Either by mistake or design, applications with large data units will have expected performance issues. OneStream has features to solve for these problems that do not require a redesign.

Data Processing and Performance

A comprehensive guide of tables, and design

Tom Shea; Peter Fugere

December 2021

TABLE OF CONTENTS

OVERVIEW

To maintain well performing application, one must understand how the underlying database works and more importantly its limitations. Understanding how a system works, allows designers and administrators to create reliable, stable, and optimal performing applications.   This white paper is intended to guide the design of those optimal data processing strategies for the OneStream platform.

First, this document will provide a detailed look at the data structures used by the stage engine as well as those used by the in-memory financial analytic engine, providing a deep understanding of how the OneStream stage engine functions in relation to the in-memory financial analytic engine. The relationship between stage engine data structures and finance engine data structures will be discussed in detail. Understanding how data is stored and manipulated by these engines will help consultants build OneStream applications that are optimized for high-volume data processing.

Second, the workflow engine configuration will be examined in detail throughout the document since it acts as the controller / orchestrated of most tasks in the system. The workflow engine is the primary tool used to configure data processing sequences and performance characteristics in an OneStream application. The are many different workflow structures and settings that specifically relate to data processing and these settings will be discussed in relation to the processing engine that they impact.

Finally, this document will define best practices and logical data processing limits. This will include suggestions on how to create workflow structures and settings for specific data processing workloads. With respect to data defining processing limits, this document will help define practical / logical data processing limits in relation to hard/physical data processing limits and will provide a detailed explanation of the suggested logical limits. This is an important topic because in many situations the physical data processing limit will accept/tolerate that amount of data that is being processed, but the same data may be able to be processed in a much more efficient manner by adhering to logical limits and building the appropriate workflow structures to partition data.

These concepts are particularly important because they enable efficient storage, potential parallel processing and high-performance reporting/consumption when properly implemented.

STAGE ENGINE OVERVIEW

The primary goal of the OneStream platform is to turn DATA into INFORMATION. The stage engine plays a critical part in the information process. The OneStream platform supports analytic information generation across a wide variety of business use-cases. Different use-cases place a variety of demands on the stage engine. Certain use-cases place a premium on audit and transparency above all other requirements. Other use-cases demand the fastest possible transformation and audit/transparency are less concerning.

The stage engine is an important part of the OneStream platform. It is responsible for interacting with the “outside world” to read, transform, enrich, and store source data in a way that can be reliably consumed by other data dependent engines in the platform. The Stage engine is more than an ETL. This engine must deliver all the capabilities of an ETL with the added value of transformation transparency (audit) and source system data summarization / exploration (drill-back and drill-around).

Within the OneStream community, the Stage engine is described as **Data Preparation Engine**. This term fits because more accurately describes the overall capabilities and responsibilities of the engine. The problem of collecting source system data becomes exponentially more difficult from a functionality and a performance perspective when you incorporate transparency audit and data quality requirements.

USE-CASES

CONSOLIDATION / STATUTORY REPORTING

- Data Context (High Durability – protected and sacred)
- Book-of-Record data with strict audit and transparency requirements
- Stage standard / full audit data loads should be used.
- High data retention requirements due to statutory compliance.
    - Requires source documents storage.
    - Requires Transformation Rule Instance storage and lineage.
    - Requires data storage for 10 years or more.
- Extremely high data validation requirements.
- Extremely high financial processing intelligence.
- * Direct Data Loads * should **NEVER** be used for statutory reporting.
    - Direct data loads do not store detailed transformation audit information.
    - Direct data loads do not store transformation rule instances.

FINANCIAL PLANNING

- Data Context (High Durability – protected and sacred)
- Book-of-Record (Related) but does not carry the same transparency requirements.
- Financial planning information is highly aligned to consolidation and most data preparation requirements are like external statutory reporting requirements since data is actual and planning data is expected to be fully aligned and auditable.
- Medium data retention requirements due to statutory compliance.
    - Financial plans and guidance are part of external stake holder reporting.
    - Financial plans must be maintained and verifiable in support of public company earnings guidance.
- Extremely high data validation requirements.
- Extremely high financial processing intelligence.

OPERATIONAL PLANNING

- Data Context (Low Durability – Agile and Iterative)
- Operation planning data and associated metadata changes frequently which impacts the frequency and volume of data that must be processed.
- Operational planning data is collected and manipulated as part of an iterative process that eventually needs to be rationalized and moved into the Financial Planning data as a specific segment / subject area of the broader financial performance picture.
- Agile planning data that has low data transparency requirements.

- Operational planning source data tends to be more transactionally oriented which requires a sub-ledger processing mentality where plans are captured at an item level (register) to match operational and function data requirements.
- Medium data validation requirements.
- Medium-Low financial processing intelligence.
- * Direct Data Loads * are an appropriate processing method for operational planning use-cases.
    - Data loading frequency is high (data is more throw away)
    - Data audit / transparency are not a priority.
    - Data volumes are high and optimal processing time is required.
    - Avoiding detailed audit storage is provides performance and scalability optimizations.

TRANSACTIONAL ANALYTICS (FINANCIAL SIGNALING)

- Data Context (Extremely Low Durability – Snapshot throw-away daily view)
- Transactional data involves high volume data processing since intermediate system processes, GL Posting, are being bypassed to go straight to the transactional data source.
- Transactional data is very “un-intelligent” and requires significant enrichment to turn it into financial intelligent information.
    - Translations to common currency
    - Account type mathematical interpretation
    - Summarization
    - Hierarchical aggregation with account type interpretation
- Low data validation requirements.
    - Data is processed daily.
    - Data is stale / irrelevant quickly.
    - Data is dropped and recreated frequently.
- Medium-Low financial processing intelligence.
    - Financial intelligence is an especially important part of the transactional information value chain.
    - Exceedingly high data volumes must be enriched quickly, and medium-low financial intelligence must be applied to the resulting data so that it can be related back to the higher-level financial information that it is intended to describe (signaling).
- * Read Only Snapshot *
- * BiBlend Data Loads * are used to prepare transactional analytics optimized reporting tables.
    - Daily frequency is most common (Nightly refresh of current transactional view)
    - Data is stored as Relation Table with Column Store Indexes
        - Enables fast / high volume data preparation.
        - Provides extremely fast reporting / query response times on hundreds of millions – billions of rows.
    - May require very-high compute resources depending on data volume and data enrichment.

CORE CAPABILITIES

DATA PARSING

- Flat File Parsing
- Delimited File Parsing

- Direct Data Connections
    - Database OLE DB / ODBC
    - Web Services
    - Programmatic Custom Integrations (.Net)
- Data Management Sequence
    - Used to extract data from a cube and to map it to another target.
    - Supports cube-to-cube data movements that require transformations.
    - Extract cube data to staging table as fact-table source that can be consumed by external systems.

DATA ENRICHMENT

- Derivative Rule engine allows the stage engine to "Derive" new rows based on a powerful relational expression engine.
- Business rule enabled parsing engine allows for unlimited source data enrichment.
- Rich Event model enables standard OneStream Stage engine workflow processes to be supplemented with "Customer Specific" process extensibility.

DATA TRANSFORMATIONS

- In-Memory mapping / transformation engine supports sophisticated rule structures without the need to sacrifice auditability.
- Market leading transformation engine provides an unmatched set of transformation rule capabilities than can be supplemented further with business rule extensibility.
    - One-To-One
    - In (List)
    - Composite (Cross Dimensional Dependencies)
    - Range
    - Mask (Wildcards for both criteria and target – "Two Sided Wildcards")
- Conditional / business rule expressions can be written on all rule types except One-To-One.
- High-Performance parallel transformations enable hardware scalable processing.

CONTENT VALIDATION

- Check Rules (Derivate Rule Type) enable validation of source data quality.
- Event Rules (Business Rule) enable custom evaluation of in memory source data quality.

TRANSFORMATION VALIDATION

- Standard validation and exception handling for source data transformation.
- Fully transparent analysis of transformation mapping process (Drill-To-Rule) to eliminate "Black Box" transformation processing.

LIGHTS-OUT AUTOMATION (FULLY TRANSPARENT)

- Batch File Harvesting
    - The engine can be orchestrated by the Workflow Batch File Harvesting engine.

- Source files or trigger files (Empty file that trigger direct integration) can be dropped into an “open batch” folder for automated processing tied to a scheduled Data Management job.
- Batch file processes supports parallel processing of files for increased through-put.

- Business Rule Custom Automation
  - The OneStream BRApi provides a complete set of APIs that enable custom orchestration of data loads within larger data processing jobs.
- Automated tasks can still be executed via workflow providing full transparency to the automated task which makes error investigation and reprocessing easy.

DATA TRANSPARENCY

- Transformation Rule Auditing
  - Instancing for historical referencing
  - Keyed relationship between map rules and transformed source data.
- Detail Data Storage
  - Source data is stored at the level that it is read to maintain discrete and auditable linkage between external system source data and the transformed OneStream financial data.
  - *New* OneStream release 6.4 introduces Direct Loads which enable the optional bypassing of storing detailed transformation data when not needed.
- Summary Data Storage
  - In many cases transformation rules map multiple source values to a single target intersection resulting the need to aggregate multiple rows to a single value.
  - The summary target data is used as an audit linkage table between the financial cube and the source detail data.
- Process Execution and Task Logging

MULTIPLE TARGET USES

- Financial Cube (OneStream is Book-Of-Record System)
  - From a financial reporting and financial planning perspective, the primary purpose of the stage engine is to organize source data into a set of valid intersections that can be written to cells in a financial cube.
  - Successfully loading a financial cube requires a great deal of audit and validation because potentially low-quality external source data must be transformed into specific cube intersections (Cells).
- BiBlend (Financially Intelligent Transactional Analytics Snapshots)
  - The stage engine can leverage the structural definitions of a Financial Cube for the purpose of creating financially intelligent column-stored indexed high-performance relational analytics data stores.
  - BiBlend is an extension of the primary stage engine that uses the core parse and transform capabilities transform transactional source data and aggregate it using financial cube metadata definitions. The data is stored in a relational column store index.
  - BiBlend delivers high-performance / financially intelligent snapshot views of transactional analytic data by leveraging OneStream financial cube metadata to enrich, summarize and aggregate source data.
- Other (Non-OneStream)

- Ability to load data to stage data tables only enabling other downstream systems to leverage the power of the OneStream stage data preparation engine.

STAGE ENGINE PROCESSING

All stage process flows leverage the same parsing and transformation processes (*Blue Blocks in Flow Diagrams Below*). This means that setting data sources and transformation rules is the same across all load types. The primary difference between the different processes is how the data is stored, what data is stored and how the data will be consumed by a downstream process (Cube or Column-Store Indexed Table). The OneStream platform was built with reusability as a primary objective to support easy maintainability of applications. This allows administrators to learn the stage engine foundational features and to leverage those features across different stage processing types.

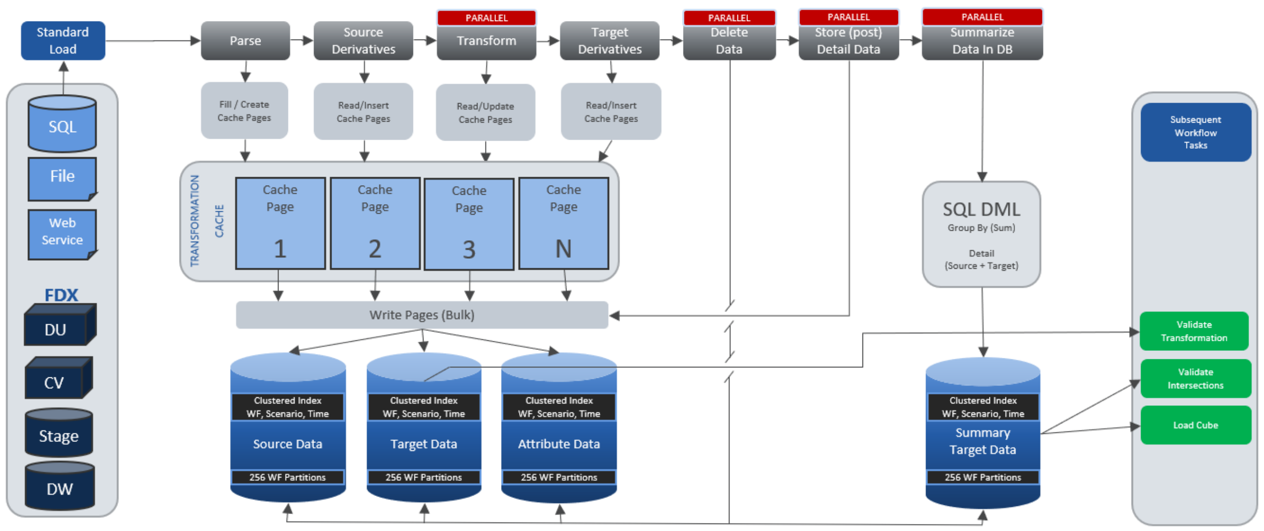
The following sections provide detailed processing diagrams of the three main types of stage engine processes.

STANDARD LOAD PROCESSING (FULL AUDIT)

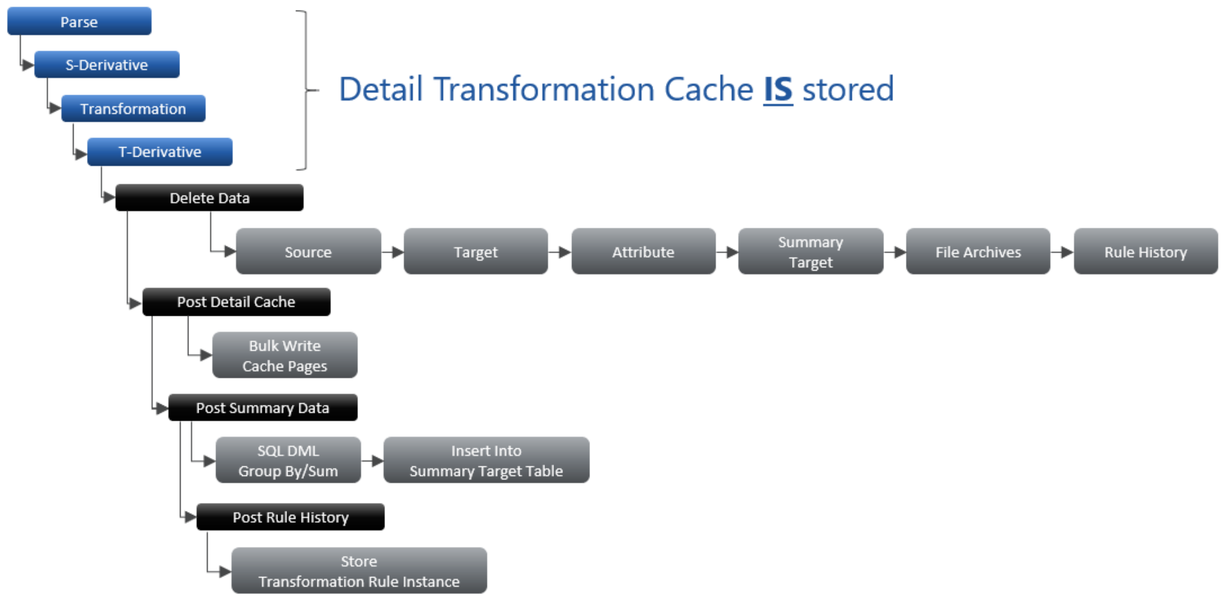
Standard data loads leverage the OneStream stage engine to create an end-to-end audit trail for every number that is written to an analytic financial cube. Executing a standard data load stores source data file, transformation rule instances, detail transformed data, summary transformed data and detail workflow execution information.

Target:	Financial Analytic Engine (Cube Load)
Goal:	Highest audit / quality to support financial information
Auditability:	Full Audit, Drill-Down, Drill Around

STORAGE DIAGRAM



PROCESS FLOW DIAGRAM

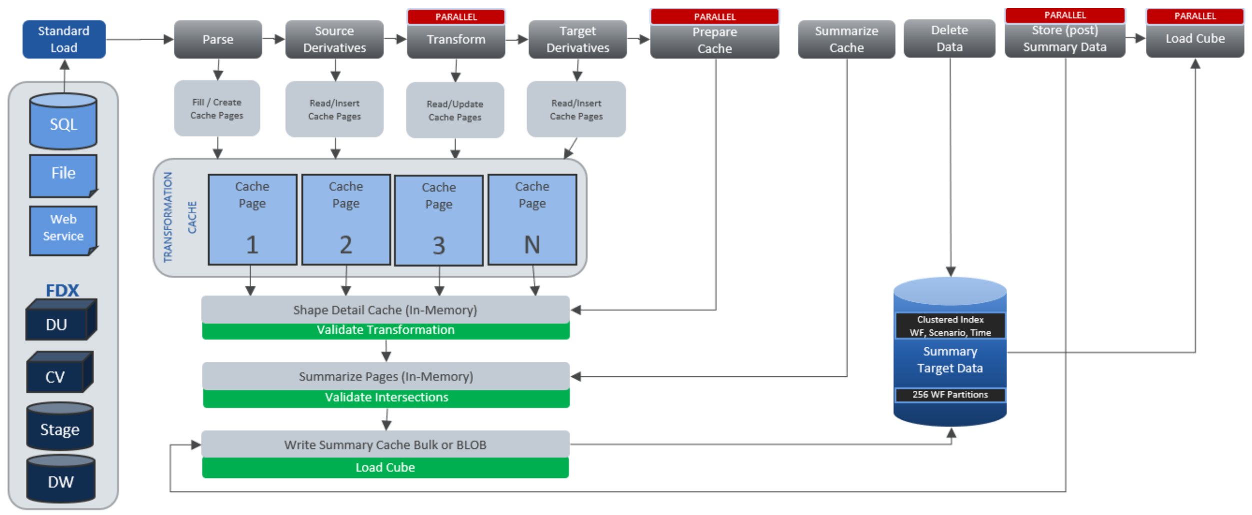


DIRECT LOAD PROCESSING (NO AUDIT)

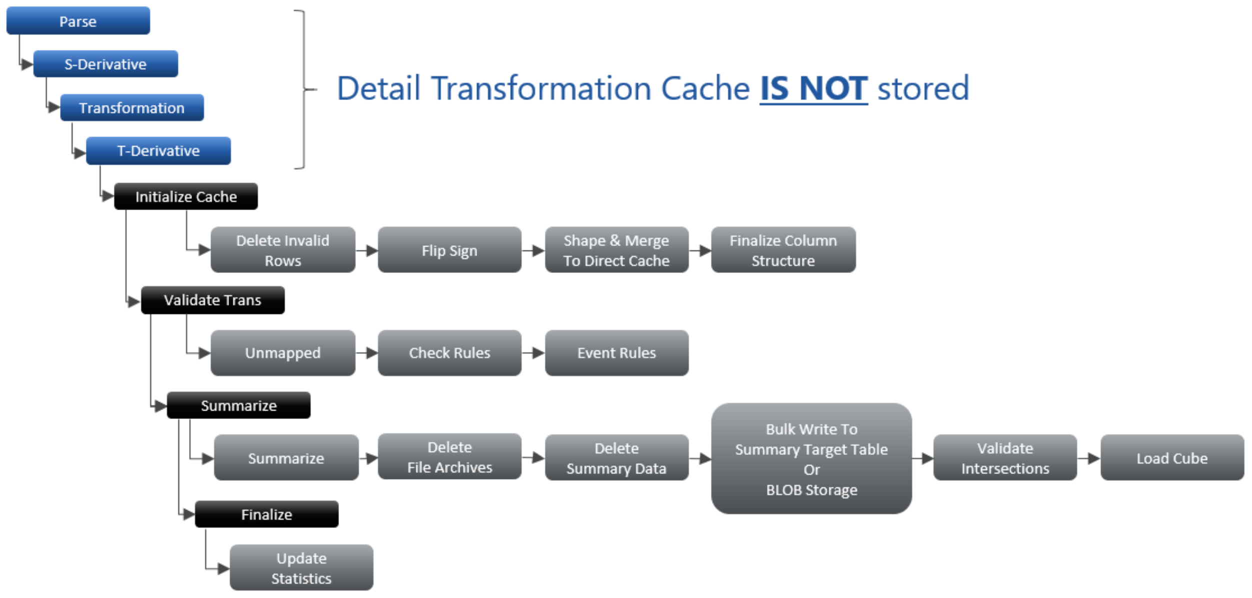
Direct Load leverage the stage engine to load an analytic financial cube, but detail transformed data and transformation rule instances are NOT stored to reduce the storage and performance burden encountered with a full audit load (standard load).

