conga

CPQ Best Practices: Governor Limits

Table of Contents

CPQ Best Practices: Governor Limits explains Salesforce governor limits and best practices for extending Conga CPQ.

Topic	Description
What's Covered	This section explains the best practices to configure and extend CPQ functionalities. The section covers error scenarios and recommendations to handle such errors.
Primary Audience	This document is intended for: • Implementation teams that customize and extend the standard functionality provided by CPQ on Salesforce • CPQ Administrators
IT Environment	Refer to the latest *Conga CPQ Release Notes* for information on System Requirements and Supported Platforms.
Other Resources	• Refer to *Conga CPQ* Release Notes for information on system requirements and supported platforms, new features and enhancements, resolved issues, and known issues for a specific release.

This section provides the following information:

- Established governor limits of Salesforce
- Best Practices to avoid governor limits
- Handling governor limits when encountered
- Guidelines for CPQ
- Recommendations to improve the performance in different areas of CPQ
- Parameters contributing to the performance of CPQ

Before using CPQ, you must be familiar with the following:

- Advanced Salesforce administration
- Salesforce and Conga terms and definitions

Select one of the following topics for more information:

- About Governor Limits
- Guardrail for Using CPQ

About Governor Limits

Salesforce governor execution limits ensure the efficient use of resources on the Force.com multi-tenant platform, which is a single resource shared by many customers and organizations. To ensure that no customer monopolizes resources, Salesforce.com has created a set of limits that governs the code execution. Whenever a governor limit is exceeded, the platform halts the execution of the program and displays an error.

Apex code is executed on Salesforce.com servers as part of atomic transactions. Apex transactions ensure the integrity of data. The Apex run-time engine strictly enforces limits to ensure that the Apex code or processes do not monopolize shared resources.

A single trigger execution is one transaction. Most of the governor limits are per-transaction based and some limits are not transaction based. For example, the number of API calls in an org or email notifications that are reset every 24 hours. Salesforce has established the following governor limits.

- Per-Transaction Apex Limits
- Per-Transaction Certified Managed Package Limits
- Lightning Platform Apex Limits
- Static Apex Limits
- Size-Specific Apex Limits
- Miscellaneous Apex Limits

For example, Per-Transaction Apex Limits count for each Apex transaction. For Batch Apex, these limits are reset for each execution of a batch of records in the execute method.

> ⓘ When CPQ features are implemented in a specific way, they can reach the Per-Transaction Apex Limits. Therefore, this section describes Per-Transaction Apex Limits. For other Salesforce governor limits, refer to https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_gov_limits.htm.

Established Salesforce Governor Limits

Salesforce governor limits are implemented in the following key areas.

Key Area	Limit
Apex CPU Timeout	Salesforce has a timeout limit for transactions based on the CPU usage. If a transaction consumes too much CPU time, Salesforce ends it as a long-running transaction. The timeout is set to a maximum of 10 to 15 seconds for each transaction.
Heap Size	The A*pex heap size too large* error occurs when too much data is being stored in memory during processing. This is the amount of memory allocated to objects defined in the Apex Code. The limit depends on the type of execution (for example, synchronous and asynchronous calls). Salesforce has set a limit of 6 MB for synchronous transactions and 12 MB for asynchronous transactions.
View State	The view state of a web page is composed of the data that is necessary to maintain the state during a server request. Salesforce has set this limit to 170 KB.
Salesforce Object Query Language (SOQL)	There is a limit to the number of records that can be retrieved or updated by a SOQL or data manipulation language (DML) query. Salesforce displays an error when this limit is exceeded. Salesforce has set the maximum number of records that can be returned in a single query to 50000.

The following table lists the limits for synchronous Apex and asynchronous Apex (Batch Apex and @future methods) when they are different. Otherwise, this table lists only one limit that applies to both synchronous and asynchronous Apex.

Description	Synchronous Limit	Asynchronous Limit
Total number of SOQL queries issued	100	200
Total number of records retrieved by SOQL queries	50000	
Total number of records retrieved by *Database.getQueryLocator*	10000	
Total number of SOSL queries issued	20	

