KPI Metrics Metadata Configuration Guide

An Open Source Asset for use with TIBCO® Data Virtualization

|  |  |
| --- | --- |
| **Project Name** | AS Assets KPI Metrics |
| **Document Location** | This document is only valid on the day it was printed. The source of the document will be found in the ASAssets\_KPI folder (https://github.com/TIBCOSoftware) |
| **Purpose** | Self-paced instructional |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| 1.0 | Aug 30 2019 | Mike Tinius | Initial revision |

Related Documents

|  |  |
| --- | --- |
| **Name** | **Version** |
| How To Use Utilities.pdf | 2019Q301 |

Supported Versions

|  |  |
| --- | --- |
| **Name** | **Version** |
| TIBCO® Data Virtualization | 7.0.8 or later |
| AS Assets Utilities open source | 2019Q301 or later |

**Table of Contents**

1 Introduction 4

Purpose 4

Audience 4

References 4

Overview 4

2 Requirements 5

3 Use Cases 6

4 Configuration 7

Configure Metadata Constants 7

Configure Trigger 10

5 KPImetrics Metatdata Resources 11

Configuration Resources 11

Published Resources 11

KPImetrics Catalog 11

Metadata Table Relationship Diagram 13

Metadata Data Source Tables 14

Metadata Data Source Tables and Procedures for KPI\_<database\_type> 14

Metadata System Triggers and Load Scripts 17

1. Introduction

## Purpose

The purpose of this document is to provide guidance on how configure and use the AS Assets KPI Metadata.

## Audience

This document is intended to provide guidance for the following users:

* Data Virtualization Administrators – provides a guide for installation.
* Architects – provides the KPImetrics architecture.
* Data professionals – provides background on the published views and usage.
* Operations users – provides insight into triggers and procedures that are executed.
* Project Managers – provides general information on KPImetrics.

## References

Product references are shown below. Any references to CIS or DV refer to the current TIBCO® Data Virtualization.

* TIBCO® Data Virtualization was formerly known as
  + Cisco Data Virtualization (DV)
  + Composite Information Server (CIS)

## Overview

Please review the document “**KPImetrics Overview.pdf**”.

1. Requirements

The following requirements and pre-requisites must be met:

* See requirements section in KPImetrics Configuration Guide vx.yy.pdf.

1. Use Cases

Metadata Metrics – The following use cases are examples of design-time metrics. Design-time is different than KPI metrics run-time metrics.

1. How many views do not properly adhere to the layer rules? – compliance with architecture.
   1. Each layer should invoke the appropriate layer below it.  Should never invoke source views.

**reportMetadataNonCompliantLayers**

1. Which connector/adapter is used by which views

**reportMetadataDatasource**

1. Source View is compliant with additional columns: source code, fetchtimestamp etc.

**reportMetadataNonCompliantColumns**

1. # views by layers

**reportNumResourcesByLayer**

1. Owner of views.  Who has modified.

**vMetadataResource**

1. # policy, roles, policy name, attributes, description

**vMetadataPolicy**

**vMetadataPolicyAssignmnt**

1. Metadata regarding access and authorization for a give resource associated with access groups.

**vMetadataPrivilege**

1. Configuration

### Configure Metadata Constants

Background Information:

The procedure “10\_pqInsert\_Metadata\_Tables\_METADATA\_Constants” is used to configure various constants for a given “project”. A project has a base path which encompasses all of the layer folders and resources.

This procedure “DOES NOT” need to be executed manually. It will be executed each time the trigger “kpimetricsTrig\_40\_Cache\_METADATA\_TABLES” executes. The trigger executes Cache\_METADATA\_TABLES which in turn executes “10\_pqInsert\_Metadata\_Tables\_METADATA\_Constants”. It does this so that all metadata is kept in synch with the same LOAD\_DATE across all of the tables.

Instructions:

* Configure the following /shared/ASAssets/KPImetrics/Physical/Metadata/DDL/Common/ 10\_pqInsert\_Metadata\_Tables\_METADATA\_Constants.
* Configure the section “INSERT METADATA\_CONST\_NAME ROWS”
  + Modify the concatenated string below as needed. Add a row for each "project" name to capture metadata for.
  + PROJECT\_NAME: A unique name that will be assigned a unique ID.
  + EXECUTE\_FLAG: Y=execute this row. N=do not execute when triggered.
  + ARCHIVE\_FLAG: Y=archive rows before processing. N=do not archive.
  + ARCHIVE\_PURGE\_DAYS: The number of days to purge from the current date.
  + PROJECT\_DESC: A description of the project path.
  + Maintain the existing structure with double pipe separating the line and single pipe separating a column.

