opentext™

OpenText™ Project and Portfolio Management (PPM)

Software version: Content Pack 1.0

Vertica for PPM Reporting Customization Guide

Go to Help Center online
https://admhelp.microfocus.com/ppm/

Document release date: May 2020

Send Us Feedback

Let us know how we can improve your experience with the Vertica for PPM Reporting Customization Guide.
Send your email to: admdocteam@opentext.com

Legal Notices

Contents

Introduction

This guide provides step-by-step instructions on how to customize Vertica for OpenText ™ PPM reports. This guide also provides information about the ETL structure and Extract-Transform-Load (ETL) steps, which can help you better understand how the ETL process works.

Vertica for PPM Reports Customization

Read this section for instructions on how to customize the Vertica for PPM reports. After reading this guide, you can:

- Add new fact entity
- Add new dimension entity
- Add attributes for existing entities
- Change ETL rules

Read the following sections for details:

- "Before You Customize Reports" below
- "Customizing Vertica for PPM Reports" on page 11

Before You Customize Reports

Before you start to customize Vertica for PPM reports, make sure the following environments are available:

- Vertica database cluster
- Vertica for PPM content pack

  For instructions on how to install Vertica for PPM content pack, refer to the *Vertica for PPM Administrator Guide for Content Pack 1.0*.

You also need to have ETL hands-on experience before customization.

Note:

- To test the customization process, it is suggested that you run the sample content packs on a test environment first.
- Do not use the production environment for testing or content development because content packs cannot be uninstalled.

You also need to obtain the sample content packs and understand the content pack structures by reading the following sections:

Obtaining Sample Content Packs for Customization

To obtain the sample content packs, follow these steps:

1. Go to Operational Reports Content for Project and Portfolio Management - Downloads page.
2. Select **Vertica for PPM 1.0**.
3. Download the **Vertica_Reporting_Customization_Samples.zip** file.
4. Extract the entire contents of **Vertica_Reporting_Customization_Samples.zip** to your local drive.

Content Pack Structure

Generally speaking, two content pack (.cp) folders are needed under the *<VDW_HOME>*`/Content` directory for a single task:

- `CUSTOMIZATION_PPM.cp`: Contains JSON files that define source entities, extraction entities, and stream entities.
- `CUSTOMIZATION_TARGET.cp`: Contains JSON files that define target entities.

Note: When deploying content packs, you always need to deploy `CUSTOMIZATION_TARGET.cp` first. Otherwise, you may not be able to deploy `CUSTOMIZATION_PPM.cp` and process the rest of tasks.

The folder structure of the content packs is shown as follows:

- CUSTOMIZATION_PPM.cp (root folder)
  - dwmetadata (folder)
    - entities (folder)
      - Source entity JSON (file)

    - streams (folder)
      - Stream entity JSON (file)
  - extmetadata (folder)
    - Extractor entity JSON (file)
  - cp.json (file)
- CUSTOMIZATION_TARGET.cp (root folder)
  - dwmetadata (folder)
    - entities (folder)
      - Target entity JSON (file)
  - cp.json (file)

The following table describes the JSON files.

JSON file	Description
Target entity JSON	Stores information related to the target table, such as field names and relation information with other tables.
Source entity JSON	Stores information related to the source table such as field names and incremental extraction information.
Stream entity JSON	Stores information related to the stream, such as the source entity, target entity, and SSI transforming SQLs.
Extractor entity JSON	Stores information related to data extraction, such as SQLs for extracting data.
CP JSON	Stores information related to content packs, such as target entities and source entities. This file is the entrance for the corresponding content pack.

Running Sample Content Packs

Before you customize reports, it is suggested you run the following content pack examples in your test environment.

1. Run the following command on Linux to ensure that $VDW_HOME is configured correctly:

```
Echo $VDW_HOME
```

If $VDW_HOME is correctly configured, the directory that you have Vertica for PPM content pack installed returns. For example,

```
/VDW_HOME
```

Otherwise, add $VDW_HOME as a system environment variable and point it to the Vertica for PPM content pack directory.

2. Unpack the `Vertica_Reporting_Customization_Samples.zip` file.

   Two content pack folders are included: `CUSTOMIZATION_PPM.cp` and `CUSTOMIZATION_TARGET.cp`.

   For instructions on obtaining the sample content packs, see "Obtaining Sample Content Packs for Customization" on page 7.

3. Place the `CUSTOMIZATION_PPM.cp` and `CUSTOMIZATION_TARGET.cp` files in the *<VDW_HOME>*`/Content` directory.

4. Run the `ContentManager.sh` script under the *<VDW_HOME>*`/bin` directory to deploy the `CUSTOMIZATION_TARGET.cp`:

   ```
   sh ContentManager.sh --instruction install --cpname CUSTOMIZATION_TARGET;
   ```

   You can find the following message from `ContentManager.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   ```
   The content of package CUSTOMIZATION_TARGET was successfully installed
   ```

   After you run the command, the following tables are generated:

   - The DIM_ CUSTOMIZATION_CONTACTS table is generated for extending dimension entities.
   - The FACT_CUSTOMIZATION_RESOURCE_DEMAND table is generated for extending fact entities.
   - The DIM_CUSTOMIZATION_TIME_SHEETS table is generated for creating new dimension entities.
   - The FACT_ CUSTOMIZATION_TIME_ACTUALS table is generated for creating new fact entities.

5. Run the `ContentManager.sh` script under the *<VDW_HOME>*`/bin` directory to deploy `CUSTOMIZATION_PPM.cp`:

   ```
   sh ContentManager.sh --instruction install --cpname CUSTOMIZATION_PPM;
   ```

   You can find the following message from `ContentManager.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   ```
   The content of package CUSTOMIZATION_PPM was successfully installed
   ```

6. Run the `ExtractorEngine.sh` script under the *<VDW_HOME>*`/bin` directory to extract data from the PPM database to flat files:

   `sh ExtractorEngine.sh --streamname` *<Stream_Name>* `--instancename` *<PPM_Instance_Name>*

   You can find the following message from `ExtractorEngine.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   `Extractor was successfully executed. The BATCH ID is:` *<Batch_ID>*.

   You also need to make sure that no warning messages are displayed in the command line interface.

   - Replace *<Stream_Name>* with the following depending on the sample content packs you use:
     - `CUSTOMIZATION_CONTACTS_STREAM` for extended dimension entities
     - `CUSTOMIZATION_RESOURCE_DEMAND_STREAM` for extended fact entities
     - `CUSTOMIZATION_TIME_SHEETS_STREAM` for new dimension entities
     - `CUSTOMIZATION_TIME_ACTUALS_STREAM` for new fact entities
   - *<PPM_Instance_Name>* is the PPM instance name you specified when installing the Vertica for PPM content pack.

7. Run the `FlowEngine.sh` script under the *<VDW_HOME>*`/bin` directory to process ETL:

   `sh FlowEngine.sh --batch` *<Batch_ID>* `--streamname` *<Stream_Name>* `--instancename` *<PPM_Instance_Name>*

   You can find the following message from `FlowEngine.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

```
ETL process was executed successfully
```

- *<Batch_ID>* used in this command is the batch ID that was generated in Step 6.
- Replace *<Stream_Name>* with the following depending on the sample content packs you use:
  - CUSTOMIZATION_CONTACTS_STREAM for extending dimension entities
  - CUSTOMIZATION_RESOURCE_DEMAND_STREAM for extending fact entities
  - CUSTOMIZATION_TIME_SHEETS_STREAM for new dimension entities
  - CUSTOMIZATION_TIME_ACTUALS_STREAM for new fact entities

8. Connect to the Vertica database and check whether the data has been loaded successfully:

   `select * from` *<Target_Schema>.<Table_Name>*

   - *<Target_Schema>* is the name for the schema that contains target data and tables for reporting.
   - Replace *<Table_Name>* with the following for different sample content packs:
     - DIM_ CUSTOMIZATION_CONTACTS for extended dimension entities
     - FACT_CUSTOMIZATION_RESOURCE_DEMAND for extended fact entities
     - DIM_CUSTOMIZATION_TIME_SHEETS for new dimension entities
     - FACT_CUSTOMIZATION_TIME_ACTUALS for new fact entities

For more information about these scripts, refer to "Administration Tasks" of the *Vertica for PPM Administrator Guide for Content Pack 1.0*.

Customizing Vertica for PPM Reports

This section provides instructions on customizing Vertica for PPM reports:

- "Extending or Creating Entities" on the next page
- "Customizing ETL Rules" on page 35

Extending or Creating Entities

You can extend existing entities or create new entities for both dimension and fact tables. This section shows you how to customize the entities step by step.

For examples of how to extend or create dimension and fact entities, see "Scenarios of Extending or Creating Entities" on page 19.

Note: It is suggested to run the sample content packs in your test environment before extending entities. See "Running Sample Content Packs " on page 8 for details.

Defining Content Packs

To extend or create entities, you need to define content packs by following these steps:

1. Create two content pack folders: `CUSTOMIZATION_PPM.cp` and `CUSTOMIZATION_TARGET.cp`.
2. Create folders under these two content packs. For detailed structures, see "Content Pack Structure " on page 7.
3. Under `CUSTOMIZATION_TARGET.cp`, do the following:
   a. Under the `dwmetadata\entities` directory, create and define the target entity JSON file according to the sample content packs at HP Live Network. For more information for obtaining the sample content packs, see "Obtaining Sample Content Packs for Customization" on page 7.

      Note the following when defining the target entity JSON file:

      - For details about the attributes in this JSON file, see "Entity Attribute Descriptions" on page 55.
      - The **entity_name** attribute should be unique because the table name is generated according to the entity name.
      - The **segmented_by** attribute is used for cluster segmentation. See the Vertica documentation for details.

- The **schema** attribute includes the new attributes:
  - **attribute_name**: The attribute name.
  - **scd** (for dimension tables only): Determines whether to update the record (`scd1`) or to insert a new record (`scd2`).
- To extend OOTB entities, you need to define the ***_associated_*** attribute:
  - **lookup_entity_name**: The name of the entity that needs to be extended.
  - **role_entity_name**: Includes `CUSTOMIZATION_` as the prefix and `_EXTEND` as the suffix; for example, `CUSTOMIZATION_CONTACTS_EXTEND`.
- ***_associated_*** is required for extended entities while optional for new entities. Refer to the following table for details.

