

**Oracle® Agile Product Lifecycle Management for Process  
Custom Section Denormalization Guide**

Release 6.2.4.x

**F58003-01**

May 2022

**ORACLE®**

# Copyrights and Trademarks

Agile Product Lifecycle Management for Process

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle

Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

## Contents

|   |           |
|---|-----------|
| <b>PREFACE.....</b>   | <b>6</b>  |
| Audience .....  | 6         |
| Variability of Installations .....  | 6         |
| Documentation Accessibility.....  | 6         |
| Access to Oracle Support .....  | 6         |
| Software Availability .....   | 6         |
| <b>CHAPTER 1—OVERVIEW .....</b>   | <b>7</b>  |
| <b>CHAPTER 2—DENORMALIZATION PROCESS .....</b>  | <b>8</b>  |
| 1. Configuration .....  | 8         |
| 2. Table Creation.....  | 8         |
| 3. Denormalization.....   | 8         |
| Near Real-Time Denormalization (NRTD) .....   | 9         |
| Batch Denormalization.....  | 9         |
| <b>CHAPTER 3—CUSTOM SECTION TABLE DENORMALIZATION APPROACHES .....</b>                        | <b>10</b> |
| 1. Direct-Map Approach: Custom Sections Map Directly to a Table .....                         | 10        |
| Limitation: Extended Attribute Data Types Must Match for All Rows in a Column .....           | 11        |
| 2. Pivot Approach: Custom Sections Pivot onto Table.....                                      | 12        |
| Limitation: Extended Attribute Data Types Must Match for All Columns in a Row .....           | 13        |
| Limitation: Repeatable Rows Cannot be Included for Denormalization for a Pivot Approach ..... | 14        |
| <b>CHAPTER 4—BASE DATA TYPES.....</b>   | <b>15</b> |
| Extended Attribute Base Types.....  | 15        |
| Text Based Values .....   | 15        |
| Numeric Data and Base Unit of Measure Values.....   | 16        |
| Qualitative Lookup Limitation.....  | 17        |
| eQuestionnaire Limitation .....   | 17        |
| <b>CHAPTER 5—APPLICATION SUPPORT FOR DENORMALIZATION .....</b>                                | <b>18</b> |
| Configuration .....   | 18        |
| Custom Section Data Admin Screens.....  | 18        |
| Validate Button .....   | 22        |
| Row and Column Popup.....   | 23        |

|  |           |
|--|-----------|
| Changes to Denormalization Metadata .....                  | 25        |
| Security Roles .....                                       | 25        |
| Import/Export .....  | 25        |
| Configuration Settings .....                               | 26        |
| <b>CHAPTER 6—INSTALLATION .....</b>                        | <b>27</b> |
| Installing the scripts .....                               | 27        |
| Enabling Custom Section Denorm in the Application .....    | 27        |
| User Access .....  | 27        |
| Feature Configuration .....                                | 27        |
| Near Real-Time Denorm configuration and installation ..... | 28        |
| <b>CHAPTER 7—EXECUTION .....</b>                           | <b>32</b> |
| Execution Scripts .....                                    | 32        |
| Table Creation Script (DENORM_GEN_DDL) .....               | 32        |
| Data Denormalization Script (DENORM_PROC_MAPPING) .....    | 35        |
| Monitoring Batch Status .....                              | 39        |
| Monitoring Near-Real Time Denorm Status .....              | 39        |

## Preface

### Audience

This guide is intended for client programmers involved with integrating Oracle Agile Product Lifecycle Management for Process. Information about using Oracle Agile PLM for Process resides in application-specific user guides. Information about administering Oracle Agile PLM for Process resides in the *Oracle Agile Product Lifecycle Management for Process Administrator User Guide*.

### Variability of Installations

Descriptions and illustrations of the Agile PLM for Process user interface included in this manual may not match your installation. The user interface of Agile PLM for Process applications and the features included can vary greatly depending on such variables as:

- Which applications your organization has purchased and installed
- Configuration settings that may turn features off or on
- Customization specific to your organization
- Security settings as they apply to the system and your user account

### Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

### Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

### Software Availability

Oracle Software Delivery Cloud (OSDC) provides the latest copy of the core software. Note the core software does not include all patches and hot fixes. Access OSDC at:

<http://edelivery.oracle.com>

## Chapter 1—Overview

Custom Sections provide a very flexible approach to managing customer information in the application. A Custom Section is made up of rows and columns; each cell may contain an Extended Attribute, which is a structured data type (such as Numeric, Text, etc.). There are many different Extended Attribute types that can be used, and each one is stored in a different database table, and may also reference other tables (such as Units of Measures, or list of data). As such, writing database queries against these data structures can be very complex and time-consuming, often involving over 20 database tables.

The Custom Section Denormalization (CS Denorm) feature builds and populates simple data structures for Custom Sections, which are easier to understand and report against while providing improved query performance. Data from the core extended attribute and Custom Section detail tables is denormalized into a single database table for a given Custom Section. Customers select which Custom Sections (and which rows and columns) to denormalize and indicate how the target database table(s) should be structured. The CS Denorm process then reads this information, pulls the relevant Custom Section data from specifications (or other business objects), and populates that data into a single, simplified database table created solely for that Custom Section.

Customers that leverage CS Denorm for reports will benefit from:

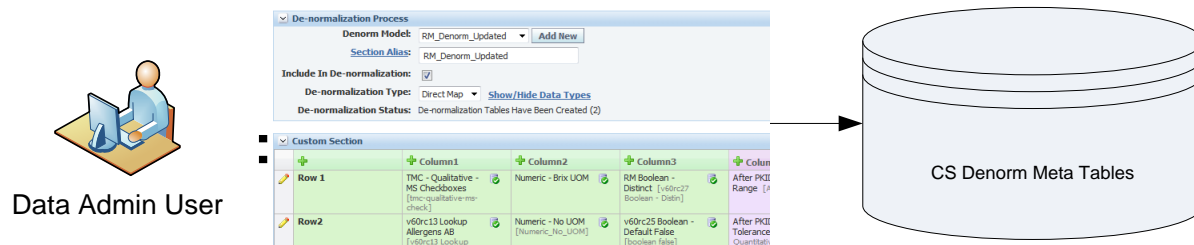
- **Reduced complexity** – Report writers no longer need to understand the complex Custom Section and Extended Attribute internal data structures.
- **Lower development cost & improved delivery time** – Since the denormalized data for a Custom Section is stored in a single table, the SQL needed is very easy to write. Report writing is simpler and easier to maintain.
- **Improved query performance** – Since the data for a given custom section can now be accessed via one simple database table, there can be enormous performance improvements.

## Chapter 2—Denormalization Process

There are three main steps to the CS Denorm solution: Configuration, Table Creation, and Denormalization.

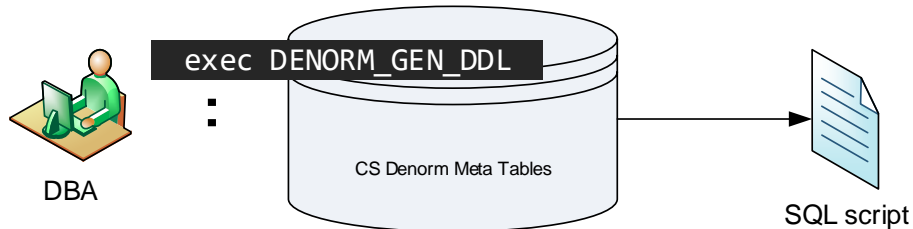
### 1. Configuration

The Data Administration user interface for Custom Sections allows users to specify if the Custom Section should be denormalized, which rows and columns should be included, the Denormalization approach, and other details. See [Application Support](#) for more details.