SET projectName = 'TestSpoke';

SET METADATA\_CONST\_NAME\_str = METADATA\_CONST\_NAME\_str ||

PROJECT\_NAME EXECUTE\_FLAG ARCHIVE\_FLAG ARCHIVE\_PURGE\_DAYS PROJECT\_DESC

'||'||projectName ||'|'|| 'Y' ||'|'|| 'N' ||'|'|| 30 ||'|'|| 'TestSpoke project desc' ||

''; -- This is always the last line

* Configure the section “INSERT METADATA\_CONST\_PATH ROWS”
  + Modify the concatenated string below as needed. Add a row for each base path within the "project" to capture metadata for.
    - * Modify projectName, pathSH, pathDS.
      * The variable "pathWS" is not currently supported for web services.
      * Modify the PROJECT\_PATH and RESOURCE\_TYPES as per your requirements.
  + Maintain the existing structure with double pipe separating the line and single pipe separating a column.
  + PROJECT\_NAME: A foreign key reference to METADATA\_CONST\_NAME which provides a unique name that will be assigned a PROJECT\_NAME\_ID that is unique.
  + PROJECT\_PATH: A unique key for this table which drives all of the processing for Cache\_METADATA\_TABLES procedure to load data.
  + RESOURCE\_TYPES: - A comma-separated list of resource types to process.
    - * When using pathSH for shared area then [TABLE,PROCEDURE]
      * When using pathDS for /services/databases then [LINK]
      * When using pathWS for /services/webservices then [LINK]
    - NOTE: Web Services are not currently supported.

SET projectName = 'TestSpoke';

SET pathSH = '/shared/00\_DataFederation/TestSpoke';

SET pathDS = '/services/databases/PWC/TestSpoke';

SET pathWS = '';

SET METADATA\_CONST\_PATHS\_str = METADATA\_CONST\_PATHS\_str ||

--PROJECT\_NAME PROJECT\_PATH RESOURCE\_TYPES

'||'||projectName ||'|'|| pathSH ||'|'|| 'TABLE,PROCEDURE' ||

'||'||projectName ||'|'|| pathDS ||'|'|| 'LINK' ||

--'||'||projectName ||'|'|| pathWS ||'|'|| 'LINK' ||

''; -- This is always the last line

* Configure the section “INSERT METADATA\_CONST\_LAYERS ROWS”
  + Modify the concatenated string below as needed. Only modify the layer type and parent path after the standard project path.
    - * Modify projectName, pathSH, pathDS.
      * The variable "pathWS" is not currently supported for web services.
      * Modify the PROJECT\_PATH, LAYER\_TYPE, PARENT\_PATH and GENERATE\_LINEAGE as per your requirements.
  + Maintain the existing structure with double pipe separating the line and single pipe separating a column.
  + PROJECT\_NAME: A foreign key reference to METADATA\_CONST\_NAME which provides a unique name that will be assigned a PROJECT\_NAME\_ID that is unique.
  + PROJECT\_PATH: Provides a foreign key back to META\_DRIVER table.
  + LAYER\_TYPE: A unique string describing the layer to acquire metadata for.
  + PARENT\_PATH: The actual path in DV which is associated with the LAYER\_TYPE.
  + GENERATE\_LINEAGE: Y=Generate lineage for this layer path. N=Do not generate lineage for this layer path.
  + Rules:
    - * A LAYER\_TYPE that is a parent to a sub-folder is allowed and it will not cause duplication of resources. This concept will work in any layer including /shared and published /services/databases.
      * The table METADATA\_CONST\_LAYERS is validated for duplicates. If a duplicate layer and PARENT\_PATH is found an exception is thrown.
      * Each LAYER\_TYPE should have a unique name within a given PROJECT\_NAME\_ID.

For example,

1) Given the following layer type designations, there is a grandparent-parent-child folder relationship represented here:

Note: The number of levels/layers is NOT restricted.

