, through December 31, 9999<br>Time range: 00:00:00 through 23:59:59.997<br>Internally, Workday stores all Date type data in UTC format (coordinated universal time). If you bring in external data with time zone information, Workday converts the data to UTC. If you bring in external data with no time zone information, Workday doesn't convert the data and stores it as UTC. |
| Double | Double-precision 64-bit floating point number. | 4.94065645841246544e-324d to 1.79769313486231570e+308d (positive or negative) |

| Dataset Field Type | Description | Range of Values |
|---|---|---|
| Instance | A Workday-specific field type that contains Workday Instance values. Each field also retains information about the business object the Instance field is based on. Instance fields represent a 1-to-1 relationship between 2 objects. | A hexadecimal value that references a WID. |
| Integer | 32-bit integer (whole number). | -2,147,483,648 to 2,147,483,647 |
| Long | 64-bit long integer (whole number). | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| Multi-Instance | A Workday-specific field type that contains a set of (zero or more) Workday Instance values. Each field also retains information about the business object the Instance values are based on. Multi-Instance fields represent a 1-to-many relationship between 2 objects. | Zero or more hexadecimal values that reference a WID. |
| Numeric | A variable length numeric value that supports a total of 38 digits before and after the decimal point, but a maximum of 18 digits after. | Numeric values can represent any valid positive or negative value given the specified number of digits before and after the decimal point.<br>All Numeric fields have an associated number of digits before and after the decimal point (also known as precision and scale). For Numeric fields that come from a Workday report, Workday assigns the digit values. For Numeric fields that come from external data, you can specify the digit values. |
| Text | Variable length non-Unicode text (also known as string) data. | Maximum text length of 2,147,483,647 characters. |

## Table Field Types

If your source data contains currency data, you must specify Text as the field type. Then when you create a derived dataset from the table, you can create a Prism calculated field to convert the Text field to a Currency field.

Tables use these field types:

| Table Field Type | Description | Range of Values |
|---|---|---|
| Boolean | A Workday specific field type that contains boolean values. | `True`, `False` |
| Date | Date combined with a time of day with fractional seconds based on a 24-hour clock. | Date range: January 1, 1753, through December 31, 9999<br>Time range: 00:00:00 through 23:59:59.997<br>Internally, Workday stores all Date type data in UTC format (coordinated universal time). If you bring in external data with time zone information, Workday converts the data to UTC. If you bring in external data with no time zone information, Workday doesn't convert the data and stores it as UTC. |
| Instance | A Workday-specific field type that contains Workday Instance values. Each field also retains information about the business object the Instance field is based on. | A hexadecimal value that references a WID. |

| Table Field Type | Description | Range of Values |
|---|---|---|
| Numeric | A variable length numeric value that supports a total of 38 digits before and after the decimal point, but a maximum of 18 digits after. | Numeric values can represent any valid positive or negative value given the specified number of digits before and after the decimal point.<br>All Numeric fields have an associated number of digits before and after the decimal point (also known as precision and scale).<br>You specify the precision and scale of each Numeric field. Example: `Numeric(19,2)` can store any value from 0.00 to 9,223,372,036,854,775,807.99 and `Numeric(19,0)` can only store whole numbers from 0 to 9,223,372,036,854,775,807. |
| Text | Variable length non-Unicode text (also known as string) data. | Maximum text length of 2,147,483,647 characters. |

## Mapping Report Field Types to Dataset Field Types

When you create a dataset from a custom report, Workday:

- Creates a field in the dataset for every report field that uses a supported field type.
- Assigns a field type to the fields in the dataset.

Workday only retains report fields that have field types that datasets support. Workday doesn't include any field that uses an unsupported dataset field type.

After creating a dataset from a custom report, verify that the assigned field types are correct for your dataset needs. Example: You might need to change a field type from Numeric to Integer to match the field type when joining the dataset with another dataset.

This table explains how a report field type maps to a dataset field type when you create a dataset.

| Report field Type | Dataset Field Type | Notes |
|---|---|---|
| Numeric | Numeric, Integer, or Long | The field type in the dataset depends on the number of digits before and after the decimal point in the report:<br>• Integer. Zero digits after, and less than or equal to 9 digits before the decimal point.<br>• Long. Zero digits after, and greater than 9 digits and less than or equal to 18 digits before the decimal point.<br>• Numeric. All other Numeric values.<br>For Numeric fields, Prism Analytics assigns the number of digits before and after the decimal point based on the values in the custom report. |
| Text | Text | |
| Rich Text | Not supported | |
| Date | Date | |
| DateTimeZone | Not supported | |
| Time | Not supported | |
| Currency | Currency | |
| Boolean | Boolean | |
| Multi-Instance | Multi-Instance | Datasets only include Multi-Instance report fields that are located in:<br>• The primary business object.<br>• A related business object that has a 1-to-1 relationship with the primary business object. |
| Single Instance | Instance | |
| Self-Referencing Instance | Instance | |

## Mapping Dataset Field Types to Prism Data Source Field Types

When you publish a dataset, Workday creates a Prism data source.

This table explains how a dataset field type maps to a Prism data source field type when you publish a dataset.

| Dataset Field Type | Prism Data Source Field Type | Notes |
|---|---|---|
| Text | Text | |
| Boolean | Boolean | |
| Date | Date | Workday doesn't include any time information in the Date field in the Prism data source. Workday deletes any time information in the dataset Date field when you publish the dataset. |
| Currency | Currency | |
| Integer | Numeric | |
| Long | Numeric | |
| Double | Numeric | |
| Numeric | Numeric | Workday writes Numeric fields with a maximum of 20 digits before and 6 digits after the decimal point. If a Numeric field contains a value that exceeds these maximums, then Workday rounds down the value when it creates the Prism data source. |
| Instance | Single Instance | Workday assigns the business object name associated with the Instance field in the dataset. |
| Multi-Instance | Multi-Instance | Workday assigns the business object name associated with the Multi-Instance field in the dataset. |

**Related Information**

**Tasks**
Change Dataset Field Types
**Reference**
Reference: Currency Format Requirements for External Data in Datasets

### 1.5.10 | Concept: NULL Values in Datasets

If a field value in a dataset is empty, it's considered a NULL value. When you publish a dataset, Workday replaces all NULL values with a default value in the Prism data source. Prism data sources, discovery boards, and reports have no concept of NULL values.

## How Datasets Process NULL Values

A value can be NULL for these reasons:

- The raw data from an external data source is missing values for a particular field.
- The raw data from an external data source can't be parsed for the specified field type.
- A Prism calculated field expression returns an empty or invalid result.
- The dataset contains a Join stage and:
  - You configure an outer join (you include all rows from at least one of the dataset pipelines).
  - There's no match from 1 pipeline to the other.

  If you're familiar with SQL, this is similar to an unjoined foreign key.

## Default Values by Field Type

Workday replaces NULLs with these values in a Prism data source:

| Field Type | Value in Prism Data Source |
|---|---|
| Boolean | False |
| Currency | 0 (with an empty currency code) |
| Date | (Blank) |
| Numeric, Double, Integer, Long | 0 |
| Instance | (Blank) |
| Multi-Instance | (Blank) |

| Field Type | Value in Prism Data Source |
|---|---|
| Text | (Blank) |

## NULL Values in Prism Calculated Fields

The way datasets treat NULL values in Prism calculated fields is consistent with SQL standard. That means:

- Arithmetic calculations on numeric fields that involve a NULL return NULL. Example: `5 + NULL` returns NULL.
- Comparison operations that result in a Boolean field that involve a NULL return NULL. Example: `5 > NULL` returns NULL.

Currency fields have 2 components to the field value (the currency code and currency value), and as a result, they handle NULLs a little differently than numeric fields in arithmetic calculations.

- Addition and subtraction calculations on Currency fields that involve a NULL return NULL. Example: `TO_CURRENCY("5.00 USD") + NULL` returns NULL.
- Addition and subtraction calculations on Currency values that use different currency codes return NULL. Example: `TO_CURRENCY("5.00 USD") + TO_CURRENCY("5.00 EUR")` returns NULL.

You can't use Currency fields in comparison operations. However, you can test for NULL values by using this syntax:

*value* `IS NULL`
*value* `IS NOT NULL`

## NULL Values in Group By Stages

When you create a Group By stage, you select a dataset field for grouping and another field for summarizing. How Workday treats NULL values depends on how you use the field:

- For grouping fields, any NULL values result in their own group.
- For summarization fields, NULL handling depends on the summarization type.

| Summarization Type | Description |
|---|---|
| Average | Returns the average of all valid numeric values in the group. It sums all values in the provided expression and divides by the number of valid (not NULL) rows.<br>If a Currency field contains multiple currency codes, the result is NULL. |
| Count | Returns the number of all rows in a group (counts all values, both NULL and non-NULL).<br>If a Currency field contains multiple currency codes, the result is NULL. |
| Maximum | Returns the greatest of all non-NULL values, and NULL if all values are NULL.<br>If a Currency field contains multiple currency codes, the result is NULL. |
| Minimum | Returns the lowest of all non-NULL values, and NULL if all values are NULL.<br>If a Currency field contains multiple currency codes, the result is NULL. |
| Sum | Returns the total of all non-NULL values, and NULL if all values are NULL.<br>If a Currency field contains multiple currency codes, the result is NULL. |

### Related Information

**Concepts**
Concept: Table and Dataset Field Types
**Reference**
Reference: Currency Format Requirements for External Data in Datasets

### 1.5.11 | Concept: Prism Calculated Fields

A Prism calculated field is a user-created field that generates its values based on a calculation or condition, and returns a value for each input row. Values are computed based on expressions that can contain values from other fields, constants, mathematical operators, comparison operators, or built-in row functions.

Use Prism calculated fields to:

- Derive meaningful values from base fields, such as calculating someone's age based on their birthday.
- Do data cleansing, such as substituting 1 value for another.
- Compute new data values based on a number of input variables, such as calculating a profit margin value based on revenue and costs.

Sometimes you need several steps to achieve the result that you want. You can use the result of a Prism calculated field in the expressions of other Prism calculated fields, enabling you to define a chain of processing steps.

When you create a Prism calculated field, the inspector panel displays for the new field. To see a list of available functions, click the Functions Library tab in the inspector panel.

You can extract and export the expressions used in dataset pipelines as CSV files from the View Dataset Lineage report.

### Related Information

**Concepts**
Concept: Prism Expression Language
**Tasks**
Add a Prism Calculated Field to a Dataset

### 1.5.12 | Concept: Hiding Dataset Fields

A data administrator can control what fields of a dataset are visible. Hidden fields aren't visible further along in the development process. Example: if you hide a field in a stage in a pipeline, that field isn't visible in a later stage. Hidden fields aren't included in the Prism data source of a published dataset, or in an imported dataset of a derived dataset.

You might decide to hide a field to:

- **Protect sensitive data.** In some cases, you might want to hide fields to protect sensitive information. You can hide detail fields, but still allow access to summary information. Suppose that you have a dataset containing employee salary information. You might want to hide salaries per person, but still enable analysts to view average salary by department.
- **Hide unpopulated or sparse data fields.** You might have fields in your raw data that didn't have any data collected. The data collected might be too sparse to be valid for analysis. Suppose that a web application has a placeholder field for comments, but it was never implemented on the website so the comments field is empty. Hiding the field prevents analysts from using a field with mostly null values when they analyze the data.
- **Use a calculated field instead of the fields that it's based on.** You might add a Prism calculated field to transform the values of the raw data. You want your users to analyze the transformed values, not the raw values. Suppose that you have a "return reason code" field where the reason codes are numbers (1, 2, and 3). You could transform the numbers to the actual reason information (such as Didn't Fit, Changed Mind, and Poor Quality), so the data is more usable during analysis.
- **Hide Prism calculated fields that do interim processing.** As you work on your dataset to cleanse and transform the data, you might need to add interim Prism calculated fields to achieve a final result. These fields are necessary to do a processing step, but aren't intended for final consumption. You can hide these working fields so they don't clutter later stages or the dataset details.

Hide a field in the Manage Fields stage by unselecting the check box for a field. Although you can also hide a field in the inspector panel, it's a best practice to hide fields in the Manage Fields stage.

### Related Information

**Concepts**
Concept: Prism Analytics Data Management Workflow

### 1.5.13 | Concept: Dataset Integration Schedules

Schedules for dataset integrations enable you to specify when, how often, and under what criteria to import data into a dataset. Workday enables you to import data into a dataset immediately on an ad hoc basis or according to a preconfigured schedule.

You can schedule the integration to run:

- Once in the future.
- On a recurring basis (Example: daily, weekly, or monthly).
- Only after another Prism scheduled process completes at a status you specify.

## Integration Schedule Types

| Schedule Type | Description |
|---|---|
| Recurring | An integration schedule that runs at specified intervals, such as daily, weekly, or monthly. |
| Dependent | An integration schedule that depends on the completion of another Prism scheduled process. For your dependency criteria, you can specify:<br>• The process type, such as another Prism data integration.<br>• The status that triggers integration, such as the process type successfully completing.<br>Example: When the *Prism Data Acquisition Future Process* completes with no warnings or errors, begin integration.<br>Note: A Prism Analytics integration schedule can depend only on another Prism-related process, such as bringing data into a dataset or publishing another dataset. |

As you create an integration schedule, consider these actions that you can perform on it:

| Action | Description |
|---|---|
| Activate | Activate a suspended integration schedule. |

| Action | Description |
|---|---|
| Change Schedule (recurring schedules only) | Edit the run frequency (daily, monthly, weekly), start time, and date range for the integration schedule. You can also change to another scheduled recurring process. |
| Delete | Permanently delete the integration schedule. |
| Edit Environment Restrictions | Select the environment in which you want the scheduled integration to run. |
| Edit | (Recurring Schedules) Edit the schedule name, recurrence criteria, and range of recurrence dates.<br>To change the run frequency, use Change Schedule.<br>(Dependent Schedules) Edit the schedule name, dependency, trigger status, and timed delay configurations. |
| Edit Scheduled Occurrence (recurring schedules only) | Update the schedule date and time for one particular occurrence of the scheduled request.<br>You can also delete a particular occurrence of the scheduled integration. |
| Run Now | Run the integration schedule immediately on an ad hoc basis. |
| Suspend | Suspend use of the integration schedule. You can activate a suspended schedule. |
| Transfer Ownership | Transfer ownership of an integration schedule. Every process must have an assigned owner for the process to run.<br>Example: Transfer ownership if the assigned owner becomes inactive. The person you transfer ownership to must have the appropriate security access. |
| View All Occurrences (recurring schedules only) | View all future occurrences of an integration schedule within a specified range of dates and times. |
| View Details | (Recurring schedules) View schedule details, such as recurrence criteria, error messages, the schedule owner and creator, and the next 10 scheduled integrations if applicable.<br>(Dependent schedules) View schedule details, such as the dependency configuration, the schedule creator and owner, and the number of times run. |

### Related Information

**Tasks**

Manage Dataset Integration Schedules
Steps: Create a Dataset with External Data (SFTP Server)
Steps: Create a Dataset Using Workday Data

## 1.5.14 | Reference: Dataset Stages

Prism Analytics supports these stage types:

| Stage Type | Description |
|---|---|
| Import | The first stage in every pipeline in a derived dataset is an Import stage. An Import stage imports data from a source, and for derived datasets that source is the output of a table or an existing dataset. Workday automatically creates an Import stage when necessary.<br>You can't delete or add an Import stage. |
| Parse | A Parse stage is a type of stage that enables you to describe the source data in a tabular format. Workday automatically displays the Parse stage when you create a dataset using external data.<br>You can't delete or add a Parse stage. |
| Manage Fields | A Manage Fields stage is a type of stage that enables you to view field changes, select fields, and edit fields. |

| Stage Type | Description |
|---|---|
| Filter | A Filter stage is a type of stage that constrains rows in a dataset based on the filter criteria you define. Add Filter stages to limit the data in the dataset for analysis, such as on a particular region or year.<br><br>You can add or delete a Filter stage in the middle or at the end of a pipeline. |
| Group By | A Group By stage is a type of stage that enables you to summarize (aggregate) multiple values in a dataset by specified groups. You can summarize the values using a summarization type, such as MIN, MAX, or SUM. Add Group By stages to get data to the appropriate level that you need to join the data in 1 dataset with another dataset.<br><br>You can add or delete a Group By stage in the middle or at the end of a pipeline. |
| Join | A Join stage is a type of stage that combines fields from 2 dataset pipelines based on common values that exist in each pipeline. Add Join stages to view and use related data from different datasets. You can add Join stages to derived datasets only, because derived datasets are the only datasets that can use multiple datasets as their source data.<br><br>You can delete a Join stage only in the last stage of a pipeline. |
| Union | A Union stage is a type of stage that combines data from similar fields in different datasets into a single field. Add Union stages to:<br>• Combine datasets that have similar, but not identical schema.<br>• Combine datasets with the same schema but with source data from different locations, such as different SFTP servers.<br><br>You can add Union stages to derived datasets only, because derived datasets are the only datasets that can use multiple datasets as their source data.<br><br>You can delete a Union stage only in the last stage of a pipeline. |

### Related Information

**Concepts**
Concept: Dataset Stages

### 1.5.15 | Reference: Table Error File Error Codes

When you view an error file for a table load, you might see these errors:

| Error Code | Notes |
|---|---|
| 3000 | The number of fields in the source doesn't match the number of fields in the wBucket schema. You use wBuckets when you load data into a table using the REST API. |
| 3001 | Invalid data - Text field |
| 3002 | Invalid data - Integer field. When the precision and scale of a Numeric field indicates that the field should contain integer values only, you might see this error. Example: Numeric(9,0) |
| 3003 | Invalid data - Numeric field |
| 3004 | Invalid data - Date field |
| 3005 | Invalid data - Boolean field |
| 3006 | Invalid data - Instance field |
| 4000 | The number of characters for the Text field exceeds the maximum allowed (32,000 characters). |
| 4003 | The numeric value is too large for the defined precision and scale for the Numeric field. |

| Error Code | Notes |
|---|---|
| 5000 | This error code can correspond to either of these errors:<br>• The error file has reached its maximum of 10 MB and has stopped recording new error messages.<br>• The number of characters for the entire row of data exceeds the maximum allowed (500,000 characters). |

#### Related Information

**Concepts**

Concept: Table Error File

## 1.6 | Creating Tables and Datasets

### 1.6.1 | Steps: Create a Table by File Upload

#### Prerequisites

Security: *Prism: Tables Create* domain in the Prism Analytics functional area.

#### Context

You can create a table by uploading 1 or more delimited files. Workday uses the field information in the first file to define the table fields, and then it loads the data in the files into the table.

#### Steps

1. Access the Data Catalog report.
2. Select Create > Table.
3. Enter the Table Name.
4. (Optional) Change the Table API Name.
   Workday automatically selects an API name based on the table name you enter, modifying it to make it meet the name requirements. Click Change to change the API name. You can't change this name after you finish creating the table. Table API names:
   - Must be unique in the Data Catalog.
   - Can contain up to 255 characters.
   - Can only include alphanumeric and underscore characters.
   - Must start with a letter.
   - Can't end with an underscore character.
   - Can't begin with WPA_.
5. (Optional) Create or edit 1 or more Tags to organize the table in the Data Catalog.
6. (Optional) Add a Description to help others understand what data this table contains.
7. On the Select Schema Source step, select File Upload.
8. Select 1 or more delimited files to upload.
   When you upload multiple files, each file must use the same schema. Workday supports delimited files that are RFC 4180-compliant. For more information, see RFC 4180.
9. On the Edit Parsing Options step, define how to parse the data in the file.
   See Parse External Data in a Table.
10. On the Edit Schema step, review the fields Workday created based on the parsed file, and modify the fields if necessary.
    Select a field in the list and view the field details in the inspector panel on the right side. You might want to:
    - Change the field API Name. You can't change the API name after you save the table.
    - Change the Field Type if Workday assigned the wrong field type. Example: Workday assigned the Numeric field type to a field with zip code data because the example rows it evaluated only contained numerals. But you know that some zip code values might contain letters or a hyphen, so you change the field type to Text.
    - Change other field attributes, such as Date Format, based on the field type.
    - Define some field constraints that ensure the accuracy and reliability of the data in the table, such as Required or Use as External ID.

    See Reference: Table Field Attributes.
11. (Optional) Click Add Field to add 1 or more fields. In the inspector panel for the field, configure the field attributes.
    See Reference: Table Field Attributes.

#### Result

Workday creates the table and starts a data load activity to load the data in the files into the table. On the View Table report, click the Activities tab to view the data load progress. Refresh the page to get the most recent status.

#### Next Steps

If there were errors loading data into the table, download the error file from the data load on the Activities tab of the View Table report.

#### Related Information

**Concepts**

Concept: Prism Analytics Data Management Workflow
Concept: Tables
Concept: Table and Dataset Field Types

Concept: Table Error File
**Tasks**
Parse External Data in a Table
**Reference**
Reference: WPA_ Fields
Reference: Supported File Formats for External Data in Tables and Datasets
Reference: Table Field Attributes
2020R1 What's New Post: Prism Analytics Tables

### 1.6.2 | Steps: Create a Table Manually

#### Prerequisites

Security: *Prism: Tables Create* domain in the Prism Analytics functional area.

#### Context

You can create a table by manually defining each field in the table schema. When you create a table manually, the table is empty. You can load data that is compatible with the table schema into the table.

#### Steps

1. Access the Data Catalog report.
2. Select Create > Table.
3. Enter the Table Name.
4. (Optional) Change the Table API Name.
   Workday automatically selects an API name based on the table name you enter, modifying it to make it meet the name requirements. Click Change to change the API name. You can't change this name after you finish creating the table. Table API names:
   - Must be unique in the Data Catalog.
   - Can contain up to 255 characters.
   - Can only include alphanumeric and underscore characters.
   - Must start with a letter.
   - Can't end with an underscore character.
   - Can't begin with WPA_.
5. (Optional) Create or edit 1 or more Tags to organize the table in the Data Catalog.
6. (Optional) Add a Description to help others understand what data this table contains.
7. On the Select Schema Source step, select Manual Input.
   The Edit Schema step displays where you can define each field in the table schema.
8. Click Add Field to add 1 or more fields.
9. In the inspector panel for the field, configure the field attributes.
   See Reference: Table Field Attributes.

#### Next Steps

Load data into the table. Example: You can use version 2 of the REST API to load data into a table.

#### Related Information

**Concepts**
Concept: Prism Analytics Data Management Workflow
Concept: Tables
Concept: Table and Dataset Field Types
**Tasks**
Change Rows in a Table by Uploading a File
Steps: Load Data into a Table Using the REST API
**Reference**
Reference: WPA_ Fields
Reference: Supported File Formats for External Data in Tables and Datasets
Reference: Table Field Attributes

### 1.6.3 | Steps: Create a Dataset with External Data (SFTP Server)

#### Prerequisites

Security: *Prism Datasets: Create* domain in the Prism Analytics functional area.

#### Context

You can create a dataset using external data by transferring data from an SFTP server. You might want to create a dataset that gets its data from an external server when the server regularly collects or adds new data. You configure how often the dataset gets new data from the server.

For integration runs that transfer data from the SFTP server to succeed:

- The number of files must be less than 1,000.
- The time to transfer the data must be less than 6 hours.

Each file from the server should be less than 1 GB compressed (less than 10 GB uncompressed approximately).

Creating a dataset using external data creates a base dataset.

### Steps

1. Access the Data Catalog report.
2. Select Create > from SFTP.
   On the Create Dataset Retrieval - Configure File Retrieval task, you configure how to import the data from the SFTP server.
3. In the Files section, enter the filename or a filename pattern that represents 1 or more files.
   The filename is case-sensitive. You can use the asterisk (*) and question mark (?) characters as wild cards to specify a filename pattern. Use the asterisk (*) to specify zero or more characters, and use the question mark (?) to specify exactly 1 character.
4. In the Transport section, specify how to connect to the SFTP server:

| Option | Description |
|---|---|
| **SFTP Address** | Use this format: `sftp://domain_name` or `sftp://IP_address`<br><br>To specify a port number, add it to the end of the domain name or IP address. If you don't specify a port number, Workday uses port 22. |
| **Directory** | (Optional) The directory on the server that contains the files. Directory names are case-sensitive. Only include a leading slash (/) to specify a full path, not a relative path. |
| **Use Temp File** | Writes the imported data to a temporary file in Workday with a randomly generated name. After the data import is complete, Workday automatically renames the file to the correct name.<br><br>You might want to enable this option if the data import takes a very long time and might not finish before the next scheduled time to import data from the same server. |
| **Authentication Method and Details** | Select the type of security authentication that the SFTP server uses:<br>◦ User Name / Password.<br>◦ SSH Authentication. This option uses secure shell key authentication using X.509 certificates. |

5. (Optional) In the File Utilities section, Consider these options:

| Option | Description |
|---|---|
| **Delete After Retrieval** | Deletes the files on the SFTP server after the data is imported into the dataset. If Workday is unable to delete the files from the SFTP server, the data retrieval fails. |
| **Decompress** | Do not enable this option for datasets.<br><br>You can transfer files that are compressed or not. For compressed files, Workday only supports gzip compression. |
| **Decrypt Using** | If you want to decrypt the imported files using Pretty Good Privacy (PGP), select a PGP Private Key Pair. |

6. (Optional) In the Environment Restrictions section, at the Restricted To prompt, select the environment in which you want to use the settings defined in the Transport section.
   If you leave this option empty, Workday applies the transport settings to each environment in which the dataset integration runs. When a dataset integration runs in a particular environment, such as Implementation or Production, the transport settings only work if the Restricted To option matches the current environment. When the current environment and the configured environment don't match, the dataset integration fails and retrieves no files from the SFTP server. You might want to restrict the transport settings to a particular environment to avoid inadvertently transferring test data to a non-test endpoint.

   Example: You create the dataset in an Implementation environment and select Implementation in Restricted To. Later, you migrate this dataset to a Production environment and the next time the dataset integration runs, the integration fails. To ensure the dataset integration runs successfully in the Production environment, edit the dataset integration details and either clear the Restricted To option or change it to Production.

7. On the Create Dataset Retrieval - Schedule Request Type task, in Run Frequency, specify how often to import data from the SFTP server.
   If you're importing the data once in the future or on a schedule, specify the criteria for either on the Create Dataset Retrieval - Schedule Integration task. After the dataset is created, you can run the integration to bring in data to the dataset on an ad hoc basis. From the related actions menu of the View Dataset task, select Dataset > Run Integration Now.
   Note: You can't bring data into the same dataset at times that overlap with each other.
8. Specify a unique name and optional description for the dataset.
   The dataset name is what you see in the Data Catalog. After you save the dataset, you can't change the name.
9. Select how you want to update the data in the dataset when it receives new data from an integration run.

| Option | Description |
|---|---|
| **Replace** | Workday deletes the existing data in the dataset and replaces it with the data it imports from the SFTP server. |

| Option | Description |
|---|---|
| **Append** | Workday keeps the existing data in the dataset and adds to it the new data it imports from the SFTP server.<br><br>Workday imports all data in all files during every integration run. Append mode is different than incrementally updating data in a dataset. Whether the data in the dataset gets updated incrementally depends on if the SFTP server contains only incremental updates since the last integration run.<br><br>The file schema must meet these requirements:<br><br>◦ All files in every integration run must use the same parsing options (including the header row configuration) that were used during the first integration run.<br>◦ The fields must be in the same order in all files in every integration run.<br>◦ If the schema in a subsequent integration run contains new fields, the new fields must be located at the end of all previous fields.<br>◦ If the file schema in a subsequent integration run deletes one or more fields, the deleted fields must be at the end.<br>◦ Ensure that if the schema deletes fields, no future schema adds new fields, otherwise the integration run will fail. To ensure that all future integrations run successfully, always keep existing fields in the schema and only add new fields. If necessary, you can include empty (NULL) values in existing fields.<br>◦ All files in a single integration must use the same schema. |

Note: An integration fails when the schema of the new data doesn't contain a field that currently exists in the dataset, and the removed field is used in a stage in the dataset. Example: If the dataset includes a Manage Fields stage and the integration brings in data that is missing a field in the dataset, the integration fails. That's because the Manage Fields stage works on every field in the dataset.

10. Click Save.
    Workday creates the dataset, but it has no data until Workday imports the data and fields from the SFTP server during the first integration run. Depending on when you scheduled the data to import, the dataset might be empty for some time. Workday also adds 2 fields that provide information about each integration run. See: Reference: WPA_ Fields.
11. (Optional) Change the name of your integration schedule. See Manage Dataset Integration Schedules.
12. Access the Data Catalog report, right-click the dataset you just created, and select Edit.
13. Configure how to parse the data in the files from the SFTP server.
    See: Parse External Data in a Dataset.
14. (Optional) Add a Stage to a Dataset.
    You can add only some stage types to base datasets.
15. (Optional) Add a Prism Calculated Field to a Dataset.
    You can add a Prism calculated field to any stage.

### Related Information

**Concepts**
Concept: Dataset Workspace
Concept: Datasets
Concept: Dataset Stages
Concept: Dataset Pipelines
**Reference**
Reference: Supported File Formats for External Data in Tables and Datasets
Reference: WPA_ Fields

## 1.6.4 | Steps: Create a Dataset with External Data (Upload a File)

### Prerequisites

Security: *Prism Datasets: Create* domain in the Prism Analytics functional area.

### Context

You can create a dataset using external data by uploading a file. You might want to create a dataset by uploading a file when the data in the file is less likely to change over time.

When you create a dataset by uploading a file, the source data in the dataset remains the same over time. However, you can change the data in the dataset later by uploading a new file to the dataset. See: Upload a New File to a Dataset.

Creating a dataset using external data creates a base dataset.

### Steps

1. Access the Data Catalog report.
2. Select Create > from File.
3. Navigate to a file on your local machine and open it.

   The maximum size file you can upload is 500 MB.

4. Define how to parse the data in the file.
   See: Parse External Data in a Dataset.
5. Click Next to name the dataset and provide an optional description.

After you save the dataset, you can't change the name.

6. (Optional) Create or edit tags to organize the dataset in the Data Catalog report.
7. (Optional) Add a Stage to a Dataset.
   You can add only some stage types to base datasets.
8. (Optional) Add a Prism Calculated Field to a Dataset.
   You can add a Prism calculated field to any stage.

### Related Information

**Concepts**
Concept: Dataset Workspace
Concept: Datasets
Concept: Dataset Stages
Concept: Dataset Pipelines
**Tasks**
Upload a New File to a Dataset
**Reference**
Reference: Supported File Formats for External Data in Tables and Datasets

## 1.6.5 | Steps: Create a Dataset Using Workday Data

### Prerequisites

Security: *Prism Datasets: Create* domain in the Prism Analytics functional area.

### Context

You can create a dataset using Workday data. You do this by creating a dataset using an existing Workday custom report as the source for the dataset.

You configure how often the dataset gets new data from the report.

Workday retains only the fields with field types that datasets currently support.

Creating a dataset from a custom report creates a base dataset.

### Steps

1. Access the Data Catalog report.
2. Select Create > from Custom Report.
   Workday displays reports that meet the eligibility requirements for importing into Prism. See: Concept: Creating Reports to Import into Datasets.

   On the Create Dataset Retrieval - Configure Report Retrieval task, you configure how to import the data from the custom report.

3. Select a Custom Report that has the data you want to import into this dataset.
4. In the Report Criteria table, select values for the report prompts, if applicable.
   Workday filters the report data with the specified values as the report runs and before importing the data into the dataset. As you complete this step, consider:

| Option | Description |
|---|---|
| **Value Type** | This option affects how Workday determines the value for this field prompt:<br>◦ **Specify Value.** Workday uses the same value that you specify here each time it runs the report to import data into the dataset.<br>◦ **Determine Value at Runtime.** Workday uses the current value in a field you specify each time it runs the report to import data into the dataset. |
| **Value** | Workday uses the value or field you select here to filter the data in the report. |

5. (Optional) In the Environment Restrictions section, at the Restricted To prompt, select the environment in which you want to use the settings defined in the Transport section.
   If you leave this option empty, Workday applies the transport settings to each environment in which the dataset integration runs. When a dataset integration runs in a particular environment, such as Implementation or Production, the transport settings only work if the Restricted To option matches the current environment. When the current environment and the configured environment don't match, the dataset integration fails and retrieves no data from the specified custom report. You might want to restrict the transport settings to a particular environment to avoid inadvertently transferring test data to a non-test endpoint.

   Example: You create the dataset in an Implementation environment and select Implementation in Restricted To. Later, you migrate this dataset to a Production environment and the next time the dataset integration runs, the integration fails. To ensure that the dataset integration runs successfully in the Production environment, edit the dataset integration details and either clear the Restricted To option or change it to Production.

6. On the Create Dataset Retrieval - Schedule Request Type task, in Run Frequency, specify how often to import data from the custom report.
   If you're importing the data once in the future or on a schedule, specify the criteria for either on the Create Dataset Retrieval - Schedule Integration task.

   After the dataset is created, you can run the integration to bring in data to the dataset on an ad hoc basis. From the related actions menu of the View Dataset task, select Dataset > Run Integration Now. You can't bring data into the same dataset at times that overlap with each other.

   Note: An integration fails to bring in new data from the custom report when the report schema doesn't contain a field that currently exists in the dataset, and the removed field is used in a stage in the dataset. Example: If the dataset includes a Manage Fields stage and the integration brings in data that is missing a field in the dataset, the integration fails. That's because the Manage Fields stage works on every field in the dataset.

7. Specify a unique name and optional description for the dataset.
   The dataset name is what you see in the Data Catalog. After you save the dataset, you can't change the name.
8. Select how you want to update the data in the dataset when it receives new data from an integration run.

| Option | Description |
|---|---|
| **Replace** | Workday deletes the existing data in the dataset and replaces it with the data it imports from the custom report. |
| **Append** | Workday keeps the existing data in the dataset and adds to it the new data it imports from the custom report.<br><br>Workday imports all data in the report during every integration run, resulting in duplicate data in the dataset. Select append mode for a custom report dataset when you want a snapshot of the custom report data to maintain history in the dataset for trending use cases.<br><br>Note: Ensure that you don't change the Column Heading Override XML Alias values in the custom report definition. Workday uses these values to map fields from the custom report into the dataset. |

9. Click Save.

   Workday creates the dataset, but it has no data until Workday runs the report and then imports the data and fields from the report during the first integration run. Depending on when you scheduled the data to import, the dataset might be empty for some time. Workday also adds 2 fields that provide information about each integration run. See: Reference: WPA_ Fields.

10. (Optional) Change the name of your integration schedule. See Manage Dataset Integration Schedules
11. Access the Data Catalog report, right-click the dataset you just created, and select Edit.
12. (Optional) Add a Stage to a Dataset.
    You can add only some stage types to base datasets.
13. (Optional) Add a Prism Calculated Field to a Dataset.
    You can add a Prism calculated field to any stage.

### Related Information

**Concepts**
Concept: Dataset Workspace
Concept: Datasets
Concept: Dataset Stages
Concept: Dataset Pipelines
Concept: Creating Reports to Import into Datasets
**Reference**
Reference: WPA_ Fields

## 1.6.6 | Steps: Create a Derived Dataset

### Prerequisites

Security:

- *Prism Datasets: Create* domain in the Prism Analytics functional area.
- Any of these requirements:
  - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
  - *Dataset Viewer* permission on the dataset to import into the derived dataset.
  - *Dataset Editor* permission on the dataset to import into the derived dataset.
  - *Dataset Owner* permission on the dataset to import into the derived dataset.