Attribute Description

Attribute Type	Description	Required for	Optional for
dimension_associated_dimension	Used to define the reference from a dimension table to another dimension table (for example, snowflake model). For an extended dimension entity, this attribute is used to specify which entity this table is extended from.	Extended dimension tables	New dimension tables
fact_associated_dimension	Used to define the reference from a fact table to a dimension table.	Extended fact tables; Use only one of the attrbutes for a single entity	New fact tables
fact_associated_fact	Used to define the reference from a fact table to another fact table. For an extended fact entity, this attribute is used to specify which entity this table is extended from.		(Not supported for new fact tables)

b. Create and define `cp.json` according to the samples provided at HP Live Network. For more information for obtaining the sample content packs, see "Obtaining Sample Content Packs for Customization" on page 7.

   Note the following when defining `cp.json`:

   - The value of **content_pack_name** should be the same as defined in **content_pack** of the target entity.
   - `cp.json` in `CUSTOMIZATION_TARGET.cp` must have **target_entities** defined.
   - For detailed descriptions of the attributes in this JSON file, see "Entity Attribute Descriptions" on page 55

   After you complete this step, the target entity JSON file is defined.

4. In `CUSTOMIZATION_PPM.cp`, do the following:

   a. Under the `dwmetadata\entities` directory, create and define the source entity JSON file according to the samples provided at HP Live Network. For more information, see "Obtaining Sample Content Packs for Customization" on page 7.

      When defining the source entity JSON file, note the following:

      - **schema**: Includes the attribute definitions for the source entity.
      - **attribute_name**: Defines the attribute name.
      - For detailed descriptions of the attributes in this JSON file, see "Entity Attribute Descriptions" on page 55.

      After you complete this step, the source entity JSON file is defined.

   b. Under `dwmetadata\streams`, create and define the stream entity JSON file according to the samples provided at HP Live Network. For more information, see "Obtaining Sample Content Packs for Customization" on page 7.

      When defining the stream entity JSON file, note the following:

      - **content_pack**: Aligns with **content_pack** defined in the source entity JSON file.
      - **stream_name**: Includes `CUSTOMIZATION_` as the prefix.

- **source_entities_includes**: Includes the source entity of the stream; for example, `CUSTOMIZATION_KCRT_CONTACTS`.
- **target_entities_includes**: Includes the target entity of the stream; for example, `CUSTOMIZATION_CONTACTS`.
- **transforms**: see "ETL Step 2: SSI" on page 45for a definition.
- **post_target_transforms**: Optional. If you want to process other SQLs after data is loaded to the target table, include SQLs in this attribute. For more information, see "ETL Step 10: POSTTARGET" on page 53.
- For detailed descriptions of the attributes in this JSON file, see "Entity Attribute Descriptions" on page 55.

After you complete this step, the stream entity JSON file is defined.

c. Under `extmetadata`, create and define the extractor entity JSON file according to the samples provided at HP Live Network. For more information, see "Obtaining Sample Content Packs for Customization" on page 7.

When defining the extractor entity JSON file, note the following:

- **content_pack**: Aligns with **content_pack** defined in the stream entity and source entity.
- **entity_name**: Includes `CUSTOMIZATION_` as the prefix and `_EXT` as the suffix.
- **source_entity_name**: Defines the source table name.
- **extraction_view**: Selects the attributes defined in the source entity JSON file.
- For detailed descriptions of the attributes in this JSON file, see "Entity Attribute Descriptions" on page 55.

After you complete this step, the extractor entity JSON file is defined.

d. Create and define `cp.json` according to the samples provided at HP Live Network. For more information, see "Obtaining Sample Content Packs for Customization" on page 7.

When defining `cp.json`, note the following:

- **content_pack_name** should be unique and align with **content_pack** defined in stream entity, source entity, and extractor entity.
- `cp.json` in `CUSTOMIZATION_PPM.cp` must have the **streams**, **source_entities**, and **extraction_entities** attributes defined.
- For detailed descriptions of the attributes in this JSON file, see "Entity Attribute Descriptions" on page 55.

Deploying Content Packs

To deploy content packs, follow these steps:

1. Run the following command on Linux to ensure that $VDW_HOME is configured correctly:

   ```
   Echo $VDW_HOME
   ```

   If $VDW_HOME is correctly configured, the directory that you have Vertica for PPM content pack installed returns. For example,

   ```
   /VDW_HOME
   ```

   Otherwise, add $VDW_HOME as a system environment variable and point it to the Vertica for PPM content pack directory.
2. Place `CUSTOMIZATION_PPM.cp` and `CUSTOMIZATION_TARGET.cp` under *<VDW_HOME>*`/Content`.
3. Run the `ContentManager.sh` script under the *<VDW_HOME>*`/bin` directory to deploy `CUSTOMIZATION_TARGET.cp`:

   ```
   sh ContentManager.sh --instruction install --cpname CUSTOMIZATION_TARGET;
   ```

   You can find the following message from `ContentManager.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   ```
   The content of package CUSTOMIZATION_TARGET was successfully installed
   ```

   The DIM_ CUSTOMIZATION_CONTACTS table is generated in the Vertica database.
4. Run the `ContentManager.sh` script under the *<VDW_HOME>*`/bin` directory to deploy `CUSTOMIZATION_PPM.cp`:

   ```
   sh ContentManager.sh --instruction install --cpname CUSTOMIZATION_PPM;
   ```

You can find the following message from `ContentManager.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

```
The content of package CUSTOMIZATION_PPM was successfully installed
```

5. Run the `ExtractorEngine.sh` script under the *<VDW_HOME>*`/bin` directory to extract data from the PPM database to flat files:

   `sh ExtractorEngine.sh --streamname` *<Stream_Entity_Name>* `--instancename` *<PPM_Instance_Name>*

   You can find the following message from `ExtractorEngine.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   `Extractor was successfully executed. The BATCH ID is:` *<Batch_ID>*.

   - *<Stream_Entity_Name>*: Should be the same as defined in **stream_name** of the stream entity JSON file.
   - *<PPM_Instance_Name>*: The PPM instance name you specified when installing the Vertica for PPM content pack.

6. Run the `FlowEngine.sh` script under the *<VDW_HOME>*`/bin` directory to process ETL:

   `sh FlowEngine.sh --batch` *<Batch_ID>* `--streamname` *<Stream_Entity_Name>* `--instancename` *<PPM_Instance_Name>*

   You can find the following message from `FlowEngine.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   ```
   ETL process was executed successfully
   ```

   - *<Batch_ID>*: The batch ID that was generated in Step 5.
   - *<Stream_Entity_Name>*: Should be the same as defined in **stream_name** of the stream entity JSON file.
   - *<PPM_Instance_Name>*: The PPM instance name you specified when installing the Vertica for PPM content pack.

7. Connect to the Vertica database and check whether the data has been loaded successfully:

   `select * from` *<Target_Schema>*`.DIM_`*<Target_Entity_Name>*

Or

`select * from <Target_Schema>.FACT_<Target_Entity_Name>`

- `<Target_Schema>`: The name for the schema that contains target data and tables for reporting.
- `<Target_Entity_Name>`: Should be the same as defined in **entity_name** of the target entity JSON file.

To find the new target tables in views, you need to manually update the views in the Vertica database with the following SQL queries:

```
CREATE OR REPLACE VIEW <vdwtarget_schema>.<customized target table>_V
AS
SELECT <OOTB target table>.*, <customized target table>.<extended field> as
extended_field
FROM <vdwtarget_schema>.<CUSTOMIZED target table> right join <vdwtarget_
schema>.<OOTB target table> on
<vdwtarget_schema>.<CUSTOMIZED target table>.MD_ENTERPRISE_KEY =
<vdwtarget_schema>.<OOTB target table>.MD_ENTERPRISE_KEY;
```

Replace the following variables:

- `<vdwtarget_schema>`: The schema that contains the target tables
- `<customized target table>`: The target table that is created
- `<OOTB target table>`: The OOTB target table that is customized
- `<extended field>`: The field that is to be extended

You can find all views on specified tables in the *OpenText PPM 9.50 Data Model Guide*.

Note the following when deploying the content packs:

- Always deploy `CUSTOMIZATION_TARGET.cp` before deploying `CUSTOMIZATION_PPM.cp`.
- If you change the content packs after deployment, you need to run `ContentManager.sh` again to make the changes effective.
- You need to create a Shell to call the Extractor Engine and Flow Engine, and run ETL on a regular basis with `crontab`. You also need to make sure the script can run after the `vdwetljob.sh` process completes. For how to create the Shell script, you can take `vdwCustomizationEtlJob.sh` that is included in the sample package as an

example.

For instructions on obtaining the sample package, see "Obtaining Sample Content Packs for Customization" on page 7.

For more information about these scripts, refer to "Administration Tasks" of the *Vertica for PPM Administrator Guide for Content Pack 1.0*.

Scenarios of Extending or Creating Entities

You can better understand the procedure of report customization by reading the following scenarios:

- "Scenario: Extending Dimension Entities" below
- "Scenario: Creating New Dimension Entities" on page 27

By leveraging these scenarios, you can also do the following with the sample content packs that are provided on HP Live Network:

- Extending fact entities
- Creating fact entities

For more information for obtaining the sample content packs, see "Obtaining Sample Content Packs for Customization" on page 7.

Scenario: Extending Dimension Entities

If you want to extend dimension entities for the KCRT_CONTACTS table, follow these steps. The **USER_ID** and **ENABLE_FLAG** fields are to be added.

