A Guidebook for Anaplan model building

v2019.01

community.anaplan.com/planual

Introduction

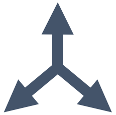
Welcome to The Planual, the definitive guidebook for Anaplan modelling best practices. Over the last year we have consulted with employees, customers, partners, and Master Anaplanners to collect, evaluate, and define the techniques and structures that produce the most optimal models in the Anaplan platform. We also bring our many years of modelling experience in Anaplan and other similar products. In what follows, we are happy to provide the first ever version of “the rules.”

In building these standards we considered three main aspects:

Performance

Usability

Sustainability



The standards (rules) contained in the Planual balance these three aspects to ensure the best outcome across the whole Anaplan user base—model builders and end users alike. We have tried to make the rules as general as possible. The Planual is not about formula syntax per se, although there are elements for formula structure included when applicable.

Importantly, these rules are a guide, and there will inevitably be occasions that justify deviating from them. To accommodate these, we have included some of the known exceptions to the rules. However, we recommend that you follow the standards as much as possible (they’re proven to work!) and carefully consider the consequences when you move away from them.

As with the Anaplan platform itself, this will be an evolving document. We will add to it and amend the rules as appropriate as new functionality gets delivered. Feedback is a gift and a core value at Anaplan. We want your feedback to make improvements and corrections; these will help make the Planual the best it can be.

Finally, I would like to thank everyone who has helped in putting this together; without the creativity and collaboration of all who contributed, this would not have been possible.

I hope you find the Planual useful, learn from it, and ultimately build the best Anaplan models for you and your users.

Yours in modelling,

[illegible]

David Smith: Vice President – Operational Excellence Group **June 2019**

The Zen of Anaplan

Ten guidelines

- Just because you CAN doesn’t mean you SHOULD
- Calculate once, reference many times
- Simple > complex
- Only calculate when you need to
- Sparsity is not your enemy
- Big and fast > small and slow
- Booleans = great. Text = bad
- Text joins = worse
- Don’t let exceptions overcomplicate the model
- Sums and lookups are good, but never together

PLANS

PLANS is set of rules that aim to help structure and design Anaplan models. Together, they clarify good model design for all users—individual Anaplanners, teams, and anyone handing models over to others.

PLANS stands for:

Performance	To optimize the Anaplan Engine (Hyperblock) certain formulas and structures work better than others. One of the advantages of following the standards, is that performance improvements are a by-product of the other elements. While there will occasionally be scenarios that call for "breaking the rules," we can be aware of the consequences and use them carefully.
Logical	Build models and formulae to be logical. Complex formula calculations are more efficient when are broken up. Create modules to have a sole purpose, following a logical structure of Inputs, Calculations, and Outputs. These should be supported by Data and System Modules.
Auditable	Use notes and comments to help describe the calculations. Break up formulae for better understanding, performance and maintainability
Necessary	Don't duplicate expressions, store and calculate once, reference many times, use appropriate dimensionality, don't calculate when you don't need to.
Sustainable	Build with the future in mind and think about process cycles and updates. Make the model flexible to allow for easy changes to be made when required. Avoid hard coding (e.g. SELECT and direct references to list items).

Planual Conventions

The Planual is organised by Articles, Chapters, Rules and associated Exceptions.

Rules are numbered as part of the articles and chapter:

1.01-01 Never use SELECT with Time

Article Number = 1 (Library)

Chapter Number = 01 (Time)

Rule Number = 01 (Never use SELECT with Time)

Cross references are shown under the rule description:

2.02-14 Avoid using SELECT

Also included are references to Community articles which cover the rule content in more detail. Entering the title in Community search should take you to the article in question:

Time Range Application

Exceptions to the rules are shown under each rule (as applicable) suffixed by a letter and shown in italics:

1.01-01a Generic Time periods

1.01 Time

1.01-01	**Never use SELECT with Time**
	This goes against the Sustainable element of PLANS and the hard coding can cause issues when updating the timescales of the model. Modules with Time formatted items to be used as sums and lookup are the better option
	2.02-14 Avoid using SELECT
1.01-01a Generic Time periods	***It is OK to use SELECT with Generic Time periods such as Actual Period, Current Period, YTD, YTG, ALL Periods***

1 Central Library

This section describes the rules for the Anaplan Central Library. The Central Library is where we hold and create the structures or **dimensions** of the model.

1.01 Time

1.01-01	**Never use SELECT with Time**
	This goes against the Sustainable element of PLANS and the hard coding can cause issues when updating the timescales of the model. Modules with Time formatted items to be used as sums and lookup are the better option
	2.02-14 Avoid using SELECT
1.01-01a Generic Time periods	***It is OK to use SELECT with Generic Time periods such as Actual Period, Current Period, YTD, YTG, ALL Periods***
1.01-02	**Use the Model Calendar by default**
	As the Model Calendar is dynamic, it should be used for the majority of modules. Choose suitable time settings that cover most of the requirements
	Time Range Application

1.01-03 Use Current Period

Utilising the Current Period within the Time Settings allows the use of CURRENTPERIODSTART(), CURRENTPERIODEND() and can be used to create lookup modules to hold the current period automatically