### Context

When you first create a derived dataset, Workday creates the Primary Pipeline. Import other datasets into the derived dataset so you can blend data together.

You can add a stage to any pipeline in the dataset. However, some stages, such as the Join stage, can only be added to the Primary Pipeline.

When you add a Join or Union stage to the Primary Pipeline, you must select another pipeline in the derived dataset to blend with the Primary Pipeline. Workday uses the last stage of that pipeline as the input to the Join or Union stage.

### Steps

1. Access the Data Catalog report.
2. Select Create > Derived Dataset.
3. Select a table or dataset from the list.
4. Enter a name for the dataset.
   After you save the dataset, you can't change the name.
5. (Optional) Create or edit tags to organize the dataset in the Data Catalog report.
6. Click Edit to return to the Edit Dataset task.
7. The Edit Dataset task displays 1 pipeline (the Primary Pipeline) that contains an Import stage.
8. Create 1 or more additional pipelines by importing an existing dataset.
   See: Import a Table or Dataset into a Derived Dataset.
9. (Optional) Add a Stage to a Dataset.

You can add any stage type to derived datasets, but you can add some stage types to the Primary Pipeline only. Consider adding a Join stage to the Primary Pipeline to blend data from 2 pipelines.

10. (Optional) Add a Prism Calculated Field to a Dataset.
    You can add a Prism calculated field to any stage in any pipeline.

### Related Information

**Concepts**

Concept: Dataset Workspace
Concept: Datasets
Concept: Dataset Stages
Concept: Dataset Pipelines

## 1.6.7 | Import a Table or Dataset into a Derived Dataset

### Prerequisites

- Any of these security requirements:
  - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
  - *Dataset Editor* permission on the derived dataset.
  - *Dataset Owner* permission on the derived dataset.
- Any of these security requirements:
  - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
  - *Dataset Viewer* permission on the dataset to import into the derived dataset.
  - *Dataset Editor* permission on the dataset to import into the derived dataset.
  - *Dataset Owner* permission on the dataset to import into the derived dataset.

### Context

A derived dataset is based on 1 or more existing tables or datasets. Use derived datasets to blend and combine together data from multiple sources. In order to blend and combine data from multiple sources, you need to import multiple tables or datasets into the derived dataset. When you first create a derived dataset, you base it on an existing table or dataset. Afterward, you must import other tables or datasets into the derived dataset.

When you import a table or dataset into a derived dataset, Workday creates a new pipeline. The pipeline name is the same as the table or dataset name you import. You can add stages to the new pipeline.

Once a derived dataset has multiple tables or datasets imported into it, you can add a stage, such as a Join stage, to the Primary Pipeline to blend data with any other pipeline.

### Steps

1. Access the Edit Dataset task for a derived dataset.
2. In the Pipelines panel, click Add Pipeline.
3. Select a table or dataset from the list.

### Result

The Pipelines panel displays the new pipeline with the first stage being an Import stage. The pipeline name is the same as the table or dataset you imported.

### Next Steps

- (Optional) Add a stage, such as a Join stage, that blends together data from the Primary Pipeline and the pipeline you added.

### Related Information

**Tasks**

Steps: Create a Derived Dataset

## 1.6.8 | Manage Dataset Integration Schedules

### Prerequisites

Security: *Prism Datasets: Manage* domain in the Prism Analytics functional area.

### Context

You can manage how you set up integration schedules for base datasets created from:

- SFTP
- Custom reports

You can schedule the integration to run:

- Once in the future.
- On a recurring basis (Example: daily, weekly, or monthly).
- Only if another Prism scheduled process completes at a status you specify.

### Steps

1. Access the View Dataset report for the dataset.

2. From the related actions menu, select Dataset > Edit Integration Details.
3. (Recurring schedules) As you set up the schedule, consider:

| Option | Description |
|---|---|
| **Catch Up Behavior** | Select how many times the scheduled integration runs after maintenance issues cause errors.<br><br>Example: If you schedule an integration to run multiple times in a week when your environment is down for maintenance, you can limit the process to run once instead of catching up all missed occurrences. |

4. (Dependent schedules) As you set up the schedule, consider:

| Option | Description |
|---|---|
| **Dependency** | Select By Type and 1 of the Prism-related scheduled future process types on which the integration depends:<br>◦ Prism Data Acquisition Future Process<br>◦ Scheduled Publish Dataset Process<br><br>Select the schedule on which the integration depends. |
| **Trigger on Status** | Select the status of the scheduled future process that triggers integration.<br><br>Example: You select Prism Data Acquisition Future Process as your process type and a future integration schedule called *Dataset Integration Schedule: Monthly Acquisition Expenses.* In the Trigger on Status field, you select Completed.<br><br>Workday brings data into the dataset only after the *Dataset Integration Schedule: Monthly Acquisition Expenses* integration successfully completes. |
| **Time Delayed Configuration** | (Optional) Specify the number of days, hours, or minutes to delay triggering integration. You might want to delay integration to review the latest source files. |

5. (Optional) Change the name of the schedule in the Request Name field. Workday assigns a name to the schedule based on the name of the dataset and prepends *Dataset Integration Schedule:* to the name.
6. (Optional) Perform actions such as transferring ownership of the schedule or editing 1 scheduled occurrence.
    a. Access the View Integration Details report for the dataset.
    b. Find the integration schedule in the Request Name column on the Schedules tab.
    c. From the related actions menu of the integration schedule, select Schedule Future Process and then the desired action.

### Result

Workday imports data into the dataset based on the criteria you specified.

You can view the status of all scheduled integration processes in the Process Monitor and Scheduled Future Processes reports. The status includes the date and time of the last successful integration. The last successful integration date informs you about the freshness of the data brought into the dataset. Example: If the last successful integration date is 1 week ago, but your integration schedule is set to run daily, this discrepancy could indicate a failure in the integration process.

### Related Information

**Concepts**
Concept: Dataset Integration Schedules

## 1.6.9 | Upload a New File to a Dataset

### Prerequisites

- Any of these security requirements:
    - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
    - *Dataset Editor* permission on the dataset.
    - *Dataset Owner* permission on the dataset.

### Context

When you create a dataset by uploading a file, the data in that dataset stays the same over time. If you have a new version of the source file, you can upload it to the same dataset. You might want to upload a new file to update an existing dataset instead of creating a new dataset. When you update an existing dataset, you maintain any relationships with datasets that depend on the existing dataset.

When you upload a new file, all existing data is replaced with the data in the new file.

Sometimes, the fields in the source file might change, also known as a schema change. Fields might be added, deleted, or moved. When the schema changes and the new file is imported into the dataset, you must edit the dataset and save it to incorporate the changes in the dataset. If you don't save the dataset, it'll continue

to use the old schema definition. The next time the dataset is published, you might get inconsistent data in the Prism data source.

Note: Uploading a file fails when the schema of the new file doesn't contain a field that currently exists in the dataset, and the removed field is used in a stage in the dataset. Example: If the dataset includes a Manage Fields stage and you try to upload a file that is missing a field in the dataset, the upload fails. That's because the Manage Fields stage works on every field in the dataset.

### Steps

1. Access the View Dataset report for the dataset you want to update with new data.
2. Click Upload File.
3. In the confirmation dialog, click Upload.
4. Navigate to and select the local file.
   Workday informs you that the schema of the source might have changed and displays the Edit Dataset button.
5. Click Edit Dataset.
   The Edit Dataset task displays. If the schema changed, the Save button is active.
6. If the Save button is active, click Save.

### Result

Workday replaces the data in the dataset with the data in the file you uploaded. It updates the fields in the dataset if the schema in the uploaded file is different.

### Next Steps

Verify that no schema changes broke any Prism calculated fields, stages, derived datasets, or Workday reports that depend on the dataset whose schema changed.

### Related Information

**Concepts**
Concept: Dataset Schema Changes
**Tasks**
Steps: Create a Dataset with External Data (Upload a File)

## 1.6.10 | View Prism Data Usage

### Prerequisites

- Any of these security requirements:
  - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
  - *Prism Datasets: Owner Manage* domain in the Prism Analytics functional area.
  - *Prism: Tables Manage* domain in the Prism Analytics functional area.
  - *Prism: Tables Owner Manage* domain in the Prism Analytics functional area.
  - *Prism Datasets: Publish* domain in the Prism Analytics functional area.
  - *Dataset Viewer* permission on 1 or more datasets.
  - *Dataset Editor* permission on 1 or more datasets.
  - *Dataset Owner* permission on 1 or more datasets.
  - *Table Viewer* permission on 1 or more tables.
  - *Table Editor* permission on 1 or more tables.
  - *Table Owner* permission on 1 or more tables.

### Context

You can view how much Prism data your organization has used on your tenant. You might want to view your Prism data usage to ensure you're in compliance with your purchase agreement with Workday.

Workday displays this data usage information:

- Total Disk Used. This value summarizes the disk spaced used for source data for all tables and datasets in your tenant, including the ones you don't have permission on.
- Total Rows Published. This value summarizes all rows in published datasets in your tenant, including the datasets you don't have permission on.
- **Table and dataset usage.** This grid lists all tables and datasets you have permission on, and includes the disk space used and number of published rows per table and dataset.

### Steps

1. Access the Data Catalog report.
2. Click Activities.
3. Click the Prism Usage tab.

## 1.6.11 | View Dataset Dependencies

### Prerequisites

- Any of these security requirements:
  - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
  - *Prism Datasets: Owner Manage* domain in the Prism Analytics functional area.
  - *Prism Datasets: Publish* domain in the Prism Analytics functional area.
  - *Prism Datasets: Create* domain in the Prism Analytics functional area.
  - *Dataset Viewer* permission on the dataset.
  - *Dataset Editor* permission on the dataset.
  - *Dataset Owner* permission on the dataset.

### Context

You create a derived dataset by importing a table or dataset on which the derived dataset is based. The derived dataset depends on the table or dataset you import into it. You can view these dataset dependencies.

### Steps

1. Access the Data Catalog page.
2. Select a dataset to open its inspector panel.
3. Scroll down to the Imported By section to view the derived datasets that imported the selected dataset.

## 1.6.12 | View Table and Dataset Lineage

### Prerequisites

- Any of these security requirements for dataset lineage:
  - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
  - *Prism Datasets: Owner Manage* domain in the Prism Analytics functional area.
  - *Prism Datasets: Publish* domain in the Prism Analytics functional area.
  - *Dataset Viewer* permission on the dataset.
  - *Dataset Editor* permission on the dataset.
  - *Dataset Owner* permission on the dataset.
- Any of these security requirements for table lineage:
  - *Prism: Tables Manage* domain in the Prism Analytics functional area.
  - *Prism: Tables Owner Manage* domain in the Prism Analytics functional area.
  - *Prism Datasets: Publish* domain in the Prism Analytics functional area.
  - *Table Viewer* permission on the table.
  - *Table Editor* permission on the table.
  - *Table Owner* permission on the table.

### Context

When you bring in data and transform it in Prism Analytics, you can create complex workflows containing multiple tables and datasets. You create a derived dataset by importing a table or dataset on which the derived dataset is based. The derived dataset depends on the table or dataset you import into it. You can visually see these dependencies by viewing the lineage for a table or dataset.

Viewing the lineage enables you to see dependencies, and to trace the origin of a derived dataset back to its tables and base datasets. The lineage gives you insight into the potential consequences of changes you make to your data (impact analysis).

### Steps

1. Access the Data Catalog report.
2. Right-click a table or dataset whose lineage you want to view, and select View Lineage.

   On the View Table Lineage or View Dataset Lineage report, the graph displays dependencies in both directions from the selected object where applicable:

   - Upstream and downstream dependencies. When you view the lineage of a derived dataset, the graph displays the datasets imported into the derived dataset, and any other derived datasets that import this derived dataset.
   - Downstream dependencies only. When you view the lineage of a table or base dataset, the graph displays any derived datasets that import this table or base dataset.

### Related Information

**Tasks**

View Dataset Dependencies

## 1.6.13 | Reference: Supported File Formats for External Data in Tables and Datasets

To bring in non-Workday data as a table or dataset, Workday parses the data into records (rows) and fields. Datasets support these source file formats:

| Format | Description |
|---|---|

| Format | Description |
|---|---|
| Delimited Text | A delimited file is a plain text file format for describing tabular data. Comma-separated value (CSV) files are the most common delimited files. It refers to any file that:<br>• Is plain text (typically ASCII or Unicode characters)<br>• Has 1 record per line.<br>• Has records divided into fields.<br>• Has the same sequence of fields for every record.<br>Records are separated by line breaks, and fields within a line are separated by a special character called the delimiter (usually a comma or tab character).<br>If the delimiter also exists in the field values, it must be escaped. Workday supports single character escapes (such as a backslash), as well as enclosing field values in double quotes (as is common with CSV files). |

#### 1.6.14 | Reference: WPA_ Fields

When you create any table or a base dataset that uses an integration, Workday automatically creates extra fields in the table or dataset. These fields help you to uniquely identify rows in the table or dataset from different integration runs.

| Field Name | Description |
|---|---|
| WPA_LoadID | This field returns a value of type Text containing a unique identifier of the integration run or data load activity that imported the current row of data into the dataset or table.<br>Workday adds this field to both tables and datasets. |
| WPA_LoadTimestamp | This field returns a value of type Date (to the millisecond) containing the date and time of the integration run that imported the current row of data into the dataset or table.<br>Workday adds this field to both tables and datasets. |
| WPA_RowID | This field returns a value of type Text containing a unique row identifier for each row in a data load activity or integration run.<br>Workday adds this field to tables. |
| WPA_UpdateID | This field returns a value of type Text containing a unique identifier of the data load activity that updated the current row of data in the table.<br>Workday adds this field to tables. |
| WPA_UpdateTimestamp | This field returns a value of type Date (to the millisecond) containing the date and time of the data load activity that updated the current row of data in the table.<br>Workday adds this field to tables. |

You can't modify or delete these fields, but you can hide them. Use these fields with the other fields to uniquely identify rows of data in the table or dataset from multiple integrations.

You can also use these fields to group data together from a single data load or integration. Example: you can create a Group By stage and group on the WPA_LoadID field and Count the number of rows from each integration run.

**Related Information**

**Tasks**

Steps: Create a Dataset with External Data (SFTP Server)
Steps: Create a Dataset Using Workday Data

## 1.7 | Editing Tables

### 1.7.1 | Parse External Data in a Table

**Prerequisites**

Security:

- *Prism: Tables Create* domain in the Prism Analytics functional area when creating a table.
- Any of these security requirements when editing a table:
  - *Prism Datasets: Owner Manage* domain in the Prism Analytics functional area.
  - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
  - *Table Editor* permission on the table.
  - *Table Owner* permission on the table.
  - *Can Insert Table Data* permission on the table.

### Context

When you load a delimited file into a table, you must define how Workday parses the data. You define the parsing options on the Edit Parsing Options step when you load data into a table, such as creating a table by uploading a file, or when adding more rows to an existing table.

Workday supports delimited files that are RFC 4180-compliant. For more information, see RFC 4180.

### Steps

1. Access the Edit Parsing Options step for loading data into a table.
2. Configure the parsing options.
   As you complete this task, consider:

| Option | Description |
|---|---|
| **Row Delimiter** | Specifies the single character that separates rows (or records) in your source data files.<br><br>In most delimited files, rows are separated by a new line, such as the line feed character, carriage return character, or carriage return plus line feed. Line feed is the standard new line representation on UNIX-like operating systems. Other operating systems (such as Windows) might use carriage return individually, or carriage return plus line feed. Selecting Any New Line causes Workday to recognize any of these representations of a new line as the row delimiter. |
| **Field Delimiter** | Specifies the single character that separates the fields (or columns) of a row in your source data files. Comma is the most common field delimiter. |
| **Field Names** | Specifies the default name of each field. You can change the field names after you finish defining the parsing options.<br><br>Field names can't start with a space or with WPA_.<br><br>Workday automatically treats the first line in each source file as a header row instead of as a row of data. If you don't want to use the first line as names for your fields, clear Use values from first row. |
| **Escape Character** | Specifies the single character used to escape the Quote Character or another instance of the Escape Character when a Quote Character is specified. Workday reads an escape character as data only if it's escaped with another escape character.<br><br>If your data values contain quote characters as data, those characters must be escaped and the entire field value must be enclosed with the Quote Character. If not, then Workday assumes that the quote character denotes a new field. |
| **Quote Character** | The character that encloses a single field value, if any.<br><br>Some delimited files use the quote character to enclose individual data values. The quote character is typically the double quote character (").<br><br>If a field value contains a field delimiter as data, then the field value must be enclosed in the Quote Character, otherwise Workday assumes that the field delimiter denotes a new field.<br><br>If a field value contains the quote character as data, then the field value must be enclosed in the Quote Character and it must be escaped, either by the Escape Character or another quote character.<br><br>If a field value contains a row delimiter (such as a new line character) as data, then the field value must be enclosed in the Quote Character.<br><br>Suppose that you have a row with these 3 data values:<br><br>`weekly special`<br>`wine, beer, and soda`<br>`"2 for 1" or 9.99 each`<br><br>If the field delimiter is a comma, the quote character is a double quote, and the escape character is a double quote, then a correctly formatted row in the source data looks like:<br><br>`"weekly special", "wine, beer, and soda", """2 for 1"" or 9.99 each"` |

| Option | Description |
|---|---|
| **Rows to ignore** | Specifies the number of lines at the beginning of the file to ignore when reading the source file. To use this with the Use values from first row option, ensure that the line containing the field names is visible and is the first remaining line. |
| **Jagged Rows** | Select these options when the schema of the source file isn't an exact match of the table schema, and you want Workday to ignore any missing or extra fields at the end of the file schema. |
| **Field Options** | These options control how to handle whitespace characters in Text fields.<br>◦ Trim leading spaces outside quotes. This option removes whitespace characters outside the quotes of Text fields before the quote character.<br>◦ Trim trailing spaces outside quotes. This option removes whitespace characters outside the quotes of Text fields after the quote character. |

### Related Information

**Tasks**
Steps: Create a Table by File Upload

## 1.7.2 | Change Rows in a Table by Uploading a File

### Prerequisites

Any of these security requirements:

- *Prism Datasets: Owner Manage* domain in the Prism Analytics functional area.
- *Prism Datasets: Manage* domain in the Prism Analytics functional area.
- *Table Editor* permission on the table.
- *Table Owner* permission on the table.
- *Can Delete Table Data* permission on the table.
- *Can Insert Table Data* permission on the table.
- *Can Truncate Table Data* permission on the table.
- *Can Update Table Data* permission on the table.

### Context

You can change the rows of data in a table based on data in 1 or more delimited files that you upload. You can:

- Insert new rows.
- Selectively update specific rows based on a key field.
- Selectively delete specific rows based on a key field.
- Insert news rows and update existing rows in the same activity, commonly known as upsert.

How you change the rows in the table depends on the operation you select, such as upsert or delete. The files you upload should contain the data you want to change in the table, such as new rows to insert, or existing rows to update or delete.

When updating or deleting specific rows, you must ensure:

- The target table has a field defined as the external ID.
- The source files must include the external ID field.

### Steps

1. Access the View Table report for a table.
2. From the related actions menu, select Table > Load Data.
3. On the Source step, select 1 or more delimited files to upload.
   When you upload multiple files, each file must use the same schema. Workday supports delimited files that are RFC 4180-compliant. For more information, see RFC 4180.
4. On the Source Options step, define how to parse the data in the files.
   See Parse External Data in a Table.
5. On the Target step, select the Target Operation to perform on the table with the uploaded files.

| Option | Description |
|---|---|
| **Insert** | Workday keeps the existing data in the table and adds to it the new data in these files. This operation is also known as Append. |
| **Truncate and Insert** | Workday deletes the existing data in the table and replaces it with the data in these files. This operation is also known as Replace. |
| **Delete** | Workday deletes a row from the table if it matches a row in the file. To select this operation, the table must already have 1 field configured as the external ID. |
| **Update** | Workday updates a row in the table if the row in the file matches an existing row in the table. To select this operation, the table must already have 1 field configured as the external ID. |

| Option | Description |
|---|---|
| **Upsert** | Workday updates a row in the table if a matching row already exists, and inserts the row if it doesn't exist. To select this operation, the table must already have 1 field configured as the external ID. |

6. (Required for Delete, Update, and Upsert) On the Mapping step, select a field in the target table to use as the operation key.
   Example: When you select Delete as the operation, specify the Delete Key, and when you specify Upsert as the operation, specify the Upsert Key.
7. On the Review step, verify the information before loading the data into the table. You can go back and make any correction if necessary.

### Result

Workday starts a data load activity to change the data in the table based on the data in the files. On the View Table report, click the Activities tab to view the data load progress. Refresh the page to get the most recent status.

### Next Steps

If there were errors changing data into the table, download the error file from the data load activity on the Activities tab of the View Table report.

### Related Information

**Concepts**
Concept: Tables
Concept: Table Error File
**Tasks**
Delete Data from a Table
**Reference**
Reference: Table Field Attributes
Reference: WPA_ Fields

## 1.7.3 | Edit a Table

### Prerequisites

Any of these security requirements:

- *Prism Datasets: Owner Manage* domain in the Prism Analytics functional area.
- *Prism Datasets: Manage* domain in the Prism Analytics functional area.
- *Table Editor* permission on the table.
- *Table Owner* permission on the table.
- *Table Schema Editor* permission on the table.

### Context

You can edit a table by changing the table display name or changing the schema.

You can change the table schema by adding fields, deleting fields, or changing field attributes, such as the field type. However, you can only change field attributes when the table contains no data.

### Steps

1. Access the View Table report for a table.
2. Click Edit, or from the related actions menu, select Table > Edit.
3. (Optional) Click Add Field to add 1 or more fields. In the inspector panel for the field, configure the field attributes.
   See Reference: Table Field Attributes.
4. (Optional) Delete a field by clicking the trash can in the right-most column of a field in the list.
5. (Optional) Change the field attributes of an existing field.
   a. Select a field in the list, and view the field details in the inspector panel on the right side.
   b. In the inspector panel for the field, change the field attributes. You can't change the API name.
      See Reference: Table Field Attributes.
6. Click Next.
7. (Optional) Change the Table Display Name.
8. (Optional) Create or edit 1 or more Tags to organize the table in the Data Catalog.

### Related Information

**Concepts**
Concept: Tables

## 1.7.4 | Reference: Table Field Attributes

When you add or edit a field in a table in the Data Catalog, you define these attributes:

| Field Attribute | Notes |
|---|---|
| Display Name | You can change this name at any time. Field names can't start with a space or with WPA_. |

| Field Attribute | Notes |
|---|---|
| API Name | The API name must be unique in the table. Workday automatically selects an API name based on the field name you enter, modifying it to make it meet the name requirements. Click Change to change the API name. You can't change the API name after you save the table. |
| Field Type | Select the field type that the values in this field must match to be recognized as valid data. You need to configure additional field attributes for some field types you select. |
| Date Format | (Required for Date fields) Select the date format that the values in this field must match to be recognized as valid date data. |
| Digits Before and Digits After | (Required for Numeric fields) Enter the maximum number of digits before and after the decimal point that the values in this field can have to be recognized as valid numeric data. The sum of these 2 options must be less than or equal to 38. |
| Business Object | (Required for Instance fields) Select the business object to associate with the values in this Instance field. |
| Description | (Optional) Add a helpful field description that explains the meaning and data value characteristics of the field. |
| Required | Specifies that the field must contain data. Make a field required to ensure it doesn't contain a NULL value when you insert or update data in the table. When you insert or update data in a table and this field is NULL, Workday rejects the row and instead includes it in the error file. |
| Default Value | Use the Default Value to define a value for a field if the uploaded source file schema doesn't include that field. When the source file schema doesn't include a field, Workday uses the default value for all rows in the source file.<br>Note: The Default Value is only used when the source file schema is missing a field, not when a particular field value is NULL. |
| Use as External ID | Use this attribute to mark a single field in a table as a key. Specify a field as the external ID when the values in the field uniquely identify each row from its source.<br><br>Define a field as the external ID if you want to update or delete data in the table based on data in an external file. This attribute is similar to a primary key in a relational database.<br><br>When using this attribute, consider:<br><br>• You can only define 1 field in a table as the external ID field.<br>• Ensure that each field value in the external ID field is unique. If the field values aren't unique, you'll get unexpected results. Workday doesn't enforce the uniqueness.<br>• You can't define a default value for fields used as an external ID. The field value must come from the external source and can't be NULL.<br>• Workday automatically marks the external ID field as required. |

#### Related Information

**Concepts**
Concept: Tables

## 1.8 | Editing Datasets

### 1.8.1 | Parse External Data in a Dataset

#### Prerequisites

- Base dataset using external data (from uploading a file or connecting to a server) exists in the Data Catalog.
- Security:
  - *Prism Datasets: Create* domain in the Prism Analytics functional area when creating a dataset.
  - Any of these security requirements when editing an existing dataset:
    - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
    - *Dataset Editor* permission on the dataset.
    - *Dataset Owner* permission on the dataset.

#### Context

When you bring external data into a dataset, you must describe the source data in a tabular format. You do this by describing how to parse the data.

Your data must:

- Be in plain text file format.
- Have 1 record per line.
- Have the same sequence of fields for every record separated by a common delimiter (such as a comma or tab).

Delimited records are separated by line breaks, and fields within a line are separated by a special character called the delimiter (usually a comma or tab character). If the delimiter also exists in the field values, it must be escaped. Datasets support single character escapes (such as a backslash), as well as enclosing field values in double quotes.

**Steps**

1. Access the Edit Dataset task for a base dataset using external data.
2. Edit the Parse stage.
   As you complete this task, consider:

| Option | Description |
|---|---|
| **Row Delimiter** | Specifies the single character that separates rows (or records) in your source data files.<br><br>In most delimited files, rows are separated by a new line, such as the line feed character, carriage return character, or carriage return plus line feed. Line feed is the standard new line representation on UNIX-like operating systems. Other operating systems (such as Windows) might use carriage return individually, or carriage return plus line feed. Selecting Any New Line causes Workday to recognize any of these representations of a new line as the row delimiter. |
| **Field Delimiter** | Specifies the single character that separates the fields (or columns) of a row in your source data files. Comma and tab are the most common field delimiters. |
| **Field Names** | Specifies the default name of each field. You can change the field names in the Parse stage after you finish defining the parsing options.<br><br>Field names can't start with ( or with WPA_.<br><br>Workday automatically treats the first line in each source file as a header row instead of as a row of data. If you do not want to use the first line as names for your fields, deselect the Field Names check box. |
| **Escape Character** | Specifies the single character used to escape the Quote Character or another instance of the Escape Character when a Quote Character is specified. Workday reads an escape character as data only if it's escaped with another escape character.<br><br>If your data values contain quote characters as data, those characters must be escaped and the entire field value must be enclosed with the Quote Character. If not, then Workday assumes that the quote character denotes a new field.<br><br>For comma-separated values (CSV) files, it's common practice to escape field delimiters by enclosing the entire field value within double quotes. If your source data uses this convention, then you should specify a Quote Character. |
| **Quote Character** | Some delimited files use the quote character to enclose individual data values. The quote character is typically the double quote character (").<br><br>If a field value contains a field delimiter as data, then the field value must be enclosed in the Quote Character, otherwise Workday assumes that the field delimiter denotes a new field.<br><br>If a field value contains the quote character as data, then the field value must be enclosed in the Quote Character and it must be escaped, either by the Escape Character or another quote character.<br><br>If a field value contains a row delimiter (such as a new line character) as data, then the field value must be enclosed in the Quote Character *and* Field values contain new lines must be selected.<br><br>Suppose that you have a row with these 3 data values:<br><br>`weekly special`<br>`wine, beer, and soda`<br>`"2 for 1" or 9.99 each`<br><br>If the field delimiter is a comma, the quote character is a double quote, and the escape character is a backslash, then a correctly formatted row in the source data looks like:<br><br>`"weekly special","wine, beer, and soda","\"2 for 1\" or 9.99 each"` |
| **Rows to ignore** | Specifies the number of lines at the beginning of the file to ignore when reading the source file while creating and publishing the dataset. To use this with the From First Table Row option, ensure that the line containing the field names is visible and is the first remaining line. |

| Option | Description |
| --- | --- |
| **Field values contain new lines** | Check this option if your source data might contain new line characters as part of a field value.<br><br>When enabled, Workday reads the new line characters inside quote characters as part of the field value instead of as a row delimiter. Workday interprets any row delimiter character outside of quote characters as a new record.<br><br>Enabling this option might impact the time to publish a dataset if Workday reads very large source files.<br><br>Note that you might get unexpected results if you enable this option and the source file has malformed data (such as when a field value has either an opening or closing quote character, but not both). Try to ensure that your source data is well formed when using this option. |
| **Trim trailing and leading whitespace characters in Text fields** | Select this check box if you want to remove whitespace characters at the beginning and end of Text fields. |

### Related Information

**Concepts**
Add a Stage to a Dataset
**Tasks**
Steps: Create a Dataset with External Data (SFTP Server)
Steps: Create a Dataset with External Data (Upload a File)

## 1.8.2 | Add a Prism Calculated Field to a Dataset

### Prerequisites

- Any of these security requirements:
  - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
  - *Dataset Editor* permission on the dataset.
  - *Dataset Owner* permission on the dataset.

### Context

You can transform data in a dataset by adding Prism calculated fields to the dataset. A Prism calculated field has a name, a description, and an expression. The expression describes a processing step you want to perform on other fields in the dataset. You might want to use a Prism calculated field to:

- Convert a field type to another field type.

  Example: You could change an Integer field type to a Long field type, so that you can use the EPOCH_MS_TO_DATE function on it.

- Extract values from a Currency field.

  Example: You could extract the currency codes from a Currency field using the EXTRACT_CODE function.

- Combine the values from 2 Text fields into 1 Text field.

  Example: You could combine separate fields consisting of Last Name and First Name into 1 field using the CONCAT function.

Prism calculated fields contain expressions that can take other fields as input. You might need to create several Prism calculated fields to achieve the result you want. You can use the result of a Prism calculated field in the expressions of other Prism calculated fields to define a chain of processing steps.

Prism calculated fields change the number of fields in a dataset, they don't change the number of records.

To delete a Prism calculated field, access the Edit Dataset task, and click the Prism calculated field you want to remove. Right-click the field header, and select Delete Field. Deleting a field might cause errors if other Prism calculated fields refer to the deleted field.

### Steps

1. Access the Edit Dataset task for a dataset.
2. Select a dataset pipeline (required for derived datasets) and stage into which to add the Prism calculated field.
3. Add a new field.
4. Enter an expression in the expression editor.
   If you use Currency fields that contain different codes, Workday treats the result of those calculations as NULL.
5. In the inspector panel, enter a name.
   Field names can't start with ( or with WPA_.
6. Save the Prism calculated field by clicking Enter or Return on your keyboard.
   Clicking another field on the page also saves the changes to the Prism calculated field.
7. (Optional) You can insert single and multiline comments into any location within a Prism expression. Workday treats all text between these characters as comments: /* */
   Note: Workday won't consider the data values as comments if you enclose these characters and the comment within double quotation marks.

### Related Information

**Concepts**
Concept: Prism Calculated Fields
Concept: Hiding Dataset Fields
Concept: Prism Expression Language

### 1.8.3 | Add a Stage to a Dataset

#### Prerequisites

Any of these security requirements:

- *Prism Datasets: Manage* domain in the Prism Analytics functional area.
- *Dataset Editor* permission on the dataset.
- *Dataset Owner* permission on the dataset.

#### Context

Add stages to change your data from 1 format to another format.

1 way you transform data in a dataset is by adding a stage. There are different types of stages. You can add a new stage to the end of any pipeline in your dataset. Some stage types can only be added to derived datasets.

#### Steps

1. Access the Edit Dataset task for a dataset.
2. (Required for derived datasets) Select the dataset pipeline where you want to add the stage.
   Select the primary pipeline of a derived dataset for some stage types, such as Join or Union.
   Base datasets have only 1 pipeline, which is the primary pipeline.
3. Click Add Stage, and select the type of stage to add. You can add a Group By, Filter, or Manage Fields stage to a pipeline.
4. Configure the stage parameters.

   The parameters you define depend on the type of stage that you're adding.

#### Related Information

**Reference**
Reference: Filter Stages
Reference: Group By Stages
Reference: Join Stages
Reference: Union Stages

### 1.8.4 | Manage Dataset Fields

#### Prerequisites

- *Prism Datasets: Create* domain in the Prism Analytics functional area when creating a dataset.
- Any of these security requirements:
  - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
  - *Dataset Editor* permission on the dataset.
  - *Dataset Owner* permission on the dataset.

#### Context

You can use the Manage Fields stage to view field changes, select fields, and edit fields.

Note: Decide on field names before you define Prism calculated fields and stages. Changing a field name later on will break Prism calculated field expressions and stages that rely on it.

#### Steps

1. Access the Edit Dataset task.
2. Add a Manage Fields stage to edit fields in the dataset.
   Workday recommends that you add the Manage Fields stage at the:
   - Beginning of the primary pipeline of a derived dataset.
   - End of a primary pipeline that you intend to publish.
3. As you complete the Manage Fields stage, consider:

| Option | Description |
|---|---|
| **Input Name** | Clear the check box to hide the field from future stages. You can hide fields to protect sensitive data or to use a calculated field instead of the fields that it's based on. Hide unpopulated or sparse fields or Prism calculated fields that do interim processing. |

| Option | Description |
|---|---|
| **Output Name** | Ensure that the new name:<br>• Doesn't start with ( or with WPA_.<br>• Is unique within the dataset.<br>• Is unique to any potential future field names that come from the source files.<br>When you change a field name in the dataset and add a new field with the same name, you get unexpected results in the data when the schema updates. |
| **Output Type** | The field type determines which functions can use the field as an argument. Create a Prism calculated field to change a field type to a Date or Currency field type or to change a Currency field type to a numeric field type.<br><br>For numeric types, you can specify the number of digits that go before and after the decimal point.<br><br>For Instance and Multi-Instance types, you can select the business object name associated with the values in the field.<br><br>Numeric field types include Integer, Long, Double, or Numeric. |

### Related Information

**Concepts**

Concept: Hiding Dataset Fields
Concept: Prism Calculated Fields
Concept: Table and Dataset Field Types

**Tasks**

Add a Prism Calculated Field to a Dataset
Convert Dataset Text Fields to Date Fields

## 1.8.5 | Change Dataset Field Types

### Prerequisites

- Any of these security requirements:
  - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
  - *Dataset Editor* permission on the dataset.
  - *Dataset Owner* permission on the dataset.

### Context

After importing external data into a dataset, if Workday assigned a different field type than the one you want, then you can change the field type. You might want to change the field type to:

- Accommodate some calculations you want to do.

  Example: You could change an Integer field type to a Long field type to use the EPOCH_MS_TO_DATE function on it.

- Assign a field type to a value that Workday doesn't automatically recognize.

  Example: If Workday doesn't automatically recognize an instance value, change it to the Instance field type.

  Example: If Workday doesn't automatically recognize a date value, change it to the Date field type.

