



How To Use Cache Framework

An Open Source Asset for use with TIBCO® Data Virtualization

TIBCO Software empowers executives, developers, and business users with Fast Data solutions that make the right data available in real time for faster answers, better decisions, and smarter action. Over the past 15 years, thousands of businesses across the globe have relied on TIBCO technology to integrate their applications and ecosystems, analyze their data, and create real-time solutions. Learn how TIBCO turns data—big or small—into differentiation at www.tibco.com.

Project Name	AS Assets Cache Management
Document Location	This document is only valid on the day it was printed. The source of the document will be found in the ASAssets_CacheManagement folder (https://github.com/TIBCOSoftware)
Purpose	Developer Guide



www.tibco.com

Global Headquarters
3303 Hillview Avenue
Palo Alto, CA 94304

Tel: +1 650-846-1000
+1 800-420-8450
Fax: +1 650-846-1005

Revision History

Version	Date	Author	Comments
0.1	09/07/2010	Calvin Goodrich	Initial revision
0.2	03/06/2013	Calvin Goodrich	Updated to take advantage of the incremental caching framework of CIS 6.2.1
0.3	05/13/2014	Calvin Goodrich	Updated to reflect collection of AS tools and utilities into /shared/ASAssets
1.0	11/04/2014	Mike Tinius	Created the Cache Framework
1.1	11/20/2014	Mike Tinius	Added supported for SQL Server 2008 cache database. Added copyright text.
1.2	02/09/2015	Mike Tinius	Fixed path issue with copy privileges. Fixed issue with view resources always being created in upper case. Allows for mixed case views and DB resources. Reduced number of INFO statements by changing 2 nd and 3 rd tier to DEBUG statements.
1.3	12/014/2017	Mike Tinius	Transitioned to Tibco for release 2017Q4
2018Q1	March 2018	Mike Tinius	Updated to use introspectResources.
2019Q300	07/15/2019	Mike Tinius	Added MemSql and Postgres Support. Added ability to migrate cached views into the cache framework. Added ability for DeployCache to only rebuild the cache tables if they don't exist or have been modified otherwise do nothing. Modified all custom function references to explicit paths. Requires Utilities 2019Q200. Deprecated and removed the use of 2 pre-callback implementation and post-callback implementation procedures in order to simplify. Now only a single pre and post callback procedure is maintained. Deprecated and removed Constants variables [ApplicationCacheProcImplPath, DefaultPreCallbackImplSuffix, DefaultPostCallbackImplSuffix]. Added incremental cache attributes INCR_INITIAL_LOAD_SCRIPT and INCR_DELTA_LOAD_SCRIPT to track the location of those scripts.
2019Q301	08/01/2019	Mike Tinius	Modified all occurrences of getConstant to getConstantV2. Deprecating getConstant.
2019Q302	08/20/2019	Mike Tinius	Modified /Scripts/PublishedImpl/CachingDataModifyScript <ul style="list-style-type: none"> Fixed bug to correctly detect the last usage of getConstant in the script text.

			<ul style="list-style-type: none"> Added code to modify CachingData and change getConstant function to the fully qualified getConstantV2. <p>Modified /Scripts/PublishedImpl/DeconfigureCache</p> <ul style="list-style-type: none"> Fixed a bug to not throw an exception if the call back scripts are not found but expected to be there. <p>Added new procedure "ConfigureCache_Multi_Table" in each template to automatically add an entry into CachingData and configure the cache in one invocation using multi-table caching. Ease of use.</p>
2020Q100	01/03/2020	Mike Tinius	Various Bug Fixes
2020Q200	05/15/2020	Mike Tinius	Modified framework to allow for a separate cache database and system database that contains cache framework system tables and cache_status and cache_tracking.

Related Documents

Name	Version
How To Use Utilities.pdf	2020Q200

Supported Versions

Name	Version
TIBCO® Data Virtualization	7.0 or later
AS Assets Utilities open source	2020Q200 or later

Table of Contents

1	Introduction	8
	Purpose	8
	Audience.....	8
	References	8
2	Overview 9	
	Caching Overview	9
	Full Caching	9
	Incremental Caching.....	9
	Caching Data Source Targets	10
	Unsupported Features.....	10
	Partitioned Incremental Caching	10
	Historical11	
	Schema Differences.....	11
	Caching Terms	11
	Business Requirements	12
	Technical Requirements.....	12
3	Installation.....	14
	Requirements for Installation.....	14
	Installation Steps	14
4	Cache Framework Simple Use Case	17
	Introduction.....	17
	Pre-requisites	17
	Cache Lifecycle	17
	Configure a view cache	17
	Refresh a cache	18
	Deploy a cache.....	18
	De-configure a view cache	23
5	Configure Cache Database	25
	Introduction.....	25
	Configure Cache Database Use Current	25
	Configure Cache Database User Other	25
	Configure Native Loading.....	26
6	Configure Cache Framework	29
	Introduction.....	29
	Configuring an Application Template	29
	1) Copy Application Template.....	29
	2) Modify Constants	30
	3) Initialize All Scripts.....	32
	4) Migrate Pre-existing Cached Views	33
	5) Modify CachingData Procedure	34
	6) Configure the View Cache.....	34
	7) View AUDIT_LOG	36
	8) Refresh the Cache.....	37

9)	Deploy the Cache	37
10)	De-configure the Cache.....	38
7	Cache Framework Features	39
	Log to a generic audit table script	39
	Cache Types	40
	Cache Features and Attributes.....	40
	Distribution Columns.....	40
	Multi-Table Configuration.....	41
	Incremental and Hybrid Configuration	42
	Incremental Merge Configuration	43
	Full Cache Pre/Post Procedure Configuration	45
	Drop/Create Indexes	46
	Execute Table Statistics	48
	Create Cache Schedule	48
	Create Cache Trigger Schedule	49
	Rolling Purge Window	50
	Reset Maintenance Level.....	52
	Validation.....	52
	Ability to Initialize Framework.....	52
	Ability to De-configure a Cache.....	53
8	Cache Framework Deployment	55
	Overview of Deployment	55
	How to Deploy Cache Objects	56
	DeployCache Procedure	56
	Sample PDTTool Deployment Plan.....	58
9	Cache Framework Implementation.....	60
	Full Caching Methodology.....	60
	Incremental Caching Methodology.....	60
	Hybrid Incremental Caching Methodology	60
	Hybrid Incremental Merge Caching Methodology	60
	Scripts.....	61
	Cache Framework Application Template Folder	62
	Constants()	62
	CachingData()	63
	CachingData_Display()	64
	CachingData_Insert()	64
	AuditLogDisplay()	64
	ConfigureCache()	65
	ConfigureCache_Multi_Table()	65
	ConfigureCache_Multi_ColumnStore() [deprecated]	67
	ConfigureCacheAdhoc()	67
	DeconfigureCache()	69
	DeployCache().....	69
	MigrateCachedViews()	70
	PurgeCacheData().....	71
	RefreshCache()	71
	_Execute_All_Scripts()	72
	Create_AUDIT_LOG().....	72
	Create_AUDIT_LOG_SEQ()	72

Create_CACHING_DATA()	72
CreateDBSequence()	73
Drop_AUDIT_LOG()	73
Drop_AUDIT_LOG_SEQ()	73
Drop_CACHING_DATA()	73
DropDBSequence()	73
RebindPackagedQueries()	74
TEMP_DS	74
ExecuteDDL()	74
ExecuteDDLSystem()	74
GetIntResult()	75
GetIntResultSystem()	75
ShowIndexes()	75
ShowTables()	75
Published Procedures	75
ClearPlanCache()	75
ClearRepositoryCache()	76
ConfigureCache()	76
ConfigureCache_Multi_Table()	76
ConfigureCacheAdhoc()	76
CopyPrivilegesFromParent()	76
DeconfigureCache()	77
IntrospectTables()	77
RefreshCache()	77
UpdateResourceOwner()	77
Admin Interface Procedures	77
AdminInterface/ClearPlanCache()	77
AdminInterface/ClearRepositoryCache()	78
AdminInterface/ConfigureCache()	78
AdminInterface/ConfigureCache_Multi_Table()	78
AdminInterface/ConfigureCacheAdhoc()	78
AdminInterface/CopyPrivilegesFromParent()	78
AdminInterface/DeconfigureCache()	79
AdminInterface/IntrospectTables()	79
AdminInterface/RefreshCache()	79
AdminInterface/UpdateResourceOwner()	79
10 Appendix A – Configure Cache Framework Sample	80
Configuring the Cache Framework Sample	80
Background Information	80
Configure Oracle Sample:	81
Configure SQL Server Sample:	84
Configure Netezza Sample:	87
Configure MemSql Sample:	90
Configure Postgres Sample:	93
11 Appendix B – Merge Cache Types	96
Merge Cache Types	96
Merge and Slowly Changing Dimensions	96
Type 0	96
Type 1	96
Type 2	97
Type 3	98

Type 4 (CDC style)	98
Merge Type 1 – Cache Framework Example	99
Merge Type 2 – Cache Framework Example	100
Merge Type 4 – Cache Framework Example	102
12 Appendix C – Adding a New Cache Type	104
Adding a New Cache Type to the Source Code	104
13 Appendix D – Adding a New Cache Data Source Type	106
Adding a New Cache Data Source to the Source Code	106

1 Introduction

Purpose

Data Virtualization (DV) 7.0 / 8.x, features a framework to support the development of custom full and pull-based incremental caching solutions. This document describes such a solution developed in the field by the Professional Services Group (PSG) team. At this time, it only supports full and incremental caching of resources of views or tables (procedure output is not supported.)

It is highly recommended to acquire Tibco PSG help to implement the Cache Framework. Do not contact Tibco Support for any Tibco Open Source code.

Audience

This document is intended to provide guidance for the following users:

- Architects.
- Developers.
- Administrators.

References

Product references are shown below. Any references to CIS or DV refer to the current TIBCO® Data Virtualization.

- TIBCO® Data Virtualization was formerly known as
 - Cisco Data Virtualization (DV)
 - Composite Information Server (CIS)

2 Overview

Caching Overview

Caching in DV is the materialization of a resource (either view, table, or stored procedure) and storing of the materialized data in an external database or flat file (similar to the way a Materialized View operates in an Oracle database.) Resources in DV are configured to store materialized data in one or more tables in a relational data source. Once the basic configuration is done, a number of options exist as to when and/or how often a resource's cache is refreshed. However, this refresh is a full refresh of the entire data set every time a refresh is run (meaning an entirely new result set is loaded into the cache data table, after which old result sets from any previous refreshes are removed.)

Full Caching

Full caching is a method of updating all the cache data with the current set of rows from the source(s) of record. Each time the refresh is done it pulls all of the data. This works efficiently for relatively small data sets. It also works the best when caching complex views in the Business or Application layer of the Data Abstraction best practices as those views typically do not yield multi-million row tables. Additionally, due to complexities of joins it may not be possible to implement an incremental cache strategy due to conflicting keys.

Incremental Caching

Incremental caching is a method of updating existing cache data with new, updated, or deleted rows from the source(s) of record.

Push-based incremental caching involves being notified of changes by an application or change data capture mechanism and updating the cache data as source data changes. The requirements and setup for this within DV are described in Appendix F of the DV User's Guide.

Pull-based incremental caching (which is the implementation that this document discusses) is more of a batch-oriented process where a refresh is initiated in DV and a mechanism detects what data has changed since the last cache refresh (full or incremental) and applies those changes to the cache data as a single transaction.

There are a number of ways to detect change in data, including scraping database transaction logs, attaching CRUD triggers to database tables, and using a combination of columns to indicate when a row was last inserted, updated or deleted (deleted row data may also be stored in a separate table altogether.)

In this solution, two methods of detecting data change are supported "out of the box":

- One or more "last updated" columns and, for those tables that track it, a separate table of deleted rows to indicate when data was deleted. Also, a way of uniquely identifying each row in the cache data table (with either an actual or logical primary key) is required.
- A separate "delta" table is maintained by the updating application indicating what changes have taken place in the underlying data. This table contains only one record for each corresponding record in the underlying data with an extra column indicating what the last operation was. For example, when a delta row is inserted, the last operation is indicated by a value of 'C'. When the underlying data is updated, the delta row is updated (instead of a second row being inserted) with the last operation changed to 'U'. Again, a way of uniquely identifying each row in the cache data table is required.