LAYER\_TYPE: PARENT\_PATH:

Note: 01\_SourceViewLayer is a parent to 01\_SourceViewLayer\_svThirdParty

01\_SourceViewLayer /shared/00\_DataFederation/TestSpoke/01\_SourceViewLayer

Note: 01\_SourceViewLayer\_svThirdParty is a parent to 01\_SourceViewLayer\_svThirdParty\_A and 01\_SourceViewLayer\_svThirdParty\_B

01\_SourceViewLayer\_svThirdParty /shared/00\_DataFederation/TestSpoke/01\_SourceViewLayer/012\_svThirdParty

01\_SourceViewLayer\_svThirdParty\_A /shared/00\_DataFederation/TestSpoke/01\_SourceViewLayer/012\_svThirdParty/012\_svThirdParty\_A

01\_SourceViewLayer\_svThirdParty\_B /shared/00\_DataFederation/TestSpoke/01\_SourceViewLayer/012\_svThirdParty/012\_svThirdParty\_B

2) Given the following resources, the layer type will be assigned from the child (lowest folder) up to the grandparent (highest) folder.

LAYER\_TYPE RESOURCE\_PATH

01\_SourceViewLayer\_svThirdParty\_A /shared/00\_DataFederation/TestSpoke/01\_SourceViewLayer/012\_svThirdParty/012\_svThirdParty\_A/012\_svThirdParty\_A1/customers

01\_SourceViewLayer\_svThirdParty\_A /shared/00\_DataFederation/TestSpoke/01\_SourceViewLayer/012\_svThirdParty/012\_svThirdParty\_A/customers

01\_SourceViewLayer\_svThirdParty\_B /shared/00\_DataFederation/TestSpoke/01\_SourceViewLayer/012\_svThirdParty/012\_svThirdParty\_B/012\_svThirdParty\_B1/customers

01\_SourceViewLayer\_svThirdParty\_B /shared/00\_DataFederation/TestSpoke/01\_SourceViewLayer/012\_svThirdParty/012\_svThirdParty\_B/012\_svThirdParty\_B2/customers

01\_SourceViewLayer\_svThirdParty\_B /shared/00\_DataFederation/TestSpoke/01\_SourceViewLayer/012\_svThirdParty/012\_svThirdParty\_B/customers

01\_SourceViewLayer\_svThirdParty /shared/00\_DataFederation/TestSpoke/01\_SourceViewLayer/012\_svThirdParty/customers

01\_SourceViewLayer /shared/00\_DataFederation/TestSpoke/01\_SourceViewLayer/011\_svInternal/tutorial/customers

01\_SourceViewLayer /shared/00\_DataFederation/TestSpoke/01\_SourceViewLayer/DS\_ORDERS/tutorial/customers

The following demonstrates how to setup the constants.

SET projectName = 'TestSpoke';

SET pathSH = '/shared/00\_DataFederation/TestSpoke';

SET pathDS = '/services/databases/PWC/TestSpoke';

SET pathWS = '';

SET METADATA\_CONST\_LAYERS\_str = METADATA\_CONST\_LAYERS\_str ||

--PROJECT\_NAME PROJECT\_PATH LAYER\_TYPE PARENT\_PATH GENERATE\_LINEAGE

'||'||projectName ||'|'|| pathSH||'|'|| '00\_DataSource'||'|'|| pathSH||'/00\_DataSource'||'|'|| 'N'||

'||'||projectName ||'|'|| pathSH||'|'|| '01\_SourceViewLayer' ||'|'||pathSH||'/01\_SourceViewLayer' ||'|'|| 'N'||

'||'||projectName ||'|'|| pathSH||'|'|| '02\_ConformingViewLayer'||'|'||pathSH||'/02\_ConformingViewLayer'||'|'|| 'N'||

'||'||projectName ||'|'|| pathSH||'|'|| '031\_CommonEntityModel'||'|'||pathSH||'/03\_CommonModelLayer/031\_CommonEntityModel'||'|'||'N'||

'||'||projectName ||'|'|| pathSH||'|'|| '032\_CommonDimensionalModel'||'|'||pathSH||'/03\_CommonModelLayer/032\_CommonDimensionalModel' ||'|'||'N'||

'||'||projectName ||'|'|| pathSH||'|'|| '033\_CommonAnalyticalModel'||'|'||pathSH||'/03\_CommonModelLayer/033\_CommonAnalyticalModel'||'|'||'N'||

'||'||projectName ||'|'|| pathSH||'|'|| '034\_CommonIntegrationModel'||'|'||pathSH||'/03\_CommonModelLayer/034\_CommonIntegrationModel'||'|'||'N'||

'||'||projectName ||'|'|| pathSH||'|'|| '041\_BusinessDemandModel'||'|'||pathSH||'/04\_BusinessDeliveryLayer/041\_BusinessDemandModel'||'|'||'N'||

'||'||projectName ||'|'|| pathSH||'|'|| '042\_BusinessDemandView'||'|'||pathSH||'/04\_BusinessDeliveryLayer/042\_BusinessDemandView'||'|'||'Y'||