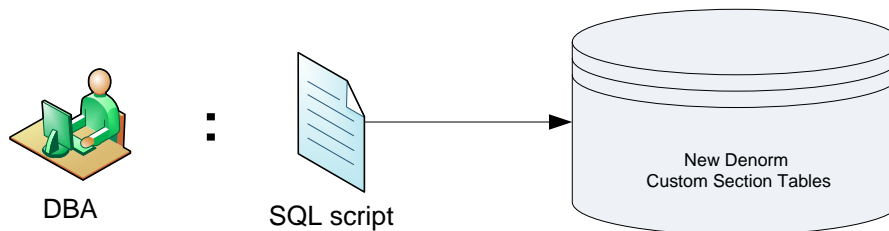


### 2. Table Creation

A database script reads the configuration data specified in the Data Admin screens and dynamically generates SQL code to create a single database table for each Custom Section flagged for denormalization. This is a manual process in which a database administrator runs the stored procedure and saves the result output as a script to generate the new Custom Section tables.



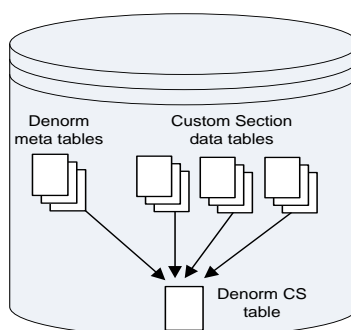
The script is then be used to generate the Denormalized Custom Section tables.



### 3. Denormalization

The Denormalization process runs on a regular basis to pull the custom section data from GSM Specifications, Sourcing Approvals, etc, and populates that data into the tables created for denormalization.





**1 - The data from the denormalization metadata tables is combined with the various Custom Section and Extended Attribute tables in PLM4P and is then pushed into the single table created for the given Custom Section**

There are two denormalization approaches: Near real-time denormalization and batch denormalization:

### Near Real-Time Denormalization (NRTD)

When a business object (such as a GSM specification) is saved, the Near Real-Time Denormalization (NRTD) process will determine if any of the custom sections flagged for denormalization on that item have been added, deleted, or modified. If so, a custom section denormalization request is added to a queue, which is then processed by a service on the Remoting Container application. This service will trigger a denormalization process for each request in the queue, which processes only the relevant custom sections on that specific business object. Since the denormalization is only occurring for small sets of data, the NRTD Service can run on an ongoing basis, keeping the denormalized data very closely in synch with the live data.

The frequency of the NRTD Service can be configured, along with options for logging, email notifications (if any errors occur), and more. See [Near Real Time Denorm configuration and installation](#) for details.

### Batch Denormalization

Prior to the Near Real Time Denormalization, denormalization was processed in a scheduled batch manner; all denormalized custom sections for all business objects that have been updated since the last batch denormalization were processed together. While this process should NO LONGER be required if using NRTD, there may be ONE-OFF instances where clients still wish to use this. As such, it is still available for use.

A stored procedure (DENORM\_PROC\_MAPPING) is available that processes each custom section set up for denormalization and pushes the relevant data from all objects that have been modified since the last run into the target tables. At times, this may denormalize custom section data that has NOT been changed, since it only evaluates if the specification (or other owning object) has been saved since the last denorm run. Do consider this process as a low priority because of a performance concern.

Log entries may optionally be written to a database table to record the execution results of each batch denormalization run.

Additional Details can be found in the [Data Denormalization Scripts](#) section below.

## Chapter 3—Custom Section Table Denormalization Approaches

When designing each custom section and configuring it for denormalization, users must determine how the denormalized table should be created. This decision may ultimately affect the way the custom section is set up. There are two different ways that a custom section can be denormalized: **Direct-map** and **Pivot**.

### 1. Direct-Map Approach: Custom Sections Map Directly to a Table

This approach converts a given custom section into a table structure that mimics the view of a custom section in the UI of the application, where columns in the UI correspond to columns in the table, and row data in the UI correspond to entries in the table.

Assuming the following custom section template #10001:

|      | Column1        | Column2        | Column3 |
|------|----------------|----------------|---------|
| Row1 | Numeric EA+UOM | Range EA+NoUOM | Text EA |
| Row2 | Numeric EA+UOM | Range EA+NoUOM | Text EA |

And assuming it appears on a specification (PKID: 1004-32-232ef) like so:

|      | Column1 | Column2                           | Column3 |
|------|---------|-----------------------------------|---------|
| Row1 | 5 g     | Min: 2.5<br>Target: 5<br>Max: 7.5 | Hello   |
| Row2 | 10 g    | Min: 25<br>Target: 50<br>Max: 75  | World   |

A table named DENORM\_CS\_10001 will be created with the following structure:

| Column      | Position | Type         | Nullable? |
|-------------|----------|--------------|-----------|
| fkOwner     | 1        | CHAR (40)    | NO        |
| ROW_NAME    | 2        | VARCHAR(500) | NO        |
| Column1     | 3        | FLOAT        | YES       |
| Column1_UOM | 4        | VARCHAR(32)  | YES       |

| Column         | Position | Type         | Nullable? |
|----------------|----------|--------------|-----------|
| Column2_Min    | 5        | FLOAT        | YES       |
| Column2_Target | 6        | FLOAT        | YES       |
| Column2_Max    | 7        | FLOAT        | YES       |
| Column3        | 8        | varchar(500) | YES       |

The custom section appearing on the specification will be represented by the following two lines in the newly created denorm table:

| fkOwner          | ROW_NAME | Column1 | Column1_<br>UOM | Column2_<br>Min | Column2_<br>Target | Column2_<br>Max | Column3 |
|------------------|----------|---------|-----------------|-----------------|--------------------|-----------------|---------|
| 1004-32-232ef... | Row1     | 5       | g               | 2.5             | 5                  | 7.5             | Hello   |
| 1004-32-232ef... | Row2     | 10      | g               | 25              | 50                 | 75              | World   |

Note that the name of the rows and columns for the denormalized table can be configured in the Data Admin UI.

### Limitation: Extended Attribute Data Types Must Match for All Rows in a Column

Custom sections allow customers to setup a different extended attribute (EA) types in each cell. This means a column could have different data types as you go further down in the column. A custom section that contains different EA type base data types in the same column cannot be de-normalized using this approach as the resulting table would have a column of unknown type.

Given the following custom section #1002:

|      | Column1             |
|------|---------------------|
| Row1 | Numeric<br>EA+NoUOM |
| Row2 | Text EA             |

A table named DENORM\_CS\_10002 cannot be created as the data type of Column1 is unknown:

| Column   | Position | Type                      | Nullable? |
|----------|----------|---------------------------|-----------|
| fkOwner  | 1        | CHAR (40)                 | NO        |
| ROW_NAME | 2        | NVARCHAR(500)             | NO        |
| Column1  | 3        | NVARCHAR(max)<br>OR FLOAT | YES       |

**NOTE:** Some different extended attributes types have the same base data type; for instance, a Text and a Qualitative extended attribute both have a base type of *String*, and therefore *can* be denormalized together. See [Chapter 4—Base Data Types](#) below. The Data Admin user interface will inform users when there are data type conflicts.

## 2. Pivot Approach: Custom Sections Pivot onto Table

This approach “pivots” a given custom section into a table structure where each row turns into a column and each column is represented by a row.