Use the Manage Fields stage to change most field types. However, create a Prism calculated field to make these field type changes:

| From Field Type | To Field Type | Function |
|---|---|---|
| Text | Date | TO_DATE |
| Currency | Numeric, Double, Integer, or Long | EXTRACT_AMOUNT |
| Numeric, Double, Integer, or Long | Currency | BUILD_CURRENCY |
| Text | Currency | TO_CURRENCY |
| Instance, Multi-Instance | Multi-Instance | CREATE_MULTI_INSTANCE |

### Steps

1. Access the Edit Dataset task.
2. Select a dataset pipeline (required for derived datasets).
3. Add a Manage Fields stage.
4. Edit the Manage Fields stage by changing the field type in the Output Type drop-down menu.

Note: Every Instance field type must have a business object name. If you change the field type to Instance, click the settings icon and enter the business object name.

### Related Information

**Concepts**
Concept: Table and Dataset Field Types
Concept: Prism Calculated Fields
**Tasks**
Add a Prism Calculated Field to a Dataset
**Reference**
Reference: Currency Format Requirements for External Data in Datasets

## 1.8.6 | Convert Dataset Text Fields to Date Fields

### Prerequisites

- Dataset must have a field containing date data.
- Any of these security requirements:
  - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
  - *Dataset Editor* permission on the dataset.
  - *Dataset Owner* permission on the dataset.

### Context

When you create a dataset using external data, the data might contain date or time information. When you create a dataset, Workday reads the data to determine which field type to assign to each field. If Workday recognizes the format of a date field, it automatically assigns the Date field type. However, Workday only recognizes some date formats. If Workday doesn't recognize a date format, then it assigns the Text field type to the field.

If Workday doesn't recognize a date field format, you can create a Prism calculated field to convert the Text field type to a Date field type.

### Steps

1. Access the Edit Dataset task for a dataset.
2. Select a dataset pipeline (required for derived datasets) and stage into which to add the Prism calculated field.
3. Add a new field.
4. In the expression editor for this new field, enter an expression that uses the TO_DATE function.
   When you enter the expression using the TO_DATE function, make sure that you use the Text field you want to convert and the date format that best matches the values in that Text field. To quickly enter a field name, type a left square bracket ( [ ).

   Suppose that you have a field called *start_date* that contains data that looks like *25-May-2017*. Use this expression:

   ```
   TO_DATE([start_date], "dd-MMM-yyyy")
   ```

5. In the inspector panel, enter a name.
   Field names can't start with ( or with WPA_.
6. Save the Prism calculated field by clicking Enter on your keyboard.
   Clicking another field on the page also saves the changes to the Prism calculated field.

### Result

Prism creates a new field and populates it by converting the data to the Date field type.

### Related Information

**Concepts**
Concept: Table and Dataset Field Types
Concept: Prism Calculated Fields
**Tasks**
Change Dataset Field Types
Add a Prism Calculated Field to a Dataset
**Reference**
Reference: Supported Date Formats for External Data in Tables and Datasets

## 1.8.7 | Change the Dataset Example Rows

### Prerequisites

- Any of these security requirements:
  - *Prism Datasets: Manage* domain in the Prism Analytics functional area.
  - *Dataset Editor* permission on the dataset.
  - *Dataset Owner* permission on the dataset.

### Context

Workday displays a subset of dataset rows to give you insight into your source data when you edit a dataset. Workday automatically displays the first 1,000 records starting with the first file in the dataset. You can change the number of example rows to read from the source data to display in the dataset workspace.

When you change the number of example rows displayed, Workday reads the data from the files again and updates the statistics and field information for each field in the inspector panel.

You might want to increase the number of example rows to get more precise statistics on each field in the inspector panel. You might want to decrease the number of sample rows to improve performance, especially if the dataset has a lot of fields.

Note: You can disable all example data temporarily to improve responsiveness if your dataset has a lot of fields. After you make some changes to the dataset, such as adding a new stage or calculated field, you can then enable the example data by selecting the number of rows to display.

**Steps**

1. Access the Edit Dataset task for a dataset.
2. Click the menu that lists the number of example rows to display, and select a different number of example rows to view.
   Workday reads the data from the source files again, recalculates the statistics for each field, and displays the rows in the dataset workspace.

### 1.8.8 | Reference: Filter Stages

You can convert filter conditions created in Basic mode to filter expressions in Advanced mode. If you switch back to Basic mode from Advanced mode, you need to define filter conditions again. Workday doesn't convert Advanced mode filter expressions to Basic mode filter conditions.

| Edit Mode | Description |
|---|---|
| Basic | This mode displays prompts to help you define filter criteria, creating filter conditions.<br><br>This mode is useful if you don't want to learn the details of the Prism Analytics expression language to write a filter expression manually. |
| Advanced | This mode enables you to define the filter criteria by writing an expression (the filter expression). Filter expressions must evaluate to true or false.<br><br>This mode is useful if you want to write a filter expression that can’t be expressed in Basic mode. |

## Basic Mode

Define these options when you configure Filter stages in Basic mode:

| Filter Stage Option | Description |
|---|---|
| If All/If Any | • If All—All filter conditions must be met for a row to remain in the output of the Filter stage. This option works like using an AND operator between each filter condition.<br>• If Any—Any filter condition can be met for the row to remain in the output of the Filter stage. This option works like using an OR operator between each filter condition. |
| Filter Condition | Click Add Filter to add a new filter condition. Select a field and operator from the prompts, and then enter a value in the empty text field. Workday reads the value in the text field exactly as is. You don't need to add any quotation marks or escape characters for Text field types.<br><br>If the field you select is a currency field, you can enter an amount and select a currency code or select only a currency code.<br><br>When the filter condition is configured as desired, click the check mark button to save the filter condition to the Filter stage. You can change the filter condition at any time by clicking its edit button. |

## Advanced Mode

In Advanced mode, use the Prism expression language to write a filter expression that evaluates to true or false. The typical format of a filter expression is:

*field_name comparison_operator comparison_value*

The comparison value must be of the same field type as the field in the expression.

You can also use logical operators (such as AND and OR) or arithmetic operators (such as + or /) to define more complex expressions.

When the field name includes a space or a special character, enclose the field name in square brackets: `[ticker symbol]`.

When the comparison value is for a Text field type, enclose the value in double quotes (""). Example:

```
[Zip Code] = "94111-5224"
```

Examples for Text Fields

```
[movie title] LIKE ("* and the*")

[contingent workers] IN ("Larry", "Curly", "Moe")

schedule NOT IN ("Saturday", "Sunday")

status IS NOT NULL
```

Examples for Numeric and Currency Fields

```
TO_STRING([zip code]) LIKE("94*")

[sale price] < 50.00

age >= 21

((EXTRACT_AMOUNT([Total Sales]) > 1000) AND (EXTRACT_CODE_TEXT([Total Sales]) == "USD"))
```

Examples for Date Fields

```
[purchase date] BETWEEN 2019-06-01T00:00:00.000Z AND 2019-07-31T00:00:00.000Z

[graduation] >= 1990-01-01
```

Examples for Instance and Multi-Instance Fields

```
INSTANCE_IS_SUPERSET_OF([Cost Center - Manager], [Cost Center])

[Cost Center] IS EMPTY

[Journal Lines] IS NOT NULL

[Cost Center 1] != [Cost Center 2]

NOT INSTANCE_EQUALS([Cost Center 1], [Cost Center 2])

INSTANCE_CONTAINS_ANY([Regions], "070b0d082eee44e1928c808cc739b35f", "f4c49debb3dc483baa8707dfe683503c")
```

Filter Expressions on Date Type Fields

Filter expressions on Date type fields must be in either of these formats:

*yyyy-MM-ddTHH:mm:ss:SSSZ*
*yyyy-MM-dd*

Don't enclose comparison values for Date fields in quotation marks or use any other punctuation. If the date value is in Text format rather than Date format, the value must be enclosed in quotes like all text values.

When specifying a range of dates, always write the earlier date first.

If the filter expression is a shortened version of the full format, then any values not included are assigned a value of zero (0). Example: the expression `BETWEEN 2019-06-01 AND 2019-07-31` is equivalent to this expression:

```
BETWEEN 2019-06-01T00:00:00.000Z AND 2019-07-31T00:00:00.000Z
```

The expression above doesn't include any values from July 31, 2019. To include values from July 31, 2019, use `BETWEEN 2019-06-01 AND 2019-08-01`.

Filter Expressions on Currency Type Fields

Currency type fields in filter expressions must be in a format recognized by Workday. If a Currency field contains any value that doesn't meet these requirements, Workday treats the value as NULL.

Within an Advanced mode filter expression, you can use multiple currency codes.

Example

```
((EXTRACT_CODE_TEXT([Annual Salary]) == "EUR") AND (EXTRACT_AMOUNT([Annual Salary]) >= 90000)) OR ((EXTRACT_CODE_TEXT([Annual Salary]) == "USD") AND (EXTRACT_AMOU
```

Comments in Filter Expressions

You can insert single and multiline comments into any location within a Prism expression. Workday treats all text between these characters as comments: /* */
Note: Workday won't consider the data values as comments if you enclose these characters and the comment within double quotation marks.

### Related Information

**Concepts**
Concept: Dataset Stages
**Tasks**
Add a Stage to a Dataset
**Reference**
Reference: Join Stages
Reference: Union Stages
Reference: Currency Format Requirements for External Data in Datasets

### 1.8.9 | Reference: Group By Stages

As you complete this stage, consider:

| Group By Stage Option | Description |
|---|---|
| Choose Grouping Fields | Select 1 or more fields by which to group values together. If you summarize (aggregate) values in a group that contains different currency codes, Workday returns NULL values. |
| Add Summarization Fields | Define 1 or more summarization types (aggregate functions) that apply to each grouping field. |

You can select from these summarization types:

| Summarization Type | Description |
|---|---|
| Average | Average returns the average of all valid numeric values for the specified grouping field. It sums all values in the specified field and divides by the number of valid (NOT NULL) rows. You can calculate the average on any numeric field. |
| Count | Count returns the number of rows for the specified grouping field. |
| Max | Max returns the largest (maximum) value from the specified grouping field. You can calculate the maximum on any numeric or date field. |
| Min | Min returns the smallest (minimum) value from the specified grouping field. You can calculate the minimum on any numeric or date field. |
| Sum | Sum returns the total of all values from the specified grouping field. You can calculate the sum on any numeric field. |

#### Related Information

**Concepts**
Concept: Dataset Stages
**Tasks**
Add a Stage to a Dataset
**Reference**
Reference: Filter Stages
Reference: Union Stages

### 1.8.10 | Reference: Join Stages

You can include 2 datasets (dataset pipelines) in a Join stage. You can add additional Join stages in the pipeline if you need to join multiple datasets.

| Join Stage Option | Description |
|---|---|
| Join Pipeline | Select a dataset pipeline to join with the primary pipeline. If there aren't any pipelines available, select Add Another Pipeline to create a pipeline by importing a dataset. |
| Match Rows | Select 1 or more fields from each dataset pipeline whose values should match each other. Select the fields that uniquely identify rows in each dataset pipeline. Defining the matching rows is similar to defining a primary key/foreign key relationship in relational database terms. |

| Join Stage Option | Description |
|---|---|
| Join Type | Select the join type. The join type specifies which rows from each dataset pipeline to include in the join result.<br>• Inner Join. Workday includes rows that have matching values that exist in both pipelines. If a row from 1 pipeline doesn't match a row in the other pipeline, the row is omitted from the join result.<br>• Left Outer Join. Workday includes all rows in the Primary pipeline and searches for a matching row in the other pipeline. If there's no matching row in the other pipeline, Workday populates each field from the other pipeline with NULL values. When the imported pipeline includes multiple matching rows, then Workday includes both rows in the join result.<br>• Right Outer Join. Workday includes all rows in the imported dataset pipeline and searches for a matching row in the Primary pipeline. If there's no matching row in the Primary pipeline, Workday populates each field from the Primary pipeline with NULL values. When the Primary pipeline includes multiple matching rows, then Workday includes both rows in the join result.<br>• Full Outer Join. Workday includes all rows from both pipelines. If a row from 1 pipeline doesn't match a row in the other pipeline, Workday populates each field from the nonmatching pipeline with NULL values. When there are multiple matching rows, Workday includes all rows in the join result.<br>Note that Workday replaces all NULL values with default values when you publish the dataset. |
| Select Fields | Select which fields from each pipeline to include in the join result. Any field you don't include is dropped from that stage in the pipeline and all later stages. |

**Related Information**

**Concepts**
Concept: Dataset Stages
**Tasks**
Add a Stage to a Dataset
**Reference**
Reference: Filter Stages
Reference: Group By Stages

### 1.8.11 | Reference: Union Stages

You can include 2 datasets (dataset pipelines) in a Union stage, and add additional Union stages in the pipeline if you need to combine multiple datasets.

Define at least 1 set of matched fields (field mapping) in a Union stage before saving it. If you don't specify a field in an input dataset for a field mapping, Workday will use a NULL value in that field from that input dataset.

| Union Stage Option | Description |
|---|---|
| Union Pipeline | Select a dataset pipeline to combine with the primary pipeline. If there aren't any pipelines available, select Add Another Pipeline to create a pipeline by importing a dataset. |
| Match Fields—Union Output | The name of the field that will be output from the Union stage for each field mapping. |
| Match Fields—Primary Pipeline | The field from the primary pipeline to match with a field from the union pipeline. |
| Match Fields—*Union pipeline name* | The field from the union pipeline to match with a field from the primary pipeline. |
| Rematch | Click this button if you want to discard all field mappings and return to the default field mappings that Workday detects and configures. |
| Clear All | Click this button to discard all field mappings. Then define at least 1 field mapping in the Union stage before saving it. |
| Include All | Click this button to create a field mapping for each field in the input dataset pipeline. Then you can select which fields to match from the other input dataset pipeline. |

## Preparing Fields From Input Datasets

The datasets you want to combine in a Union stage might not have the same schema. Example: 1 dataset might have first name and last name information in a single field, and the other has that information in 2 fields. When this is the case, you can create Prism calculated fields for each input dataset so they can be combined in a Union stage.

You can create Prism calculated fields for input datasets in these locations:

- In the original dataset. Any Prism calculated field you create in a dataset is available to both that dataset and any derived dataset that imports it as an input dataset.
- In the dataset pipeline of the derived dataset. Any Prism calculated field you create in a pipeline stage of a derived dataset is available only to that derived dataset. It doesn't get pushed back to the original input dataset. You might want to create a Prism calculated field in the pipeline of an imported dataset if you need a field to use a different field type, but you don't want to change the field type of the original dataset. Example: You could create a Prism calculated field to change a zip code field from Integer to Text to match it with a Text zip code field in another input dataset.

### Related Information

**Concepts**
Concept: Dataset Stages
**Tasks**
Add a Stage to a Dataset
**Reference**
Reference: Filter Stages
Reference: Group By Stages
Reference: Join Stages

## 1.8.12 | Reference: Currency Format Requirements for External Data in Datasets

External data that you bring into a dataset might contain fields with currency values. If Workday recognizes the format of a Currency field, it automatically assigns the Currency field type.

For Workday to recognize a single field value as valid currency data, it must meet these requirements:

- It must contain a numeric amount.
- The numeric amount can only use a period as the decimal separator.
- The numeric amount can't use any character to separate thousands.
- It must contain a valid 3-digit currency code that Workday recognizes.
- The 3-digit currency code can occur either before or after the numeric amount.
- It can contain valid currency symbols that Workday recognizes, but it must also contain the 3-digit currency code.
- It can't contain extraneous characters, but can contain extra spaces before or after the numeric amount or currency code.

If a Currency field contains any value that doesn't meet these requirements, Workday treats the value as NULL.

Example: Workday recognizes these single data values as valid currency data:

- 3000.00 USD
- $3000.00 USD
- USD 3000.00
- USD $3000.00
- -$3,000.00 USD
- ($3,000.00) USD

### Related Information

**Concepts**
Concept: Table and Dataset Field Types
**Tasks**
Change Dataset Field Types
Add a Prism Calculated Field to a Dataset

## 1.8.13 | Reference: Supported Date Formats for External Data in Tables and Datasets

External data that you bring into the Data Catalog might contain fields with date or time values. Workday only supports some date formats. How Workday uses the date formats depends on the object you create:

- Table. When you define a Date field in the schema of a table, you can specify any of the supported date formats. The date values in the external data must match the specified date format in order for the row to be valid and loaded into the table.
- Base dataset. If Workday recognizes the format of a date field in the external file, it automatically assigns the Date field type when parsing the file.

Workday supports these date formats as well as any shortened versions of them:

| Format Type | Format |
|---|---|

| Format Type | Format |
|---|---|
| Date and time | yyyy-MM-dd'T'HH:mm:ss.SSSZZ<br>yyyy-MM-dd'T'HH:mm:ssZZ<br>yyyy-MM-dd'T'HH:mm:ss<br>EEE, dd MMM yyyy HH:mm:ss Z<br>MM/dd/yy h:mm:ss a ZZ<br>MM/dd/yy h:mm:ss a<br>MM/dd/yy H:mm:ss ZZ<br>MM/dd/yy H:mm:ss<br>yy-MM-dd h:mm:ss a ZZ<br>yy-MM-dd h:mm:ss a<br>yy-MM-dd H:mm:ss ZZ<br>yy-MM-dd H:mm:ss<br>yyyy-MM-dd HH:mm:ss.SSS |
| Date only | yyyy-MM-ddZZ<br>yy-MM-dd<br>yyyy-MM-dd<br>MM/dd/yy |
| Time only | 'T'HH:mm:ssZZ<br>'T'HH:mm:ss<br>HH:mm:ssZZ<br>HH:mm:ss |

**Related Information**

**Concepts**
Concept: Table and Dataset Field Types
**Tasks**
Change Dataset Field Types
**Reference**
TO_DATE

### 1.8.14 | Reference: Date Format Symbols

Workday recognizes specific characters as symbols to represent part of a date format when you create and edit tables and datasets. This section describes the symbols to use and the patterns use them in when you define your date format. The count and order of the symbols determine the date format.

Workday treats any characters in the pattern that aren't in the ranges of a-z or A-Z as quoted delimiter text. Example: Workday treats the slash (/) and colon (:) characters delimiter text even if they aren't escaped with single quotes.

| Symbol | Meaning | Presentation | Examples | Notes |
|---|---|---|---|---|
| G | era | text | AD | |
| C | century of era (0 or greater) | number | 20 | |
| Y | year of era (0 or greater) | year | 1996 | Numeric presentation for year and week year fields are handled specially. Example: If the count of 'y' is 2, the year will be displayed as the zero-based year of the century, which is two digits. |

| Symbol | Meaning | Presentation | Examples | Notes |
|---|---|---|---|---|
| x | week year | year | 1996 | Numeric presentation for year and week year fields are handled specially. Example: If the count of 'y' is 2, the year will be displayed as the zero-based year of the century, which is two digits. |
| w | week number of week year | number | 27 | |
| e | day of week (number) | number | 2 | |
| E | day of week (name) | text | Tuesday; Tue | If the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used. |
| y | year | year | 1996 | |
| D | day of year | number | 189 | |
| M | month of year | month | July; Jul; 07 | 3 or more uses text, otherwise uses a number |
| d | day of month | number | 10 | If the number of pattern letters is 3 or more, the text form is used; otherwise the number is used. |
| a | half day of day | text | PM | |
| K | hour of half day (0-11) | number | 0 | |
| h | clock hour of half day (1-12) | number | 12 | |
| H | hour of day (0-23) | number | 0 | |
| k | clock hour of day (1-24) | number | 24 | |
| m | minute of hour | number | 30 | |
| s | second of minute | number | 55 | |
| S | fraction of second | number | 978 | |
| z | time zone | text | Pacific Standard Time; PST | If the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used. |
| Z | time zone offset/id | zone | -0800; -08:00; America/Los_Angeles | 'Z' outputs offset without a colon, 'ZZ' outputs the offset with a colon, 'ZZZ' or more outputs the zone ID. |
| ' | escape character for text-based delimiters | delimiter | | |
| '' | literal representation of a single quote | literal | ' | |

## 1.9 | Securing Data in Tables and Datasets

### 1.9.1 | Set Up Table Sharing

#### Prerequisites

Security: *Set Up: Assignable Roles* domain in the Organization and Roles functional area.

#### Context

To enable sharing a table with another user or security group, you create tenant-specific roles that correspond to the table-related Workday provided roles. Sharing tables is a way to control access to individual tables.

#### Steps

1. Access the Maintain Assignable Roles task.
2. Set up roles for the table-related Workday-provided roles.

When you set up these tenant-specific roles, consider:

- The name you enter in Role Name becomes the prompt value in the Permission column on the Edit Table Sharing task.
- The security groups you select in Role Assignees Restricted To determine which users and groups you can share a table with.
- The security groups you select in Assigned by Security Groups determine which users can share a table.

Add the roles in the table below, but substitute security groups that your organization needs instead of Prism Data Writer and Prism Data Administrator:

| Role Name | Workday Role | Enabled for | Self-Assign | Role Assignees Restricted to | Assigned by Security Groups |
|---|---|---|---|---|---|
| Table Owner - Prism | 01. Table Owner - Prism | Prism Tables | Yes | Prism Data Writer | Prism Data Administrator<br>Prism Table Owner (Workday Owned)<br>Security Administrator |
| Table Editor - Prism | 02. Table Editor - Prism | Prism Tables | | Prism Data Writer | Prism Data Administrator<br>Prism Table Owner (Workday Owned)<br>Security Administrator |
| Table Schema Editor - Prism | 03. Table Schema Editor - Prism | Prism Tables | | Prism Data Writer | Prism Data Administrator<br>Prism Table Owner (Workday Owned)<br>Security Administrator |
| Table Viewer - Prism | 04. Table Viewer - Prism | Prism Tables | | Prism Data Writer | Prism Data Administrator<br>Prism Table Owner (Workday Owned)<br>Security Administrator |
| Table Schema Viewer - Prism | 05. Table Schema Viewer - Prism | Prism Tables | | Prism Data Writer | Prism Data Administrator<br>Prism Table Owner (Workday Owned)<br>Security Administrator |
| Can Truncate Table Data - Prism | 06. Can Truncate Table Data - Prism | Prism Tables | | Prism Data Writer | Prism Data Administrator<br>Prism Table Owner (Workday Owned)<br>Security Administrator |
| Can Delete Table Data - Prism | 07. Can Delete Table Data - Prism | Prism Tables | | Prism Data Writer | Prism Data Administrator<br>Prism Table Owner (Workday Owned)<br>Security Administrator |
| Can Update Table Data - Prism | 08. Can Update Table Data - Prism | Prism Tables | | Prism Data Writer | Prism Data Administrator<br>Prism Table Owner (Workday Owned)<br>Security Administrator |
| Can Insert Table Data - Prism | 09. Can Insert Table Data - Prism | Prism Tables | | Prism Data Writer | Prism Data Administrator<br>Prism Table Owner (Workday Owned)<br>Security Administrator |

| Role Name | Workday Role | Enabled for | Self-Assign | Role Assignees Restricted to | Assigned by Security Groups |
|---|---|---|---|---|---|
| Can Select Table Data - Prism | 10. Can Select Table Data - Prism | Prism Tables | | Prism Data Writer | Prism Data Administrator<br>Prism Table Owner (Workday Owned)<br>Security Administrator |

Note: You can substitute any user-based security groups your organization has created for Prism-related tasks instead of selecting Prism Data Writer or Prism Data Administrator.

### Related Information

**Concepts**
Concept: Sharing Tables and Datasets
**Tasks**
Share a Table with Others

## 1.9.2 | Set Up Dataset Sharing

### Prerequisites

Security: *Set Up: Assignable Roles* domain in the Organization and Roles functional area.

### Context

To enable sharing a dataset with another user or security group, you create tenant-specific roles that correspond to the dataset-related Workday provided roles. Sharing datasets is a way to control access to individual datasets.

### Steps

1. Access the Maintain Assignable Roles task.
2. Set up roles for these Workday-provided roles: *Prism Dataset Owner*, *Prism Dataset Editor*, and *Prism Dataset Viewer*.

   When you set up these tenant-specific roles, consider:

   - The name you enter in Role Name becomes the prompt value in the Permission column on the Edit Dataset Sharing task.
   - The security groups you select in Role Assignees Restricted To determine which users and groups you can share a dataset with.
   - The security groups you select in Assigned by Security Groups determine which users can share a dataset.

   Add the roles in the table below, but substitute security groups that your organization needs instead of Prism Data Writer and Prism Data Administrator:

| Role Name | Workday Role | Enabled for | Self-Assign | Role Assignees Restricted to | Assigned by Security Groups |
|---|---|---|---|---|---|
| Dataset Owner | Prism Dataset Owner | Prism Dataset | Yes | Prism Data Writer | Prism Data Administrator<br>Prism Dataset Owner (Workday Owned)<br>Security Administrator |
| Dataset Editor | Prism Dataset Editor | Prism Dataset | | Prism Data Writer | Prism Data Administrator<br>Prism Dataset Owner (Workday Owned)<br>Security Administrator |
| Dataset Viewer | Prism Dataset Viewer | Prism Dataset | | Prism Data Writer | Prism Data Administrator<br>Prism Dataset Owner (Workday Owned)<br>Security Administrator |

Note: You can substitute any user-based security groups your organization has created for Prism-related tasks instead of selecting Prism Data Writer or Prism Data Administrator.

### Related Information

**Concepts**
Concept: Sharing Tables and Datasets
**Tasks**
Share a Dataset with Others

### 1.9.3 | Share a Table with Others

#### Prerequisites

Any of these security requirements:

- *Prism: Tables Owner Manage* domain in the Prism Analytics functional area.
- *Table Owner* permission on the table.

#### Context

When you create a table, you're the table owner. Being the owner of a table means you have Table Owner permission on the table. As a table owner, you can share it with other users and security groups. The table sharing feature is a way to control access to individual tables.

You might want to share a table with someone else so they can edit it, or import it into a derived dataset.

Table permissions control either:

- Metadata. Example: Table Schema Editor, Table Schema Viewer.
- Data. Example: Can Delete Table Data, Can Truncate Table Data, Can Select Table Data, Can Insert Table Data.
- Metadata and Data. Example: Table Owner, Table Editor, Table Viewer.

#### Steps

1. Access the View Table report.
2. Select Actions > Security > Edit Table Sharing.
3. Assign a user or security group to the desired Permission.
   Note: When you grant someone one of the data permissions, that user must have access to view the table schema. Example: If you want to grant Can Insert Table Data permission, you must also grant either Table Schema Viewer or Table Viewer permission.
4. (Optional) Share the table with more users, or remove access from users listed in the table.
5. When you're done sharing the table with others, click OK.

#### Related Information

**Concepts**
Concept: Sharing Tables and Datasets
**Tasks**
Set Up Table Sharing

### 1.9.4 | Share a Dataset with Others

#### Prerequisites

Any of these security requirements:

- *Prism Datasets: Owner Manage* domain in the Prism Analytics functional area.
- *Dataset Owner* permission on the dataset.

#### Context

When you create a dataset, you're the dataset owner. Being the owner of a dataset means you have Dataset Owner permission on the dataset. As a dataset owner, you can share it with other users and security groups. The dataset sharing feature is a way to control access to individual datasets.

You might want to share a dataset with someone else so they can edit it, or import it into another (derived) dataset.

Note: If you want to share a derived dataset with someone, they must also have at least viewer permission on all upstream objects up until you reach the base datasets and tables, or until you reach a dataset with Relax Sharing Rules enabled and functional.

#### Steps

1. Access the View Dataset report.
2. Select Actions > Security > Edit Dataset Sharing.
3. Add a new row to the table, and select the Permission to assign to a user or security group.
   You can configure these permissions:

| Option | Description |
|---|---|
| **Dataset Viewer** | Users with this permission can:<br>◦ View this dataset.<br>◦ Import this dataset into a derived dataset. |
| **Dataset Editor** | User with this permission can do all tasks of a dataset viewer plus:<br>◦ Make changes to (edit) this dataset. |
| **Dataset Owner** | User with this permission can do all tasks of a dataset editor plus:<br>◦ Delete this dataset.<br>◦ Share this dataset. |

4. (Optional) Share the dataset with more users, or remove access from users listed in the table.
5. When you're done sharing the dataset with others, click OK.

### Related Information

**Concepts**

Concept: Security in Prism Analytics

## 1.9.5 | Edit Prism Data Source Security in a Dataset

### Prerequisites

- Security: *Prism: Manage Data Source* domain in the Prism Analytics functional area.

### Context

Before you publish a dataset, configure the security (security domains and securing entities) that Workday applies to the data in the Prism data source. The configured securing entities work with the configured security domains and their security groups to determine which users have access to which rows, fields, and field values in a Prism data source.

A securing entity is an Instance or Multi-Instance field that you use to constrain access to particular instance values for reporting and analytics. A securing entity:

- Is typically a role-enabled Instance field, such as Cost Center or Supervisory Organization.
- Is the Person Instance field (secured to the *Person Data: Person Reports* domain) for self-service security groups.
- Determines which instance values Workday displays to a user based on the role assigned to the user.

Use securing entities to control row-level and field value-level access in a Prism data source for users in constrained security groups.

For a user to have access to a particular row or field value in a Prism data source, they must be a member of 1 of these security groups:

- An unconstrained security group that has permissions on a domain configured in the data source security.
- A constrained security group that has permissions on a domain configured in the data source security, and the corresponding securing entity is configured.

Workday restricts user access to data in a Prism data source for these security groups:

- All unconstrained
- Role-based constrained
- Aggregation when role-based
- Intersection when role-based

Workday has tested and supports using securing entity fields that use these business objects:

- Company
- Company Hierarchy
- Cost Center
- Cost Center Hierarchy
- Person
- Location Hierarchy
- Region
- Region Hierarchy
- Supervisory Organization

### Steps

1. Access the View Dataset report for the dataset you want to apply security to.
2. Select Actions > Security > Edit Data Source Security.
3. In the Domains prompt, select 1 or more security domains to use to determine who can see the Prism data source.
   If you specify a security domain that has a constrained security group, then you must specify an appropriate securing entity.
4. (Optional) In the Securing Entities prompt, select 1 or more Instance or Multi-Instance fields in the dataset.
   The securing entities work with the:
   - Data Source Security domains to determine row-level access for a user.
   - Field Level Security domains to determine field value-level access for a user.

   Note: Workday uses any in common logic when evaluating the contextual security using a Multi-Instance field.
   Note: When you specify more than 1 securing entity that relates to the same security group, Workday uses the OR condition between them. Depending on how your security groups are set up, a user might see some additional rows or field values. Make sure you test the report results to ensure that the report produces expected results for each user.
5. In the Default Domain(s) for Dataset Fields prompt, select 1 or more security domains that Workday applies to every field in the dataset unless you override the domain for a particular field in the next section.
   Note: When you add new fields to the dataset, Workday applies this default domain to the new fields. You might want to consider specifying a domain with more restrictive access. Then you can override the default domain on a per field basis to allow more access as necessary.
6. (Optional) You can select different domains to apply to specific dataset fields to override the default domains.
7. Review any Security Configuration Audit messages to learn more about any issues with the configured securing entities and domains.
8. (Optional) Click Back to make any changes to the configured security options based on the audit messages.
9. Select the Apply Security check box to apply your changes.

   If you want to restrict access to rows using any of these security group types, Workday can't honor those restrictions:

   - Segment-based security groups
   - Job-based security groups
   - Manager's Manager security group

### Result

Workday saves the security information. You can view the current security status by selecting Actions > Security > View Data Source Security.

**Example**

Suppose that you select these domains containing these security groups. To enforce contextual security at the row-level and field value-level, then use these fields as securing entities:

| Security Domain | Contains This Security Group | Use This Securing Entity |
|---|---|---|
| Custom Domain 28 | HR Partner (By Location) | Location |
| Custom Domain 29 | Manager | Supervisory Organization |
| Custom Domain 30 | HR Administrator | None required.<br>HR Administrator is an unconstrained security group, so it doesn't require a securing entity. |
| Public Reporting Items | None. | None required.<br>This domain provides access to all publicly available fields and Workday-delivered data sources. |

**Next Steps**

Publish the dataset to create the Prism data source. Workday applies the security restrictions to the data in the Prism data source.

### 1.9.6 | Concept: Security in Prism Analytics

Prism Analytics uses Workday's strong, flexible, and configurable security model to control access to data, objects, and tasks. Which users have access to what data depends on where in the Prism data workflow they are.

## Phase 1 and Phase 2: Create and edit tables and datasets

The first phase in the data management workflow is to bring data into the Prism Analytics Data Catalog. You bring in data by creating a table or a base dataset. You transform data by creating a derived dataset.

Datasets include metadata and a subset of data as a small collection of example rows. Tables include metadata and all data rows in the table. When it comes to security with tables and datasets, you control access to the metadata and data together.

Workday controls who can do what with tables and datasets in these ways:

- **Security administrator grants access.** Your Workday security administrator can configure Workday security domains to grant groups of users to be able to create, edit, or manage tables and datasets. Depending on how the administrator configures the security domains, some users might be able to create, view, edit, or act as an owner of all tables or datasets. Your Workday security administrator can configure these Workday security domains so that groups of users can create, edit, or manage tables and datasets:
  - *Prism Datasets: Create*
  - *Prism Datasets: Manage*
  - *Prism Datasets: Owner Manage*
  - *Prism: Manage Data Source*
  - *Prism: Tables Create*
  - *Prism: Tables Manage*
  - *Prism: Tables Owner Manage*
- **Table sharing and dataset sharing.** The user who created a table or dataset can share it with particular users and grant different levels of access to each user.

  When you create a table or dataset, you're the table owner or dataset owner. Being the owner means that you have Table Owner or Dataset Owner permission on the table or dataset. As an owner, you can grant different levels of access by assigning permissions to another user. Example: You can assign Table Viewer or Can Truncate Table Data permission to a table, or Dataset Editor permission to a dataset.

Access to a table or dataset is unconstrained. This means that any user who can create or view a table or dataset can view all fields and data (example data for datasets), regardless of the origin of the data. When you create a dataset from a Workday report, Workday removes all security domains configured for the business objects in the dataset.

This unconstrained access only applies when you use the View Table, Edit Table, View Dataset, and Edit Dataset tasks and reports. It doesn't apply to the published data in a Prism data source.

Instead, you define the security to apply to the published data at the end of the dataset editing process, before publishing the dataset.

## Phase 3: Apply security to the published data.

After you've configured a dataset to transform its data as you need, it's time to apply security to the published data. By applying security to the published data, you can ensure that both Workday and non-Workday data have the proper restrictions applied when viewed in a report.

You configure the published data security by editing the dataset, but Workday applies the security to the published data in the form of a Prism data source.

To apply security to published data, you must be in the *Prism: Manage Data Source* security domain. From the dataset Actions menu, select Security > Edit Data Source Security.

You can restrict access to the published data in a Prism data source at these levels:

- **Data source-level.** Specify 1 or more security domains that apply to the Prism data source that Workday creates. These domains determine which users can see the Prism data source. This is sometimes known as table-level security. If no domains are configured, Workday uses the *Prism: Default to Dataset Access* domain.
- **Row-level.** Optionally, you can enforce row-level security by specifying in the Securing Entities prompt 1 or more Instance or Multi-Instance fields in the dataset, such as Supervisory Organization. The securing entities work with the configured data source-level security domains to determine which users have access to which rows (and field values) in a Prism data source.
- **Field-level.** Specify 1 or more security domains to apply to the fields in the Prism data source. These domains determine which users can see each field in the Prism data source.
- **Field value-level.** Workday uses any configured Securing Entities with the configured field-level security domains to determine which users have access to which field values in a Prism data source.