'||'||projectName ||'|'|| pathDS||'|'|| 'PublishedDS\_tutorial' ||'|'|| pathDS ||'|'||'Y'||

--'||'||projectName ||'|'|| pathWS||'|'|| 'PublishedWS\_tutorial' ||'|'|| pathDS ||'|'||'Y'||

''; -- This is always the last line

* Configure the section “INSERT METADATA\_CONST\_VALIDATE ROWS”
  + Modify the concatenated string below as needed.
    - * Modify projectName, pathSH, pathDS.
      * The variable "pathWS" is not currently supported for web services.
      * Modify the PROJECT\_PATH, LAYER\_TYPE, RULE\_TYPE and RULE\_DESC as per your requirements.
  + Maintain the existing structure with double pipe separating the line and single pipe separating a column.
  + PROJECT\_NAME: A foreign key reference to METADATA\_CONST\_NAME which provides a unique name that will be assigned a PROJECT\_NAME\_ID that is unique.
  + LAYER\_TYPE: A valid layer name found in the table METADATA\_CONST\_LAYERS.
  + RULE\_TYPE: Valid values=[ENFORCE\_LAYER|ENFORCE\_COLUMN]
  + RULE\_DESC: Enforce the rule type.
    - * When RULE\_TYPE=ENFORCE\_COLUMN
        1. Enforces which columns must be present in all of the views for a given layer type. Comma-separated list of case-sensative column names.
      * When RULE\_TYPE=ENFORCE\_LAYER
        1. Enforces which source layer resource can invoke which target layer resource. Comma-separated list of valid LAYER\_TYPES.
        2. If a resource can invoke another resource in the same layer then add its own layer to the list.

SET projectName = 'TestSpoke';

SET pathSH = '/shared/00\_DataFederation/TestSpoke';

SET pathDS = '/services/databases/PWC/TestSpoke';

SET pathWS = '';

SET METADATA\_CONST\_VALIDATE\_str = METADATA\_CONST\_VALIDATE\_str ||

--PROJECT\_NAME PROJECT\_PATH LAYER\_TYPE RULE\_TYPE RULE\_DESC

'||'||projectName ||'|'||pathSH ||'|'|| '01\_SourceViewLayer' ||'|'||'ENFORCE\_LAYER' ||'|'|| '00\_DataSource'||

'||'||projectName ||'|'||pathSH ||'|'|| '01\_SourceViewLayer' ||'|'||'ENFORCE\_COLUMN'||'|'|| 'fetchTimeStamp,systemSourceCode'||

'||'||projectName ||'|'||pathSH ||'|'|| '02\_ConformingViewLayer'||'|'||'ENFORCE\_LAYER'||'|'|| '01\_SourceViewLayer'||

'||'||projectName ||'|'||pathSH ||'|'|| '031\_CommonEntityModel'||'|'||'ENFORCE\_LAYER'||'|'|| '02\_ConformingViewLayer'||

'||'||projectName ||'|'||pathSH ||'|'|| '032\_CommonDimensionalModel'||'|'|| 'ENFORCE\_LAYER'||'|'|| '02\_ConformingViewLayer'||

'||'||projectName ||'|'||pathSH ||'|'|| '033\_CommonAnalyticalModel'||'|'||'ENFORCE\_LAYER'||'|'|| '02\_ConformingViewLayer'||

'||'||projectName ||'|'||pathSH ||'|'|| '034\_CommonIntegrationModel'||'|'|| 'ENFORCE\_LAYER'||'|'|| '02\_ConformingViewLayer'||

'||'||projectName ||'|'||pathSH ||'|'|| '041\_BusinessDemandModel'||'|'||'ENFORCE\_LAYER'||'|'|| '031\_CommonEntityModel,032\_CommonDimensionalModel,034\_CommonIntegrationModel,041\_BusinessDemandModel'||

'||'||projectName ||'|'||pathSH ||'|'|| '042\_BusinessDemandView'||'|'||'ENFORCE\_LAYER'||'|'|| '041\_BusinessDemandModel'||

'||'||projectName ||'|'||pathDS ||'|'|| 'PublishedDS\_tutorial' ||'|'||'ENFORCE\_LAYER' ||'|'|| '042\_BusinessDemandView'||

--'||'||projectName ||'|'||pathWS ||'|'|| 'PublishedWS\_tutorial' ||'|'||'ENFORCE\_LAYER' ||'|'|| '042\_BusinessDemandView'||