Assuming the following custom section template #10003:

|      | Column1        | Column2        | Column3        |
|------|----------------|----------------|----------------|
| Row1 | Text EA        | Text EA        | Text EA        |
| Row2 | Numeric EA+UOM | Numeric EA+UOM | Numeric EA+UOM |

And assuming it appears on a specification (PKID: 1004-32-232ee) like so:

|      | Column1 | Column2 | Column3 |
|------|---------|---------|---------|
| Row1 | Hi      | Hello   | Hiyas   |
| Row2 | 10 g    | 15 mg   | 20 kg   |

A table named DENORM\_CS\_10003 will be created with the following structure:

| Column      | Position | Type          | Nullable? |
|-------------|----------|---------------|-----------|
| fkOwner     | 1        | CHAR (40)     | NO        |
| COLUMN_NAME | 2        | NVARCHAR(500) | NO        |

| Column   | Position | Type          | Nullable? |
|----------|----------|---------------|-----------|
| Row1     | 3        | NVARCHAR(max) | YES       |
| Row2     | 4        | FLOAT         | YES       |
| Row2_UOM | 5        | VARCHAR(32)   | YES       |

The data on the custom section will be represented thus:

| fkOwner       | COLUMN_NAME | Row1  | Row2 | Row2_UOM |
|---------------|-------------|-------|------|----------|
| 1004-32-232ee | Column1     | Hi    | 10   | g        |
| 1004-32-232ee | Column2     | Hello | 15   | mg       |
| 1004-32-232ee | Column3     | Hiyas | 20   | kg       |

#### Limitation: Extended Attribute Data Types Must Match for All Columns in a Row

A custom section that contains different EA types in the same row cannot be de-normalized using this approach as the resulting table would have a column of unknown type.

Given the following custom section #1002:

|      | Column1          | Column2 |
|------|------------------|---------|
| Row1 | Numeric EA+NoUOM | Text EA |

A table named DENORM\_CS\_10002 cannot be created as the data type of Row1 is unknown:

| Column      | Position | Type                       | Nullable? |
|-------------|----------|----------------------------|-----------|
| fkOwner     | 1        | CHAR (40)                  | NO        |
| COLUMN_NAME | 2        | NVARCHAR(500)              | NO        |
| Row1        | 3        | NVARCHAR(max)<br>OR FLOAT? | YES       |

**NOTE:** Some different extended attributes types have the same base data type; for instance, a Text and a Qualitative extended attribute both have a base type of String, and therefore *can* be denormalized together. The Data Admin user interface will inform users when there are data type conflicts.

**Limitation: Repeatable Rows Cannot be Included for Denormalization for a Pivot Approach**

A custom section that contains repeatable rows cannot be de-normalized using the pivot approach. When pivoting, a row turns into a column in the created database table; since all columns must be defined to generate the CREATE TABLE syntax, repeatable rows would break that schema.

## Chapter 4—Base Data Types

This section describes how the custom section data is denormalized for different extended attribute types and values

### Extended Attribute Base Types

The following extended attribute types are considered to have the same base data type and therefore can exist in the same denormalized column:

| TEXT                         | NUMERIC            | BOOLEAN            |
|------------------------------|--------------------|--------------------|
| Free Text                    | Numeric            | Boolean            |
| Qualitative                  | Calculated Numeric | Calculated Boolean |
| Qualitative Lookup           |                    |                    |
| Calculated Text              |                    |                    |
| Long Free Text               |                    |                    |
| Multi-lingual Free Text      |                    |                    |
| Long Multi-lingual Free Text |                    |                    |

All other extended attribute types are considered to have a unique base data type and therefore cannot be combined in the same denormalization column.

### Text Based Values

Text, Calculated Text, Long Free Text, Qualitative, and Qualitative Lookup extended attributes are denormalized using the English only values. Multi-select values are listed as one record, in comma delimited format.

Multi-lingual Free Text and Long Multi-lingual Free Text are controlled by configuration `Denorm.AllowMultiLanguage`. You can use `SetupAssistant.exe` to maintain this configuration value.

## Numeric Data and Base Unit of Measure Values

When executing the CS Denorm process, there is an option to include a conversion to the base Unit of Measure for an extended attribute. When adding numeric data for a custom section in the user interface, users may choose different UOM values.

Given the following example in the UI:

|      | Column1 |
|------|---------|
| Row1 | 0.5 g   |
| Row2 | 1000 mg |

There are two ways of denormalizing the data:

- 1) **Excluding base UOM values** – this would result in a table similar to the UI, in which the UOM column would keep the UI values:

|      | Column1 | Column1_UOM |
|------|---------|-------------|
| Row1 | 0.5     | g           |
| Row2 | 1000    | mg          |

- 2) **Including the base UOM values** – this would result in a table like above, but also adds columns for Base UOM support:

|      | Column1 | Column1_UOM | Column1_BASE | Column1_UOM_BASE |
|------|---------|-------------|--------------|------------------|
| Row1 | 0.5     | g           | .5           | g                |
| Row2 | 1000    | mg          | 1            | g                |

Likewise, Quantitative Range and Quantitative Tolerance extended attributes also can include Base UOM information for their Min, Max, and Target values.

See the [Execution](#) section for details.



### Qualitative Lookup Limitation

When configuring a Qualitative Lookup in Data Admin, an *internal* PLM4P category, such as Allergens, can be selected. However, the Custom Section Denorm process currently only supports denormalization of the following *internal* lookup categories:

- Countries
- Additives
- Allergens
- Intolerances

Alternatively, external lookup categories, as available via custom lookups are fully supported.

### eQuestionnaire Limitation

The CS Denorm process does not include custom section data from questionnaires in EQ. Once a questionnaire is imported into a GSM specification, however, the custom section data from the specification does get included.

## Chapter 5—Application Support for Denormalization

### Configuration

Use the following Feature Configuration entry to enable the CS Denorm features:

**Portal.DataAdmin.CustomSection.De-normalizationProcess.Enabled**

Setting this configuration entry value to true will display the new denormalization sections in Data Admin. At least one row and one column will need to be marked for denormalization in the custom section.

The following feature configuration is used to enable the Near Real Time Denormalization process:

**Common.CustomData.NearRealTimeDenormRequest.CustomSections.Enabled**

Setting this configuration value to true will begin generating denormalization requests when objects with denormalized customs sections are updated.

### Custom Section Data Admin Screens

The Data Administration of custom sections allows users to specify which custom sections they want denormalized, which rows and columns should be included, the denormalization approach, and the names of the database table, columns, and row values that will be used when the tables are created.

A section titled “De-normalization Process” is visible in the custom section data administration screens if the feature configuration is enabled.

De-normalization Process

Denorm Model: Example\_CS\_Denorm [Add New](#)

Section Alias: Example\_CS\_Denorm

Include In De-normalization: ☒

De-normalization Type: Direct Map [Show/Hide Data Types](#)

De-normalization Status: Will Be Included in the Denorm Process When Activated (0)

### Denorm Model

A Denorm Model represents how the custom section should be denormalized. Multiple denorm models can be defined, with each model resulting in a different database table, with different columns and rows selected for denormalization. The drop down field allows you to switch between the defined denorm models, if any. Denorm models are named by the Section Alias field below.

Once a valid denorm model has been created and validated, additional denorm models can be created by clicking the Add New button. Clicking Add New will create a new denorm model, clearing the rest of the fields from the form so that the new table details can be specified.

## Section Alias

This field determines the table name that will be used for the denormalized custom section table. The value must be unique across all custom sections, and must be no longer than a configurable number of characters (default is 30). The configuration setting, `AliasMaxLength`, is in the `ExtendedAttributeSectionBuilderService` configuration node in the `CustomerSettings.config` file. See the [Configuration Settings](#) section for details.

### Section Alias rules

The **section alias value must adhere to database vendor specific rules**. For instance, Oracle DB requires table names to be 30 characters or less, while SQL Server does not have such a restriction; certain reserved words cannot be used; etc.