Defining Content Packs

To define content packs, follow these steps:

1. Create two content pack folders: `CUSTOMIZATION_PPM.cp` and `CUSTOMIZATION_TARGET.cp`.
2. Create folders under these two content packs. For detailed structures, see "Content Pack Structure " on page 7.
3. Under `CUSTOMIZATION_TARGET.cp`, do the following:

a. Under the `dwmetadata\entities` directory, create and define the target entity JSON file `CUSTOMIZATION_CONTACTS.json` by copying the following to the file:

```
{
	"metadata_layout_version":"1.0",
	"version":"1.0",
	"content_pack":"CUSTOMIZATION_TARGET",
	"entity_name":"CUSTOMIZATION_CONTACTS",
	"entity_business_name":"CUSTOMIZATION CONTACTS",
	"entity_description":"CUSTOMIZATION CONTACTS",
	"dimension":{
		"dimension_business_name":"CUSTOMIZATION CONTACTS",
		"is_conformed":"true",
		"dimension_type":"Primary",
		"storage_strategy":{
			 "segmented_by":"default",
				"partition_by":"na"
		}
	},
	"schema":[
		{
			"attribute":"USER_ID",
			"attribute_name":"USER_ID",
			"attribute_business_name":"USER ID",
			"attribute_description":"USER ID",
			"attribute_type":"dimension",
			"scd":"scd1",
			"target_data_type":"int",
			"size":"na",
			"is_required":"false"
		},
		{
			"attribute":"ENABLED_FLAG",
			"attribute_name":"ENABLED_FLAG",
			"attribute_business_name":"ENABLED FLAG",
			"attribute_description":"ENABLED FLAG",
			"attribute_type":"dimension",
			"scd":"scd2",
			"target_data_type":"varchar",
			"size":"1",
			"is_required":"false"
		}
	],
    "dimension_associated_dimension":[
		{
			"lookup_entity_name":"Contacts",
			"role_entity_name":"CUSTOMIZATION_CONTACTS_EXTEND",
			"role_entity_business_name":"Contacts",
			"description":"Contacts"
```

```
                }
        ]
}

    "dimension_associated_dimension":[
                {
                        "lookup_entity_name":"Contacts",
                        "role_entity_name":"Extends_Contacts",
                        "role_entity_business_name":"Contacts",
                        "description":"Contacts"
                }
        ]
}
```

`CUSTOMIZATION_CONTACTS.json` defines:

- Two fields in the target table: **USER_ID** and **ENABLED_FLAG**
- The target entity that is to be extended: `Contacts`

After you complete this step, the target entity JSON file is defined.

b. Create and define `cp.json` by copying the following to the file:

```
{
       "metadata_layout_version": "1.0",
       "content_pack_name": "CUSTOMIZATION_TARGET",
       "version": "1.0",
       "description": "Project Management & Portfolio Management Content Pack,
Shared Entities.",
       "require": {
               "platform": ">=1.0.0"
       },
       "target_entities": [
               {
                       "name": "CUSTOMIZATION_CONTACTS"
               }
       ]
}
```

`cp.json` defines:

- The content pack name: `CUSTOMIZATION_TARGET`
- The target entity as defined in `CUSTOMIZATION_CONTACTS.json`: `CUSTOMIZATION_CONTACTS`

After you complete this step, the `cp.json` file is defined.

4. In `CUSTOMIZATION_PPM.cp`, do the following:

a. Under the `dwmetadata\entities` directory, create and define the source entity JSON file `CUSTOMIZATION_KCRT_CONTACTS.json`:

```
{
    "metadata_layout_version": "1.0",
    "version": "1.0",
    "content_pack": "CUSTOMIZATION_PPM",
    "source_product": "PPM",
    "entity_name": "CUSTOMIZATION_KCRT_CONTACTS",
    "entity_business_name": "PPM contacts table with customized field",
    "entity_description": "Contacts info with customized field",
    "schema": [
        {
            "attribute": "CONTACT_ID",
            "attribute_name": "CONTACT_ID",
            "attribute_business_name": "Contact ID",
            "attribute_description": "Contact ID",
            "sql_data_type":"INT",
            "size": "na",
            "is_bk": "true",
            "is_cdc": "false",
            "is_required": "true",
            "column_sequence": "1"
        },
               {
            "attribute": "USER_ID",
            "attribute_name": "USER_ID",
            "attribute_business_name": "User Id",
            "attribute_description": "User ID of the contact",
            "sql_data_type":"INT",
            "size": "na",
            "is_bk": "false",
            "is_cdc": "false",
            "is_required": "false",
            "column_sequence": "2"
        },
        {
            "attribute": "ENABLED_FLAG",
            "attribute_name": "ENABLED_FLAG",
            "attribute_business_name": "ENABLED FLAG",
            "attribute_description": "ENABLED FLAG",
            "sql_data_type":"VARCHAR",
            "size": "1",
            "is_bk": "false",
            "is_cdc": "false",
            "is_required": "false",
            "column_sequence": "3"
        }
    ]
}
```

`CUSTOMIZATION_KCRT_CONTACTS.json` defines:

- Three fields in the source table: **CONTACT_ID** and **ENABLED_FLAG** fields that are extended; **USER_ID** that is the primary key of the source table
- The source entity name: `CUSTOMIZATION_KCRT_CONTACTS`

After you complete this step, the source entity JSON file is defined.

b. Under `dwmetadata\streams`, create and define the stream entity JSON file `CUSTOMIZATION_CONTACTS_STREAM.json`:

```
{
	"metadata_layout_version":"1.0",
	"version":"1.0",
	"content_pack":"CUSTOMIZATION_PPM",
	"source_product":"PPM",
	"stream_name":"CUSTOMIZATION_CONTACTS_STREAM",
	"source_entities_includes":[
		{
			"source_entity_include":"CUSTOMIZATION_KCRT_CONTACTS"
		}
	],
	"target_entities_includes":[
		{
			"target_entity_include":"CUSTOMIZATION_CONTACTS"
		}
	],
	"transforms":[
		{
		"sql": "drop table if exists PPM_CUSTOMIZATION_CONTACTS_${MD_SOURCE_
INSTANCE_ID}_SSI CASCADE;
--==--==--
create table PPM_CUSTOMIZATION_CONTACTS_${MD_SOURCE_INSTANCE_ID}_SSI (
		MD_BUSINESS_KEY                         varchar(1000),
		CUSTOMIZATION_CONTACTS_EXTEND_BUSINESS_KEY     varchar(1000),
		CUSTOMIZATION_CONTACTS_EXTEND_ENTERPRISE_KEY     INT,
		USER_ID   INT,
				ENABLED_FLAG      varchar(1),

		MD_BATCH_ID                INT,
		MD_PROCESS_ID              INT,
		MD_SOURCE_INSTANCE_ID      INT,
		MD_FLAG               varchar(10)
);
--==--==--
insert into PPM_CUSTOMIZATION_CONTACTS_${MD_SOURCE_INSTANCE_ID}_SSI (
		MD_BUSINESS_KEY,
		CUSTOMIZATION_CONTACTS_EXTEND_BUSINESS_KEY,
```

```
                USER_ID,
                                ENABLED_FLAG,
                MD_BATCH_ID ,
                MD_PROCESS_ID ,
                MD_SOURCE_INSTANCE_ID ,
                MD_FLAG
)
select
                tab.MD_BUSINESS_KEY,
                tab.CUSTOMIZATION_CONTACTS_EXTEND_BUSINESS_KEY,
                tab.USER_ID,
                                tab.ENABLED_FLAG,
                ${MD_BATCH_ID} AS MD_BATCH_ID ,
                ${MD_PROCESS_ID} AS MD_PROCESS_ID ,
                ${MD_SOURCE_INSTANCE_ID} AS MD_SOURCE_INSTANCE_ID ,
                tab.MD_FLAG
from (
                select
                                t1.md_source_instance_id || ':' || t1.md_
business_key as MD_BUSINESS_KEY ,
       t1.md_source_instance_id || ':' || t1.CONTACT_ID as CUSTOMIZATION_CONTACTS_
EXTEND_BUSINESS_KEY,
                                USER_ID,
                                ENABLED_FLAG,

                                decode(t1.md_
flag,'NEW','NEW','DEL','DEL','UPD') as md_flag,
                                row_number() over( partition by t1.md_source_
instance_id || ':' || t1.md_business_key) multi_flag
                from CUSTOMIZATION_CONTACTS_STREAM_CUSTOMIZATION_KCRT_
CONTACTS_${MD_SOURCE_INSTANCE_ID}_EXT t1
                where t1.md_pf_flag = 'D') tab
where tab.multi_flag = 1;
--==--==--
       SELECT ANALYZE_STATISTICS('PPM_CUSTOMIZATION_CONTACTS_${MD_SOURCE_INSTANCE_
ID}_SSI');
--==--==--
"
               }
       ]
}
```

`CUSTOMIZATION_CONTACTS_STREAM.json` defines:

- Source entities that provide data
- Target entities that accept data
- SQLs mainly to transform data from source entities to target entities

After you complete this step, the stream entity JSON file is defined.

c. Under `extmetadata`, create and define the extractor entity JSON file `CUSTOMIZATION_KCRT_CONTACTS_EXT.json`:

```
{
	"metadata_layout_version":"1.0",
	"version":"1.0",
	"source_product":"PPM",
	"content_pack": "CUSTOMIZATION_PPM",
	"entity_name":"CUSTOMIZATION_KCRT_CONTACTS_EXT",
	"source_entity_name":"KCRT_CONTACTS",
	"extractor" : "OracleDBExtractor",
	"extraction":[
		{
		"extraction_view" : "SELECT contact_id, user_id, ENABLED_FLAG FROM KCRT_
CONTACTS",
		"source_product_version" : "9.30"
		}
	]
}
```

`CUSTOMIZATION_KCRT_CONTACTS_EXT.json` defines:

- The extractor entity name: `CUSTOMIZATION_KCRT_CONTACTS_EXT`
- **source_entity_name** as the source table name
- **extraction_view**: SQLs for extracting data

After you complete this step, the extractor entity JSON file is defined.