1.01-03a Daily Timescales	***The lowest granularity for Time Settings is weekly, so if the Current Period needs to reflect a daily timescale, the Current Period setting cannot be used***
1.01-03b Non-Admin updates	***Should you need non- Workspace Administrators to update the setting, a formatted line item can be used instead***

1.01-04 Consider All Periods

Consider the use of All Periods. This is effectively the Top Level for time and whilst increasing the model cell count very slightly it does allow for flexibility in modelling

1.01-05 Exclude subtotals by default

Turn the "include" settings off by default and only include these if absolutely necessary. You can set different "includes" settings for different Time Ranges

1.01-06 Time Range Naming

Keep the naming short using the FYxx-FYyy format. This allows the user to see the time range in the module blueprint without referring back to the Time Range itself

Time Range Application

1.01-07 Time Ranges

Use Time Ranges to optimise the modules where the default model calendar is not appropriate. Consider the dimensionality for the data and set up the Time Range accordingly

Time Range Application

1.01-08 Daily Timescales on large timescales

Review the need for daily granularity for long timescales. Use time ranges to restrict the daily calendars and use PREVIOUS rather than CUMULATE

2.02-10 Using PREVIOUS vs CUMULATE

Time Range Application

1.02 Versions

1.02-01 Use Current Version setting

Utilising the Current Version (and Actual Version) within the Version Settings allows the use of CURRENTVERSION() and ACTUALVERSION() for use in calculations and SELECT statements and ISCURRENTVERSION and ISACTUALVERSION() for Boolean checks

2.02-14 Avoid using SELECT

1.02-02	**Do not use formulas in the version settings** Whilst this is a simple construct and it might be OK for simple models; it effectively adds size to models every time versions are used in a module even if you don't want the variance. Also, in modules with 3 or more dimensions if you have calculation line items and aggregations the results may not always be accurate; you cannot amend the summary method for a version formula within the versions setting
1.02-03	**Edit To / Edit From** Use these as a simple, administrator lead control for read and write access to versions. For more granular control use Dynamic Cell Access
1.02-04	**Always move switchover dates forward** Using switchover means the historic periods are effectively cleared out, so switchover should be moved forward only. Moving it backwards will leave zero in the respective period removing any values that were previously held

1.03 Users and Roles

1.03-01	**Only give write access to lists when needed** Unless the end users need to edit the list (add, delete members), access does not need to be set. Adding roles to lists increases the memory usage, so only use when necessary. A user can edit data in a module without needing to have access to the list

1.03-02 Performance and size can be dependent on the Users list

When Users list is heavily used, the number of users added can have a significant impact, be mindful of this and remove model access from any Workspace users when not required

1.03-03 Set a landing dashboard by role

Different roles should require a different entry point. Avoid a generic landing page if possible and create a specific dashboard tailored for each role

1.04 Contents

1.04-01 Keep modules hidden for end users’ roles

Users should only be entering data through dashboards, so remove Modules from the contents panel

1.04-02 Utilize Show Content On/Off for Different Functional Areas

To reduce maintenance, create separate Functional Areas for dashboards and modules (e.g. Reports and Report Modules). In the Contents tab, turn Show New Content "On" for all dashboard Functional Areas and "Off" for all module Functional Areas. New dashboards will then be automatically added to the Contents when the role security is updated, but modules will not.

1.05 Lists

1.05-01	**Naming Convention** Hierarchies should be prefixed with a letter signifying the name of the hierarchy and a number indicating the level. e.g. P1 Product Category, P2 Product Family, P3 Products. The is no need to add "L" as a prefix to lists 1.08-01 Don't use Emojis Good Practice Naming Conventions
1.05-02	**Always use a code** Using codes is more efficient for loading and using lists so strive to always have a code for lists. This is especially important for numbered lists. 1.05-04 Numbered List should always have a code
1.05-02a Static non hierarchy lists	***For simple static lists (such as Yes/No) it is OK not to have a code, but even then, Y and N can be used as codes***

1.05-03	**Avoid using List Properties**
	List properties are the same as line items, but have many limitations, so keep it simple and have one place for calculations i.e. Module.line item
1.05-03a Reference module line items	***Where the following exceptions are used, reference module line items through a formula where possible to keep the audit trail valid***
1.05-03b Numbered lists (and related actions)	***Assign actions, and associated filters require list properties***
1.05-03c Exports	***List properties can be used as export labels***
1.05-03d Conditional Dashboard navigation	***List properties are needed to facilitate navigation to different dashboards based on the selection in the list***
1.05-03e Dependent drop downs	***List properties are needed to create driver and dependant lists***

1.05-04	**Numbered List should always have a code**
	If you are using Create or Assign actions for Numbered lists, try and incorporate a "code creation" action, as part of another process
	1.05-02 Always use a code
1.05-04a Combination of Properties	***If there is no code in the source, you will have to use combination of properties. Try and avoid if possible as this is slower to import and more difficult to use without a code***

1.05-05 Format the Display Name property as a list if possible

Format the Display Name property for Numbered lists as List Formatted, not text. This is more efficient, saves a line item, minimizes text fields and simplifies mappings

1.05-06 Only use Top Level for ultimate parent

Only use Top level for lists that will need sums so not Currency codes, or True/False (Indicators), or children in composite lists

1.05-07 Avoid Top Level for large flat lists

If you need the totals, look to add dummy parents to make the calculations more efficient