Description	Synchronous Limit	Asynchronous Limit
Total number of records retrieved by a single SOSL query	2000	
Total number of DML statements issued	150	
Total number of records processed as a result of DML statements, *Approval.process*, or *database.emptyRecycleBin*	10000	
Total stack depth for any Apex invocation that recursively initiates triggers due to insert, update, or delete statements	16	
Total number of callouts (HTTP requests or web services calls) in a transaction	100	
Maximum cumulative timeout for all callouts (HTTP requests or web services calls) in a transaction	120 seconds	
Maximum number of methods with the @future annotation allowed for each Apex invocation	50	0 in batch and future contexts; 1 in queueable context
Maximum number of Apex jobs added to the queue with *System.enqueueJob*	50	1
Total number of send email methods allowed	10	
Total heap size	6 MB	12 MB
Maximum CPU time on the Salesforce servers	10000 milliseconds	60000 milliseconds
Maximum execution time for each Apex transaction	10 minutes	
Maximum number of push notification method calls allowed for each Apex transaction	10	

Description	Synchronous Limit	Asynchronous Limit
Maximum number of push notifications that can be sent in each push notification method call	2000	

For more information on Execution Governors and Limits, refer to https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_gov_limits.htm.

For more information on Salesforce Developer Limits and Allocations Quick Reference, refer to https://resources.docs.salesforce.com/220/latest/en-us/sfdc/pdf/salesforce_app_limits_cheatsheet.pdf.

Best Practices to Avoid Governor Limits

As a Salesforce Code Developer, you must follow some best practices to ensure that your code is scalable and does not exceed the governor limit. The following are such best practices:

- Bulkify your code.
- Avoid SOQL queries or DML statements inside For loops.
- Bulkify your helper methods.
- Use collections, streamlining queries, and efficient For loops.
- Streamline multiple triggers on the same object.
- Query large data sets.
- Use the limits of Apex methods to avoid exceeding governor limits.
- Use @future appropriately.
- Use batch Apex if you are working for more than 50000 records.

For more information, refer to Apex best Practices.

Best Practices for Customization

While implementing the following customization, you might encounter problems.

- **Roll up Summary Fields:** When you define Roll up summary fields on the Quotation object and clone a quote with line items, you may encounter issues like Apex CPU Time Out, because the calculation happens each time a new line item is created. Avoid defining Roll up summary fields if possible.

- **Formula Fields:** Because formula fields are calculated at runtime, they impact performance. For example, if you have 50 formula fields on a line item and there are 100 line items in the cart, 5000 formulas are calculated in a single SOQL. Use numeric expressions instead.
- **SOQL:** Any Query that takes 200 milliseconds or more is considered long-running. Optimize the query by requesting relevant fields and indexing fields in the *where* clause. Enable debug, keep database logs, keep the value of **Apex Profiling** to *Finish* in Debug Level Details, and turn off everything in log filter.

The following table lists the CPQ objects and the customization you can perform on those objects.

CPQ Objects	Customization
• Proposal • Proposal Line Item • Product Configuration • Line Item • Product Attribute Value • Quote • Quote Line Item • Opportunity	• Custom Fields • Formula Fields • Roll up Summary Fields • Workflows • Process Builder • Validation rules • Triggers • SOQL • Callbacks

Follow the recommendations when you customize CPQ objects:

- Execute triggers on appropriate events, not on all events. For example, execute triggers before insert, after an update, and so on.
- Avoid defining a trigger on Line Item objects.
- Invoke any Apex record trigger, class, or extension for more than 200 records.
- Review any DML statements in a trigger, because these statements can trigger another set of triggers.
- Avoid using rollup summary fields if you have a trigger in place on the same object. Perform the required calculation in the trigger. If it cannot be avoided, the recommended maximum limit is 20.
- Avoid using more than 10 formula fields on one object, such as Line Item object and Proposal Line Item object, because it impacts CPQ performance. If you are creating a formula field on a line item only to access it in a Pricing Callback (PCB), you can use CPQ numeric expressions, so that you can avoid creating metadata fields. For more information, see CPQ for Administrators.
- Merge any workflow on a line item object into Pricing Callback if you have one.

- Ensure that the Product Attribute Value (PAV) fields are in sync on all PAV objects such as Proposal PAV, Asset PAV, Agreement PAV, and Order PAV.
- Consider creating skinny tables because they speed up the execution of SOQL queries.
- Avoid calling managed methods or APIs in a loop.
- Limit customization to the objects mentioned.
- Do not interpret the functional purpose of any standard field based on the value stored. When in doubt, file a support case for clarification.
- Creating triggers on common objects such as task, attachment, and so on should have checks to bypass in case of parent objects not relevant to that function. Check for this in preexisting code written before Conga is implemented.
- Only use Option Filter Callback when needed. Measure code and optimize custom code performance.
- Avoid using Process Builder and workflows on CPQ objects.
- Avoid the combination of Process Builder, workflows, and trigger within the same object. Instead, move the logic from Process builder and workflows to triggers. You must not define more than five triggers on one object.