d. Create and define `cp.json`:

```
{
	"metadata_layout_version": "1.0",
	"content_pack_name": "CUSTOMIZATION_PPM",
	"source_product": "PPM",
	"version": "1.0",
	"description": "Project Management & Portfolio Management Content Pack,
Extend Entities.",
	"require": {
	"platform": ">=1.0.0",
		"cp": [
			{
				"name": "CUSTOMIZATION_TARGET",
				"version": ">=1.0.0"
			}
		]
	},
	"streams": [
		{
			"name": "CUSTOMIZATION_CONTACTS_STREAM"
```

```
                }
        ],
        "source_entities": [
                {
                        "name": "CUSTOMIZATION_KCRT_CONTACTS"
                }
        ],
        "extraction_entities": [
                {
                        "name": "CUSTOMIZATION_KCRT_CONTACTS_EXT"
                }
        ]
}
```

`cp.json` defines:

- The content pack name: `CUSTOMIZATION_PPM`
- Source entities: Defined in `CUSTOMIZATION_KCRT_CONTACTS.json`
- Stream entities: Defined in `CUSTOMIZATION_CONTACTS_STREAM.json`
- Extract entities: Defined in `CUSTOMIZATION_KCRT_CONTACTS_EXT.json`

Deploying Content Packs

To deploy content packs, follow these steps:

1. Place `CUSTOMIZATION_PPM.cp` and `CUSTOMIZATION_TARGET.cp` under *<VDW_HOME>*`/Content`.
2. Run the `ContentManager.sh` script under the *<VDW_HOME>*`/bin` directory to deploy `CUSTOMIZATION_TARGET.cp`:

   ```
   sh ContentManager.sh --instruction install --cpname CUSTOMIZATION_TARGET;
   ```

   You can find the following message from `ContentManager.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   ```
   The content of package CUSTOMIZATION_TARGET was successfully installed
   ```

   The DIM_ CUSTOMIZATION_CONTACTS table is generated in the Vertica database.
3. Run the `ContentManager.sh` script under the *<VDW_HOME>*`/bin` directory to deploy `CUSTOMIZATION_PPM.cp`:

   ```
   sh ContentManager.sh --instruction install --cpname CUSTOMIZATION_PPM;
   ```

You can find the following message from `ContentManager.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

```
The content of package CUSTOMIZATION_PPM was successfully installed
```

4. Run the `ExtractorEngine.sh` script under the *<VDW_HOME>*`/bin` directory to extract data from the PPM database to flat files:

   ```
   sh ExtractorEngine.sh --streamname CUSTOMIZATION_CONTACTS_STREAM --instancename <PPM_Instance_Name>
   ```

   You can find the following message from `ExtractorEngine.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   ```
   Extractor was successfully executed. The BATCH ID is: <Batch_ID>.
   ```

   <PPM_Instance_Name> is the PPM instance name you specified when installing the Vertica for PPM Content Pack.

5. Run the `FlowEngine.sh` script under the *<VDW_HOME>*`/bin` directory to process ETL:

   ```
   sh FlowEngine.sh --batch <Batch_ID> --streamname CUSTOMIZATION_CONTACTS_STREAM --instancename <PPM_Instance_Name>
   ```

   You can find the following message from `FlowEngine.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   ```
   ETL process was executed successfully
   ```

   <Batch_ID> used in this command is the batch ID that was generated in Step 4.

6. Connect to the Vertica database and check whether the data has been loaded successfully:

   ```
   select * from <Target_Schema>.DIM_CUSTOMIZATION_CONTACTS
   ```

   <Target_Schema> is the name for the schema that contains target data and tables for reporting.

Scenario: Creating New Dimension Entities

If you want to create new dimension entities for the TM_TIME_SHEETS table, follow these steps. The **TIME_SHEET_ID** and **DESCRIPTION** fields are to be added.

Defining Content Packs

To define content packs, follow these steps:

1. Create two content pack folders: `CUSTOMIZATION_PPM.cp` and `CUSTOMIZATION_TARGET.cp`.
2. Create folders under these two content packs. For detailed structures, see "Content Pack Structure " on page 7.
3. Under `CUSTOMIZATION_TARGET.cp`, do the following:
   a. Under the `dwmetadata\entities` directory, create and define the target entity JSON file `CUSTOMIZATION_TIME_SHEETS.json` by copying the following to the file:

```
{
    "metadata_layout_version": "1.0",
    "version": "1.0",
    "content_pack": "CUSTOMIZATION_TARGET",
    "entity_name": "CUSTOMIZATION_TIME_SHEETS",
    "entity_business_name": "Customization TIME SHEETS",
    "entity_description": "Customization TIME SHEETS",
       "dimension":{
               "dimension_business_name":"Customization time sheets",
               "is_conformed":"true",
               "dimension_type":"primary",
               "storage_strategy":{
                       "segmented_by":"default",
                       "partition_by":"na"
               }
       },
    "schema": [
           {
            "attribute": "TIME_SHEET_ID",
            "attribute_name": "TIME_SHEET_ID",
            "attribute_business_name": "TIME SHEET ID",
            "attribute_description": "TIME SHEET ID",
            "attribute_type": "dimension",
            "scd": "scd1",
            "target_data_type": "INT",
            "size": "na",
            "is_required": "false"
        },
           {
            "attribute": "DESCRIPTION",
            "attribute_name": "Description",
            "attribute_business_name": "Customization Description",
            "attribute_description": "Customization Description",
```

```
            "attribute_type": "dimension",
            "scd": "scd1",
            "target_data_type": "varchar",
            "size": "650",
            "is_required": "false"
        }
    ]
}
```

`CUSTOMIZATION_TIME_SHEETS.json` defines:

- Two fields in the target table: **TIME_SHEET_ID** and **DESCRIPTION**.

After you complete this step, the target entity JSON file is defined.

b. Create and define `cp.json` by copying the following to the file. The value of **content_pack_name** should be the same value defined in **content_pack** in step 3a:

```
{
       "metadata_layout_version": "1.0",
       "content_pack_name": "CUSTOMIZATION_TARGET",
       "version": "1.0",
       "description": "Project Management & Portfolio Management Content Pack,
Shared Entities.",
       "require": {
              "platform": ">=1.0.0"
       },
       "target_entities": [
              {
                     "name": "CUSTOMIZATION_TIME_SHEETS"
              }
       ]
}
```

`cp.json` defines:

- The target entity `CUSTOMIZATION_TIME_SHEETS`, as defined in `CUSTOMIZATION_TIME_SHEETS.json`
- The content pack name: `CUSTOMIZATION_TARGET`

After you complete this step, the `cp.json` file is defined.

4. In `CUSTOMIZATION_PPM.cp`, do the following:

a. Under the `dwmetadata\entities` directory, create and define the source entity JSON file `CUSTOMIZATION_TM_TIME_SHEETS.json`:

```
{
    "metadata_layout_version": "1.0",
    "version": "1.0",
    "content_pack": "CUSTOMIZATION_PPM",
    "source_product": "PPM",
    "entity_name": "CUSTOMIZATION_TM_TIME_SHEETS",
    "entity_business_name": "Customization PPM TIME SHEETS table",
    "entity_description": "Customization TIME SHEETS info",
    "schema": [
        {
            "attribute": "TIME_SHEET_ID",
            "attribute_name": "TIME_SHEET_ID",
            "attribute_business_name": "Time sheet id",
            "attribute_description": "Time sheet id",
            "sql_data_type": "INT",
            "size": "na",
            "is_bk": "true",
            "is_cdc": "false",
            "is_required": "true",
            "column_sequence": "1"
        },
        {
            "attribute": "DESCRIPTION",
            "attribute_name": "DESCRIPTION",
            "attribute_business_name": "Description",
            "attribute_description": "Description",
            "sql_data_type": "VARCHAR",
            "size": "650",
            "is_bk": "false",
            "is_cdc": "false",
            "is_required": "false",
            "column_sequence": "2"
        }
    ]
}
```

`CUSTOMIZATION_TM_TIME_SHEETS.json` defines:

- Two fields in the source table: **TIME_SHEET_ID** and **DESCRIPTION**
- The source entity name: `CUSTOMIZATION_TM_TIME_SHEETS`

After you complete this step, the source entity JSON file is defined.

b. Under `dwmetadata\streams`, create and define the stream entity JSON file `CUSTOMIZATION_TIME_SHEETS_STREAM.json`:

```
{
       "metadata_layout_version":"1.0",
       "version":"1.0",
       "content_pack":"CUSTOMIZATION_PPM",
```

```
        "source_product":"PPM",
        "stream_name":"CUSTOMIZATION_TIME_SHEETS_STREAM",
        "source_entities_includes":[
                {
                "source_entity_include":"CUSTOMIZATION_TM_TIME_SHEETS"
                }
        ],
        "target_entities_includes":[
                {
                "target_entity_include":"CUSTOMIZATION_TIME_SHEETS"
                }
        ],
        "transforms":[
                {
                "sql": "drop table if exists PPM_CUSTOMIZATION_TIME_SHEETS_${MD_SOURCE_
INSTANCE_ID}_SSI CASCADE;
--==--==--
create table PPM_CUSTOMIZATION_TIME_SHEETS_${MD_SOURCE_INSTANCE_ID}_SSI (
                MD_BUSINESS_KEY                         varchar(1000),
                        CUSTOMIZATION_TIME_SHEETS_EXTEND_BUSINESS_KEY     varchar(1000),
                CUSTOMIZATION_TIME_SHEETS_EXTEND_ENTERPRISE_KEY     INT,
                                DESCRIPTION     varchar(650),


                MD_BATCH_ID                INT,
                MD_PROCESS_ID              INT,
                MD_SOURCE_INSTANCE_ID      INT,
                MD_FLAG                varchar(10)
) unsegmented all nodes;
--==--==--
insert into PPM_CUSTOMIZATION_TIME_SHEETS_${MD_SOURCE_INSTANCE_ID}_SSI (
                MD_BUSINESS_KEY        ,
                        CUSTOMIZATION_TIME_SHEETS_EXTEND_BUSINESS_KEY,
                        DESCRIPTION,
                MD_BATCH_ID ,
                MD_PROCESS_ID ,
                MD_SOURCE_INSTANCE_ID ,
                MD_FLAG
)
select
                        tab.MD_BUSINESS_KEY     ,
                        tab.CUSTOMIZATION_TIME_SHEETS_EXTEND_BUSINESS_KEY,
                        tab.DESCRIPTION,
                ${MD_BATCH_ID} AS MD_BATCH_ID ,
                ${MD_PROCESS_ID} AS MD_PROCESS_ID ,
                ${MD_SOURCE_INSTANCE_ID} AS MD_SOURCE_INSTANCE_ID ,
                tab.MD_FLAG
from (
                select
                                t1.md_source_instance_id || ':' || t1.md_
```