Note that while table names for an Oracle database must be 30 characters or fewer, the CS Denorm process actually creates a version of the table to process the denorm data, which is named “STG\_<table alias>”. Therefore, **when using an Oracle database, be sure to restrict the Section Alias to 26 characters or fewer**. This may be done in two ways: (1) Setting the `AliasMaxLength` configuration value. However, this would also enforce the alias length restriction on row and column aliases. (2) Adding a Custom Validation for the length of the section alias value.

Custom validation using the Validation Framework can be added to enforce any desired restrictions. See the [Validation](#) section.

**Users should take care to avoid name collisions with existing PLM4P internal tables. The table generation stored procedure will indicate if a conflict exists.**

Clicking the Section Alias link will copy the custom section name into the field, replace any spaces with underscores (‘\_’) and trim the name to the configurable max length.

## Include In De-normalization

This checkbox marks a custom section to be included in the denormalization process, once the custom section is made Active.

### Un-checking an already denormalized Custom Section














If a custom section has already been denormalized, and this checkbox is unchecked, then the generated denormalized custom section table will no longer be included in the Data Denormalization process. Additionally, the next time the Table Creation process is performed this denormalized custom section table will be deleted from the database.

## De-normalization Type

Selects the type of denormalization used for this custom section:

- **Direct Map** – The custom section is created as a table structure that mimics the view of a custom section in the UI of the application, where columns in the UI are mapped as columns in the database table. Using this approach requires that the base extended attribute data types must match for all rows (marked for denorm) in a column (marked for denorm). For examples, you cannot have a Numeric extended attribute in column 1, row 1, and a Date extended attribute in column 1, row 2. In this scenario, the pivot de-normalization type may be used. See the [Custom Section Denormalization Approaches](#) section for details.
- **Pivot** – This approach "pivots" a given custom section into a table structure where each row turns into a column and each column is represented by a row. Repeatable rows cannot be de-normalized using the pivot approach. Using this approach requires that the base extended attribute data types must match for all columns (marked for denorm) in a row (marked for denorm). See the [Custom Section Denormalization Approaches](#) section for details. Note that this option can be removed altogether or restricted to users in certain groups, by modifying the configuration settings for the ExtendedAttributeSectionBuilderService configuration node in the CustomerSettings.config file. See the [Configuration Settings](#) section for details.
- **None** – Prevents this section from *ever* being denormalized. Once a section has been marked with a denormalization approach of None, it *cannot be changed*. Note that this option is turned off by default, but can be enabled for all users or restricted to users in certain groups, by modifying the configuration settings for the ExtendedAttributeSectionBuilderService configuration node in the CustomerSettings.config file. See the [Configuration Settings](#) section for details.

To see the base data types of the custom section cells, click the **Show/Hide Data Types** link. This will change the custom section grid to display a color-coded base data type for each cell, to help plan the denormalization approach. Clicking the link again will restore the original view.

| Custom Section  |   |  |   |   |  |
|---|---|--|---|---|--|
|   | + | + Column1  | + Column2   | + Column3   | + Column 4  |
|  Row 1 |   | [Text]  | [Numeric]  | [Boolean]  | [Quantitative Range]   |
|  Row2  |   | [Text]  | [Numeric]  | [Boolean]  | [Quantitative Tolerance]   |
|  Row 3 |   | [Text]  | [Numeric]  | [Boolean]  | ---  |

## De-normalization Status



This read-only field displays the custom section's current denormalization process status. The numeric value in the parenthesis corresponds to the metadata table DENORM\_CS\_TABLE status value for reference purposes.













The User Interface uses the Include in De-normalization checkbox *and* the custom section's status to workflow the denormalization status. That is, the denormalization status depends on the custom section status, as follows:

| <i>Denorm<br/>Checkbox<br/>status</i> | <i>Custom Section<br/>Status</i> | <i>Denorm Status</i>             | <i>Notes</i>   |
|---------------------------------------|----------------------------------|----------------------------------|--|
| <b>Unchecked</b>                      | any status                       | -1: Not for Denormalization      | The Custom Section will not be denormalized  |
| <b>Checked</b>                        | New                              | 0: Not ready for denormalization | Denorm meta data (alias, etc) is stored, but the Custom Section is ignored by the denorm scripts. Once the Custom Section is made Active, this status is promoted to a 1.  |
| <b>Checked</b>                        | Active, Archived, Inactive       | 1: Ready for Table generation    | A Denormalized table can now be created for the Custom Section. Once the table generation scripts are executed, the status is moved to <b>2: Ready for denormalization</b> |
| <b>Checked</b>                        | Active, Archived, Inactive       | 2: Ready for denormalization     | Custom Sections in this status will now be picked up in the Data Denormalization process   |

### Denormalization Icons

Custom Section cells that are selected for de-normalization (for the given Denorm Model) will display one of two new icons:

- The “Include in De-normalization” icon  will be displayed if both the column and row for that cell are marked as Include In Denorm, but their statuses are not Ready For Denormalization (meaning those columns and rows have not been created in the new denorm table.)
- The “Denormalized” icon  will display when both the column and row have been created in the denormalized table.

| Custom Section   |   |  |   |  |
|--|---|--|---|--|
|  | + | + Column1  | + Column2   | + Column3  |
|  <b>Row 1</b> |   | TMC - Qualitative - MS Checkboxes [tmc-qualitative-ms-check]  | Numeric - Brix UOM                 | RM Boolean - Distinct [v60rc27 Boolean - Distin]  |
|  <b>Row2</b>  |   | v60rc13 Lookup Allergens AB [v60rc13 Lookup Allergens]        | Numeric - No UOM [Numeric_No_UOM]  | v60rc25 Boolean - Default False [boolean false]   |
|  <b>Row 3</b> |   | Free Text - Multiple Line [free text_multi]                   | Numeric - Mass [numeric_mass]      | Test Boolean [T_BI]                               |

A legend at the bottom of the grid describes the icons to the user.

**Note:** The Extended Attribute's Attribute ID value will be displayed in brackets after the EA name.

Clicking a cell with an extended attribute will launch a read-only popup displaying the extended attribute information.

### Validate Button

The Validate button performs validation of the custom section data and validation of the denormalization data. This allows users to check the setup of their custom sections prior to saving. Warning messages appear in the Warnings grid at the top of the page.

The Validate button also calls to the Validation Framework to see if there are any additional custom validation rules to evaluate. Customers can add their own validation rules for custom sections and for the custom section denorm information.

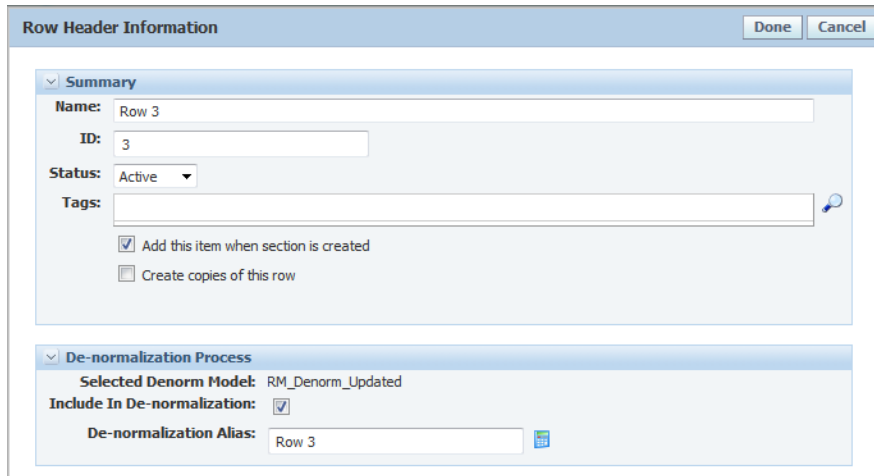
- Validation for custom section data uses rule of type "1020", and is available for save and validate events in the ValidationSettings.xml file.
- Validation for custom section denorm data uses rule of type "denorm1020", and is available for save and validate events in the ValidationSettings.xml file.
- See the Validation Training documentation for more details.