Either of these methods may be altered / updated to fit other types of data change detection.

Caching Data Source Targets

The cache management framework currently supports the following cache target databases:

- Oracle 11g, 12c
- SQL Server 2008, 2012, 2014, 2016, Azure SQLpass
- Azure Synapse Analytics (SQL Server DW)
 - Cache tables only. The cache_status, cache_tracking and cache framework tables must be in SQL Server or Azure SQLPass.
 - Caching support: Full Single and Full multi-table.
- Netezza 7 [Merge not supported therefore only inserts are supported.]
- MemSql
- Postgres 9.1
- Other databases can be supported by engaging a Professional Services engineer. Depending on the requirements, it could take 2-4 weeks to do the port and test. Appendix D provides the background on what is required and what will be done.

Unsupported Features

The following features are not currently implemented.

Partitioned Incremental Caching

Partitioned incremental caching is similar to incremental caching, but instead of caching the entire data set, only part of the data is cached and the rest of the data is accessed from the original system(s) of record on demand.

As an example, the cost to cache over 100 years of data cannot be justified when older data will be infrequently accessed. For this reason, a view of such data can be partitioned in such a way that only the most recent data is cached. To implement partitioned caching, some additional components are needed to allow a user to seamlessly and transparently query data from both halves of the partitioned data.

In the current solution, the partitioning is accomplished using a named "activity date" column and the newer data is cached. With some tweaking of the code, this behavior can be altered to fit other partitioning needs.

Historical

Historical data refers to system of record data that was cached but is no longer available in the system of record. This type of data can be transient in nature where it only lives for a short period of time in the system of record. It may also be data that gets purged from the system of record over a period of time. Currently the Cache Framework has no way of handling the backup of this historical data in the event of a deployment that is failed on an incremental cache. Currently, the Cache Framework must be able to recover a full cache refresh from the system of record in the event of a failed cache deployment or failed refresh for an incremental cache.

Schema Differences

Currently, the Cache Framework can only support the same cache database data source name, catalog name and schema name in all deployment environments. In other words, the DV path must be exactly the same in all environments. The issue is that when the DV path to the cache database is different in the target deployed environment, it causes the cache to be marked as failed. For incremental caches, this is not the desired behavior as this invalidates the cache.

Caching Terms

- 1) Cache Target – The cache target is one of the supported cache databases.
- 2) Application – The application refers to the business line, business area or subject area in which the cache framework will be applied. There is one cache target database per application.
- 3) Context – All procedures within the cache framework are executed using the context of the "constants" procedure which defines the application paths and cache database to be used.
 - a. Additionally, the context is important from a deployment perspective as the CACHING_DATA table is used to track the views to be cached for a given application.

- b. All audit log entries are saved with the context of the organization name and application name.

Business Requirements

The primary requirement is to create a virtual data model using the Data Virtualization (DV) layer. This data model federates and virtualizes data from various data sources servicing specific business areas.

As part of the primary requirements, the virtual model should be able to provide historical information of various source tables. Since all sources of data do not store history, therefore history should to be maintained in DV layer.

Therefore, the caching framework should be able to configure caching on resources

- that are cached in traditional method to improve performance, where traditional approach requires overwrite complete record sets for each cache cycle, or
- configure resources that are cached incrementally to build history.

Technical Requirements

- 1) Provide a generic cache framework that supports various databases for cache targets.
- 2) Support the following cache types for code generation
 - a. **Incremental** (no staging) – incrementally acquire newly inserted rows.
 - b. **Full single table** (uses cachekey) – refresh full cache to a single table.
 - c. **Full multi-table** (no cachekey) – refresh full cache to a round-robin of multiple tables.
 - d. **Hybrid** (incremental + staging)
 - e. **Merge** – ability to merge inserts/updates/deletes from source into incremental cache.
 - f. **Rolling window** – Purges data that falls outside the window. Need to define purge window and frequency of purge.
 - g. **Partition** – ability to reference a cache and live partition at the same time. [Future]
- 3) Support the following cache features
 - a. Distribution columns for Netezza
 - b. Drop/Create indexes
 - c. Execute table statistics
 - d. Migrate non-cache framework views into the cache framework.
 - e. Initial/Delta load procedures for incremental refresh.

- f. Pre- and Post-cache procedure execution for full-table and multi-table refresh.
 - g. Native column options are appended at the end of the column list as in CREATE TABLE tablename (column-list, **column-options**) table-options.
 - h. Native table options are appended after the create statement as in CREATE TABLE tablename (column-list, column-options) **table-options**.
 - i. Native index options options are appended after the create index as in CREATE INDEX indexname ON tablename (column-list) **index-options**.
 - j. Cache Schedule (Manual and Scheduled)
 - i. Create cache schedule using period within view
 - ii. Create separate trigger to execute “RefreshCache” procedure
 - iii. Cache policies can be used with full cache only.
- 4) Provide a general audit logging framework to correlate messages.
- 5) Ability to initialize the framework
- 6) Ability to de-configure a cache resource
- 7) Ability to deploy a cache resource or all resources by deploying the CachingData procedure along with the cached resources.
- a. Views are not deployed if the all cache resources exist and the view and cache table signatures match. Only deploy cache framework views if necessary.

3 Installation

Requirements for Installation

- Data Virtualization 7.0 with Service Pack 1 or higher installed.
- Data Virtualization 8.0 or higher
- AS Development Utilities version 2019Q301 [Utilities_2019Q301.car] installed.
- A cache database must be created, introspected, and configured for caching. This means that:
 - a. ***Cache status and cache tracking tables should be created and configured on the cache target data source connection.***
 - b. The cache target data source user credentials must have privileges to
 - i. Create, drop and get sequences
 - ii. Create and drop indexes
 - iii. Create and drop tables
 - iv. Execute table statistics

Installation Steps

- If an existing Cache Management folder exists in both /shared/ASAssets/CacheManagement and /services/databases/ASAssets/CacheManagement, take a backup export before you begin.
 - Select each folder using control-click and export both CacheManagement folders to a single backup car file.
- Import Cache Framework base code with overwrite which will wipe out any existing code.
 - Background of what gets imported:
 - When the cache framework solution is imported, the structure /shared/ASAssets/CacheManagement/CacheFramework is created. In the rest of this document <CF_Folder> will be used to reference this location.
 - When the cache framework virtual database is imported, the structure /services/databases/ASAssets/CacheManagement/CacheFramework is created. In the rest of this document <CDB_Folder> will be used to reference this location.

- [1] Create the published database “ASAssets” if it does not exist:
/services/databases/ASAssets
- [2] Right click on the root folder (/) or “Desktop (username)” and import “with overwrite” the cache framework solution
CacheFramework_YYYYQnnn.car.
 - This will include the
/shared/ASAssets/CacheManagement/CacheFramework and
/services/databases/ASAssets/CacheManagement
- (Optional) Import Cache Framework sample code
 - Background of what gets imported:
 - When the cache framework sample is imported, the structure
/shared/ASAssets/CacheManagement/CacheFrameworkSample is created. In the rest of this document <CS_Folder> will be used to reference this location.
 - [1] If you have previously modified the CacheFrameworkSample resources then back them up first.
 - [2] There are 2 options for importing samples.
 - Import (2a) all of the sample code or import (2b) selected ones. Decide which option below to do.
 - [2a] (Option 1) To import “ALL” samples right click on the root folder (/) or “Desktop (username)” and import “with overwrite”
 - [2a.1] Right click on the root folder (/) or “Desktop (username)” and import “with overwrite” the **ALL** cache framework sample solution
CacheFrameworkSample_ALL_YYYYQnnn.car.
 - [2b] (Option 2) To import selected samples follow these steps:
 - [2b.1] Right click on the root folder (/) or “Desktop (username)” and import “with overwrite” the **Netezza** cache framework sample solution
CacheFrameworkSample_Netezza_YYYYQnnn.car.
 - [2b.2] Right click on the root folder (/) or “Desktop (username)” and import “with overwrite” the **Oracle** cache framework sample solution
CacheFrameworkSample_Oracle_YYYYQnnn.car.

- [2b.3] Right click on the root folder (/) or "Desktop (username)" and import "with overwrite" the **SQL Server** cache framework sample solution `CacheFrameworkSample_SqlServer_YYYYQnnn.car`.
 - [2b.4] Right click on the root folder (/) or "Desktop (username)" and import "with overwrite" the **MemSql** cache framework sample solution `CacheFrameworkSample_MemSql_YYYYQnnn.car`.
 - [2b.5] Right click on the root folder (/) or "Desktop (username)" and import "with overwrite" the **Postgres** cache framework sample solution `CacheFrameworkSample_Postgres_YYYYQnnn.car`.
- Set privileges
 - [1] Set privileges on `/shared/ASAssets/CacheManagement` for the group "**all/composite**" as READ EXECUTE SELECT and push down recursively as "make child resources look like this resource".
 - [2] Set privileges on `/services/databases/ASAssets/CacheManagement` for the group "**all/composite**" as READ EXECUTE SELECT and push down recursively as "make child resources look like this resource".
 - Initialize the administrative **interface database**. This is a DV internal data source that is used to point back to DV.
 - Open the data source located at `<CF_Folder>/Scripts/AdminInterface/CIS_Internal` and modify the connection parameters to point back to the DV server that the scripts are installed on. Use an admin user who has rights to execute and copy privileges. Typically, all you need to do is change the port and password for "admin" or provide an equivalent admin type of user.
 - Proceed to "[Configuring an Application Template](#)" to configuring the cache scripts.
 - Proceed to "[Configuring the Cache Framework Sample](#)" for configuring the sample.
 - 1) If not using one of the supported databases as a caching database, please contact your PSG account manager to engage the services of a PSG engineer who may be able to update the solution to use your caching database type.

4 Cache Framework Simple Use Case

The following is a discussion on the simple use case.

Introduction

The cache framework has a lot of capabilities. To harness those capabilities, this is not the section to read. If you want to get started quickly and easily then continue reading. The simple use case discusses the lifecycle of a cache and how to use multi-table cache to get started.

Pre-requisites

Before using the cache framework, a template folder must be copied and configured based on the type of database that will be used for caching.

See Section 6 – [Configure Cache Framework](#)

The database must be setup for caching.

See Section 5 – [Configure a Cache Database](#)

Cache Lifecycle

The cache framework has a lot of capabilities. To harness those capabilities, this is not the section to read. If you want to get started quickly and easily then continue reading. The lifecycle of a cache is as following:

- 1) Configure a view cache – use the multi-table API `ConfigureCache_Multi_Table()`.
- 2) Refresh a cache – use the refresh cache API `RefreshCache()`.
- 3) Deploy a cache – use the deploy cache API `DeployCache()`.
- 4) De-configure a cache – use the de-configure cache API `DeconfigureCache()`.

Now that the cache framework has been configured and a cache database is setup, let's continue walking through the lifecycle.

Configure a view cache

The first step is to identify the view that you want to cache. The API shown below will configure the cache and create the database tables.

What gets created by the cache framework:

- 1) The cache tab is configured on the target view.
 - o Drop indexes check-box on cache tab is checked by default.
- 2) Drop and create 3 cache tables.
- 3) Create indexes if configured on the target view.
- 4) Introspect 3 cache tables.

- 5) Automatically update the “CachingData” procedure used for deployment.
- 6) Automatically update the database table “CACHING_DATA” for tracking cache framework cached views.
- 7) Execute statistics pre-configured by default
- 8) Create/Drop indexes pre-configured by default

What does not get configured by the API ConfigureCache_Multi table:

- 1) Cluster-safe trigger with schedule [advanced feature capable]
- 2) Pre- or Post-procedures [advanced feature capable]

Instructions:

- Using TDV Studio, navigate to the view to be cached.
- Right-click on the view and select copy. This copies the path.
- Execute ConfigureCache_Multi_Table.
- Paste the path into the “inResourcePath” parameter.
- Visit the section shown below for details about tableConstraints and tableOptions.