```
business_key as MD_BUSINESS_KEY,
t1.md_source_instance_id || ':' || t1.TIME_SHEET_ID as CUSTOMIZATION_TIME_
SHEETS_EXTEND_BUSINESS_KEY,
                                  DESCRIPTION,
                                decode(t1.md_
flag,'NEW','NEW','DEL','DEL','UPD') as md_flag,
                                row_number() over( partition by t1.md_source_
instance_id || ':' || t1.md_business_key
                                                  ) multi_flag
                 from CUSTOMIZATION_TIME_SHEETS_STREAM_CUSTOMIZATION_TM_TIME_
SHEETS_${MD_SOURCE_INSTANCE_ID}_EXT t1
                                where t1.md_pf_flag = 'D') tab
where tab.multi_flag = 1;
--==--==--
       SELECT ANALYZE_STATISTICS('PPM_CUSTOMIZATION_TIME_SHEETS_${MD_SOURCE_
INSTANCE_ID}_SSI');
--==--==--
"
               }
       ]
}
```

`CUSTOMIZATION_TIME_SHEETS_STREAM.json` defines:

- Source entities that provide data
- Target entities that accept data
- SQLs mainly to transform data from source entities to target entities

After you complete this step, the stream entity JSON file is defined.

c. Under `extmetadata`, create and define the extractor entity JSON file `CUSTOMIZATION_TM_TIME_SHEETS_EXT.json`:

```
{
    "metadata_layout_version": "1.0",
    "version": "1.0",
    "source_product": "PPM",
    "content_pack": "CUSTOMIZATION_PPM",
    "entity_name": "CUSTOMIZATION_TM_TIME_SHEETS_EXT",
    "source_entity_name": "TM_TIME_SHEETS",
    "extractor": "OracleDBExtractor",
    "extraction": [
        {
            "extraction_view": "SELECT TIME_SHEET_ID, DESCRIPTION FROM TM_
TIME_SHEETS",
            "source_product_version": "9.30"
        }
    ]
}
```

`CUSTOMIZATION_TM_TIME_SHEETS_EXT.json` defines:

- The extractor entity name
- **source_entity_name** as the source table name
- **extraction_view**: SQLs for extracting data

After you complete this step, the extractor entity JSON file is defined.

d. Create and define `cp.json`:

```
{
	"metadata_layout_version": "1.0",
	"content_pack_name": "CUSTOMIZATION_PPM",
	"source_product": "PPM",
	"version": "1.0",
	"description": "Project Management & Portfolio Management Content Pack,
Extend Entities.",
	"require": {
	"platform": ">=1.0.0",
		"cp": [
			{
				"name": "CUSTOMIZATION_TARGET",
				"version": ">=1.0.0"
			}
		]
	},
	"streams": [
		{
			"name": "CUSTOMIZATION_TIME_SHEETS_STREAM"
		}
	],
	"source_entities": [
		{
			"name": "CUSTOMIZATION_TM_TIME_SHEETS"
		}
	],
	"extraction_entities": [
		{
			"name": "CUSTOMIZATION_TM_TIME_SHEETS_EXT"
		}
	]
}
```

`cp.json` defines:

- The content pack name: `CUSTOMIZATION_PPM`
- Source entities: Defined in `CUSTOMIZATION_TM_TIME_SHEETS.json`

- Stream entities: Defined in `CUSTOMIZATION_TIME_SHEETS_STREAM.json`
- Extract entities: Defined in `CUSTOMIZATION_TM_TIME_SHEETS_EXT.json`

Deploying Content Packs

To deploy content packs, follow these steps:

1. Place `CUSTOMIZATION_PPM.cp` and `CUSTOMIZATION_TARGET.cp` under *<VDW_HOME>*`/Content`.
2. Run the `ContentManager.sh` script under the *<VDW_HOME>*`/bin` directory to deploy `CUSTOMIZATION_TARGET.cp`:

   ```
   sh ContentManager.sh --instruction install --cpname CUSTOMIZATION_TARGET;
   ```

   You can find the following message from `ContentManager.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   ```
   The content of package CUSTOMIZATION_TARGET was successfully installed
   ```

   The DIM_ CUSTOMIZATION_CONTACTS table is generated in the Vertica database.
3. Run the `ContentManager.sh` script under the *<VDW_HOME>*`/bin` directory to deploy `CUSTOMIZATION_PPM.cp`:

   ```
   sh ContentManager.sh --instruction install --cpname CUSTOMIZATION_PPM;
   ```