Handling Governor Limits

This section describes some of the key exceptions observed in CPQ and describes how to deal with governor limits if the code encounters them after configuring CPQ. In general, you need to analyze logs and follow the best coding practices to resolve the errors. For more information, refer to Salesforce App Limits Cheatsheet.

Exception 1: System.LimitException: Apttus_Config2: Too many DML rows: 10001

This exception occurs when you try to perform DML operations on more than 10000 records at a time.

DML is different than SOQL. DML is the process typically used to insert, update, upsert, or delete records; SOQL is the query language used to return rows. Per the governor limits, the maximum number of records retrieved by SOQL queries is 50000. The maximum number of records processed as a result of DML statements, *Approval.process*, or *database.emptyRecycleBin* is 10000.

Sample Business Use Cases

- The Hierarchy View update job displays the *Too many queries* error, which has impacted the price book and display of products (products are not displayed under the correct categories on the catalog).
- When adding items to the option group *Destination Sets* in CPQ Admin Console in Production, an error is received for the Bundle LOGIQ S8 R3 (ULS_PCL_GI_0025).

Best Practices

- A separate class implementing the *Database.Batchable* interface allows CPQ to handle DML in batches of records.
- You can use <apex:actionPoller> in a Visualforce page.
- If the trigger has DML functionalities for many rows (more than 10000), it must be in Batch Apex.

Exception 2: Visualforce Remoting Exception: Apex heap size too large

Salesforce enforces an Apex Heap Size Limit of 6 MB for synchronous transactions and 12 MB for asynchronous transactions.

The *Apex heap size too large* exception occurs when too much data is being stored in memory during processing. The limit depends on the type of execution (for example, synchronous or asynchronous calls).

Sample Business Use Cases

When the constraint rule maintenance batch job is executed from the Maintenance tab, it runs erroneously. Especially for constraint rules with *Product Field Set* in **Product Scope** in constrain rule condition and criteria, a lot of memory is consumed to create Product Constraint View records. Eventually the batch crosses Salesforce governor limit of 12 MB memory usage per asynchronous transaction and skips creating views for some rules. The following exception occurs: *Apex heap size too large: 13808228*.

The Apex governor limits for heap size are as follows:

Total Heap Size	
Synchronous Limit	6 MB
Asynchronous Limit	12 MB
Email Services	32 MB

Best Practices

- Do not use class-level variables to store a large amount of data.
- Utilize SOQL For loops to iterate and process data from large queries.
- Construct methods and loops that allow variables to go out of scope as soon as they are no longer needed.
- Reduce the limits to the SOQLs and use transient keyword.

Exception 3: System.LimitException: Apex CPU time limit exceeded

Salesforce has a timeout limit for transactions based on CPU usage. If transactions consume too much CPU time, Salesforce terminates the execution of long-running transactions. The maximum CPU time on the Salesforce servers is 10000 milliseconds for synchronous transactions or 60000 milliseconds for asynchronous transactions.

Sample Business Use Cases

- Users could not see products in the cart after executing the Category Maintenance job.
- Users cannot execute the Update Product Constraints View and they receive an error - *Apex CPU Time Limit Exceeded*.
- Users encountered a "system.LimitException" error when they had more than 100000 products, defined a constraint rule with 98 condition products, and configured product scope as FieldSet. If there are a large number of fields defined in that Product Field Set, user encountered the error.

Best Practices

- Remove unnecessary code and loops.
- Use collections (Set, Hashmap, or List) efficiently.
- Avoid using a loop within another loop.
- SOQL query should be indexable.
- Avoid unnecessary re-initialization of variables.
- Use Aggregated SOQL (database operations are not counted in this limit, so avoid arithmetic operations in Apex).
- Check how much time workflow rules and process builders take.

In CPQ

- Define the following settings before running the Category Maintenance job.
  APTS_UpdateViewUseDmlLimit = true
  APTS_UpdateViewProductBatchSize = 50
- Optimize the constraint rule of *ProductScope = Product Field Set* to a lower number (this always breaks because of Salesforce limit).
- Avoid using many Custom Formula fields.
- To avoid "system.LimitException" error, create multiple constraint rules to reduce the number of condition products in a single constraint rule. Also, lower the number of fields in the Product Field Set.

Exception 4: System.LimitException: Too many query rows: 50001

There is a limit to the number of records that that can be retrieved by the SOQL query, which is 50000 records. The 50000 limit is an overall per-transaction limit and not a per-query limit. If you attempt to query the more than 50000 records, you will get an exception.