Target:	Analytic Cube Load
Goal:	Fast data load with lower audit / storage burden
Auditability:	No Detail Audit, No Mapping Instance Audit, No Drill-Down, No Drill-Around

STORAGE DIAGRAM



PROCESS FLOW DIAGRAM

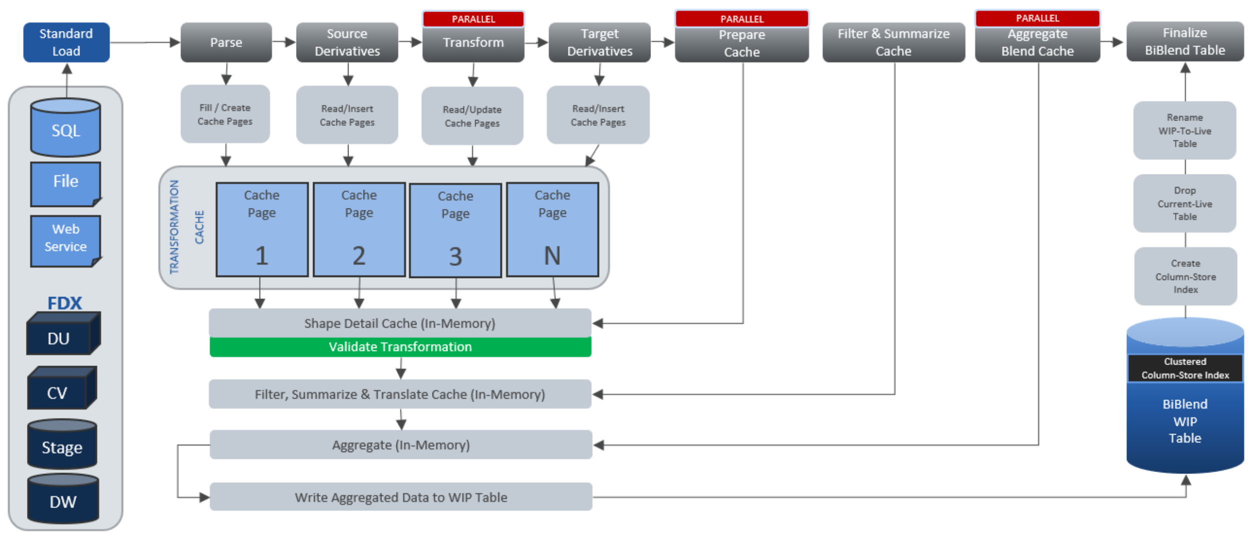


BIBLEND PROCESSING (CLUSTERED COLUMN-STORE INDEX)

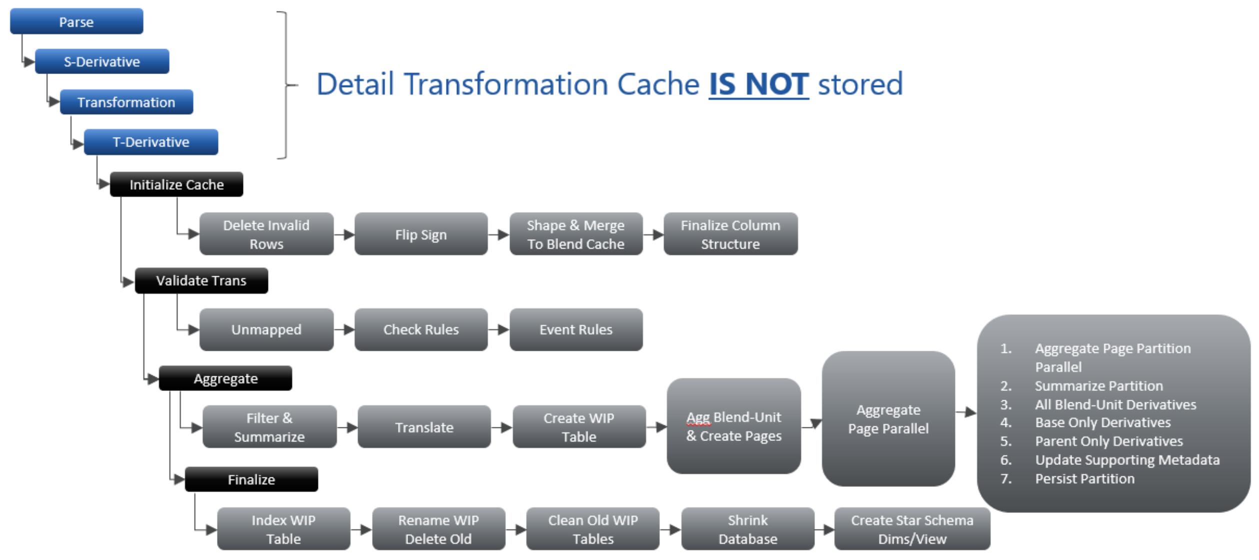
Bi-Blend Loads leverage the stage engine and elements of the financial analytic engine to create a relational table using column-store index technology for high-volume analytics. Bi-Blend loads use platform metadata structures to aggregate source data and enrich the aggregated data with basic financial intelligence.

Target:	Relational Column Store Index Table
Goal:	Fast transactional processing, financial enrichment, and snapshot creation
Auditability:	Low audit requirement, Data is intended to be dropped and recreated daily.

STORAGE DIAGRAM



PROCESS FLOW DIAGRAM



STAGE DATA STRUCTURES

The stage engine uses a combination of in-memory processing and relational database storage to deliver the parsing, transforming, and validating tasks that it provides to the OneStream platform. From a relational storage perspective, the stage engine stores metadata that defines data source definitions and transformation rule profiles/groups. The combination of these metadata values is used to initialize the stage engine so that it can parse and transform external data into high quality data that can be consumed, enriched, and presented by other engines in the platform.

STAGE ENGINE RELATIONAL TABLES

The list below details the relational tables that are used to store stage metadata and data.

METADATA TABLES

Table Name	Description
StageRuleProfiles	Contains Transformation Rule Group Combinations
StageRuleProfileMembers	Contains Transformation Rule Group Profile Instance
StageRuleGroups	Contains Combination of Rules for a Dimension
StageRules	Contains Detail List for Rules by Type
ParserLayouts	Data Source Definition
ParserDimensions	Data Source Dimension Specification
ParserDimensionAttributes	Data Source Dimension Detail Properties

METADATA INSTANCE TABLES (AUDIT HISTORY)

Table Name	Description
StageRuleProfilesHistory	History of Transformation Rule Group Combinations
StageRuleProfileMembersHistory	History of Transformation Rule Group Profile Instance
StageRuleGroupsHistory	History of Combination of Rules for a Dimension
StageRulesHistory	History of Detail List for Rules by Type

TRANSFORMATION DATA TABLES

Table Name	Description
StageSourceData	Detail Level Source Data Storage
StageTargetData	Detail Level Target Data Storage
StageAttributeData	Detail Level Attribute Data Storage
StageSummaryTargetData	Summary View of Joined Source & Target Tables
StageBiBlendInformation	BiBlend Execution Status
StageDirectLoadInformation	Direct Load Execution Status / Blob Storage

CONSUMPTION DATA TABLES

Table Name	Description
StageToFinanceLoadResult	Finance Engine Equivalent of StageSummaryTargetData Table
StageToFinanceValidationError	Intersection Error Detail
(1-N) BiBlendTables (Named by WorkflowClusterPk)	BiBlend Column-Store Tables used for Transactional Analysis

CONSUMPTION DATA VIEWS

Table Name	Description
vStageSourceAndTargetData	Joined view of Source and Target Data
vStageSourceAndTargetDataWithAttributes	Joined view of Source, Target, and Attribute Data
vStageSummaryTargetData	View of Summary Target Data used to Load a Cube

DATA TABLE PARTITIONS AND CLUSTERED INDEXES

All stage data tables use a common storage scheme at the relational table level. It is important to understand the concepts behind the partitioned/clustered Index storage architecture.

PARTITIONED TABLES

Partitioning is a relational database storage mechanism that allows a single table to act like many smaller tables (Partitions) within the primary table. Partitions are based on a key value which defines how to segment data as it read and/or written to the table.

In the case of the stage data tables the Partition Key is the first 4-byte segment the Globally Unique Identifier (GUID) that is assigned to any a workflow profile as its unique key. The partition function used by the OneStream platform for stage data tables creates 256 partitions. This means that every time data is to be stored for a given Import Workflow Profile, the first 4-bytes of its unique id are converted to an integer between 1-256. By relating a workflow profile's unique Id to a partition key and writing data to different partitions, more through-put can be achieved by the stage engine. Hopefully, this is obvious, when two or more workflows are trying to write data to a stage data table at the same time there is a high probability that the workflows will be writing to different partitions and therefore, they will not block/lock each other's inserts/updates/deletes.

CLUSTERED INDEXES

Now let us build on the discussion around partitions and add in the concept of clustered indexes. When you create a table (or a partition—a table within a table) you typically create indexes to speed up the access of data within the table/partition. Traditionally, indexes are created as hash indexes or pointers that are built to allow fast access to data that may be scattered all over the table (different storage blocks on disk). This type of index is powerful and performs well but can create performance problems for large Create / Read / Update / Delete (CRUD) operations because the database engine must locate and update data in different pages of data that can be out of sequence. For example, imagine updating a large spreadsheet that has a million rows of data and you need to delete rows that have a Status field value "Old."   If there was a hash index on the status field it would help you locate the rows that contain the desired status (Rows: 100, 2999, 3400, 999999). As a user you would have to locate each row and then delete it. The hash index would help you find the rows quickly by telling you to go to rows (100, 2999, 3400, 999999), but you would then have to spend time moving to those rows and deleting them.

Next, let us look at clustered indexes. A clustered index physically stores data on disk in the order of the indexed column(s). So why is this important? When an application needs to work with buckets of data (*Think about the OneStream WorkflowClusterPk—WorkflowProfileKey, ScenarioId, TimeId, [Example: Houston.Import, Actual, 2021M3]*) it is a big advantage in performance for CRUD operations to be able to locate the bucket of records in the index and access/manipulate them in one chunk. The OneStream stage engine is always working with buckets of data and never gets to focus on a single row. Clustered indexes are a particularly important mechanism that helps the stage engine find the extents of a WorkflowClusterPk. Since the platform is workflow focused, having the data organized this way makes it easy to find these buckets of data and operate on them in large chunks. Using the same example that was discussed with the hash index we can see this benefit. If a Clustered Index were used on the status field, the rows that contained a status value "Old" would all be in order (Example: 1000, 1001, 1002, 1003). As a user, you could just locate row 1000 and then delete rows 1000-1003 in one operation. This same advantage applies for the OneStream stage engine when try to locate and execute a CRUD operation on a bucket of data.

WORKFLOW AND STAGE ENGINE RELATIONSHIP

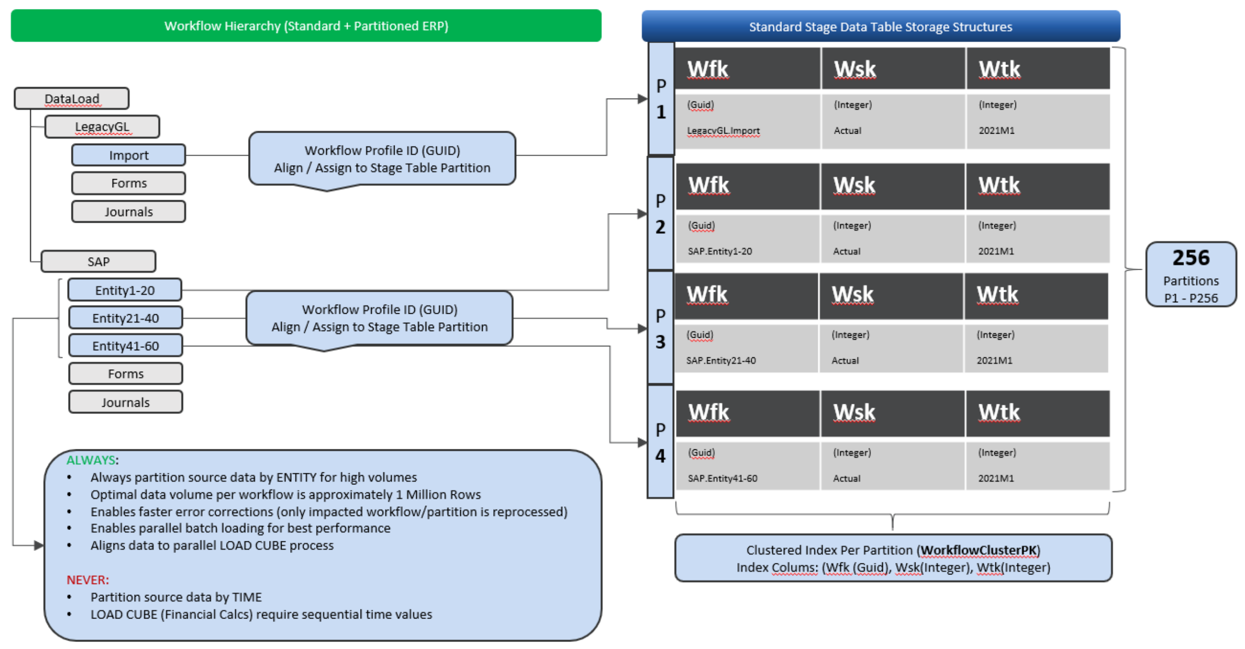
The stage engine, like most processing engines in the platform, is orchestrated by the Workflow engine. A workflow profile is used to define the combination metadata structures that are used when an import (stage) oriented workflow is executed. Workflow definitions also play a significant role in aligning stage data stored in relation tables to table partitions and clustered indexes.

The workflow profile Key (Guid) defines the partition (Bucket). In addition, the workflow point-of-view (Workflow Profile, Scenario, and Time) relates the clustered index structure used in all stage relational tables. From a platform perspective, this bucket of information is referred as a WorkflowClusterPk (this is a common object used in the OneStream platform Business Rule API – BRApi). This is a key concept because the stage engine primarily reads / writes data using the WorkflowClusterPk. Whenever the stage engine performs processing sequence, it is always executed against 1-N WorkflowClusterPk's (Data Buckets).

As mentioned above, the standard stage storage and index structures rely on the Cluster Index (Wfk, Wsk, Wtk) as the primary way to locate and perform CRUD operations stage data. This means that as a consumer of stage data you should ALWAYS query (Where Clause) by the clustered index columns first. There are little-to-no other indexes on the stage tables because of the potential to cause deadlocks and contention when the stage engine is reading/writing/deleting large volumes of data in parallel.

Creating multiple Import Child workflows is the key to data processing scale within the OneStream platform. Using workflow to partition high volume data loads provides for unlimited data volume processing with a high degree of performance tuning capabilities. Using many import workflow children as partitions instead of a single import workflow child creates opportunities for parallel processing as well as targeted error correction / reprocessing of source data.

WORKFLOW-TO-STAGE DATA STORAGE RELATIONSHIP DIAGRAM


Workflow Hierarchy (Standard + Partitioned ERP)
Standard Stage Data Table Storage Structures
DataLoad
LegacyGL
Import
Forms
Journals
SAP
Entity1-20
Entity21-40
Entity41-60
Forms
Journals
Workflow Profile ID (GUID)
Align / Assign to Stage Table Partition
Workflow Profile ID (GUID)
Align / Assign to Stage Table Partition
P 1
P 2
P 3
P 4
Wfk
Wsk
Wtk
(Guid)
(Integer)
(Integer)
LegacyGL.Import
Actual
2021M1
SAP.Entity1-20
SAP.Entity21-40
SAP.Entity41-60
256
Partitions
P1 - P256
Clustered Index Per Partition (WorkflowClusterPK)
Index Colums: (Wfk (Guid), Wsk(Integer), Wtk(Integer)

ALWAYS:

- Always partition source data by ENTITY for high volumes
- Optimal data volume per workflow is approximately 1 Million Rows
- Enables faster error corrections (only impacted workflow/partition is reprocessed)
- Enables parallel batch loading for best performance
- Aligns data to parallel LOAD CUBE process

NEVER:

- Partition source data by TIME
- LOAD CUBE (Financial Calcs) require sequential time values

SOURCE DATA PARTIONING CONCEPTS

Now that stage data storage structures have been defined, it is time to discuss the rationale behind partitioning and best way to design a data loading workflow for high-volume and high-performance processing. Since the stage engine is always operating on buckets of data (WorkflowClusterPk) it is important that data in the bucket can be located as a group of rows as efficiently as possible. When a relational table is partitioned, it behaves like separate independent table.

PARTITIONING AND PARALLEL PROCESSING ILLUSTRATIVE EXAMPLE

Let us use a case of different flavored soda-pop as an example of accessing buckets of information. Think of a case of twenty-four six oz cans of soda (six cola, six grapes, six lime, six cherry). You can think of each case of soda (twenty-four cans) representing a stage data table partition. Each flavor would relate to the clustered index (six cans of the same flavor). So whenever, we access a case of soda, we could locate the first can of each flavor and then know that next five cans would match the flavor we are after.