''; -- This is always the last line

### Configure Trigger

Enabling triggers starts the processing of KPI metadata data. The trigger “kpimetricsTrig\_40\_Cache\_METADATA\_TABLES” is turned off by default. It must be turned on in order to begin the processing of

1. Modify /shared/ASAssets/KPImetrics/Configuration/**defaultTriggersToEnable** and change the trigger kpimetricsTrig\_40\_Cache\_METADATA\_TABLES from OFF to ON if you want to capture metadata.
2. When updateTriggers is executed, it will turn on and off the trigger automatically according to how the trigger is set in defaultTriggersToEnable.
3. KPImetrics Metatdata Resources

## Configuration Resources

This section outlines the resources that are used for configuration of KPImetrics Metadata.

## Published Resources

This section outlines the resources that are published under the ASAssets virtual database to expose metrics data. Resources are organized under catalogs and schemas based upon their functionality.

### KPImetrics Catalog

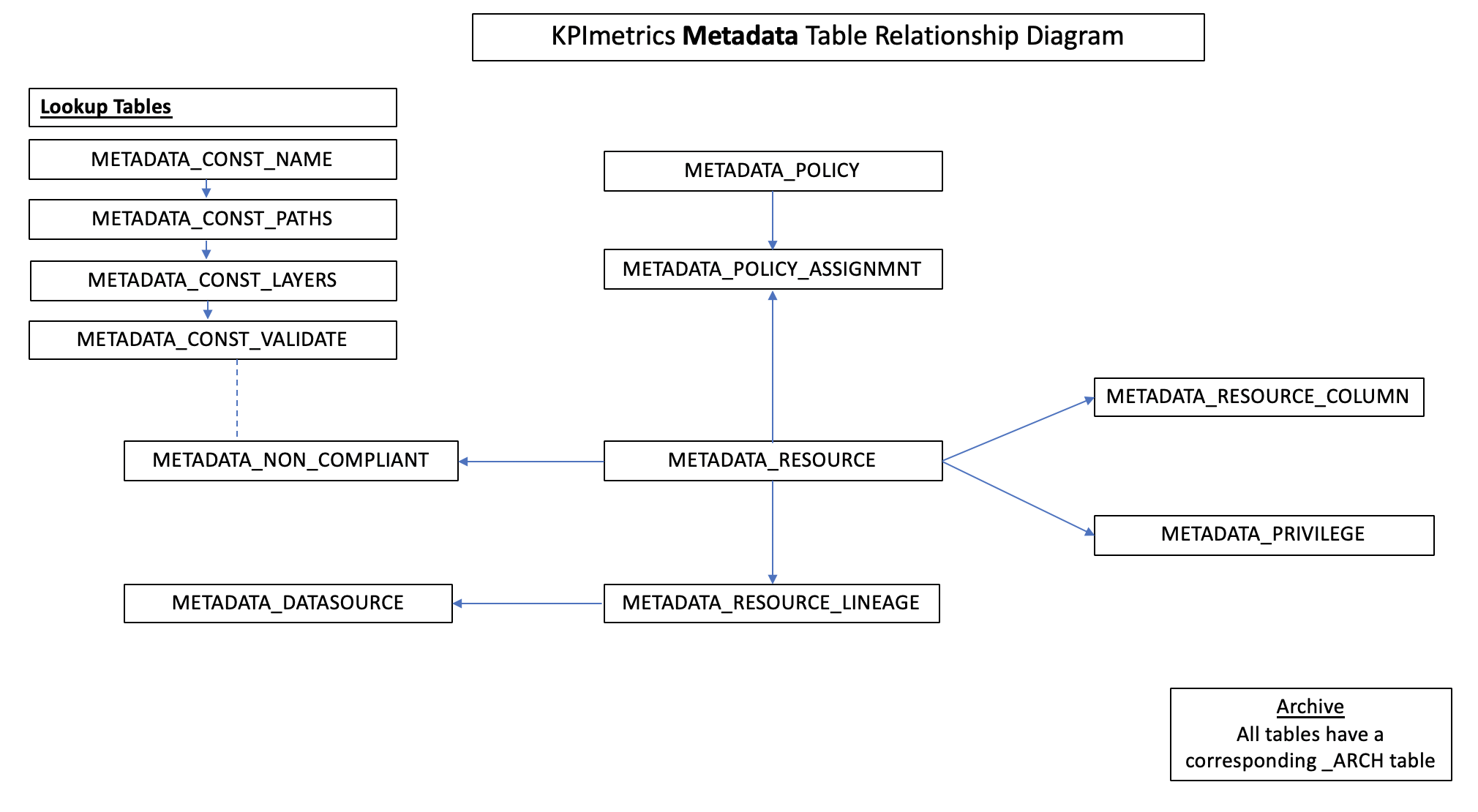
#### metadata schema

|  |  |
| --- | --- |
| **Resource (Resource Type)** | **Description** |
| reportMetadataDatasource (TABLE) | Report on what datasource a view is using. Lineage from the view to the datasource. This can be joined with vMetadataDatasource to get additional datasource information. It may also be joined with vMetadataResource to get additional resource information. |
| reportMetadataNonCompliantColumns (TABLE) | Report of the non-compliant columns as configured by METADATA\_CONST\_VALIDATE. |
| reportMetadataNonCompliantLayers (TABLE) | Report of the non-compliant layers as configured by METADATA\_CONST\_VALIDATE. |
| reportNumResourcesByLayer (TABLE) | Report of the number of resources in each layer. |
| reportResourceColumns (TABLE) | Report of all the resources and their columns. |
| vMetadataConstName (TABLE)  vMetadataConstNameArch (ARCHIVE TABLE) | Provides a listing of all configured metadata constant name projects. This is the main driver table. It contains the project name, project name id, environment name, execute flag and archive flag. |
| vMetadataConstPaths (TABLE)  vMetadataConstPathsArch (ARCHIVE TABLE) | Provides a listing of all configured metadata constant name projects. This table controls the paths that are connected with the vMetadataConstName view. |
| vMetadataConstLayers (TABLE)  vvMetadataConstLayersArch (ARCHIVE TABLE) | Provides a listing of all of the metadata layer types for each project path. |
| vMetadataConstValidate (TABLE)  vMetadataConstValidateArch (ARCHIVE TABLE) | Provides a listing of all of the metadata layer validations for each project path. |