Sample Business Use Cases

- When running Category Maintenance, the following error occurs: *Apttus_Config2:Too many query rows: 50001*
- When trying to add more than 100 products from Catalog, CPQ displays a message: *Apttus_Config2:Too many query rows: 50001*

- When there are more than 400 lines added to the quote, the performance of CPQ is impacted.

Best Practices

- Use Batch Apex, in which the 50000 limit counts for each batch execution.
- These limits count for each Apex transaction. For Batch Apex, these limits are reset for each execution of a batch of records in the execute method.
- Limit the results by adding more criteria or using a LIMIT 50000 statement in SOQL.

In CPQ

- All constraint conditions or actions must have a narrow search.
- Avoid using *Product field set*.

Exception 5: System.LimitException: Too many SOQL queries: 101

This exception occurs when you exceed the SOQL queries governor limit. You can run up to a total of 100 SOQL queries in a single call or context.

Sample Business Use Cases

- If you finalize a cart that has 50 to 60 ramp lines, the cart stops responding.
- Each product added to the cart has a constraint rule. When you add 90 line items to the cart, constraint rules fail to work.

Best Practices

- Change the code by following the Apex Code best practices so that the number of SOQL queries triggered is less than 100.
- Change the context; use the @future annotation, which runs the code asynchronously.
- Ensure that you do not have a recursive loop calling a SOQL.
- Follow Apex Code key principals while writing triggers and bulk requests.
- Do not perform any DML or CRUD operation inside a For loop.

In CPQ

- Enable the **Run Misc Finalization Task in Async Mode** setting in Config System Properties (Developer Setting).

Guardrail for Using CPQ

CPQ is a complex application with configurability and extension capability. There are multiple objects interconnected that provide this capability. The relationship between objects is used to perform the compute-intensive configuration and pricing. This permutation-combination of various objects, their relationship, and data in each object contribute to the overall computation. It is difficult to arrive at a limit for each object that can be marked as a guardrail.

This section provides information on product guardrail parameters that contribute to CPQ performance.

Guidelines for CPQ

This section describes guidelines for CPQ configuration. CPQ performance is impacted when the guidelines are exceeded.

> These are general guidelines recommended by Conga. You must test in your org to determine an optimal limit for your implementation.

Description	Guideline
Maintenance Batch Job	
Number of **Product Hierarchy View** records for a single category, to execute Category Maintenance Batch job	30000
Cart	

Description	Guideline
Maximum number of line items to be selected for mass actions on the cart (using the **Max Allowed Lines For Mass Actions** setting) ⓘ The default value for selecting the line items for mass action is 10. When you enable the **Perform Mass Actions in Parallel** custom setting for selecting a large number of line items on the cart for mass action, the **Max Allowed Lines For Mass Actions** custom setting is overridden.	10
Number of columns in the Cart page	10
Rules	
Number of actions in constraint rules	10
Number of rules associated with products in any leaf category	1000
Number of conditions in each constraint rule	10
Number of rules for each category	1000
Number of rules for each bundle or options	10
Number of assets and service bundle for each constraint rule	1000
Category	
Number of levels in a category hierarchy	3
Number of subcategories	10
Product Groups	

Description	Guideline
Product Group Members for each Price List	10000
Number of products for each product group	200
Number of Product Groups a product can be associated with	200
Option Groups/Bundle Structure	
Number of options for each option group	1000
Nested bundle structure ⓘ You can have a bundle > subbundle > option.	3
Attributes	
Number of attributes ⓘ To overcome the 800 field limit of Salesforce on the Product Attribute Value (PAV) object, CPQ provides three PAV extension objects. You can create 800 attributes in each of these out-of-the-box PAV extension objects. CPQ does not support custom PAV extension objects from the CPQ September '22 release.	800
Number of attribute value matrix entries	10000
Pricing	
Number of price dimensions	4
Number of matrix entries for each price list item	1000

Description	Guideline
Number of price rules entries	500
Number of Price List setups in an org ⓘ When customer-specific pricing is required, use contractual price list instead of standard price list.	100
Approvals	
Number of approval steps ⓘ In Salesforce Admin setup, you can configure 100 approval steps (with a combination of standard steps and approval rules). In Salesforce runtime, you can submit, approve, or reject 350 approval requests simultaneously.	100

Recommendations to Improve CPQ Performance

This section describes the recommendations to improve the performance in different areas of CPQ.