1.05-08 Use Composite lists rather than ragged

Composite lists are more flexible and calculate more efficiently so try and "balance" the hierarchies whenever possible

1.05-08a Chart of Accounts, or financial reporting hierarchies

Chart of Accounts, financial reporting hierarchies are valid uses of non-composite lists

1.05-09 Create Placeholders in general lists

Add place holders to organise General Lists and add Subsets and Line Item Subsets at the bottom

1.08-01 Don't use Emojis

1.05-10	**Never delete list and reload the list as a daily occurrence** Use a unique key to update to values rather than delete and re-load
1.05-11	**Do not embed Numeric values or Dates within the Code of a list** Dates and values are "data" and should not be part of a code
1.05-12	**Use formulas to derive properties of the list** Based off the code of the list it should be possible to derive the properties; Calculating the values is more efficient than storing text fields
1.05-13	**Use Flat lists to store Metadata (in a system module)** Avoid having hierarchies in Data Hub; these should only be applicable to the main planning models 5.07-01 No Hierarchies

1.06 Subsets

1.06-01	**Naming convention** Prefix the subset with the name of the list (e.g. P3 Products: Active Products) 1.08-01 Don't use Emojis

1.06-02	**Don't use subsets on large lists** It is better to create a list on its own if the Subset is more than 75% of the list
1.06-03	**Avoid single item subsets** If possible, try and avoid single item subsets, if there is a top level in the list, a single item subset will always have two members

1.07 Line Item Subsets

1.07-01	**Naming Convention** Prefix with LIS: then a description or the module used in the Line Item Subset. For Line item subsets using multiple modules, use a generic name that best describes its purpose 1.08-01 Don't use Emojis
1.07-02	**Version Line Item subsets** Use Line Item subsets to create different numeric formulae to avoid multiple Ifs

1.08 Emojis

1.08-01	**Don't use Emojis** Don't use Emoji or symbols in any naming of lists, modules or actions. They can cause issues with integration and different browsers

2 Engine

This section describes the rules around the Anaplan Engine. At the heart of Anaplan is the **Hyperblock** in-memory engine which combines the cell-based flexibility of a spreadsheet, the scalability and querying capabilities of relational databases as well as the calculation/aggregation abilities of multidimensional cubes.

2.01 Modules

2.01-01	**Naming Convention** As short as is practical. Use Alpha numeric to help identify modules from long names. There is no need to prefix modules in line with DISCO. It is better to align them with functional areas 4.01-10 Turn off the Module name Naming Conventions
2.01-01a Space saving for User Facing Modules	***User facing modules can be renamed to a more descriptive name if required to save on dashboard space, but be mindful of not cramming too much onto a dashboard as per the general design principles***
2.01-02	**Functional Areas** Use functional areas to categorise modules
2.01-03	**Add "dummy" modules as separators** Use these to provide sections and ordering to the Modules list, but don't use Emojis 1.08-01 Don't use Emojis

2.01-04 Use the DISCO methodology for Module design

Within functional areas, group "like" modules together. Design for the type of module and use the appropriate structures

Best Practices for Module Design

2.01-05 Don’t use DISCO as functional areas

The DISCO convention is designed to categorise types of modules. Within a functional area you may have multiple types of modules

2.01-04 Use the DISCO methodology for Module design

2.01-05a Simple models	***If the model is simple, DISCO can be used as the naming for functional areas***

2.01-06 Avoid using Subsidiary Views

Subsidiary views are difficult to work with and audit. Try and minimise their use especially for calculation modules; create groups of calculation modules instead with like dimensionality

2.01-12 Group formulas with like dimensionality

2.01-06a Display on Dashboards for analysis or filtering	***Sometimes, to enhance the end user experience, attributes are needed to be shown on dashboards***
2.01-06b Ratio Formulas	***Numeric values for ratios need to be in the same module and are unlikely to need the full dimensionality***
2.01-06c Synchronization for some dashboards	***If you have dashboards that need to synchronize from a module, sometimes a specific line item is needed***
2.01-06d Line items needed for sorting	***Sometimes to facilitate a sort, the line items don't need the full module dimensionality***
2.01-06e Reporting or Export modules	***To utilise filtering in reports or to provide attributes for exports, you may need to have line items for the non-dimensional attributes. Ensure these are set as subsidiary views to minimise the calculations***

2.01-07 Time Settings Module

Create all functions and filters relating only to time in separate modules by week, month, quarter, year etc. They will only re-calculate on model opening and when the time settings change

2.01-08	**Each major hierarchy should have a System module** Create a System module to support all standing data and attributes about the hierarchy if it is used. As a minimum have the code and parent as line items. if the hierarchy is not referenced in a formula, as a filter or a selector module on a dashboard then it is not needed Best Practices for Module Design
2.01-09	**Use Lookup or Constants Modules** Create a module with no dimensions to hold assumptions for Time, and other "SELECT" values 2.02-14 Avoid using SELECT
2.01-10	**Calculation Modules** Turn off summary options by default. It is better to use SUM for any downstream aggregation if not all levels are needed
2.01-10a All aggregation levels are needed	***If all aggregation levels are needed, then it is faster to use the native aggregation than use SUM formula***
2.01-11	**Keep the Dimension Order consistent** Ensure the dimension order is consistent between modules. Calculations are faster if the common dimensions are in the same order in the applies to. The size of the list is not as critical as the order Dimension Order