#### **IMPORTANT:**

Once a custom section has been denormalized, all changes to it are locked down. See the [Security Roles](#) section for details.

## Row and Column Popup

A section titled “De-normalization Process” is included in the Row and Column Header Information data admin screens.



The screenshot shows a 'Row Header Information' dialog box. It has a title bar with 'Done' and 'Cancel' buttons. The dialog contains two expandable sections. The 'Summary' section is expanded and shows fields for Name (Row 3), ID (3), Status (Active), and Tags. There are checkboxes for 'Add this item when section is created' (checked) and 'Create copies of this row' (unchecked). The 'De-normalization Process' section is also expanded and shows 'Selected Denorm Model' (RM\_Denorm\_Updated), 'Include In De-normalization' (checked), and 'De-normalization Alias' (Row 3).

The Row and Column Popups include the following fields.

### Denorm Model

Indicates the current Denorm Model that the column/row denormalization information is meant for.

### Include In De-normalization

This checkbox marks the Row or Column to be included in the denormalization process, once the Custom Section is made Active.

#### Un-checking an Already Denormalized Row or Column

If a custom section has already been denormalized, and this checkbox is unchecked, then this row or column will no longer be included in the data denormalization process. The custom section’s denormalization status is also workflowed back to Ready for Table Creation so that the next time the table generation process executes, the denormalized table will be recreated without this row or column.

### De-normalization Alias

When the denormalization approach is Direct Map, the De-normalization alias for the Column pop is restricted and must be no longer than a configurable number of characters (default is 30). When the denormalization approach is Pivot, the De-normalization alias for the Row pop (which will get pivoted to a database column) is restricted in the same way. The configuration setting, `AliasMaxLength`, is in the `ExtendedAttributeSectionBuilderService` configuration node in the `CustomerSettings.config` file. See the [Configuration Settings](#) section for details.

**Row and Column Alias Rules**

Note that while column names for an Oracle database must be 30 characters or fewer, the CS Denorm process actually can create additional columns for a given extended attribute type. For instance, a numeric extended attribute that has a UOM will have an additional UOM column added to the resulting table. The name of the column is the denorm alias name followed by one of the following, depending on the EA Type:

<alias>\_UOM – for Unit of Measure

<alias>\_TGT – Target

<alias>\_MIN – Min

<alias>\_MAX – Max

<alias>\_TOL – Tolerance

When choosing to denormalized and include BaseUOM values, the denorm alias becomes even longer:

<alias>\_UOM\_BASE – for Unit of Measure

<alias>\_TGT\_BASE – Target

<alias>\_MIN\_BASE – Min

<alias>\_MAX\_BASE – Max

Therefore, when using an Oracle database, be sure to restrict the Column or Row Alias to 26 characters or fewer, or 21 characters or fewer, if including BaseUOM values. This may be done in two ways: (1) Setting the AliasMaxLength configuration value. However, this would also enforce the alias length restriction on section, and row/column aliases, so this may be too restrictive. (2) Adding a Custom Validation for the length of the row/column alias value.

Custom validation using the Validation Framework can be added to enforce any desired restrictions. See the [Validation](#) section.

Clicking the De-normalization Alias link copies the row or column name into the field. When the denormalization approach is Direct Map, the copied column name replaces any spaces with underscores ('\_') and trim the name to the configurable max length. When the denormalization approach is Pivot, the copied Row name replaces any spaces with underscores ('\_') and trims the name to the configurable max length.

**Pivot**

Note that when using the Pivot denormalization approach, the *name of the row will become the column name* used in the denormalized table, and vice versa.



## Changes to Denormalization Metadata

Once a custom section has had its denormalized table created and is in a status of Ready for Denormalization, changes to it are locked down (See Security Roles below). If changes to the denormalization configuration are made, the status of the custom section denormalization will be reverted, and that custom section will stop being included in the denormalization process.

Changes to any of the following fields will revert the denormalization status:

- Section alias
- Row Alias
- Column Alias
- Denormalization Approach
- Unchecking the Include in Denorm checkbox
- Adding a new row or column for denormalization

## Security Roles

Two new security roles have been added to support the custom section denormalization feature:

1. The user must have the role of [**Custom\_Section\_Denorm\_Enabler**] to edit the custom section's "Include in De-normalization" checkbox.
2. Once a section has been activated (status of active, inactive, archived) and is flagged as Include in De-normalization, a user needs the role of [**Denormalized\_Custom\_Section\_Editor**] to make *any* edits to the custom section. If the user doesn't have this role they will not see the Edit action button. Instead they see the following information panel message. "This section has been marked for inclusion in the de-normalization process. You do not have the proper privileges to edit this Custom Section."

## Import/Export

The import and export process for custom section data includes any CS Denorm information configured.

When importing CS Denorm data, the Denorm status may be automatically adjusted for the Target environment.

For example, if the Denorm status of a custom section is "Ready for denormalization (2)" in the Source environment, it means that the denormalized table has been generated and the custom section's denormalized table is ready for denormalization data. In the Target environment, however, this may not be the case (e.g., the Custom Section didn't exist there, or is different from the Source environment, and therefore the denormalized table hasn't been created, etc.).

Therefore, *the import process may revert the Denorm Status* value back from [Ready for denormalization \(2\)](#) to [Ready for Table generation \(1\)](#), and add a notification for that custom section to the email message generated on import.

## Configuration Settings

Configuring de-normalization types is controlled in the CustomerSettings.config file, using the ExtendedAttributeSectionBuilderService settings. Permission for each de-normalization approach type can be set using the following options:

- **All** – All users have access to the de-normalization type
- **None** – No users have access to the de-normalization type
- **A UGM Group PKID list, separated by a pipe character ('|')** – Only users in the specified group(s) have access to the de-normalization type. For example:  

```
<envvar name="DenormType_NONE_Groups" value="20550b7c241e-ff3c-4cf8-b2bc-2a814195833c|2055c8000cde-5312-4ef8-910d-f60c4e849874" />
```

 will only allow users in the groups identified by these PKIDs to select None.

Additionally, the maximum length for the Denorm aliases (for the section, columns, and rows) can be declared here using the AliasMaxLength setting.

See the configuration example below.

```
<ExtendedAttributeSectionBuilderService configChildKey="name">
  <envvar name="MaximumNumberOfRowsThroughCopy" value="250"/>
  <envvar name="DenormType_PIVOT_Groups" value="All"/>
  <envvar name="DenormType_NONE_Groups" value="None"/>
  <envvar name="AliasMaxLength" value="30"/>
</ExtendedAttributeSectionBuilderService>
```

Note that the denormalization approach of Direct Map is always on, and therefore cannot be configured.

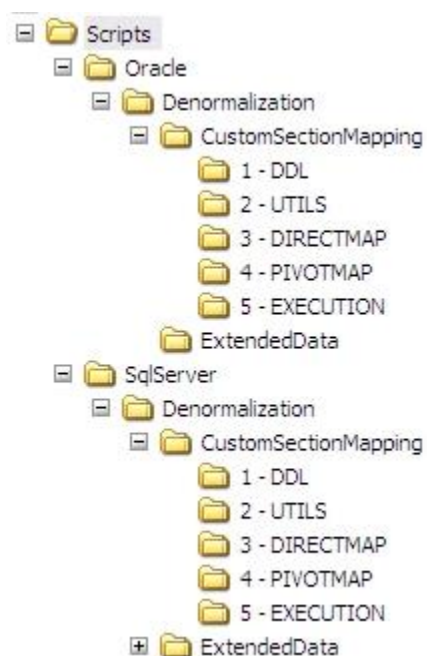
Changes to the CustomerSettings.config file require an IIS reset.