## Phase 4: Publish a dataset.

You make the data described in a dataset available for analysis by publishing a dataset. When you publish a dataset, Workday creates the Prism data source, loads it with the transformed data, and applies the appropriate security restrictions to the data.

To publish a dataset with data source security applied, you must have access to the *Prism Datasets: Publish* security domain.

Optionally, you can publish the dataset without any data source security applied. When you publish with no data source security applied, Workday applies the *Prism: Default to Dataset Access* security domain to the Prism data source. The *Prism: Default to Dataset Access* domain provides contextual access to a Prism data source based on your access to the underlying dataset.

## Phase 5: Analyze and visualize the data.

Workday controls access to data in a Prism data source according to the data source security applied before publishing. The data source security applied determines who can see the Prism data source, and which rows, fields, and field values each user can see when they query the Prism data source in a visualization or report.

### Related Information

**Tasks**
Share a Dataset with Others
Edit Prism Data Source Security in a Dataset

### 1.9.7 | Concept: Sharing Tables and Datasets

Workday enables you to have fine-grained control over what you can do with tables and datasets. Sharing tables and datasets is a way to control access to individual tables and datasets.

When your tenant is set up for table and dataset sharing, table owners and dataset owners can share a table or dataset with another user or security group. Example: You can control who can view a dataset, edit the schema of a table, insert data into a table, or delete table data.

## How Inherited Permissions Work to Enable Table and Dataset Sharing

Workday provides sharing permission control using Workday roles, Workday-owned security groups, and inherited permissions.

Most Workday roles are tied to an organization. However, table-related and dataset-related Workday roles are tied to an object type, the table or dataset. By tying a role to an object type, Workday enables you to control which permissions a user inherits for a particular table or dataset.

Workday maps each table-related and dataset-related role to a Workday owned security group, and that security group automatically inherits permissions from 1 or more security domains.

Example: Workday maps the Workday role "Table Schema Editor" to the "Prism Table Schema Editor (Workday Owned)" security group, and that security group inherits View and Modify permissions on the *Prism: Tables Manage Schema* domain.

Example: Workday maps the Workday role "Prism Dataset Editor" to the "Prism Dataset Editor (Workday Owned)" security group, and that security group inherits View and Modify permissions on the *Prism: Datasets Manage* domain.

As a result of these connections, you can enable table and dataset sharing by creating a tenant-specific role and mapping it to a table-related or dataset-related Workday role. The tenant-specific role becomes the table or dataset permission that you can share with others.

### Related Information

**Tasks**
Share a Table with Others
Share a Dataset with Others

### 1.9.8 | Concept: Relax Sharing Options

By default, to access the current dataset, you must have access to all upstream datasets and tables. Example: To share a derived dataset, you must have owner permission on the derived dataset and owner permission on all upstream datasets and tables.

However, you can relax some permission requirements so that users require access on fewer upstream datasets and tables to have access to the current dataset.

To relax the sharing restrictions with tables and datasets, Workday provides these options when you edit dataset sharing and table sharing:

| Option Name | Description |
|---|---|

| Option Name | Description |
|---|---|
| Relax Sharing Rules | When you enable Relax Sharing Rules on a table or dataset:<br>• Owners of datasets derived from this table or dataset only need a viewer role on this table or dataset to share their derived dataset.<br>• (Derived datasets only) Owners of this dataset can share it with other users without requiring those users to have a role on any upstream datasets or tables.<br>You can protect sensitive data in upstream datasets and tables by eliminating the need for access by using Relax Sharing Rules in downstream datasets. |
| Prevent Relax Sharing on Derived Datasets | When you enable Prevent Relax Sharing on Derived Datasets, Workday revokes the Relax Sharing Rules functionality on all downstream derived datasets.<br>You might want to enable Prevent Relax Sharing on Derived Datasets to prevent others from sharing derived datasets they own without requiring your permission on this table or dataset.<br>Note: The Relax Sharing Rules option is only functional when no dataset or table upstream from it has Prevent Relax Sharing Rules on Downstream Datasets enabled. |

To configure these options, you must have access to the *Prism: Manage Relax Sharing* domain in the Prism Analytics functional area.

Example: You create a table called Payrolls that contains sensitive data, and you create a derived dataset off of it called Filtered Payrolls that filters out the sensitive data. You want Betty Liu to view the Filtered Payrolls dataset, but not the Payrolls table. To accomplish this sharing scenario, you enable Relax Sharing Rules on Filtered Payrolls, and then assign Betty Liu viewer permission on it. Now, Betty can view Filtered Payrolls and create a derived dataset from it even though she doesn't have viewer permission on the Payrolls table. Also, she can share the derived dataset she created without having owner permission on Filtered Payrolls or Payrolls.

| To do this... | You must have... |
|---|---|
| • View the current dataset or table.<br>• Copy the current dataset or table.<br>• Import the current dataset or table into a derived dataset. | • At least viewer permission on the current dataset or table.<br>• At least viewer permission on all upstream objects up until you reach the base datasets and tables, or until you reach a dataset with Relax Sharing Rules enabled and functional. |
| Edit the current dataset or table. | • At least editor permission on the current dataset or table.<br>• At least viewer permission on all upstream objects up until you reach the base datasets and tables, or until you reach a dataset with Relax Sharing Rules enabled and functional. |
| Share the current dataset or table (the action that is specific to owners). | Owner permission on the current dataset or table, and either:<br>• Owner permission on all upstream datasets and tables, or<br>• At least owner permission on all upstream objects up until you reach a dataset or table with Relax Sharing Rules enabled and functional. On the dataset or table with Relax Sharing enabled, you only need viewer permission. |
| View the lineage of the current table or dataset. | At least viewer permission on the current dataset or table.<br>Note that when you view lineage, you only see the upstream or downstream tables and dataset on which you have viewer permission (or better). |

#### Related Information

**Tasks**
Share a Table with Others
Share a Dataset with Others

## 1.10 | Preparing Datasets for Publishing

### 1.10.1 | Enable Contextual Publishing for Datasets

#### Prerequisites

Security: These domains in the System functional area:

- *Security Activation*
- *Security Configuration*

### Context

When you create and edit datasets, you typically need to build reports and visualizations to test the data you're transforming in the dataset. To test your work, you must publish your dataset, which requires access to the *Prism Datasets: Publish* domain.

You can set up your tenant to restrict users to publish datasets based on their contextual access to the dataset. Contextual publishing enables dataset users to publish a dataset based on their dataset permission, such as Dataset Owner or Dataset Editor.

You might want to enable contextual publishing to enable dataset users to test their work without giving them unconstrained access to publish all datasets.

To enable contextual publishing, you must have already enabled dataset sharing by creating assignable roles that correspond to dataset-related Workday roles.

### Steps

1. Create a role-based (constrained) security group.

   Select 1 of these roles for the Assignable Role:

   - Dataset Owner
   - Dataset Editor

   Select Role has access to the positions they support for the Access Rights to Multiple Job Workers option.

   See Create Role-Based Security Groups.

   You can create a role-based (constrained) security group to enable users to publish datasets they have Dataset Owner permission on. Example:

| Security Group Property | Example |
|---|---|
| Name | Dataset Owner (role-based) |
| Assignable Role | Dataset Owner |
| Access Rights to Organizations | Applies To Current Organization And Unassigned Subordinates |
| Access Rights to Multiple Job Workers | Role has access to the positions they support |

2. Edit Domain Security Policies.
   Edit the domain security policy for the *Prism Datasets: Publish* domain in the Prism Analytics functional area. Add the role-based (constrained) security group you created, and assign both View and Modify task permissions.
3. Activate Pending Security Policy Changes.

## 1.10.2 | Publish a Dataset as a Prism Data Source Manually

### Prerequisites

- Security: *Prism Datasets: Publish* domain in the Prism Analytics functional area.

### Context

This section describes how to publish a dataset immediately on an ad hoc basis.

### Steps

1. Access the View Dataset report for the dataset you want to publish.
2. Click Publish, or from the related actions menu, select Publishing > Publish Dataset.
   Workday:
   - Reads the source data in the dataset.
   - Transforms the source data using the transformation logic defined in the dataset.
   - Creates a Prism data source and loads it with the transformed data. The Prism data source has the same name as the dataset.
   - Applies the appropriate security restrictions to the data.
3. View the publishing process status from the Publishing Activities tab on the View Dataset report.
   Refresh the browser to see the most current status. The status includes the date and time that Workday last published the dataset successfully. The last successful published date informs you about the freshness of the data in the Prism data source. Example: If the last successful publish date is 1 week ago, but your publishing schedule is set to publish daily, this discrepancy could indicate a failure in the publishing process.
   You can also view the status of manually run publish requests in the Process Monitor report.
4. (Optional) You can cancel the publishing process by clicking Cancel Publishing on the View Dataset report.

### Result

You can view the Prism data source by accessing the View Prism Data Source report, and selecting the name of the dataset you published.

### Related Information

**Tasks**
Unpublish a Dataset

## 1.10.3 | Create Dataset Publish Schedules

### Prerequisites

Security: *Prism Datasets: Publish* domain in the Prism Analytics functional area.

### Context

You can create a publish schedule for a dataset. You can schedule the publish to run:

- On a recurring basis (Example: daily, weekly, or monthly)
- Only if another Prism scheduled process completes at a status you specify.

### Steps

1. Access the View Dataset report for the dataset you want to publish.
2. From the related actions menu, select Publishing > Create Schedule.
3. In Run Frequency, specify how often to publish the dataset. The choices include creating a dependent publish schedule.
4. Select the criteria for the schedule.
5. (Recurring schedules) As you configure the schedule, consider:

| Option | Description |
|---|---|
| **Priority** | Unavailable for publish schedules. |
| **Catch Up Behavior** | Select how many times the scheduled publish runs after maintenance issues cause errors.<br><br>Example: If you schedule a publish to run multiple times in a week when your environment is down for maintenance, you can limit the process to run once instead of catching up all missed occurrences. |

6. (Dependent schedules) As you configure the schedule, consider:

| Option | Description |
|---|---|
| **Dependency** | Select By Type and 1 of the Prism-related scheduled future process types on which the publish depends:<br>◦ Prism Data Acquisition Future Process<br>◦ Scheduled Publish Dataset<br><br>Select the schedule on which the publish depends. |
| **Trigger on Status** | Select the status of the scheduled future process that triggers publishing the dataset.<br><br>Example: You select Scheduled Publish Dataset as your process type and a future publish schedule called All Currencies. In the Trigger on Status field, you select Completed.<br><br>Workday publishes the dataset only after the scheduled All Currencies publish successfully completes. |
| **Time Delayed Configuration** | (Optional) Specify the number of days, hours, or minutes to delay triggering the publish. You might want to delay publishing to review the latest source files. |

7. (Optional) Change the name of your publish schedule.

   Workday assigns a name to the schedule based on the name of the dataset and prepends *Publish Schedule:* to the name. You can change the name of the schedule in the Request Name field when you edit the schedule. Workday displays this name in the Process Monitor and Scheduled Future Processes reports to help you identify a specific process request.

### Result

Workday publishes a dataset based on the criteria you specified. When a dataset is published, Workday:

- Reads the source data in the dataset.
- Transforms the source data using the transformation logic defined in the dataset.
- Creates a Prism data source if it doesn't already exist and loads it with the transformed data. The Prism data source has the same name as the dataset.
- Applies the appropriate security restrictions to the data.

View the status of all scheduled publishing processes on the Activities page.

Refresh the browser to see the most current status. The status includes the date and time (UTC) that the dataset was last published successfully. The last successful published date informs you about the freshness of the data in the Prism data source. Example: If the last successful published date is 1 week ago, but your publish schedule is set to publish daily, this discrepancy could indicate a failure in the publishing process.

### Related Information

**Concepts**

Concept: Security in Prism Analytics

**Tasks**

Manage Scheduled Future Processes
**Reference**
FAQ: Dataset Publish Schedules
**Examples**
Example: Create Dependent Publish Schedules for Datasets

### 1.10.4 | Concept: Publishing Datasets

You make data available for analysis by publishing a dataset. When you publish a dataset, Workday creates a Prism data source and:

- Loads it with the transformed data.
- Applies the appropriate security restrictions to the data.

Workday enables you to publish a dataset immediately on an ad hoc basis or according to a preconfigured schedule.

You can schedule a publish to run:

- On a recurring basis (Example: daily, weekly, or monthly).
- Only after another Prism scheduled process completes at a status you specify.

You can't publish the same dataset at times that overlap with each other.

### 1.10.5 | Concept: Dataset Publish Schedules

Schedules for publishing datasets enable you to specify when, how often, and under what criteria to publish a dataset. Publish schedules differ from publishing immediately on an ad hoc basis using Run Now.

## Publish Schedule Types

You can create these types of schedules:

| Schedule Type | Description |
|---|---|
| Recurring | A publish schedule that runs at specified intervals, such as daily, weekly, or monthly. |
| Dependent | A publish schedule that depends on the completion of another Prism scheduled process. For your dependency criteria, you can specify:<br>• The process type, such as bringing data into a dataset.<br>• The status that triggers publishing, such as the process type successfully completing.<br>Example: When the *Prism Data Acquisition Future Process* completes with no warnings or errors, begin publishing.<br>Note: A Prism Analytics publish schedule can depend only on another Prism-related process, such as bringing data into a dataset or publishing another dataset. |

After you create a publish schedule, consider these actions that you can perform on it:

| Action | Description |
|---|---|
| Activate | Activate a suspended publish schedule. |
| Change Schedule (recurring schedules only) | Edit the run frequency (daily, monthly, weekly), start time, and date range for the publish schedule. You can also change to another scheduled recurring process. |
| Delete | Permanently delete the publish schedule. |
| Edit Environment Restrictions | Select the environment in which you want the scheduled publish to run. |
| Edit Schedule | (Recurring Schedules) Edit the schedule name, recurrence criteria, and range of recurrence dates.<br>Note: To change the run frequency, use Change Schedule.<br>(Dependent Schedules) Edit the schedule name, dependency, trigger status, and timed delay configurations. |
| Edit Scheduled Occurrence (recurring schedules only) | Update the schedule date and time for one particular occurrence of the scheduled request.<br>You can also delete a particular occurrence of the scheduled publish. |

| Action | Description |
|---|---|
| Expire Schedule | Permanently stop the publish schedule. Expiring a schedule doesn't delete it.<br>Note: You can't activate an expired schedule. |
| Run Now | Publish the dataset immediately on an ad hoc basis. |
| Suspend | Suspend use of the publish schedule. You can activate a suspended schedule. |
| Transfer Ownership | Transfer ownership of a publish schedule. Every process must have an assigned owner for the process to run.<br>You might want to transfer ownership if the assigned owner becomes inactive. The person you transfer ownership to must have the appropriate security access. |
| View All Occurrences (recurring schedules only) | View all future occurrences of a publish schedule within a specified range of dates and times. |
| View Schedule | (Recurring schedules) View schedule details, such as recurrence criteria, error messages, the schedule owner and creator, and the next 10 scheduled launches if applicable.<br>(Dependent schedules) View schedule details, such as the dependency configuration, the schedule creator and owner, and the number of times run. |

### Related Information

**Tasks**
Create Dataset Publish Schedules
**Reference**
FAQ: Dataset Publish Schedules

## 1.10.6 | Concept: Coordinating Publish Schedules with Dataset Integration

The last integration that completes before publishing begins determines the freshness of data in a Prism data source. To ensure that data is as fresh as possible, schedule enough time for an integration to complete before a dataset publishing job begins.

As you create publish schedules for datasets that have scheduled integrations, consider how these scenarios affect the freshness of data in the Prism data source:

| Scenario | Impact |
|---|---|
| Integration completes before a scheduled publish begins. | The published dataset will use the latest available data from the dataset source file. |
| Integration completes after a scheduled publish begins. | The published dataset will use the data from the dataset source file that existed before integration began. |

To help ensure that integration completes before publishing begins, you can create a publish schedule that depends on the successful integration of the data.

### Related Information

**Tasks**
Create Dataset Publish Schedules

## 1.10.7 | Example: Create Dependent Publish Schedules for Datasets

This is an example of a publish schedule with a dependency.

### Scenario

As an HR Analyst, you're responsible for creating a weekly report that includes employee hiring and turnover data from your New York office. You'd like the data in the report to be as fresh as possible. You need to:

- Bring in the latest hiring and turnover data from your New York office every Monday morning.
- Make the data available for reporting only after you've brought in the latest data.

### Prerequisites

- Create a dataset from SFTP integration. Schedule the integration to occur every Monday at 8 a.m. EST. Name the dataset *New York Hiring and Turnover Weekly*. Run the integration once to enable the publish option.
- Security: *Prism Datasets: Publish* domain in the Prism Analytics functional area.

### Steps

1. Access the View Dataset report for the *New York Hiring and Turnover Weekly* dataset.
2. From the related actions menu of the View Dataset report, select Publishing > Create Schedule.
3. In Run Frequency, select Dependent.
4. In the Dependency field, select By Type > Prism Data Acquisition Future Process > Dataset Integration Schedule: New York Hiring and Turnover Weekly.
5. In the Trigger on Status field, select Completed.
6. (Optional) You can change the publish schedule name at any time.

### Result

Workday now publishes the *New York Hiring and Turnover Weekly* dataset only after the successful integration of data from the SFTP server.

The HR Analyst can:

- View the publish schedule request in the Process Monitor and Scheduled Future Processes reports.
- View the status of all scheduled publish processes on the Activities page.

### Related Information

**Concepts**
Concept: Publishing Datasets
Concept: Dataset Publish Schedules
**Tasks**
Create Dataset Publish Schedules
Steps: Create a Dataset with External Data (SFTP Server)

## 1.10.8 | FAQ: Dataset Publish Schedules

- How do I edit a publish schedule?
- How do I edit or delete only 1 occurrence of a recurring publish schedule?
- How do I change a recurring publish schedule?
- How do I edit environment restrictions for a publish schedule?
- How do I delete a publish schedule?
- How do I expire a publish schedule?
- How do I suspend a publish schedule?
- How do I activate a suspended publish schedule?
- How do I view the details of a publish schedule?
- How do I view all occurrences of a recurring publish schedule?
- How do I immediately run a publish schedule on an ad hoc basis?
- How do I transfer ownership of a publish schedule?

**How do I edit a publish schedule?**

From the related actions menu of the View Dataset report, select Publishing > Edit Schedule.

**How do I edit or delete only 1 occurrence of a recurring publish schedule?**

1. From the related actions menu of the View Dataset report, select Publishing > View Schedule.
2. From the related actions menu of the View Scheduled Future Process report, select Schedule Future Process > Edit Scheduled Occurrence.

**How do I change a recurring publish schedule?**

1. From the related actions menu of the View Dataset report, select Publishing > View Schedule.
2. From the related actions menu of the View Scheduled Future Process report, select Schedule Future Process > Change Schedule.

**How do I edit environment restrictions for a publish schedule?**

1. From the related actions menu of the View Dataset report, select Publishing > View Schedule.
2. From the related actions menu of the View Scheduled Future Process report, select Schedule Future Process > Edit Environment Restrictions.

**How do I delete a publish schedule?**

1. From the related actions menu of the View Dataset report, select Publishing > View Schedule.
2. From the related actions menu of the View Scheduled Future Process report, select Schedule Future Process > Delete.

**How do I expire a publish schedule?**

From the related actions menu of the View Dataset report, select Publishing > Expire Schedule.

**How do I suspend a publish schedule?**

1. From the related actions menu of the View Dataset report, select Publishing > View Schedule.
2. From the related actions menu of the View Scheduled Future Process report, select Schedule Future Process > Suspend.

**How do I activate a suspended publish schedule?**

1. From the related actions menu of the View Dataset report, select Publishing > View Schedule.
2. From the related actions menu of the View Scheduled Future Process report, select Schedule Future Process > Activate.
   Note: You can only activate expired schedules.

**How do I view the details of a publish schedule?**

From the related actions menu of the View Dataset report, select Publishing > View Schedule.

**How do I view all occurrences of a recurring publish schedule?**

1. From the related actions menu of the View Dataset report, select Publishing > View Schedule.
2. From the related actions menu of the View Scheduled Future Process report, select Schedule Future Process > View All Occurrences.

**How do I immediately run a publish schedule on an ad hoc basis?**

1. From the related actions menu of the View Dataset report, select Publishing > View Schedule.
2. From the related actions menu of the View Scheduled Future Process report, select Schedule Future Process > Run Now.

**How do I transfer ownership of a publish schedule?**

1. From the related actions menu of the View Dataset report, select Publishing > View Schedule.
2. From the related actions menu of the View Scheduled Future Process report, select Schedule Future Process > Transfer Ownership.

### Related Information

**Concepts**
Concept: Publishing Datasets
Concept: Dataset Publish Schedules
**Tasks**
Create Dataset Publish Schedules
Manage Scheduled Future Processes

## 1.11 | Deleting Prism Analytics Data

### 1.11.1 | Truncate Data in a Table

### Prerequisites

Any of these security requirements:

- *Prism: Tables Manage* domain in the Prism Analytics functional area.
- *Prism: Tables Owner Manage* domain in the Prism Analytics functional area.
- *Table Editor* permission on the table.
- *Table Owner* permission on the table.
- *Can Truncate Table Data* permission on the table.

### Context

You can remove all data in a table dataset by truncating the table. Truncating a table removes the data, but retains the schema.

You might want to truncate a table if:

- The table integration uses Append mode and after several integrations the dataset contains some bad data.
- The table contains data that your organization later deems to be sensitive.

If you've previously published a derived dataset based on this table, the associated Prism data source still contains data. If you also want to remove the data from the associated Prism data source, then you must publish the derived dataset again. Publishing a derived dataset from a truncated table makes the associated Prism data source empty, but active.

### Steps

1. Access the View Table report for the table you want to truncate.
2. Select Actions > Table > Truncate.

### 1.11.2 | Delete Data from a Table

### Prerequisites

Any of these security requirements:

- *Prism: Tables Manage* domain in the Prism Analytics functional area.
- *Prism: Tables Owner Manage* domain in the Prism Analytics functional area.
- *Table Editor* permission on the table.
- *Table Owner* permission on the table.
- *Can Delete Table Data* permission on the table.
- *Can Truncate Table Data* permission on the table.

### Context

You can delete all rows from a table that came from a particular data load activity.

### Steps

1. Access the View Table report.

2. Select Actions > Table > Delete Rows.
3. Select how to specify the rows to delete.

| Option | Description |
|---|---|
| **Previous Loads** | Select this option to delete all rows that were previously added in a particular load. |

4. To delete rows by a particular load, click the load you want to delete.

### Related Information

**Tasks**

Change Rows in a Table by Uploading a File

## 1.11.3 | Truncate Data in a Dataset

### Prerequisites

Any of these security requirements:

- *Prism Datasets: Manage* domain in the Prism Analytics functional area.
- *Dataset Editor* permission on the dataset.
- *Dataset Owner* permission on the dataset.

### Context

You can remove the data in a base dataset by truncating the dataset. Truncating a dataset removes the data from the dataset, but retains its metadata, such as schema and transformations.

You might want to truncate a dataset if:

- The dataset integration uses Append mode and after several integrations the dataset contains some bad data.
- The dataset contains data that your organization later deems to be sensitive.

If you've previously published this dataset or a derived dataset based on this dataset, the associated Prism data source still contains data. If you also want to remove the data from the associated Prism data source, then you must publish the dataset again. Publishing a truncated dataset makes the associated Prism data source empty, but active.

### Steps

1. Access the View Dataset report for the dataset you want to truncate.
2. Select Actions > Dataset > Truncate.

### Related Information

**Concepts**

Concept: Deleting Prism Data

## 1.11.4 | Unpublish a Dataset

### Prerequisites

- Prism data source exists in Workday, but no reports use it.
- Security: *Prism Datasets: Publish* domain in the Prism Analytics functional area.

### Context

After you publish a dataset, you can unpublish it if necessary. When you unpublish a dataset, Workday removes the Prism data source that is based on the dataset, including all data in it. You might want to unpublish a dataset if you need to delete the dataset. You can only unpublish a dataset if no reports use the associated Prism data source.

### Steps

1. Access the View Dataset page for the dataset you want to unpublish.
2. Select Actions > Publishing > Unpublish Dataset.

### Result

Workday removes the Prism data source and its rows.

### Related Information

**Concepts**

Concept: Deleting Prism Data

## 1.11.5 | Delete Rows from a Prism Data Source

### Prerequisites

Security: *Prism Datasets: Publish* domain in the Prism Analytics functional area.

### Context

When you publish a dataset, Workday creates a Prism data source and populates it with rows containing the transformed data from the dataset. You can delete the rows in a Prism data source. When you delete rows in a Prism data source, the Prism data source is empty and becomes inactive.

You might want to delete the rows in a Prism data source if the rows contain incorrect data and you don't want analysts creating reports using the bad data. You can then edit the dataset to correct the transformation logic and publish the dataset again. When you republish the dataset, Workday populates the empty, inactive Prism data source with the new data.

Any reports that use an inactive Prism data source will be broken until you publish the dataset again.

### Steps

1. Access the View Prism Data Source page, and select the Prism data source whose rows you want to delete.
2. Select Actions > Prism Data Source > Delete Published Rows.

### Result

Workday removes all data from the Prism data source, and changes the Prism data source status to inactive.

### 1.11.6 | Concept: Deleting Prism Data

You create datasets and publish them to create Prism data sources filled with data. However, there might be times when you need to remove data from your tenant.

You can remove data and Prism objects:

- **Remove the rows from a Prism data source.** When you delete the rows in a Prism data source, the Prism data source is empty (all rows are deleted) and becomes inactive. You might want to delete the rows in a Prism data source if the rows contain incorrect data and you don't want analysts creating reports using the bad data. You can remove rows from a Prism data source whether or not any reports or visualizations use it.
- **Unpublish a dataset.** When you unpublish a dataset, Workday deletes the Prism data source from your tenant, including all data in it. You might want to unpublish a dataset to delete it from the Data Catalog. You can only unpublish a dataset when no reports currently use the associated Prism data source.
- **Remove the data from a dataset (truncate).** You can remove the data in a base dataset by truncating the dataset. Truncating a dataset removes all data from the dataset. However, the dataset retains its metadata, such as schema and transformations. You might want to truncate a dataset if the dataset integration uses Append mode and after several integrations the dataset contains some bad data. If you've previously published this dataset or a derived dataset based on this dataset, the associated Prism data source still contains data. If you also want to remove the data from the associated Prism data source, then you must publish the dataset again. Publishing a truncated dataset makes the associated Prism data source empty, but active.
- **Delete a dataset.** When you delete a dataset, Workday removes the dataset definition from the Data Catalog and removes the source data stored on disk in your tenant. You might want to delete a dataset to make more space available to store data in your Data Catalog. You can only delete a dataset if it's not currently published and isn't imported into any derived dataset. Right-click a dataset from the Data Catalog and select Delete.

### Related Information

**Tasks**

Truncate Data in a Dataset
Unpublish a Dataset
Delete Rows from a Prism Data Source

## 1.12 | Reference: Prism Expression Language

### 1.12.1 | Concept: Prism Expression Language

Prism Analytics includes its own expression language that's composed of functions and operators. You use the Prism expression language to create an *expression*, which you can use to create new values or filter existing values.

An expression computes or produces a value by combining fields, constant values, operators, and functions. An expression outputs a value of a particular field type, such as Numeric or Text values. Simple expressions can be a single constant value, the values of a given field, or just a function. You can use operators to join two or more simple expressions into a complex expression.

You can use expressions in datasets:

- **Calculated fields.** Use expressions to define calculated fields that operate on the source data. A calculated field expression generates its values based on a calculation or condition, and returns a value for each input row. Calculated field expressions can contain values from other fields, constants, mathematical operators, comparison operators, or built-in row functions.
- **Filter stages.** Use an expression in a Filter stage to limit the scope of the source data of a dataset.

The expression builder helps you create calculated field expressions in a dataset. It displays the available fields in the dataset, plus the list of the Prism functions. It validates your expressions for correct syntax, input field types, and so on.

### Function Inputs and Outputs

Functions take one or more input values and return an output value. Input values can be a literal value or the name of a field that contains a value. In both cases, the function expects the input value to be a particular field type such as Text or Integer. Example: The CONCAT() function combines Text inputs and outputs a new Text.

This example demonstrates how to use the CONCAT() function to concatenate the values in the *month*, *day*, and *year* fields separated by the literal forward slash character:

```
CONCAT(month,"/",day,"/",year)
```

A function return value might be the same as its input type or it might be an entirely new field type. Example: The TO_DATE() function takes a Text as input, but outputs a Date value. If a function expects a Text, but is passed another field type as input, the function returns an error.

Typically, functions are classified by what field type they take or what purpose they serve. Example: CONCAT() is a text function and TO_DATE() is a field type conversion function.

## Nesting Functions

Functions can take other functions as arguments. Example: You can use the CONCAT function as an argument to the TO_DATE() function. The final result is a Date value in the format 10/31/2014.

```
TO_DATE(CONCAT(month,"/",day,"/",year),"MM/dd/yyyy")
```

The nested function must return the correct field type. So, because TO_DATE() expects text input and CONCAT() returns a text, the nesting succeeds.

## Referring to Fields in the Current Dataset

When specifying a field name in an expression, you can simply refer to the field by its name if it doesn't contain spaces or special characters. Example: This expression converts the values of the *sales* field from a Numeric field type to an Integer field type:

```
TO_INT(sales)
```

Enclose field names with square brackets ([ ]) if they contain spaces, special characters, reserved keywords (such as function names), or start with numeric characters. Example:

```
TO_INT([Sale Amount])
```

```
TO_INT([2013_data])
```

```
TO_INT([count])
```

If a field name contains a ] (closing square bracket), you must escape the closing square bracket by doubling it ]]. Suppose you have this field name:

```
[Total Sales]
```

You enclose the entire field name in square brackets and escape the closing bracket that is part of the actual field name:

```
[[Total Sales]]]
```

## Literal Text Values

To specify a literal or actual Text value, enclose the value in double quotes ( "). Example: This expression converts the values of a *gender* field to the literal values of *male*, *female*, or *unknown*:

```
CASE WHEN gender="M" THEN "male"
WHEN gender="F" THEN "female"
ELSE "unknown" END
```

To escape a literal quote within a literal value itself, double the literal quote character. Example:

```
CASE WHEN height="60"" THEN "5 feet" WHEN height="72""
THEN "6 feet" ELSE "other" END
```

The REGEX() function is a special case. In the REGEX() function, Text expressions are also enclosed in quotes. When a Text expression contains literal quotes, double the literal quote character. Example:

```
REGEX(height, "\d\'(\d)+"")
```

## Literal Date Values

To refer to a Date value in a Filter stage expression, you must use this format (or any shortened version of it) without any enclosing quotation marks or other punctuation:

*yyyy-MM-ddTHH:mm:ss:SSSZ*

Example:

```
[Order Date] BETWEEN 2016-06-01T00:00:00.000Z AND 2016-07-31T00:00:00.000Z
```

If the Filter expression is a shortened version of the full format, then Prism assigns any values that aren't included a value of zero (0). Example: This expression:

```
[Order Date] >= 2017-01-01
```

Is equivalent to:

```
[Order Date] >= 2017-01-01T00:00:00.000Z
```

If the date value is in Text format rather than a Date format, you must enclose the value in quotes.

To refer to a literal date value in a calculated field expression, you must specify the format of the date and time components using TO_DATE, which takes a Text literal argument and a format string. Example:

```
CASE WHEN [Order Date]=TO_DATE("2013-01-01 00:00:59 PST","yyyy-MM-dd HH:mm:ss z")
THEN "free shipping" ELSE "standard shipping" DONE
```

## Literal Numeric Values

For literal numeric values, you can just specify the number itself without any special escaping or formatting. Example:

```
CASE WHEN [is married]=1 THEN "married" WHEN [is married]=0
THEN "not_married" ELSE NULL END
```

### Related Information

**Tasks**

Add a Prism Calculated Field to a Dataset

## 1.12.2 | Comparison Operators

Comparison operators are used to compare the equivalency or inequivalency of 2 expressions of the same field type. The result of a comparison expression is a Boolean value (returns true, false, or NULL for invalid, such as comparing a text value to a numeric value). Boolean expressions are most often used to specify data processing conditions or filter criteria.

Example: You can use comparison operators in a `CASE` expression:

```
CASE WHEN age <= 25 THEN "0-25"
WHEN age <= 50 THEN "26-50"
ELSE "over 50" END
```

This expression compares the value in the age field to a literal number value. If true, it returns the appropriate Text value.

| Operator | Meaning | Example |
|---|---|---|
| = or == | Equal to | `order_date = "12/22/2016"` |
| > | Greater than | `age > 18` |
| !> | Not greater than (equivalent to <) | `age !> 8` |
| < | Less than | `age < 30` |
| !< | Not less than (equivalent to >=) | `age !< 12` |
| >= | Greater than or equal to | `age >= 20` |
| <= | Less than or equal to | `age <= 29` |
| <> or != or ^= | Not equal to | `age <> 30` |
| `BETWEEN` *min_value* `AND` *max_value* | Test whether a date or numeric value is within the *min* and *max* values (inclusive). | `year BETWEEN 2014 AND 2016` |
| `IN(`*list*`)` | Test whether a value is within a set. | `product_type IN("tablet","phone","laptop")` |
| `LIKE("`*pattern*`")` | Simple inclusive case-insensitive character pattern matching. The * character matches any number of characters. The ? character matches exactly 1 (a single) character. | `last_name LIKE("?utch*")`<br>Matches *Kutcher*, *hutch* but not *Krutcher* or *crutch*<br>`company_name LIKE("workday")`<br>Matches *Workday* or *workday* |
| *value* `IS NULL` | Check whether a field value or expression is null (empty). | `ship_date IS NULL`<br>Evaluates to *true* when the *ship_date* field is empty |

## 1.12.3 | Logical Operators

Use logical operators in expressions to test for a condition. Logical operators define Boolean expressions.

You might want to use logical operators in Filter transformations or CASE expressions. Filters test if a field or value meets some condition, such as testing if the value in a Date field falls between 2 other dates:

```
BETWEEN 2016-06-01 AND 2016-07-31
```

| Operator | Meaning | Example |
|---|---|---|
| AND | Test whether 2 conditions are true. | |
| OR | Test if either of 2 conditions are true. | |

| Operator | Meaning | Example |
|---|---|---|
| `NOT` | Reverses the value of other operators. | • `year NOT BETWEEN 2013 AND 2016`<br>• `first_name NOT LIKE("Jo?n*")`<br>Excludes *John*, *jonny* but not *Jon* or *Joann*<br>• `Weekday NOT IN("Saturday","Sunday")`<br>• `purchase_date IS NOT NULL`<br>Evaluates to *true* when the *purchase_date* field is not empty |

#### 1.12.4 | Arithmetic Operators

Arithmetic operators perform basic math operations on 2 expressions of the same field type resulting in a numeric value.