Execute: <ApplicationFolder>/Execute/ConfigureCache_Multi_Table()

See section: Cache Framework Application Template Folder – [ConfigureCache_Multi_Table](#)

Refresh a cache

The second step is to refresh the cache.

Monitor the cache progress in the following areas:

- 1) RefreshCache output when executed manually.
- 2) Studio – Manager tab – Cached Resources tab
- 3) Studio – Manager tab – Requests tab – Purge Completed Requests to see latest
 - a. Sort by Duration descending to see what has been running the longest

Instructions:

- Using TDV Studio, navigate to the view to be cached.
- Right-click on the view and select copy.
- Execute RefreshCache.
- Paste the path into the “resourcePath” parameter.

Execute: <ApplicationFolder>/Execute/RefreshCache()

See section: Cache Framework Application Template Folder – [RefreshCache](#)

Deploy a cache

The third step is to deploy the cache into a higher environment. The objective is to export the cached view along with its dependent resources. Import the car file into a higher environment which is doing a code promotion. A more detailed explanation of this process can be found in the section “[Cache Framework Deployment](#)”.

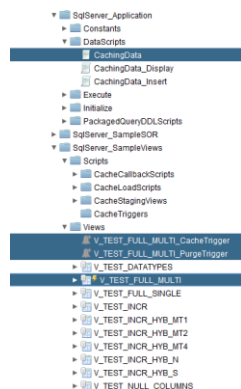
The last requirement is to export the “CachingData” procedure which identifies which resources are under the control of the cache framework. This is how the DeployCache() will know which views to create the tables for and how to create those tables.

The DeployCache() is the main driver script for deploying cache views and tables. The script performs the following:

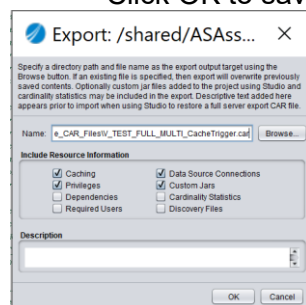
- 1) Execute the CachingData_Insert procedure to insert CachingData() rows into CACHING_DATA table.
- 2) Determine if the cache needs to be deployed based on whether cache tables exist in TDV and the generated view DDL columns and types matches the cache table columns and types excluding the cachekey column if it exists.
 - a. If this is a pre-existing cached view and the tables exist and they match the view signature then no action is taken.
 - b. If deploy cache then perform the following.
 - i. Disable the cache views according to CACHING_DATA query.
 - ii. Invoke ConfigureCacheScript to create the tables but don't configure the views.
 - iii. Update impacted resources.
 - iv. Enable the cache views according to CACHING_DATA query.

Setup LLE Environment:

- Export resources from a lower environment such as development.
 - Select multiple resource using click and control-click
 - Right-click on one of the highlighted resources and “Export”



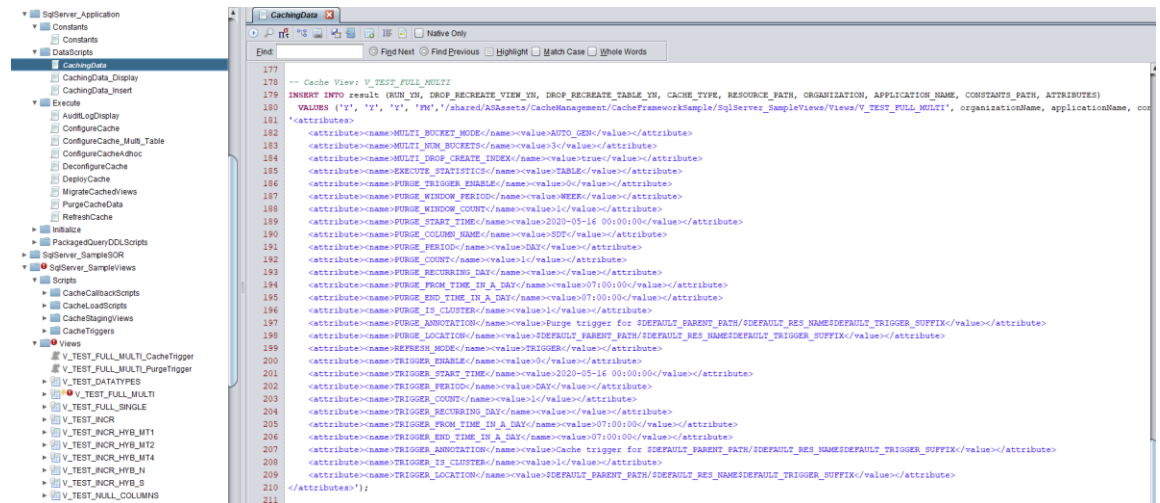
- Brows to a folder and name the .CAR file
 - Uncheck Dependencies
 - Click OK to save – For TDV 8.x add a password



Higher Environment:

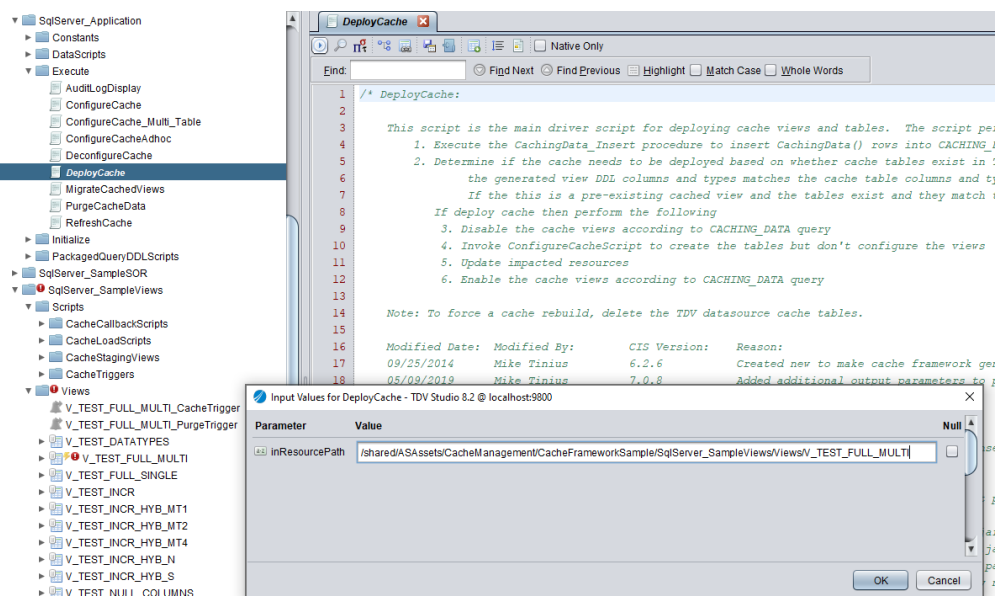
- Import into higher environment

- Right-click on root (/) or Desktop(user) and browse for the .CAR file exported from the development , lower-level environment (LLE).
- Click “Import”
- Initially the cache will be red (impacted) because the cache tables do not yet exist.
- Notice that in the screen shot below, the CacheData procedure is being displayed which shows what was previously configured in the development. It can also be seen that the cache is marked red (impacted).

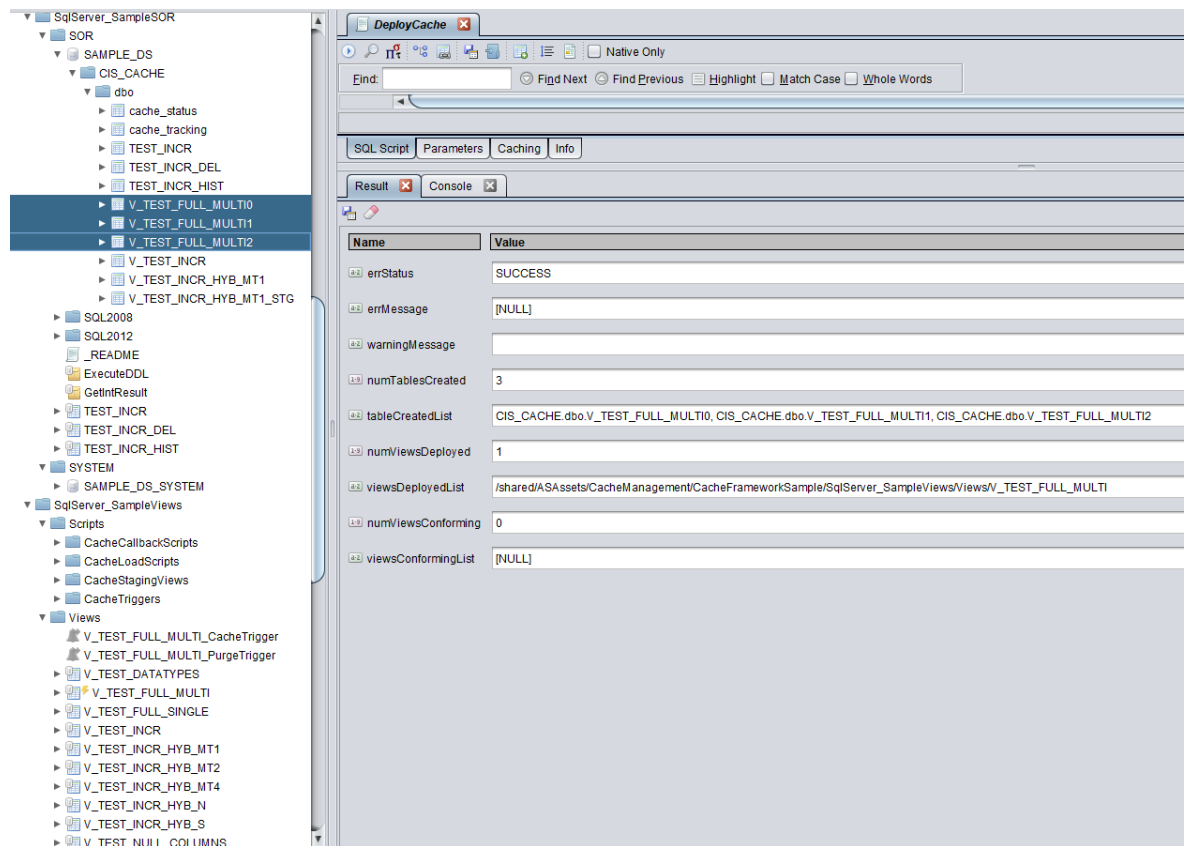


Instructions to deploy a single view:

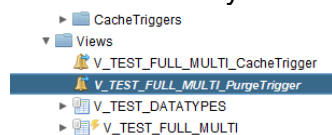
- Using TDV Studio, navigate to the view to be cached.
- Right-click on the view and select copy. This copies the path.
- Execute DeployCache.
- Paste the path into the “resourcePath” parameter.
- The screen shot below represents all of the instructions:



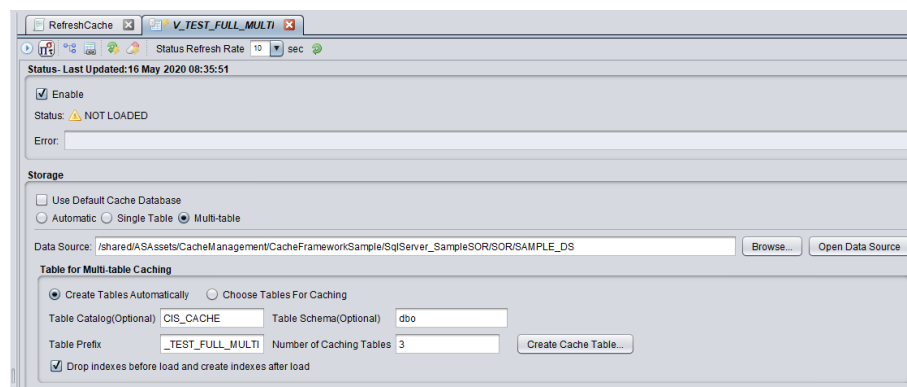
- Upon completion of deployment the view is no longer impacted and the tables have been created:



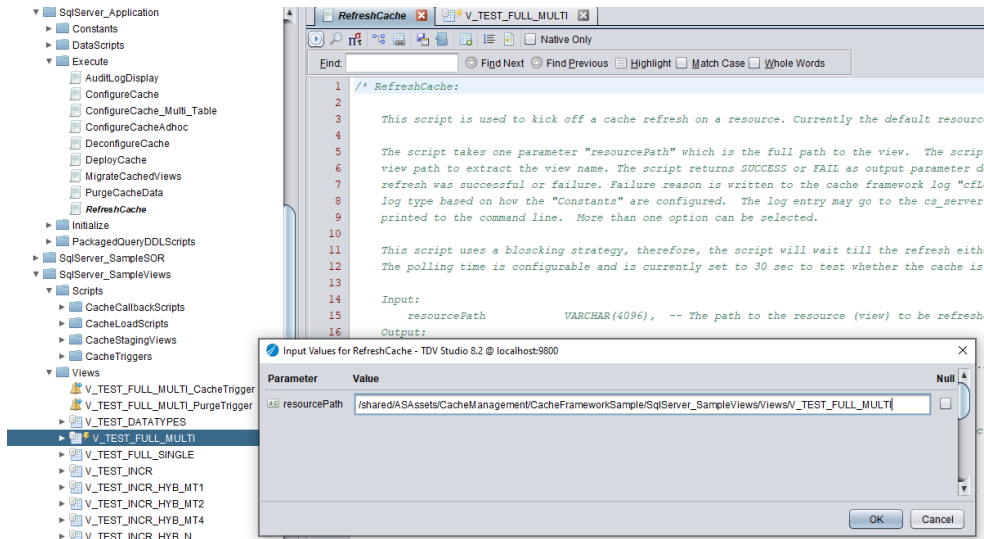
- If the cache and purge triggers were not previously enabled, they will need to be enabled manually.