2.01-12 Group formulas with like dimensionality

Create Calculation modules to house sets of line items that use the same dimensionality. Avoid using Subsidiary views

2.01-06 Avoid using Subsidiary Views

2.01-13 Separate Transaction data from Attribute/Property data

Keep the non-time-based data in a separate module from static attributes

2.01-14 Avoid using Select Levels and filters together

Don't use Select levels combined with filters on the same hierarchy. It is a double filter and inefficient. Use one or the other. Setting a Boolean line item in a System module with 'none' as the summary option will often achieve the same result

2.01-15 Filters in separate modules

Try and keep filters in separate System modules. They can then be reused for different modules. In the notes section list where this filter is used to prevent accidental deletion

2.01-16 Use Data Tags

If not using for other reasons, use Data Tags to signify the DISCO category

2.01-17	**DCA Access Driver modules** Create separate modules for Access Drivers for relevant combinations to be reused. Use different line items for different settings if needed

2.02 Formulas

2.02-01	**Nested IFs** Avoid using multiple IFs. It is better to split the formula into more line items, use LOOKUPs or alternative constructs
2.02-02	**<12 expressions in a formula** If it takes you longer than one simple sentence to describe what the formula is doing, it is probably too long. Try not to mix expressions
2.02-03	**No Repeated expressions** If the expression is repeated in the formula (or other modules), put it on a separate line item. "Calculate once, reference many times"

2.02-04	**Text Strings** Treat text strings with caution. Try and avoid multiple joins and split common joins to separate line items 2.02-05 Create "joins" in smallest hierarchy Formula Optimisation in Anaplan
2.02-05	**Create "joins" in smallest hierarchy** If a text string join is needed, create the joins in the smaller hierarchies first to minimise the size of the text strings 2.02-04 Text Strings Formula Optimisation in Anaplan
2.02-06	**Comparing numbers** It is faster to check A=B rather than A-B=0
2.02-07	**Use Booleans instead of 1s & 0s** Booleans take up 1/8th of the space as a number, so unless a numeric value is needed use TRUE or FALSE

2.02-08 Never use SUM and LOOKUP together

SUM and LOOKUP used in the same expression generally cause large formula calculations and may cause intermediate relationship calculations especially if Time is a dimension or when the source and target structures are very different. Splitting them into different modules and line items considerably reduces the size of the intermediate calculations

2.02-09 Aggregation rules (ANY, ALL, FIRSTNONBLANK)

Utilise these summary methods to minimise the use of additional line items and IF statements

2.02-10 Using PREVIOUS vs CUMULATE

For long timescales, using PREVIOUS is faster than CUMULATE due to the number of "reads" required for the calculation. So, the expression should be: 'Calc line item' = 'data line item' + PREVIOUS('Calc line item') rather than CUMULATE('data line item').

2.02-10a Short Timescales

Where the number of periods is small, Cumulate is faster

2.02-11 Avoid using TEXTLIST

TEXTLIST requires a lot of memory for calculations and should be avoided if possible, using two dimensional modules and Boolean flags with ANY is a good alternative

2.02-12 No Hardcoding to List members

Avoid direct references to list item. E.g. IF ITEM(list)=list.xx

2.02-13 Only use POST for its specific purpose

Don't use POST for simple data offsets. OFFSET, LAG or MOVINGSUM are more efficient

2.02-14 Avoid using SELECT

Avoid hard coding using SELECT if possible. Use a constants module and use LOOKUP instead

1.01-01 Never use SELECT with Time

2.02-14a Versions

It is OK to use SELECT for versions

2.02-14b For top level items on non-composite lists

In Non-Composite lists, it is not possible to use LOOKUP to return the value, so it is OK to reference the Top Level item with SELECT. For other items, it is better to use dummy parent lists and additional modules

2.02-15 FINDITEM on blanks

If the list being referenced does not contain blanks, there is no need to check for blanks with IF ISNOTBLANK(List) THEN FINDITEM(List, Text) ELSE BLANK

2.02-16 Use conditionals to stop unnecessary calculations

In multiple conditional statements, try and include a conditional to prevent further referencing in the formula if the condition is satisfied

2.02-17 Put the most common condition first

For formula efficiency, put the conditional with the most common occurrence first in the formula

2.02-18 Break up formulas

The engine works more efficiently when calculations are broken up into separate line items. So, break up formula expressions where possible. This is especially true for calculations that are referenced many times and/or calculations that don't change that often

Formula Structure for Performance

2.02-19 Don't daisy chain

Always refer back to the ultimate source if possible, to avoid creating more dependencies than necessary. This allows more parallel calculations to be run increasing efficiency and speeding up calculations

2.02-20 Don't use RANK formulas with long timescales

RANK is a calculation intensive formula that cannot multi thread. Used in conjunction with long timescales and/or a large version list this can lead to poorly performing calculations. The same applies to RANKCUMULATE

2.03 Line Items

2.03-01	**Turn Summary options off by default** Most line items in the model will not need summary calculations, so it is good practice to turn summaries off initially and add then back when necessary. This is especially true for Calculation modules and conditional formatting line items 2.01-10 Calculation Modules
2.03-01a User facing modules on dashboards	***For Input or Output modules, specifically those line items displayed on Dashboards, summary options are likely to be needed***
2.03-02	**Try not to use TEXT as a format, or limit it as much as possible** Anaplan (as with computers in general) is optimized for numbers and Booleans. Text formats use more memory than any other format due to the way computers need to store the data. Minimise the use of text if possible 2.02-04 Text Strings
2.03-03	**Headers** Ensure headers are set to No Data to avoid unnecessary calculations