Catalog Page

- If there is only one category, enable the **Hide top level category** setting to display subcategories as categories.
- Remove subbundles from parent bundles and treat subbundles as separate products.
- Use product structure (such as bundles), not rules (such as constraint, validation) wherever possible.
  For example, there is a requirement to include Product B if the user selects Product A. If you model A and B as standard items and a constraint rule to include B for A, this is

not an optimal way of modeling. Instead, create bundles and auto-include Product B when the user selects bundle Product A.

- Minimize the number of options
    - Remove inactive options (end of life products) from bundles periodically.
    - Add products (options) to the price list in the region where they are sold.
    - Remove non-priced options from the bundle structure. Use attributes instead.
    - Avoid creating options for non-tangible products. Use attributes instead. For example, a country should be an attribute and not an option.
    - Set frequently used options as default.
- Avoid SKU proliferation (creating multiple instances of the options to handle quantity-based pricing).
- Leverage the **Price included bundle** option wherever applicable.
- Avoid creating duplicate data to maintain quality.
- Standardize naming conventions for rules and attributes.
- Use multiple option groups instead of maintaining all options under one option group.
- Create and manage *Public* favorite configurations effectively on the Catalog page. Use *Private* favorite configurations instead.
- Reuse and share configurations through favorites.

Cart Page

- Filtered lookup columns on the cart may impact CPQ performance.
- Manage the pricing callback code efficiency.
- Avoid defining multiple deal guidance setup for a single criteria.
- Avoid customizing a large cart.
- Verify if Option Groups with Max Cardinality = 1 can be used as picklist values.

Rules

- Keep the use of Product Visibility Rules to minimal to optimize catalog load performance. Create a separate price list or constraint rule.
- Create option specific rules. You can configure rules for common options without being specific to bundles.
- Define the criteria in a constraint rule condition wherever applicable, instead of constraint rule action criteria.
- Reduce the number of constraint rules by merging the constraint actions with similar conditions.
- Write your criteria on one condition. Do not repeat criteria on other conditions.

- Use client-side constraint rules to get better CPQ performance, that is, avoid server round-trip.
- Avoid using Formula fields in Constraint Rule Action criteria especially when you are using client-side constraint rules. Instead, use Conga numeric expressions.
- Avoid using formula fields on constraint rule conditions
- Avoid nested rule conditions.
- Avoid rules where the scope includes both bundles and options.
- Avoid creating redundant rules.
- Avoid creating multiple conditions with the same product when using AND association. The following table illustrates an example.

Constraint Rule Field	Value
Condition 1	*Product A*
Condition 2	*Product A with Attribute A*
Association	*1 AND 2*

  In the above-mentioned example, the condition is only fulfilled when both instances of the product are added to the cart.
- Do not create a server-side constraint rule with constraint condition using a formula field attribute from Product Attribute Rule Extension object.

Installed Product Page

- The number of renewals per trip is 20 by default. To avoid CPQ time out, reduce the number in the **Max Renews Per Trip** setting in Installed Products Settings.

Pricing on Cart

- Use Conga expressions wherever possible.
- Price Batch Size
  - The Price Batch Size must be high (higher the value faster the performance).
  - Define Price Batch Size correctly.
  - Measure and evaluate the highest possible value. Do not set up a value that can result in governance limits.
- Enable defer pricing only if the user experience is impacted (performance latency). Enabling defer pricing can increase the latency in the step when pricing is executed.

- Use the **Compute Totals In Separate Step** setting to avoid the CPU time limit issue to a large extent. This setting indicates whether the totaling should be performed in a separate step or if it should be combined with the base pricing step. This setting ensures that the totaling, which is dependent on the number of lines, is done as a separate remoting call.
- Manage deal guidance rules.
    - Consolidate rules wherever possible.
    - Simplify cross object fields.
    - Deal guidance limit varies based on the pricing complexity, number of lines, and deal guidance complexity.
- Manage the pricing callback code efficiency. For best practices for defining pricing callback, see Pricing Callback Classes.

Recommendations to Import Legacy Products into Conga CPQ

This topic describes best practices for migrating purchased product information from legacy CRM and ERP systems into Conga CPQ and for tracking subsequent renewals.

CPQ Implementation without Billing

To import assets in a pure CPQ implementation without Billing, it is best to create the asset line-item data directly in the Conga environment. This section describes best practices for importing legacy data, the asset types that may be imported, and those asset types' characteristics.