Let us say that we are having a party and we need to unpack ninety-six cans of different flavored soda. We have two choices on how to receive the soda, one giant case of ninety-six cans, or four cases of twenty-four cans. In addition, only one person can work on a case at a time. The person that is going to open the case and locate the flavors relates to a workflow (person performing the task… like the OneStream workflow engine). The cans for each flavor must be put in a party-bucket of ice containing only the same flavor.

Example 1: A single case with ninety-six cans (Partitioned – Single Threaded)

In this case one person (Single Workflow) will have to open the case locate the first flavor, carry all cans to the party-bucket of ice for that flavor, then return to the case and repeat the process for the next flavor.

In this case, a single person (workflow) will have all the resources to themselves. Each different flavor party-bucket will only ever have one person dropping off cans at any given time (single-threaded). We can see that the data is organized and partitioned, but the possibility of parallel processing is not being taken advantage of.

Example 2: Four cases with twenty-four cans (Partitioned – Multi Threaded)

In this case four people (Partitioned Workflows) could all be accessing their own case and moving six cans for each flavor in parallel. You can imagine that this process would go much faster than a single person working on a large case. This is because the work to be done was partitioned so that multiple people could perform the task at the same time.

This seems like a much more efficient process, so why not break the case up further into eight cases of twelve cans and then we could have more people (workflows helping to get the work done twice as fast again). Seems logical, but what about the potential traffic-jam at the party-bucket. If we have eight people all trying to drop their flavored soda at the party-buck containing the ice, some of those people (workflows) would have to wait for the person in front of them to drop of their cans.

This problem illustrates what happens if you try to do too much parallel processing. When processing four cases of twenty for cans, a person would be fifty percent less likely to run into / block another person at the party-bucket. If you switch to eight cases the probability of a collision at the party-bucket is much higher. In this case, the process could become queued, and eight workers could take longer than four.

Conclusion: Partitioned Parallel Loads Increase Throughput and Scale

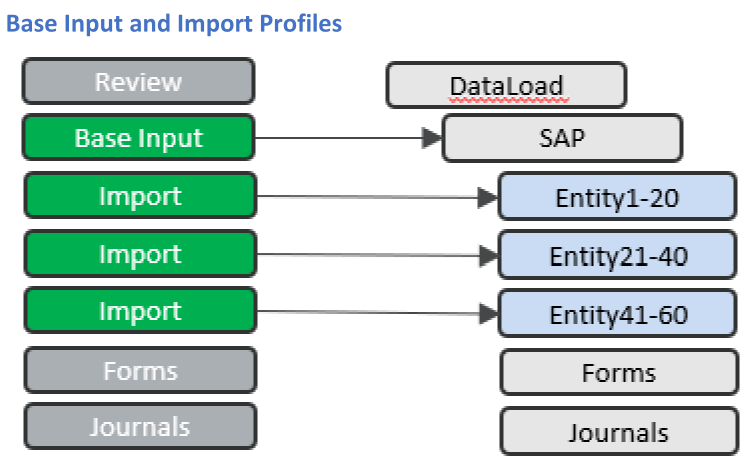
Partitioning and clustering (Cases and Groups of Flavors) enable the possibility for greater throughput, high-volume, and high-performance processing. These concepts are the key to building the most scalable and high-performance data loading implementation withing the OneStream platform. However, there will always be constraints on the system in the form of downstream resource limitations (Party-Bucket in the soda-pop example or a Database Server from a stage processing example). This means that you cannot infinitely partition and parallel process workloads because you will eventually hit a bottleneck in another dependent resource. It is possible to increase the capacity and therefore concurrent throughput by increasing downstream resources (Get a larger Party-Bucket that allows two people to access it at a time or add more database resources to handle greater concurrent processing), but as you can imagine in our example, party-bucket size would have a limit just like database resources have a limit.

WORKFLOW CONFIGURATION FOR STAGE ORCHESTRATION

The OneStream workflow engine supports many distinct types of workflow profiles. The stage engine is orchestrated through the definition and configuration **Import Workflow Profiles**. Import workflow profiles align to the Import channel of the Origin dimension and surface configuration properties that specifically control the stage processing engine.

WORKFLOW PROFILE TYPES USED TO CONTROL THE STAGE ENGINE.

Workflow is used to define behaviors or units of work that must be accomplished within an OneStream application. Workflow structures consist of hierarchical relationships that define dependencies between workflow profiles and layout a sequence of steps that must be executed to complete a task. Individual workflows are referred to as profiles and they contain properties (settings) that determine the specific behaviors that a workflow profile should exhibit when executed. There are many different types of workflows which are documented as part of the core OneStream platform documentation, but for this discussion we are going to concentrate on the two workflow profile types that are used for stage orchestration.



BASEINPUT (INPUT PARENT PROFILE)

- Used to control different channels of input for a workflow and can only have children of the three input types which relate directly to the platform Origin dimension.
    - Origin Channels
        - Import
            - Data Integration via Stage Engine
            - Flat files
            - Direct Integration
            - Web Services
            - Data Management
        - Forms
            - Data entry via CubeViews
            - Data entry via Excel
        - Adjustment
            - Two-Sided accounting data entry via journals
- Defines the entities associates with the workflow.
- Associated Cube

IMPORT (CHILD ASSOCIATED WITH ORIGIN “IMPORT” MEMBER)

- Workflow profile child that defines a Data Integration workflow
- Defines import behavior.
    - Workflow Type
        - Standard (Full Audit)
        - Direct Load (No Detail Audit)
        - Blend (BiBlend Column-Store Index Transactional Analytics)

        - External Consumption using the “Import (Stage Only)”
    - Data Source
    - Transformation Rules
- Defines Constraints and Performance Controls
    - Insert Type
    - Allow Loads to Unassigned Entities
    - Cache Page Size
    - Cache Pages In-Memory Limit

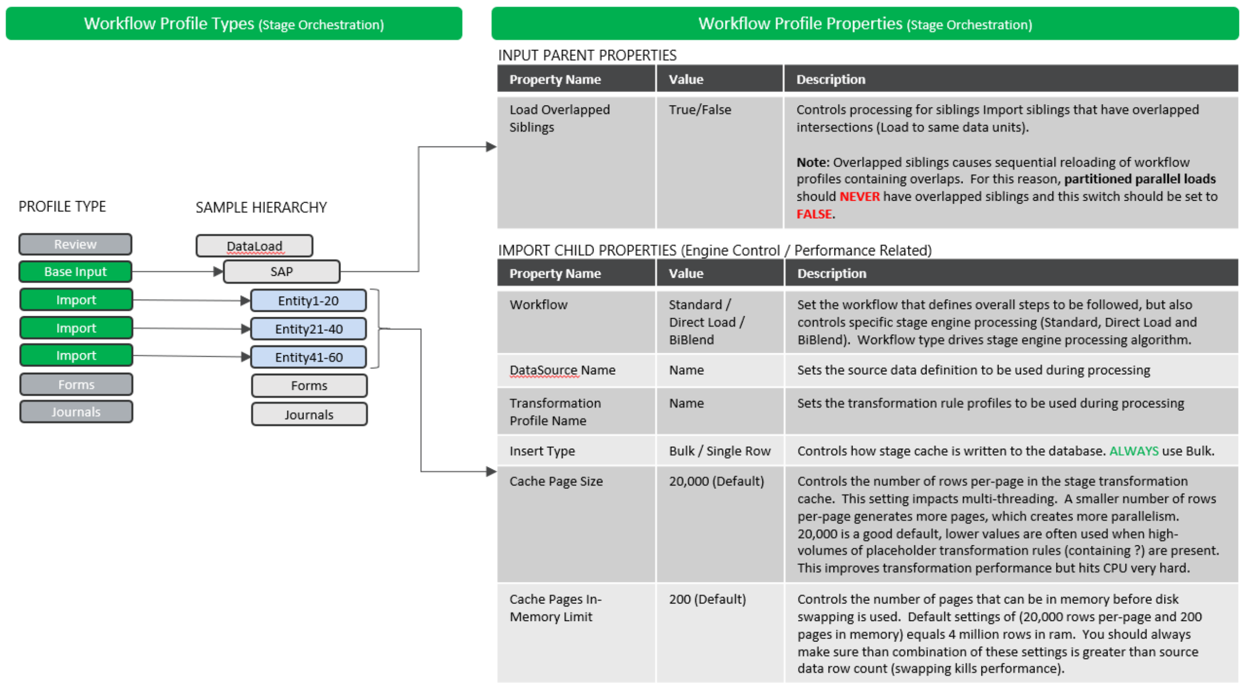
DATA COLLECTION WORKFLOW STRUCTURES

One of the most important tasks that needs to be performed when building an OneStream application, is defining the data collection process. The business processes and source system topography as a customer will dictate the workflow design characteristics required for efficient and transparent data sourcing.

Some customers are highly centralized and have few source systems. In cases like this the customer usually opts for a centralized data collection strategy that runs in a “Lights Out” batch processing mode. End users interact with the system primarily focused on adjusting data and consuming it.

Other customers are highly decentralized and have many different source systems which creates the need for much higher end user involvement and process structure. In this case, the OneStream workflow is used to create an outline the of the business process need to support the collection, processing, and validation of the many sources of data required to create a consistent company level view of information.

WORKFLOW TYPE AND PROPERTY DIAGRAM



INPUT PARENT PROPERTIES

Property Name	Value	Description
Load Overlapped Siblings	True/False	Controls processing for siblings Import siblings that have overlapped intersections (Load to same data units). **Note:** Overlapped siblings causes sequential reloading of workflow profiles containing overlaps. For this reason, **partitioned parallel loads** should **NEVER** have overlapped siblings and this switch should be set to **FALSE**.

IMPORT CHILD PROPERTIES (Engine Control / Performance Related)

Property Name	Value	Description
Workflow	Standard / Direct Load / BiBlend	Set the workflow that defines overall steps to be followed, but also controls specific stage engine processing (Standard, Direct Load and BiBlend). Workflow type drives stage engine processing algorithm.
DataSource Name	Name	Sets the source data definition to be used during processing
Transformation Profile Name	Name	Sets the transformation rule profiles to be used during processing
Insert Type	Bulk / Single Row	Controls how stage cache is written to the database. ALWAYS use Bulk.
Cache Page Size	20,000 (Default)	Controls the number of rows per-page in the stage transformation cache. This setting impacts multi-threading. A smaller number of rows per-page generates more pages, which creates more parallelism. 20,000 is a good default, lower values are often used when high-volumes of placeholder transformation rules (containing ?) are present. This improves transformation performance but hits CPU very hard.
Cache Pages In-Memory Limit	200 (Default)	Controls the number of pages that can be in memory before disk swapping is used. Default settings of (20,000 rows per-page and 200 pages in memory) equals 4 million rows in ram. You should always make sure than combination of these settings is greater than source data row count (swapping kills performance).

DATA COLLECTION TYPES

The following sections detail the best approaches for configuring workflow within application that support decentralized end-user driven data collection processes, centralized “Lights-Out” data collection processes, and hybrid data collection processes that incorporate elements of both.

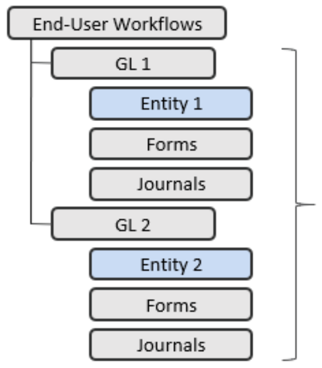
USER GUIDED DATA COLLECTION

End-User guided processing usually involves simultaneous processing of many different small-to-moderate system process jobs. For example, a company may have many different business units and each business unit may have many different source systems. In this case, workflows are typically created for each business unit / system combination and an end user executes the workflow process defined for the system when the data is available.

This type of processing has advantages and disadvantages. First, because the data is decentralized, there is natural data partitioning and system resource dispersion that occurs. This is an advantage from a computing resource perspective. By creating many workflows that are aligned to the many different source systems you end up with a high degree of potential parallel processing potential. Many different users can be reading data from many different systems and writing the data to many different WorkflowUnitClusterPK’s at the same time without a lot of resource contention.

The disadvantage of End-User involvement in complex business processes is the high degree of coordination required across different user groups and the high potential for human error. Decentralized business processes and system topographies usually drive higher end-user focused business processes. The OneStream workflow engine is used to create defined repeatable steps that guiding end-users through complex business process in a transparent and high-quality way.

In the end, building a workflow structure to model decentralize processes creates a foundation for process automation. The OneStream workflow engine enables all workflows that do not require human input to be completely automated.



End User Workflows are **NATURALLY** partitioned

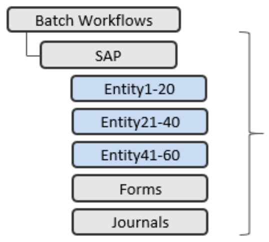
- Workflow typically follows the number of disparate source systems
- End up with many Workflows that have reasonable data volume
- Users efficiently execute different workflows in parallel

LIGHTS-OUT BATCH DATA COLLECTION

Most OneStream implementations involve some degree of Lights-Out batch processing to accomplish automation. Automated batch loading is useful when a data needs to be refreshed on a regular cycle. Batching loading is a valuable tool in the OneStream platform because it enables data to be processed in off-times (overnight) while eliminating the need for human involvement in the process.

OneStream enables lights-out batch processing through the automation of a workflow or by leveraging custom business rules. The most straight forward method to setup automated data loads in OneStream is by using the *Batch File Harvesting* engine in combination with a scheduled *Data Management* task. It is also possible to create completely custom automation routines using stage engine BRApi functions. Workflow definitions are used as the backbone of an automated process in the same way that they are used to control and guide end-user processes. However, when workflows are built to support automation, the workflow design concepts switch from end-user guidance to data partitioning and performance.

For example, a single high-volume data source may be used to load and process millions of rows that will be consumed by many other workflows. In cases like this, you can see how the workflow design would switch from creating a single workflow to load millions of rows to a design that would partition the millions of rows so that they can be loading in a parallel batch.



Batch Workflows must be **INTENTIONALLY** partitioned

- Sourcing data from one or more high volume data sources
- Do **NOT** create a single Workflow with millions of rows
- Create separate workflows with reasonable data volume
- Partitioned workflows can be batched executed in parallel
- Partitioned workflows enable faster error correction (only reprocess impacted partition)

HYBRID DATA COLLECTION

The OneStream platform supports combinations of End-User and Automated batch loading to accommodate about any business process. Workflow structures can be different by cube and scenario type allowing for complete lights-out batch loading while still enabling end-users to follow guided workflows to enrich or complete data not provided by a source system.

FINANCIAL ANALYTIC ENGINE OVERVIEW

Financial intelligence is at the core of the OneStream platform and the Financial Analytic Engine is delivers this capability as an in-memory multi-dimensional computation engine. The primary function of this engine is to enrich source system data with calculations, translations, eliminations, and aggregations. The combination of these features is referred to as consolidation (Financially Intelligent Aggregation). Another very import feature in the OneStream financial analytic engine is called Calc-Status. This feature allows OneStream to create a transparent view of the calculation state of data within the system. Calc-Status seems like and obvious feature for an analytic system, but most products in the market do not have this capability due to the complexity of the computer science problem required to track data changes across trillions of cells.

OneStream is a unique platform in the market because it this proprietary in-memory financial analytics engine. Very few products can perform level of sophisticated financial computations required to produce statutory book-of-record financial statements. Most products in the CPM market are based on standard aggregations engines that simply add numbers together and lack the ability to implement financially intelligent computations.

FINANCIAL INTELLIGENCE

Financial intelligence means that the computational engine has built-in capabilities that enable it to understand the mathematical relationships defined by the double entry accounting standard. In addition, financial intelligence requires the ability to perform specialized aggregations that are aware of financial model constraints and intricacies. Finally, transparency is a critical requirement of financial intelligence, and all computations must be auditable to create confidence in the results produced by the system.

FINANCIAL LOGIC REQUIREMENTS

- Account Type Math (Understand Asset/Liabilities/Equity/Income/Expense relationships)
- Financial Calculation Status and Locking
- Currency Translation
- Partial Ownership Aggregation
- Two-Pass Aggregations
- Scenario and Time Base Hierarchies (Organization-By-Period)
- Node Level Adjustments (Statutory Adjustments Specific to Parent)
- Intercompany Elimination
- Currency Overrides
- Rollforwards / Movements
- Cache Flow Computations (Movement Tracking)
- Variance Computations (Waterfall)
- Financial Validations
- Allocations

IN-MEMORY ANALYTICS

You often hear the term "in-memory," but what does it mean and why is it important? Analytic applications are tasked with processing and enriching large volumes of source data with the expectation the data will be turned into information that can quickly and reliable analyzed by users. The OneStream financial analytic engine delivers information to users by keeping blocks of data (Data Units) in memory and available to for fast access by end users.

DATA UNIT DEFINITION

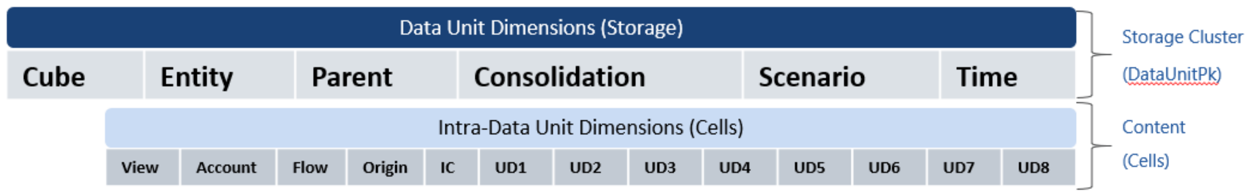
Dimensions of Data Unit Cube – These dimensions define each Data Unit.

- Cube
- Consolidation
- Entity
- Scenario
- Parent
- Time

These six dimensions define the Data Unit, the other twelve dimensions are part of the data unit. The Data Unit consists of the stored data records for the above combination of dimensional intersections. Only base records of the twelve dimensions are stored in the database. Parent members of dimensions within the Data Unit are calculated only in memory, and do not exist in the database. This is not necessarily true for dimension outside the

Data Unit. For example, the entity parent data is stored. When you reference any combination of these Dimensions, a “Data Unit” is created in the server’s memory. The server calculates parent Members of Account, IC, Flow, and User-Defined – dynamically – and generates a small Cube of this data. The greater the size of the Data Unit, the larger the strain placed on the system.

The combination of these six values represents a storage key / cluster for a group of related data cells. The best way to visualize this is think of how it would be presented in Excel. The storage cluster represents the Excel file name or Bucket (Cube, Entity, Parent, Cons, Scenario and Time) and each row of the spreadsheet contain the Data Unit’s stored detailed intersections (View, Account, Flow, Origin, IC, UD1-8).



Designing the Data Unit is the most important part of a useful design. There are a few critical concepts that must be understood to enable architect and designers to build the most flexible and high-performance models possible.