2.04 Saved Views

2.04-01	**Naming convention** Give the view a meaningful name (e.g. Dashboard Grid, Filtered View, Export etc.) There is no need to add the module name within the name of the view Naming Convention
2.04-02	**Filtering on views** Filter on a Boolean and only have one filter per view 4.02-01 Use efficient filters
2.04-03	**Keep the Default View clean** Keep the default view as is, no hiding, filtering, or conditional formatting
2.04-04	**Nested Views** If possible, nest the dimensions in order of size (largest to smallest)

3 UX Principles

This section covers general **design principles**. They are broad UX guidelines that apply to all applications, but they have been tailored for Anaplan

3.01 Hierarchy of information

3.01-01	**Principle** Done well this makes the content easy to read. It creates a path for the viewer's eye to follow through the page/screen
3.01-02	**Tell a story** Use a landing page to provide key summary information. This is useful as it orients new users around the model's contents, provides a summary as well as navigation to more detailed pages
3.01-03	**Summaries first** Place summary information first within a category, with detailed data pages afterwards. Add instructions that guide and explain what users are to do.
3.01-04	**Order of importance** Consider what's most important, and place that at the top of the page, e.g. KPIs

3.02 Smart Grouping

3.02-01	**Gestalt Principle** The mind tends to group items together to simplify input. Grouping your pages together visually helps the user to make sense of what they're seeing, and find what they're looking for quicker
3.02-02	**Contents page** Split content up using categories in a way that makes sense to your users e.g. functional areas. Consider the story, and group and order you pages accordingly
3.02-03	**Group like content** Group related and/or connected data together

3.03 Reduce Visual Load

3.03-01	**Hick's Principle** Fewer items on a page enable faster decision making

3.03-02	**Summarize data** When working on your landing pages, try to minimize or summarize the data on these pages and provide links through to more detailed pages via the contents 3.01-03 Summaries first
3.03-03	**Keep it clean** Limit the use of color in charts and graphs, and keep the page as clean as possible
3.03-04	**Split it up** Consider using multiple pages instead of putting all elements on one

3.04 Progressive disclosure

3.04-01	**Principle** Present data in digestible chunks. start simple and expose complexity when needed and not before
3.04-02	**Summary** Rather than put everything on a page, provide key information and link to another page to provide users with more detail 3.03-02 Summarize data

3.04-03 **Detailed Analysis**

On detailed pages users can do more granular tasks, such as perform detailed analysis, and manipulate and edit the data. Ideally, the driving grid of the page should represent a more detailed view of the previous page

3.04-04 **Use Contents**

Use the contents panel to place links that are useful to navigate to and from the current page but aren’t necessary to see on the same page

3.05 Use Consistency and Standards

3.05-01 **Consistency is key**

Users should not have to wonder whether different words, situations, or actions mean the same thing. Use consistent forms (e.g. images, naming, words) and colors to communicate

3.05-02 **Naming convention**

Name the same elements across different pages with the same name, keep links to pages and target page names consistent, so that users aren't confused when moving between pages

3.05-03	**Placement** Use consistent placement of those elements, by placing the same information in the same spot across pages
3.05-04	**Colors** Use consistent colors for the same metrics shown in different places

3.06 Provide Help and Guidance

3.06-01	**Guidance** Provide users with easy access to help on how to use the model
3.06-02	**Instructions** Use text and instructions where possible, although keep it simple and don't overwhelm the user with verbose language
3.06-03	**Tool Tips** Use customizable tooltips to help describe the visualizations and give a user better information about what they’re seeing

3.07 Use the correct data type

3.07-01	**Grids of data** Grids are best for presenting data when it's useful to see large sets of specific values and understand their meaning e.g. sales figures against products and time. Grids shouldn't be used for visualising high-level trends or comparisons between sets of data
3.07-02	**Line Charts** Line charts are best for seeing trends over a continuous timeframe e.g. revenue by month. They're especially useful for comparing multiple categories of data e.g. revenue of different products by month. Use contrasting colours so that users can easily differentiate series. Display a legend to make it clear what each series means Line Charts
3.07-03	**Column Charts** Column charts are great for comparing values across different line or list items e.g. FY actuals vs forecast. Column charts are also useful for comparing items over time e.g. revenue by month Column Charts

3.07-04 Bar Charts

Bar charts are great for comparing values across different line or list items e.g. FY actuals vs forecast. They shouldn't be used for comparing values over time. Bar charts are sometimes preferable over column charts if data labels are too long to fit on the X-axis

Bar Charts

3.07-05 Stacked Column/Bar Charts

Stacked Column Charts are best used to display part-to-whole relationships with multiple series e.g. revenue split into margin and expenses by time. Use contrasting colours so that users can easily differentiate stacked data

3.07-06 Pie Charts

Pie charts are best to use when you are trying to compare parts of a whole. They do not show changes over time

Pie Charts

3.07-07 Funnel Charts

Funnel charts are often used to represent stages in a sales process and show the amount of potential revenue for each stage. This type of chart can also be useful in identifying potential problem areas in an organization's sales processes. A funnel chart is similar to a stacked percent bar chart