Example: You can calculate the *gross profit margin percentage* using the values of a *total_revenue* and *total_cost* field:

```
((total_revenue - total_cost) / total_cost) * 100
```

Example: You can use the plus (+) operator to combine `Text` values:

```
"Firstname" + " " + "Lastname"
```

You can use the plus (+) and minus (-) operators to add or subtract `Date` values.

| Operator | Meaning | Example |
|---|---|---|
| `+` | Addition | `amount + 10`<br>Add 10 to the value of the *amount* field. |
| `-` | Subtraction | `amount - 10`<br>Subtract 10 from the value of the *amount* field. |
| `*` | Multiplication | `amount * 100`<br>Multiply the value of the *amount* field by 100. |
| `/` | Division | `bytes / 1024`<br>Divide the value of the *bytes* field by 1024 and return the quotient. |

#### 1.12.5 | Conversion Functions

##### 1.12.5.1 | BUILD_CURRENCY

## Description

`BUILD_CURRENCY` is a row function that constructs a Currency field from a numeric value and a Text or Instance value that contains a valid currency code. When the currency code isn't valid, this function returns NULL.

## Syntax

```
BUILD_CURRENCY(number_expression,currency_code_expression)
```

## Return Value

Returns a value of type `Currency`.

## Input Parameters

**number_expression**
Required. A field or expression of type `Double`, `Numeric`, `Integer`, or `Long`.

**currency_code_expression**
Required. A field or expression of type `Text` or `Instance` that contains valid currency code data.

## Examples

Convert the values of the Sale Price field (Numeric type) to a Currency field type using the currency codes from the Currency Code field:

```
BUILD_CURRENCY([Sale Price], [Currency Code])
```

#### 1.12.5.2 | CAST

## Description

CAST is a row function that converts data values from one field type (data type) to another.

You can use CAST to convert these field types:

| From Field Type | To Field Types |
|---|---|
| Boolean | Boolean, Numeric, Double, Integer, Long, Text |
| Date | Date, Text |
| Numeric | Boolean, Numeric, Double, Integer, Long, Text |
| Double | Boolean, Numeric, Double, Integer, Long, Text |
| Integer | Boolean, Numeric, Double, Integer, Long, Text |
| Long | Boolean, Numeric, Double, Integer, Long, Text |
| Instance | Instance, Text |
| Multi-Instance | Multi-Instance (using a different business object) |
| Text | Boolean, Numeric, Double, Instance, Integer, Long, Text |

## Syntax

```
CAST(field_name AS field_type)
```

## Return Value

Returns one value per row of the specified field type.

## Input Parameters

**field_name**
Required. A field or expression of a supported field type.

**field_type**
Required. The field type to convert the data values into.

When specifying Boolean as the field type, CAST converts the value of zero (0) to False, and all other values to True.

When specifying Instance or Multi-Instance as the field type, you must specify the business object using its unique identifier (WID). Use this syntax:

```
Instance(business_object_WID)
```

```
Multi_Instance(business_object_WID)
```

When specifying Numeric as the field type, specify the number of digits to the left of the decimal point (integers) and the number of digits to the right of decimal point (decimals). Use this syntax:

```
Decimal(integers,decimals)
```

Ensure that the number of integer digits specified is large enough to capture all possible data values. If the value for a row has more integer digits than the number of integer digits specified in the function, then CAST returns NULL. Example: `CAST(99.9 AS decimal(1,1))` returns NULL.

## Examples

Convert the values of the *Region* field from Text to Instance:

```
CAST([Region] AS Instance(eecb565181284b6a8ae8b45dc3ed1451))
```

Convert the values of the *average_rating* field to a Numeric field type:

```
CAST([average_rating] AS decimal(1,2))
```

`CAST(99.99 AS decimal(10,3))` returns `99.990`.

`CAST(99.99 AS decimal(20,2))` returns `99.99`.

`CAST(99.999 AS decimal(10,2))` returns `100.00`.

`CAST(99.99 AS decimal(1,2))` returns `NULL`.

#### 1.12.5.3 | EPOCH_MS_TO_DATE

## Description

`EPOCH_MS_TO_DATE` is a row function that converts `Long` values to `Date` values, where the input number represents the number of milliseconds since the epoch.

## Syntax

```
EPOCH_MS_TO_DATE(long_expression)
```

## Return Value

Returns one value per row of type `Date` in UTC format *yyyy-MM-dd HH:mm:ss:SSS Z*.

## Input Parameters

**long_expression**
Required. A field or expression of type `Long` representing the number of milliseconds since the epoch date (January 1, 1970 00:00:00:000 GMT).

## Examples

Convert a number representing the number of milliseconds from the epoch to a human-readable date and time:

`EPOCH_MS_TO_DATE(1360260240000)` returns *`2013-02-07T18:04:00:000Z`* or February 7, 2013 18:04:00:000 GMT

Or if your data is in seconds instead of milliseconds:

`EPOCH_MS_TO_DATE(1360260240 * 1000)` returns *`2013-02-07T18:04:00:000Z`* or February 7, 2013 18:04:00:000 GMT

#### 1.12.5.4 | EXTRACT_AMOUNT

## Description

`EXTRACT_AMOUNT` is a row function that takes a Currency value and extracts the numeric amount as a Numeric value.

## Syntax

```
EXTRACT_AMOUNT(currency_expression)
```

## Return Value

Returns a value of type `Numeric`.

## Input Parameters

**currency_expression**
Required. A field or expression of type `Currency`.

## Examples

Get the numeric values from the Salary field (Currency field type):

```
EXTRACT_AMOUNT([Salary])
```

#### 1.12.5.5 | EXTRACT_CODE

## Description

`EXTRACT_CODE` is a row function that takes a Currency value and extracts the currency code as an Instance value.

## Syntax

```
EXTRACT_CODE(currency_expression)
```

## Return Value

Returns a value of type `Instance`.

## Input Parameters

**currency_expression**
Required. A field or expression of type `Currency`.

## Examples

Get the currency code information from the Salary field (Currency field type) as an Instance field:

```
EXTRACT_CODE([Salary])
```

#### 1.12.5.6 | EXTRACT_CODE_TEXT

## Description

`EXTRACT_CODE_TEXT` is a row function that takes a Currency value and extracts the currency code as a Text value.

## Syntax

```
EXTRACT_CODE_TEXT(currency_expression)
```

## Return Value

Returns a value of type `Text`.

## Input Parameters

**currency_expression**
Required. A field or expression of type `Currency`.

## Examples

Get the currency code information from the Salary field (Currency field type) as a Text field:

```
EXTRACT_CODE_TEXT([Salary])
```

#### 1.12.5.7 | TO_BOOLEAN

## Description

`TO_BOOLEAN` is a row function that converts `Text`, `Boolean`, `Integer`, `Long`, or `Numeric` values to `Boolean`.

## Syntax

```
TO_BOOLEAN(expression)
```

## Return Value

Returns one value per row of type `Boolean`.

## Input Parameters

**expression**
Required. A field or expression of type `Text`, `Boolean`, `Integer`, `Long`, or `Numeric`.

The function converts these values to true:

```
1, 1.0, "true", "t", "yes", "y", "1"
```

The function converts these values to false:

```
0, 0.0, "false", "f", "no", "n", "0"
```

The function converts all other values to NULL.

## Examples

Convert the values of the *is_contingent* field to a Boolean:

```
TO_BOOLEAN(is_contingent)
```

These expressions return true:

```
TO_BOOLEAN("TRUE")
```

```
TO_BOOLEAN("1")
```

```
TO_BOOLEAN(1.0)
```

These expressions return false:

```
TO_BOOLEAN("False")
```

```
TO_BOOLEAN("0")
```

```
TO_BOOLEAN(0.0)
```

These expressions return NULL:

```
TO_BOOLEAN("correct")
```

```
TO_BOOLEAN("1.0")
```

```
TO_BOOLEAN(1.1)
```

#### 1.12.5.8 | TO_CURRENCY

## Description

`TO_CURRENCY` is a row function that converts `Text` values that contain valid currency-formatted data to `Currency` values.

## Syntax

```
TO_CURRENCY(expression)
```

## Return Value

Returns a value of type `Currency`.

## Input Parameters

**expression**
Required. A field or expression of type `Text` that represents a valid currency-formatted value.
A valid currency-formatted value meets these requirements:

- Includes both the numeric value and currency code.
- Uses a period to separate digits before and after the decimal.
- Doesn't include any character to separate the thousands place.

Example: `10000 EUR` and `12345.678 USD`.

## Examples

Convert this text value to a Currency field type:

```
TO_CURRENCY("1234.56 USD")
```

Convert the values of the Grant Price field to a Currency field type using the currency codes in the Grant Code field:

```
TO_CURRENCY(CONCAT(TO_STRING([Grant Price]), [Grant Code]))
```

Convert the Sale Price field Text field to a Currency field, but first transform the occurrence of any N/A values to NULL values using a `CASE` expression:

```
TO_CURRENCY(CASE WHEN [Sale Price]="N/A" then NULL ELSE [Sale Price] END)
```

#### 1.12.5.9 | TO_DATE

## Description

`TO_DATE` is a row function that converts `Text` values to `Date` values, and specifies the format of the date and time elements in the string.

## Syntax

```
TO_DATE(string_expression,"date_format")
```

## Return Value

Returns one value per row of type `Date` (which by definition is in UTC).

## Input Parameters

**string_expression**
Required. A field or expression of type `Text`.

**date_format**
Required. A pattern that describes how the date is formatted.

## Examples

Define a new `Date` Prism calculated field based on the *order_date* base field, which contains timestamps in the format of: *2014.07.10 at 15:08:56 PDT*:

```
TO_DATE(order_date,"yyyy.MM.dd 'at' HH:mm:ss z")
```

Define a new `Date` Prism calculated field by first combining individual *month*, *day*, *year*, and *depart_time* fields (using `CONCAT`), and performing a transformation on *depart_time* to make sure three-digit times are converted to four-digit times (using `REGEX_REPLACE`):

```
TO_DATE(CONCAT(month,"/",day,"/",year,":",
REGEX_REPLACE(depart_time,"\b(\d{3})\b","0$1")),"MM/dd/yyyy:HHmm")
```

Define a new `Date` Prism calculated field based on the *created_at* base field, which contains timestamps in the format of: *Sat Jan 25 16:35:23 +0800 2014* (this is the timestamp format returned by Twitter's API):

```
TO_DATE(created_at,"EEE MMM dd HH:mm:ss Z yyyy")
```

### Related Information

**Reference**
Reference: Date Format Symbols

#### 1.12.5.10 | TO_DECIMAL

## Description

`TO_DECIMAL` is a row function that converts `Text`, `Boolean`, `Integer`, `Long`, `Double`, or `Numeric` values to `Numeric` values with the default number of digits before and after the decimal point.

## Syntax

```
TO_DECIMAL(expression)
```

## Return Value

Returns one value per row of type `Numeric` with the default number of digits before and after the decimal point.

## Input Parameters

**expression**
Required. A field or expression of type `Text` (must be numeric characters), `Boolean`, `Integer`, `Long`, `Double`, or `Numeric`.

## Examples

Convert the values of the *average_rating* field to a Numeric field type:

```
TO_DECIMAL(average_rating)
```

Convert the *average_rating* field to a Numeric field type, but first transform the occurrence of any *NA* values to *NULL* values using a `CASE` expression:

```
TO_DECIMAL(CASE WHEN average_rating="N/A" then NULL ELSE average_rating END)
```

#### 1.12.5.11 | TO_DOUBLE

## Description

`TO_DOUBLE` is a row function that converts `Text`, `Boolean`, `Integer`, `Long`, or `Double` values to `Double` (a type of numeric) values.

## Syntax

```
TO_DOUBLE(expression)
```

## Return Value

Returns one value per row of type `Double`.

## Input Parameters

**expression**
Required. A field or expression of type `Text` (must be numeric characters), `Boolean`, `Integer`, `Long`, or `Double`.

## Examples

Convert the values of the *average_rating* field to a Double field type:

```
TO_DOUBLE(average_rating)
```

Convert the *average_rating* field to a Double field type, but first transform the occurrence of any *NA* values to *NULL* values using a `CASE` expression:

```
TO_DOUBLE(CASE WHEN average_rating="N/A" then NULL ELSE average_rating END)
```

#### 1.12.5.12 | TO_INT

## Description

`TO_INT` is a row function that converts `Text`, `Boolean`, `Integer`, `Long`, or `Double` values to `Integer` (whole number) values. When converting `Double` values, everything after the decimal will be truncated (not rounded up or down).

## Syntax

```
TO_INT(expression)
```

## Return Value

Returns one value per row of type `Integer`.

## Input Parameters

**expression**
Required. A field or expression of type `Text`, `Boolean`, `Integer`, `Long`, or `Double`. If a `Text` field contains non-numeric characters, the function returns NULL.

## Examples

Convert the values of the *average_rating* field to an Integer field type:

```
TO_INT(average_rating)
```

Convert the *flight_duration* field to an Integer field type, but first transform the occurrence of any *NA* values to NULL values using a `CASE` expression:

```
TO_INT(CASE WHEN flight_duration="N/A" then NULL ELSE flight_duration END)
```

#### 1.12.5.13 | TO_LONG

## Description

`TO_LONG` is a row function that converts `Text`, `Boolean`, `Integer`, `Long`, `Decimal`, `Date`, or `Double` values to `Long` (whole number) values. When converting `Decimal` or `Double` values, everything after the decimal will be truncated (not rounded up or down).

## Syntax

```
TO_LONG(expression)
```

## Return Value

Returns one value per row of type `Long`.

## Input Parameters

**expression**
Required. A field or expression of type `Text` (must be numeric characters only, no period or comma), `Boolean`, `Integer`, `Long`, `Decimal`, `Date`, or `Double`. When a `Text` field value includes a decimal, the function returns a NULL value.

## Examples

Convert the values of the *average_rating* field to a Long field type:

```
TO_LONG(average_rating)
```

Convert the *average_rating* field to a Long field type, but first transform the occurrence of any *NA* values to *NULL* values using a `CASE` expression:

```
TO_LONG(CASE WHEN average_rating="N/A" then NULL ELSE average_rating END)
```

#### 1.12.5.14 | TO_STRING

## Description

`TO_STRING` is a row function that converts values of other data types to `Text` (character) values.

## Syntax

```
TO_STRING(expression)
```

```
TO_STRING(date_expression,date_format)
```

## Return Value

Returns one value per row of type `Text`.

## Input Parameters

**expression**
A field or expression of type `Text`, `Boolean`, `Integer`, `Long`, `Numeric`, `Double`, `Instance`, or `Multi-Instance`. When you convert an Instance or Multi-Instance field to a string, this function returns the unique identifier (WID), not the display name, of the field value. When a Multi-Instance field contains more than 1 value, this function concatenates each value into a single string with no spaces.

**date_expression**
A field or expression of type `Date`.

**date_format**
If converting a `Date` to `Date`, a pattern that describes how the date is formatted. See TO_DATE for the date format patterns.

## Examples

Convert the values of the *sku_number* field to a Text field type:

```
TO_STRING(sku_number)
```

Convert values in the *age* column into range-based groupings (binning), and cast output values to a `Text`:

```
TO_STRING(CASE WHEN age <= 25 THEN "0-25" WHEN age <= 50 THEN "26-50" ELSE "over 50" END)
```

Convert the values of a *timestamp* Date field to `Text`, where the timestamp values are in the format of: *2002.07.10 at 15:08:56 PDT*:

```
TO_STRING(timestamp,"yyyy.MM.dd 'at' HH:mm:ss z")
```

### 1.12.6 | Date Functions

#### 1.12.6.1 | DAYS_BETWEEN

## Description

`DAYS_BETWEEN` is a row function that calculates the whole number of days (ignoring time) between two `date` values (`value1` - `value2`).

## Syntax

```
DAYS_BETWEEN(date_1,date_2)
```

## Return Value

Returns one value per row of type `Integer`.

## Input Parameters

**date_1**
Required. A field or expression of type `Date`.

**date_2**
Required. A field or expression of type `Date`.

## Examples

Calculate the number of days to ship a product by subtracting the value of the *order_date* field from the *ship_date* field:

```
DAYS_BETWEEN(ship_date,order_date)
```

Calculate the number of days since a product's release by subtracting the value of the *release_date* field from the current date (the result of the `TODAY` expression):

```
DAYS_BETWEEN(TODAY(),release_date)
```

#### 1.12.6.2 | DATE_ADD

## Description

`DATE_ADD` is a row function that adds the specified time interval to a `date` value.

## Syntax

```
DATE_ADD(date,quantity,"interval")
```

## Return Value

Returns a value of type `Date`.

## Input Parameters

**date**
Required. A field name or expression that returns a `date` value.

**quantity**
Required. An integer value. To add time intervals, use a positive integer. To subtract time intervals, use a negative integer.

**interval**
Required. One of the following time intervals:

- **`millisecond`** - Adds the specified number of milliseconds to a date value.
- **`second`** - Adds the specified number of seconds to a date value.
- **`minute`** - Adds the specified number of minutes to a date value.
- **`hour`** - Adds the specified number of hours to a date value.
- **`day`** - Adds the specified number of days to a date value.
- **`week`** - Adds the specified number of weeks to a date value.
- **`month`** - Adds the specified number of months to a date value.
- **`quarter`** - Adds the specified number of quarters to a date value.
- **`year`** - Adds the specified number of years to a date value.
- **`weekyear`** - Adds the specified number of weekyears to a date value.

## Examples

Add 45 days to the value of the *invoice_date* field to calculate the date a payment is due:

```
DATE_ADD(invoice_date,45,"day")
```

#### 1.12.6.3 | EXTRACT

## Description

`EXTRACT` is a row function that returns the specified portion of a `date` value.

## Syntax

```
EXTRACT("extract_value",date)
```

## Return Value

Returns the specified extracted value as type `Integer`. `EXTRACT` removes leading zeros. For example, the month of April returns a value of `4`, not `04`.

## Input Parameters

**extract_value**
Required. One of the following extract values:

- **`millisecond`** - Returns the millisecond portion of a date value. For example, an input date value of `2012-08-15 20:38:40.213` would return an integer value of `213`.
- **`second`** - Returns the second portion of a date value. For example, an input date value of `2012-08-15 20:38:40.213` would return an integer value of `40`.
- **`minute`** - Returns the minute portion of a date value. For example, an input date value of `2012-08-15 20:38:40.213` would return an integer value of `38`.
- **`hour`** - Returns the hour portion of a date value. For example, an input date value of `2012-08-15 20:38:40.213` would return an integer value of `20`.
- **`day`** - Returns the day portion of a date value. For example, an input date value of `2012-08-15` would return an integer value of **`15`**.
- **`week`** - Returns the ISO week number for the input date value. For example, an input date value of `2012-01-02` would return an integer value of `1` (the first ISO week of 2012 starts on Monday January 2). An input date value of `2012-01-01` would return an integer value of `52` (January 1, 2012 is part of the last ISO week of 2011).
- **`month`** - Returns the month portion of a date value. For example, an input date value of `2012-08-15` would return an integer value of `8`.
- **`quarter`** - Returns the quarter number for the input date value, where quarters start on January 1, April 1, July 1, or October 1. For example, an input date value of `2012-08-15` would return a integer value of `3`.
- **`year`** - Returns the year portion of a date value. For example, an input date value of `2012-01-01` would return an integer value of `2012`.
- **`weekyear`** - Returns the year value that corresponds to the ISO week number of the input date value. For example, an input date value of `2012-01-02` would return an integer value of `2012` (the first ISO week of 2012 starts on Monday January 2). An input date value of `2012-01-01` would return an integer value of `2011` (January 1, 2012 is part of the last ISO week of 2011).

**date**
Required. A field name or expression that returns a `date` value.

## Examples

Extract the hour portion from the *order_date* Date field:

```
EXTRACT("hour",order_date)
```

Cast the value of the *order_date* `Text` field to a date value using `TO_DATE`, and extract the ISO week year:

```
EXTRACT("weekyear",TO_DATE(order_date,"MM/dd/yyyy HH:mm:ss"))
```

#### 1.12.6.4 | HOURS_BETWEEN

## Description

`HOURS_BETWEEN` is a row function that calculates the whole number of hours (ignoring minutes, seconds, and milliseconds) between two `date` values (`value1` - `value2`).

## Syntax

```
HOURS_BETWEEN(date_1,date_2)
```

## Return Value

Returns one value per row of type `Integer`.

## Input Parameters

**date_1**
Required. A field or expression of type `Date`.

**date_2**
Required. A field or expression of type `Date`.

## Examples

Calculate the number of hours to ship a product by subtracting the value of the *ship_date* field from the *order_date* field:

```
HOURS_BETWEEN(ship_date,order_date)
```

#### 1.12.6.5 | MILLISECONDS_BETWEEN

## Description

`MILLISECONDS_BETWEEN` is a row function that calculates the whole number of milliseconds between two `date` values (`value1` - `value2`).

## Syntax

```
MILLISECONDS_BETWEEN(date_1,date_2)
```

## Return Value

Returns one value per row of type `Integer`.

## Input Parameters

**date_1**
Required. A field or expression of type `Date`.

**date_2**
Required. A field or expression of type `Date`.

## Examples

Calculate the number of milliseconds it took to serve a web page by subtracting the value of the *request_timestamp* field from the *response_timestamp* field:

```
MILLISECONDS_BETWEEN(request_timestamp,response_timestamp)
```

#### 1.12.6.6 | MINUTES_BETWEEN

## Description

`MINUTES_BETWEEN` is a row function that calculates the whole number of minutes (ignoring seconds and milliseconds) between two `date` values (`value1` - `value2`).

## Syntax

```
MINUTES_BETWEEN(date_1,date_2)
```

## Return Value

Returns one value per row of type `Integer`.

## Input Parameters

**date_1**
Required. A field or expression of type `Date`.

**date_2**
Required. A field or expression of type `Date`.

## Examples

Calculate the number of minutes it took for a user to click on an advertisement by subtracting the value of the *impression_timestamp* field from the *conversion_timestamp* field:

```
MINUTES_BETWEEN(impression_timestamp,conversion_timestamp)
```

#### 1.12.6.7 | SECONDS_BETWEEN

## Description

`SECONDS_BETWEEN` is a row function that calculates the whole number of seconds (ignoring milliseconds) between two `date` values (`value1` - `value2`).

## Syntax

```
SECONDS_BETWEEN(date_1,date_2)
```

## Return Value

Returns one value per row of type `Integer`.

## Input Parameters

**date_1**
Required. A field or expression of type `Date`.

**date_2**
Required. A field or expression of type `Date`.

## Examples

Calculate the number of seconds it took for a user to click on an advertisement by subtracting the value of the *impression_timestamp* field from the *conversion_timestamp* field:

```
SECONDS_BETWEEN(impression_timestamp,conversion_timestamp)
```

#### 1.12.6.8 | TODAY

## Description

`TODAY` is a scalar function that returns the current system date and time as a `date` value (no time information). It can be used in other expressions involving `Date` type fields, such as `YEAR_DIFF`. Note that the value of `TODAY` is only evaluated at the time a dataset is published (it is not re-evaluated with each query).

## Syntax

```
TODAY()
```

## Return Value

Returns the current system date (no time) as a `date` value.

## Examples

Calculate a user's age using `YEAR_DIFF` to subtract the value of the *birthdate* field in the *users* dataset from the current date:

```
YEAR_DIFF(TODAY(),users.birthdate)
```

Calculate the number of days since a product's release using `DAYS_BETWEEN` to subtract the value of the *release_date* field from the current date:

```
DAYS_BETWEEN(TODAY(),release_date)
```

#### 1.12.6.9 | TRUNC

## Description

`TRUNC` is a row function that truncates a `date` value to the specified format.

## Syntax

```
TRUNC(date, "format")
```

## Return Value

Returns a value of type `Date` truncated to the specified format.

## Input Parameters

**date**
Required. A field or expression that returns a `date` value.

**format**
Required. One of the following format values:

- **`millisecond`** - Returns a date value truncated to millisecond granularity. Has no effect since millisecond is already the most granular format for date values. For example, an input date value of `2012-08-15 20:38:40.213` would return a date value of `2012-08-15 20:38:40.213`.
- **`second`** - Returns a date value truncated to second granularity. For example, an input date value of `2012-08-15 20:38:40.213` would return a date value of `2012-08-15 20:38:40.000`.
- **`minute`** - Returns a date value truncated to minute granularity. For example, an input date value of `2012-08-15 20:38:40.213` would return a date value of `2012-08-15 20:38:00.000`.
- **`hour`** - Returns a date value truncated to hour granularity. For example, an input date value of `2012-08-15 20:38:40.213` would return a date value of `2012-08-15 20:00:00.000`.
- **`day`** - Returns a date value truncated to day granularity. For example, an input date value of `2012-08-15 20:38:40.213` would return a date value of `2012-08-15 00:00:00.000`.
- **`week`** - Returns a date value truncated to the first day of the week (starting on a Monday). For example, an input date value of `2012-08-15` (a Wednesday) would return a date value of `2012-08-13` (the Monday prior).
- **`month`** - Returns a date value truncated to the first day of the month. For example, an input date value of `2012-08-15` would return a date value of `2012-08-01`.
- **`quarter`** - Returns a date value truncated to the first day of the quarter (January 1, April 1, July 1, or October 1). For example, an input date value of `2012-08-15 20:38:40.213` would return a date value of `2012-07-01`.
- **`year`** - Returns a date value truncated to the first day of the year (January 1). For example, an input date value of `2012-08-15` would return a date value of `2012-01-01`.
- **`weekyear`** - Returns a date value trucated to the first day of the ISO weekyear (the ISO week starting with the Monday which is nearest in time to January 1). For example, an input date value of `2008-08-15` would return a date value of `2007-12-31`. The first day of the ISO weekyear for 2008 is December 31, 2007 (the prior Monday closest to January 1).

## Examples

Truncate the *order_date* date field to day granularity:

```
TRUNC(order_date,"day")
```

Cast the value of the *order_date* `Text` field to a date value using `TO_DATE`, and truncate it to day granularity:

```
TRUNC(TO_DATE(order_date, "MM/dd/yyyy HH:mm:ss"), "day")
```

#### 1.12.6.10 | YEAR_DIFF

## Description

`YEAR_DIFF` is a row function that calculates the fractional number of years between two `date` values (`value1` - `value2`).

## Syntax

```
YEAR_DIFF(date_1,date_2)
```

## Return Value

Returns one value per row of type `Double`.

## Input Parameters

**date_1**
Required. A field or expression of type `Date`.

**date_2**
Required. A field or expression of type `Date`.

## Examples

Calculate the number of years a user has been a customer by subtracting the value of the *registration_date* field from the current date (the result of the `TODAY` expression):

```
YEAR_DIFF(TODAY(),registration_date)
```

Calculate a user's age by subtracting the value of the *birthdate* field from the current date (the result of the `TODAY` expression):

```
YEAR_DIFF(TODAY(),birthdate)
```

### 1.12.7 | Informational Functions

#### 1.12.7.1 | IS_VALID

## Description

`IS_VALID` is a row function that returns `0` if the returned value is `NULL`, and `1` if the returned value is `NOT NULL`. This is useful for computing other calculations where you want to exclude NULL values (such as when computing averages).

## Syntax

```
IS_VALID(expression)
```

## Return Value

Returns `0` if the returned value is `NULL`, and `1` if the returned value is `NOT NULL`.

## Input Parameters

**expression**
Required. A field name or expression.

## Examples

Define a Prism calculated field using `IS_VALID`. This returns a row count only for the rows where this field value is `NOT NULL`. If a value is `NULL`, it returns 0 for that row. In this example, we create a Prism calculated field (*sale_amount_not_null*) using the *sale_amount* field as the basis.

```
IS_VALID(sale_amount)
```

Then you can use the *sale_amount_not_null* Prism calculated field to calculate an accurate average for *sale_amount* that excludes `NULL` values:

```
SUM(sale_amount)/SUM(sale_amount_not_null)
```

### 1.12.8 | Instance Functions

#### 1.12.8.1 | CREATE_MULTI_INSTANCE

## Description

`CREATE_MULTI_INSTANCE` is a row function that constructs a Multi-Instance field from one or more provided Multi-Instance or Instance fields.

## Syntax

```
CREATE_MULTI_INSTANCE(field_name [, field_name])
```

## Return Value

Returns one value per row of type Multi-Instance.

## Input Parameters

**field_name**
Required. A field of type Multi-Instance or Instance. All instance values must use the same business object.

## Examples

Create a Multi-Instance field out of multiple Instance fields:

```
CREATE_MULTI_INSTANCE([Journal1], [Journal2], [Journal3])
```

Create a Multi-Instance field out of instance values:

```
CREATE_MULTI_INSTANCE(
CAST("070b0d082eee44e1928c808cc739b35f" AS Instance(eecb565181284b6a8ae8b45dc3ed1451)),
CAST("f4c49debb3dc483baa8707dfe683503c" AS Instance(eecb565181284b6a8ae8b45dc3ed1451))
)
```

#### 1.12.8.2 | INSTANCE_CONTAINS_ANY

## Description

INSTANCE_CONTAINS_ANY is a row function that compares a Multi-Instance or Instance field to either a Multi-Instance field, an Instance field, or to a list of instance values, and returns True if at least one instance value exists in the first argument, and False if none of them exist.

## Syntax

```
INSTANCE_CONTAINS_ANY(input_field, comparison_value [, comparison_value])
```

## Return Value

Returns one value per row of type Boolean. This function returns NULL when it receives a Text value that isn't formatted as a valid instance value (WID format).

## Input Parameters

**input_field**
Required. A field of type Multi-Instance or Instance.

**comparison_value**
Required. A field of type Multi-Instance or Instance, or a Text value of a valid instance value.

## Examples

Compare the Worktags Multi-Instance field to the Instance fields Cost Center 1 and Cost Center 2:

```
INSTANCE_CONTAINS_ANY([Worktags], [Cost Center 1],[Cost Center 2])
```

#### 1.12.8.3 | INSTANCE_COUNT

## Description

INSTANCE_COUNT is a row function that returns the total number of instance values in a Multi-Instance or Instance field. This function returns 0 when the field is empty.

## Syntax

```
INSTANCE_COUNT(field_name)
```

## Return Value

Returns one value per row of type Integer.

## Input Parameters

**field_name**
Required. A field of type Multi-Instance or Instance.

## Examples

Count the number of instance values in the Multi-Instance field Journal Lines:

```
INSTANCE_COUNT([Journal Lines])
```

#### 1.12.8.4 | INSTANCE_EQUALS

## Description

INSTANCE_EQUALS is a row function that compares a Multi-Instance or Instance field to either a Multi-Instance field, an Instance field, or to a list of instance values, and checks if the first argument exactly matches the instance values provided in the other arguments.

## Syntax

```
INSTANCE_EQUALS(input_field, comparison_value [, comparison_value])
```

## Return Value

Returns one value per row of type Boolean. This function returns NULL when it receives a Text value that isn't formatted as a valid instance value (WID format).

## Input Parameters

**input_field**
Required. A field of type Multi-Instance or Instance.

**comparison_value**
Required. A field of type Multi-Instance or Instance, or a Text value of a valid instance value.

## Examples

Compare the Worktags Multi-Instance field to the Instance fields Cost Center 1 and Cost Center 2:

```
INSTANCE_EQUALS([Worktags], [Cost Center 1],[Cost Center 2])
```

#### 1.12.8.5 | INSTANCE_IS_SUPERSET_OF

## Description

INSTANCE_IS_SUPERSET_OF is a row function that compares a Multi-Instance field to either a Multi-Instance field, an Instance field, or to a list of instance values, and returns True if every instance value exists in the first argument, and False if at least one doesn't exist.

## Syntax

```
INSTANCE_IS_SUPERSET_OF(input_field, comparison_value [, comparison_value])
```

## Return Value

Returns one value per row of type Boolean. This function returns NULL when it receives a Text value that isn't formatted as a valid instance value (WID format).

## Input Parameters

**input_field**
Required. A field of type Multi-Instance.

**comparison_value**
Required. A field of type Multi-Instance or Instance, or a Text value of a valid instance value.

## Examples

Compare the Worktags Multi-Instance field to the Instance fields Cost Center 1 and Cost Center 2:

```
INSTANCE_IS_SUPERSET_OF([Worktags], [Cost Center 1],[Cost Center 2])
```

### 1.12.9 | Logical Functions

#### 1.12.9.1 | CASE

## Description

CASE is a row function that evaluates each row in the dataset according to one or more input conditions, and outputs the specified result when the input conditions are met.

## Syntax

```
CASE WHEN input_condition [AND|OR input_condition]THEN output_expression [...] [ELSE other_output_expression] END
```

## Return Value

Returns one value per row of the same type as the output expression. All output expressions must return the same field type.

If there are multiple output expressions that return different field types, then you will need to enclose your entire CASE expression in one of the field type conversion functions, such as TO_INT, to explicitly cast all output values to a particular field type.

## Input Parameters

**WHEN *input_condition***
Required. The `WHEN` keyword is used to specify one or more Boolean expressions (see the supported conditional operators). If an input value meets the condition, then the output expression is applied. Input conditions can include other row functions in their expression, but cannot contain summarization functions or measure expressions. You can use the `AND` or `OR` keywords to combine multiple input conditions.

**THEN *output_expression***
Required. The `THEN` keyword is used to specify an output expression when the specified conditions are met. Output expressions can include other row functions in their expression, but cannot contain summarization functions or measure expressions.

**ELSE *other_output_expression***
Optional. The `ELSE` keyword can be used to specify an alternate output expression to use when the specified conditions are not met. If an `ELSE` expression is not supplied, `ELSE NULL` is the default.

**END**
Required. Denotes the end of `CASE` function processing.

## Examples

Convert values in the *age* column into range-based groupings (binning):

```
CASE WHEN age <= 25 THEN "0-25" WHEN age <= 50 THEN "26-50" ELSE "over 50" END
```

Transform values in the *gender* column from one string to another:

```
CASE WHEN gender = "M" THEN "Male" WHEN gender = "F" THEN "Female" ELSE "Unknown" END
```

The *vehicle* column contains the following values: truck, bus, car, scooter, wagon, bike, tricycle, and motorcycle. The following example converts multiple values in the *vehicle* column into a single value:

```
CASE WHEN vehicle in ("bike","scooter","motorcycle") THEN "two-wheelers" ELSE "other" END
```

### Related Information

**Reference**
Comparison Operators
Logical Operators
Arithmetic Operators

#### 1.12.9.2 | COALESCE

## Description

`COALESCE` is a row function that returns the first valid value (`NOT NULL` value) from a comma-separated list of expressions.

## Syntax

```
COALESCE(expression[,expression][,...])
```

## Return Value

Returns one value per row of the same type as the first valid input expression.

## Input Parameters

**expression**
At least one required. A field name or expression.

## Examples

The following example shows an expression to calculate employee yearly income for exempt employees that have a *salary* and non-exempt employees that have an *hourly_wage*. This expression checks the values of both fields for each row, and returns the value of the first expression that is valid (`NOT NULL`).

```
COALESCE(hourly_wage * 40 * 52, salary)
```

### 1.12.10 | Math Functions

#### 1.12.10.1 | DIV

## Description

`DIV` is a row function that divides two `Long` values and returns a quotient value of type `Long` (the result is truncated to 0 decimal places).

## Syntax

```
DIV(dividend,divisor)
```

## Return Value

Returns one value per row of type `Long`.