Notice here the dimensions that define the Data Unit are what we would expect to be dense dimensions. Dense dimensions are populated at or close to 100%. We would expect one would load to data to every entity or close to it. The dimensions that are part of the Data Unit are typically what we would call Sparse dimensions. Not only are they not all commonly used for every intersection, but combinations of these dimensions create gaps. You could use all the products for the inventory account, but products for the Deferred Tax Asset does not make sense.

The Data Unit performance is driven by populated intersections, not intersections. If you have five billion combinations, and only will load a few thousand base records, the performance should be extremely fast.

DATA UNIT IMPACT ON SERVER RAM REQUIREMENTS

As the designer of an OneStream analytic model, **your primary consideration regarding model performance** should be Data Unit size. Remember, the system must move Data Units in and out of memory as users request/update cells stored within a Data Unit. To illustrate this point, assume that your analytic design resulted in the creation of many Data Units greater than 1,000,000 rows and a max Data Unit of 5,000,000 rows. Why is this a problem? First anytime a user wants a single cell from the 5,000,000 row Data Unit all 5,000,000 rows must be loaded (If not in memory) just to get that single cell. The estimated amount of memory required to hold that 5,000,000 row Data Unit is (5,000,000 * 3,200 Bytes) is 16GB. This single Data Unit will take 16GB of ram which means that is server will need to allocate and deallocate a substantial number of resources for a single Data Unit which is just one of potentially thousands of Data Units that it is trying to manage. Not to mention the Calc-On-The-Fly-Aggregation that must be performed on the Intra-Data Unit dimensions which can be a huge multiplier cell volume.

DETERMINE THE ENTITY

The Cube will be defined by the purpose and workflow of the data. Scenario will be views of the data for varying periods and rates, for example, Rolling Forecast, Actual Consolidation. Time as a dimension is static, as we all use weeks months and years. So, the one dimension that we have the most control over to define our Data Unit is the Entity.

Choosing the correct dimension to be used as the Entity is particularly important. This selection will define the storage buckets that the system uses when a user wants to interact with a group of cells (Intra-Data Unit Cells). In general, you want to favor more Data Units with a reasonable number of content cells rather than a small number of Data Units with an enormous number of Data Cells. <u>Typically, an application with more smaller Data Units will perform better than a large application with only a few.</u>

Any application that has a Data Unit close to 1 million records will begin to see performance issues. As that number of those records within the given Data Unit increase, the performance will decrease at an increasing rate. You really need to study and manage your model design to make sure that you are building a sustainable model and you should make sure that you have picked the proper dimension to be used as the entity/storage block dimension.

Understanding that Intra-Data Unit dimension use Calc-On-The-Fly-Aggregation for all parent values is especially important. In contrast to the Data Unit Entity dimension which aggregates and stores the parent values of the entity as Data Units (Why it is called the Storage Cluster), all intra-Data Unit dimension only store base values and the OneStream analytic engine must derive the parent values when a user requests a cell that references a parent. This concept is critical to creating a high performance read/write application because end users can change a value within a Data Unit, and all parent values will instantly reflect the correct aggregated values. However, this also places some logical constraints on the size of a Data Unit. If you design a model with massive Data Units, the Calc-On-The-Fly-Aggregation process will begin to slow down because the system will be tasked with creating billions and billions of cells on the fly. For this reason, the logical limits on the max number of rows in a Data Unit is about 1-2 million. This number will flex up/down depending on the number of dimensions that are being used in the model because the UDx dimension have massive impact on the amount of Calc-On-The-Fly-Aggregation that must be performed. If the model is using all UDx dimensions with a large member count in each dimension, the logical max Data Unit size that would yield reliable performance may be less 1 million rows. On the other hand, of only one UDx dimension is being used with a reasonable member of members is being used, then the logical max Data Unit size that would yield satisfactory performance may be closer to 5 million rows.

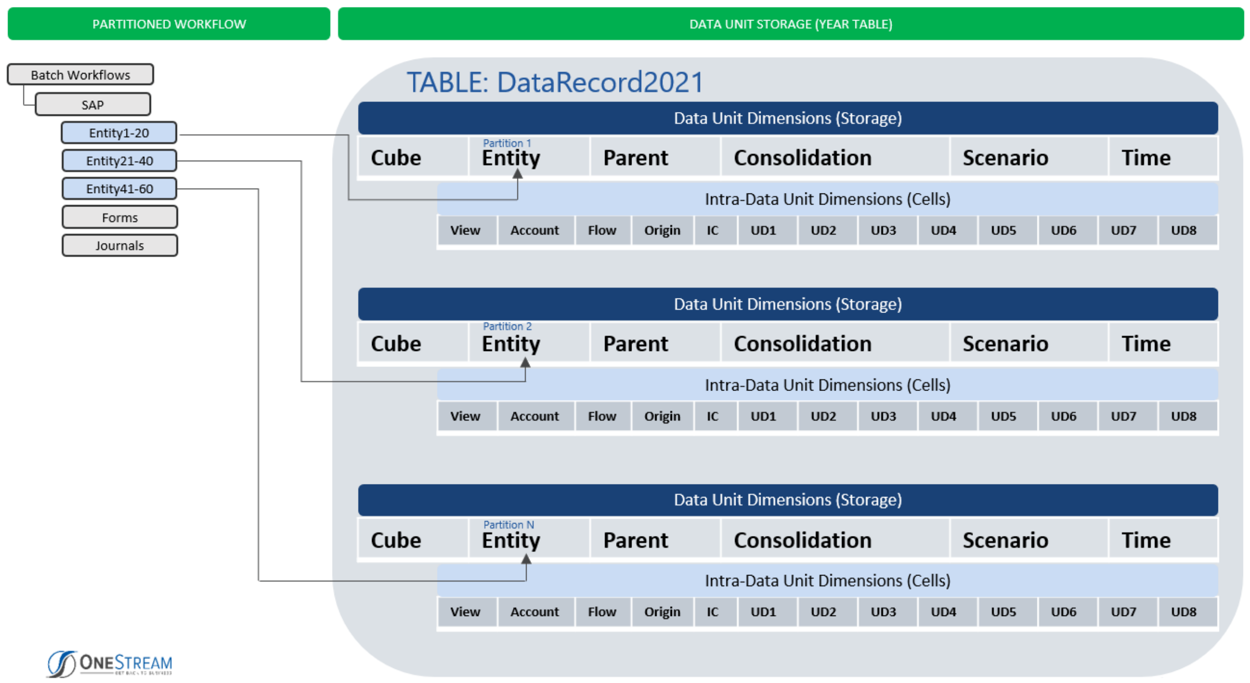
STORAGE

Data Units are physically stored in a relational database and are partitioned by year and entity. This means that data for each year is stored in a separate table and within each year table the data is partitioned by one thousand integers (Entity ID). A Data Unit is managed by a storage key of (Cube, Entity, Parent, Cons, Scenario, Time) that is focused on a single time value (Example: 2021M1). However, Data Unit values for a year are stored together in a single row. Each data record containing Data Unit values is stored in the proper year table which means that the Data Unit storage key is reduced to (Cube, Entity, Parent, Cons, Scenario). Time is not a part of the key because data is stored by year (time bucked) and then each row contains 1–12-time columns for standard monthly storage or one binary storage column which is required to hold variable time columns such as 13 periods or weekly.

Consider this is how you could see a No Data value in your Data Unit. If a value was loaded in January, and not in March the record for the year will still be there. It is something to consider when writing rules.

Entity ID is used as a table partitioning key to enable high throughput during parallel processing. When the financial analytic engine is performing data loads, calculation, aggregation, and consolidations parallel processing by entity is used to get the best performance out of the system. By partitioning the data record tables by entity ID, application server parallel processing throughput is closely aligned with database storage access which permits maximum throughput. Hopefully, the relationship between Stage Workflow Partition and Data Unit loading is obvious. Since the goal of a workflow/stage data loading configuration is to feed data into a cube by loading its

Data Units, you can now see that partitioning stage workflows used for data loading by groups of entities nicely aligns to the Data Unit partitioned storage structure which enables use of parallel processing for data loads.



CELLS VS ROWS

It is important to note that in-memory analytics are based on cells (intersections of dimensions) and not rows. Data Unit cells are organized and stored by time values (Year, then Month or Week). When a Data Unit is saved to its relational storage table the cells are written to a table that contains the year of the cells, next the specific intersection is located (combination of all dimension members: excluding time), finally the appropriate time column value is located, and the cell value is stored. When a Data Unit is requested in-memory (Calculation or Cube View needs a cell of data), the system finds all rows that match the Data Unit definition (Cube, Entity, Parent, Cons, Scenario, Time) and loads all the matching rows into memory. Each row will contain all time members specified for the scenario which enables the analytic engine to turn the row into cells (one cell for each time value).

STORED VS DERIVED DATA

In-memory data represents the cells of information that users have imported, input and/or calculated as stored information. As this stored information is changed and written to the relational database store the in-memory version of the data is refreshed and synchronized across all application servers that are in use to deliver information to the users that are working with the data. It is very important to understand the stored data that is loaded into memory only represents a fraction of the potential cells that OneStream financial analytic engine delivers can deliver to the end user. This means that many of the cells that a user requests are derived at the time they are requested using **Calc-On-The-Fly-Aggregation**. This is a critical concept that must be understood to when designing OneStream analytic applications. Without the combination of in-memory data combined with **Calc-On-The-Fly-Aggregation** it would be impossible for a financial analytic application to perform.

Multi-Dimensional applications are based on a combinations of metadata structures called dimensions. The combination of dimensions is referred to as a cube. The potential number of cells in any given cube is the factorial of all the members in each dimension (200 Entities * 1000 Accounts * 100 Cost Centers = 20 million Cells). As you can see, factorial math is a powerful force, and a tiny three-dimensional model can quickly reach 20 million cells. A standard cube in OneStream contains 18 dimensions which means even a cube with a small number of members in each dimension will potentially reach billions or trillions of cells. With this potential volume of data, it is not possible to store all combinations of cells and achieve the performance levels that users expect from an analytic application. Consequently, the OneStream financial analytic engine uses a combination of stored and in-memory cells to deliver information.

INTRA DATA UNIT DIMENSIONS: CALC-ON-THE-FLY-AGGREGATION

If a Data Unit (Cube, Entity, Parent, Cons, Scenario, Time) has 1 million base rows (stored) with a monthly frequency, there will be 12 million base cells for the year represented as 12 Data Units each containing one million cells for a specific time value. In this case the system is managing 12 million stored cells but keep this only represent a small portion of the potential cells defined by the analytic model because many cells are derived by the analytic application at reporting time using **Calc-On-Fly-Aggregation**.

To illustrate how easily **in-memory data volumes** can increase, assume the dimension outline for our 1 million cell Data Unit mirrors the cube structure used in the GolfStream Demo application. This would mean that account, flow, IC and 5 customer dimensions are in use in use and the parent combinations in these dimensions are not stored. This means that the analytic engine must close the gap in a computational sense between the stored intersections and the potential Calc-On-The-Fly intersections defined by the intra Data Unit dimensions (Account, Origin, Flow, IC, UD1-UD8). It is important to note that from a data consumption standpoint, the OneStream analytic engine automatically determines how to deliver a cell value. If the number is stored it can just be returned from memory. If is derived, then it needs to be manufactured on the fly by creating the cross-factor of all Intra-Data Unit dimension base and parent values.

The best way to visualize this process is by walking through an example. Using the demo application, we can observe how much stored Data Unit volume explodes when derived Calc-On-The-Fly values are considered. The BiBlend engine is useful for this analysis because is stores all dimension combinations which in turn lets us evaluate the difference between **Stored Aggregation** and **Calc-On-The-Fly Aggregation**. By processing the Houston data through the BiBlend engine we can see that 900 base rows stored in the Houston Data Unit explode to about 1.2 million after all parent combinations are aggregated and stored in the BiBlend column-store index table. This represents an explosion factor of 1,400 times the base stored cells (1.2 million / 900 base stored cells). Now we see the power and the computational challenge of Calc-On-The-Fly Aggregation.

In the analytic engine, only 900 rows exist in the sample Houston Data Unit, but the system can deliver 1.2 million cells on demand when the user asks for a number. The analytic engine only stores 900 rows during the load process, which is efficient in terms of storage, but it must derive a **HUGE** number of cells at read time to deliver all possible combinations of the Intra-Data Unit dimensions. If we apply the sample 1,400x explosion factor observed for Houston, to our 1 million cell Data Unit example, we can see that the number of cells that must be derived grows exponentially (1 million x 1,400 = 1.4 billion Cells). As mentioned, Data Unit overview section, the size of the Data Unit and the combination of Intra-Data Unit dimensions is important to the performance of the Calc-On-Fly-Aggregation performance. When a Data Unit grows beyond 2 million cells the Intra-Data Unit dimension explosions factor will likely push the derived cell burden to a level that does not perform well. Potentially billions of cells would need to be derived on demand at read time.

USAGE

With this understanding of the Data Unit storage key and content structure we can gain some insight into how the OneStream in-memory analytic engine works with blocks of data (Data Units). Many of us have encountered a massive spreadsheet (large XLSX with formulas and more than a million rows in the spreadsheet) and just opening the spreadsheet on a local computer takes a long time. In addition, you typically see formulas and recalculations slow down dramatically. A typical OneStream application contains thousands of Data Units that must be managed (Created, Update, Delete, Read) within the computational resource constraints of the servers running the application and the physics boundaries of factorial math.

The more dimensions that are used within an OneStream application the easier it is to create large Data Units when parents are consolidated or aggregate. Consider that when this Data Unit is loaded into memory to create the multidimensional view of the data there is tremendous amount of processing that needs to happen. When we look at data in a report, we are looking at it is using 'by' dimensions. Accounts 'by' Cost Center 'by' Location 'by' and so on… Each of those subtotals by each dimension is calculated with a complex proprietary process. Each added dimension and increase of volume of data yields a model that will impact performance.

Another concern is that one of the easiest ways to create a data explosion that results in large parent Data Units is to create a model that has a content cell dimension (Intra-Data Unit Dimension) that is unique to the entity at the base level. For example, if customer dimension is used, and customers are unique for each entity in the Data Unit storage key, then each parent entity that is consolidated or aggregated with have all unique customer data below the parent in its Data Unit. This type of design always results in large Data Units and usually means that the wrong dimension is being used as the entity which controls the storage block.

DATA UNIT EXPLOSION

Multidimensional analytic models are based on the combinations of member values in structures called dimensions. In OneStream, these structures are called cubes and cubes contain many Data Unit which hold the detailed cells that result from the intersection of members in the dimensions that belong to the cube (Think of cubes as an outline structure that defines the dimensions structures to combine).

What is Data Unit Explosion and why should an OneStream architect care about this phenomenon? Data Unit explosion describes a situation where a high number of potential cells in a cube become populated. You are thinking to yourself, the main the reason that we are creating a cube is to fill it with data. However, you must always be respectful of the power of factorial math. As soon as you design a cube that utilizes multiple members in two or three customer dimensions, the potential number of cell combinations that could contain data within a given Data Unit will easily hit the billions and it is not uncommon to see potential cell counts in the trillions.

EXAMPLE: SIMPLE FIVE DIMENSION CUBE

1. 100 Accounts
2. 100 Flow Members
3. 100 UD1 Members
4. 100 UD2 Members
5. 100 UD3 Members

10 billion Potential Cells Per Data Unit (*Factorial: 100 x 100 x 100 x 100 x 100)*

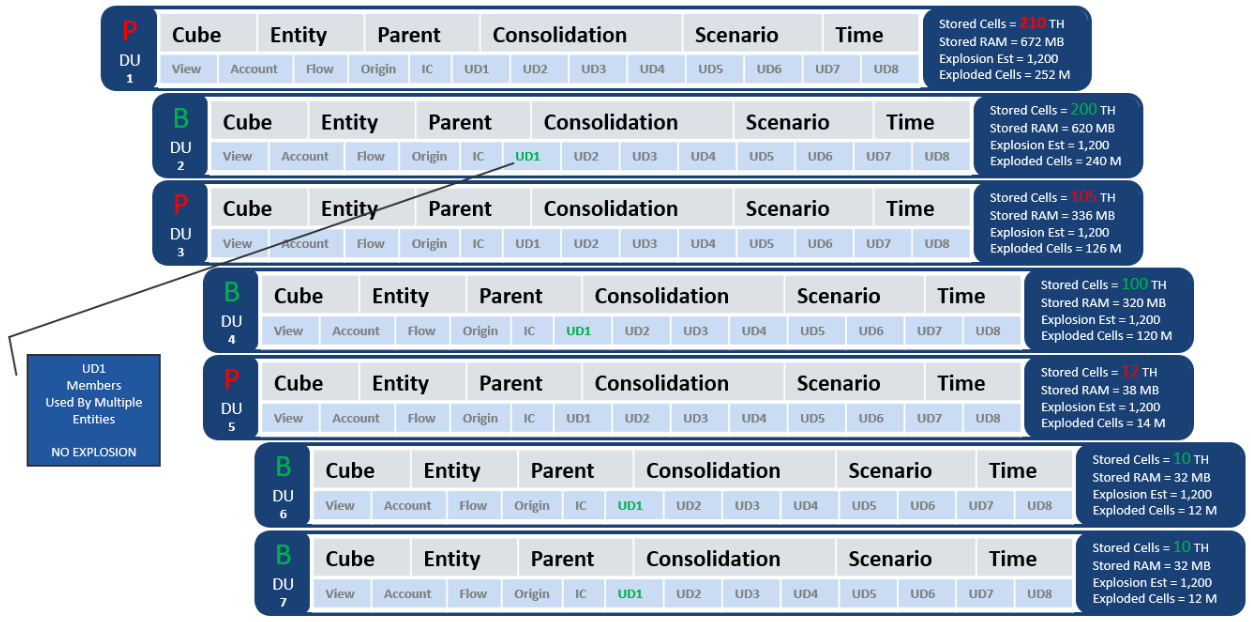
The simple example above demonstrates how easy it is to create substantial number of potential cells in each Data Unit. This example only combines five dimensions with 100 members each and the potential cell count for each Data Unit in the cube is ten billion cells. Also remember that each cube holds many Data Units, so the overall model always hits trillions of potential cells when you add up all potential cells across all Data Units. Hopefully, this tiny cube structure demonstrates just how quickly potential cells add up because of factorial math. Having many potential cells is not a problem, in fact, every practical OneStream model has a large number of potential cells in each Data Unit. The OneStream analytic engine is built to handle very large models and it is very efficient at finding the cells within a model that have data. The ratio of cells that have data to potential cells in the Data Unit is used to calculate the data density ratio (Store Cells / Potential Cells). In most cases, the ratio of cells containing data to potential cells is low. Because of this fact, the OneStream engine must be exceptionally good and finding cells that have data with a vast population of potential cells (Spare Data Filtering).