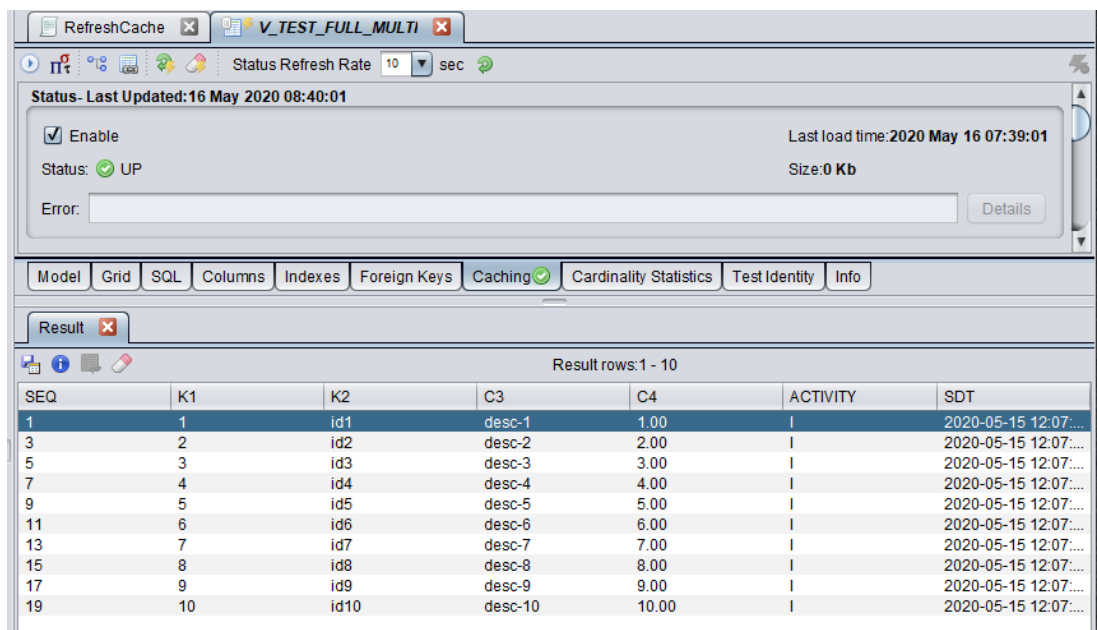
- Currently the cache is not loaded.



- Refresh the cache
 - Right-click on the cache view and select copy to copy the path
 - Open RefreshCache and paste the path into "resourcePath"
 - Click OK



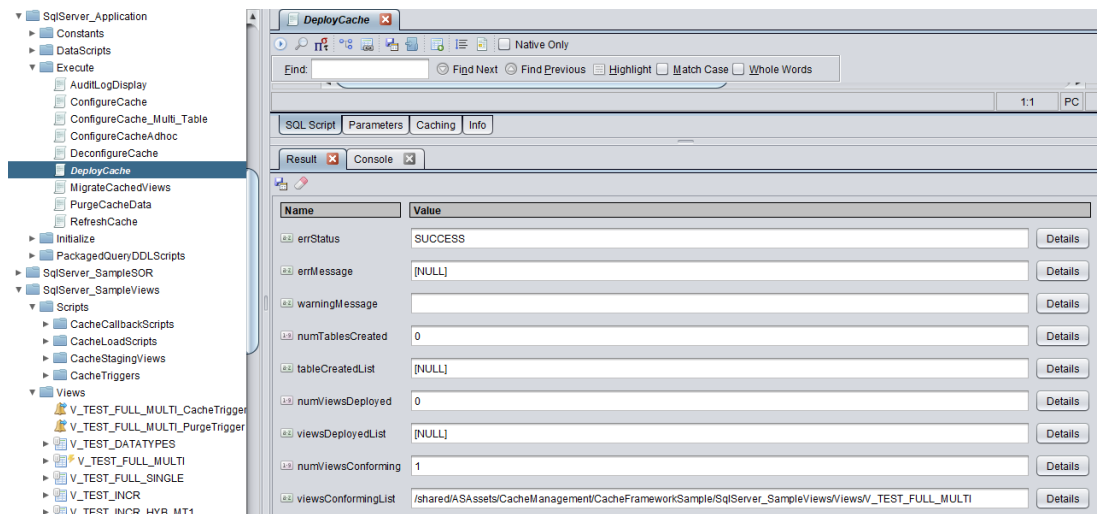
- Refresh Results



Instructions to deploy a all views:

- Using TDV Studio, navigate to the view to be cached.
- Right-click on the view and select copy. This copies the path.
- Execute DeployCache.

- Do “NOT” provide a parameter. The procedure will do a lookup on the CachingData and correlate with the existing view to determine which ones require the tables to be built or rebuilt.
 - If the tables do not exist, they are created.
 - If the table signature is different than the view signature, the tables will be dropped and recreated.
 - If the tables exist and the view signature matches, the tables are NOT recreated. The example below shows what it looks like when the view is conforming to the table signature:



Execute: <ApplicationFolder>/Execute/DeployCache()

See section: Cache Framework Application Template Folder – [DeployCache](#)

De-configure a view cache

The last step is to de-configure a cache. The API will destroy the cache, all of its related cache resources and the associated database tables. If a cached view is being de-commissioned, then the DeconfigureCache must be executed in all environments where it is deployed. If it is not executed, then the database tables will be left intact along with the cache_status table entry.

Instructions:

- Using TDV Studio, navigate to the view to be cached.
- Right-click on the view and select copy. This copies the path.
- Execute DeconfigureCache
- Paste the path into the “inResourcePath” parameter.
- deleteCachingDataEntry – Delete the CachingData procedure entry and the CACHING_DATA table entry
 - Recommended to use 1=Delete the entries.
- deleteCallBackScripts – Delete the callback scripts for any type of cache. Internally, this is known as the firstRefreshCallback and secondRefreshCallback scripts.
 - Recommended to use 1=Delete the scripts.

Execute: <ApplicationFolder>/Execute/DeconfigureCache()

See section: Cache Framework Application Template Folder – [DeconfigureCache](#)

What gets destroyed by the cache framework:

- 1) Disable primary cache view (disable cache)
- 2) If Staging table used
 - a. Destroy staging view
 - b. Destroy CIS staging table
 - c. Drop staging table in database
 - d. Delete the staging table cache_status row
- 3) Destroy CIS cache load or callback scripts. Optional to destroy custom callback implementation scripts.
 - a. Destroy first call back script [Initial Load or Pre-Callback and Pre-Callback Impl (optional)]
 - b. Destroy second call back script [Initial Load or Pre-Callback and Pre-Callback Impl (optional)]
- 4) Destroy CIS cache table
- 5) Drop cache tables in database
- 6) Deconfigure primary cache view
 - a. Destroy cache configuration
 - b. Delete the cache table cache_status row
- 7) Destroy the cache refresh trigger if it exists
- 8) Destroy the cache purge trigger if it exists
- 9) Delete CACHING_DATA entry
- 10) Delete AUDIT_LOG_SEQ entry
- 11) Verify the Deconfiguration

5 Configure Cache Database

The following is a discussion on how to configure a view using the cache database.

Introduction

This section will go into detail about setting up the cache database.

Configure Cache Database Use Current

This section will describe the standard way to setup a cache database where the cache tables, cache_status and cache_tracking are all stored in this database.

The picture below shows the standard way that the cache_status and cache_tracking tables are configured. They use the “**Current data source**”.

The screenshot shows the 'SAMPLE_DS1' configuration window. The 'Cache Status and Tracking Tables' section has two radio buttons: 'Current data source' (selected) and 'Use a different data source'. Below these are fields for 'Status Table' and 'Tracking Table', both pointing to the same path: '/shared/ASAssets/CacheManagement/CacheFrameworkSample/SqlServer_SampleSOR/SOR/SAMPLE_DS1/CIS_CACHE/dbo/cache_status' and '/shared/ASAssets/CacheManagement/CacheFrameworkSample/SqlServer_SampleSOR/SOR/SAMPLE_DS1/CIS_CACHE/dbo/cache_tracking' respectively. The 'Connection Information' section has tabs for 'Basic' and 'Advanced'. The 'Basic' tab is active, showing fields for 'Adapter Name' (Microsoft SQL Server 2012), 'Host' (localhost), 'Port' (1433), 'Database Name' (CIS_CACHE), 'Login' (sa), and 'Password' (masked with asterisks). There are also checkboxes for 'Save Password' and 'Pass-through Login' (Disabled), and dropdown menus for 'Transaction Isolation' (Read Committed) and 'Authentication' (BASIC).

Configure Cache Database User Other

This section describes how to setup a cache database to store the cache tables within but store the cache_status and cache_tracking in a different database. This is not the typical scenario but when the cache database does not support multiple SQL statements per transaction then it will be necessary to “**use a different data source**”.

Azure Synapse Analytics (SQL Server DW) is an example of this situation. The picture below shows the alternative way that the cache_status and cache_tracking tables are configured. They **“use a different data source”**.

The screenshot shows the 'SAMPLE_DS' configuration window. At the top, there are buttons for 'Enable Data Source', 'Add/Remove Resources...', and 'Test Connection'. Below this is the 'Cache Status and Tracking Tables' section, which has two radio buttons: 'Current data source' and 'Use a different data source'. The 'Use a different data source' option is selected, and the text 'rver_SampleSOR/SYSTEM/SAMPLE_DS_SYSTEM' is entered in the field next to it. There are 'Browse...' and 'Open Data Source' buttons next to this field. Below this, there are 'Status Table' and 'Tracking Table' labels with corresponding 'Browse...' buttons. The 'Connection Information' section is also visible, showing fields for 'Adapter Name', 'Host', 'Port', 'Database Name', 'Login', and 'Password'. There are also checkboxes for 'Save Password' and 'Pass-through Login', and dropdown menus for 'Transaction Isolation' and 'Authentication'.

Depending on the capabilities of the target cache database, it may be necessary to set the transaction isolation level to “NONE” as shown in the picture below. For SQL Azure Synapse Analytics (SQL Server DW) this will be required due to the nature of how Azure Synapse Analytics (SQL Server DW) accepts DDL transactions to CREATE/DROP objects.

This is a close-up of the 'Transaction Isolation' dropdown menu. The text 'Transaction Isolation:' is on the left, and the dropdown menu itself shows 'None' as the selected option.