## Input Parameters

**dividend**
Required. A field or expression of type `Long`.

**divisor**
Required. A field or expression of type `Long`.

## Examples

Cast the value of the *file_size* field to `Long` and divide by 1024:

```
DIV(TO_LONG(file_size),1024)
```

#### 1.12.10.2 | EXP

## Description

`EXP` is a row function that raises the mathematical constant `e` to the power (exponent) of a numeric value and returns a value of type `Double`.

## Syntax

```
EXP(power)
```

## Return Value

Returns one value per row of type `Double`.

## Input Parameters

**power**
Required. A field or expression of a numeric type.

## Examples

Raise *e* to the power in the *Value* field.

```
EXP(Value)
```

When the *Value* field value is 2.0, the result is equal to 7.3890 when truncated to four decimal places.

#### 1.12.10.3 | FLOOR

## Description

`FLOOR` is a row function that returns the largest integer that is less than or equal to the input argument.

## Syntax

```
FLOOR(double)
```

## Return Value

Returns one value per row of type `Double`.

## Input Parameters

**double**
Required. A field or expression of type `Double`.

## Examples

Return the floor value of *32.6789*:

```
FLOOR(32.6789) returns 32.0
```

#### 1.12.10.4 | HASH

## Description

`HASH` is a row function that evenly partitions data values into the specified number of buckets. It creates a hash of the input value and assigns that value a bucket number. Equal values will always hash to the same bucket number.

## Syntax

```
HASH(field_name,integer)
```

## Return Value

Returns one value per row of type `Integer` corresponding to the bucket number that the input value hashes to.

## Input Parameters

**field_name**
Required. The name of the field whose values you want to partition. When this value is NULL and the integer parameter is a value other than zero or NULL, the function returns zero, otherwise it returns NULL.

**integer**
Required. The desired number of buckets. This parameter can be a numeric value of any field type, but when it is a non-integer value, the value is truncated to an integer. When the value is zero or NULL, the function returns NULL. When the value is negative, the function uses absolute value.

## Examples

Partition the values of the *username* field into 20 buckets:

```
HASH(username,20)
```

#### 1.12.10.5 | LN

## Description

`LN` is a row function that returns the natural logarithm of a number. The natural logarithm is the logarithm to the base `e`, where `e` (Euler's number) is a mathematical constant approximately equal to 2.718281828. The natural logarithm of a number `x` is the power to which the constant `e` must be raised in order to equal `x`.

## Syntax

```
LN(positive_number)
```

## Return Value

Returns the exponent to which base *e* must be raised to obtain the input value, where *e* denotes the constant number 2.718281828. The return value is the same field type as the input value.

For example, `LN(7.389)` is 2, because *e* to the power of 2 is approximately 7.389.

## Input Parameters

**positive_number**
Required. A field or expression that returns a number greater than 0. Inputs can be of type `Integer`, `Long`, `Double`.

## Examples

Return the natural logarithm of base number *e*, which is approximately *2.718281828*:

```
LN(2.718281828) returns 1

LN(3.0000) returns 1.098612

LN(300.0000) returns 5.703782
```

#### 1.12.10.6 | MOD

## Description

`MOD` is a row function that divides two `Long` values and returns the remainder value of type `Long` (the result is truncated to 0 decimal places).

## Syntax

```
MOD(dividend,divisor)
```

## Return Value

Returns one value per row of type `Long`.

## Input Parameters

**dividend**
Required. A field or expression of type `Long`.

**divisor**
Required. A field or expression of type `Long`.

## Examples

Cast the value of the *file_size* field to `Long` and divide by 1024:

```
MOD(TO_LONG(file_size),1024)
```

#### 1.12.10.7 | POW

## Description

`POW` is a row function that raises the a numeric value to the power (exponent) of another numeric value and returns a value of type `Double`.

## Syntax

```
POW(index,power)
```

## Return Value

Returns one value per row of type `Double`.

## Input Parameters

**index**
Required. A field or expression of a numeric type.

**power**
Required. A field or expression of a numeric type.

## Examples

Calculate the compound annual growth rate (CAGR) percentage for a given investment over a five year span.

```
100 * POW(end_value/start_value, 0.2) - 1
```

Calculate the square of the *Value* field.

```
POW(Value,2)
```

Calculate the square root of the *Value* field.

```
POW(Value,0.5)
```

The following expression returns 1.

```
POW(0,0)
```

#### 1.12.10.8 | ROUND

## Description

`ROUND` is a row function that rounds a numeric value to the specified number of decimal places and returns a value of type `Double`.

## Syntax

```
ROUND(numeric_expression,number_decimal_places)
```

## Return Value

Returns one value per row of type `Double`.

## Input Parameters

**numeric_expression**
Required. A field or expression of any numeric type.

**number_decimal_places**
Required. An integer that specifies the number of decimal places to round to.

## Examples

Round the number *32.4678954* to two decimal places:

`ROUND(32.4678954,2) returns 32.47`

### 1.12.11 | Text Functions

#### 1.12.11.1 | CIDR_MATCH

## Description

`CIDR_MATCH` is a row function that compares two `Text` arguments representing a CIDR mask and an IP address, and returns `1` if the IP address falls within the specified subnet mask or `0` if it does not.

## Syntax

```
CIDR_MATCH(CIDR_string, IP_string)
```

## Return Value

Returns an `Integer` value of `1` if the IP address falls within the subnet indicated by the CIDR mask and `0` if it does not.

## Input Parameters

**CIDR_string**
Required. A field or expression that returns a `Text` value containing either an IPv4 or IPv6 CIDR mask (Classless InterDomain Routing subnet notation). An IPv4 CIDR mask can only successfully match IPv4 addresses, and an IPv6 CIDR mask can only successfully match IPv6 addresses.

**IP_string**
Required. A field or expression that returns a `Text` value containing either an IPv4 or IPv6 internet protocol (IP) address.

## Examples

Compare an IPv4 CIDR subnet mask to an IPv4 IP address:

`CIDR_MATCH("60.145.56.0/24","60.145.56.246") returns 1`

`CIDR_MATCH("60.145.56.0/30","60.145.56.246") returns 0`

Compare an IPv6 CIDR subnet mask to an IPv6 IP address:

`CIDR_MATCH("fe80::/70","FE80::0202:B3FF:FE1E:8329") returns 1`

`CIDR_MATCH("fe80::/72","FE80::0202:B3FF:FE1E:8329") returns 0`

#### 1.12.11.2 | CONCAT

## Description

`CONCAT` is a row function that returns a `Text` by concatenating (combining together) the results of multiple `Text` expressions.

## Syntax

```
CONCAT(value_expression[,value_expression][,...])
```

## Return Value

Returns one value per row of type `Text`.

## Input Parameters

**value_expression**
At least one required. A field name of any type, a literal string or number, or an expression that returns any value.

## Examples

Combine the values of the *month*, *day*, and *year* fields into a single date field formatted as *MM/DD/YYYY*.

```
CONCAT(month,"/",day,"/",year)
```

#### 1.12.11.3 | EXTRACT_COOKIE

## Description

`EXTRACT_COOKIE` is a row function that extracts the value of the given cookie identifier from a semi-colon delimited list of cookie key/value pairs. This function can be used to extract a particular cookie value from a combined web access log Cookie column.

## Syntax

```
EXTRACT_COOKIE("cookie_list_string",cookie_key_string)
```

## Return Value

Returns the value of the specified cookie key as type `Text`.

## Input Parameters

**cookie_list_string**
Required. A field of type `Text` or literal string that has a semi-colon delimited list of cookie *key=value* pairs.

**cookie_key_string**
Required. The cookie key name for which to extract the cookie value.

## Examples

Extract the value of the *vID* cookie from a literal cookie string:

`EXTRACT_COOKIE("SSID=ABC; vID=44", "vID") returns 44`

Extract the value of the *vID* cookie from a field named *Cookie*:

```
EXTRACT_COOKIE(Cookie,"vID")
```

#### 1.12.11.4 | EXTRACT_VALUE

## Description

`EXTRACT_VALUE` is a row function that extracts the value for the given key from a string containing delimited key/value pairs.

## Syntax

```
EXTRACT_VALUE(string,key_name [,delimiter] [,pair_delimiter])
```

## Return Value

Returns the value of the specified key as type `Text`.

## Input Parameters

**string**
Required. A field of type `Text` or literal string that contains a delimited list of key/value pairs.

**key_name**
Required. The key name for which to extract the value.

**delimiter**
Optional. The delimiter used between the key and the value. If not specified, the value `u0003` is used. This is the Unicode escape sequence for the `start of text` character.

**pair_delimiter**
Optional. The delimiter used between key/value pairs when the input string contains more than one key/value pair. If not specified, the value `u0002` is used. This is the Unicode escape sequence for the `end of text` character.

## Examples

Extract the value of the *lastname* key from a literal string of key/value pairs:

`EXTRACT_VALUE("firstname;daria|lastname;hutch","lastname",";","|") returns` *hutch*

Extract the value of the *email* key from a `Text` field named *contact_info* that contains strings in the format of *key:value,key:value*:

```
EXTRACT_VALUE(contact_info,"email",":",",")
```

#### 1.12.11.5 | FILE_NAME

## Description

`FILE_NAME` is a row function that returns the original file name from the source file system. This is useful when the source data that comprises a dataset comes from multiple files, and there is useful information in the file names themselves (such as dates or server names). You can use `FILE_NAME` in combination with other text processing functions to extract useful information from the file name.

## Syntax

```
FILE_NAME()
```

## Return Value

Returns one value per row of type `Text`.

## Examples

Your dataset is based on daily log files that use an 8 character date as part of the file name. For example, *20120704.log* is the file name used for the log file created on July 4, 2012. The following expression uses `FILE_NAME` in combination with `SUBSTRING` and `TO_DATE` to create a date field from the first 8 characters of the file name.

```
TO_DATE(SUBSTRING(FILE_NAME(),0,8),"yyyyMMdd")
```

Your dataset is based on log files that use the server IP address as part of the file name. For example, *172.12.131.118.log* is the log file name for server 172.12.131.118. The following expression uses `FILE_NAME` in combination with `REGEX` to extract the IP address from the file name.

```
REGEX(FILE_NAME(),"(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\.log")
```

#### 1.12.11.6 | HEX_TO_IP

## Description

HEX_TO_IP is a row function that converts a hexadecimal-encoded Text value to a text representation of an IP address.

## Syntax

```
HEX_TO_IP(string)
```

## Return Value

Returns a value of type Text representing either an IPv4 or IPv6 address. The type of IP address returned depends on the input string. An 8 character hexadecimal string returns an IPv4 address. A 32 character long hexadecimal string returns an IPv6 address.

IPv6 addresses are represented in full length, without removing any leading zeros and without using the compressed :: notation. For example, 2001:0db8:0000:0000:0000:ff00:0042:8329 rather than 2001:db8::ff00:42:8329.

Input strings that don't contain either 8 or 32 valid hexadecimal characters return NULL.

## Input Parameters

**string**
Required. A field or expression that returns a hexadecimal-encoded Text value. The hexadecimal string must be either 8 characters long (in which case it's converted to an IPv4 address) or 32 characters long (in which case it's converted to an IPv6 address).

## Examples

Return a plain text IP address for each hexadecimal-encoded string value in the *byte_encoded_ips* column:

```
HEX_TO_IP(byte_encoded_ips)
```

Convert an 8 character hexadecimal-encoded string to a plain text IPv4 address:

HEX_TO_IP(AB20FE01) returns *171.32.254.1*

Convert a 32 character hexadecimal-encoded string to a plain text IPv6 address:

HEX_TO_IP(FE80000000000000202B3FFFE1E8329) returns *fe80:0000:0000:0000:0202:b3ff:fe1e:8329*

#### 1.12.11.7 | INSTR

## Description

INSTR is a row function that returns an integer indicating the position of a character within a string that is the first character of the occurrence of a substring. The INSTR function is similar to the FIND function in Excel, except that the first letter is position 0 and the order of the arguments is reversed.

## Syntax

```
INSTR(search_string,substring,position,occurrence)
```

## Return Value

Returns one value per row of type Integer. The first position is indicated with the value of zero (0).

## Input Parameters

**search_string**
Required. The name of a field or expression of type Text (or a literal string).

**substring**
Required. A literal string or name of a field that specifies the substring to search for in *search_string*. Note that to search for the double quotation mark ( " ) as a literal string, you must escape it with another double quotation mark: ""

**position**
Optional. An integer that specifies at which character in *search_string* to start searching for *substring*. A value of 0 (zero) starts the search at the beginning of *search_string*. Use a positive integer to start searching from the beginning of *search_string*, and use a negative integer to start searching from the end of *search_string*. When no position is specified, INSTR searches at the beginning of the string (0).

**occurrence**
Optional. A positive integer that specifies which occurrence of *substring* to search for. When no occurrence is specified, INSTR searches for the first occurrence of the substring (1).

## Examples

Return the position of the first occurrence of the substring "http://" starting at the end of the *url* field:

```
INSTR(url,"http://",-1,1)
```

The following expression searches for the second occurrence of the substring "st" starting at the beginning of the string "besttteststring". INSTR finds that the substring starts at the seventh character in the string, so it returns 6:

```
INSTR("besttteststring","st",0,2)
```

The following expression searches backward for the second occurrence of the substring "st" starting at 7 characters before the end of the string "besttteststring". INSTR finds that the substring starts at the third character in the string, so it returns 2:

```
INSTR("besttteststring","st",-7,2)
```

#### 1.12.11.8 | JAVA_STRING

## Description

`JAVA_STRING` is a row function that returns the unescaped version of a Java unicode character escape sequence as a `Text` value. This is useful when you want to specify unicode characters in an expression. For example, you can use `JAVA_STRING` to specify the unicode value representing a control character.

## Syntax

```
JAVA_STRING(unicode_escape_sequence)
```

## Return Value

Returns the unescaped version of the specified unicode character, one value per row of type `Text`.

## Input Parameters

**unicode_escape_sequence**
Required. A `Text` value containing a unicode character expressed as a Java unicode escape sequence. Unicode escape sequences consist ofa backslash ' \' (ASCII character 92, hex 0x5c), a ' u' (ASCII 117, hex 0x75), optionally one or more additional ' u' characters, and four hexadecimal digits (the characters ' 0' through ' 9' or ' a' through ' f' or ' A' through ' F'). Such sequences represent the UTF-16 encoding of a Unicode character. For example, the letter ' a' is equivalent to ' \u0061'.

## Examples

Evaluates whether the *currency* field is equal to the yen symbol.

```
CASE WHEN currency == JAVA_STRING("\u00a5") THEN "yes" ELSE "no" END
```

#### 1.12.11.9 | JOIN_STRINGS

## Description

`JOIN_STRINGS` is a row function that returns a `Text` by concatenating (combining together) the results of multiple `Text` values with the separator in between each non-null value.

## Syntax

```
JOIN_STRINGS(separator,value_expression[,value_expression][,...])
```

## Return Value

Returns one value per row of type `Text`.

## Input Parameters

**separator**
Required. A field name of type `Text`, a literal string, or an expression that returns a `Text`.

**value_expression**
At least one required. A field name of any type, a literal string or number, or an expression that returns any value.

## Examples

Combine the values of the *month*, *day*, and *year* fields into a single date field formatted as *MM/DD/YYYY*.

```
JOIN_STRINGS("/",month,day,year)
```

The following expression returns NULL:

```
JOIN_STRINGS("+",NULL,NULL,NULL)
```

The following expression returns a+b:

```
JOIN_STRINGS("+","a","b",NULL)
```

#### 1.12.11.10 | JSON_DECIMAL

## Description

`JSON_DECIMAL` is a row function that extracts a `Numeric` value from a field in a JSON object.

## Syntax

```
JSON_DECIMAL(json_string, "json_field")
```

## Return Value

Returns one value per row of type `Numeric`.

## Input Parameters

**json_string**
Required. The name of a field or expression of type `Text` (or a literal string) that contains a valid JSON object.

**json_field**
Required. The key or name of the field value you want to extract.

For top-level fields, specify the name identifier (key) of the field.

To access fields within a nested object, specify a dot-separated path of field names (for example *top_level_field_name.nested_field_name*).

To extract a value from an array, specify the dot-separated path of field names and the array position starting at 0 for the first value in an array, 1 for the second value, and so on (for example, *field_name.0*).

If the name identifier contains dot or period characters within the name itself, escape the name by enclosing it in brackets (for example, [ *field.name.with.dot*].[ *another.dot.field.name*]

If the field name is null (empty), use brackets with nothing in between as the identifier, for example `[ ]`.

## Examples

If you had a *top_scores* field that contained a JSON object formatted like this (with the values contained in an array):

```
{"practice_scores":["538.67","674.99","1021.52"], "test_scores":["753.21","957.88","1032.87"]}
```

You could extract the third value of the *test_scores* array using the expression, which returns `"1032.87"`:

```
JSON_DECIMAL(top_scores,"test_scores.2")
```

#### 1.12.11.11 | JSON_DOUBLE

## Description

`JSON_DOUBLE` is a row function that extracts a `Double` value from a field in a JSON object.

## Syntax

```
JSON_DOUBLE(json_string, "json_field")
```

## Return Value

Returns one value per row of type `Double`.

## Input Parameters

**json_string**
Required. The name of a field or expression of type `Text` (or a literal string) that contains a valid JSON object.

**json_field**
Required. The key or name of the field value you want to extract.

For top-level fields, specify the name identifier (key) of the field.

To access fields within a nested object, specify a dot-separated path of field names (for example *top_level_field_name.nested_field_name*).

To extract a value from an array, specify the dot-separated path of field names and the array position starting at 0 for the first value in an array, 1 for the second value, and so on (for example, *field_name.0*).

If the name identifier contains dot or period characters within the name itself, escape the name by enclosing it in brackets (for example, [ *field.name.with.dot*].[ *another.dot.field.name*]

If the field name is null (empty), use brackets with nothing in between as the identifier, for example `[ ]`.

## Examples

If you had a *top_scores* field that contained a JSON object formatted like this (with the values contained in an array):

```
{"practice_scores":["538.67","674.99","1021.52"], "test_scores":["753.21","957.88","1032.87"]}
```

You could extract the third value of the *test_scores* array using the expression, which returns `"1032.87"`:

```
JSON_DOUBLE(top_scores,"test_scores.2")
```

#### 1.12.11.12 | JSON_INTEGER

## Description

`JSON_INTEGER` is a row function that extracts an `Integer` value from a field in a JSON object.

## Syntax

```
JSON_INTEGER(json_string, "json_field")
```

## Return Value

Returns one value per row of type `Integer`.

## Input Parameters

**json_string**
Required. The name of a field or expression of type `Text` (or a literal string) that contains a valid JSON object.

**json_field**
Required. The key or name of the field value you want to extract.

For top-level fields, specify the name identifier (key) of the field.

To access fields within a nested object, specify a dot-separated path of field names (for example *top_level_field_name.nested_field_name*).

To extract a value from an array, specify the dot-separated path of field names and the array position starting at 0 for the first value in an array, 1 for the second value, and so on (for example, *field_name.0*).

If the name identifier contains dot or period characters within the name itself, escape the name by enclosing it in brackets (for example, [ *field.name.with.dot*].[ *another.dot.field.name*]

If the field name is null (empty), use brackets with nothing in between as the identifier, for example `[]`.

## Examples

If you had an *address* field that contained a JSON object formatted like this:

```
{"street_address":"123 B Street", "city":"San Mateo", "state":"CA", "zip_code":"94403"}
```

You could extract the *zip_code* value using the expression, which returns `"94403"`:

```
JSON_INTEGER(address,"zip_code")
```

If you had a *top_scores* field that contained a JSON object formatted like this (with the values contained in an array):

```
{"practice_scores":["538","674","1021"], "test_scores":["753","957","1032"]}
```

You could extract the third value of the *test_scores* array using the expression, which returns `"1032"`:

```
JSON_INTEGER(top_scores,"test_scores.2")
```

#### 1.12.11.13 | JSON_LONG

## Description

`JSON_LONG` is a row function that extracts a `Long` value from a field in a JSON object.

## Syntax

```
JSON_LONG(json_string, "json_field")
```

## Return Value

Returns one value per row of type `Long`.

## Input Parameters

**json_string**
Required. The name of a field or expression of type `Text` (or a literal string) that contains a valid JSON object.

**json_field**
Required. The key or name of the field value you want to extract.

For top-level fields, specify the name identifier (key) of the field.

To access fields within a nested object, specify a dot-separated path of field names (for example *top_level_field_name.nested_field_name*).

To extract a value from an array, specify the dot-separated path of field names and the array position starting at 0 for the first value in an array, 1 for the second value, and so on (for example, *field_name.0*).

If the name identifier contains dot or period characters within the name itself, escape the name by enclosing it in brackets (for example, [ *field.name.with.dot*].[ *another.dot.field.name*]

If the field name is null (empty), use brackets with nothing in between as the identifier, for example `[]`.

## Examples

If you had a *top_scores* field that contained a JSON object formatted like this (with the values contained in an array):

```
{"practice_scores":["538","674","1021"], "test_scores":["753","957","1032"]}
```

You could extract the third value of the *test_scores* array using the expression, which returns `"1032"`:

```
JSON_LONG(top_scores,"test_scores.2")
```

#### 1.12.11.14 | JSON_STRING

## Description

`JSON_STRING` is a row function that extracts a `Text` value from a field in a JSON object.

## Syntax

```
JSON_STRING(json_string, "json_field")
```

## Return Value

Returns one value per row of type `Text`.

## Input Parameters

**json_string**
Required. The name of a field or expression of type `Text` (or a literal string) that contains a valid JSON object.

**json_field**
Required. The key or name of the field value you want to extract.

For top-level fields, specify the name identifier (key) of the field.

To access fields within a nested object, specify a dot-separated path of field names (for example *top_level_field_name.nested_field_name*).

To extract a value from an array, specify the dot-separated path of field names and the array position starting at 0 for the first value in an array, 1 for the second value, and so on (for example, *field_name.0*).

If the name identifier contains dot or period characters within the name itself, escape the name by enclosing it in brackets (for example, [ *field.name.with.dot*].[ *another.dot.field.name*]

If the field name is null (empty), use brackets with nothing in between as the identifier, for example `[]`.

## Examples

If you had an *address* field that contained a JSON object formatted like this:

```
{"street_address":"123 B Street", "city":"San Mateo", "state":"CA", "zip":"94403"}
```

You could extract the *state* value using the expression:

```
JSON_STRING(address,"state")
```

If you had a *misc* field that contained a JSON object formatted like this (with the values contained in an array):

```
{"hobbies":["sailing","hiking","cooking"], "interests":["art","music","travel"]}
```

You could extract the first value of the *hobbies* array using the expression, which returns `"sailing"`:

```
JSON_STRING(misc,"hobbies.0")
```

#### 1.12.11.15 | LENGTH

## Description

`LENGTH` is a row function that returns the count of characters in a `Text` value.

## Syntax

```
LENGTH(string_expression)
```

## Return Value

Returns one value per row of type `Integer`.

## Input Parameters

**string_expression**
Required. The name of a field or expression of type `Text` (or a literal string).

## Examples

Return count of characters from values in the *name* field. For example, the value *Bob* would return a length of *3*, *Julie* would return a length of *5*, and so on:

```
LENGTH(name)
```

#### 1.12.11.16 | REGEX

## Description

`REGEX` is a row function that performs a whole string match against a `Text` value with a regular expression and returns the portion of the string matching the first capturing group of the regular expression.

## Syntax

```
REGEX(string_expression,"regex_matching_pattern")
```

## Return Value

Returns the matched `Text` value of the first capturing group of the regular expression. If there is no match, returns `NULL`.

## Input Parameters

**string_expression**
Required. The name of a field or expression of type `Text` (or a literal string).

**regex_matching_pattern**
Required. A regular expression pattern based on the regular expression pattern matching syntax of the Java programming language. To return a non-NULL value, the regular expression pattern must match the entire `Text` value.

## Regular Expression Constructs

See the Regular Expression Reference for information on the constructs used for defining a regular expression matching pattern.

## Capturing and Non-Capturing Groups

Groups are specified by a pair of parenthesis around a subpattern in the regular expression. A pattern can have more than one group and the groups can be nested. The groups are numbered 1-*n* from left to right, starting with the first opening parenthesis. There is always an implicit group 0, which contains the entire match. For example, the pattern:

```
(a(b*))+(c)
```

contains three groups:

```
group 1: (a(b*))
group 2: (b*)
group 3: (c)
```

**Capturing Groups**

By default, a group *captures* the text that produces a match, and only the most recent match is captured. The `REGEX` function returns the string that matches the first capturing group in the regular expression. For example, if the input string to the expression above was `abc`, the entire `REGEX` function would match to `abc`, but only return the result of group 1, which is `ab`.

**Non-Capturing Groups**

In some cases, you may want to use parenthesis to group subpatterns, but not capture text. A non-capturing group starts with `(?:` (a question mark and colon following the opening parenthesis). For example, `h(?:a|i|o)t` matches `hat` or `hit` or `hot`, but does not capture the `a`, `i`, or `o` from the subexpression.

## Examples

Match all possible email addresses with a pattern of *username@provider.domain*, but only return the *provider* portion of the email address from the *email* field:

```
REGEX(email,"^[a-zA-Z0-9._%+-]+@([a-zA-Z0-9._-]+)\.[a-zA-Z]{2,4}$")
```

Match the request line of a web log, where the value is in the format of:

```
GET /some_page.html HTTP/1.1
```

and return just the requested HTML page names:

```
REGEX(weblog.request_line,"GET\s/([a-zA-Z0-9._%-]+\.[html])\sHTTP/[0-9.]+")
```

Extract the inches portion from a `height` field where example values are `6'2"`, `5'11"` (notice the escaping of the literal quote with a double double-quote):

```
REGEX(height, "\d\'(\d)+""")
```

Extract all of the contents of the `device` field when the value is either `iPod`, `iPad`, or `iPhone`:

```
REGEX(device,"(iP[ao]d|iPhone)")
```

#### Related Information

**Concepts**
Concept: Regular Expressions in Prism
**Reference**
Regex Literal and Special Characters
Regex Character Classes
Regex Line and Word Boundaries
Regex Quantifiers
Regex Capturing Groups

#### 1.12.11.17 | REGEX_REPLACE

## Description

`REGEX_REPLACE` is a row function that evaluates a `Text` value against a regular expression to determine if there is a match, and replaces matched strings with the specified replacement value.

## Syntax

```
REGEX_REPLACE(string_expression,"regex_match_pattern","regex_replace_pattern")
```

## Return Value

Returns the *`regex_replace_pattern`* as a `Text` value when *`regex_match_pattern`* produces a match. If there is no match, returns the value of *`string_expression`* as a `Text`.

## Input Parameters

**string_expression**
Required. The name of a field or expression of type `Text` (or a literal string).

**regex_match_pattern**
Required. A string literal or regular expression pattern based on the regular expression pattern matching syntax of the Java programming language. You can use capturing groups to create backreferences that can be used in the *`regex_replace_pattern`* . You might want to use a string literal to make a case-sensitive match. For example, when you enter *jane* as the match value, the function matches *jane* but not *Jane*. The function matches all occurrences of a string literal in the string expression.

**regex_replace_pattern**
Required. A string literal or regular expression pattern based on the regular expression pattern matching syntax of the Java programming language. You can refer to backreferences from the *`regex_match_pattern`* using the syntax *`$n`* (where *`n`* is the group number).

## Regular Expression Constructs

See the Regular Expression Reference for information on the constructs used for defining a regular expression matching pattern.

## Examples

Match the values in a *phone_number* field where phone number values are formatted as `xxx.xxx.xxxx` and replace them with phone number values formatted as `(xxx) xxx-xxxx`:

```
REGEX_REPLACE(phone_number,"([0-9]{3})\.([[0-9]]{3})\.([[0-9]]{4})","\($1\) $2-$3")
```

Match the values in a *name* field where name values are formatted as `firstname lastname` and replace them with name values formatted as `lastname, firstname`:

```
REGEX_REPLACE(name,"(.*) (.*)","$2, $1")
```

Match the string literal *mrs* in a *title* field and replace it with the string literal *Mrs*.

```
REGEX_REPLACE(title,"mrs","Mrs")
```

#### Related Information

**Concepts**
Concept: Regular Expressions in Prism
**Reference**
Regex Literal and Special Characters
Regex Character Classes
Regex Line and Word Boundaries
Regex Quantifiers
Regex Capturing Groups

#### 1.12.11.18 | REVERSE

## Description

`REVERSE` is a row function that returns the characters of a string value in the opposite order.

## Syntax

```
REVERSE(string_expression)
```

## Return Value

Returns one value per row of type `Text`.

## Input Parameters

**string_expression**
Required. The name of a field or expression of type `Text` (or a literal string).

## Examples

Return the string *123 Main Street* in reverse order:

```
REVERSE("123 Main Street")
```

Returns `teertS niaM 321`.

#### 1.12.11.19 | SUBSTRING

## Description

`SUBSTRING` is a row function that returns the specified characters of a `Text` value based on the given start and optional end position.

## Syntax

```
SUBSTRING(search_string,start,end)
```

## Return Value

Returns one value per row of type `Text`.

## Input Parameters

**search_string**
Required. The name of a field or expression of type `Text` (or a literal string).

**start**
Required. An integer that specifies where the returned characters start (inclusive), with 0 being the first character of the string. If *start* is greater than the number of characters, then an empty string is returned. If *start* is greater than *end*, then an empty string is returned.

**end**
Optional. A positive integer that specifies where the returned characters end (exclusive), with the *end* character not being part of the return value. If *end* is greater than the number of characters, or is not specified, then the whole string value (from *start*) is returned.

## Examples

Return the first letter of the *name* field:

```
SUBSTRING(name,0,1)
```

#### 1.12.11.20 | TO_LOWER

## Description

`TO_LOWER` is a row function that converts all alphabetic characters in a `Text` value to lower case.

## Syntax

```
TO_LOWER(string_expression)
```

## Return Value

Returns one value per row of type `Text`.

## Input Parameters

**string_expression**
Required. The name of a field or expression of type `Text` (or a literal string).

## Examples

Return the literal input string *123 Main Street* in all lower case letters:

`TO_LOWER("123 Main Street") returns` *`123 main street`*

#### 1.12.11.21 | TO_PROPER

### Description

`TO_PROPER` is a row function that returns a `Text` value with the first letter of each word capitalized.

### Syntax

```
TO_PROPER(string_expression)
```

### Return Value

Returns one value per row of type `Text`.

### Input Parameters

**string_expression**
Required. The name of a field or expression of type `Text` (or a literal string).

### Examples

```
TO_PROPER("123 Alameda de las Pulgas, San Mateo CA")
```

Returns `123 Alameda De Las Pulgas, San Mateo Ca`

#### 1.12.11.22 | TO_UPPER

### Description

`TO_UPPER` is a row function that converts all alphabetic characters in a `Text` value to upper case.

### Syntax

```
TO_UPPER(string_expression)
```

### Return Value

Returns one value per row of type `Text`.

### Input Parameters

**string_expression**
Required. The name of a field or expression of type `Text` (or a literal string).

### Examples

```
TO_UPPER("123 Main Street")
```

Returns `123 MAIN STREET`

#### 1.12.11.23 | TRIM

### Description

`TRIM` is a row function that removes leading and trailing spaces from a `Text` value.

### Syntax

```
TRIM(string_expression)
```

### Return Value

Returns one value per row of type `Text`.

### Input Parameters

**string_expression**
Required. The name of a field or expression of type `Text` (or a literal string).

### Examples

Return the value of the *area_code* field without any leading or trailing spaces. Example:

TRIM(area_code)

`TRIM(" 650 ")` returns `650`.

`TRIM(" 650 123-4567 ")` returns `650 123-4567`. Note that the extra spaces in the middle of the string aren't removed, only the spaces at the beginning and end of the string.

#### 1.12.11.24 | XPATH_STRING

## Description

XPATH_STRING is a row function that takes XML and returns the first string matching the given XPath expression.

## Syntax

```
XPATH_STRING(xml_expression,"xpath_expression")
```

## Return Value

Returns one value per row of type Text.

If the XPath expression matches more than one string in the given XML node, this function will return the *first* match only. To return all matches, use XPATH_STRINGS instead.

## Input Parameters

**xml_expression**
Required. The name of a field of type Text or a literal string that contains a valid XML node (a snippet of XML consisting of a parent element and one or more child nodes).

**xpath_expression**
Required. An XPath expression that refers to a node, element, or attribute within the XML string passed to this expression. Any XPath expression that complies to the XML Path Language (XPath) Version 1.0 specification is valid.

## Examples

These example XPATH_STRING expressions assume you have a field in your dataset named *address* that contains XML-formatted strings such as this:

```
<list>
  <address type="work">
     <street1>1300 So. El Camino Real</street1>
     <street2>Suite 600</street2>
     <city>San Mateo</city>
     <state>CA</state>
     <zipcode>94403</zipcode>
  </address>
  <address type="home">
     <street1>123 Oakdale Street</street1>
     <street2/>
     <city>San Francisco</city>
     <state>CA</state>
     <zipcode>94123</zipcode>
  </address>
</list>
```

Get the *zipcode* value from any *address* element where the *type* attribute equals *home*:

```
XPATH_STRING(address,"//address[@type='home']/zipcode")
```

returns: 94123

Get the *city* value from the second *address* element:

```
XPATH_STRING(address,"/list/address[2]/city")
```

returns: San Francisco

Get the values from all child elements of the first address element (as one string):

```
XPATH_STRING(address,"/list/address")
```

returns: 1300 So. El Camino RealSuite 600 San MateoCA94403

### 1.12.12 | URL Functions

#### 1.12.12.1 | URL_AUTHORITY

## Description

URL_AUTHORITY is a row function that returns the authority portion of a URL string. The authority portion of a URL is the part that has the information on how to locate and connect to the server.

## Syntax

```
URL_AUTHORITY(URL_string)
```

## Return Value

Returns the authority portion of a URL as a Text value, or NULL if the input string is not a valid URL.

For example, in the string `http://www.workday.com/company/contact.html`, the authority portion is `www.workday.com`.

In the string `http://user:password@mycompany.com:8012/mypage.html`, the authority portion is `user:password@mycompany.com:8012`.

In the string `mailto:username@mycompany.com?subject=Topic`, the authority portion is `NULL`.

## Input Parameters

**URL_string**
Required. A field or expression that returns a `Text` value in URI (uniform resource identifier) format of: *`protocol`*`:`*`authority`*`[/`*`path`*`][?`*`query`*`][#`*`fragment`*`]`.

The authority portion of the URL contains the host information, which can be specified as a domain name ( `www.workday.com`), a host name ( `localhost`), or an IP address ( `127.0.0.1`). The host information can be preceeded by optional user information terminated with @ (for example, `username:password@workday.com`), and followed by an optional port number preceded by a colon (for example, `localhost:8001`).

## Examples

Return the authority portion of URL string values in the *referrer* field:

```
URL_AUTHORITY(referrer)
```

Return the authority portion of a literal URL string:

`URL_AUTHORITY("http://user:password@mycompany.com:8012/mypage.html")` returns `user:password@mycompany.com:8012`.

#### 1.12.12.2 | URL_FRAGMENT

## Description

`URL_FRAGMENT` is a row function that returns the fragment portion of a URL string.

## Syntax

```
URL_FRAGMENT(URL_string)
```

## Return Value

Returns the fragment portion of a URL as a `Text` value, `NULL` if the URL or does not contain a fragment, or `NULL` if the input string is not a valid URL.