On the other hand, Data Unit explosion refers to the situation where many potential cells are loaded with data. This always results in a model performance problem. Why? Is the OneStream engine not able to scale? The answer is physics limitations. OneStream supports the largest possible model configurations in the analytic software market (18 Dimensions Per Data Unit), but at a certain point data cell volume will overcome the performance capabilities of the fastest multi-thread CPU's and algorithms. Therefore, it is important to understand how model design and data density impact the performance of your design.

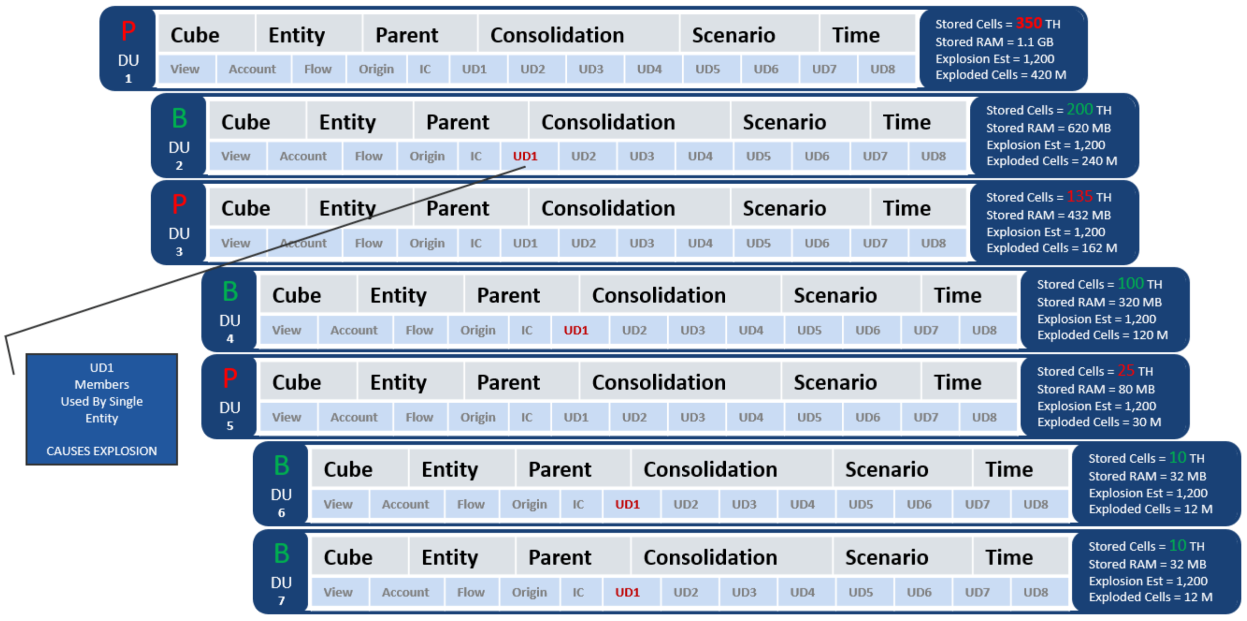
Now that we have a foundational understanding of Data Unit explosion, let us examine the root causes of this phenomenon. There are three primary causes of Data Unit explosion.

1. Incorrect Dimension selected as ENTITY dimension resulting in too many dimension combinations within the intra-dimension members. This situation usually results in many potential cells in a single Data Unit that is then directly loaded from a source system which results in excessive data cell population and inferior performance.
2. Formula spray is another cause of data explosion. This situation occurs when a rule writer erroneously creates a looping construct that iterates of many potential cells in one or more Data Units and populates the cells with data creating high data density. This type of data explosion is kept to a minimum by the OneStream engine due to formula expression constraints, but it is still a potential problem in situations where large allocations rules are being used. Allocation rules tend to spray data to many cells and in many circumstances, they spray insignificant "Near Zero" values to many cells.
3. Entity-To-UDx Distinct relationships are another cause of data explosion at parent Data Units. If any UD member is only value for a single base entity, then there is no collapsing of data rows during the consolidation/aggregation process. This means that each row of a base entity that is added to a parent will become a unique row in the parent. A typical example of data explosion cause by Entity-To-UDx distinct relationships is found when a company uses different cost centers for each legal entity. This situation results in all parent entities containing all cost center combinations of all base members below the parent. This creates massive Data Units at the parent level and should be avoided by creating common UDx members that represent a summary of the distinct items (Grouping). The distinct UDx members should be moved to an entity dimension and a separate model should be built to update and analyze the detail items (This is the advantage of OneStream shared metadata/ multi-model capabilities). The details can then be pulled into the primary cube at the summary level thereby preventing data explosion.

Data Unit Math: Entities Common UDx Members



Data Unit Math: Entities Distinct UDx Members



DATA UNITS IN ACTION (ANALYTIC PROCESSING)

Now that we have established an understanding of Data Units, we have a foundation to understand what drives analytic processing. Based on our definition of a data unit, we can see that there is the potential to hold millions of cells in memory as cached data units and depending how the application is setup there are many ways that this data can easily explode or multiply.

Data Units represent a logical unit of work that the analytic engine can operate on. Most of the control mechanisms and patterns in the analytic engine focus on managing Data Units. Calculations, Cube Views, and any other platform areas that touch a data cell involve interaction with a Data Unit. The analytic engine uses Data Units to cache data that is most frequently operated on by a user (Consolidation, Form Entry, GetCell, etc.). This is how the system can respond user reporting needs quickly (the data is in-memory and available). However, this creates an obvious problem with respect to system computational resources.

OneStream applications can be large and contain thousands of Data Units (you can think of a Data Unit as a spreadsheet containing data for Cube, Entity, Parent, Cons, Scenario, Time). In most cases, there is not enough memory available on a server to hold all available Data Units in memory at one time. OneStream has a sophisticated memory management mechanism that swaps Data Units in an out of memory according to server memory limits and Data Unit request / usage patterns (Oldest / least used Data Units are discarded first when a memory limit is encountered).

Since a Data Unit must be loaded / unloaded from memory on a regular basis, it is important to mindful of the number of stored cells in the Data Unit. Each row of information containing the columns of time values that make up a Data Unit is estimated to be 3,200 bytes in size for monthly data and much more for weekly data. Using this value, we can get a clue as to how much memory will be consumed when a specific Data Unit is loaded into memory. For example, if a monthly Data Unit has 900 stored cell (900 rows with twelve columns) the memory footprint required to load the rows for the Data Unit would be 2,880,000 Bytes or 2.8 MB (900 x 3,200 Bytes/Row). This is a small Data Unit example based on the GolfStream demo application and the sample Data Unit, [Houston, Actual, 2018M3].

Limitation on the maximum number of rows in a Data Unit as evolved over time. In the first generation of in-memory applications (16-Bit) 10,000 stored rows with only 4 dimensions was a logical limit. The second generation of in-memory applications (32-Bit) stored 100,000 rows with 8 dimensions was a logical limit. The third generation (64-Bit) of in-memory applications with 18 dimensions pushed the logical limit to around 2,000,000-5,000,000 stored rows depending on how many custom dimensions are in use and how complex the hierarchies are in those dimensions. OneStream's in-memory analytic engine does not have hard limits but reporting performance for Calc-On-The-Fly-Aggregation (discussed in detail later) and server memory eventually become natural constraints as Data Unit row volume grows.

Repeating our memory estimation formula from above for a 2,000,000 cell Data Unit, the application server would be required to load 6,400,000,000 Bytes or 6.4 GB (2,000,000 x 3,200 Bytes/Row) into memory for a single Data Unit. Also, keep in mind that the memory footprint described here only accounts for stored cells. OneStream's analytic engine can derive many times this number of cells using dynamic calculations and Calc-On-The-Fly-Aggregation. Large dimensional models can easily have a Calc-On-The-Fly-Aggregation multiplier more than 1,000 times the number of stored cells, which would turn our example of 2,000,000 cells stored into 2,000,000,000 (2 billion) potential cells in memory.

STORED AGGREGATION (BIBLEND ENGINE)

We can see that **Calc-On-Fly-Aggregation** is a powerful computational and analytic feature of the analytic engine. Why would we ever want or need to use BiBlend and go through the trouble of computing and storing all Intra-Data Unit combinations at aggregation time? This takes a lot of memory, computational resources, and storage, why not just use an analytic cube with Calc-On-The-Fly-Aggregation?

The answer is simple, you cannot out-compute factorial math. If we upsize our example a sample Data Unit (Cube, Entity, Parent, Cons, Scenario, Time) to 2 million cells, the exploded cells that need created during **Calc-On-Fly-Aggregation** becomes 2.8 billion (2 million cells * 1,400 = 2.8 billion). This means that we would only load and store 2 million cells, but the system would be expected to calc 2.7+ billion cells on the fly (monumental challenge for even the fastest CPU). This number of derived cells is large and computationally expensive to produce. If a single Data Unit gets too large and the Intra-Data Unit dimensions have a lot of members / parents, the cost of **Calc-On-Fly-Aggregation** will become too high to yield good reporting performance. Consequently, **Calc-On-Fly-Aggregation** performance will start to diminish after 2 million base rows in a single Data Unit. **Calc-On-Fly-Aggregation** is a critical capability when data is changing frequently because the calc on the fly numbers are always correct and derived when asked for, but this can take a long time when you are talking about 2.7 billion cells.

To combat this, the BiBlend engine does the work to calculate and stores all parent combinations which makes the data available in a relational table for immediate reporting but at the cost of having to calculate all combinations a head of time. This means that you trade off having an expensive process at read time (Calc-On-Fly-Aggregation) for an expensive one-time preparation cost of creating and writing all cell combinations. The other tradeoff is the lack of real-time parent totals when a base value changes. With BiBlend if a base value changes, all data must be re-aggregated so that the parent values reflect the proper totals. BiBlend is useful when data is being used in a snapshot mode and values can be calculated and used for a period and then batch updated to reflect new data (Nightly operational data loads or select aggregations of cube data against high-volume Data Units).

BiBlend is valuable as a reporting option if Data Units get too large to yield the desired reporting performance. Stored aggregation with BiBlend requires a targeted data mindset vs a broad data mindset. This means BiBlend should be used to compute a subset of the potential cells and store them for fast relational reporting or dashboard analytics. With BiBlend, the idea is to only compute and store the cells that you need to report on and not all combinations. Again, the potential number of cells for a single Data Unit can reach billions of cells and a single cube may have thousands of Data Units. Consequently, BiBlend should be used as a target stored aggregation tool, and you do NOT want to attempt to compute all potential cells (could be trillions of cells).

CASE STUDY ON DATA UNITS

As an example, a recent customer use-case demonstrated how important it is to select the proper entity dimension. In this example, the customer, was attempting to replace an existing high-volume analytic solution but it was decided during design to use “Legal Entity” as the entity dimension in OneStream which only contained 30 members and this dimension did NOT exist in the model that was being replaced (**RED FLAG**). Once the model was built and the data was loaded it was discovered that the top Data Unit had more that 20 million rows (Required 90GB of RAM to load that single Data Unit). This is a classic example of choosing the wrong dimension for the Entity which controls the storage block and on top of that including a dimension in the OneStream model that did not exist in the model being replaced. Any time there is a difference in dimensionality between two models, there is going to be a dramatic difference in cells/potential data which makes it impossible to compare models. A single additional dimension with 3 members would create a 3x multiplier on the amount of data that OneStream would need to process relative to the legacy system.

The model in the use-case above was refactored to eliminate “Legal Entity” from the model and to move one the UDx dimensions into the Data Unit storage key as the entity. The UDx dimension that was chosen had two characteristics that made it the appropriate dimension to be used as the Entity. First, it aligned to the way that users would want to access the data. This is important because the system will perform better if the storage cluster is aligned with the primary way the users request data. When Data Units are aligned to the primary end-user analysis approach, the engine can find and load cells efficiently without loading a lot of extraneous data to fulfill the request. For this use-case, it turns out that legal entity was not used at all, and the users always looked for data based on UDx values. This Second, the UDx dimension that was chosen had a much higher count of members (20,000 members) which dramatically reduced the top Data Unit row count to a value well within the best practice limits of 1-2 Million rows. Having many smaller Data Units means the analytic engine can move many smaller Data Unit's in/out of memory much more efficiently then dealing with a small number of massive Data Units.

FINANCIAL ANALYTIC ENGINE PROCESSING

OneStream allows you to create a formula on a dimension member or in a Finance Business Rule file.

MEMBER FORMULAS

Member formulas are written using the Formula property on individual Scenario, Account, Flow, or UD members in the Dimension Library page. The primary reasons for writing formulas on dimension members are:

- Promotes reusability of dimensions and members with their associated calculations across multiple cubes.
- Enables parallel processing for performance optimizations using advanced multi-threading that executes multiple formulas at the same time.
- Provides intuitive formula organization.
- Supports Drill Down from a calculated amount to the amounts that were used as sources for the calculation.
- Supports the ability to vary formulas by Scenario and Time so that changes can be made without affecting the calculations for older data or data in other Scenarios.

FINANCE BUSINESS RULES

Formulas can also be written using one or more Business Rule files. The primary reasons for writing formulas in a Finance Business Rule are:

- The formula requires extensive cross-dimensional dependencies, and it is not clear which member an equivalent Member Formula should be written on.
- The formula requires complex sequential logic, variables, or conditional statements that affect multiple dependent calculations.
- Your application requires custom algorithms for currency Translation, Share, or Intercompany Eliminations. Note: Due to OneStream's sophisticated built-in translation and consolidation algorithms, most applications only require member formulas for the Data Unit Calculation Sequence (DUCS) (a.k.a., Chart Logic). Custom Business Rules for Translation, Share, and Intercompany Eliminations are not typically needed.
- Custom Calculate generally for Planning calculation execution.

There is not a preference of using Member Formulas over Business Rules. While using Member Formulas can multithread the process as all rules is calculated that the same time, threads are also needed for consolidation within the Entity dimension.

WHEN DO FORMULAS EXECUTE?

This section defines the types of calculations and calculation sequences that are executed during the different analytic processing routines used by OneStream.

CUBE PROCESSING:

1) Calculation (a.k.a., Chart Logic)

    a. Executes the standard calculation sequence for a single Data Unit. A Data Unit refers to a group of data cells for a specific Cube, Entity, Parent, Consolidation, Scenario, and Time member. Except for Dynamic Cell Calculations (described elsewhere), all Member Formulas are written to execute as part of a Data Unit's Calculation Sequence.

2) Translation
    a. Executes a currency translation that occurs when the data for an entity's local currency needs to be translated to a foreign currency. The translation step executes after the system has run the Data Unit Calculation Sequence for an entity's local currency. Then OneStream's default translation algorithms use Foreign Exchange (FX) rates to generate and store a corresponding translated Data Unit. Finally, OneStream executes the Data Unit Calculation Sequence on the translated Data Unit to produce the final translated amounts.
3) Consolidation
    a. OneStream 's analytic engine provides pre-built financial intelligence through a statutory Consolidation dimension that defines a sequence of Data Unit calculations and aggregations which include currency Translations, parent-level Adjustments, complex Ownership computations, and Intercompany Eliminations.
4) Dynamic Cell Calculation
    a. Dynamic Cell Calculations are a special type of Member Formula for Account, Flow, or UD members. They are used to generate amounts for a member on-the-fly, i.e., the results are calculated on-demand and are not stored. Dynamic Cell Calculations are often used for metric accounts (e.g., ratios involving other accounts) and are appropriate when the result of the dynamic calculation is not needed as the source number for another stored calculation. The use of Dynamic Cell Calculations can result in improved consolidation performance because they do not generate stored numbers and are typically only executed when a number needs to be displayed, not during the consolidation process.

CALCULATION SEQUENCE OF A SINGLE DATA UNIT:

The items below detail the specific list of tasks that are executed for each Data Unit's calculation process. As an example, all the following steps are executed for a single Data Unit when a user selects "Calculate" for a single Entity, Scenario, and Time period.

DATA UNIT CALCULATION SEQUENCE (DUCS)

1. Clear previously calculated data for the Data Unit
    - If Using Hybrid Scenarios – the copy from the scenario will run here.
2. Run the Scenario's Member Formula (typically used for seeding a Scenario's data from another Scenario or from a prior year).
3. Run reverse translations by calculating Flow members from other "Alternate Currency Input" Flow members. This is part of OneStream's built-in ability for an entity to accept input using multiple currencies.
4. Execute Finance Business Rules (1 and 2). Up to 8 Business Rule files can be attached to each Cube.
5. Run Formula Passes (1 – 4) for the cube's Account dimension members, then Flow members, and then UD members. The Formula Pass is specified using each member's Formula Type property in the Dimension Library page.
    - Account, Flow, UD1, UD2, … UD8 (*Member Formula Execution*)
6. Execute Finance Business Rules (3 and 4)

7. Run Formula Passes (5 – 8)
    - Account, Flow, UD1, UD2, … UD8 (*Member Formula Execution*)
8. Execute Finance Business Rules (5 and 6)
9. Run Formula Passes (9 – 12)
    - Account, Flow, UD1, UD2, … UD8 (*Member Formula Execution*)
10. Execute Finance Business Rules (7 and 8)
11. Run Formula Passes (13 – 16)
    - Account, Flow, UD1, UD2, … UD8 (*Member Formula Execution*)

STATUTORY CONSOLIDATION SEQUENCE:

OneStream's consolidation process is executed recursively for a hierarchy of entities. For each entity, a series of Data Unit Calculation Sequences are executed using multiple Consolidation members. While some of the Consolidation members are associated only with the entity (e.g., the entity's local currency Consolidation member), many of the Consolidation members are only valid for the Relationship between the entity and its parent entity (note: if an entity has multiple parents, each parent-child combination is a separate Relationship). Therefore, the consolidation process for a single child entity and parent involves multiple Data Units. As described earlier, a Data Unit is defined as a specific Entity, Parent, Consolidation, Scenario, and Time member. Since an entity hierarchy contains many entities, OneStream's consolidation process executes multiple Data Unit Calculation Sequences for every parent-child Relationship (typically thousands). The specific processing logic applied to each Consolidation dimension member is explained below.

TOP - Contributes to the parent Entity's *LOCAL* member

Sum (SHARE + ELIMINATION + OWNERPOSTADJ)

OWNERPOSTADJ (*Execute DUCS, FinanceFunctionType = Calculate)*

Stores Journal entries that are needed after the Share and Elimination calculations have been performed for an entity and parent.

ELIMINATION (*Execute DUCS, FinanceFunctionType = Calculate)*

*Elimination algorithms (**Use default or implement FinanceFunctionType.ConsolidateElimination in a BR)***

Start with SHARE Data Unit and eliminate data cells where IC is not None. If a common parent entity was reached for both the entity and the IC entity, generate an offset amount and a plug Account amount.

SHARE (*Execute DUCS, FinanceFunctionType = Calculate, only if not using default calc-on-the-fly)*

Default Share is calculated on-the-fly

Sum (TRANSLATED + OWNERPREADJ) * % Consolidation

*Share algorithms (**Use default or stored or implement FinanceFunctionType.ConsolidateShare in a BR)***

OWNERPREADJ (*Execute DUCS, FinanceFunctionType = Calculate)*

Stores Journal entries that are needed before the Share and Elimination calculations have been performed. This is a Relationship-level member, so the data is for a specific parent entity.