   You can find the following message from `ContentManager.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   ```
   The content of package CUSTOMIZATION_PPM was successfully installed
   ```
4. Run the `ExtractorEngine.sh` script under the *<VDW_HOME>*`/bin` directory to extract data from the PPM database to flat files:

   `sh ExtractorEngine.sh --streamname CUSTOMIZATION_TIME_SHEETS_STREAM --instancename` *<PPM_Instance_Name>*

   You can find the following message from `ExtractorEngine.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   `Extractor was successfully executed. The BATCH ID is:` *<Batch_ID>*.

<PPM_Instance_Name> is the PPM instance name you specified when installing the Vertica for PPM Content Pack.

5. Run the `FlowEngine.sh` script under the *<VDW_HOME>*`/bin` directory to process ETL:

   `sh FlowEngine.sh --batch` *<Batch_ID>* `--streamname CUSTOMIZATION_TIME_SHEETS_STREAM --instancename` *<PPM_Instance_Name>*

   You can find the following message from `FlowEngine.log` under *<VDW_HOME>*`/logs` if the content pack is deployed successfully:

   ```
   ETL process was executed successfully
   ```

   <Batch_ID> used in this command is the batch ID that was generated in Step 4.

6. Connect to the Vertica database and check whether the data has been loaded successfully:

   `select * from` *<Target_Schema>*`.DIM_CUSTOMIZATION_TIME_SHEETS`

   <Target_Schema> is the name for the schema that contains target data and tables for reporting.

Customizing ETL Rules

What is ETL Rules

Extract-Transform-Load (ETL) is the process of extracting data from data source, transforming data, and loading data to the target data warehouse. ETL, as the core of Business Intelligence, is a critical step for deploying data warehouse.

ETL contains the following steps:

1. EXT
2. SSI
3. XREF
4. MSI
5. XFR
6. KEY LOOKUP

7. TARGET
8. TSNP
9. HIERARCHY
10. POST TARGET

For details of these steps, see "ETL Steps Introduction" on page 41.

Why Customize ETL Rules

The PPM database includes SQLs that can process the ETL steps. At the same time, OpenText PPM also supports ETL process customization to meet various business needs. Because the business environments vary, you can optimize the ETL performance by customizing ETL rules.

How to Customize ETL Rules

You can customize ETL rules by running the following command:

`sh ArtifactRegister.sh --artifactfile` *<DATA FILE PATH>* `--streamname` *<STREAM NAME>* `--etlstep` *<ETL STEP>* `--register CUSTOMIZATION`

Supported Parameters

Parameter	Mandatory?	Sample Value	Description
`streamname`	Yes	`PPM_PERSON_STREAM`	The name of the stream in which the ETL rules need to be customized; See "Entity Attribute Descriptions" on page 55 for details.
`etlstep`	Yes	`EXT|SSI|TSNP|XREF|MSI|XFR|KEYLOOKUP|TARGET|HIERARCHY|POSTTARGET`	ETL step that needs to be customized

Supported Parameters, continued

Parameter	Mandatory?	Sample Value	Description
`artifactfile`	Yes	`/temp/XREF.SQL`	SQL file that contains customized ETL logic
`register`	No	`CUSTOMIZATION`	All non-HP provided SQL commands should be registered as `CUSTOMIZATION`
`help`	No		Prints put the help message

For detailed explanation, see "Administration Tasks" in the *Vertica for PPM Administrator Guide for Content Pack 1.0*.

Cusomization Validation

All the ETL rules are stored in the ETL_GENERATED_ARTIFACT table of the management schema. You can query customized ETL rules by running the following SQLs:

`SELECT * FROM <MANAGER_SCHEMA>.ETL_GENERATED_ARTIFACT,<METADATA_SCHEMA>.ETL_METADATA where ETL_GENERATED_ARTIFACT.STREAM_ID = ETL_METADATA.OBJECT_ID AND ETL_GENERATED_ARTIFACT.REGISTERED_BY='CUSTOMIZATION' AND ETL_METADATA.OBJECT_NAME='<STREAM_NAME>'`

- *<MANAGER_SCHEMA>* is the name that you specified for the schema that contains management tables when installing the Vertica for PPM content pack.
- *<METADATA_SCHEMA>* is the name that you specified for the schema that contains metadata when installing the Vertica for PPM content pack.
- *<STREAM_NAME>* is the stream name that is specified when running `ArtifactRegister.sh`.

Best Practices

Customizing ETL rules is a very complex process. Follow these steps for customization:

1. Query the system with the following command to find out the default rule for a specified ETL step under a certain stream:

   `SELECT ARTIFACT_CONTENT FROM <MANAGER_SCHEMA>.ETL_GENERATED_ARTIFACT,<METADATA_SCHEMA>.ETL_METADATA where ETL_GENERATED_ARTIFACT.STREAM_ID = ETL_METADATA.OBJECT_ID AND ETL_GENERATED_ARTIFACT.REGISTERED_BY='SYSTEM' AND ETL_METADATA.OBJECT_NAME='<STREAM_NAME>' AND ETL_GENERATED_ARTIFACT.ETL_STEP='<ETL_RULE_STEP_NAME>'`

   - *<MANAGER_SCHEMA>* is the name that you specified for the schema that contains management tables when installing the Vertica for PPM content pack.
   - *<METADATA_SCHEMA>* is the name that you specified for the schema that contains metadata when installing the Vertica for PPM content pack.
   - *<STREAM_NAME>* is the name of the stream that needs to be customized.
   - *<ETL_RULE_STEP_NAME>* is the ETL step that needs to be customized. Possible values are `SSI, XREF, MSI, XFR, KEYLOOKUP, TARGET, HIERARCHY, TNSP, and POSTTARGET`.
2. Copy the SQLs from the **ARTIFACT_CONTENT** field to a TXT file, such as `data.txt`. Adjust the SQLs and keep variables such as **MD_BATCH_ID**, **MD_PROCESS_ID**, and **MD_SOURCE_INSTANCE_ID** as is.
3. Run the following command to customize the ETL rule:

   `sh ArtifactRegister.sh --artifactfile data.txt --streamname <STREAM_NAME> --etlstep <ETEL_RULE_STEP_NAME> --register CUSTOMIZATION`

Note:

- Do not change the metadata of OOTB entities.
- Do not add new fields to the existing staging or target tables.

- During the system upgrade, the system bypasses all customized ETL artifacts. Sometimes it may cause ETL to break down. If that happens, correct your ETL SQLs and register again.

ETL Architecture

The Vertica for PPM content pack includes a flexible ETL Engine that can generate Vertica SQLs to process data based on metadata defined in the JSON format in content packs. Leveraging the flexible metadata and ETL Engine, you can easily extended ETL Content to extract data from more PPM tables and store them in the Vertica database for later data consumption.

All content are packaged to content packs. Two types of content packs are available: source content pack and target content pack. Source content pack includes source entity metadata, extraction entity metadata, and stream definitions. Whereas target content pack only contains target entities. For explanations of source entity, target entity, extraction entity, and stream, see "Entity Attribute Descriptions" on page 55.

After all metadata are developed, administrators can use the Content Manager tool to deploy that content pack. For detailed instructions on how to deploy a content pack, see *Vertica for PPM Administrator Guide*. After that, administrators can schedule ETL job with `crontab` to start data extraction.

ETL Architecture Overview

Typically an ETL process contains 10 steps. Flow Engine calls Template Engine to generate SQLs based on the ETL templates for each entity during run time and Flow Engine executes those 10 steps one by one.

ETL templates contains most of the common ETL patterns, so you only need to focus on the business logic when developing new content.

In general, ETL templates support the following ETL patterns:

- Change Data Capture

  ETL Engine captures only change data based on the **is_cdc** attributes defined in the source metadata. Deleted data can be captured based on the **is_bk** attributes defined in the source metadata.

- Slow Changing Dimension

  ETL Engine supports slow changing dimension. It automatically keeps the dimension history if you mark a field as **scd2**.

- Late Arriving Dimension

  ETL Engine automatically generates dummy record for late arriving dimension data, and updates the dummy record when data arrives.

- Surrogate Key Generation

  ETL Engine generates surrogate key automatically. It generates the enterprise key to identify records and the primary key to identify the history of records.

- Hierarchy Flatten

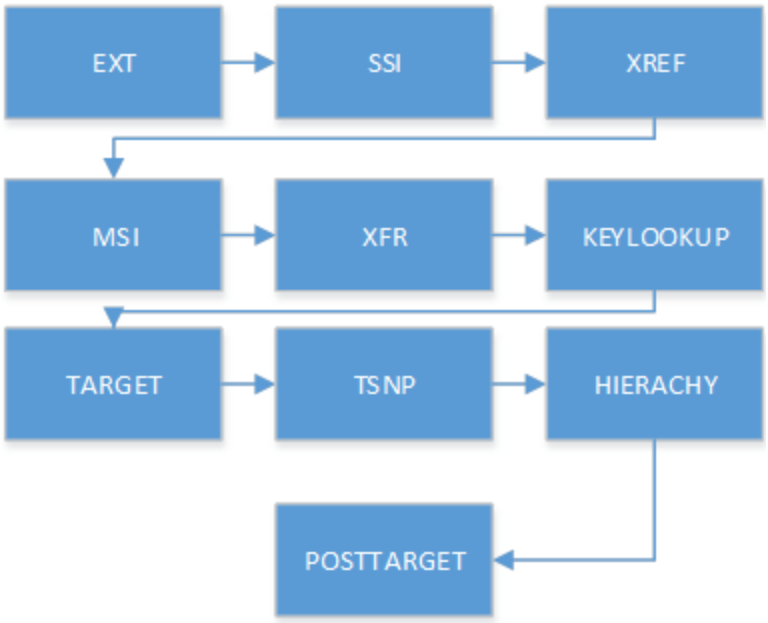
  Based on the target metadata definition, ETL Engine stores hierarchy relation information in the BRIDGE_HIEARCH tables. Unlimited hierarchy levels are supported.

- Date Timezone Conversion

  ETL Engine converts date to the data warehouse timezone automatically. You need to specify the PPM database timezone and PPM data warehouse timezone during installation.

ETL Steps Introduction

The ETL process includes 10 steps as shown in the following figure.



For detailed description of each ETL step and the structure of key staging tables of that step, read the following sections.

ETL Step 1: EXT

EXT is the first step of ETL. During EXT, the staging table schema aligns with the source entity schema.

Flat files are input in this step, while *_EXT tables are output.

The EXT step does the following:

- Loading data from flat files to staging tables

  Flat files created by Extractor Engine are copied from flat files to staging tables (naming convention: *_SRC and *_DEL_SRC).
- Cleaning data with null/duplicated business key
  - The system checks the data type and data length from the flat files. The system also checks if the data field is null. It is required that the data is not null. Rejected data is loaded to the VALF (validation fail) tables (naming convention: *_VALF). The VALF tables have the same table structures but all fields' data type is `VARCHAR (50000)`, which is to make sure that the data is not rejected again.

  - The system checks if there is duplicated data based on the **is_bk** attribute defined in the source entity. Duplicated data is moved to the VALF tables as well.
- Generating hash codes for records

  The system generates a hash code for each record and puts the hash code to the **MD_HASH_CODE** field. The system use the hash code to compare table for data updates.
- Prefilling data from snapshot tables to make sure that records can be joined in the SSI step.

  To reduce the load on the PPM database, Extractor Engine only extracts changed data. The system puts all data that was loaded to snapshot tables (naming convention: *_TSNP) in the TSNP step. In the EXT step, the system pulls data back from the snapshot tables, to make sure that all data can be joined in the SSI step.
- Checking for updated, deleted, and inserted data
  - Through comparing snapshot tables (naming convention: *_TSNP), the system tags data with `UPD` for update and `NEW` for insert.
  - Extractor Engine extracts the business key fields for all records from the PPM database and compare it with the snapshot tables for deleted records.

The following table lists the key staging tables.

Key Staging Tables

Table Name	Table Description
*<STREAM_NAME>_<SOURCE_ ENTITY_NAME>_<MD_SOURCE_ INSTANCE_ID>*_SRC	Stores data loaded from flat files for further process.
*<STREAM_NAME>_<SOURCE_ ENTITY_NAME>_<MD_SOURCE_ INSTANCE_ID>*_SRC	Contains all business key fields, used for detecting deleted data.
*<STREAM_NAME>_<SOURCE_ ENTITY_NAME>_<MD_SOURCE_ INSTANCE_ID>*_TSNP	Contains all data that was loaded and the history of records.

Key Staging Tables , continued

Table Name	Table Description
*<STREAM_NAME>_<SOURCE_ ENTITY_NAME>_<MD_SOURCE_ INSTANCE_ID>*_VALF	Stores rejected data.
*<STREAM_NAME>_<SOURCE_ ENTITY_NAME>_<MD_SOURCE_ INSTANCE_ID>*_EXT	Contains data that has been processed in the EXT step and is going to be passed to the SSI step.

Most of the staging tables in this step contains the following fields.

System Reserved Fields in Staging Tables

Field Name	Field Description
MD_BATCH_ ID	Data Batch ID. Extractor Engine puts data to flat files and gives a batch ID for those data. Administrators can use this ID to track the data in the staging tables and the target tables, and also check Extractor Engine and Flow Engine log files during troubleshooting.