Configure Native Loading

This section describes how to configure TDV and the data source to use native loading.

Each database is different. It is always advised to check the latest capabilities of a database to see what is supported and what is not. A good example of this is that the regular SQL Server supports BCP on both Windows and Linux platforms. However,

SQL Server Azure SQLPass and Azure Synapse Analytics (SQL Server DW) do not support BCP on Linux but they do support it on Windows.

Oracle uses DB-Links for high-speed loading of data between different oracle databases. However, to load data between non-Oracle and Oracle requires the Oracle Gateway product which may cost extra.

The point is clear that it is important to understand your platform environment, your data load requirements and ultimately how fast you want to load data.

The example that is provided below will be for SQL Server on an operating system that supports BCP.

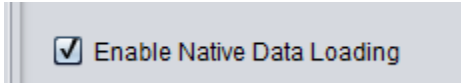
TDV Administration Configuration parameters that should be set:

- **Enable Native Loading** – [true] – Indicates whether cache loading using native mechanisms, such as Oracle Database Links, is enabled.
- **Enable Parallel Loading** – [true] – Indicates whether parallel cache loading is enabled.
- **Enable Bulk Data Loading** – [true] – Setting to true will:
 1. Allow INSERT/SELECT statements inserting rows into Netezza, Teradata, DB2 or SQL Server to use bulk loading methods.
 2. Allow procedures cached into Netezza, DB2 or SQL Server to use bulk loading methods.
- SQL Server Data Source settings
 - **Microsoft BCP utility path** – [C:\Program Files\Microsoft SQL Server\Client SDK\ODBC\170\Tools\Binn\bcp.exe]
 - The absolute file path of the Microsoft BCP utility.
 - This value is locally defined. It will not be altered when restoring a backup and will not be replicated in a cluster.
 - **Column Delimiter** – [|~|]
 - column delimiter character to be used for the BCP utility while loading data. If not defined, the value in Configuration > Data Sources > Common to Multiple Source Types > Data Sources Data Transfer > Column Delimiter will be used.
- Common Data Source settings
 - **Buffer Flush Threshold**. The minimum value is 1000. – [50000]
 - Certain types of data ship SQL executions might buffer very large tables before delivery to the data ship target. This configuration parameter can be used to limit the size of the buffer. The buffer is flushed when this limit is reached.

- **Default Commit Row Limit – [50000]**
 - When a cache to database table is processing, this value specifies a row count for a commit to occur while inserting cache data. The transaction will be committed every time after the number of rows that the cache data is inserted has reached this limit.

Data source Settings:

- Data source Advance Tab – Check the box to “**Enable Native Data Loading**”.



- A note about “**Enable Native Data Loading**”.
 - For SQL Server data source, there is a scenario where the BCP utility is not configured but checking the box will still allow an increase in performance of loading the data when the cache table is in the same data source as the source tables.
 - In this scenario, the objective is to take very complex views that take a long time to return data and cache them. When the “Enable Native Data Loading” is checked, TDV will recognize the internal “INSERT SELECT” statement is coming from the same data source as the cache table and issue that command internally in the database. This is very fast. Without this, TDV will SELECT the data into TDV in small batches and use JDBC to write them out to the cache table. This will take a lot longer.

6 Configure Cache Framework

The following is a discussion on how to configure a view using the cache framework.

Introduction

The concept of porting an application template is to provide the ability to generate and deploy DV cache objects. The application template provides all of the necessary hooks into the generic framework code for implementing and deploying cache resources. It is important to understand that the application sets the context. There are different perspectives that can be considered when defining the context.

- 1) **Subject Area context** – Conceptually, this is the better choice irrespective of whether there is one cache database or multiple. By having an application configured for each subject area, it allows the audit log messages to be customized with same “*Organization Name*” and “*Application Name*”. In this scenario, the application will only see their audit log messages because the other messages are filtered out. It also allows the application users to control their own list of cache resources separate from other applications within the CACHING_DATA table.
- 2) **Target cache database context** – Technically speaking only one context is required for each cache database being referenced in DV. This is true because the paths to the cache database are referenced in the “Constants” procedure. However, this also means that all audit log messages will have the same Organization and Application Name and the list of cache resources will be within a single context thus there is no way to separate out the notion of different subjects.
- 3) It is strongly advised that the view to be cache have a primary key index created on it (that includes the cache key column.) In any case, a column or combination of columns is required in order to uniquely identify a row of cached data.
- 4) If the “last updated” type of incremental caching is going to be used, then an index on the “last updated” column(s) would be advisable as well.

Configuring an Application Template

Configuring a template is a one-time action that should be done by a DV architect with overall knowledge of the application and cache database target.

1) Copy Application Template

- a. Copy the application template applicable for your cache database type:

- b. Oracle: <CF_Folder>/ApplicationTemplateOracle
- c. SQL Server: <CF_Folder>/ApplicationTemplateSqlServer
- d. Netezza: <CF_Folder>/ApplicationTemplateNetezza
- e. MemSql: <CF_Folder>/ApplicationTemplateMemSql
- f. Postgres: <CF_Folder>/ApplicationTemplatePostgres
- g. Copy and paste the folder into your own application project folder.
 - i. Resources will automatically rebind to your folder location.
 - ii. It is “highly” recommended to copy the cache framework template within the top-level project folder so that the correct privileges are “pushed” down from that project folder allowing the developers to be able to execute the necessary API’s. Additionally, the cache framework may create additional resources such as callback scripts, load scripts and staging views that all require proper privileges for users to be execute.
- h. Rename the target “TG” folder to meet your needs which shall be known as <TG_Folder> throughout this document. For example, ApplicationTemplateOracle becomes CacheOracleApplication or whatever name it is given.

2) Modify Constants

- a. Location: <TG_Folder>/Constants/**constants**
- b. Minimum constants to modify:
 - i. **ApplicationBasePathConstant** – the full path of your new application template folder.
 - ii. **SystemConstantsType** – Modify this if the cache database uses a different database for the cache_status and cache_tracking tables which are known as the “system” tables. In addition to the cache status and tracking, the cache framework tables are also created in this database.
 - iii. **DEBUG_LEVEL** – Set for DEBUG_LEVEL_INFO but it is recommended to set it to DEBUG_LEVEL_DEBUG until you are comfortable with how everything is working.
 - iv. **DEBUG_LOGGING_TYPE** – Set to save to DB (AUDIT_LOG) and PRINT to the command line. Remove PRINT to disable printing to command line. Recommended: PRINT only.
 - v. **OrganizationName** – The organization such as “MyOrganization”
 - vi. **ApplicationName** – Your application name. Recommend staying with architectural concepts such as Business Line/Business

Area/Subject Area so that audit log messages can be correlated with folders in DV. Functional concept. The AUDIT_LOG table may be shared across various organizations and applications when they are using the same cache database.

- vii. **ApplicationScriptsBasePath** – The DV scripts base path for this application. Developer must have privileges to create and delete resources. The default is ApplicationBasePathConstant|| '/Scripts' but it could be a hard-coded path. It is recommended that it be inside the project folder so that privileges can be applied appropriately.
- viii. **DefaultCacheDSPath** – The DV path to the cache data source. Set the actual path to the data source.
- ix. **DefaultCacheDSCatalogName** – The name of the catalog if applicable or NULL. For example, SQL Server will require a catalog.
- x. **DefaultCacheDSSchemaName** – The name of the schema if applicable or NULL. For example, MemSql is null.
- xi. All other variables are optional and can be configured as needed. Some examples to consider otherwise use the defaults as they are set in the Constants procedure.
 - 1. **DefaultCacheTablePrefix** – NULL or short string like 'C_' to signify a cache database table prefix
 - 2. **DefaultStageTablePrefix** – A short string value such as 'S_' or 'STG_' to signify a staging table prefix.
 - 3. **DefaultCacheTableSuffix** – NULL or short string like '_C' to signify a cache database table suffix. This is not recommended.
 - 4. **cachePollingInterval** – Currently set to 5000 milliseconds which is 5 seconds. Frequency (in ms) to poll for the cache refresh outcome.
- xii. **DefaultSystemDSPath** – The DV path to the system data source. Set the actual path to the system data source where the cache_status, cache_tracking and cache framework tables will be stored. This is used when the cache database is setup to “**Use a different data source**” for cache_status and cache_tracking.
- xiii. **DefaultSystemDSCatalogName** – The name of the system datasource catalog if applicable or NULL. For example, SQL Server will require a catalog.
- xiv. **DefaultSystemDSSchemaName** – The name of the system datasource schema if applicable or NULL. For example, MemSql is null.

3) Initialize All Scripts

- a. Note: This only needs to be done during the initial configuration of the Cache Framework application template scripts.
- b. Location: <TG_Folder>/Initialize
- c. **Execute All Scripts** – This will execute all initialization scripts in order.
 - i. **CAUTION:**
 1. If multiple “application” folders are sharing the same cache database, then this initialization step only should be done once for the configuration of the first application. Any subsequent execution of these scripts will result in dropping and recreating existing database objects and thus “DELETING” other application metadata.
 2. If this is the case, then only execute the “RebindPackagedQueries” script for each new cache application folder that uses the same cache database as other applications.
 - ii. Input: Drop_Tables_Before_Creating
 1. 1=drop table/sequence before creating.
 2. 0=do not drop table/sequence before creating
- d. **Note: The above script will execute the following individual scripts:**
 - i. Rebind the packaged queries to your cache data source
 1. Location: <TG_Folder>/Initialize
 2. ***RebindPackagedQueries*** – Execute to rebind the two packaged queries found in /PackageQueryDDLScripts. There are no input parameters. The information for rebinding is contained within the Constants procedure.
 - a. ExecuteDDL (all cache database types)
 - b. ExecuteDDLSystem (all system database types)
 - c. GetIntResult (all cache database types)
 - d. GetIntResultSystem (all system database types)
 - e. ShowIndexes (MemSql)
 - f. ShowTables (MemSql)
 - ii. Initialize database tables and Sequence
 1. **CAUTION:**

- a. If multiple “application” folders are sharing the same cache database, then this initialization step only has to be done once for the configuration of the first application. Any subsequent execution of these scripts will result in dropping and recreating existing database objects and thus “DELETING” other application metadata.
2. Location: <TG_Folder>/Initialize
3. Database privileges required:
 - a. Create/Drop Sequence
 - b. Create/Drop Table
 - c. Create/Drop Index
4. **CreateDBSequence** : Create DB sequence
CACHE_FRAMEWORK_SEQ. Does not drop automatically.
5. **Create_AUDIT_LOG** : Will automatically drop if exists.
6. **Create_AUDIT_LOG_SEQ** : Will automatically drop if exists.
7. **Create_CACHING_DATA** : Will automatically drop if exists.
8. About DROP – Use these if you want to decommission the Cache Framework entirely from a target cache database.
 - a. Drop_CACHING_DATA : Drop CACHING_DATA table
 - b. Drop_AUDIT_LOG_SEQ : Drop AUDIT_LOG_SEQ table
 - c. Drop_AUDIT_LOG : Drop AUDIT_LOG table
 - d. DropDBSequence : Drop the DB sequence
CACHE_FRAMEWORK_SEQ

4) Migrate Pre-existing Cached Views

- a. This is [optional]
- b. Location: <TG_Folder>/Execute
- c. **MigrateCachedViews** – This script provides a way to migrate views into the Cache Framework that are not under the control of the Cache Framework. Being under control of the Cache Framework means having a row in the CACHING_DATA table for each cached view and an equivalent row in the "CachingData" procedure.
 - i. Given a starting path, this procedure will interrogate each view and determine if it is currently in the Cache Framework or not. If not, it will add the view to the Cache Framework. Alternatively, the user

may point directly to a cached view and only bring in that one view into the Cache Framework.

- ii. After completion of this procedure, if there are views that you do not want in the Cache Framework, then remove them from the "CachingData" procedure and execute "CachingData_Insert" to remove and insert the remainder of the views in "CachingData".