TRANSLATED (*Execute DUCS, FinanceFunctionType = Calculate)*

The "Translated" member is a shortcut to the Consolidation member that matches the parent entity's default currency (e.g., USD). It stores data that was generated by multiplying local currency data by FX rates, and then executing the DUCS.

*Translation algorithms (**Use default or implement FinanceFunctionType.Translate in a BR)***

Determine FX Rates and multiply to create a translated Data Unit from the local currency's Data Unit.

LOCAL ***(Execute DUCS, FinanceFunctionType = Calculate)***

	The “Local” member is a shortcut to the Consolidation member for the entity’s default currency (e.g., EUR). It stores input data in base entities, or data that was consolidated by combining data from the TOP consolidation member from lower-level child entities.

The diagram above illustrates the Consolidation dimension members that are involved when OneStream executes the consolidation process for each entity and parent entity Relationship.

Starting from the bottom of the diagram, OneStream first runs the Data Unit Calculation Sequence for the Consolidation member that represents the child entity’s local currency. This first step is equivalent to selecting a single entity and running the Calculate process. Review the full list of Consolidation members by selecting the Consolidation dimension from OneStream’s Point of View pane. OneStream contains approximately 200 built-in currencies corresponding to all of the known currencies in the world plus some custom currencies. A subset of those currencies, based on the ones that are enabled in your Application Properties, are listed under the tree item labeled “Currencies.” For example, your application may contain Consolidation dimension members for USD, EUR, GBP, and CAD. Each entity has a Currency property (specified in the Dimension Library) that indicates which of those currencies should be used as the entity’s Local currency. Therefore, the Local Consolidation member is potentially different for each entity. “Local” is effectively a shortcut that refers to the appropriate currency member that represents the entity’s Currency setting.

After the Data Unit Calculation Sequence has been executed for the child entity’s Local currency, the entity’s local currency numbers need to be translated if the parent entity has a different local currency. Like the “Local” consolidation member, the “Translated” consolidation member is a shortcut to the currency that represents that currently selected parent entity’s default currency setting. If the child entity and parent entity use the same default currency, then no translation is needed during the consolidation process. If translation is required, OneStream’s default translation algorithm starts with data in the child entity’s Data Unit for its Local currency, then it looks up the appropriate Foreign Exchange (FX) rates based on the source and destination currencies, and it creates a corresponding Data Unit for the entity’s foreign currency Consolidation member. If desired, you can use a Cube setting to augment or replace OneStream’s default translation by writing a Business Rule that determines FX rates differently than OneSTream’s default process and/or perform custom processing to generate a translated Data Unit from a local Data Unit. After the translated Data Unit has been created, the Data Unit Calculation Sequence is executed. Therefore, OneStream provides the ability to write custom Business Rules to control how local currency numbers are converted to foreign currency numbers, and then after the initial set of foreign currency numbers have been generated, it executes the DUCS (i.e., it runs the Member Formulas) to generate additional calculated numbers from the base set of translated numbers.

The OwnerPreAdj Consolidation member is for Journal adjustments that are needed before the Share and Elimination algorithms are executed for an entity and a specific parent entity. The Data Unit Calculation Sequence is executed during the consolidation process. A Journal entry for OwnerPreAdj is different than a Journal adjustment using a foreign currency Consolidation member (e.g., USD, EUR) because OwnerPreAdj applies only to a specific parent entity while an adjustment to a currency Consolidation member would affect the results for all the entity’s parents that used that currency as its Local Consolidation member.

The Share Consolidation member is the sum of the Translated and OwnerPreAdj Consolidation members multiplied by the Percent Consolidation that was specified for the parent-child entity Relationship. For performance optimization, OneStream calculates the Share Data Unit on-the fly (not stored). However, Cube settings can be changed to store the data for Share, or to implement a Business Rule to generate custom amounts for Share.

The Data Unit for the Elimination Consolidation member is generated and stored using OneStream's built-in Intercompany Elimination algorithms. The process starts with the Share Data Unit and eliminates data cells where IC is not None. If a common parent entity was reached for both the entity and the IC entity, the algorithm generates an offset amount and a plug Account amount. Finally, the Data Unit Calculation Sequence is executed for the Elimination member and the results are stored. Alternatively, Cube settings can be changed to implement a Business Rule to generate custom amounts for Elimination.

The OwnerPostAdj Consolidation member is for Journal adjustments that are needed after the Share and Elimination algorithms are executed for an entity and a specific parent entity. The Data Unit Calculation Sequence is executed during the consolidation process.

The Top Consolidation member is the sum of the Share plus Elimination plus OwnerPostAdj Consolidation members. It is calculated on-the-fly (not stored) and it represents the final amounts that a child entity contributes to the parent entity.

The Data Unit for a parent entity's Local Consolidation member is generated by summing the Data Units from the Top Consolidation member for all the parent's immediate child entities, and then executing the Data Unit Calculation Sequence.

CALCULATIONS

DYNAMIC CELL CALCULATION (A.K.A., DYNAMIC CALC)

A *Dynamic Cell Calculation* is an in-memory calculation that is executed on demand when a cell containing a dynamic member formula is requested. A Dynamic Calc formula computes a value for a single cell. It is run every time the cell needs to be displayed and the result is not stored. Dynamic Cell Calculations provide a performance benefit for the consolidation process because the amount is calculated when requested for display, and it does not have to be written to the database. However, for reporting there is a performance consideration because data is calculated on demand. The most common usages of Dynamic Calcs are for ratio or percentage calculations.

DYNAMIC CELL CALCULATIONS

- Can reference other Dynamic Cell Calculations.
- Can reference Stored Calculations.
- Do not naturally aggregate parent members
  - If a parent member in an Account, Flow, or UD dimension has a child member that is calculated using a Dynamic Calculation formula, the parent member will not automatically include that child in its aggregated amount.
  - Aggregation can be achieved by writing another Dynamic Calculation on the parent member to perform the aggregation logic.
- Should <u>NOT</u> be used by other stored calculations.

STORED CALCULATION

Consolidation performance is directly impacted by the volume and complexity of stored calculations. Careful consideration should be given to each stored calculation since a single poorly written rule can result in a large amount of data being written to the cube which would negatively impact consolidation performance. If a lot of Member Formulas are used or if data volumes are not being considered when writing Member Formulas, it is easy to generate over 100,000 stored numbers (or more) from just 1,000 initially loaded numbers. The quantity of stored numbers is the most crucial factor when optimizing consolidation performance.

STORED CALCULATIONS

- Can reference other stored calculations.
    - Parent members aggregate naturally.

FINANCIAL ANALYTIC ENGINE DATA STRUCTURES

The financial analytic engine uses a combination of in-memory processing and relational database storage to deliver the analytic processing capabilities to the OneStream platform. From a relational storage perspective, the finance engine stores metadata that defines cubes and dimension hierarchies, and Data Unit intersections are also stored relationally. The finance engine uses a sophisticated in-memory representation(cache) of both metadata and data to deliver fast access the trillions of cells maintained by the engine.

The list below details the relational tables that are used to store finance analytic engine metadata and data. Please note **ALL** financial analytic table are used exclusively by the financial analytic engine and should **NOT** be queried directly by third party processes or business rules. All access to data stored in these table should be through API calls only.

Metadata Tables

Table Name	Description
Dim	Contains Dimension Definitions
Member	Contains Dimension Member Definitions
MemberDescription	Contains Member Alternate Descriptions
MemberProperty	Contains Member Detail Properties
Relationship	Contains Relationship Definitions
RelationshipProperties	Contains Relationship Detail Properties

Analytic Data Tables

Table Name	Description
DataRecord20XX	Data Unit Intersection Storage (12 Period Time) – Year
BinaryData20XX	Data Unit Intersection Storage (Variable Time) - Year

PERFORMANCE TUNING

There is really no way around it when you have performance issues. The hardware and software have limitations. If you have performance issues, unless you made a fatal mistake, you have exceeded either what the hardware or the software can do. First, we will talk about the single most important design consideration, the data unit. Next, we will talk about how you got here with your client. Then finally this paper will cover what you can now do in OneStream that does not include a complete rebuild. The good news is that once upon a time, rebuild and redesign were your only options.

If you made a mistake, the support team at OneStream can help you track it down. It could be something as simple (but tough to find) like a setting on the server or database that needs to be updated. You could have a rule that is causing some data explosion. You could have a poorly written report that is calling so much data to memory it is slowing the system.

MEMORY MANAGEMENT

When experiencing performance issues watching the memory utilization can tell you quite a bit about what is happening. If memory spikes and stays maxed out, there could be a couple issues going on. The IT team can focus on setting to ease the pressure. By changing the number of threads available you can impact performance. However, this does not mean the consulting team has nothing to do. A spike like this is also a sign of data explosion or other rules issues. The team should review the rule log an investigate is any rules are taking a long time to run. Anything over a second is high. Try turning that rule or section of rules off and see if that resolves the issue.

TOOLS AVAILABLE

When a consolidation operation is running slower than would be expected, there are a few tools available to diagnose what could be contributing to the slow performance.

- Long-Running Formulas Debug Option within the Application Server Configuration File. A formula that is running for 3 to 5 seconds or more is something that should be investigated; this is a very long amount of time for a formula to compute.
    - This setting allows the system to write to the error log any formulas that – when the consolidation is run – take longer than the numeric value specified to complete processing.
    - Located in the `XFAppServerConfig.xml` file under the Multithreading Settings section of the configuration file.
    - Requires IISRESET on all application servers for change to take effect.

```
<NumSecondsBeforeLoggingSlowFormulas>5</NumSecondsBeforeLoggingS
lowFormulas>
```

- Application Analysis Reports are available in the Standard Application Reports (available on Marketplace).
    - Reports provide analysis of the Dimension Statistics, Formula Statistics, Data Statistics and Data Unit Statistics. These Reports are discussed in further detail later in this chapter.

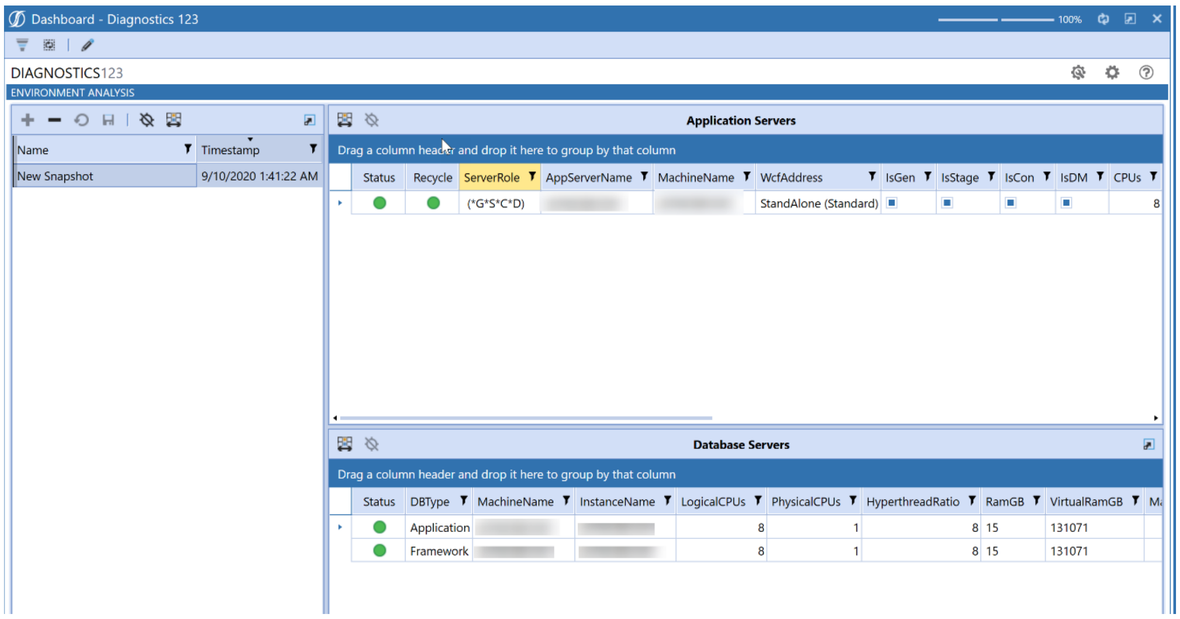
The **Diagnostics 123** Marketplace solution is another solution that is available for system administrators to monitor and diagnose overall environment and task health and performance. Diagnostics 123 provides many of the same features available in the Environment Tool, but in a different viewable format (PDF printable Reports).

Diagnostics 123 is broken up into three main pages:

1. Environment Analysis.
2. Task Analysis.
3. Live Monitoring.

The Environment Analysis page is the first page to access after setting the Data Unit sample year. It is recommended to set the Data Unit sample year to the *last full year* that data was loaded for, as it is used to calculate Data Unit sizing for environment analysis.

Environment analysis creates snapshots of the environment hardware state by gathering the information from the environment monitoring features in the software. The snapshots provide details on the application servers and database server.



Name	Timestamp
New Snapshot	9/10/2020 1:41:22 AM

Application Servers

Drag a column header and drop it here to group by that column

Status	Recycle	ServerRole	AppServerName	MachineName	WcfAddress	IsGen	IsStage	IsCon	IsDM	CPUs
●	●	(*G*S*C*D)	[redacted]	[redacted]	StandAlone (Standard)	■	■	■	■	8

Database Servers

Drag a column header and drop it here to group by that column

Status	DBType	MachineName	InstanceName	LogicalCPUs	PhysicalCPUs	HyperthreadRatio	RamGB	VirtualRamGB	M
●	Application	[redacted]	[redacted]	8	1	8	15	131071	
●	Framework	[redacted]	[redacted]	8	1	8	15	131071	

Figure 14.15

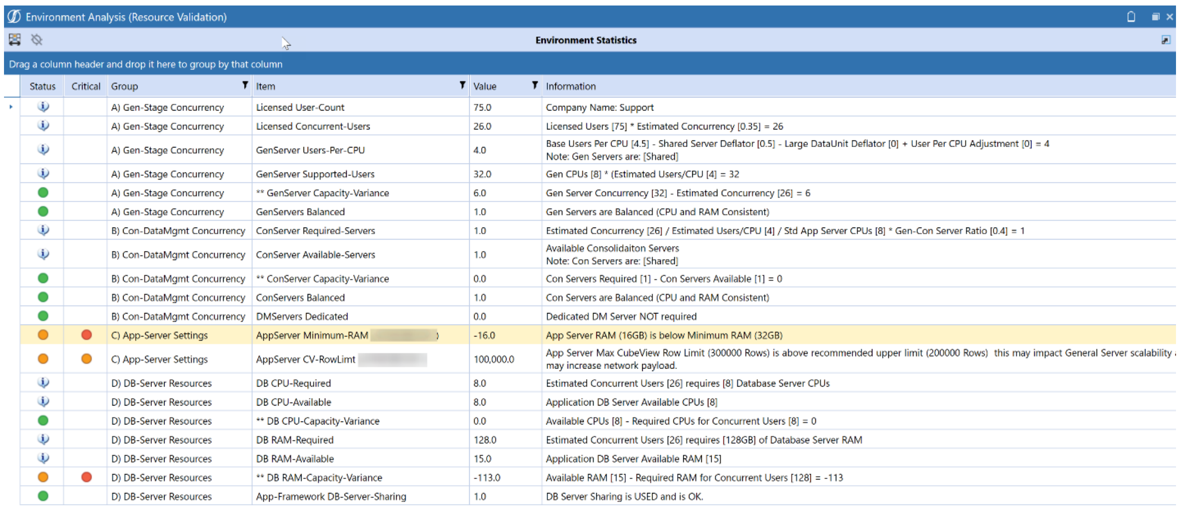
The application servers table lists each of the application servers contained in the environment as well as additional information

- Role Assignment.
- Number of CPUs.
- RAM.

- Application Server Configuration File Settings (Parallelism, Reserved Memory Setting, etc.).
    - CPUMipScore.
        - This is important to note as it shows how fast the CPUs are on the server. The higher the MIP recorded score, the faster the processors in the environment. Fast processors are best for consolidation and data management server roles in the environment.

The database servers table lists the database server that is currently being used for the application and framework databases in the environment, and corresponding resources.

The Analysis button within the page will perform a resource validation that displays a traffic light Report, based on calculations performed in the solution. This is a great tool to verify that the environment has the appropriate number of general application servers and consolidation servers based on historical metrics of supporting 4.5 concurrent users per CPU before including any deflators due to large Data Units and shared server roles.



Environment Analysis (Resource Validation)

Environment Statistics

Drag a column header and drop it here to group by that column

Status	Critical	Group	Item	Value	Information
ℹ		A) Gen-Stage Concurrency	Licensed User-Count	75.0	Company Name: Support
ℹ		A) Gen-Stage Concurrency	Licensed Concurrent-Users	26.0	Licensed Users [75] * Estimated Concurrency [0.35] = 26
ℹ		A) Gen-Stage Concurrency	GenServer Users-Per-CPU	4.0	Base Users Per CPU [4.5] - Shared Server Deflator [0.5] - Large DataUnit Deflator [0] + User Per CPU Adjustment [0] = 4 Note: Gen Servers are: [Shared]
ℹ		A) Gen-Stage Concurrency	GenServer Supported-Users	32.0	Gen CPUs [8] * (Estimated Users/CPU [4] = 32
🟢		A) Gen-Stage Concurrency	** GenServer Capacity-Variance	6.0	Gen Server Concurrency [32] - Estimated Concurrency [26] = 6
🟢		A) Gen-Stage Concurrency	GenServers Balanced	1.0	Gen Servers are Balanced (CPU and RAM Consistent)
ℹ		B) Con-DataMgmt Concurrency	ConServer Required-Servers	1.0	Estimated Concurrency [26] / Estimated Users/CPU [4] / Std App Server CPUs [8] * Gen-Con Server Ratio [0.4] = 1
ℹ		B) Con-DataMgmt Concurrency	ConServer Available-Servers	1.0	Available Consolidaiton Servers Note: Con Servers are: [Shared]
🟢		B) Con-DataMgmt Concurrency	** ConServer Capacity-Variance	0.0	Con Servers Required [1] - Con Servers Available [1] = 0
🟢		B) Con-DataMgmt Concurrency	ConServers Balanced	1.0	Con Servers are Balanced (CPU and RAM Consistent)
🟢		B) Con-DataMgmt Concurrency	DMServers Dedicated	0.0	Dedicated DM Server NOT required
🟠	🔴	C) App-Server Settings	AppServer Minimum-RAM [redacted])	-16.0	App Server RAM (16GB) is below Minimum RAM (32GB)
🟠	🟠	C) App-Server Settings	AppServer CV-RowLimt [redacted]	100,000.0	App Server Max CubeView Row Limit (300000 Rows) is above recommended upper limit (200000 Rows) this may impact General Server scalability may increase network payload.
ℹ		D) DB-Server Resources	DB CPU-Required	8.0	Estimated Concurrent Users [26] requires [8] Database Server CPUs
ℹ		D) DB-Server Resources	DB CPU-Available	8.0	Application DB Server Available CPUs [8]
🟢		D) DB-Server Resources	** DB CPU-Capacity-Variance	0.0	Available CPUs [8] - Required CPUs for Concurrent Users [8] = 0
ℹ		D) DB-Server Resources	DB RAM-Required	128.0	Estimated Concurrent Users [26] requires [128GB] of Database Server RAM
ℹ		D) DB-Server Resources	DB RAM-Available	15.0	Application DB Server Available RAM [15]
🟠	🔴	D) DB-Server Resources	** DB RAM-Capacity-Variance	-113.0	Available RAM [15] - Required RAM for Concurrent Users [128] = -113
🟢		D) DB-Server Resources	App-Framework DB-Server-Sharing	1.0	DB Server Sharing is USED and is OK.