| vMetadataResource (TABLE)  vMetadataResourceArch (ARCHIVE TABLE) | Provides a listing of all of the metadata resources for each project path. This is the core table that provides information about a resource. |
| vMetadataResourceColumn (TABLE)  vMetadataResourceColumnArch (ARCHIVE TABLE) | Provides a listing of all of the metadata resource columns for each resource and project path. This is the core table that provides information about columns and their position. |
| vMetadataResourceLineage (TABLE)  vMetadataResourceLineageArch (ARCHIVE TABLE) | Provides a listing of the lineage for each resource at each layer. Lineage will be different based on which layer is referenced. Each record also references the datasource that is used. |
| vMetadataDatasource (TABLE)  vMetadataDatasourceArch (ARCHIVE TABLE) | Provides a complete reference on the datasources that exist within the project paths referenced by vMetadataConstName. |
| vMetadataNonCompliant (TABLE)  vMetadataNonCompliantArch (ARCHIVE TABLE) | Provides information on all non-compliant resources for all layers. |
| vMetadataPolicy (TABLE)  vMetadataPolicyArch (ARCHIVE TABLE) | Provides information on all policies for RBS [rule-based security] and CBS [column-based security] rows for a given project path. |
| vMetadataPolicyAssignmnt (TABLE)  vMetadataPolicyAssignmntArch (ARCHIVE TABLE) | Provides the policy assignments for each policy. |
| vMetadataPrivilege (TABLE)  vMetadataPrivilegeArch (ARCHIVE TABLE) | Provides a listing of all group and user privileges for each resource for a given project path. |

### 

## Metadata Table Relationship Diagram

The following provides a table relationship diagram for the database tables and procedures used by KPImetrics Metadata.



## Metadata Data Source Tables

The following provides a description for the database tables used by KPImetrics Metadata.

### Metadata Data Source Tables and Procedures for KPI\_<database\_type>

Location: /shared/ASAssets/KPImetrics/Physical/Metadata/KPI\_<database\_type>

The KPImetrics module provides data source for all currently supported storage database platforms under /shared/ASAssets/KPImetrics/Physical/Metadata.

Currently the KPImetrics module includes the following KPImetrics data sources

* /shared/ASAssets/KPImetrics/Physical/Metadata/KPI\_oracle
* /shared/ASAssets/KPImetrics/Physical/Metadata/KPI\_sqlserver

The following tables have been created in CIS\_KPI schema to capture the required data. Each table has a corresponding archive table.