For example, in the string `http://www.workday.com/contact.html#phone`, the fragment portion is `phone`.

In the string `http://www.workday.com/contact.html`, the fragment portion is `NULL`.

In the string `http://workday.com/news.php?topic=press#Workday%20News`, the fragment portion is `Workday%20News`.

## Input Parameters

**URL_string**
Required. A field or expression that returns a `Text` value in URI (uniform resource identifier) format of: *`protocol`*`:`*`authority`*`[/`*`path`*`][?`*`query`*`][#`*`fragment`*`]`.

The optional fragment portion of the URL is separated by a hash mark ( `#`) and provides direction to a secondary resource, such as a heading or anchor identifier.

## Examples

Return the fragment portion of URL string values in the *request* field:

```
URL_FRAGMENT(request)
```

Return the fragment portion of a literal URL string:

`URL_FRAGMENT("http://workday.com/news.php?topic=press#Workday%20News")` returns `Workday%20News`.

Return and decode the fragment portion of a literal URL string:

`URLDECODE(URL_FRAGMENT("http://workday.com/news.php?topic=press#Workday%20News"))` returns `Workday News`.

#### 1.12.12.3 | URL_HOST

## Description

`URL_HOST` is a row function that returns the host, domain, or IP address portion of a URL string.

## Syntax

```
URL_HOST(URL_string)
```

## Return Value

Returns the host portion of a URL as a `Text` value, or `NULL` if the input string is not a valid URL.

For example, in the string `http://www.workday.com/company/contact.html`, the host portion is `www.workday.com`.

In the string `http://admin:admin@127.0.0.1:8001/index.html`, the host portion is `127.0.0.1`.

In the string `mailto:username@mycompany.com?subject=Topic`, the host portion is `NULL`.

## Input Parameters

**URL_string**
Required. A field or expression that returns a `Text` value in URI (uniform resource identifier) format of: *`protocol:authority[/path][?query][#fragment]`*.

The authority portion of the URL contains the host information, which can be specified as a domain name ( `www.workday.com`), a host name ( `localhost`), or an IP address ( `127.0.0.1`).

## Examples

Return the host portion of URL string values in the *referrer* field:

```
URL_HOST(referrer)
```

Return the host portion of a literal URL string:

`URL_HOST("http://user:password@mycompany.com:8012/mypage.html")` returns `mycompany.com`.

#### 1.12.12.4 | URL_PATH

## Description

`URL_PATH` is a row function that returns the path portion of a URL string.

## Syntax

```
URL_PATH(URL_string)
```

## Return Value

Returns the path portion of a URL as a `Text` value, `NULL` if the URL or does not contain a path, or `NULL` if the input string is not a valid URL.

For example, in the string `http://www.workday.com/company/contact.html`, the path portion is `/company/contact.html`.

In the string `http://admin:admin@127.0.0.1:8001/index.html`, the path portion is `/index.html`.

In the string `mailto:username@mycompany.com?subject=Topic`, the path portion is `username@mycompany.com`.

## Input Parameters

**URL_string**
Required. A field or expression that returns a `Text` value in URI (uniform resource identifier) format of: *`protocol:authority[/path][?query][#fragment]`*.

The optional path portion of the URL is a sequence of resource location segments separated by a forward slash ( `/`), conceptually similar to a directory path.

## Examples

Return the path portion of URL string values in the *request* field:

```
URL_PATH(request)
```

Return the path portion of a literal URL string:

`URL_PATH("http://workday.com/company/contact.html")` returns `/company/contact.html`.

#### 1.12.12.5 | URL_PORT

## Description

`URL_PORT` is a row function that returns the port portion of a URL string.

## Syntax

```
URL_PORT(URL_string)
```

## Return Value

Returns the port portion of a URL as an `Integer` value. If the URL does not specify a port, then returns `-1`. If the input string is not a valid URL, returns `NULL`.

For example, in the string `http://localhost:8001`, the port portion is `8001`.

## Input Parameters

**URL_string**
Required. A field or expression that returns a `Text` value in URI (uniform resource identifier) format of: *`protocol:authority[/path][?query][#fragment]`*.

The authority portion of the URL contains the host information, which can be specified as a domain name ( `www.workday.com`), a host name ( `localhost`), or an IP address ( `127.0.0.1`). The host information can be followed by an optional port number preceded by a colon (for example, `localhost:8001`).

## Examples

Return the port portion of URL string values in the *referrer* field:

```
URL_PORT(referrer)
```

Return the port portion of a literal URL string:

`URL_PORT("http://user:password@mycompany.com:8012/mypage.html")` returns `8012`.

#### 1.12.12.6 | URL_PROTOCOL

## Description

`URL_PROTOCOL` is a row function that returns the protocol (or URI scheme name) portion of a URL string.

## Syntax

```
URL_PROTOCOL(URL_string)
```

## Return Value

Returns the protocol portion of a URL as a `Text` value, or `NULL` if the input string is not a valid URL.

For example, in the string `http://www.workday.com`, the protocol portion is `http`.

In the string `ftp://ftp.workday.com/articles/workday.pdf`, the protocol portion is `ftp`.

## Input Parameters

**URL_string**
Required. A field or expression that returns a `Text` value in URI (uniform resource identifier) format of: *`protocol`*`:`*`authority`*`[/`*`path`*`][?`*`query`*`][#`*`fragment`*`]`

The protocol portion of a URL consists of a sequence of characters beginning with a letter and followed by any combination of letter, number, plus ( `+`), period ( `.`), or hyphen ( `-`) characters, followed by a colon ( `:`). For example: `http:`, `ftp:`, `mailto:`

## Examples

Return the protocol portion of URL string values in the *referrer* field:

```
URL_PROTOCOL(referrer)
```

Return the protocol portion of the literal URL string:

`URL_PROTOCOL("http://www.workday.com")` returns `http`.

#### 1.12.12.7 | URL_QUERY

## Description

`URL_QUERY` is a row function that returns the query portion of a URL string.

## Syntax

```
URL_QUERY(URL_string)
```

## Return Value

Returns the query portion of a URL as a `Text` value, `NULL` if the URL or does not contain a query, or `NULL` if the input string is not a valid URL.

For example, in the string `http://www.workday.com/contact.html`, the query portion is `NULL`.

In the string `http://workday.com/news.php?topic=press&timeframe=today#Workday%20News`, the query portion is `topic=press&timeframe=today`.

In the string `mailto:username@mycompany.com?subject=Topic`, the query portion is `subject=Topic`.

## Input Parameters

**URL_string**
Required. A field or expression that returns a `Text` value in URI (uniform resource identifier) format of: *`protocol`*`:`*`authority`*`[/`*`path`*`][?`*`query`*`][#`*`fragment`*`]`.

The optional query portion of the URL is separated by a question mark ( `?`) and typically contains an unordered list of `key=value` pairs separated by an ampersand ( `&`) or semicolon ( `;`).

## Examples

Return the query portion of URL string values in the *request* field:

`URL_QUERY(request)`

Return the query portion of a literal URL string:

`URL_QUERY("http://workday.com/news.php?topic=press&timeframe=today")` returns `topic=press&timeframe=today`.

#### 1.12.12.8 | URLDECODE

## Description

`URLDECODE` is a row function that decodes a `Text` value that has been encoded with the `application/x-www-form-urlencoded` media type. URL encoding, also known as percent-encoding, is a mechanism for encoding information in a Uniform Resource Identifier (URI). When sent in an HTTP GET request, `application/x-www-form-urlencoded` data is included in the query component of the request URI. When sent in an HTTP POST request, the data is placed in the body of the message, and the name of the media type is included in the message `Content-Type` header.

## Syntax

`URLDECODE(URL_string)`

## Return Value

Returns a value of type `Text` with characters decoded as follows:

- Alphanumeric characters (a-z, A-Z, 0-9) remain unchanged.
- The special characters hyphen (`-`), comma (`,`), underscore (_), period (`.`), and asterisk (*) remain unchanged.
- The plus sign (+) character is converted to a space character.
- The percent character (`%`) is interpreted as the start of a special escaped sequence, where in the sequence *`%HH`*, *HH* represents the hexadecimal value of the byte. Some common escape sequences:

| Percent Encoding Sequence | Value |
|---|---|
| `%20` | space |
| `%0A` or `%0D` or `%0D%0A` | newline |
| `%22` | double quote (") |
| `%25` | percent (%) |
| `%2D` | hyphen (-) |
| `%2E` | period (.) |
| `%3C` | less than (<) |
| `%3D` | greater than (>) |
| `%5C` | backslash (\) |
| `%7C` | pipe (\|) |

## Input Parameters

**URL_string**
Required. A field or expression that returns a `Text` value. It is assumed that all characters in the input string are one of the following: lower-case letters (a-z), upper-case letters (A-Z), numeric digits (0-9), or the hyphen ( `-`), comma ( `,`), underscore ( _), period ( `.`) or asterisk ( `*`) character. The percent character ( `%`) is allowed, but is interpreted as the start of a special escaped sequence. The plus character ( +) is allowed, but is interpreted as a space character.

## Examples

Decode the values of the *url_query* field:

`URLDECODE(url_query)`

Convert a literal URL encoded string (`N%2FA%20or%20%22not%20applicable%22`) to a human-readable value:

`URLDECODE("N%2FA%20or%20%22not%20applicable%22")` returns `N/A or "not applicable".`

### 1.12.13 | Window Functions

#### 1.12.13.1 | AVG

## Description

`AVG` is a window aggregate function that partitions rows into groups, orders rows by a field, and returns the average of all valid numeric values in the group. It sums all values in the group and divides by the number of valid (NOT NULL) rows. You can use `AVG` to calculate moving averages.

The `PARTITION BY` clause determines which fields to use to partition a set of input rows into groups. The `ORDER BY` clause determines how to order the rows in the partition.

Workday separates the input rows into groups according to the partitioning fields, orders the rows according to the ordering fields, and then computes the aggregate expression (average for this function) in each group.

## Syntax

```
AVG(input_field) OVER(
  PARTITION BY partitioning_field [, partitioning_field]
  ORDER BY ordering_field [ASC | DESC] [, ordering_field [ASC | DESC]]
  RANGE BETWEEN value PRECEDING AND CURRENT ROW |
  ROWS win_boundary | BETWEEN win_boundary AND win_boundary
)

where win_boundary can be:

  UNBOUNDED PRECEDING
  value PRECEDING
  UNBOUNDED FOLLOWING
  value FOLLOWING
  CURRENT ROW
```

## Return Value

Returns a value of type `Numeric` or `Double` depending on the type of *input_field*.

## Input Parameters

***input_field***
Required. The field on which to perform the aggregate function. You can use any numeric field or a Currency field.

**OVER()**
Required. `OVER` must be used within an `AVG` expression.

**PARTITION BY *partitioning_field***
Required. Use the `PARTITION BY` clause to specify 1 or more fields to use to partition a group of input rows. You can specify any field type except Currency. Example: You specify the Month field as the partitioning field, so Workday groups into a single partition all records that have the same value for Month.

**ORDER BY *ordering_field***
Required. Use the `ORDER BY` clause to specify how to order the input rows in the partition using the values in the specified field within each partition. You can specify any field type except Currency. However, you must use a numeric field type, such as Integer or Numeric when you use the `RANGE` clause.

You can use the `DESC` or `ASC` keywords to sort in descending order (high to low values, NULLs are last) or ascending order (low to high values, NULLs are first) for each ordering field. If you don't specify a sort order for an ordering field, Workday automatically sorts rows in ascending order.

**ROWS | RANGE**

Required. The `ROWS` and `RANGE` clauses define the specific number of rows (relative to the current row) within the partition by specifying a window frame. You define the window frame by specifying start and end points within the partition, known as window boundaries. The window frame is the set of input rows in each partition over which to calculate the aggregate expression (average for this function). The window frame can include one, several, or all rows of the partition.

Both `ROWS` and `RANGE` specify the range of rows relative to the current row, but `RANGE` operates logically on values (logical association) and `ROWS` operates physically on rows in the dataset (physical association).

`RANGE` limits the window frame to contain rows that have their values within the specified range, relative to the current value. `ROWS` limits the window frame to contain rows that are physically next to the current row.

Use `RANGE` to define absolute window boundaries, such as the past 3 months or year to date. When you use `RANGE`, the `ORDER BY` clause must use a numeric field type, such as Integer or Numeric.

Example: Suppose you have an Integer field called MonthNum that represents the number of the month in the year (values 1 to 12). To specify all values from the past 3 months, you would order by MonthNum and use `RANGE BETWEEN 2 PRECEDING AND CURRENT ROW`. This RANGE clause includes the current month and the previous 2 months, resulting in 3 months total.

Note: When you publish a dataset that contains a window function using `RANGE`, the number of rows in the window must be 1000 or less. If a particular window exceeds 1000 rows, the publish job fails.

***win_boundary***

Required. The window boundaries define the start and end points of the window frame. Window boundaries are relative to the current row.

A `PRECEDING` clause defines a window boundary that is lower than the current row (the number of rows to include before the current row). The `FOLLOWING` clause defines a window boundary that is greater than the current row (the number of rows to include after the current row).

If you specify only 1 window boundary, then Workday uses the current row as the other boundary in the window frame (either the upper or lower boundary depending on the expression syntax). The `UNBOUNDED` keyword includes all rows in the direction specified. When you need to specify both a start and end of a window frame, use the `BETWEEN` and `AND` keywords.

When specifying a specific number of rows, the *value* must be 100 or less.

Example: `ROWS 2 PRECEDING` means that the window is 3 rows in size, starting with 2 rows preceding until and including the current row.

Example: `ROWS UNBOUNDED FOLLOWING` means that the window starts with the current row and includes the current row and all rows that come after the current row.

## Examples

You can calculate the average sales for each employee:

```
AVG([Sales]) OVER(PARTITION BY [Employee] ORDER BY [SalesDate] DESC ROWS UNBOUNDED PRECEDING)
```

You can calculate the rolling 12 month average:

```
AVG([fieldA]) OVER(PARTITION BY [fieldB] ORDER BY [Month] RANGE 11 PRECEDING)
```

The `Month` field must be a numeric field type, such as Integer or Numeric.

You can calculate the previous year to date average:

```
AVG([fieldA]) OVER(PARTITION BY [fieldB] ORDER BY [Year] RANGE 1 PRECEDING)
```

The `Year` field must be a numeric field type, such as Integer or Numeric.

#### 1.12.13.2 | COUNT

## Description

`COUNT` is a window aggregate function that partitions rows into groups, orders rows by a field, and returns the total number of valid rows (NOT NULL) in the group. You can use `COUNT` together with other functions to calculate cumulative aggregates.

The `PARTITION BY` clause determines which fields to use to partition a set of input rows into groups. The `ORDER BY` clause determines how to order the rows in the partition.

Workday separates the input rows into groups according to the partitioning fields, orders the rows according to the ordering fields, and then computes the aggregate expression (count for this function) in each group.

## Syntax

```
COUNT(input_field) OVER(
  PARTITION BY partitioning_field [, partitioning_field]
  ORDER BY ordering_field [ASC | DESC] [, ordering_field [ASC | DESC]]
  RANGE BETWEEN value PRECEDING AND CURRENT ROW |
  ROWS win_boundary | BETWEEN win_boundary AND win_boundary
)

where win_boundary can be:

  UNBOUNDED PRECEDING
  value PRECEDING
  UNBOUNDED FOLLOWING
  value FOLLOWING
  CURRENT ROW
```

## Return Value

Returns a value of type `Long`.

## Input Parameters

***input_field***
Required. The field on which to perform the aggregate function. You can use any numeric field or a Currency field.

**OVER()**
Required. `OVER` must be used within an `COUNT` expression.

**PARTITION BY *partitioning_field***
Required. Use the `PARTITION BY` clause to specify 1 or more fields to use to partition a group of input rows. You can specify any field type except Currency. Example: You specify the Month field as the partitioning field, so Workday groups into a single partition all records that have the same value for Month.

**ORDER BY *ordering_field***
Required. Use the `ORDER BY` clause to specify how to order the input rows in the partition using the values in the specified field within each partition. You can specify any field type except Currency. However, you must use a numeric field type, such as Integer or Numeric when you use the `RANGE` clause.

You can use the `DESC` or `ASC` keywords to sort in descending order (high to low values, NULLs are last) or ascending order (low to high values, NULLs are first) for each ordering field. If you don't specify a sort order for an ordering field, Workday automatically sorts rows in ascending order.

**ROWS | RANGE**

Required. The ROWS and RANGE clauses define the specific number of rows (relative to the current row) within the partition by specifying a window frame. You define the window frame by specifying start and end points within the partition, known as window boundaries. The window frame is the set of input rows in each partition over which to calculate the aggregate expression (count for this function). The window frame can include one, several, or all rows of the partition.

Both ROWS and RANGE specify the range of rows relative to the current row, but RANGE operates logically on values (logical association) and ROWS operates physically on rows in the dataset (physical association).

RANGE limits the window frame to contain rows that have their values within the specified range, relative to the current value. ROWS limits the window frame to contain rows that are physically next to the current row.

Use RANGE to define absolute window boundaries, such as the past 3 months or year to date. When you use RANGE, the ORDER BY clause must use a numeric field type, such as Integer or Numeric.

Example: Suppose you have an Integer field called MonthNum that represents the number of the month in the year (values 1 to 12). To specify all values from the past 3 months, you would order by MonthNum and use RANGE BETWEEN 2 PRECEDING AND CURRENT ROW. This RANGE clause includes the current month and the previous 2 months, resulting in 3 months total.

Note: When you publish a dataset that contains a window function using RANGE, the number of rows in the window must be 1000 or less. If a particular window exceeds 1000 rows, the publish job fails.

***win_boundary***

Required. The window boundaries define the start and end points of the window frame. Window boundaries are relative to the current row.

A PRECEDING clause defines a window boundary that is lower than the current row (the number of rows to include before the current row). The FOLLOWING clause defines a window boundary that is greater than the current row (the number of rows to include after the current row).

If you specify only 1 window boundary, then Workday uses the current row as the other boundary in the window frame (either the upper or lower boundary depending on the expression syntax). The UNBOUNDED keyword includes all rows in the direction specified. When you need to specify both a start and end of a window frame, use the BETWEEN and AND keywords.

When specifying a specific number of rows, the *value* must be 100 or less.

Example: ROWS 2 PRECEDING means that the window is 3 rows in size, starting with 2 rows preceding until and including the current row.

Example: ROWS UNBOUNDED FOLLOWING means that the window starts with the current row and includes the current row and all rows that come after the current row.

## Examples

You can count the sales for each employee:

```
COUNT([Sales]) OVER(PARTITION BY [Employee] ORDER BY [SalesDate] DESC ROWS UNBOUNDED PRECEDING)
```

You can calculate the rolling 12 month count:

```
COUNT([fieldA]) OVER(PARTITION BY [fieldB] ORDER BY [Month] RANGE 11 PRECEDING)
```

The Month field must be a numeric field type, such as Integer or Numeric.

You can calculate the previous year to date count:

```
COUNT([fieldA]) OVER(PARTITION BY [fieldB] ORDER BY [Year] RANGE 1 PRECEDING)
```

The Year field must be a numeric field type, such as Integer or Numeric.

#### 1.12.13.3 | FIRST

## Description

FIRST is a window aggregate function that partitions rows into groups, orders rows by a field, and returns the value from the first row in the group.

The PARTITION BY clause determines which fields to use to partition a set of input rows into groups. The ORDER BY clause determines how to order the rows in the partition.

Workday separates the input rows into groups according to the partitioning fields, orders the rows according to the ordering fields, and then computes the aggregate expression (first for this function) in each group.

## Syntax

```
FIRST(input_field) OVER(
   PARTITION BY partitioning_field [, partitioning_field]
   ORDER BY ordering_field [ASC | DESC] [, ordering_field [ASC | DESC]]
   ROWS start_window_boundary
   )
```

where *start_window_boundary* can be:

```
  UNBOUNDED PRECEDING
```

## Return Value

Returns a value of the same type as *input_field*.

## Input Parameters

***input_field***
Required. The field on which to perform the aggregate function. You can use any numeric field or a Currency field.

**OVER()**
Required. `OVER` must be used within a `FIRST` expression.

**PARTITION BY *partitioning_field***
Required. Use the `PARTITION BY` clause to specify 1 or more fields to use to partition a group of input rows. You can specify any field type except Currency. Example: You specify the Month field as the partitioning field, so Workday groups into a single partition all records that have the same value for Month.

**ORDER BY *ordering_field***
Required. Use the `ORDER BY` clause to specify how to order the input rows in the partition using the values in the specified field within each partition. You can specify any field type except Currency.

You can use the `DESC` or `ASC` keywords to sort in descending order (high to low values, NULLs are last) or ascending order (low to high values, NULLs are first) for each ordering field. If you don't specify a sort order for an ordering field, Workday automatically sorts rows in ascending order.

**ROWS**

Required. The `ROWS` clause defines the specific number of rows (relative to the current row) within the partition by specifying a window frame. You define the window frame by specifying start and end points within the partition, known as window boundaries. The window frame is the set of input rows in each partition, relative to the current row, over which to calculate the aggregate expression (first for this function). The window frame can include one, several, or all rows of the partition.

***window_boundary***

Required. The window boundaries define the start and end points of the window frame. Window boundaries are relative to the current row.

A `PRECEDING` clause defines a window boundary that is lower than the current row (the number of rows to include before the current row). If you specify only 1 window boundary, then Workday uses the current row as the last row in the window frame (the upper boundary). The `UNBOUNDED` keyword includes all rows in the direction specified.

#### 1.12.13.4 | LAG

## Description

`LAG` is a window aggregate function that partitions rows into groups, orders rows by a field, and returns the value of a field in the row at the specified offset before (above) the current row in the group.

The `PARTITION BY` clause determines which fields to use to partition a set of input rows into groups. The `ORDER BY` clause determines how to order the rows in the partition.

Workday separates the input rows into groups according to the partitioning fields, orders the rows according to the ordering fields, and then computes the aggregate expression (lag for this function) in each group.

## Syntax

```
LAG(input_field, offset, default_value) OVER(
   PARTITION BY partitioning_field [, partitioning_field]
   ORDER BY ordering_field [ASC | DESC] [, ordering_field [ASC | DESC]]
   )
```

## Return Value

Returns one value per row of the same type as the *input_field*.

## Input Parameters

***input_field***
Required. The field on which to perform the aggregate function. You can specify any field type.

***offset***
Optional. The number of rows before the current row whose value to return. Must be a literal number greater than or equal to zero (0) and less than or equal to 100. If you don't specify the offset, Workday uses the value of 1.

***default_value***
Optional. The value this function returns when the offset row is outside the currently defined window or when the value in the offset row is NULL. *default_value* must be the same type as *input_field*. If you don't specify a default value, Workday uses the value of NULL.

**OVER()**
Required. `OVER` must be used within a `LAG` expression.

**PARTITION BY *partitioning_field***
Required. Use the `PARTITION BY` clause to specify 1 or more fields to use to partition a group of input rows. You can specify any field type except Currency. Example: You specify the Month field as the partitioning field, so Workday groups into a single partition all records that have the same value for Month.

**ORDER BY *ordering_field***

Required. Use the ORDER BY clause to specify how to order the input rows in the partition using the values in the specified field within each partition. You can specify any field type except Currency.

You can use the DESC or ASC keywords to sort in descending order (high to low values, NULLs are last) or ascending order (low to high values, NULLs are first) for each ordering field. If you don't specify a sort order for an ordering field, Workday automatically sorts rows in ascending order.

## Examples

Example: You have a dataset with these rows and fields.

| row_ID | Employee_Name | Eff_Date | Salary |
|---|---|---|---|
| 1 | Goh | 1/1/18 | 49000 |
| 2 | Goh | 1/1/19 | 56000 |
| 3 | Goh | 1/1/17 | 44000 |
| 4 | Freeman | 1/1/18 | 65000 |
| 5 | Freeman | 1/1/17 | 57000 |
| 6 | Freeman | 1/1/19 | 69000 |
| 7 | Smith | 1/1/18 | 51000 |
| 8 | Smith | 1/1/19 | 56000 |
| 9 | Smith | 1/1/16 | 44000 |

You can order the rows for each employee in ascending (ASC) order by the effective date ([Eff_Date]) field, so the most recent salary comes first in each partition.

Use this expression in the [Salary_Increase] field to calculate the change in salary between each change in effective date:

```
[Salary] - (LAG([Salary], 1, [Salary]) OVER(
   PARTITION BY [Employee_Name]
   ORDER BY [Eff_Date] ASC)
   )
```

You get these results:

| row_ID | Employee_Name | Eff_Date | Salary | Salary_Increase |
|---|---|---|---|---|
| 2 | Goh | 1/1/19 | 56000 | 7000 |
| 1 | Goh | 1/1/18 | 49000 | 5000 |
| 3 | Goh | 1/1/17 | 44000 | 0 |
| 6 | Freeman | 1/1/19 | 69000 | 4000 |
| 4 | Freeman | 1/1/18 | 65000 | 8000 |
| 5 | Freeman | 1/1/17 | 57000 | 0 |
| 8 | Smith | 1/1/19 | 56000 | 5000 |
| 7 | Smith | 1/1/18 | 51000 | 7000 |
| 9 | Smith | 1/1/16 | 44000 | 0 |

### Related Information

**Concepts**

Workday 32 What's New Post: Prism Analytics Data Preparation

#### 1.12.13.5 | LAST

## Description

LAST is a window aggregate function that partitions rows into groups, orders rows by a field, and returns the value from the last row in the group.

The PARTITION BY clause determines which fields to use to partition a set of input rows into groups. The ORDER BY clause determines how to order the rows in the partition.

Workday separates the input rows into groups according to the partitioning fields, orders the rows according to the ordering fields, and then computes the aggregate expression (last for this function) in each group.

## Syntax

```
LAST(input_field) OVER(
   PARTITION BY partitioning_field [, partitioning_field]
   ORDER BY ordering_field [ASC | DESC] [, ordering_field [ASC | DESC]]
   ROWS BETWEEN start_window_boundary AND end_window_boundary
   )
```

where *start_window_boundary* can be:

```
  CURRENT ROW
```

and where *end_window_boundary* can be:

```
  UNBOUNDED FOLLOWING
```

## Return Value

Returns a value of the same type as *input_field*.

## Input Parameters

***input_field***
Required. The field on which to perform the aggregate function. You can use any numeric field or a Currency field.

**OVER()**
Required. `OVER` must be used within a `LAST` expression.

**PARTITION BY *partitioning_field***
Required. Use the `PARTITION BY` clause to specify 1 or more fields to use to partition a group of input rows. You can specify any field type except Currency. Example: You specify the Month field as the partitioning field, so Workday groups into a single partition all records that have the same value for Month.

**ORDER BY *ordering_field***
Required. Use the `ORDER BY` clause to specify how to order the input rows in the partition using the values in the specified field within each partition. You can specify any field type except Currency.

You can use the `DESC` or `ASC` keywords to sort in descending order (high to low values, NULLs are last) or ascending order (low to high values, NULLs are first) for each ordering field. If you don't specify a sort order for an ordering field, Workday automatically sorts rows in ascending order.

**ROWS**

Required. The `ROWS` clause defines the specific number of rows (relative to the current row) within the partition by specifying a window frame. You define the window frame by specifying start and end points within the partition, known as window boundaries. The window frame is the set of input rows in each partition, relative to the current row, over which to calculate the aggregate expression (last for this function). The window frame can include one, several, or all rows of the partition.

***window_boundary***

Required. The window boundaries define the start and end points of the window frame. Window boundaries are relative to the current row.

A `FOLLOWING` clause defines a window boundary that is after the current row (the number of rows to include after the current row). Workday uses the current row as the first row in the window frame (the lower boundary). The `UNBOUNDED` keyword includes all rows in the direction specified.

#### 1.12.13.6 | LEAD

## Description

`LEAD` is a window aggregate function that partitions rows into groups, orders rows by a field, and returns the value of a field in the row at the specified offset after (below) the current row in the group.

The `PARTITION BY` clause determines which fields to use to partition a set of input rows into groups. The `ORDER BY` clause determines how to order the rows in the partition.

Workday separates the input rows into groups according to the partitioning fields, orders the rows according to the ordering fields, and then computes the aggregate expression (lead for this function) in each group.

## Syntax

```
LEAD(input_field, offset, default_value) OVER(
   PARTITION BY partitioning_field [, partitioning_field]
   ORDER BY ordering_field [ASC | DESC] [, ordering_field [ASC | DESC]]
   )
```

## Return Value

Returns one value per row of the same type as the *input_field*.

## Input Parameters

***input_field***
Required. The field on which to perform the aggregate function. You can specify any field type.

***offset***
Optional. The number of rows after the current row whose value to return. Must be a literal number greater than or equal to zero (0) and less than or equal to 100. If you don't specify the offset, Workday uses the value of 1.

***default_value***
Optional. The value this function returns when the offset row is outside the currently defined window or when the value in the offset row is NULL. *default_value* must be the same type as *input_field*. If you don't specify a default value, Workday uses the value of NULL.

**OVER()**
Required. `OVER` must be used within a `LEAD` expression.

**PARTITION BY *partitioning_field***
Required. Use the `PARTITION BY` clause to specify 1 or more fields to use to partition a group of input rows. You can specify any field type except Currency. Example: You specify the Month field as the partitioning field, so Workday groups into a single partition all records that have the same value for Month.

**ORDER BY *ordering_field***
Required. Use the `ORDER BY` clause to specify how to order the input rows in the partition using the values in the specified field within each partition. You can specify any field type except Currency.

You can use the `DESC` or `ASC` keywords to sort in descending order (high to low values, NULLs are last) or ascending order (low to high values, NULLs are first) for each ordering field. If you don't specify a sort order for an ordering field, Workday automatically sorts rows in ascending order.

## Examples

Example: You have a dataset with these rows and fields.

| row_ID | Employee_Name | Eff_Date | Salary |
|---|---|---|---|
| 1 | Goh | 1/1/18 | 49000 |
| 2 | Goh | 1/1/19 | 56000 |
| 3 | Goh | 1/1/17 | 44000 |
| 4 | Freeman | 1/1/18 | 65000 |
| 5 | Freeman | 1/1/17 | 57000 |
| 6 | Freeman | 1/1/19 | 69000 |
| 7 | Smith | 1/1/18 | 51000 |
| 8 | Smith | 1/1/19 | 56000 |
| 9 | Smith | 1/1/16 | 44000 |

You can order the rows for each employee in descending (DESC) order by the effective date ([Eff_Date]) field, so the most recent salary comes first in each partition.

Use this expression in the [Salary_Increase] field to calculate the change in salary between each change in effective date:

```
[Salary] - (LEAD([Salary], 1, [Salary]) OVER(
    PARTITION BY [Employee_Name]
    ORDER BY [Eff_Date] DESC)
    )
```

You get these results:

| row_ID | Employee_Name | Eff_Date | Salary | Salary_Increase |
|---|---|---|---|---|
| 2 | Goh | 1/1/19 | 56000 | 7000 |
| 1 | Goh | 1/1/18 | 49000 | 5000 |
| 3 | Goh | 1/1/17 | 44000 | 0 |
| 6 | Freeman | 1/1/19 | 69000 | 4000 |
| 4 | Freeman | 1/1/18 | 65000 | 8000 |
| 5 | Freeman | 1/1/17 | 57000 | 0 |
| 8 | Smith | 1/1/19 | 56000 | 5000 |
| 7 | Smith | 1/1/18 | 51000 | 7000 |
| 9 | Smith | 1/1/16 | 44000 | 0 |

### Related Information

**Concepts**

Workday 32 What's New Post: Prism Analytics Data Preparation

**1.12.13.7 | MAX**

## Description

MAX is a window aggregate function that partitions rows into groups, orders rows by a field, and returns the maximum (highest) value in the group.

The PARTITION BY clause determines which fields to use to partition a set of input rows into groups. The ORDER BY clause determines how to order the rows in the partition.

Workday separates the input rows into groups according to the partitioning fields, orders the rows according to the ordering fields, and then computes the aggregate expression (maximum for this function) in each group.

## Syntax

```
MAX(input_field) OVER(
  PARTITION BY partitioning_field [, partitioning_field]
  ORDER BY ordering_field [ASC | DESC] [, ordering_field [ASC | DESC]]
  RANGE BETWEEN value PRECEDING AND CURRENT ROW |
  ROWS win_boundary | BETWEEN win_boundary AND win_boundary
)

where win_boundary can be:

  UNBOUNDED PRECEDING
  value PRECEDING
  UNBOUNDED FOLLOWING
  value FOLLOWING
  CURRENT ROW
```

## Return Value

Returns a value of type Numeric or Double depending on the type of *input_field*.

## Input Parameters

***input_field***
Required. The field on which to perform the aggregate function. You can use any numeric field or a Currency field.

**OVER()**
Required. OVER must be used within a MAX expression.

**PARTITION BY *partitioning_field***
Required. Use the PARTITION BY clause to specify 1 or more fields to use to partition a group of input rows. You can specify any field type except Currency. Example: You specify the Month field as the partitioning field, so Workday groups into a single partition all records that have the same value for Month.

**ORDER BY *ordering_field***
Required. Use the ORDER BY clause to specify how to order the input rows in the partition using the values in the specified field within each partition. You can specify any field type except Currency. However, you must use a numeric field type, such as Integer or Numeric when you use the RANGE clause.

You can use the DESC or ASC keywords to sort in descending order (high to low values, NULLs are last) or ascending order (low to high values, NULLs are first) for each ordering field. If you don't specify a sort order for an ordering field, Workday automatically sorts rows in ascending order.

**ROWS | RANGE**

Required. The ROWS and RANGE clauses define the specific number of rows (relative to the current row) within the partition by specifying a window frame. You define the window frame by specifying start and end points within the partition, known as window boundaries. The window frame is the set of input rows in each partition over which to calculate the aggregate expression (maximum for this function). The window frame can include one, several, or all rows of the partition.

Both ROWS and RANGE specify the range of rows relative to the current row, but RANGE operates logically on values (logical association) and ROWS operates physically on rows in the dataset (physical association).

RANGE limits the window frame to contain rows that have their values within the specified range, relative to the current value. ROWS limits the window frame to contain rows that are physically next to the current row.

Use RANGE to define absolute window boundaries, such as the past 3 months or year to date. When you use RANGE, the ORDER BY clause must use a numeric field type, such as Integer or Numeric.

Example: Suppose you have an Integer field called MonthNum that represents the number of the month in the year (values 1 to 12). To specify all values from the past 3 months, you would order by MonthNum and use RANGE BETWEEN 2 PRECEDING AND CURRENT ROW. This RANGE clause includes the current month and the previous 2 months, resulting in 3 months total.

Note: When you publish a dataset that contains a window function using RANGE, the number of rows in the window must be 1000 or less. If a particular window exceeds 1000 rows, the publish job fails.

***win_boundary***

Required. The window boundaries define the start and end points of the window frame. Window boundaries are relative to the current row.

A `PRECEDING` clause defines a window boundary that is lower than the current row (the number of rows to include before the current row). The `FOLLOWING` clause defines a window boundary that is greater than the current row (the number of rows to include after the current row).