Figure 14.16

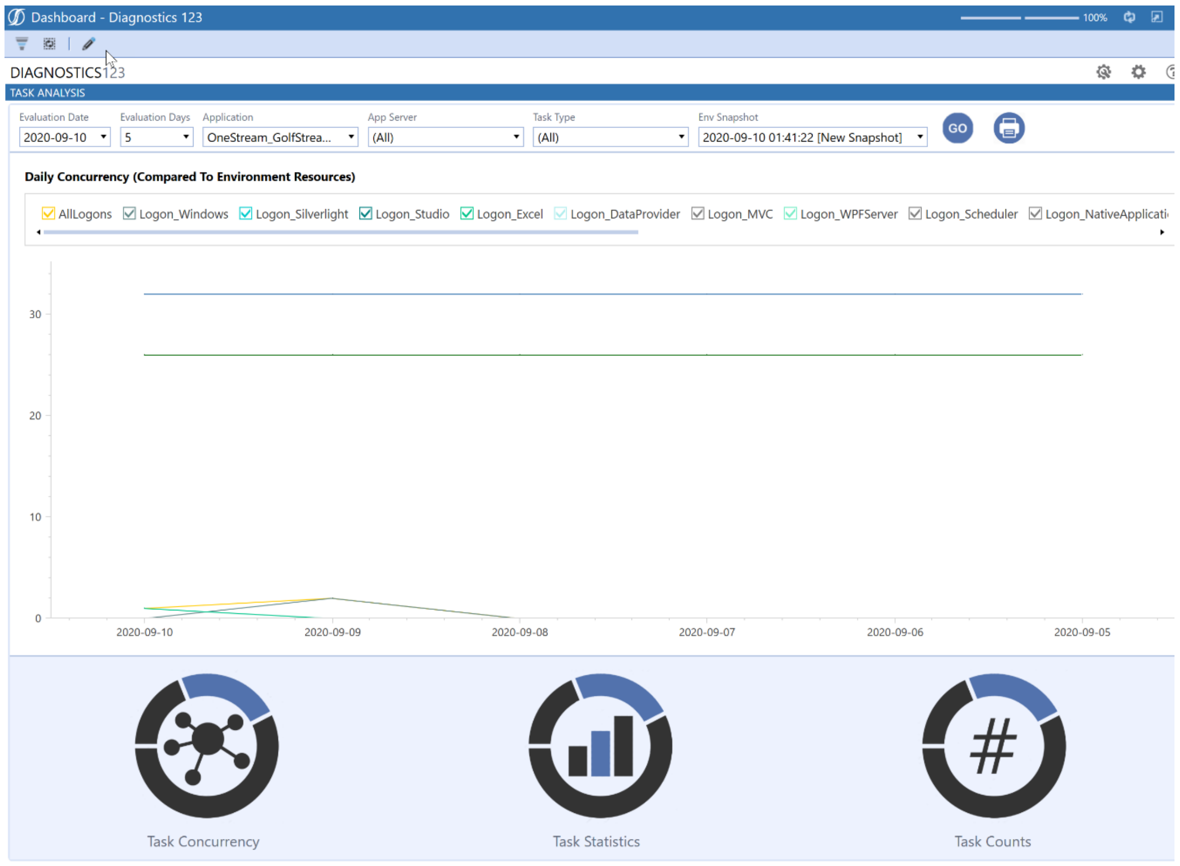
There are also some additional analysis Reports that are available for viewing within the environment analysis page:

- Large Data Unit Detail Report.
    - This Report will list out the top 200 large Data Units in the application, based on the year set in the settings.
    - If Data Unit size is getting large (larger than 250,000 records at the top parent), this will have an impact on the number of concurrent users that can be supported per CPU in the system. That's because each user will be holding onto a CPU longer on the general application servers to pull back the data in a Cube View Report, and will therefore decrease the number of concurrent users supported per CPU. This will be an indicator to assist in sizing the appropriate number of general application server CPUs for the environment.
- Memory Manager Report.

- This Report displays how often the .NET memory manager executes in the environment to remove the oldest Data Units from the analytic cache in the environment. If the memory manager is executing at a high rate on a daily basis, this is an indication that the servers require additional RAM resources to handle the analytic model and data volumes being processed. An example of these entries can be seen in the OneStream error log below:

  Summary: The Data Cache Memory Manager removed Data Units from the cache. Initially there were 2,399 Data Units and 7,984,332 records, and the largest Data Unit contained 357,552 records. Afterwards there were 896 Data Units and 2,475,821 records, and the largest Data Unit contained 168,664 records.

The Task Analysis page facilitates research tasks by concurrency, statistics, and overall counts. This is very useful for viewing daily logins to the environment by module (Windows App, Excel, Studio, API) to view true concurrency in the system over a period of time.



Evaluation Date: 2020-09-10
Evaluation Days: 5
Application: OneStream_GolfStrea...
App Server: (All)
Task Type: (All)
Env Snapshot: 2020-09-10 01:41:22 [New Snapshot]

Figure 14.17

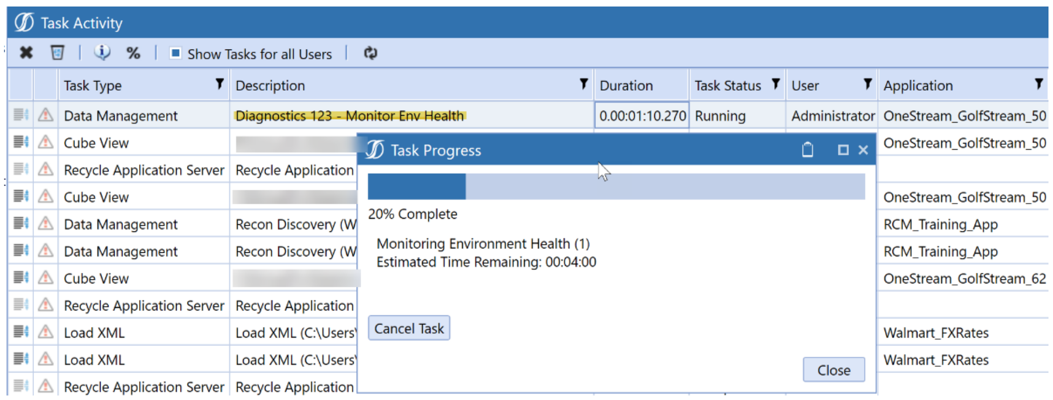
The graph displays a line to show the estimated user concurrency and also the maximum supported user concurrency. They can be used as a guide against the total user logons to verify that there are enough general application servers in the environment to support the user community.

The **Task Concurrency** module will display charts showing the number of tasks that were run on a day, by task type, and then allow for drill down into the graphs to view the data by hour and minute, as well as the individual task in task activity.

Task Statistics display daily runtime statistics by Type (`Max,Min,Avg`) for the task Type selected. This data can be filtered by application servers and by applications.

Task Counts display the total number of daily tasks that were run for a particular task type. For example, if the user selects Cube View, it will display the total number of Cube View tasks run for a day, and display how many were completed, failed, or cancelled.

The **Live Monitoring Module** provides environment and task health data over a given timeframe. This is the same functionality that is available in the Environment Tool under Monitoring, but is displayed in an analysis traffic light Report and also a pivot grid. The user is able to set a duration of time to monitor the system and an interval in seconds where the system will capture pre-defined environment metrics and then display them in a traffic light Report. This process is performed via a data management job in the solution to capture the metrics, before writing the results to a Report with explanations.



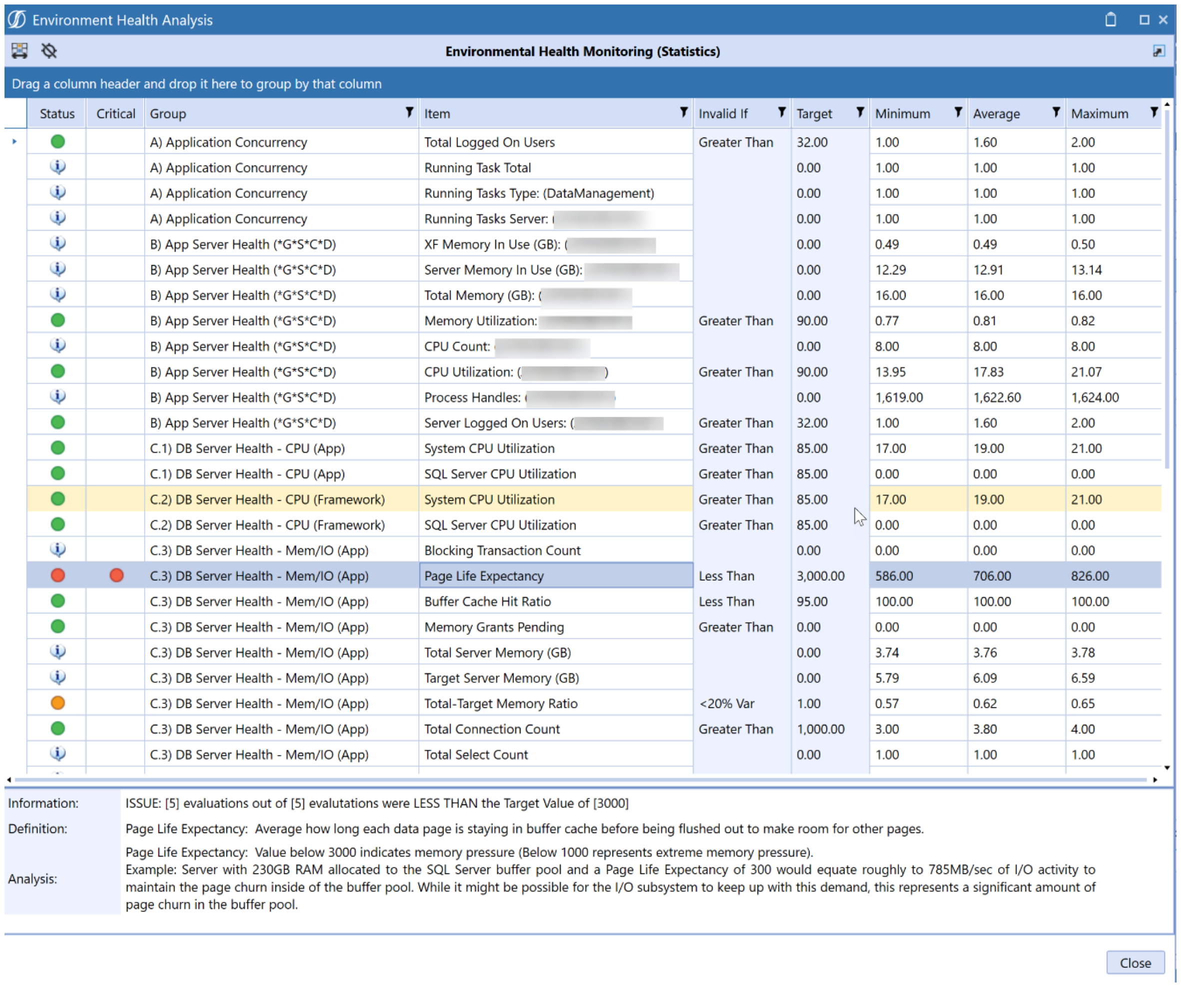
Task Type	Description	Duration	Task Status	User	Application
Data Management	Diagnostics 123 - Monitor Env Health	0.00:01:10.270	Running	Administrator	OneStream_GolfStream_50
Cube View	[redacted]				OneStream_GolfStream_50
Recycle Application Server	Recycle Application				
Cube View	[redacted]				OneStream_GolfStream_50
Data Management	Recon Discovery (W				RCM_Training_App
Data Management	Recon Discovery (W				RCM_Training_App
Cube View	[redacted]				OneStream_GolfStream_62
Recycle Application Server	Recycle Application				
Load XML	Load XML (C:\Users\				Walmart_FXRates
Load XML	Load XML (C:\Users\				Walmart_FXRates
Recycle Application Server	Recycle Application				

20% Complete

Monitoring Environment Health (1)
Estimated Time Remaining: 00:04:00

Figure 14.18

When the data management job is complete, the results can be viewed in the Analysis Report, which displays the results captured, and highlights any items that are critical to address in the environment.



Environment Health Analysis

Environmental Health Monitoring (Statistics)

Drag a column header and drop it here to group by that column

Status	Critical	Group	Item	Invalid If	Target	Minimum	Average	Maximum
🟢		A) Application Concurrency	Total Logged On Users	Greater Than	32.00	1.00	1.60	2.00
ℹ		A) Application Concurrency	Running Task Total		0.00	1.00	1.00	1.00
ℹ		A) Application Concurrency	Running Tasks Type: (DataManagement)		0.00	1.00	1.00	1.00
ℹ		A) Application Concurrency	Running Tasks Server: ([redacted]		0.00	1.00	1.00	1.00
ℹ		B) App Server Health (*G*S*C*D)	XF Memory In Use (GB): ([redacted]		0.00	0.49	0.49	0.50
ℹ		B) App Server Health (*G*S*C*D)	Server Memory In Use (GB): [redacted]		0.00	12.29	12.91	13.14
ℹ		B) App Server Health (*G*S*C*D)	Total Memory (GB): ([redacted]		0.00	16.00	16.00	16.00
🟢		B) App Server Health (*G*S*C*D)	Memory Utilization: [redacted]	Greater Than	90.00	0.77	0.81	0.82
ℹ		B) App Server Health (*G*S*C*D)	CPU Count: ([redacted]		0.00	8.00	8.00	8.00
🟢		B) App Server Health (*G*S*C*D)	CPU Utilization: ([redacted])	Greater Than	90.00	13.95	17.83	21.07
ℹ		B) App Server Health (*G*S*C*D)	Process Handles: ([redacted])		0.00	1,619.00	1,622.60	1,624.00
🟢		B) App Server Health (*G*S*C*D)	Server Logged On Users: ([redacted]	Greater Than	32.00	1.00	1.60	2.00
🟢		C.1) DB Server Health - CPU (App)	System CPU Utilization	Greater Than	85.00	17.00	19.00	21.00
🟢		C.1) DB Server Health - CPU (App)	SQL Server CPU Utilization	Greater Than	85.00	0.00	0.00	0.00
🟢		C.2) DB Server Health - CPU (Framework)	System CPU Utilization	Greater Than	85.00	17.00	19.00	21.00
🟢		C.2) DB Server Health - CPU (Framework)	SQL Server CPU Utilization	Greater Than	85.00	0.00	0.00	0.00
ℹ		C.3) DB Server Health - Mem/IO (App)	Blocking Transaction Count		0.00	0.00	0.00	0.00
🔴	🔴	C.3) DB Server Health - Mem/IO (App)	Page Life Expectancy	Less Than	3,000.00	586.00	706.00	826.00
🟢		C.3) DB Server Health - Mem/IO (App)	Buffer Cache Hit Ratio	Less Than	95.00	100.00	100.00	100.00
🟢		C.3) DB Server Health - Mem/IO (App)	Memory Grants Pending	Greater Than	0.00	0.00	0.00	0.00
ℹ		C.3) DB Server Health - Mem/IO (App)	Total Server Memory (GB)		0.00	3.74	3.76	3.78
ℹ		C.3) DB Server Health - Mem/IO (App)	Target Server Memory (GB)		0.00	5.79	6.09	6.59
🟠		C.3) DB Server Health - Mem/IO (App)	Total-Target Memory Ratio	<20% Var	1.00	0.57	0.62	0.65
🟢		C.3) DB Server Health - Mem/IO (App)	Total Connection Count	Greater Than	1,000.00	3.00	3.80	4.00
ℹ		C.3) DB Server Health - Mem/IO (App)	Total Select Count		0.00	1.00	1.00	1.00

Information:	ISSUE: [5] evaluations out of [5] evalutations were LESS THAN the Target Value of [3000]
Definition:	Page Life Expectancy: Average how long each data page is staying in buffer cache before being flushed out to make room for other pages.
Analysis:	Page Life Expectancy: Value below 3000 indicates memory pressure (Below 1000 represents extreme memory pressure). Example: Server with 230GB RAM allocated to the SQL Server buffer pool and a Page Life Expectancy of 300 would equate roughly to 785MB/sec of I/O activity to maintain the page churn inside of the buffer pool. While it might be possible for the I/O subsystem to keep up with this demand, this represents a significant amount of page churn in the buffer pool.

Close

Figure 14.19

The Live Monitoring module also offers a means to monitor a particular task Type and identify the task as unhealthy if it is running for a long period of time (identifying environment resources at that period of time). This can be useful when a process is not responding, and if there is a particular environment resource that is under pressure, which needs to be addressed accordingly.

BEST PRACTICES

Some clients will be ok with a two-hour consolidation, others are upset if takes more than a few minutes. A useful design will consider client expectations. Know that the more data records you have in your Data Unit, the longer loads will take, longer calculations and consolidation will take, and all reports will be slower. Doing everything you can to manage the size of the Data Unit, even if the client is ok with a ‘slow’ time, is the right thing to do. No one ever gets unhappy if its faster than they expected.

While OneStream can handle Data Units in the 6-8 million records (or more), the hardware requirements are steep. In the best performing applications, the top of the house consolidation Entity should have approximately 250,000 total data records in a Data Unit This equates to 3,000,000 total data cells for the year. During a full year consolidation, 3,000,000 data cells will be processed. Once the data records start to reach beyond this point, physics and technology play a bigger role in performance. And even at this point hardware will be a consideration.

If the processor is using all its available horsepower calculating these Data Units, what is available for everything else the processor is supposed to be doing?

BOOK-OF-RECORD DATA PROCESSING

At this point if you have been building the application and it is not performing you will need to make some changes. The last resort will be a redesign. This is obviously not ideal. The client will ask what the cost and impact to the timeline is. You also lose credibility. It is not ideal. But the goal here is to leave a great application. You have some options to work through before falling back to restarting.