Funnel Charts

3.07-08 Timeline charts

A timeline chart is an effective way to visualize a process using chronological order. Since details are displayed graphically, important points in time can be easily seen and understood. They are often used for managing a project's schedule, timeline charts function as a sort of calendar of events within a specific period of time

Timeline Charts

3.07-09 Waterfall Charts

A waterfall chart can be used for analytical purposes, especially for understanding or explaining the gradual transition in the value of an entity which is subjected to an increase or decrease in value. Often, a waterfall chart is used to show changes in revenue or profit between two time periods

Waterfall Charts

3.07-10 Combination Charts

Combination charts are useful where you want to validate the relationship between two variables that have different magnitudes and scales of measurement but are related in a meaningful way

Combination Charts

3.08 Give users visibility into status

3.08-01	**Provide Feedback** Ideally, the pages should always keep users informed about what is going on, through appropriate feedback within reasonable time. Being informed about the current status helps users decide what to do next in order to accomplish their goals, and figure out if their actions were effective or if they made a mistake

3.09 Match with real world scenarios

3.09-01	**Use familiar language** Speak the users' language, with words, phrases and concepts familiar to the user
3.09-02	**Consider the Business Process** It may be helpful to map out the process. Then in Anaplan, follow the real-world conventions and follow the real-world process when deciding which information to place where, so that it appears in a natural and logical order

3.10 Check in with end uses frequently

3.10-01	**During the design** Put users at the heart of the design. Check in with them before to learn what their needs are and help define what and how you build your instance
3.10-02	**During the build** Check in with them during your building process to ensure you are on the right track
3.10-03	**After Go-live** Check in after to make sure everything is working well for them
3.10-04	**User Feedback** Interview users for the initial check in, talk them through your ideas and pages during your building process. Afterwards, you can interview them, or have them use the system and watch them as they do. See where they get stuck or are having troubles and make adjustments to suit their needs

4 UX Build

This section covers the specifics of Anaplan **Dashboards**

4.01 Dashboards

4.01-01	**Back to front design** At the beginning of the design process, take the user stories and the UX Principles. Start with the end in mind Designing Dashboards
4.01-02	**Avoid horizontal scrolling** Try and avoid the screen scrolling horizontally. It is better to fix the grid and scroll within the grid 4.01-03 Consider the screen size of the end users
4.01-03	**Consider the screen size of the end users** Don't build based on your screen size. Consider the smallest size that users will be viewing the dashboard and build to that size. You should change the screen resolution to double check how the dashboards will look for the end users 4.01-02 Avoid horizontal scrolling

4.01-04	**Add Page Selectors rather than module selectors** Use separate page selectors rather than module page selectors, especially if the dimensions are repeated in multiple dashboard elements 4.01-05 Turn off module page selectors
4.01-05	**Turn off module page selectors** Show the page selector is appropriate but try and drive the selection from a general page selector 4.01-04 Add Page Selectors rather than module selectors
4.01-06	**Deep Level hierarchies** Create a separate dashboard with selection modules to keep the main dashboards clean. This also allows for more efficiency navigation up and down the hierarchy tree
4.01-07	**Avoid buttons for Navigation** Use the contents page for navigation rather than clutter the dashboards with buttons
4.01-07a Iterative processes	***When the user needs to toggle back and forth between dashboards, it is acceptable to have navigation buttons***
4.01-08	**Avoid Refresh buttons** Educate the users to use the Refresh option in the toolbar rather than action "refresh" buttons to open dashboards

4.01-09 Publish charts with full filters initially

If the chart has a filtered axis, set the filter such that all items are shown, then publish the chart. Then set the filter criteria back as desired; the chart will then respect the filter

4.01-10 Turn off the Module name

Use dashboard text to describe the grid, rather than display technical module names

2.01-01 Naming Convention

4.01-10a Space Saving

If screen space is at a premium, heading text can take up more space than desired, but do consider the overall user experience and don't clutter up the dashboard

4.01-11 Publish Process Actions instead of Import/Export Actions

Import and Export actions need to be republished if modified or replaced, whereas Process Actions are always consistent with whatever is held within the Process, making management easier

5.01-02 User Facing Actions in a process

4.01-12 Avoid using Delete action buttons on dashboards for end users

End users could potentially delete historical data, so keep the list element maintenance separate as part of the administration process

4.01-11 Publish Process Actions instead of Import/Export Actions

4.01-13 Store image URLs in a separate module

Keep all of the image URLs for a list in a separate System module. Be mindful of the correct dimensionality

2.01-04 Use the DISCO methodology for Module design

4.01-14 Enable Personal Dashboards

Use Personal Dashboards to enable users to personalise the views of the data

6.04-03 Master Dashboards

Personal Dashboards - Tips and Tricks

4.01-15 Limit modules with many formatted list line items

Too many list formatted line items displayed on a dashboard can adversely affect performance. Try and limit the number of displayed line items in a single module; split them into separate modules if necessary

4.02 Filters

4.02-01 Use efficient filters

For most efficient performance resolve filters to a single Boolean condition for each axis

4.02-02 Keep Filters in separate modules

Try and keep filters in separate System modules. They can then be reused for different modules