MD_ PROCESS_ID	ETL process ID. Flow Engine generates process ID and stores the ID in tables. Administrators can use the ID to check the Flow Engine log and the process status in the Derby database.
MD_ SOURCE_ INSTANCE_ ID	Source Instance ID. System assigns an ID for PPM instance when the administrator registers a new PPM instance to the system. Vertica for PPM users can use this ID to separate data from different PPM instances.
MD_FLAG	Indicates whether this row is new, is updated, or needs to be deleted.
MD_PF_FLAG	Indicates whether row is prefilled. P means the data is prefilled from the snapshot tables. D means the data is loaded from flat files during the test load.
MD_ BUSINESS_ KEY	Primary key that is mapped to one or more source columns. The system concatenates MD_SOURCE_INSTANCE_ID as part of MD_ BUSINESS_KEY.
MD_HASH_ CODE	Stores system-generated hash codes. The system compares this field with the snapshot tables for data update.

ETL Step 2: SSI

SSI is the only step that does not have any ETL templates. Flow Engine executes SQLs defined in the stream JSON file under the **transforms** attribute.

*_EXT tables are input in this step, while *_SSI tables are output.

You can customize Vertica for PPM reports totally in your way in the SSI step. However, you need to understand the major functions of this step:

- Handling business logic

  This is the only step where the entity-specific logic can be handled. You can input SQLs for handling the business logic, such as calculation and aggregation.
- Transforming data models from the source model (defined in source entities) to the target model (defined in target models)

  The output table of the SSI step (naming convention: *_SSI) should align with the target model in both the field name and data granularity.
- Cleaning unnecessary prefilled data

  To improve the ETL performance, you need to remove any unnecessary prefilled data. You can refer to the SQLs that contain `where tab1.md_pf_flag='D` or `tab2.md_flag='D'` in the sample ETL entities.

  For more information about obtaining the sample content packs, see "Obtaining Sample Content Packs for Customization" on page 7.
- Combining multiple source entities into one target entity

  If your target entity needs data from multiple source entities, combine these source entities properly to populate data to the *_SSI tables.

You also need to pay attention to the following during the SSI development:

- **MD_FLAG** field

  MD_FLAG is a very important field that controls the data loading behaviors in the following ETL steps. You need to mark the record as `NEW` if the record is a newly created and `UPD` if the record already exists in the target tables. If you mark the record as `DEL`, it means that record needs to be deleted later.

You need to consider it thoroughly when you combine multiple source entities. Data from different entities carries different **MD_FLAG** from the EXT step.

- Fields need to be populated in the *_SSI tables
  - **MD_BUSINESS_KEY**: This record is used to identify the data granularity. The system generates the enterprise key later based on the value of **MD_BUSINESS_KEY**. Make sure this value is unique and is not changed during the test load. Records with different **MD_BUSINESS_KEY** fields are considered as different records. It is suggested that prefix *<MD_SOURCE_INSTANCE_ID>* is included as part of **MD_BUSINESS_KEY**.
  - **MD_BATCH_ID**: You can populate this field with the value from the *_EXT table or the value from the *<MD_BATCH_ID>* runtime variable.
  - **MD_SOURCE_INSTANCE_ID**: You can populate this field with the value from the *_EXT table or the value from the *<MD_SOURCE_INSTANCE_ID>* runtime variable.
  - **MD_PROCESS_ID**: You can populate this field with the value from the *_EXT table or the value from the *<MD_PROCESS_ID>* runtime variable.
- Temporary tables that you create in the SSI step

  Consider the Vertica storage strategy to speed up ETL when you run ETL on top of the Vertica database cluster. You can use `UNSEGMENTED ALL NODES` for small tables and `SEGMENTED by hash (md_business_key) all nodes` for large tables in most cases. For details about Vertica storage impact on performance, refer to the Vertica documentation.

The following table lists the key staging tables.

Key Staging Tables

Table Name	Table Description
*<SOURCE_PRODUCT_NAME>_<SOURCE_ENTITY_NAME>_<MD_SOURCE_INSTANCE_ID>*__SSI	SSI table is an output table of the SSI step. It shall align with the target model on both schema and data granularity

The SSI table contains the following system reserved fields.

System Reserved Fields in Staging Tables

Field Name	Field Description
<ROLE_ ENTITY_ NAME>_ BUSINESS_ KEY	If you define any entity lookup in the target entity in **dimension_ associated_dimension**, **fact_associated_dimension**, or **fact_ associated_fact** attibutes, you need to create a field in the SSI table, and put relevant business key values into this field. The name of this field should contain *<ROLE_ENTITY_NAME>* defined in the target entity as the prefix . The value stored in this field should be the same as in **MD_ BUSINESS_KEY** of the `lookup_entity_name` entity. Otherwise ETL Engine is not able to build reference between entities in the following steps.

ETL Step 3: XREF

In the XREF step, the system generates **MD_ENTERPRISE_KEY** for records based on the value of **MD_BUSINESS_KEY**. Unlike many other staging tables that are dropped and re-created every time when the ETL Job runs, *_XREF tables (naming convention: *_XREF) are not cleared. *_XREF tables maintain a mapping relationship between **MD_BUSINESS_KEY** and **MD_ENTERPRISE_KEY** to make sure each **MD_ENTERPISE_KEY** is uniquely mapping to a **MD_BUSINESS_KEY**.

The system also generates **MD_ENTERPRISE_KEY** for other reference entities if the referred record is not arrived. This is a part of the Late Arriving Dimension function.

*_SSI tables are input in this step, while *_XREF tables are output.

The following table lists the key staging tables.

Key Staging Tables

Table Name	Table Description
<TARGET_ENTITY_ NAME>_ XREF	Contains a mapping relationship between **MD_ BUSINESS_KEY** and **MD_ENTERPRISE_KEY**

The XREF table contains the following system reserved fields.

System Reserved Fields in Staging Tables

Field Name	Field Description
MD_BATCH_ID	For late arriving data, the batch ID is set to `-1`. The system updates **MD_BATCH_ID** when data arrives
BUSINESS_ KEYVALUE	Stores business key values that are read from **MD_BUSINESS_ KEY** in the *_SSI tables
ENTERPRISE_ KEY	Stores generated sequence numbers

ETL Step 4: MSI

In the MSI step, the system obtains the enterprise key generated from **MD_ BUSINESS_KEY** in the XREF step and stores it in the MSI step result tables (such as *_MSI tables). If the target entity has references to other entities, it looks up the *_ XREF tables of other entities to obtain the enterprise key of the referenced entities and stores it in the **<ROLE_ENTITY_NAME>_ENTERPRISE_KEY** field.

The system also converts all date fields from the PPM timezone to the data warehouse timezone. If the PPM timezone is enabled with daylight saving, daylight saving is also applied to the data warehouse timezone.

Vertica for PPM content pack leverages the daylight-saving information in the Java Runtime Environment (JRE). Thus you need to update the JRE on a regular basis.

During EXT, the hash code for source entities has been generated. Because the model and data granularity has been changed in the SSI step, in the MSI step, the system regenerates the hash code for records. To verify if an **SCD2** field has been updated, the system generates hash codes for all **SCD1** fields and all **SCD2** fields respectively.

The system also checks the *_XREF tables to understand if dummy records need to be generated for late arriving data. If the *_XREF table contains records in which the value of **MD_BATCH_ID** is `-1`, dummy records are generated.

*_SSI tables and *_XREF tables are input in this step, while *_MSI tables are output.

The following table lists the key staging tables.

Key Staging Tables

Table Name	Table Description
*<SOURCE_PRODUCT_NAME>_<TARGET_ENTITY_NAME>_<MD_SOURCE_INSTANCE_ID>_*MSI	Output table of the MSI step, including the enterprise key generated in the XREF step, and the dummy records for late arriving data.

The MSI table contains the following system reserved fields.

System Reserved Fields in Staging Tables

Field Name	Field Description
MD_ENTERPRISE_KEY	Unique identifier for a record. Generated base on **MD_BUSINESS_KEY** populated in the SSI step.
*<ROLE_ENTITY_NAME>_*BUSINESS_KEY	Business key of entities that will be referred to. This should be exactly the same as MD_BUSINESS_KEY of the referred entity.
*<ROLE_ENTITY_NAME>_*ENTERPRISE_KEY	Enterprise key of the entities that are referred to. The system automatically populates this field through looking up the *_XREF table of the referred entity based on the value of **<ROLE_ENTITY_NAME>_BUSINESS_KEY**.
MD_HASH_CODE_SCD1	Hash code that is generated by the **SCD1** columns for the dimension table.
MD_HASH_CODE_SCD2	Hash code that is generated by the **SCD2** columns for the dimension table.
MD_HASH_CODE	Hash code that is generated by all columns for the fact table.

ETL Step 5: XFR

In the XFR step, the system separates incoming data to different staging tables based on **MD_FLAG**. New records are put to the *_XFRN tables and deleted records are put to the *_XFRD tables. SCD1-updated records are put to the *_XFRU tables. However, for SCD2-updated records, a copy of SCD2 history records is

stored in the *_XFRN tables. Later, the copy is inserted to the target tables as the latest records of SCD2. A copy is also stored in the *_XFRU2 tables.

The system also verifies if a dummy record is available in the target tables because of the late arriving data. If so, even if incoming data contains the SCD2 changes, the system stores the record to the *_XFRU tables only to make sure the dummy records are updated accordingly.

For fact entities, SCD2 is not supported, system puts all updated records in the *_XFRU tables.

*_MSI tables are input in this step, while *_XFRN, *_XFRD, *_XFRU, and *_XFRU2 tables are output.

The following table lists the key staging tables.

Key Staging Tables

Table Name	Table Description
*<SOURCE_PRODUCT_NAME>_<TARGET_ENTITY_NAME>_<MD_SOURCE_INSTANCE_ID>*_DIM/FACT_XFRN	Contains all data that is inserted into the target tables in the TARGET step.
*<SOURCE_PRODUCT_NAME>_<TARGET_ENTITY_NAME>_<MD_SOURCE_INSTANCE_ID>*_DIM/FACT_XFRD	Contains all data that is deleted in the target tables in the TARGET step. The system does not physically delete the records. Instead, it updates **MD_ACTIVESTATUSIND** to N and **MD_DELETEDDATE** to reflect the deleted data.
*<SOURCE_PRODUCT_NAME>_<TARGET_ENTITY_NAME>_<MD_SOURCE_INSTANCE_ID>*_DIM/FACT_XFRU	Contains all data that is updated in the target tables in the TARGET step.
*<SOURCE_PRODUCT_NAME>_<TARGET_ENTITY_NAME>_<MD_SOURCE_INSTANCE_ID>*_DIM/FACT_XFRU2	Contains all data that is updated in the target tables in the TARGET step. The *_XFU2 tables only need to update **MD_TRANSLASTIND** to N and **MD_TRANSENDDATE** to sysdate to reflect the last effective date of the SCD2 history records.

The XFR table contains the following system reserved fields.

System Reserved Fields in Staging Tables

Field Name	Field Description
MD_CREATEDDATE	The date when this record was created
MD_DELETEDDATE	The date when this record was deleted
MD_LASTMODDATE	The date when this record was updated most recently
MD_TRANSLASTIND	• Y: This is the latest record. • N: This is a history record. Applies to SCD2 history records. For other records without history, the value will be 'Y'
MD_TRANSENDDATE	The last effective date of this record. Applies to the SCD2 history records.