CUBE DESIGN

So, remember the Data Unit size is driven by data values in each cell. Hopefully, you made the Accounts, IC, and UD members all sparse dimensions. As the data moves up the entity hierarchy there will be some overlap, but not 100%. For example, if you have a product dimension the first entity may have 10% of the members populated. The second entity may have 20% populated. That would mean at a parent level the product dimension is between 10% and 30% populated. Eventually as you keep moving up the entity tree, the dimension may be come dense.
I might look at a dimension with hundreds of members and ask – is there a report for this single entity that shows all this detail. Not likely they will have a report with thousands of columns. If you are challenged on this, refer to the client issue from above. They might not know what they are looking for. They want the detail, and honestly do not care how they get to it.
You have an option here to create a detailed and summary cube design. Work with them to find the break point in the design where there is no value in seeing all that detail in one report. Did you combine dimensions? Maybe break them out. You must determine the Cubes by purpose and Dimension grouping. Security and Workflow requirements will drive the need for more Cubes.

DETERMINING THE DIMENSIONS IN THE DATABASE

- One conceptual Dimension to one Dimension Type
    - Example: Dollar amounts; statistics and rates are all measures and should be in the same Dimension
    - Example: Currency and Scenario are two different concepts and should be in different Dimensions
- All determined Dimensions do not need to necessarily exist in the same Cube or Scenario view
    - Create a Cube / Dimension Matrix:

For example, the following table demonstrates which Dimension and Dimension detail *(where necessary)*, is included in each Cube:

Dimensions	**Cube 1** **(CORP Financial Data - SUMMARY)**	**Cube 2** **(Plan/Forecast)**	**Cube 3** **(Subsidiary - DETAIL)**
Time Periods	X	X	X
Years	X	X	X

Scenarios	X	X	X
Flow	X	X	X
Accounts	IFRS	Only salary relevant	Mgmt. COA
Entities	X	X	X
UD1 Products	-	X	-
UD2 Projects	Project groups	-	X

“UNSPECIFIED” DIMENSION MEMBERS

In the case where specific data points do not exist across one or more Dimensions *(e.g., Balance Sheet Account detail does not exist across the Product and Customer Dimensions)*, use *None* Members in each of these Dimensions to designate a placeholder Member to contain the data.

TRANSACTIONAL ANALYTICS

CUBE INTEGRATION

Cubes may be linked by creating:

- Linked Cubes via the Entity Dimension
- Rules to copy data from one Cube to another

WHY THIS HELPS

It is never a smart design practice to use a single cube, unless the application is and will always be small. So, the summary detail cube is a great approach if the data requirement ever changes. They will still have all the detail they need but get significant performance improvements as the data consolidates. Data unit’s sizes stay manageable in this design.

HIGH VOLUME DATA TRANSFORMATION

ANALYTIC BLEND

There is a saying when you are a hammer, every problem looks like a nail. You might be comfortable with the cube, but it is not for every problem. Analytic Blend is used to report on large volumes of transactional data that is not appropriate to store in the traditional financial cube. Transactional data should never be loaded with actual financials. This keeps this data out of the cube but keeps it in a SQL table you can report on. The BI Blend allows for some subtotaling and simple calculations.

Analytic Blend processing occurs on the stage application servers in the environment. There are two main ways to execute processing of Analytic Blend within OneStream.

OVERNIGHT BATCH JOBS

- Run via Data Management to populate the Analytic Blend database table with transactional data for reporting
- Appropriate for large analytic blend implementations (millions of records generated)

INTERACTIVE WORKFLOW

- Analytic Blend runs interactively by the user via in-line workflow tasks. Users access the workflow as they normally do, and execute Load and Transform to populate the Analytic Blend database table
- Appropriate for small analytic blend tasks

WHY THIS HELPS

Simply the Data Units are smaller with the detail in a table and not in the cube.

DATA LOADING

So, when you have a design issue the first place you see problems is importing the data. The files are exceptionally large. You will need to break up the files by entity and load them in batches. This is called partitioning the data. You will also see issues transforming the data, again because of the volumes.

UNDERSTANDING TRANSFORMATION RULE PERFORMANCE

The performance of the Execution of Transformation Rules in the load and transform process is dependent on the types of transformation rules used within the transformation rule profile assigned to the workflow profile. Each type of transformation rule will have an associated processing cost associated with it. Each of the transformation rule processing types that are available have been broken up by cost, below.

Once the data has been successfully imported, transformed, and validated, the data is then loaded to the cube. The Load Cube step of the stage workflow process moves the data from the stage tables in the application database into the data record tables in the cube. The first time the cube is loaded for a Workflow/Scenario/Time combination, the process will perform database record **Inserts** into the following three database tables in the application database:

- Calc Status Table (`CalcStatus`)
- Time Stamp Table (`Data UnitCacheTimeStamp`)
- Data Record Table (`DataRecordxxxx` or `BinaryDataxxx`)

Any subsequent load cube operations that are performed for the same Workflow/Scenario/Time (Workflow Data Unit) combination will be more efficient than the initial load cube as the process will perform database Updates on these tables versus inserts, which will be more efficient at the database level.

The Load Cube Step of the workflow is also intensive on the network between the application server and the database server tier in the environment. The process is performing heavily multithreaded calls per cell between the application server and the database server, which results in high network pressure between the two servers. It is especially important that there is no network latency between the stage application server and the database server to obtain peak performance with the process.

AGGREGATION ONLY

Consolidation of Actuals data is generally rigid and requires more controls, features like intercompany eliminations and data auditing are done as part of the process, because the financial reporting of Actuals is much more rigid and requires more controls. These features are not always needed. For other reporting processes. By turning off that financial intelligence, it takes out some of the heavy lifting that the system does, in favor of flexibility and speed.

Consolidation is the process of rolling up financial data heavily driven by financial and accounting rules from the business entity and the structure up to the parent level for reporting.

> *"That's because Consolidations are usually crafted to satisfy internal management and external regulatory agency reporting requirements. The most common (and effective) method to truly understand the core requirements of a Consolidation system is to begin with the end in mind by looking at the reports produced by the legacy (or current) system. Typically, these involve an Income Statement (or Profit & Loss Statement), a Balance Sheet, and a Cash Flow Statement"*
>
> *– OneStream Architect Factory*

Aggregation on the other hand bypasses most of the statutory financial and accounting rules to consolidate/aggregate data up the entity dimension to provide fast what-if scenarios. Like Consolidation, the Fast Entity Aggregation process uses the Entity and the Consolidation dimension (Aggregated member) to aggregate data to the parent level using minimal financial/accounting rules. Planning solutions do not require the detail of consolidation, auditability, and eliminations and as such, are wonderful use cases for the Aggregation Only option.

Aggregation Algorithm:

- Execute chart logic (the business rules and member formulas for calculate) on the Local Consolidation member for all Base Entities.
- Execute the following steps recursively for each Parent and its direct children starting from lower-level entities to the Parent Entities.
    - For each child
        - Translate its stored data in memory
        - Calculate its Share amount in memory
    - Add the data cells from each child in memory
    - Store the results in the Aggregated member for the Parent Entity

HYBRID SCENARIOS

Report query times can be long with large Data Units. With large Data Units, the way we present data will cache the entire Data Unit and then complete the calculations. As you can imagine, large Data Units will take more time and resources to bring across the network. The Hybrid Scenarios feature has two options: Copy Data and Share Data. The Share Data from a Source Scenario option resolves the issue created by these large Data Units. By using the Pre-Aggregated Members setting, this option sub-divides the current Data Unit into a smaller unit of data and then does the aggregation and then caches in RAM. The filtering of the members is done before it is loaded into

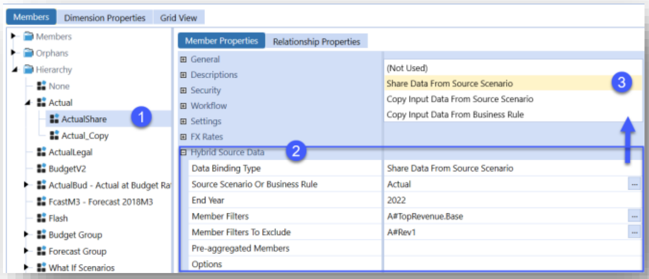
RAM. At that point it would bring the smaller data set into RAM and Share or Copy the information over to the target account(s).

You create a slice of the scenario and store it dynamically. The slice of the scenario might take a top of a UD, or summary level. The result is the Data Units being called for the reports are smaller. Your reports then query this new slice.

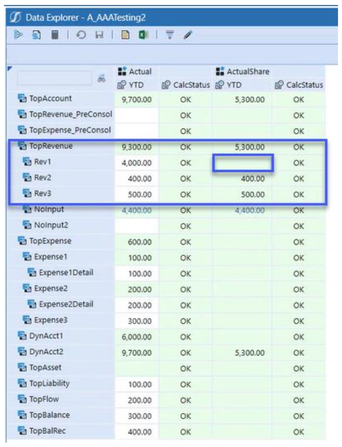
Share Data from a Source Scenario

- Shares Cube data from a specified source Scenario to a specified target Scenario
- Data results can be filtered by the following:
  - Include and/or exclude specific Dimension Members
  - source time periods and end years
  - source View: YTD or Periodic
  - pre-aggregated Members *(Data from a Parent Member in the source Scenario can be shared to a Base Member in the target Scenario)*
- Shared data results are read-only – **so this does not impact data storage**
- Shared data results are a pointer to the data stored in the source Scenario; data is not stored in the target Scenario
- A shared target Scenario also shares the source Scenario's calc status
- If standard calculations/translations/consolidations are executed from a shared target Scenario, it will run on the source Scenario

COPY AND SHARE CONFIGURATION



Data Binding Type	Share Data From Source Scenario
Source Scenario Or Business Rule	Actual
End Year	2022
Member Filters	A#TopRevenue.Base
Member Filters To Exclude	A#Rev1
Pre-aggregated Members	
Options	



Data Explorer - A_AAATesting2

	Actual		ActualShare	
	YTD	CalcStatus	YTD	CalcStatus
TopAccount	9,700.00	OK	5,300.00	OK
TopRevenue_PreConsol		OK		OK
TopExpense_PreConsol		OK		OK
TopRevenue	9,300.00	OK	5,300.00	OK
Rev1	4,000.00	OK		OK
Rev2	400.00	OK	400.00	OK
Rev3	500.00	OK	500.00	OK
NoInput	4,400.00	OK	4,400.00	OK
NoInput2		OK		OK
TopExpense	600.00	OK		OK
Expense1	100.00	OK		OK
Expense1Detail	100.00	OK		OK
Expense2	200.00	OK		OK
Expense2Detail	200.00	OK		OK
Expense3	300.00	OK		OK
DynAcct1	6,000.00	OK		OK
DynAcct2	9,700.00	OK	5,300.00	OK
TopAsset		OK		OK
TopLiability	100.00	OK		OK
TopFlow	200.00	OK		OK
TopBalance	300.00	OK		OK
TopBalRec	400.00	OK		OK

- If enabled, calculations can still run on the Hybrid Source Data Scenario setting, but the data copy will only occur when running the calculation from the Data Management Step
    - "Use Default" will defer to the Scenario's Hybrid Source Data "ExecuteCopyDuringCalc" (T/F)
    - ’True" will execute the copy regardless of the Scenario's Hybrid Source Data"ExecuteCopyDuringCalc"
    - "False" will cause the copy to be skipped regardless of the scenario's options.

PLANNING DATA PROCESSING

Another option to consider when building planning applications and solution is to us the Custom Calculate Function. The **Calculate Function Type** runs within the Data Unit Calculation Sequence (DUCS). All the steps in the DUCS are executed each time a calculation is run for a Data Unit. This means all Member Formulas, formula passes, and Business Rules will run each time. The DUCS is built into the calculation and consolidation engine and is very useful in preserving data quality by organizing many calculation functions such as clearing data and writing to Members within the Consolidation Dimension. All of this is not needed for planning, and in fact sometimes changing one number does not require a full recalculation of everything.

The **Custom Calculate Function Type** allows calculations to take place *outside* the Data Unit Calculation Sequence. Calculations can be built to be more specific to the exact view of data that a user is working with. A user can calculate sales for five departments without executing any other calculations in the Cube.

When seeding data, you should allow for those rules to run only when needed. Rules that run and yield the same numbers over and over provide no added value for the time it takes. As part of this rule, it is strongly recommended to use Durable Data rules. This will make sure the seeded data is not cleared when the rules are off.

ZERO AND ‘NEAR ZERO DATA’

Do not load zeros and values that do not provide financial significance. The first place many application reviews start with is looking at zero and data that is close to zero. It is not uncommon in poorly performing applications to see up to 20% to 30% of the data to be useless. This could be from a bad rule, or periodic zeros loaded for plan or forecast. Get rid of them!

PROBLEM STATEMENTS TO AVOID DURING DESIGN

A clever design considers the size and number of these Data Units, and ensures not only will they perform well now, but in the future as the application evolves.

So, Data Unit and hardware requirements are linked. If you know you will have many or large Data Units, there will be more of demand on processing speed and available memory. If you made assumptions and need to revisit the design, make sure you consider the hardware too. That is an additional cost the client may have not anticipated. They will either need to scale back requirements or reevaluate infrastructure resources.

Implementers get into trouble with Data Unit size in some common areas. Mostly these are a result of not challenging the requirement well enough. A good design will put the detail where it will be provided quickly and accurately – that is not always just one way.

'We don't know what we are looking for…' – Clients coming from management backgrounds or replacing management systems often only know there might be an issue, and they have no idea if it exists. The process is a 'detect' process and not 'prevent.' This is a wonderful opportunity to help them identify what they are looking for and create a programmatic way of identifying and then reporting on what they want. This is much more thoughtful way do the analysis. But it requires understanding the process they are going through.

'We have a lot of data' Clients who have massive amounts of data often desire to see detail, but when pressed they often start at a summary level. Of the problems you might encounter, this is the easier of the problems to solve because you can the break the cube between detail and summary data or use one of the tools to store detail like a stage table or relational blend.

'We didn't implement this other system right' This problem is a common one, and frankly one of the most difficult solve for. The reason is that not only are they completely convinced they need massive amounts of data, but it also might be a reason they bought the software. They may have convinced themselves that if they go to this CPM tool, they can report on data detail that should not always be in a CPM tool. Another reason this is a red flag, is, whatever the challenge was during the implementation of those other tools will surely be an issue for this project too. For example, if the ledger was led by IT, and they did not understand the business process, and now the ledger is missing detail. The finance team may have been told this will fix their problems by IT. Companies will repeat mistakes without a strong leader.

'We need to load every ten minutes and consolidate everything constantly' You have to be very careful when presented with a challenge like this. First, examine the data and understand how much is really changing. Often there is an assumption it is more fluid than it is. Second, dig into the end user process and determine if they users would be disrupted by data changing while they are working on it. Less frequent data loads and consolidations are usually more appropriate. You should also ensure the workflow is only doing the calculations you need for that specific data set the end user is working on.

Whatever the challenge presented by the client, failing to follow the guidelines, and building an application within the design framework will result in an application that performs slowly – or at a minimum not within the expectations of the client.

TEST, TEST, TEST…

The time to find out your design is not adequate, is not the two weeks before going live. I start my testing with sample data as early as two weeks after I have built metadata. I start with a prototype build and have even sat in a conference room with the entire project team participated in the design. This early prototyping will ensure that not only will the look and feel be what they expect, but I can see real data and validate all my assumptions during design.

This prototype is not meant to be a fully detailed build, but more of a quick build (2-3 days maximum) based on available information. The prototype can then be updated based on client feedback and available Metadata, etc.

SOME CLOSING THOUGHTS ON DESIGN

The rules engine offers flexibility and power for the creator of rules. However, it also offers the ability for poorly written and designed rules to explode the database size. It is critical to design and write rules as explicitly as possible. Define where the data should reside within the database to avoid calculations that populate the database with unwanted values. Rules must always be tested prior to loading in a production environment to avoid any pitfalls of poorly designed rules.

OneStream will provide detail on the data that is zero in the **Data Unit Statistics**. This is available from a custom Report or by right-clicking on a cell in a grid. (See Figure 3.1.) The number of zeros should be monitored closely, and if they either spike significantly or increase above 10% of the data, they should be addressed. You will need to identify the source of the zeros and resolve it.

It is important to note that the period is part of the Data Unit. So, if you loaded data in each month, and did not load data in the subsequent months, the system will generate either a year-to-date or periodic zero. While this is not real data, you will see that number if you loop over the cells of the Data Unit in your rules. Stored calculations also add cells of data that could require processing.

PROCESSING GUIDELINES AND LIMITS

In this section suggested or logical data processing limits will be defined. These limits are used to suggest best practice limits for optimal data processing performance. In most cases that actual / hard-limits will be much higher than the best-practice logical limits but pushing data volumes to the hard-limits will potentially create hardware and/or system processing inefficiencies.

LIMITS TABLE

Function	Logical Limit	Hard Limit	Rational
Stage Detail Rows Per Workflow ***Single Workflow Limit Only***: Partitioning Enables Unlimited Data Loads with Compute Resource Constraints.	1,000,000	24,000,000	Hard limit only exists for Direct Loads. It is always best to partition large source data loads across multiple workflows for both optimal load performance and efficient error corrections (Enables partition reprocessing instead if entire data source).
Stage Summary Rows Per Workflow ***Single Workflow Limit Only***:	1,000,000	24,000,000	Hard limit only exists for Direct Loads. It is always best to partition large source data loads across multiple workflows for both optimal load performance and

Partitioning Enables Unlimited Data Loads with Compute Resource Constraints.			efficient error corrections (Enables partition reprocessing instead if entire data source).
Data Unit Stored Rows ***Logical Limits Defined***: Represent the point at which Performance Degradation is eminent. Depending on model size, logical row limits may be higher or lower than stated.	1,000,000-2,000,000	No Hard Limit, But Memory/CPU Constrained	Limited compute resources. Exceeding 1,000,000-2,000,000 rows in a single Data Unit dramatically increases server memory requirements and CPU speed becomes increasingly important due to enormous number of cells that must be enumerated in a single block.
Entities Per Cube Logical Limits Defined: Represents the point at which performance degradation is eminent. A single entity dimension can be split into multiple linked entity dimensions and used to create partitioned cubes with groups of entities.	20,000-30,000	No Hard Limit, But Query Performance will degrade with a high number Data Units	Limited by Query Response Expectations. If a cube contains an exceptionally large count of entities, CubeViews that return all entities must load/swap many Data Units in/out of memory (This can slow down query response times). Individual entities and sub-trees will run quickly, but all entity requests will generate a lot of IO at least for the first user to access the entities on the server.

CONCLUSION

Large Data Units can create problems for loading, calculating, consolidating, and reporting data. This really is a limitation of what the hardware and networks can support. Your design needs to consider this. But from this paper, I hope you can take away some options to relieve some of the pressure points that could appear.