2.01-15 Filters in separate modules

4.02-03 Use time filters on all dashboards

Do not use show/hide function for time; these are static. Using the Time Settings module makes the filtering dynamic when the model calendar changes

2.01-07 Time Settings Module

4.02-04 Filter Selections in order of size

If using multiple conditions in a filter, include the conditions in order of list size with numeric conditions last

4.02-05 Use User filters to provide dynamic reporting

Create filter modules dimensioned by the users list to support dynamic filtering of module data

1.03-02 Performance and size can be dependent Users list

Increase end user adoption with smart filters

4.02-06 Avoid filtering on Nested Dimensions

If possible, try and avoid filtering on nested dimensions. Try and pivot the view differently. If this is not possible, ensure the filters are efficient

4.02-01 Use efficient filters

5 Integration

This section covers the various sections for **Import, Exports and Data Hubs**

5.01 Actions

5.01-01	**Naming Convention** Use numeric prefixes to signify the order within the process (e.g. 1.1 Import Products, 1.2 Import Product details). There is no need to include the technical module name in the description 5.02-01 Naming Convention Naming Convention
5.01-02	**User Facing Actions in a process** Import and Export actions need to be republished if modified or replaced, whereas Process Actions are always consistent with whatever is held within the Process, making management easier. 4.01-11 Publish Process Actions instead of Import/Export Actions
5.01-02a Numbered list actions	***Actions involving numbered lists (Create, Assign, Copy Branch and Delete Branch) cannot be placed in a process***
5.01-03	**Keep User actions (in general) to a minimum** Critically review the need for user driven actions, consider the effect on user concurrency. Attempt to use formulas instead. This may need additional modules, but the user experience could be improved as a result

5.01-04 **Delete one off actions**

For imports that are one offs, delete the import and data source

5.02 Processes

5.02-01 **Naming Convention**

Use "friendly" names for user facing processes. For data hubs or administrative processes use numeric prefixes to signify the order of the processes

5.01-01 Naming Convention

Naming Convention

5.02-02 **Keep actions in a process to a minimum**

Each action in a process triggers a recalculation so try and minimise the number of actions

5.03 Source Models

5.03-01	**Remove unwanted sources** For one off imports, or when source models are no longer required, delete the source model to keep the list tidy

5.04 Imports

5.04-01	**From source system, create a code defining all attributes** Ideally, this would be a separate file that is unique vs. using the same file for the transactional load 5.04-02 Create separate files for attributes and data
5.04-01a Code is >60 characters	***If the code is >60 characters, you will have to use combination of properties***
5.04-02	**Create separate files for attributes and data** The data file should have the key and values based on the dimensions. Non dimensional data should be in a different file 5.04-01 From source system, create a code defining all attributes

5.04-03 Data or Time period should not be part of the unique row

Consider what actually makes the row unique; include only attributes in the key, not data fields or dates

1.05-11 Do not embed Numeric values or Dates within the Code of a list

5.04-04 Only include the fields that are needed

Use the Ignore field if there are any unwanted fields

5.04-05 Pre aggregate in the source

Wherever possible, aggregations should be done in the source system. This is likely to reduce the size of the import file meaning faster imports

5.04-06 Import the correct granularity

Only imported data at the granularity needed. There is no need to bring in transactional data at a weekly level when Planning happens at the month level

5.04-07 Import using the correct formats

Line items holding the Imported line items should reflect the data. List formatted, numbers, and dates. Don't use text (unless it is a true text field)

5.04-08	**Never use a list as an import source** Import sources should always be done from a module view. This allows for filtering and only including the elements required for each import 5.04-08 Never use a list as an import source
5.04-09	**Always use saved views as import sources** Modules and saved views should be used as the source for other imports
5.04-10	**Unify the date format** Use a generic date format (e.g. YYYYMMDD) whenever possible to simplify the imports and remove the need for date mismatches and manipulations

5.05 Exports

5.05-01	**Naming Convention** Keep it simple. Export, Build etc. There is no need to include the module name in the view name Naming Convention

5.05-02 Only include what is needed

Only include the line items that are required for the import. Create multiple views if the module is used for different imports. The number of columns in the view does affect the speed of the import. The fewer the columns, the faster the import will be

5.05-03 Split views for List and Module imports

Create two views from a source module. One for the import to the list (using name, code and parent), and show for the attributes to the associated System module

5.05-02 Only include what is needed

5.05-04 Ensure view filters are optimised

Only use one filter criteria...If more are needed, combine them into one line item

4.02-01 Use efficient filters

5.06 Import Data Sources

5.06-01 Naming Convention

Rename the import sources as soon as possible. For a saved view include the module name (e.g. module.saved view). For model imports, include a shorted version of the model name (e.g. model_module.saved view)

Naming Convention

5.06-02	**Delete unwanted sources** For one off imports, or when import sources are no longer required, delete the sources to keep the source list tidy

5.07 Data Hub

5.07-01	**No Hierarchies** There should be no need for composite list hierarchies in the Hub. They can be built to "test" the actions, but after testing they should be deleted
5.07-01a Validation purposes	***If data is needed to be consolidated to check against source systems, although the flat modules with attributes can be used to sum data***
5.07-02	**No Analytical modules** Try and keep Analytical modules out of the Hub
5.07-03	**Flat lists for data export** Use the flat list structures to create modules and views for downstream exports