Maintaining Uniqueness within the Imported Data

- When importing data into the Asset Line Item object, the combination of line number (or primary line number) and business object ID must be unique. Business object ID can be quote ID, order ID, account ID, or agreement ID (if applicable). If the combination is not unique, the Installed Products page does not complete loading data.
- Maintaining line number uniqueness can be challenging when the process is batched for huge volumes. Implementation teams must ensure that the line number is sequenced programmatically to maintain uniqueness across multiple batches.

Renewing Assets

When performing asset renewal, ensure that bundles and their options have matching charging frequencies (one-time or recurring) if they have the same Charge Type. For example, When the bundle's Charge Type is Standard price and its option's charge type is Standard Price, then the Frequency of the bundle and its option should be the same (either one-time or recurring).

A sample when the bundle and its option have the same charge type (standard price):

	Case 1: Bundle | Option	**Case 2: Bundle | Option**	**Case 3: Bundle | Option**
Frequency	Onetime | Onetime	Recurring | Recurring	Onetime | Recurring
Charge Type	Standard Price | Standard Price	Standard Price | Standard price	Standard Price | Standard price
Recommended ?	Yes	Yes	No

Standalone Assets

A standalone asset line item needs the following basic data elements to be transferred successfully to Conga CPQ. Implementation teams must ensure that this data is available within the source of the purchased products.

- The Sold To account associated with the standalone asset governs its display on the Installed Products page.
- The unit sale price of the asset with its corresponding selling frequency becomes the asset's base price for any future asset-based orders.
- For a recurring entity, the asset line item's term is important for prorating future asset-based orders and for determining the renewal start dates for manual and automated renewals.
- Set the **Inactive** flag on the asset line item to false. Any asset with Inactive=True is not visible on the Installed Products page.
- Set **Has Attributes** to True if the asset has attributes.

- Set Asset Status to Activated. The Installed Products page does not recognize statuses such transactional statuses as Amended or Renewed. An asset's status is either Activated or Cancelled.

Bundled Assets

In addition to the points specified for standalone assets, you must specify a bundle's relationships to its child assets (options). Your legacy systems may not operate with the concept of a bundled asset. Entities are related either using a contract or an order. Implementation teams must ensure that all the data is available and structured appropriately for the bundled asset to be used with asset-based ordering.

- Set **Has Options** to True on the asset.
- Set **Has Attributes** to True if the asset has attributes.
- Set **Has Attributes** to True at the bundle level even if the bundle has no attributes: options within the bundle may have attributes that drive quantity or cumulative range pricing. If **Has Attributes** is set to True, the reprice action does not reflect the change.
- Populate the parent bundle number (representing the primary line number of the immediate parent line in the hierarchy).
- Populate the parent asset ID (representing the primary line number of the immediate parent asset). This makes a difference when user has cloned options within the bundle structure.

Examples

The following examples show the differences in the population of the parent bundle ID and parent asset ID.

A simple bundle with one option

Asset Name	Charge Type	Selling Term	Start Date	End Date	Is Primary Line	Primary Line Number	Bundle Asset Line Item	Parent Bundle Number	Parent Asset Line Item
Supernet Transportation Service	Subscription Fee	12	1/1/2021	12/31/2021	Yes	1			

Asset Name	Charge Type	Selling Term	Start Date	End Date	Is Primary Line	Primary Line Number	Bundle Asset Line Item	Parent Bundle Number	Parent Asset Line Item
AAP Options	Subscription Fee	12	1/1/2021	12/31/2021	Yes	2	Supernet Transportation Service	1	Supernet Transportation Service

A bundle with two options where one of the options is a bundle

Asset Name	Charge Type	Selling Term	Start Date	End Date	Is Primary Line	Primary Line Number	Bundle Asset Line Item	Parent Bundle Number	Parent Asset Line Item
Supernet Transportation Service	Subscription Fee	12	1/1/2021	12/31/2021	Yes	4			
AAP Options	Subscription Fee	12	1/1/2021	12/31/2021	Yes	7	Supernet Transportation Service	4	Supernet Transportation Service
VPN	Subscription Fee	12	1/1/2021	12/31/2021	Yes	5	Supernet Transportation Service	4	Supernet Transportation Service
Level 2 VPN	Subscription Fee	12	1/1/2021	12/31/2021	Yes	6	Supernet Transportation Service	5	VPN