If you specify only 1 window boundary, then Workday uses the current row as the other boundary in the window frame (either the upper or lower boundary depending on the expression syntax). The `UNBOUNDED` keyword includes all rows in the direction specified. When you need to specify both a start and end of a window frame, use the `BETWEEN` and `AND` keywords.

When specifying a specific number of rows, the *value* must be 100 or less.

Example: `ROWS 2 PRECEDING` means that the window is 3 rows in size, starting with 2 rows preceding until and including the current row.

Example: `ROWS UNBOUNDED FOLLOWING` means that the window starts with the current row and includes the current row and all rows that come after the current row.

## Examples

Example: You have a dataset with these rows and fields.

| Supervisory Org | Quarter | Name | Comp Change |
|---|---|---|---|
| Marketing | 2019-Q1 | Goh | 2000.00 |
| Marketing | 2019-Q1 | Freeman | 1000.00 |
| Marketing | 2019-Q1 | Smith | 2500.00 |
| Consulting | 2019-Q1 | Gomez | 5000.00 |
| Consulting | 2019-Q1 | Kimura | 3000.00 |
| Consulting | 2019-Q1 | Fitz | 3500.00 |
| Consulting | 2019-Q2 | Gomez | 0 |
| Consulting | 2019-Q2 | Kimura | 2000.00 |
| Consulting | 2019-Q2 | Fitz | 1500.00 |

You can calculate the highest change in compensation ([Comp Change] field) for each supervisory org in each quarter.

To ensure that Workday returns the same value for every row in a partition, order the rows in descending (DESC) order by the same field as the input field so the highest compensation change comes first in each partition.

Use this expression in the [Max Comp Change] field:

```
MAX([Comp Change]) OVER(
   PARTITION BY [Supervisory Org], [Quarter]
   ORDER BY [Comp Change] DESC ROWS UNBOUNDED PRECEDING
   )
```

You get these results:

| Supervisory Org | Quarter | Name | Comp Change | Max Comp Change |
|---|---|---|---|---|
| Consulting | 2019-Q1 | Gomez | 5000.00 | 5000.00 |
| Consulting | 2019-Q1 | Fitz | 3500.00 | 5000.00 |
| Consulting | 2019-Q1 | Kimura | 3000.00 | 5000.00 |
| Consulting | 2019-Q2 | Kimura | 2000.00 | 2000.00 |
| Consulting | 2019-Q2 | Fitz | 1500.00 | 2000.00 |
| Consulting | 2019-Q2 | Gomez | 0 | 2000.00 |
| Marketing | 2019-Q1 | Smith | 2500.00 | 2500.00 |
| Marketing | 2019-Q1 | Goh | 2000.00 | 2500.00 |
| Marketing | 2019-Q1 | Freeman | 1000.00 | 2500.00 |

### Related Information

**Concepts**

Workday 32 What's New Post: Prism Analytics Dataset Window Functions

**1.12.13.8 | MIN**

## Description

MIN is a window aggregate function that partitions rows into groups, orders rows by a field, and returns the minimum (lowest) value in the group.

The PARTITION BY clause determines which fields to use to partition a set of input rows into groups. The ORDER BY clause determines how to order the rows in the partition.

Workday separates the input rows into groups according to the partitioning fields, orders the rows according to the ordering fields, and then computes the aggregate expression (minimum for this function) in each group.

## Syntax

```
MIN(input_field) OVER(
  PARTITION BY partitioning_field [, partitioning_field]
  ORDER BY ordering_field [ASC | DESC] [, ordering_field [ASC | DESC]]
  RANGE BETWEEN value PRECEDING AND CURRENT ROW |
  ROWS win_boundary | BETWEEN win_boundary AND win_boundary
)

where win_boundary can be:

  UNBOUNDED PRECEDING
  value PRECEDING
  UNBOUNDED FOLLOWING
  value FOLLOWING
  CURRENT ROW
```

## Return Value

Returns a value of type Numeric or Double depending on the type of *input_field*.

## Input Parameters

***input_field***
Required. The field on which to perform the aggregate function. You can use any numeric field or a Currency field.

**OVER()**
Required. OVER must be used within a MIN expression.

**PARTITION BY *partitioning_field***
Required. Use the PARTITION BY clause to specify 1 or more fields to use to partition a group of input rows. You can specify any field type except Currency. Example: You specify the Month field as the partitioning field, so Workday groups into a single partition all records that have the same value for Month.

**ORDER BY *ordering_field***
Required. Use the ORDER BY clause to specify how to order the input rows in the partition using the values in the specified field within each partition. You can specify any field type except Currency. However, you must use a numeric field type, such as Integer or Numeric when you use the RANGE clause.

You can use the DESC or ASC keywords to sort in descending order (high to low values, NULLs are last) or ascending order (low to high values, NULLs are first) for each ordering field. If you don't specify a sort order for an ordering field, Workday automatically sorts rows in ascending order.

**ROWS | RANGE**

Required. The ROWS and RANGE clauses define the specific number of rows (relative to the current row) within the partition by specifying a window frame. You define the window frame by specifying start and end points within the partition, known as window boundaries. The window frame is the set of input rows in each partition over which to calculate the aggregate expression (minimum for this function). The window frame can include one, several, or all rows of the partition.

Both ROWS and RANGE specify the range of rows relative to the current row, but RANGE operates logically on values (logical association) and ROWS operates physically on rows in the dataset (physical association).

RANGE limits the window frame to contain rows that have their values within the specified range, relative to the current value. ROWS limits the window frame to contain rows that are physically next to the current row.

Use RANGE to define absolute window boundaries, such as the past 3 months or year to date. When you use RANGE, the ORDER BY clause must use a numeric field type, such as Integer or Numeric.

Example: Suppose you have an Integer field called MonthNum that represents the number of the month in the year (values 1 to 12). To specify all values from the past 3 months, you would order by MonthNum and use RANGE BETWEEN 2 PRECEDING AND CURRENT ROW. This RANGE clause includes the current month and the previous 2 months, resulting in 3 months total.

Note: When you publish a dataset that contains a window function using RANGE, the number of rows in the window must be 1000 or less. If a particular window exceeds 1000 rows, the publish job fails.

***win_boundary***

Required. The window boundaries define the start and end points of the window frame. Window boundaries are relative to the current row.

A `PRECEDING` clause defines a window boundary that is lower than the current row (the number of rows to include before the current row). The `FOLLOWING` clause defines a window boundary that is greater than the current row (the number of rows to include after the current row).

If you specify only 1 window boundary, then Workday uses the current row as the other boundary in the window frame (either the upper or lower boundary depending on the expression syntax). The `UNBOUNDED` keyword includes all rows in the direction specified. When you need to specify both a start and end of a window frame, use the `BETWEEN` and `AND` keywords.

When specifying a specific number of rows, the *value* must be 100 or less.

Example: `ROWS 2 PRECEDING` means that the window is 3 rows in size, starting with 2 rows preceding until and including the current row.

Example: `ROWS UNBOUNDED FOLLOWING` means that the window starts with the current row and includes the current row and all rows that come after the current row.

## Examples

Example: You have a dataset with these rows and fields.

| Supervisory Org | Quarter | Name | Comp Change |
|---|---|---|---|
| Marketing | 2019-Q1 | Goh | 2000.00 |
| Marketing | 2019-Q1 | Freeman | 1000.00 |
| Marketing | 2019-Q1 | Smith | 2500.00 |
| Consulting | 2019-Q1 | Gomez | 5000.00 |
| Consulting | 2019-Q1 | Kimura | 3000.00 |
| Consulting | 2019-Q1 | Fitz | 3500.00 |
| Consulting | 2019-Q2 | Gomez | 0 |
| Consulting | 2019-Q2 | Kimura | 2000.00 |
| Consulting | 2019-Q2 | Fitz | 1500.00 |

You can calculate the lowest change in compensation ([Comp Change] field) for each supervisory org in each quarter.

To ensure that Workday returns the same value for every row in a partition, order the rows in ascending (ASC) order by the same field as the input field so the lowest compensation change comes first in each partition.

Use this expression in the [Min Comp Change] field:

```
MIN([Comp Change]) OVER(
   PARTITION BY [Supervisory Org], [Quarter]
   ORDER BY [Comp Change] ASC ROWS UNBOUNDED PRECEDING
   )
```

You get these results:

| Supervisory Org | Quarter | Name | Comp Change | Min Comp Change |
|---|---|---|---|---|
| Consulting | 2019-Q1 | Kimura | 3000.00 | 3000.00 |
| Consulting | 2019-Q1 | Fitz | 3500.00 | 3000.00 |
| Consulting | 2019-Q1 | Gomez | 5000.00 | 3000.00 |
| Consulting | 2019-Q2 | Gomez | 0 | 0 |
| Consulting | 2019-Q2 | Fitz | 1500.00 | 0 |
| Consulting | 2019-Q2 | Kimura | 2000.00 | 0 |
| Marketing | 2019-Q1 | Freeman | 1000.00 | 1000.00 |
| Marketing | 2019-Q1 | Goh | 2000.00 | 1000.00 |
| Marketing | 2019-Q1 | Smith | 2500.00 | 1000.00 |

### Related Information

**Concepts**

Workday 32 What's New Post: Prism Analytics Dataset Window Functions

**1.12.13.9 | RANK**

## Description

RANK is a window aggregate function used to assign a ranking number to each row in a group. If multiple rows have the same ranking value (there's a tie), then Workday assigns the same rank value to the tied rows and skips the subsequent rank position.

The PARTITION BY clause determines which fields to use to partition a set of input rows into groups.

The ORDER BY clause determines how to order the rows in the partition before they're ranked.

Workday separates the input rows into groups according to the partitioning fields, orders the rows according to the ordering fields, and then computes the aggregate expression (rank for this function) in each group. The ranked rows in each group start at 1.

## Syntax

```
RANK() OVER(
   PARTITION BY partitioning_field [, partitioning_field]
   ORDER BY ordering_field [ASC | DESC] [, ordering_field [ASC | DESC]]
   )
```

## Return Value

Returns a value of type Integer.

## Input Parameters

**OVER()**
Required. OVER must be used within a RANK expression.

**PARTITION BY *partitioning_field***
Required. Use the PARTITION BY clause to specify 1 or more fields to use to partition a group of input rows. You can specify any field type except Currency. Example: You specify the Month field as the partitioning field, so Workday groups together into a single partition all records that have the same value for Month.

**ORDER BY *ordering_field***
Required. Use the ORDER BY clause to specify how to order the input rows in the partition using the values in the specified field within each partition. You can specify any field type except Currency.

You can use the DESC or ASC keywords to sort in descending order (high to low values, NULLs are last) or ascending order (low to high values, NULLs are first) for each ordering field. If you don't specify a sort order for an ordering field, Workday automatically sorts rows in ascending order.

## Examples

Example: You have a dataset with these rows and fields.

| Employee | Sales Date | Sales |
|---|---|---|
| Goh | 12/31/2018 | 140 |
| Goh | 11/30/2018 | 60 |
| Goh | 10/31/2018 | 140 |
| Freeman | 12/31/2018 | 160 |
| Freeman | 11/30/2018 | 60 |
| Freeman | 10/31/2018 | 110 |
| Smith | 12/31/2018 | 140 |
| Smith | 11/30/2018 | 60 |
| Smith | 10/31/2018 | 120 |

You can rank the sales for each employee in descending order, so the highest sales is given the ranking of 1. Use this expression in the Rank Sales by Employee field:

```
RANK() OVER(PARTITION BY [Employee] ORDER BY [Sales] DESC)
```

You get these results:

| Employee | Sales Date | Sales | Rank Sales by Employee |
|---|---|---|---|
| Goh | 12/31/2018 | 140 | 1 |
| Goh | 10/31/2018 | 140 | 1 |
| Goh | 11/30/2018 | 60 | 3 |
| Freeman | 12/31/2018 | 160 | 1 |
| Freeman | 10/31/2018 | 110 | 2 |
| Freeman | 11/30/2018 | 60 | 3 |

| Employee | Sales Date | Sales | Rank Sales by Employee |
|---|---|---|---|
| Smith | 12/31/2018 | 140 | 1 |
| Smith | 10/31/2018 | 120 | 2 |
| Smith | 11/30/2018 | 60 | 3 |

You can also rank the sales for each date in descending order, so the highest sales is given the ranking of 1. Use this expression in the Rank Sales by Date field:

```
RANK() OVER(PARTITION BY [Sales Date] ORDER BY [Sales] DESC)
```

You get these results:

| Employee | Sales Date | Sales | Rank Sales by Date |
|---|---|---|---|
| Freeman | 12/31/2018 | 160 | 1 |
| Goh | 12/31/2018 | 140 | 2 |
| Smith | 12/31/2018 | 140 | 2 |
| Goh | 11/30/2018 | 60 | 1 |
| Freeman | 11/30/2018 | 60 | 1 |
| Smith | 11/30/2018 | 60 | 1 |
| Goh | 10/31/2018 | 140 | 1 |
| Smith | 10/31/2018 | 120 | 2 |
| Freeman | 10/31/2018 | 110 | 3 |

Notice that tied values are given the same rank number and the following rank position is skipped.

#### 1.12.13.10 | ROW_NUMBER

## Description

`ROW_NUMBER` is a window aggregate function that partitions rows into groups, orders rows by a field, and assigns a unique, sequential number to each row in a group, starting at 1 for the first row in each group. `ROW_NUMBER` always assigns a unique value to each row in a group. You might want to use `ROW_NUMBER` to create a unique ID for each row in your dataset.

The `PARTITION BY` clause determines which fields to use to partition a set of input rows into groups.

The `ORDER BY` clause determines how to order the rows in the partition before they're assigned a sequential number.

Workday separates the input rows into groups according to the partitioning fields, orders the rows according to the ordering fields, and then computes the aggregate expression (row numbering for this function) in each group. The numbered rows in each group start at 1.

## Syntax

```
ROW_NUMBER() OVER(
   PARTITION BY partitioning_field [, partitioning_field]
   ORDER BY ordering_field [ASC | DESC] [, ordering_field [ASC | DESC]]
   )
```

## Return Value

Returns a value of type `Integer`.

## Input Parameters

**OVER()**
Required. `OVER` must be used within a `ROW_NUMBER` expression.

**PARTITION BY *partitioning_field***
Required. Use the `PARTITION BY` clause to specify 1 or more fields to use to partition a group of input rows. You can specify any field type except Currency. Example: You specify the Month field as the partitioning field, so Workday groups together into a single partition all records that have the same value for Month.

**ORDER BY *ordering_field***
Required. Use the `ORDER BY` clause to specify how to order the input rows in the partition using the values in the specified field within each partition. You can specify any field type except Currency.

You can use the `DESC` or `ASC` keywords to sort in descending order (high to low values, NULLs are last) or ascending order (low to high values, NULLs are first) for each ordering field. If you don't specify a sort order for an ordering field, Workday automatically sorts rows in ascending order.

## Examples

Example: You have a dataset with these rows and fields.

| Employee | Sales Date | Sales |
|---|---|---|
| Goh | 12/31/2018 | 140 |
| Goh | 11/30/2018 | 60 |
| Goh | 10/31/2018 | 140 |
| Freeman | 12/31/2018 | 160 |
| Freeman | 11/30/2018 | 60 |
| Freeman | 10/31/2018 | 110 |
| Smith | 12/31/2018 | 140 |
| Smith | 11/30/2018 | 60 |
| Smith | 10/31/2018 | 120 |

You can assign a unique ID to the sales of each employee in descending order, so the highest sales is given the ranking of 1. Use this expression in the Sales Num by Employee field:

```
ROW_NUMBER() OVER(PARTITION BY [Employee] ORDER BY [Sales] DESC)
```

You get these results:

| Employee | Sales Date | Sales | Sales Num by Employee |
|---|---|---|---|
| Goh | 12/31/2018 | 140 | 1 |
| Goh | 10/31/2018 | 140 | 2 |
| Goh | 11/30/2018 | 60 | 3 |
| Freeman | 12/31/2018 | 160 | 1 |
| Freeman | 10/31/2018 | 110 | 2 |
| Freeman | 11/30/2018 | 60 | 3 |
| Smith | 12/31/2018 | 140 | 1 |
| Smith | 10/31/2018 | 120 | 2 |
| Smith | 11/30/2018 | 60 | 3 |

You can also assign a unique ID to the sales for each date in descending order, so the highest sales is given the ranking of 1. Use this expression in the Sales Num by Date field:

```
ROW_NUMBER() OVER(PARTITION BY [Sales Date] ORDER BY [Sales] DESC)
```

You get these results:

| Employee | Sales Date | Sales | Sales Num by Date |
|---|---|---|---|
| Freeman | 12/31/2018 | 160 | 1 |
| Goh | 12/31/2018 | 140 | 2 |
| Smith | 12/31/2018 | 140 | 3 |
| Goh | 11/30/2018 | 60 | 1 |
| Freeman | 11/30/2018 | 60 | 2 |
| Smith | 11/30/2018 | 60 | 3 |
| Goh | 10/31/2018 | 140 | 1 |
| Smith | 10/31/2018 | 120 | 2 |
| Freeman | 10/31/2018 | 110 | 3 |

#### 1.12.13.11 | SUM

## Description

SUM is a window aggregate function that partitions rows into groups, orders rows by a field, and returns the total of all values in the group. You can use SUM to calculate running totals.

The PARTITION BY clause determines which fields to use to partition a set of input rows into groups. The ORDER BY clause determines how to order the rows in the partition.

Workday separates the input rows into groups according to the partitioning fields, orders the rows according to the ordering fields, and then computes the aggregate expression (sum for this function) in each group.

## Syntax

```
SUM(input_field) OVER(
  PARTITION BY partitioning_field [, partitioning_field]
  ORDER BY ordering_field [ASC | DESC] [, ordering_field [ASC | DESC]]
  RANGE BETWEEN value PRECEDING AND CURRENT ROW |
  ROWS win_boundary | BETWEEN win_boundary AND win_boundary
)

where win_boundary can be:

  UNBOUNDED PRECEDING
  value PRECEDING
  UNBOUNDED FOLLOWING
  value FOLLOWING
  CURRENT ROW
```

## Return Value

Returns a value of type Numeric, Long, or Double depending on the type of *input_field*.

## Input Parameters

***input_field***
Required. The field on which to perform the aggregate function. You can use any numeric field or a Currency field.

**OVER()**
Required. OVER must be used within an SUM expression.

**PARTITION BY *partitioning_field***
Required. Use the PARTITION BY clause to specify 1 or more fields to use to partition a group of input rows. You can specify any field type except Currency. Example: You specify the Month field as the partitioning field, so Workday groups into a single partition all records that have the same value for Month.

**ORDER BY *ordering_field***
Required. Use the ORDER BY clause to specify how to order the input rows in the partition using the values in the specified field within each partition. You can specify any field type except Currency. However, you must use a numeric field type, such as Integer or Numeric when you use the RANGE clause.

You can use the DESC or ASC keywords to sort in descending order (high to low values, NULLs are last) or ascending order (low to high values, NULLs are first) for each ordering field. If you don't specify a sort order for an ordering field, Workday automatically sorts rows in ascending order.

**ROWS | RANGE**

Required. The ROWS and RANGE clauses define the specific number of rows (relative to the current row) within the partition by specifying a window frame. You define the window frame by specifying start and end points within the partition, known as window boundaries. The window frame is the set of input rows in each partition over which to calculate the aggregate expression (sum for this function). The window frame can include one, several, or all rows of the partition.

Both ROWS and RANGE specify the range of rows relative to the current row, but RANGE operates logically on values (logical association) and ROWS operates physically on rows in the dataset (physical association).

RANGE limits the window frame to contain rows that have their values within the specified range, relative to the current value. ROWS limits the window frame to contain rows that are physically next to the current row.

Use RANGE to define absolute window boundaries, such as the past 3 months or year to date. When you use RANGE, the ORDER BY clause must use a numeric field type, such as Integer or Numeric.

Example: Suppose you have an Integer field called MonthNum that represents the number of the month in the year (values 1 to 12). To specify all values from the past 3 months, you would order by MonthNum and use RANGE BETWEEN 2 PRECEDING AND CURRENT ROW. This RANGE clause includes the current month and the previous 2 months, resulting in 3 months total.

Note: When you publish a dataset that contains a window function using RANGE, the number of rows in the window must be 1000 or less. If a particular window exceeds 1000 rows, the publish job fails.

***win_boundary***

Required. The window boundaries define the start and end points of the window frame. Window boundaries are relative to the current row.

A `PRECEDING` clause defines a window boundary that is lower than the current row (the number of rows to include before the current row). The `FOLLOWING` clause defines a window boundary that is greater than the current row (the number of rows to include after the current row).

If you specify only 1 window boundary, then Workday uses the current row as the other boundary in the window frame (either the upper or lower boundary depending on the expression syntax). The `UNBOUNDED` keyword includes all rows in the direction specified. When you need to specify both a start and end of a window frame, use the `BETWEEN` and `AND` keywords.

When specifying a specific number of rows, the *value* must be 100 or less.

Example: `ROWS 2 PRECEDING` means that the window is 3 rows in size, starting with 2 rows preceding until and including the current row.

Example: `ROWS UNBOUNDED FOLLOWING` means that the window starts with the current row and includes the current row and all rows that come after the current row.

## Examples

You can calculate the total sales for each employee:

```
SUM([Sales]) OVER(PARTITION BY [Employee] ORDER BY [Sales] DESC ROWS UNBOUNDED PRECEDING)
```

You can calculate the rolling 12 month sum:

```
SUM([fieldA]) OVER(PARTITION BY [fieldB] ORDER BY [Month] RANGE 11 PRECEDING)
```

The `Month` field must be a numeric field type, such as Integer or Numeric.

You can calculate the previous year to date sum:

```
SUM([fieldA]) OVER(PARTITION BY [fieldB] ORDER BY [Year] RANGE 1 PRECEDING)
```

The `Year` field must be a numeric field type, such as Integer or Numeric.

### 1.12.14 | Regular Expression Reference

#### 1.12.14.1 | Concept: Regular Expressions in Prism

Regular expressions vary in complexity using a combination of basic constructs to describe a string matching pattern. This reference describes the most common regular expression matching patterns, but is not a comprehensive list.

Regular expressions, also referred to as regex or regexp, are a standardized collection of special characters and constructs used for matching strings of text. They provide a flexible and precise language for matching particular characters, words, or patterns of characters.

Prism Analytics regular expressions are based on the pattern matching syntax of the Java programming language. For more in depth information on writing valid regular expressions, refer to the Java regular expression pattern documentation.

You can use regular expressions in Prism calculated field expressions that use either the `REGEX` or `REGEX_REPLACE` functions.

#### 1.12.14.2 | Regex Literal and Special Characters

This section describes the regular expression syntax for referring to literal characters, special characters, nonprintable characters (such as a tab or a newline), and special character escaping.

## Literal Characters

The most basic form of pattern matching is the match of literal characters. If the regular expression is `foo` and the input string is `foo`, the match will succeed because the strings are identical.

## Special Characters

Certain characters are reserved for special use in regular expressions. These special characters are called metacharacters. If you want to use special characters as literal characters, you must escape them.

| Character Name | Character | Reserved For |
|---|---|---|
| opening bracket | [ | Start of a character class |
| closing bracket | ] | End of a character class |
| hyphen | - | Character ranges within a character class |
| backslash | \ | General escape character |
| caret | ^ | Beginning of string, negating of a character class |
| dollar sign | $ | End of string |
| period | . | Matching any single character |
| pipe | \| | Alternation (OR) operator |

| Character Name | Character | Reserved For |
|---|---|---|
| question mark | ? | Optional quantifier, quantifier minimizer |
| asterisk | * | Zero or more quantifier |
| plus sign | + | Once or more quantifier |
| opening parenthesis | ( | Start of a subexpression group |
| closing parenthesis | ) | End of a subexpression group |
| opening brace | { | Start of min/max quantifier |
| closing brace | } | End of min/max quantifier |

## Escaping Special Characters

You can use these methods to treat a special character as a literal (ordinary) character:

- Precede the special character with a `\` (backslash character). Example: to specify an asterisk as a literal character instead of a quantifier, use `\*`.
- Enclose the special characters within `\Q` (starting quote) and `\E` (ending quote). Everything between `\Q` and `\E` is then treated as literal characters.
- To escape literal double-quotes in a `REGEX()` expression, double the double-quotes (`""`). Example: to extract the inches portion from a `height` field where example values are `6'2"`, `5'11"`:

  ```
  REGEX(height, "\'(\d)+""$")
  ```

## NonPrinting Characters

You can use special character sequence constructs to specify nonprintable characters in a regular expression. Some of the most commonly used constructs are:

| Construct | Matches |
|---|---|
| `\n` | newline character |
| `\r` | carriage return character |
| `\t` | tab character |
| `\f` | form feed character |

#### 1.12.14.3 | Regex Character Classes

## Character Class Constructs

A character class allows you to specify a set of characters, enclosed in square brackets, that can produce a single character match. A character class matches to a single character only. For example, `gr[ae]y` will match to `gray` or `grey`, but not to `graay` or `graey`. The order of the characters inside the brackets doesn't matter.

You can use a hyphen inside a character class to specify a range of characters. Example: `[a-z]` matches a single lower-case letter between `a` and `z`. You can also use more than 1 range, or a combination of ranges and single characters. Example: `[0-9X]` matches a numeric digit or the letter `X`. The order of the characters and the ranges doesn't matter.

A caret following an opening bracket specifies characters to exclude from a match. For example, `[^abc]` matches any character except `a`, `b`, or `c`.

| Construct | Type | Description |
|---|---|---|
| [*abc*] | Simple | Matches `a` or `b` or `c` |
| `[^abc]` | Negation | Matches any character except `a` or `b` or `c` |
| `[a-zA-Z]` | Range | Matches `a` through `z`, or `A` through `Z` (inclusive) |
| `[a-d[m-p]]` | Union | Matches `a` through `d`, or `m` through `p` |
| `[a-z&&[def]]` | Intersection | Matches `d`, `e`, or `f` |
| `[a-z&&[^xq]]` | Subtraction | Matches `a` through `z`, except for `x` and `q` |

## Predefined Character Classes

Predefined character classes are convenient shorthands for commonly used regular expressions.

| Construct | Description | Example |
|---|---|---|
| `.` | Matches any single character (except newline) | `.at` matches "cat", "hat", and also "bat" in the phrase "batch files" |

| Construct | Description | Example |
|---|---|---|
| \d | Matches any digit character (equivalent to [0-9]) | \d matches "3" in "C3PO" and "2" in "file_2.txt" |
| \D | Matches any non-digit character (equivalent to [^0-9]) | \D matches "S" in "900S" and "Q" in "Q45" |
| \s | Matches any single white-space character (equivalent to [ \t\n\x0B\f\r]) | \sbook matches "book" in "blue book" but nothing in "notebook" |
| \S | Matches any single non-white-space character | \Sbook matches "book" in "notebook" but nothing in "blue book" |
| \w | Matches any alphanumeric character, including underscore (equivalent to [A-Za-z0-9_]) | r\w* matches "rm" and "root" |
| \W | Matches any non-alphanumeric character (equivalent to [^A-Za-z0-9_]) | \W matches "&" in "stmd &", "%" in "100%", and "$" in "$HOME" |

## POSIX Character Classes (US-ASCII)

POSIX has a set of character classes that denote certain common ranges. They're similar to bracket and predefined character classes, except they take into account the locale (the local language/coding system).

| Construct | Description |
|---|---|
| \p{Lower} | A lower-case alphabetic character, [a-z] |
| \p{Upper} | An upper-case alphabetic character, [A-Z] |
| \p{ASCII} | An ASCII character, [\x00-\x7F] |
| \p{Alpha} | An alphabetic character, [a-zA-z] |
| \p{Digit} | A numeric digit, [0-9] |
| \p{Alnum} | An alphanumeric character, [a-zA-z0-9] |
| \p{Punct} | A punctuation character, one of !"#$%&'()*+,-./:;<=>?@[\]^_`{\|}~ |
| \p{Graph} | A visible character, [\p{Alnum}\p{Punct}] |
| \p{Print} | A printable character, [\p{Graph}\x20] |
| \p{Blank} | A space or tab, [ t] |
| \p{Cntrl} | A control character, [\x00-\x1F\x7F] |
| \p{XDigit} | A hexadecimal digit, [0-9a-fA-F] |
| \p{Space} | A whitespace character, [ \t\n\x0B\f\r] |

#### 1.12.14.4 | Regex Line and Word Boundaries

You can use boundary matching constructs to specify where in a string to apply a matching pattern. For example, you can search for a particular pattern within a word boundary, or search for a pattern at the beginning or end of a line.

| Construct | Description | Example |
|---|---|---|
| ^ | Matches from the beginning of a line (multi-line matches are currently not supported) | ^172 matches the "172" in IP address "172.18.1.11" but not in "192.172.2.33" |
| $ | Matches from the end of a line (multi-line matches are currently not supported) | d$ matches the "d" in "maid" but not in "made" |
| \b | Matches within a word boundary | \bis\b matches the word "is" in "this is my island", but not the "is" part of "this" or "island". \bis matches both "is" and the "is" in "island", but not in "this". |
| \B | Matches within a non-word boundary | \Bb matches "b" in "sbin" but not in "bash" |

#### 1.12.14.5 | Regex Quantifiers

## Quantifier Constructs

Quantifiers specify how often the preceding regular expression construct should match. The classes of quantifiers are:

- Greedy
- Reluctant
- Possessive

The difference between greedy, reluctant, and possessive quantifiers involves what part of the string to try for the initial match, and how to retry if the initial attempt doesn't produce a match.

By default, quantifiers are *greedy*. A greedy quantifier first tries for a match with the entire input string. If that produces a match, then it considers the match a success and the engine can move on to the next construct in the regular expression. If the first try doesn't produce a match, the engine backs-off 1 character at a time until it finds a match. So a greedy quantifier checks for possible matches in order from the longest possible input string to the shortest possible input string, recursively trying from right to left.

Adding a `?` (question mark) to a greedy quantifier makes it *reluctant*. A reluctant quantifier first tries for a match from the beginning of the input string, starting with the shortest possible piece of the string that matches the regex construct. If that produces a match, then it considers the match a success and the engine can move on to the next construct in the regular expression. If the first try doesn't produce a match, the engine adds 1 character at a time until it finds a match. So a reluctant quantifier checks for possible matches in order from the shortest possible input string to the longest possible input string, recursively trying from left to right.

Adding a + (plus sign) to a greedy quantifier makes it *possessive*. A possessive quantifier is like a greedy quantifier on the first attempt (it tries for a match with the entire input string). The difference is that unlike a greedy quantifier, a possessive quantifier doesn't retry a shorter string if it doesn't find a match. If the initial match fails, the possessive quantifier reports a failed match. It doesn't make any more attempts.

| Greedy Construct | Reluctant Construct | Possessive Construct | Description | Example |
|---|---|---|---|---|
| `?` | `??` | `?+` | Matches the previous character or construct once or not at all. | `st?on` matches "son" in "johnson" and "ston" in "johnston" but nothing in "clinton" or "version" |
| `*` | `*?` | `*+` | Matches the previous character or construct zero or more times. | `if*` matches "if", "iff" in "diff", or "i" in "print" |
| `+` | `+?` | `++` | Matches the previous character or construct 1 or more times. | `if+` matches "if", "iff" in "diff", but nothing in "print" |
| `{n}` | `{n}?` | `{n}+` | Matches the previous character or construct exactly *n* times. | `o{2}` matches "oo" in "lookup" and the first 2 o's in "fooooo" but nothing in "mount" |
| `{n,}` | `{n,}?` | `{n,}+` | Matches the previous character or construct at least *n* times. | `o{2,}` matches "oo" in "lookup" all 5 o's in "fooooo" but nothing in "mount" |
| `{n,m}` | `{n,m}?` | `{n,m}+` | Matches the previous character or construct at least *n* times, but no more than *m* times. | `F{2,4}` matches "FF" in "#FF0000" and the last 4 F's in "#FFFFFF" |

#### 1.12.14.6 | Regex Capturing Groups

You can use a pair of parentheses around a subpattern in a regular expression to define a *group*. You can use regex groups to:

- **Apply regex operators and quantifiers to an entire group at once.**
- **Create a capturing group.** You can use capturing groups to specify matching values to save or return from your regular expression. By default, a group *captures* the text that produces a match. The portion of the string that matches the grouped subexpression is captured in memory for later retrieval or use.
- **Create a non-capturing group.**

### Group Numbering

A regular expression can have more than 1 group, and the groups can be nested. The groups are numbered 1-*n* from left to right, starting with the first opening parenthesis. There is always an implicit group zero (0), which contains the entire match. Example:

```
(a(b*))+(c)
```

This pattern contains 3 groups:

```
group 1: (a(b*))
group 2: (b*)
group 3: (c)
```

### Capturing Groups and the REGEX Function

You can use the `REGEX` function to extract a portion of a string. The `REGEX` function returns the value of the *first* capturing group only.

Suppose you have a field name called *email* that contains email addresses with this pattern: *username@provider.domain*. You can use the `REGEX` function to return just the *provider* portion of the email address from the *email* field:

```
REGEX(email,"^[a-zA-Z0-9._%+-]+@([a-zA-Z0-9._-]+)\.[a-zA-Z]{2,4}$")
```

### Capturing Groups and the REGEX_REPLACE Function

You can use the `REGEX_REPLACE` function to match a string, and replace matched strings with another value. The `REGEX_REPLACE` function takes 3 arguments:

- Input string
- Matching regex
- Replacement regex

You can use capturing groups to capture backreferences, but the entire match is always returned.

## Capturing Groups and Backreferences

You can use a backreference to refer back to the matched content of a particular capturing group number. Typically, you use a backreference in the same regular expression that contains the capturing group. You specify a backreference by referring to the group number preceded by a backslash. Use `\1` to refer to capturing group 1, `\2` to refer to capturing group 2, and so on.

Suppose you want to match a pair of HTML tags and their enclosed text. You could capture the opening tag into a capturing group, and then use a backreference to match the corresponding closing tag:

```
(<([A-Z][A-Z0-9]*)\b[^>]*>.*?</\2>)
```

This regular expression contains 2 capturing groups:

- Group 1 contains the outermost parentheses and captures the entire string.
- Group 2 captures the string matched by `[A-Z][A-Z0-9]*`.

You can then refer to group 2 using the `\2` backreference to match the corresponding closing HTML tag.

When you use the `REGEX_REPLACE` function, you can use a backreference to refer to a capturing group in the *previous* regular expression. The syntax is slightly different when you use a backreference to refer to a group in the previous regex. Use a dollar sign (`$`) before the group number, such as `$1` to specify a backreference to group 1 of the previous expression.

Suppose you have a *phone_number* field where the values are formatted as `xxx.xxx.xxxx`. The following example matches the values in *phone_number* and replaces them with values formatted as `(xxx) xxx-xxxx`:

```
REGEX_REPLACE(phone_number,"([0-9]{3})\.([[0-9]]{3})\.([[0-9]]{4})","\($1\) $2-$3")
```

Notice the backreferences in the replacement expression. They refer to the capturing groups of the previous matching expression.

## Non-Capturing Groups

In some cases, you might want to use parenthesis to group subpatterns, but *not* capture text. A non-capturing group starts with `(?:` (a question mark and colon following the opening parenthesis). For example, `h(?:a|i|o)t` matches `hat` or `hit` or `hot`, but does not capture the `a`, `i`, or `o` from the subexpression.