## Chapter 6—Installation

Several database scripts are provided as part of the release. All of these scripts must be executed in order to add the various stored procedures and functions to the database.

### Installing the scripts

Locate the Scripts directory in the release package. There are two different folders: Oracle and SqlServer. Open the folder that corresponds to the database provider you are using, then open the Denormalization folder, and the CustomSectionMapping folder within.



All of the scripts in the CustomSectionMapping folder must be compiled, in the order of the folder names.

See the [Execution](#) section for details on which scripts must be called to execute the denormalization process.

### Enabling Custom Section Denorm in the Application

#### User Access

Add the [**Custom\_Section\_Denorm\_Enabler**] role to the relevant group(s) and assign that group to the relevant user(s).

#### Feature Configuration

Set the feature configuration setting,

Portal.DataAdmin.CustomSection.De-normalizationProcess.Enabled, to true in the CustomerSettings.config file, in the FeatureConfig node:

```
<add key="Portal.DataAdmin.CustomSection.De-normalizationProcess.Enabled" value="true"
configDescription="Enables the configuration of custom section denormalization" />
```

Changes to the CustomerSettings.config file require an IISReset.

## Near Real-Time Denorm configuration and installation

The installation instructions for Near Real-Time Denorm follow.

**NOTE: IT'S STRONGLY RECOMMENDED TO UTILIZE THE NEAR REAL-TIME DENORM SOLUTION IN V6.2 OR ABOVE. THIS SOLUTION PERFORMS A DIFFERENTIATED MANAGEMENT FOR PERFORMANCE PURPOSE, BY SPLITTING THE HEAVY DENORM WORK TO MULTIPLE EXECUTION SESSIONS.**

1. Stop IIS.
2. Stop the Remoting Container.
3. Enable Real Time Denorm by enabling the feature configuration entries:  
Add the following entries to the config\Custom\CustomerSettings.config file, in the FeatureConfig node:

```
<add key="Common.CustomData.NearRealTimeDenormRequest.CustomSections.Enabled"
value="true" configDescription="Evaluates if denormalized Custom Section data has been
modified since load"/>
```

4. Add the following entry to the environmentvariables.config file, in the # Port Numbers section, replacing 8114 with a valid open/unused port:

```
PLM4P.CSDenormService.Port = 8114
```

5. Customize Real Time Denorm process configuration :  
The following configuration entries allow for customization of the Real Time Denormalization process:

```
<add key="PollingIntervalInSeconds_CustomSectionRequests" value="90" />
```

The frequency (in seconds) that custom section denorm requests are processed.

```
<add key="IncludeBaseUOMColumnsInCSDenorm" value="false" />
```

If the custom section denorm tables are generated with Base Unit of Measure columns, the custom section denorm process should be set to include Base UOM columns - change this to "true".

```
<add key="ErrorNotifyFromAddress" value="@@VAR:Prodika.From.EmailAddress@@" />
```

```
<add key="ErrorNotifyToAddress" value="@@VAR:Prodika.To.EmailAddress@@" />
```

Errors in Real Time Denorm process will generate an email containing error details, along with additional useful information. These configuration settings are used to indicate the sender and the recipient of these emails. The default setting will use the values entered in environmentvariables.config, but can be modified here to use different values.

```
<add key="MaxLogFilesToKeep_CS" value="30"/>
```

The Real Time Denorm process will log basic informational messages, as well as error messages, to a rolling file log in the Denorm\_CustomSections directory within the default Logs directory. A new log file will be generated each day. These configuration settings control the number of daily log files that will be kept.

To completely disable file logging, set the MaxLogFilesToKeep\_CS value to "0" (zero). Note that errors will still be emailed and an entry in the ProdikaCommon Event Log will also be added.

```
<add key="DaysForProcessedCSRequestExpiration" value="15"/>
```

This setting is used to clean up (remove) old, successfully processed denormalization requests (from denorm\_ea\_request and denorm\_cs\_request) after the configured number of days.

Set to zero to never remove the denorm request entries.

### **OPTIONAL CONFIGURATION:**

To change any of the previous settings copy the following XML and add it to the CustomerSettings.config file, in the CustomerSettings/Core node.

```
<DenormRequestProcessingServices>
  <AppSettings configChildKey="key">
    </AppSettings>
  </DenormRequestProcessingServices>
```

Then copy the desired entries from above into the AppSettings node and modify the settings as needed.

6. Enable Near Real-Time Denorm processing for each object type.

Update the \config\Extensions\CustomPluginExtensions config file to enable Real Time Denorm processing for each object type, inserting the following entries into the ValidatePlugins node

```
<Plugin name="NearRealTimeDenormRequestPlugin"
FactoryURL="Class:Xeno.Prodika.DenormServices.DenormChangePluginFactory,DenormServices"/>

<Plugin name="PostSaveSpecPlugin" inheritFromPluginName="NearRealTimeDenormRequestPlugin"
/> <!-- Enable Real Time Denorm Request processing for GSM Specs -->

<Plugin name="PostSavePQMItemPlugin"
inheritFromPluginName="NearRealTimeDenormRequestPlugin" /> <!-- Enable Real Time Denorm
Request processing for PQM Items -->

<Plugin name="PostSaveSCRMPlugin" inheritFromPluginName="NearRealTimeDenormRequestPlugin"
/> <!-- Enable Real Time Denorm Request processing for SCRM items -->
```

```
<Plugin name="PostSaveNPDProjectPlugin"
inheritFromPluginName="NearRealTimeDenormRequestPlugin" /> <!-- Enable Real Time Denorm
Request processing for NPD Projects -->

<Plugin name="PostSaveSmartIssuePlugin"
inheritFromPluginName="NearRealTimeDenormRequestPlugin" /> <!-- Enable Real Time Denorm
Request processing for Smart Issue Requests -->
```

7. Copy the  
`<P4P_Home>\Installer\Extensions\DenormServices\CSDenormService\DenormConfig.xml` file to  
the `\config\Extensions` directory.
8. Add the following entry to `EnvironmentSettings.config` in the  
`EnvironmentSettings/RemotingContainer/RemoteServices` node, making sure to set the `isActive`  
attribute to `true`.

```
<Service name="Custom Section Near Real Time Denorm Service"
port="@@VAR:Prodika.CSDenormService.Port@" isActive="true" />
```

9. Make a backup of the existing `RemotingContainer.exe.config` file in `RemotingContainer` and then  
replace the current version with the one in this directory. File `Sample-  
RemotingContainer.exe.config` is a sample.

- In the `configuration/configSections` node, add a `sectionGroup` entry for  
`DenormRequestProcessingServices`.

```
<sectionGroup name="DenormRequestProcessingServices">

    <section name="AppSettings"
type="Xeno.Prodika.XML.XMLMergeConfigHandler,CoreAppPlatform" />

</sectionGroup>
```

- In the `configuration` node, add the `DenormRequestProcessingServices` entry.

```
<DenormRequestProcessingServices>

    <AppSettings
handler="Class:System.Configuration.NameValueSectionHandler,System"
configReplace="file:%CONFIG_HOME%\environmentvariables.config"

configExtends="file:%CONFIG_HOME%\Extensions\DenormConfig.xml,/DenormRequestProcessingServ
icesSettings/DefaultSettings"

configExtends2="file:%CONFIG_HOME%\Custom\CustomerSettings.config,/CustomerSettings/Core/D
enormRequestProcessingServices/AppSettings">
```

```

</AppSettings>

</DenormRequestProcessingServices>

```

- In configuration/Prodika.RemotingContainer/RemoteServices, add a Service entry for Custom Section Near Real Time Denorm Service.

```

<Service
    name="Custom Section Near Real Time Denorm Service"

type="Type:Xeno.Prodika.DenormServices.CustomSectionDenormRequestProcessorService,DenormSe
rvices"

    mode="Singleton" />

```

10. Copy the <P4P\_Home>\SharedLibs\DenormServices.dll file from the SharedLibs directory and place it in the following folders:

```

RemotingContainer\dependentAssemblies
web\gsm\bin
web\pqm\bin
web\scrm\bin
web\npd\bin
web\SupplierPQM\bin
web\eq\bin

```

11. Restart IIS.

12. Restart the Remoting Container.

To verify the service is running, you can check the Logs\Denorm\_CustomSections directory (if logging is enabled), or the ProdikaCommon Event log (look for a “CustomSectionDenormRequestProcessorService started” event). Once the Remoting Container is started, the denorm requests get processed after the configured number of seconds specified in the configuration.