A bundle with a nested bundled option that is cloned

Asset Name	Charge Type	Selling Term	Start Date	End Date	Is Primary Line	Primary Line Number	Bundle Asset Line Item	Parent Bundle Number	Parent Asset Line Item
Supernet Transportation Service	Subscription Fee	12	1/1/2021	12/31/2021	Yes	1			
AAP Options	Subscription Fee	12	1/1/2021	12/31/2021	Yes	2	Supernet Transportation Service	1	Supernet Transportation Service
VPN	Subscription Fee	12	1/1/2021	12/31/2021	Yes	3	Supernet Transportation Service	1	Supernet Transportation Service
Level 2 VPN	Subscription Fee	12	1/1/2021	12/31/2021	Yes	4	Supernet Transportation Service	3	VPN
Level 2 VPN	Subscription Fee	12	1/1/2021	12/31/2021	Yes	5	Supernet Transportation Service	5	VPN
VPN	Subscription Fee	12	1/1/2021	12/31/2021	Yes	6	Supernet Transportation Service	1	Supernet Transportation Service

Ramped Assets

To persist discrete data points associated with each period for billing and revenue recognition in a Quote-to-Cash (Q2C) environment, CPQ creates one asset for each period while processing a ramped (multi-year) deal. The following examples show the structure of such assets:

- Ramped assets are all connected using a primary ramp asset, acting as a loosely-coupled bundle that holds the assets for all periods.
- CPQ populates the bundle asset line item.
- You must set the **Is Primary Ramp Line** flag to True for the first period.

Examples

The following examples show the differences in the population of the parent bundle ID and parent asset ID.

A standalone ramped asset

Asset Name	Charge Type	Start Date	End Date	Is Primary Line	Is Primary Ramp Line	Primary Line Number	Bundle Asset Line Item	Parent Bundle Number	Parent Asset Line Item
CPQ	Subscription Fee	1/1/2017	12/31/2017	Yes	Yes	1			
CPQ	Subscription Fee	1/1/2018	12/31/2018			1	CPQ		
CPQ	Subscription Fee	1/1/2019	12/31/2019			1	CPQ		

A ramped asset with bundle level ramps that cascade to the options

Asset Name	Charge Type	Start Date	End Date	Is Primary Line	Is Primary Ramp Line	Primary Line Number	Bundle Asset Line Item	Parent Bundle Number	Parent Asset Line Item
Conga	Subscription Fee	1/1/2017	12/31/2017	Yes	Yes	1			

Asset Name	Charge Type	Start Date	End Date	Is Primary Line	Is Primary Ramp Line	Primary Line Number	Bundle Asset Line Item	Parent Bundle Number	Parent Asset Line Item
Conga	Subscription Fee	1/1/2018	12/31/2018			1	Conga		
Conga	Subscription Fee	1/1/2019	12/31/2019			1	Conga		
CPQ	Subscription Fee	1/1/2017	12/31/2017	Yes	Yes	2	Conga	1	Conga
CPQ	Subscription Fee	1/1/2018	12/31/2018			2	Conga	1	Conga
CPQ	Subscription Fee	1/1/2019	12/31/2019			2	Conga	1	Conga
ABO	Subscription Fee	1/1/2017	12/31/2017	Yes	Yes	3	Conga	1	Conga
ABO	Subscription Fee	1/1/2018	12/31/2018			3	Conga	1	Conga
ABO	Subscription Fee	1/1/2019	12/31/2019			3	Conga	1	Conga
Billing	Subscription Fee	1/1/2017	12/31/2017	Yes	Yes	4	Conga	1	Conga

Asset Name	Charge Type	Start Date	End Date	Is Primary Line	Is Primary Ramp Line	Primary Line Number	Bundle Asset Line Item	Parent Bundle Number	Parent Asset Line Item
Billing	Subscription Fee	1/1/2018	12/31/2018			4	Conga	1	Conga
Billing	Subscription Fee	1/1/2019	12/31/2019			4	Conga	1	Conga

A ramped asset with option-level ramps

Asset Name	Charge Type	Start Date	End Date	Is Primary Line	Is Primary Ramp Line	Primary Line Number	Bundle Asset Line Item	Parent Bundle Number	Parent Asset Line Item
Conga-Service Included	Standard Price	1/1/2017	12/31/2017	Yes	Yes	1			
Conga-Service Included	Subscription Fee	1/1/2017	12/31/2019			1	Conga-Service Included		
CPQ	Subscription Fee	1/1/2017	12/31/2017	Yes	Yes	2	Conga-Service Included	1	Conga-Service Included
CPQ	Subscription Fee	1/1/2018	12/31/2018			2	Conga-Service Included	1	Conga-Service Included