5) Modify CachingData Procedure

- a. This is done automatically when using ConfigureCache_Multi_Table procedure.
- b. Location: <TG_Folder>/DataScripts
- c. **CachingData** – A way to track and configure the list of cached views that are controlled by the Cache Framework.
 - i. CachingData is a portability mechanism. It is used as a way to migrate the known list of views in the Cache Framework to an upper environment. Anytime a new view is brought into the cache framework and then migrated to an upper environment, the CachingData procedure should also be exported and migrated. This procedure will work in conjunction with DeployCache in the upper environments.
 - ii. Migrate cached views into the Cache Framework will automatically modify the CachingData procedure and update the CACHING_DATA database table.
 - iii. ConfigureCache_Multi_Table will automatically modify the CachingData procedure and update the CACHING_DATA database table as it configures the view for multi-table and a pre-callback and post-callback procedure.
 - iv. After manually editing CachingData, execute CachingData_Insert to update the CACHING_DATA database table.
 - v. Details on configuration settings can be found in the header of the "CachingData" procedure or in the section of the document: ["Cache Framework Features and Attributes"](#).
- d. **CachingData_Insert** – Execute to load rows into CACHING_DATA table from the CachingData procedure.
 - i. Parameter: 0=Delete and insert only the rows found in CachingData procedure. 1=Delete all rows and then insert CachingData. In both cases, the delete and insert is done in the context of Organization Name and Application Name.

6) Configure the View Cache

- a. Location: <TG_Folder>/Execute

- b. **Important:** If a view is already cached and you execute any of the variations of `ConfigureCache...`, it will drop all resources and tables and rebuild the cache thus wiping out all of the data.
- c. **ConfigureCache** – Use in conjunction with `CACHING_DATA` table.
 - i. Leave `inResourcePath` blank to configure all views specified in `CACHING_DATA` or enter a specific resource path to configure only that one.
 - ii. Use a wildcard "%" at the end of the path to signify any text matching the text preceding the wildcard symbol "%".
- d. **ConfigureCacheAdhoc** – Use when needing to configure a cache in an adhoc way and don't use `CACHING_DATA` table.
 - i. Must supply all parameters for this procedure that would normally be supplied via `CACHING_DATA`.
 - ii. This capability bypasses both `CachingData` procedure and `CACHING_DATA` database table and does not save any of the cache metadata into the Cache Framework. Once the cache is created, it will not be under the control of the Cache Framework. Therefore, capabilities like `DeployCache` will not be able to be used.
- e. **ConfigureCache_Multi_Table** – Auto-configure `CACHING_DATA` table and view at the same time for multi-table caching.
 - i. `inResourcePath` – The full path to the view to be configured for multi-table caching.
 - ii. `tableConstraints` – Table Constraint(s) that can be used to qualify the table. Null if no table constraints. Database specific syntax that succeeds the column list and within the right parenthesis. `CREATE TABLE "tablename" ({ColumnList}, {TableConstraints}) {TableOptions}`
 - 1. Separators: Each column in the table constraints list must contain a valid begin and end separator.
 - a. Oracle, Netezza, Postgres `begin=" end="`
 - b. SqlServer `begin=[end=]`
 - c. MemSql `begin=` end=``
 - d. MemSql: Format: `key(<sortkeylist>) using clustered columnstore, shard key(<shardlist>)` Example: `key(`K1`, `K2`) using clustered columnstore, shard key(`S1`, `S2`)`
 - iii. `tableOptions` – Table option(s) that can be used to qualify how the table is created. Null if no table options. Database specific syntax that succeeds the column list and is outside the right parenthesis. `CREATE TABLE "tablename" ({ColumnList}, {TableConstraints}) {TableOptions}`

1. Separators: Each column in the table option list must contain a valid begin and end separator.
 - a. Oracle, Netezza, Postgres begin=" end="
 - b. SqlServer begin=[end=]
 - c. MemSql begin=` end=`
 - d. Netezza: Format: DISTRIBUTE ON (<keylist>) Example: DISTRIBUTE ON ("C1", "C2")
 - e. MemSql Example: AUTOSTATS_ENABLED
 - f. Azure Synapse Analytics (SQL Server DW) Example: WITH (DISTRIBUTION = hash("id"), CLUSTERED COLUMNSTORE INDEX)
- iv. `additionalAttributes` – Additional XML attributes to specify the cache definition.
 1. Example: trigger attributes.
 - a. IF the keyword: `CACHE_DEFAULT_TRIGGER` is used then a cache trigger will be created using default attributes. Modify `/DataScripts/Caching/Data` and `<ViewName>_CacheTrigger` with more accurate schedule information.
 - b. IF the keyword: `PURGE_DEFAULT_TRIGGER` is used then a purge trigger will be created using default attributes. Modify `/DataScripts/Caching/Data` and `<ViewName>_PurgeTrigger` with more accurate schedule information and purge column in the procedure attributes section.
 - c. IF XML `<attribute>` is used then it will be taken as a literal string and used. If any syntax is wrong, it may throw an exception.
 - v. Wildcards are not permitted in the path. Only one view at a time may be configured.
 - vi. It configures the view for three cache tables.
 - vii. It configures the view for a pre-callback and post-callback procedure.
 - viii. It configures the view to automatically drop and create indexes during a refresh.
 - ix. It does not configure a trigger or cache schedule.

7) View `AUDIT_LOG`

- a. This is only applicable if the constants `DEBUG_LOGGING_TYPE` contains "DB".
- b. Location: `<TG_Folder>/Execute`

- c. **AuditLogDisplay** – Displays the results of AUDIT_LOG for your “Organization Name” and “Application Name” in descending order or so the latest message is displayed first. It filters out other organizations and applications.

8) Refresh the Cache

- a. Location: <TG_Folder>/Execute
- b. **RefreshCache** – Enter the path of the view to refresh.
- c. Run AuditLogDisplay to view AUDIT_LOG

9) Deploy the Cache

- a. Location: <TG_Folder>/Execute
- b. **DeployCache** – This script is the main driver script for deploying cache views and tables.
 - i. **Assumptions:** The CachingData procedure contains a catalog of all of the cached views. It must be exported (migrated) to the upper environments in order for DeployCache to know what it can deploy from the catalog and how those cached views are configured.
- c. The script performs the following:
 - i. Execute the CachingData_Insert procedure to insert CachingData() rows into CACHING_DATA table
 - ii. The script determines if the cache needs to be deployed based on whether cache tables exist in TDV and the generated view DDL columns and types matches the cache table columns and types excluding the cachekey column if it exists. If deploy cache then perform the following:
 1. Disable the cache views according to CACHING_DATA query
 2. Invoke ConfigureCacheScript to create the tables but don't configure the views
 3. Update impacted resources
 4. Enable the cache views according to CACHING_DATA query
 - iii. Input: inResourcePath
 1. The specific view path to be configured or leave null to configure all views in CachingData (procedure) / CACHING_DATA (table). NOTE: It is recommended to test that this works on one configured view.
 2. This path acts as a filter for what is already configured in CACHING_DATA so the view must have already been configured in CACHING_DATA.
 3. Use a wildcard "%" at the end of the path to signify any text matching the text preceding the wildcard symbol "%".

4. If a container path is passed in with no wildcard "%", then the wildcard will be added automatically so that only view paths within this path range are deployed if applicable.
- d. Run AuditLogDisplay to view AUDIT_LOG

10) De-configure the Cache

- a. Location: <TG_Folder>/Execute
- b. **Deconfigure** – De-configure a cache or list of cache resources found in CACHING_DATA.
 - i. It is recommended to test that this works on one configured view.
 - ii. Enter the deleteCachingDataEntry value.
 1. 0=Do not delete the CachingData procedure entry and the CACHING_DATA table entry.
 2. 1=Delete the CachingData procedure entry and the CACHING_DATA table entry.
 - iii. Enter the path of the view to de-configure.
 1. If left NULL, then de-configure all cache resources found in CACHING_DATA filtered by Organization Name and Application Name.
 - iv. Run AuditLogDisplay to view AUDIT_LOG

7 Cache Framework Features

The following is a discussion on cache framework features.

Log to a generic audit table script

- 1) Supported and tested for all cache database targets.
- 2) cfLog – the main logging procedure invoked by all cache framework procedures. This procedure provides the framework for the various debug levels. This procedure is an interface procedure and does not implement the actual logging itself.
- 3) The generic “auditLogger” script is the implementation for the different debug logging types.
- 4) Debug Levels implemented
 - a. Various procedures in the Cache Framework implement different levels as shown below. The user can control which levels get logged simply by setting `DEBUG_LEVEL` in the Constants file.
 - i. `DEBUG_LEVEL_ERROR=0` - Output error message to log file with severity level `ERROR`. No command line print.
 - ii. `DEBUG_LEVEL_AUDIT=1` - Output audit message to log file with severity level `INFO`. No command line print.
 - iii. `DEBUG_LEVEL_INFO=2` - Output info message to log file with severity level `INFO`. No command line print.
 - iv. `DEBUG_LEVEL_DEBUG=3` - Output debug message to log file with severity level `INFO`. No command line print.
 - b. The user can control the debug logging type which specifies where to write the message to. The user can set `DEBUG_LOGGING_TYPE` to one or more options include `[LOG, DB, PRINT, EMAIL]`.
 - i. `LOG` – outputs message to the DV log file `cs_server.log`.
 - ii. `DB` – outputs message to the `AUDIT_LOG` table within the cache database that is supported.
 - iii. `PRINT` – outputs message to the Studio command line. This is useful for manual execution and debugging.

- iv. EMAIL – outputs message to an email. Must configure email in the DV Studio Administration, Configuration: /Composite Server/Configuration/E-Mail.

Cache Types

- 1) “FS” = Full Single table cache
- 2) “FM” = Full Multi-table cache
- 3) “IN” = Incremental cache (no staging)
- 4) Hybrid (staging + incremental)
 - a. “HS” = Hybrid with Staging for Initial and Delta (same script)
 - i. HS = Hybrid Delta Stage
 - b. “HN” = Hybrid with Staging for Initial and NO staging for Delta (different scripts)
 - i. HN = Hybrid Delta No-Stage
 - c. “MT1” = Merge Type 1 (Staging for initial script and staging for delta script)
 - i. MT1=Merge Type 1 See [Merge Type 1 Example](#)
 - d. “MT2” = Merge Type 1 (Staging for initial script and staging for delta script)
 - i. MT2=Merge Type 2 See [Merge Type 2 Example](#)
 - e. “MT4” = Merge Type 1 (Staging for initial script and staging for delta script)
 - i. MT4=Merge Type 4 See [Merge Type 4 Example](#)
- 5) Not implemented
 - a. Partition – ability to have some data cached and some data live via a UNION

Cache Features and Attributes

Distribution Columns

- 1) **Distribution Column** support for Netezza – supported
 - a. CONFIGURE: Netezza Distribution Column Attribute
 - b. **DISTRIBUTION_COLUMNS**: Column(s) that can be used to create distribution key on NZ.
 - i. `<attribute><name>DISTRIBUTION_COLUMNS</name>
<value>NULL</value></attribute>`

Multi-Table Configuration

2) Multi-Table configuration attributes

- a. **CONFIGURE**: Full Multi-Table Caching Attributes:

- b. **MULTI_BUCKET_MODE**: AUTO_GEN or MANUAL. Only AUTO_GEN is supported at this time.

```
<attribute><name>MULTI_BUCKET_MODE</name>
<value>AUTO_GEN</value></attribute>
```

- c. **MULTI_BUCKET_TABLE_PREFIX**: Provide an explicit name or the \$DEFAULT_RES_NAME is extracted dynamically from the resource being cached.

```
<attribute><name>MULTI_BUCKET_TABLE_PREFIX</name>
<value>$DEFAULT_RES_NAME</value></attribute>
```

Allowed Variables

\$DEFAULT_RES_NAME – The \$DEFAULT_RES_NAME is the name of the view and is extracted dynamically from the resource being cached.

- d. **MULTI_NUM_BUCKETS**: Default is 3. The number of cache table buckets to create by appending 0, 1, 2... to the end of the bucket table prefix.

```
<attribute><name>MULTI_NUM_BUCKETS</name> <value>3</value></attribute>
```

- e. **MULTI_DROP_CREATE_INDEX**: true/false to drop and create any indexes associated with the cache view. Netezza must be false. MemSql must be false if column store indexes are used.

```
<attribute><name>MULTI_DROP_CREATE_INDEX</name>
<value>true</value></attribute>
```

- f. Optional Configure: Multi-table Cache Pre/Post-Refresh Cache Procedures.