## Chapter 7—Execution

Once the database scripts have been added to the database, and the Feature Configuration entry has been set to true, the application can be restarted and custom sections can be configured for denormalization in the UI, as described earlier.

When the custom section denormalization UI configuration is completed and the custom section is made Active, it can be assigned to GSM specification, SCRM sourcing approvals, etc. This custom section data is now available for denormalization, and, at this point, the CS Denorm process can be run to yield results.

Executing CS Denorm involves two functional processes: [Table Creation](#) and [Data Denormalization](#). The following section describes these processes by their related database stored procedures.

### Execution Scripts

There are two main stored procedures that are used to execute the CS Denorm process:

DENORM\_GEN\_DDL and DENORM\_PROC\_MAPPING.

(These stored procedures, in turn, call several other included stored procedures and functions, which are not detailed here as clients should not be calling them directly.)

#### **Table Creation Script (DENORM\_GEN\_DDL)**

**NOTE: PLEASE ONLY DO THIS IN A WELL-PLANNED MAINTENANCE PERIOD.**

Executing the DENORM\_GEN\_DDL stored procedure generates SQL code for creating the denormalized tables for all custom section templates that have a status of “Ready for Table Generation”. Note that executing this stored procedure DOES NOT actually create any tables, but rather generates the SQL code required to generate the tables.

The results of the DENORM\_GEN\_DDL should be saved and reviewed for any errors that may have been raised. Prior to executing the results of the DENORM\_GEN\_DDL, the SQL produced should be parsed (in SQL Server, the checkmark button next to the Execute button) or compiled (in Oracle).

The text returned by the stored procedure must be executed by someone that has CREATE TABLE and DROP TABLE privileges in the target database.

Please note that this stored procedure DOES NOT create any indexes on the table being generated as that is dependent on data (amount, type, variance, etc.) and searching (frequency, type of operators, etc.) characteristics which are highly specific to a customer’s environment. A DBA should be involved for further performance tuning.

The stored procedure has an optional input parameter @INCLUDE\_BASE\_UOM indicating if the table generation should include base UOM columns.



The second optional input parameter `@RUN_DENORM_IMMEDIATELY` determines if the batch denorm process should get called immediately after table generation. This is needed if there is custom section data on business objects already, but that custom section was previously not selected for denormalization – now that the custom section id being denormalized, that already existing data needs to get pulled in. Also, when custom section template schema is modified after data denormalization, the corresponding existing data needs a refresh. The default value of 1 will trigger the batch denorm automatically.

For example:

```
EXEC DENORM_GEN_DDL 0
```

Generates the SQL to create the denormalized tables *but will not* include columns for base UOM values. Immediate batch denormalization will be triggered, even without explicitly including the second parameter.

```
EXEC DENORM_GEN_DDL 1
```

Generates the SQL to create the denormalized tables and *will* include columns for base UOM values.

```
EXEC DENORM_GEN_DDL 0, 1
```

Generates the SQL to create/rebuild the denormalized tables, *will not* include columns for base UOM values, and will trigger immediate batch denormalization.

---

**\*\* SQL Server Users** – be sure to generate the results of this stored procedure to text (not grid) and increase the setting for “Maximum number of characters displayed in each column” to 8192. This setting is found in the Tools > Options popup window, under Query Results-> SQL Server -> Results to Text.

## Tables Generated

The `DENORM_GEN_DDL` stored procedure produces SQL code that is used to create *two* tables for each denormalized custom section:

1. The target denormalization table, as specified by the section alias field in the UI.
2. A copy of that target table with an “STG\_” prefix. This “STG\_” table is later used in the denormalization process to store and process updated records prior to inserting those records into the final target table.

## Table Name Conflicts

The table name is evaluated to determine if it conflicts with any of the Agile PLM for Process internal tables. If a conflict exists, the process terminates and an error is raised:

\*\*\*ERROR\*\*\* Denormalization Table Name <table name> is an Internal PLM4P table, and is not allowed! Table skipped.

When this error occurs, the table generation process is terminated. Additionally, an error is logged to the DENORM\_LOG table with a value of “CS Denorm Table Generation ERROR” in the MODULE column.

See the DENORM\_PROC\_MAPPING [Implementation Details](#) section for more.

Customers can perform additional validations on the table names (or any property of the CS Denorm custom section information), if desired, by using the Validation Framework. This would provide the user configuring the custom section immediate feedback in the UI.

## Implementation Details

*This section describes some of the technical details of the table generation process, in order to provide some transparency and insight into the process. This level of detail is not required to run the CS Denorm process, but may be useful to better understand performance characteristics or to troubleshoot the process if any errors occur.*

The table generation process works as follows:

### Determines which custom sections to process

The DENORM\_GEN\_DDL stored procedure first retrieves all custom section entries with a status of 1 (Ready for table generation) from the DENORM\_CS\_TABLE table, and then produces SQL code to generate CREATE TABLE commands for each custom section table, one at a time.

For each custom section table, it does the following:

**Ensures the table name is not an Agile PLM4P Table** (see the [Table Name Conflicts](#) section above for details)

### Performs a last stage validation

Evaluates the rows and columns for the current custom section to ensure the custom section rows and columns are using data type consistency, as performed in the UI.

### Constructs the CREATE TABLE SQL statements

The columns for direct map or rows for pivot (from the denorm\_cs\_column and denorm\_cs\_row tables) are examined to create the CREATE TABLE SQL command, used to create the denormalized custom section table and the individual columns. All rows and columns with a denormalization status of Ready for Table Generation or Ready for Denormalization are included. The CREATE TABLE command is preceded by a DROP TABLE which will be called if the custom section table already exists.

If the INCLUDE\_BASE\_UOM input parameter is set to 1 (true), then the table creation scripts also include columns for the base value(s) and base UOM entries.

**Updates the denormalization status**

The SQL code generated also includes statements to update the denormalization status of the custom section, rows, and columns, to the status of Ready for Denormalization (2).

**Prints the resulting SQL commands**

Once all custom sections that had a “Ready for Table Generation” denormalization status have been processed, the table generation process concludes by dropping unnecessary tables.

**Constructs DROP TABLE statements**

Previously denormalized custom sections that are have generated their denormalization table but are now marked with a Not for Denormalization (-1) status are included as DROP TABLE commands.

**Data Denormalization Script (DENORM\_PROC\_MAPPING)**

**NOTE: PLEASE ONLY DO THIS IN A WELL-PLANNED MAINTENANCE PERIOD.**

**Executing the DENORM\_PROC\_MAPPING stored procedure is NOT required when using Near Real-Time Denorm process. There may be ONE-OFF instances where clients still wish to use this. As such, it is still available for use.**

Executing the DENORM\_PROC\_MAPPING stored procedure will perform the denormalization of custom section data from the various internal PLM4P tables into the newly created denorm tables created by the DENORM\_GEN\_DDL results. The stored procedure processes ANY custom section templates that have a status of “Ready for Denormalization”, and includes all of that custom section’s rows and columns that have the same denormalization status.