Asset Name	Charge Type	Start Date	End Date	Is Primary Line	Is Primary Ramp Line	Primary Line Number	Bundle Asset Line Item	Parent Bundle Number	Parent Asset Line Item
CPQ	Subscription Fee	1/1/2019	12/31/2019			2	Conga-Service Included	1	Conga-Service Included
ABO	Subscription Fee	1/1/2017	12/31/2017	Yes	Yes	3	Conga-Service Included	1	Conga-Service Included
ABO	Subscription Fee	1/1/2018	12/31/2018			3	Conga-Service Included	1	Conga-Service Included
ABO	Subscription Fee	1/1/2019	12/31/2019			3	Conga-Service Included	1	Conga-Service Included

Linkage to Child Objects

You can link an imported asset line item to its attribute detail using the asset attribute value object. You can link it to its usage detail using the asset usage tier values if applicable.

Quote-to-Cash Implementation with Standard Billing

For a Q2C implementation where there is a need to generate billing for assets, the recommended strategy is to create the order and order lines. Activation of the order and order lines will then generate the assets and the billing. The following are the best practices in those scenarios:

- Specify the billing frequency and billing method for the price list item associated with the order line item. If you do not specify, order activation will fail.
- Set the status of the order header and order lines to Pending Activation.
- Set the Line Status of the order line items to New.

Quote-to-Cash Implementation with Custom Billing

For a Q2C implementation to generate custom billing (using a billing plan) for assets, create the proposal and proposal line items, attach a billing plan to the proposal header, accept the quote, and activate the corresponding order. In such a scenario:

- Follow the standard process to create or import a proposal.
- Auto-activate orders by passing either the **Ready for Activation Date** from the proposal (if the billing is aligned to this date) or by setting the **Auto Activate Order** flag to True.
- Do not create the renewal suite on order activation while going through this process.

Renewal Pipeline Generation for Imported Assets

To generate the renewal pipeline for legacy assets, use the OnDemand mode in conjunction with the lead time.

- To group the generated renewal quotes by quote ID, use the *Apttus_QPConfig Proposalld r.Name* grouping field.
- Do not generate renewal suites for legacy assets on order activation in a batch process. This overwhelms CPQ resources, violates the governor limits, and causes unpredictable behavior.

Using a Batch Process to Create Imported Assets

- Asset import through an order involves calling several data manipulation languages (DMLs) and APIs. Executing too many orders in one batch can result in a *Too many SOQL queries: 201 error*.
- You must ensure that the batch size is optimized to prevent errors based on the size of your order and the size of each line item.

Factors of Master Data that Contribute to Performance

The CPQ Master data model represents the setup and administration of configurations, rules, and pricing of the system. This topic lists the factors of master data that contribute to the performance of CPQ. A combination of these factors, when implemented in a specific way, can impact CPQ performance.

- Number of Products
- Number of Price Lists
- Number of Price List Item
- Number of Price List Item per Price List
- Related Price Lists
- Charge Types per Price List
- Number of Price Matrices
- Number of Price Matrices per
- Price List Item
- Number of dimensions per Price Matrix
- Number of Price Rules
- Rule sets
- Rule entries
- Number of Catalog levels (Hierarchy)
- Number of Items in a catalog
- Number of Options in a Option Group
- Number of Option Groups in a bundle
- Number of Options in a Bundle
- Number of level of bundle or subbundle
- Number of Attributes
- Number of Pricing Attributes
- Product Attribute Rules
- Number of Numeric Expressions (for each product)
- Number of Constraint Rules
- Number of Constraint Rule Conditions per
- Constraint Rule
- Number of Constraint Rule Actions per Constraint Rule
- Deal Guidance
- Deal Guidance Entries
- Attribute Value Matrix
- Number of CPQ Search Filters

- Line Level Approvals

Factors of Transactional Data that Contribute to Performance

The CPQ Transactional data model represents the configuration, rules, and pricing that is applied for each transaction and how that impacts the performance at runtime. The data setup of the master data drives the behavior for each transaction. This topic lists the factors of transactional data that contribute to the performance of CPQ. A combination of these factors, when implemented in a specific way, can impact CPQ performance.

- Number of Product Configuration per Quote
- Number of line items per Product Configuration
- Number of charge types applied per Product
- Number of Attributes per bundle/Option
- Attribute Rules per Bundle/Option
- Number of constraint rules applied per cart
- Number of Product Groups one Option/Bundle is associated with
- Number of columns in a cart
- Number of Approval Processes (line Level)
- Number of Summary Groups in cart
- Related Price Lists
- Promotions related to line items
- Deal guidance
- Adjustment Line Item bucketing