- g. **FULL_PRE_CALLBACK_IMPL_PROC**: [optional] Explicit path to the **pre-refresh** callback implementation procedure or use the variable \$APPLICATION_CACHE_PROC_IMPL which defaults to the CONSTANTS variable "ApplicationCacheProcPath" which is a path to the CacheCallbackScripts folder.

```
<attribute><name>FULL_PRE_CALLBACK_IMPL_PROC</name>
<value>$APPLICATION_CACHE_PROC_IMPL</value></attribute>
```

Allowed Variables:

\$APPLICATION_CACHE_PROC_IMPL – The default application cache proc path defined in CONSTANTS along with a generated procedure name.

- h. **FULL_POST_CALLBACK_IMPL_PROC**: [optional] Explicit path to the **post-refresh** callback implementation procedure or use the variable \$APPLICATION_CACHE_PROC_IMPL which defaults to the CONSTANTS variable “ApplicationCacheProcPath” which is a path to the CacheCallbackScripts folder.

```
<attribute><name>FULL_POST_CALLBACK_IMPL_PROC</name>
<value>$APPLICATION_CACHE_PROC_IMPL</value></attribute>
```

Allowed Variables:

\$APPLICATION_CACHE_PROC_IMPL – The default application cache proc path defined in CONSTANTS along with a generated procedure name.

Incremental and Hybrid Configuration

3) Incremental and Hybrid Load script configuration

- a. Name type – format:

- i. “HS” – Hybrid Staging

1. VIEWNAME_CacheLoad

- ii. “HN” – Hybrid No Staging on Delta

1. VIEWNAME_CacheInitialLoad

2. VIEWNAME_CacheDeltaLoad

- iii. “IN” – Incremental

1. VIEWNAME_CacheInitialLoad

2. VIEWNAME_CacheDeltaLoad

- b. CONFIGURE: Incremental Caching Attributes:

- c. **INCR_KEY_NAME**: Incremental cache column name.

```
<attribute><name>INCR_KEY_NAME</name><value>UPD</value></attribute>
```

- d. **INCR_KEY_TYPE**: Incremental cache column data type for the above column.

```
<attribute><name>INCR_KEY_TYPE</name><value>TIMESTAMP</value> </attribute>
```

- e. **INCR_KEY_STARTING_VALUE**: Incremental cache initial default value if history needs to be loaded.

```
<attribute><name>INCR_KEY_STARTING_VALUE</name><value>1900-01-01
00:00:00</value></attribute>
```

- f. **INCR_INITIAL_LOAD_SCRIPT**: [optional] Explicit path to the incremental cache initialization refresh procedure or use the variable \$APPLICATION_CACHE_INCR_SCRIPTS which defaults to the CONSTANTS variable “ApplicationIncrScriptsPath” which is a path to the CacheLoadScripts folder.

```
<attribute><name>INCR_INITIAL_LOAD_SCRIPT</name>
<value>$APPLICATION_CACHE_INCR_SCRIPTS</value></attribute>
```

Allowed Variables:

\$APPLICATION_CACHE_INCR_SCRIPTS – The default application cache incremental scripts path defined in CONSTANTS along with a generated procedure name.

- g. **INCR_DELTA_LOAD_SCRIPT**: [optional] Explicit path to the incremental cache delta refresh procedure or use the variable \$APPLICATION_CACHE_INCR_SCRIPTS which defaults to the CONSTANTS variable “ApplicationIncrScriptsPath” which is a path to the CacheLoadScripts folder.

```
<attribute><name>INCR_DELTA_LOAD_SCRIPT</name>
<value>$APPLICATION_CACHE_INCR_SCRIPTS</value></attribute>
```

Allowed Variables:

\$APPLICATION_CACHE_INCR_SCRIPTS – The default application cache incremental scripts path defined in CONSTANTS along with a generated procedure name.

Incremental Merge Configuration

4) Merge configuration attributes

- a. Name type – format:

- i. “MT1” – [Merge Type 1](#)

1. VIEWNAME_CacheInitialLoad
2. VIEWNAME_CacheDeltaLoad

- ii. “MT2” – [Merge Type 2](#)

1. VIEWNAME_CacheInitialLoad
2. VIEWNAME_CacheDeltaLoad

- iii. “MT4” – [Merge Type 4](#)

1. VIEWNAME_CacheInitialLoad
2. VIEWNAME_CacheDeltaLoad

- b. **CONFIGURE**: Incremental Merge Caching Attributes:

- c. **INCR_MERGE_CACHE_VIEW_PKCSV**: Incremental Merge - This is a comma separated list of the names of the "primary key" of the cache view.
- ```
<attribute><name>INCR_MERGE_CACHE_VIEW_PKCSV</name>
<value>K1,K2</value></attribute>
```
- d. **INCR\_MERGE\_HISTORY\_TABLE\_PATH**: Incremental Merge Type 1, 2, 4  
- The DV path to the table/view containing history information about rows that were inserted, updated or deleted from the source table.
- ```
<attribute><name>INCR_MERGE_HISTORY_TABLE_PATH</name>
<value>/shared/.../TEST_INCR_HIST</value></attribute>
```
- e. **INCR_MERGE_HISTORY_TABLE_PKCSV**: Incremental Merge Type 1, 2, 4 - This is a comma separated list of the names of the "primary keys" of the history table. These columns need to match 1 for 1 in order with the key values in the Cache view. NOTE: If the history table has other columns in its primary key (such as deleted date) LEAVE THEM OUT.
- ```
<attribute><name>INCR_MERGE_HISTORY_TABLE_PKCSV</name>
<value>K1,K2</value></attribute>
```
- f. **INCR\_MERGE\_HISTORY\_TABLE\_DELDT\_COL**: Incremental Merge Type 1 - The column in the history table that indicates when the row was deleted. Similar to a "last updated" column.
- ```
<attribute><name>INCR_MERGE_HISTORY_TABLE_DELDT_COL</name>
<value>SDT</value></attribute>
```
- g. **INCR_MERGE_HISTORY_TABLE_STARTDT_COL**: Incremental Merge Type 2 - The column in the history table that indicates the bi-temporal start date.
- ```
<attribute><name>INCR_MERGE_HISTORY_TABLE_STARTDT_COL</name>
<value>SDT</value></attribute>
```
- h. **INCR\_MERGE\_HISTORY\_TABLE\_ENDDT\_COL**: Incremental Merge Type 2 - The column in the history table that indicates the bi-temporal end date.
- ```
<attribute><name>INCR_MERGE_HISTORY_TABLE_ENDDT_COL</name>
<value>EDT</value></attribute>
```
- i. **INCR_MERGE_HISTORY_TABLE_ACTIVITYDT_COL**: Incremental Merge Type 4- The column in the history table that indicates change data capture style activity date column.
- ```
<attribute><name>INCR_MERGE_HISTORY_TABLE_ACTIVITYDT_COL</name>
<value>SDT</value></attribute>
```

- j. **INCR\_MERGE\_HISTORY\_TABLE\_ACTIVITY\_COL**: Incremental Merge Type 4- The column in the history table that indicates change data capture style activity. e.g. ACTIVITY=[I|U|D].

```
<attribute><name>INCR_MERGE_HISTORY_TABLE_ACTIVITY_COL</name>
<value>ACTIVITY</value></attribute>
```

- k. **INCR\_MERGE\_HISTORY\_TABLE\_ACTIVITY\_INS\_VAL**: Incremental Merge Type 4- The column in the history table that indicates change data capture style activity insert value. e.g. ACTIVITY=I.

```
<attribute><name>INCR_MERGE_HISTORY_TABLE_ACTIVITY_INS_VAL</name>
<value>I</value></attribute>
```

- l. **INCR\_MERGE\_HISTORY\_TABLE\_ACTIVITY\_UPD\_VAL**: Incremental Merge Type 4- The column in the history table that indicates change data capture style activity update value. e.g. ACTIVITY=U.

```
<attribute><name>INCR_MERGE_HISTORY_TABLE_ACTIVITY_UPD_VAL</name>
<value>U</value></attribute>
```

- m. **INCR\_MERGE\_HISTORY\_TABLE\_ACTIVITY\_DEL\_VAL**: Incremental Merge Type 4- The column in the history table that indicates change data capture style activity delete value. e.g. ACTIVITY=D.

```
<attribute><name>INCR_MERGE_HISTORY_TABLE_ACTIVITY_DEL_VAL</name>
<value>D</value></attribute>
```

### Full Cache Pre/Post Procedure Configuration

#### 5) Full Cache Pre/Post cache procedure configuration

- a. Supported for Full and Multi-table
- b. Name format:
  - iv. VIEWNAME\_preCallback – developer modifies this procedure
  - v. VIEWNAME\_postCallback – developer modifies this procedure
- c. Implementation procedure variables passed in:
  - i. constantsPath
  - ii. cacheViewPath
  - iii. cacheViewName
  - iv. cacheKey
- d. Optional Configure: Full Cache Pre/Post-Refresh Cache Procedures:

- e. **FULL\_PRE\_CALLBACK\_IMPL\_PROC**: [optional] Explicit path to the **pre-refresh** callback implementation procedure or use the variable \$APPLICATION\_CACHE\_PROC\_IMPL which defaults to the CONSTANTS variable “ApplicationCacheProcPath” which is a path to the CacheCallbackScripts folder.

```
<attribute><name>FULL_PRE_CALLBACK_IMPL_PROC</name>
<value>$APPLICATION_CACHE_PROC_IMPL</value></attribute>
```

Allowed Variables:

\$APPLICATION\_CACHE\_PROC\_IMPL – The default application cache proc path defined in CONSTANTS along with a generated procedure name.

- f. **FULL\_POST\_CALLBACK\_IMPL\_PROC**: [optional] Explicit path to the **post-refresh** callback implementation procedure or use the variable \$APPLICATION\_CACHE\_PROC\_IMPL which defaults to the CONSTANTS variable “ApplicationCacheProcPath” which is a path to the CacheCallbackScripts folder.

```
<attribute><name>FULL_POST_CALLBACK_IMPL_PROC</name>
<value>$APPLICATION_CACHE_PROC_IMPL</value></attribute>
```

Allowed Variables:

\$APPLICATION\_CACHE\_PROC\_IMPL – The default application cache proc path defined in CONSTANTS along with a generated procedure name.

## Drop/Create Indexes

### 6) Drop/Create indexes

- a. Indexes integrated into View “Indexes” tab. If present, indexes are created otherwise no action.
- b. Cache table index name format:
  - i. Option 1: UPPER(VIEW\_NAME)\_<actual\_idx\_name>
  - ii. Option 2: INDEX\_PREFIX<actual\_idx\_name>
    1. This uses the INDEX\_PREFIX attribute to override the default upper view name with underscore format.
- c. **CONFIGURE**: Index Attributes
- d. **INDEX\_PREFIX**: Index prefix overrides the use of the upper case cached view name for prefixing the index name found on the index tab of the cached view.

```
<attribute><name>INDEX_PREFIX</name><value>IDX_</value></attribute>
```

- e. **INDEX\_TYPE**: Index type is appended after the indexNameURL in the CREATE INDEX syntax and is database specific. Syntax: CREATE INDEX <indexNameURL> [INDEX\_TYPE] ON <tableURL>(<columnList>) [INDEX\_OPTIONS]  
 <attribute><name>INDEX\_TYPE</name><value></value></attribute>
- f. **INDEX\_OPTIONS**: Index options are appended to the end of the CREATE INDEX syntax and are database specific. Syntax: CREATE INDEX <indexNameURL> [INDEX\_TYPE] ON <tableURL>(<columnList>) [INDEX\_OPTIONS]  
 <attribute><name>INDEX\_OPTIONS</name><value></value></attribute>
- g. For multi-table if the “Drop indexes before load and create indexes after refresh load” box is checked then drop and recreate is done by DV otherwise skipped.
  - i. The attributes REFRESH\_DROP\_INDEX and REFRESH\_CREATE\_INDEX are ignored.
- h. The attribute **REFRESH\_DROP\_INDEX** controls whether the indexes should be dropped during refresh. . If not specified no action is taken on indexes during refresh.
  - i. The value “ALL” specifies to drop all indexes that are configured in the view. Otherwise, a comma separate list of index names may be provided.
  - ii. **REFRESH\_DROP\_INDEX**: Drop all indexes for the cache table. [ALL, <comma separated list of indexes>]  
 <attribute><name>REFRESH\_DROP\_INDEX</name>  
 <value>ALL</value></attribute>
- i. The attribute **REFRESH\_CREATE\_INDEX** controls whether the indexes should be created during refresh. If not specified no action is taken on indexes during refresh.
  - i. The value “ALL” specifies to create all indexes that are configured in the view. Otherwise, a comma separate list of index names may be provided.
  - ii. **REFRESH\_CREATE\_INDEX**: Create all indexes for the cache table. [ALL, <comma separated list of indexes>]  
 <attribute><name>REFRESH\_CREATE\_INDEX</name>  
 <value>ALL</value></attribute>

## Execute Table Statistics

### 7) Execute table statistics

- a. The attribute **EXECUTE\_STATISTICS** controls whether to update statistics or not after the refresh cache is complete.
  - i. The value "TABLE" and performs an update statistics on the table.
  - ii. The value "EXPRESS" is for Netezza only and performs a "generate express statistics" on the table.
- b. Note: For Oracle, executing statistics on the table requires a lock on the table.