5.07-04	**Get the correct format** Get the data from IT in the correct format as well as the correct granularity 5.04-06 Import the correct granularity
5.07-05	**Use System modules** Use System modules for filtering data (current period, current FY year, etc.)
5.07-06	**Master data should be built by the source system** Try and avoid creating Master Data in the hub. This should really come from the source system(s)
5.07-07	**Keep Transactional Analysis out of the planning models** Use the Data hub (or another reporting model) to keep detailed transactional data out of the main planning models, Large amounts of historic transactional data can inflate the size of the planning model and lead to reduced performance
5.07-08	**Pre aggregate for downstream exports** It is more efficient to aggregate the data in the hub and then export it, rather than accumulate it through the import to the downstream models

5.07-09 Avoid using a Top Level for large lists of transaction data

Don't create a transaction list with a top level. The calculations will need to sum for all items in the list even if only a single item is added

5.07-10 Add intermediate sub totals to transaction lists

5.07-09a Check totals ***If a check total of all transactions is needed, but again using the flat attributes modules is preferable***

5.07-10 Add intermediate sub totals to transaction lists

If you need the totals for validation purposes, create intermediate subtotals within the transaction list. This will significantly reduce the calculation load

5.07-09 Avoid using a Top Level for large lists of transaction data

6 Application Lifecycle Management

This section covers the process of **Application Lifecycle Management** (ALM)

6.01 Revision Tags

6.01-01	**Create a Naming standard** Be consistent with how you name your revisions. It is advised to use a Major.Minor nomenclature (e.g. R1.01, R1.02, R2 etc.). Remember, the description is permanent and cannot be changed - Be careful!
6.01-02	**Create a Revision Tag at least once a day** During periods of intense development, ensure that revision tags are created regularly and tested against a "shell" model Revision Tag Best Practices
6.01-03	**Revision documentation** If you need a fuller description of revisions, create a normal list and then house the revision details in a module with other fields such as Approved date User Story, Developer, Tested date, Sign off date etc. Revision Tag Documentation

6.01-04 Synchronise Production regularly

Minimise the risk of sync errors by synchronising regularly to keep Production and Development aligned

6.01-02 Create a Revision Tag at least once a day

6.01-05 Dummy Test Model

Set up a "shell" model (using create from revision) that contains no production list members on which to test revisions and synchronises regularly

6.01-02 Create a Revision Tag at least once a day

6.02 Production Lists

6.02-01 Define Production lists before the first sync

Don't set all lists to production initially. Setting a list back to Structural (after a synchronise) will remove data from the existing production list even if the members are the exact same

Prepare Applications for ALM

6.02-02 Formula protection - Hard coding

You cannot hard code a reference to an item in a production list because that member may be deleted by an end user

Formula Protection

6.02-03	**Review the need for Production Lists** Only lists that require amending as part of the business process, or where the lists are populated by imports should be set to production lists
6.02-04	**Don't click and hope** Review the referenced by column of potential lists and check for hard coded formula references before checking them as production so you don't generate rollbacks 6.02-02 Formula protection - Hard coding

6.03 Deployed Mode

6.03-01	**Don't take the model out of deployed mode** Once ALM has been initiated, and Deployed mode turn on, the Production model should NEVER be taken out of Deployed mode
6.03-01a Development Model creation	***When copying a Production model to create the initial ALM environment, or when re-creating the DEV model as part of a "reset"***
6.03-01b There are no other exceptions!	***It is just not worth it. ALM brings control, but there are rules, and Deployed mode is the key rule***

6.03-02	**Use deployed in Dev mode for additional security** Setting deployed mode for Dev models is OK and can prevent inadvertent structural changes being made outside of normal development cycles
6.03-03	**Keep Test models in Deployed mode** Test models should be treated as Production Models. This gives a true representation of testing and also prevents inadvertent synchronises from Test to Prod

6.04 Managing changes during development

6.04-01	**Back to the Future** This is a technique that reverts the model back a revision state, allows you to make a fix, sync to Production, and then return Development back to where it was Saving incomplete changes during development
6.04-02	**Switchover** After a revision, create multiple revision tags with different switchover dates prior to starting new development Managing Frequent Structural Changes During Development

6.04-03 Master Dashboards

Master Dashboards will delete any personal dashboards. Techniques to minimise user disruption include creating a copy and have a migration process to move users to the new dashboards

4.01-14 Enable Personal Dashboards

Personal Dashboards - Tips and Tricks

6.05 Architecture

6.05-01 Make DEV the source for everything

DEV> Test, and DEV>Prod is more flexible; it allows multiple Test models, and Test models to be created and deleted without compromising Prod. It also supports segregation of duties

6.05-02 Archived Models

Compatible models (DEV, TEST or PROD) can be archived without breaking the link between them

6.05-03 Model modes

6.05-03 Model modes

When restoring Archived models, restore the model to Deployed mode if this is a Production or Test model

6.05-02 Archived Models

6.05-04	**Keep the Development model small** Keep the Development model as small as possible, Try and only use a selection of items in the production lists to minimise the model size. Use the “create from revision” functionality as part of the set up Prepare Applications for ALM
6.05-05	**Use a Data hub to populate the Development Model** Use saved views within the Data Hub to populate the Development model. This allows defined data and structures to be imported to assist initial development and component testing