RULES:

* + Only one load set of data is stored at any given point in time in the main metadata tables.
  + When METADATA\_CONST\_NAME.ARCHIVE\_FLAG=Y then each table is archived to its corresponding archive table.
  + Each node in a cluster will contain its own set of metadata rows therefore, NODE\_HOST and NODE\_PORT are a part of every key. Even though the resource name will be the same, the RESOURCE\_ID may be different on any given node. Be sure to do reporting based on a particular NODE\_HOST and NODE\_PORT.

|  |  |
| --- | --- |
| **Table Name** | **Description** |
| METADATA\_CONST\_NAME  METADATA\_CONST\_NAME\_ARCH | This table contains a unique base project path that drives all of the metadata capture for all of the tables. Only metadata is captured the project paths present in this table. The trigger specified below along with the procedure it invokes is the only mechanism for capturing metadata for all of the metadata tables listed here.  LOAD\_DATE: The timestamp of the latest metadata load.  PROJECT\_NAME\_ID: A unique sequence id for each project name.  PROJECT\_NAME: A unique name that will be assigned a PROJECT\_NAME\_ID that is unique.  ENVIRONMENT\_NAME: The environment nickname from commonValues.cisServerNickname.  EXECUTE\_FLAG: Y=execute this row. N=do not execute when triggered.  ARCHIVE\_FLAG: Y=archive rows before processing. N=do not archive. Note: all rows get deleted each time the trigger executes. Archive is the only way to maintain history.  ARCHIVE\_PURGE\_DAYS: The number of days to purge from the current date.  PROJECT\_DESC: A description of the project path.  RESOURCE\_TYPES: TABLE,PROCEDURE - A comma-separated list of resource types to process. Currently only TABLE and PROCEDURE are valid.  EXECUTE\_STATUS: The status of the latest load. SUCCESS or EXCEPTION which includes the exception message.  NODE\_HOST: Indicates which hostname/node the processing took place on. Multiple hosts/nodes in a cluster.  NODE\_PORT: Indicates the port of the DV server in which the processing took place on.  TRIGGER:  /KPImetrics/Physical/Metadata/System/ClusterSafeTriggers/ kpimetricsTrig\_40\_Cache\_METADATA\_TABLES🡪 Cache\_METADATA\_TABLES  KEY: LOAD\_DATE, PROJECT\_NAME\_ID, PROJECT\_NAME, NODE\_HOST, NODE\_PORT |
| METADATA\_CONST\_PATHS  METADATA\_CONST\_PATHS\_ARCH | This table contains a list of base project paths that drives all of the metadata capture for all of the tables. Only metadata is captured the project paths present in this table.  PROJECT\_PATH: A unique key for this table which drives all of the processing for Cache\_METADATA\_TABLES procedure to load data.  RESOURCE\_TYPES: TABLE,PROCEDURE,LINK - A comma-separated list of resource types to process.  KEY: LOAD\_DATE, PROJECT\_NAME\_ID, PROJECT\_PATH, NODE\_HOST, NODE\_PORT |
| METADATA\_CONST\_LAYERS  METADATA\_CONST\_LAYERS\_ARCH | This table contains the valid layer types for each project path. A layer type has a corresponding parent path within the project path that it correlates to.  PROJECT\_PATH: Provides a foreign key back to METADATA\_CONST\_NAME table.  LAYER\_TYPE: A unique string describing the layer to acquire metadata for.  PARENT\_PATH: The actual path in DV which is associated with the LAYER\_TYPE.  KEY: LOAD\_DATE, PROJECT\_NAME\_ID, LAYER\_TYPE, NODE\_HOST, NODE\_PORT |
| METADATA\_CONST\_VALIDATE  METADATA\_CONST\_VALIDATE\_ARCH | This table contains the layer validation rules. The rules provide for enforcing colunms within views and which views can invoke views in specific layers.  PROJECT\_PATH: Provides a foreign key back to METADATA\_CONST\_NAME table.  LAYER\_TYPE: A valid layer name found in the table METADATA\_CONST\_LAYERS.  RULE\_TYPE: Valid values=[ENFORCE\_LAYER|ENFORCE\_COLUMN]  RULE\_DESC: Enforce the rule type.  When RULE\_TYPE=ENFORCE\_COLUMN  Enforces which columns must be present in all of the views for a given layer type. Comma-separated list of case-sensative column names.  When RULE\_TYPE=ENFORCE\_LAYER  Enforces which source layer resource can invoke which target layer resource. Comma-separated list of valid LAYER\_TYPES.  If a resource can invoke another resource in the same layer then add its own layer to the list.  KEY: LOAD\_DATE, PROJECT\_NAME\_ID, LAYER\_TYPE, RULE\_TYPE, NODE\_HOST, NODE\_PORT |
| METADATA\_RESOURCE  METADATA\_RESOURCE\_ARCH | This is the core table which all other tables reference. This table contains a row for each TABLE and PROCEDURE resource found within the specified PROJECT\_PATH in the METADATA\_CONST\_NAME table.  KEY: LOAD\_DATE, PROJECT\_NAME\_ID, RESOURCE\_ID, NODE\_HOST, NODE\_PORT |
| METADATA\_RESOURCE\_COLUMN  METADATA\_RESOURCE\_COLUMN\_ARCH | This table contains all of the COLUMNS referenced by the RESOURCE\_ID in METADATA\_RESOURCE.  KEY: LOAD\_DATE, PROJECT\_NAME\_ID, RESOURCE\_ID, COLUMN\_NAME, NODE\_HOST, NODE\_PORT |
| METADATA\_RESOURCE\_LINEAGE  METADATA\_RESOURCE\_LINEAGE\_ARCH | This table contains the lineage for each resource in each layer. This will be a very large table.  KEY: LOAD\_DATE, PROJECT\_NAME\_ID, RESOURCE\_ID, LAYER\_TYPE, LINEAGE\_ORDER, NODE\_HOST, NODE\_PORT |
| METADATA\_DATASOURCE  METADATA\_DATASOURCE\_ARCH | This table contains the all of the datasource information for a given project path.  KEY: LOAD\_DATE, PROJECT\_NAME\_ID, DATASOURCE\_ID, NODE\_HOST, NODE\_PORT |
| METADATA\_NON\_COMPLIANT  METADATA\_NON\_COMPLIANT\_ARCH | This table contains information on column and layer compliancy based on the METADATA\_CONST\_VALIDATE rules tables.  KEY: LOAD\_DATE, PROJECT\_NAME\_ID, RESOURCE\_ID, NON\_COMPLIANT\_REASON, NODE\_HOST, NODE\_PORT |
| METADATA\_POLICY  METADATA\_POLICY\_ARCH | This table contains RBS [rule-based security] and CBS [column-based security] rows for a given project path.  KEY: LOAD\_DATE, PROJECT\_NAME\_ID, POLICY\_ID, NODE\_HOST, NODE\_PORT |
| METADATA\_POLICY\_ASSIGNMNT  METADATA\_POLICY\_ASSIGNMNT\_ARCH | This table contains the assignments for a policy.  KEY: LOAD\_DATE, PROJECT\_NAME\_ID, RESOURCE\_ID, POLICY\_ID, NODE\_HOST, NODE\_PORT |
| METADATA\_PRIVILEGE  METADATA\_PRIVILEGE\_ARCH | This table contains the assigned privileges for all of the resources in a given project path.  KEY: LOAD\_DATE, PROJECT\_NAME\_ID, RESOURCE\_ID, NAME, NAME\_TYPE, DOMAIN\_NAME, USER\_NAME, NODE\_HOST, NODE\_PORT |