MD_ ACTIVESTATUSIND	• Y: The record is active • N: The record is deleted

ETL Step 6: KEYLOOKUP

In the KEYLOOKUP step, the system generates the primary key for all new records; that is, records in the *_XFRN tables are based on the value of **MD_ENTERPRISE_ KEY**. The system also generates the primary key for late arriving data of referred entities. The mapping between **MD_ENTERPRISE_KEY** and the primary key is one to many because the system assigns a primary key to each history record.

*_XFRN tables are input in this step while *_ KEY_LOOKUP tables are output.

The following table lists the key staging tables.

Key Staging Tables

Table Name	Table Description
*<TARGET_ENTITY_NAME>*_ DIM/FACT_KEY_LOOKUP	Contains mapping between **MD_ENTERPRISE_ KEY** and **PK_<TARGET_ENTITY_NAME>**.

The KEY_LOOKUP table contains the following system reserved fields.

• System Reserved Fields in Staging Tables

Field Name	Field Description
PK_*<TARGET_ENTITY_NAME>*	Generated primary keys for new records

ETL Step 7: TARGET

In the TARGET step, the system loads data to the target tables. It obtains the primary key generated in the KEYLOOKUP step, inserts data from the *_XFRN tables to the target tables, updates the target tables according to the data from the *_XFRU and *_XFRU2 tables, and marks the records as deleted through setting **MD_ACTIVESTATUSIND** to `N` and updating **MD_DELETEDATE** as sysdate.

If late arriving data in the previous batch arrives, the system updates **MD_BATCH_ID** in the *_ XREF tables with the current batch ID.

*_XFRN, *_XFRD, *_XFRU, *_XFRU2, and *_KEY_LOOKUP tables are input in this step while DIM_*, FACT_*, or BRIDGE_* tables are output.

The following table lists the key staging tables.

Key Staging Tables

Table Name	Table Description
DIM_*<TARGET_ ENTITY_NAME>*	Target dimension tables that include data for reporting.
FACT_*<TARGET_ ENTITY_NAME>*	Target fact tables that include data for reporting.
BRIDGE_*<TARGET_ ENTITY_NAME>*	Target bridge tables that include data for reporting of many-to-many relationships.

ETL Step 8: TSNP

In the TSNP step, the system maintains snapshot tables. The system copies all records from *_EXT tables to *_TSNP tables to keep a snapshot. The *_TSNP tables are used as a source for prefilled data and also for Change Data Capture.

The system also updates the **LAST_EXTRACTION_VALUE** field of the DATA_SOURCE_CDC_STAMP table. Extract Engine extracts data from the PPM database incrementally based on the value of this field. The *_TSNP tables track the change history of the PPM data as well.

*_EXT tables are input in this step while *_ TSNP tables are output.

The following table lists the key staging tables.

Key Staging Tables

Table Name	Table Description
*<SOURCE_PRODUCT>_<TARGET_ENTITY_NAME>_<MD_SOURCE_INSTANCE_ID>*_TSNP	Snapshot tables that contain data extracted from PPM. It contains both the latest data and the history data.
DATA_SOURCE_CDC_STAMP	A table in the management schema. Extractor Engine relies on this field to extract data incrementally.

ETL Step 9: HIERARCHY

If a dimension entity has hierarchy definition in the target entity JSON, the system creates a hierarchy table in the HIERARCHY step. If no hierarchy is defined, this step is skipped. The hierarchy table supports unlimited hierarchy levels. The history of hierarchy changes is also kept for future reference.

DIM_* tables are input in this step while BRIDGE_HIERARCHY_* tables are output.

The following table lists the key staging table.

Key Staging Table

Table Name	Table Description
BRIDGE_HIERARCHY_*<TARGET_ENTITY_NAME>*	Stores the dimension hierarchy level information

ETL Step 10: POSTTARGET

The objectives for the POSTTARGET step are:

- To create views on top of the dimension and fact tables, so that Vertica for PPM reporting users can understand the data easily.
- To open a window for content engineers to perform any analytical functions after all data are loaded into the target tables.

The system defines templates for creating views. Vertica for PPM creates the following types of views based on the table types and metadata defined in the target table:

- Hierarchy Drill UP View /Hierarchy Drill Down View

  Those two views are created for Vertica for PPMreporting users, so they do not have to combine BRIDGE_HIEARCHY tables by themselves. If one dimension entity has hierarchy defined, the system automatically creates those two views. Users can select Hierarchy Drill UP View if they want to know all parent records and Hierarchy Drill Down View if they want to know all child records.

- User Data View

  The system creates User Data View only if a **dynamic_view** attribute is defined in the target JSON file. The system leverages the context information defined in PPM to provide **USER_DATA** fields in PPM with a meaningful name in the views.

  If a dimension entity has both **bridge** and **dynamic_view** defined, the system generates the consolidated view automatically.

- Common View

  For all other target tables, the system also creates a view layer. Extended entities can be joined with the OOTB entity through the view layer for better a user experience.

In the stream JSON definition, you can also define the **post_target_transforms** attribute. For examples, see OOTB `PPM_RESOURCE_DEMAND_STREAM.json` (under `<VDW_HOME>/Content/CUSTOMIZATION_PPM.cp/dwmetadata/streams`).

All SQL-defined **post_target_transforms** attributes are executed in this step as well.

Appendix A: Entity Attribute Descriptions

This section lists detailed explanations for each attribute of the source entity, target entity, extractor entity, stream entity, and CP JSON files.

For detailed file structures, refer to "Content Pack Structure " on page 7.

The following table shows the attribute descriptions of the source entity JSON file.

Source Entity JSON File

Field	Description
metadata_layout_ version	Version of the metadata format
version	Version of the entity metadata
content_pack	Content pack name. Should be the same as **content_pack_name** defined in `cp.json`
source_product	Source product
entity_name	Source entity name
entity_business_name	Business name of the entity
entity_description	Source entity description
schema	Target entity attribute array
attribute	Source table field that needs to be extracted
attribute_name	Attribute name. Serves as a field name in the staging tables
attribute_business_ name	Attribute business name
attribute_description	Source table field description
sql_data_type	Field data type
size	Field size

Source Entity JSON File, continued

Field	Description
is_bk	The value is `true` if the attribute serves as a business key
is_cdc	The value is `ture` if data is incrementally extracted. In most cases, this attribute is used for **LAST_UPDATE_DATE**. If it is not specified as `true`, Extractor Engine extracts all records everytime.
is_required	The value is `ture` if this field is not null
column_sequence	The column sequence when creating staging tables

The following table shows the attribute descriptions of the target entity JSON file.

Target Entity JSON file

Field	Description
metadata_layout_ version	Version of the metadata format
version	Version of this entity metadata
content_pack	Content pack name. Should be the same as **content_pack_name** defined in `cp.json`
entity_name	Target entity name
entity_business_name	Business name of the entity
entity_description	Target entity description
dimension	Included for a target dimension entity
dimension_business_ name	Dimension business name
is_conformed	Whether this dimension is conformed; for future usage
dimension_type	`Hierarchy` or `Primary`; use `Hierarchy` if you want to define hierarchical structure for this entity
storage_strategy	Includes **segmented_by** and **partition_by**

Target Entity JSON file, continued

Field	Description
segmented_by	Used for table segmentation; specify `Default` if you want to copy data of this entity to all Vertica clusters
partition_by	Used for table partition
fact	Included if this is a target fact entity
fact_business_name	Business name of the fact
fact_description	Description of the fact entity
fact_type	`ACCUMULATED` is the only supported value
schema	Target entity attribute array
attribute	Name of this attribute
attribute_name	Target entity attribute name
attribute_business_name	Business name of this attribute
attribute_description	Description of this attribute
attribute_type	If this is target fact entity, the value should be `measure`; otherwise the value is `dimension`.
scd	**scd1**: Update the record **scd2**: Insert a new record
target_data_type	Target entity attribute type
size	Target entity attribute size
is_required	The value is `ture` if this field is not null
dimension_associated_dimension	Defines the dimension if this dimension refers to other dimensions
fact_association_dimension	Defines the fact if this fact refers to other dimensions

Target Entity JSON file, continued

Field	Description
fact_association_fact	Defines the fact table if this fact table refers to other fact tables **Note:** Late arriving fact tables are not supported. Make sure the referred fact table is loaded prior to this fact table.
lookup_entity_name	Lookup entity name
role_entity_name	Alias for the referred entity
role_entity_business_name	Business name for the referred entity
description	Description of this lookup

The following table shows the attribute descriptions of the extractor entity JSON file.

Extractor Entity JSON file

Field	Description
metadata_layout_version	Version of the metadata format
version	Version of this entity metadata
source_product	Source product
content_pack	Content pack name. Should be the same as **content_pack_name** defined in `cp.json`
entity_name	Entity name
source_entity_name	Source table name
extractor	The extractor that is called to extract data. Currently `OracleDBExtractor` is supported.
extraction	Contains the extraction attributes
extraction_view	An on-the-fly view from which extractor extracts data from.
source_product_version	The version of the source product that the extractor extracts from

The following table shows the attribute descriptions of the stream entity JSON file.

Stream Entity JSON file

Field	Description
metadata_ layout_ version	Version of the metadata format
version	Version of this entity metadata
content_ pack	Content pack name. Should be the same as **content_ pack_name** defined in `cp.json`
source_ product	Source product
stream_ name	Stream entity name
source_ entities_ includes	Defines the source entity of the stream
source_ entity_ include	The source entity name of the stream
target_ entities_ includes	Defines the target entity of the stream
target_ entity_ include	The target entity name of the stream
transforms	Contains the attribute related to SQL transformation
sql	Queries that transform source entities to target entities
post_target_ transforms	Optional. If you want to process other SQLs after data is loaded to target tables, include SQLs in this attribute.
sql	Includes SQL queries that is executed as part of POSTTARGET step

The following table shows the attribute descriptions of the CP JSON file.

CP JSON File

Field	Description
metadata_ layout_version	Version of the metadata format
content_pack_ name	Content pack name
version	Version of this content pack
description	Description of this content pack
require	Dependency of this content pack
platform	Platform version that requires to support this content pack
target_entities	Target entity section. Lists all target entities that you defined in this content pack
name	Target entity name
streams	Stream entity section; lists all stream entities that you defined in this content pack
name	Stream entity name
source_entities	Source entity section; lists all source entities you defined in this content pack
name	Source entity name
extraction_ entities	Extractor entity section; lists all extraction entities you defined in this content pack
name	Extractor entity name