The stored procedure will perform a “delta” denormalization, retaining the existing records and adding any records that may have been updated since the last denormalization time.

The stored procedure has three optional input parameters:

1. **debug\_level** – This parameter sets the level of debug statements that will be printed to the screen when executing the stored procedure manually. The debug messages are not logged to a file.

Valid values are:

**0:** (default) No debug statements are printed.

**1:** Some minor debug statements are printed indicate general status messages.

**2:** Very detailed debug statements are printed that includes the SQL that is being created and executed to denormalize the data. This setting should only be used if unexpected errors occur and their output will be helpful to Customer Support.

2. **log\_level** – This parameter sets the level of logging statements that will be inserted into the **DENORM\_LOG** table.

Valid values are:

**0:** (default) No log entries are added.

**1:** Will log each custom section being denormalized.

**2:** Same as 1, but adds a log message for any skipped custom sections. Custom sections in which none of its owning objects (specification, sourcing approval, etc) have been updated since the last denorm execution will be skipped.

The number of rows written to this table will depend on the number of custom sections being denormalized, and the frequency of denormalization.

3. **INCLUDE\_BASE\_UOM** – this parameter indicates if the denormalization process should include Base UOM values. *This value must be the same one used in the DENORM\_GEN\_DDL stored procedure, or errors will occur.*

Valid values are:

**0:** (default) Base UOM data *not* is denormalized.

**1:** Base UOM data is denormalized.

For example:

**EXEC DENORM\_PROC\_MAPPING 2,2,0**

Will run the CS Denormalization for all custom section templates (in a Ready for Denorm status), printing out all debug statements, adding all logging statements to the DENORM\_LOG table, and will ignore base UOM data.

The DENORM\_PROC\_MAPPING stored procedure can be run manually, or be scheduled to run on a recurring basis using the Database server tools (such as SQL Server Agent for SQL Server).

*Note that performance can be greatly improved by increasing the number of processors and RAM available to the database server.*

### Performance considerations

Before setting up denormalization, DBAs must understand the runtime characteristics of their routine. At a minimum, they need to understand how long the routine will run and what impact it will have on users. Database server hardware makes a *very significant* impact on runtime performance of the CS Denorm process.

Additionally, the first run of denormalization for a custom section will take the longest time, but subsequent denormalization runs only pull in the changes since the last denormalization run.

Running the Denorm process manually for a limited number of custom sections is recommended to monitor the process initially.

## Implementation Details

*This section describes some of the technical details of the denormalization process, in order to provide some transparency and insight into the process. This level of detail is not required to run the CS Denorm process, but may be useful to better understand performance characteristics or to troubleshoot the process if any errors occur.*

The denormalization process works as follows:

### Determines which custom sections to process

The DENORM\_PROC\_MAPPING stored procedure first retrieves all entries with a status of 2 (Ready for denormalization) from the DENORM\_CS\_TABLE table, and then denormalizes each custom section table, one at a time.

For each custom section table, it does the following:

### Removes any relevant denormalized Custom Section data

If custom section data for a business object has previously been denormalized, but that custom section is then deleted from the business object, the CS Denorm process removes the relevant records from the target table.

- A log\_level of 2 will add an entry to the DENORM\_LOG table indicating the number of records removed for each custom section table.
- A debug\_level of 1, or greater, will print out the number of records removed for each custom section table. This should only be set for debugging purposes.

### Determines which business objects to process

- If *every* business object (Specs, Sourcing Approvals, etc) that has this custom section *has not been modified\** since the last time this section was denormalized, then this custom section's denormalization is skipped.
  - A log\_level of 2 will add an entry to the DENORM\_LOG table indicating that it was skipped.
- Otherwise, any business objects (which contain that custom section) that *have been modified\** after the last denormalization will be evaluated.
  - A log\_level of 1 or higher will add an entry to the DENORM\_LOG table indicating the denormalization process has started for this denormalized table.

### Populates the temporary tables

A temporary table is created for each column grouping in the denormalized table:

- A *column grouping* is a series of columns related to the Extended Attributes in a denormalized column (or denormalized row if using the Pivot denorm approach). For instance, a column grouping containing Quantitative Range extended attributes will

correspond to columns for the Target, Min, Max, and UOM values (and Base values for each, if using the Base UOM feature).

Data from the relevant business objects for the column grouping is then pulled into these temporary tables. A significant process of the denormalization process can occur at this stage.

- A `debug_level` of 2 will print out the SQL commands that generate and populate each of these temporary tables. This should only be set for debugging purposes.

### **Combines data from the temporary tables into the staging table**

Once all temporary tables have been created and populated for a custom section, any old data is deleted from the staging table. Next, data from the temporary tables is joined and copied into the denormalized staging ("`STG_<alias>`") table.

- A `debug_level` of 2 will print out the SQL commands that generate and populate each of these temporary tables. This should only be set for debugging purposes.

Processing time of this step will be based on the number of denormalized columns and the amount of data.

The temporary tables are then deleted.

### **Copies Denormalized data into the final table**

At this stage, the denormalization processing is complete, and the updates from the Staging table are ready to be copied into the final denormalization table. First, however, data is deleted from the final denormalization table for business objects that have been modified since the last run time. Then the data from the Staging table is inserted into the final denormalization table.

- A `log_level` of 1 or higher will add an entry to the `DENORM_LOG` table indicating the denormalization process has completed for this denormalized table, and includes the number of records affected.

The next custom section that is ready for denormalization is then processed.

**\*\*Modification Dates:** A modification to a business object may occur from various changes, some of which may have nothing to do with the actual custom section data. However, the CS Denorm process considers any change to a business object as one that requires the Custom Section denormalization to be regenerated.

## Monitoring Batch Status

The DENORM\_LOG table is populated with logging messages from the table generation and the batch denormalization process. This table can therefore be monitored for status.

- Each custom section being denormalized should log a “started” message, followed by a “completed” message which contains the number of records it added.
- If any errors occur, they are also logged, but the process continues (meaning the next custom section will be processed), so you will still get a “completed” message.
- Additionally, the skipped sections can optionally be logged.

When an error is encountered during de-normalization of a custom section template, no changes for that particular custom section template will be applied and a row will be written to the DENORM\_LOG table. Any errors that occur will be logged with a value of “CS Denorm ERROR” in the MODULE column, and the error details in the MSG column.

Customers that require other notification channels (e.g., email) may enable them programmatically by attaching an INSERT trigger to the DENORM\_LOG table, as needed.

## Monitoring Near-Real Time Denorm Status

The Real Time Denorm process will log basic informational messages, as well as error messages, to a rolling log file in the Denorm\_CustomSections directory within the default Logs directory. A new log file will be generated each day.

The MaxLogFilesToKeep\_CS configuration setting controls the number of daily log files that will be kept. To disable this file logging you must set the MaxLogFilesToKeep\_CS value to "0" (zero).

Errors will still be emailed and an entry in the Event Log will also be added.

NRTD processes individual denorm requests from the DENORM\_CS\_REQUEST table. The statuses of the NRTD requests are defined as follows:

| <i>Denorm Status</i> | <i>Notes</i>   |
|----------------------|--|
| -1: Error            | Errors notifications are sent via email and are logged to the event log  |
| 0: Pending           | Items in this status will get processed during the next NRTD Service execution on the Remoting Container         |
| 1: Processing        | Items are currently being processed  |
| 2: Completed         | Denormalization completed. These records will be removed on the next execution of the CS and EA Cleanup Service. |