### Metadata System Triggers and Load Scripts

Location: /shared/ASAssets/KPImetrics/Physical/Metadata/System

/ClusterSafeCache

/ClusterSafeTriggers

/Helpers

This section provides a quick summary of all triggers, their schedules and how they execute in a cluster.

|  |  |  |  |
| --- | --- | --- | --- |
| **Trigger Name** | **Trigger Schedule** | **Trigger Period** | **Cluster execution** |
| kpimetricsTrig\_40\_Cache\_METADATA\_TABLES | 2:00 AM | 1 day | all nodes |

This section lists all triggers and load scripts that have been defined to execute various KPImetrics procedures at regular intervals. The default execution frequencies are listed for each trigger. The load scripts have been created to load and aggregate raw data into processed KPImetrics metadata.

|  |  |
| --- | --- |
| **Trigger [schedule] 🡪 Script Name 🡪 View name** | **Description** |
| **Schedule: [1 day, 2:00 am]**  **kpimetricsTrig\_40\_Cache\_METADATA\_TABLES 🡪 pRebuildIndexesAdminInterface** 🡪 /shared/ASAssets/KPImetrics/Physical/Metadata/System/ClusterSafeCache/Cache\_METADATA\_TABLES | This trigger executes the Cache\_METADATA\_TABLES procedure. This procedure is used to capture all the metadata for all of the metadata tables.  Exceptions: Emails will be sent if there are exceptions. Review the following view (table) for issues: /services/databases/ASAssets/KPImetrics/workflow/ vCISWorkflowStatus |

### 