## Create Cache Schedule

### 8) Create Cache Schedule

- a. Create cache schedule using period within the cached view. This is the "Caching" tab on the view.
- b. REFRESH: Schedule Attributes
- c. **REFRESH\_MODE**: Schedule refresh mode [MANUAL, SCHEDULED]  
`<attribute><name>REFRESH_MODE</name><value>SCHEDULED</value></attribute>`
- d. **SCHEDULE\_MODE**: When refresh mode=SCHEDULED then [CALENDAR]  
`<attribute><name>SCHEDULE_MODE</name> <value>CALENDAR</value></attribute>`
- e. **SCHEDULE\_START\_TIME**: When refresh mode=SCHEDULED then a valid timestamp.  
`<attribute><name>SCHEDULE_START_TIME</name><value>2014-06-25 00:00:00</value></attribute>`
- f. **SCHEDULE\_PERIOD**: When refresh mode=SCHEDULED then values: SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, or YEAR.  
`<attribute><name>SCHEDULE_PERIOD</name> <value>HOUR</value></attribute>`
- g. **SCHEDULE\_COUNT**: When refresh mode=SCHEDULED then the number of Period units in the interval between cache refreshes. Values: Any positive integer.  
`<attribute><name>SCHEDULE_COUNT</name><value>1</value></attribute>`
- h. **EXPERIATION\_PERIOD**: The amount of time in milliseconds that the cache will be cleared after it is refreshed. Values: If less than zero, the period will be set to zero. If zero then the cache will never expire. If set to NULL, the enable setting will be left unaltered.  
`<attribute><name>EXPERIATION_PERIOD</name> <value>1</value></attribute>`
- i. **CLEAR\_RULE**: Indicates when old cache data should be cleared. Values: One of "NONE", "ON\_LOAD", or "ON\_FAILURE". Normally old cache data is cleared on expiration and when a cache refresh successfully completes. In the latter case the old cache data is replaced by the new cached data. If "NONE", then the normal behavior will be used. If



"ON\_LOAD", in addition to the normal behavior the old cache data will be cleared when a refresh is started. If "ON\_FAILURE", in addition to the normal behavior the old cache data will be cleared when a refresh fails. If set to NULL, the setting will be left unaltered.

```
<attribute><name>CLEAR_RULE</name><value>NONE</value></attribute>
```

## Create Cache Trigger Schedule

### 1) Create Cache Trigger Schedule

- a. The cache trigger schedule provides the added advantage over the integrated cache schedule in that it provides the granularity of "Day of Week" in the schedule.
- b. **REFRESH**: Cache Trigger Schedule Attributes
- c. **REFRESH\_MODE**: Schedule refresh mode [TRIGGER]
 

```
<attribute><name>REFRESH_MODE</name><value>TRIGGER</value></attribute>
```
- d. **TRIGGER\_ENABLE**: When refresh mode=TRIGGER then 0=disable trigger on creation. 1=enable trigger on creation.
 

```
<attribute><name>TRIGGER_ENABLE</name><value>0</value></attribute>
```
- e. **TRIGGER\_START\_TIME**: When refresh mode=TRIGGER then a valid timestamp.
 

```
<attribute><name>TRIGGER_START_TIME</name><value>2014-06-25
00:00:00</value></attribute>
```
- f. **TRIGGER\_PERIOD**: When refresh mode=TRIGGER then values: MINUTE, HOUR, DAY, WEEK, MONTH, or YEAR.
 

```
<attribute><name>TRIGGER_PERIOD</name> <value>DAY</value></attribute>
```
- g. **TRIGGER\_COUNT**: When refresh mode=TRIGGER then the number of Period units in the interval between cache refreshes. Values: Any positive integer.
 

```
<attribute><name>TRIGGER_COUNT</name><value>1</value></attribute>
```
- h. **TRIGGER\_RECURRING\_DAY**: When refresh mode=TRIGGER then NULL or a list of one or more items: [SUN, MON, TUE, WED, THU, FRI, SAT]. May be space or comma delimited.
 

```
<attribute><name>TRIGGER_RECURRING_DAY</name><value>MON WED
FRI</value></attribute>
```
- i. **TRIGGER\_FROM\_TIME\_IN\_A\_DAY**: When refresh mode=TRIGGER then NULL or the time of day to start the trigger between 00:00:00 and 23:30 in 30 min increments. Format = HH24:MM:SS.
 

```
<attribute><name>TRIGGER_FROM_TIME_IN_A_DAY</name>
<value>06:00:00</value></attribute>
```
- j. **TRIGGER\_END\_TIME\_IN\_A\_DAY**: When refresh mode=TRIGGER then NULL or the time of day to end the trigger between 00:00:00 and 23:30 in 30 min increments. Format = HH24:MM:SS.

```
<attribute><name>TRIGGER_END_TIME_IN_A_DAY</name>
<value>23:59:00</value></attribute>
```

- k. **TRIGGER\_IS\_CLUSTER**: When refresh mode=TRIGGER then NULL or 0=not once per cluster, 1=only once per cluster.

```
<attribute><name>TRIGGER_IS_CLUSTER</name> <value>1</value></attribute>
```

- l. **TRIGGER\_ANNOTATION**: When refresh mode=TRIGGER then NULL or an annotation for the trigger.

```
<attribute><name>TRIGGER_ANNOTATION</name><value>annotation</value></attribute>
```

Allowed Variables:

**\$DEFAULT\_RES\_NAME** – The \$DEFAULT\_RES\_NAME is the name of the view and is extracted dynamically from the resource being cached.

**\$DEFAULT\_RES\_PATH** – The \$DEFAULT\_RES\_PATH is the resource path and is extracted dynamically from the resource being cached.

**\$DEFAULT\_RES\_PARENT\_PATH** – The \$DEFAULT\_PARENT\_PATH is the parent resource path and is extracted dynamically from the resource being cached.

**\$DEFAULT\_TRIGGER\_SUFFIX** – The \$DEFAULT\_TRIGGER\_SUFFIX is the trigger suffix defined in the CONSTANTS as DefaultTriggerNameSuffix.

- m. **TRIGGER\_LOCATION**: When refresh mode=TRIGGER NULL or the path to the trigger. If entry is not present or null, the trigger gets created in the default /Scripts/CacheTriggers folder.

```
<attribute><name>TRIGGER_LOCATION</name><value>$DEFAULT_PARENT_PATH/$
DEFAULT_RES_NAME$DEFAULT_TRIGGER_SUFFIX</value></attribute>
```

Allowed Variables:

**\$DEFAULT\_RES\_NAME** – The \$DEFAULT\_RES\_NAME is the name of the view and is extracted dynamically from the resource being cached.

**\$DEFAULT\_RES\_PATH** – The \$DEFAULT\_RES\_PATH is the resource path and is extracted dynamically from the resource being cached.

**\$DEFAULT\_RES\_PARENT\_PATH** – The \$DEFAULT\_PARENT\_PATH is the parent resource path and is extracted dynamically from the resource being cached.

**\$DEFAULT\_TRIGGER\_SUFFIX** – The \$DEFAULT\_TRIGGER\_SUFFIX is the trigger suffix defined in the CONSTANTS as DefaultTriggerNameSuffix.

## Rolling Purge Window

### 2) Rolling Purge window

- a. Rolling Window - Data that rolls out of the window is purged
- b. Purge Trigger Schedule Attributes
- c. **PURGE\_TRIGGER\_ENABLE**: defines if the trigger is enabled or disabled upon creation. 0=disable trigger on creation. 1=enable trigger on creation.

```
<attribute><name>PURGE_TRIGGER_ENABLE</name><value>0</value></attribute>
```

- d. **PURGE\_WINDOW\_PERIOD**: defines the purge window that gets passed into the PurgeCacheData() procedure. (delete all rows WHERE PURGE\_COLUMN\_NAME <= CURRENT\_TIMESTAMP - (PURGE\_WINDOW\_COUNT \* PURGE\_WINDOW\_PERIOD)) values: MINUTE, HOUR, DAY, WEEK, MONTH, or YEAR.

```
<attribute><name>PURGE_WINDOW_PERIOD</name>
<value>WEEK</value></attribute>
```

- e. **PURGE\_WINDOW\_COUNT**: the number of period units in for the purge window and the value passed into PurgeCacheData() procedure. Values: Any positive integer.

```
<attribute><name>PURGE_WINDOW_COUNT</name> <value>1</value></attribute>
```

- f. **PURGE\_START\_TIME**: a valid starting timestamp.

```
<attribute><name>PURGE_START_TIME</name> <value>2014-06-25
00:00:00</value></attribute>
```

- g. **PURGE\_COLUMN\_NAME**: the timestamp column used for purging rows.

```
<attribute><name>PURGE_COLUMN_NAME</name> <value>SDT</value></attribute>
```

- h. **PURGE\_PERIOD**: defines the trigger interval period between purges. (how often the trigger fires) values: MINUTE, HOUR, DAY, WEEK, MONTH, or YEAR.

```
<attribute><name>PURGE_PERIOD</name><value>DAY</value></attribute>
```

- i. **PURGE\_COUNT**: the number of Period units in the interval between purges. Values: Any positive integer.

```
<attribute><name>PURGE_COUNT</name><value>1</value></attribute>
```

- j. **PURGE\_RECURRING\_DAY**: NULL or a list of one or more items: [SUN, MON, TUE, WED, THU, FRI, SAT]. May be space or comma delimited.

```
<attribute><name>PURGE_RECURRING_DAY</name> <value>MON WED
FRI</value></attribute>
```

- k. **PURGE\_FROM\_TIME\_IN\_A\_DAY**: NULL or The time of day to start the TRIGGER: between 00:00:00 and 23:30 in 30 min increments. Format = HH24:MM:SS

```
<attribute><name>PURGE_FROM_TIME_IN_A_DAY</name>
<value>06:00:00</value></attribute>
```

- l. **PURGE\_END\_TIME\_IN\_A\_DAY**: NULL or The time of day to end the TRIGGER: between 00:00:00 and 23:30 in 30 min increments. Format = HH24:MM:SS

```
<attribute><name>PURGE_END_TIME_IN_A_DAY</name>
<value>23:59:00</value></attribute>
```

- m. **PURGE\_IS\_CLUSTER**: NULL or 0=not once per cluster, 1=only once per cluster

```
<attribute><name>PURGE_IS_CLUSTER</name> <value>1</value></attribute>
```

- n. **PURGE\_ANNOTATION**: NULL or an annotation for the trigger.

```
<attribute><name>PURGE_ANNOTATION</name> <value>annotation</value></attribute>
```

Allowed Variables:

**\$DEFAULT\_RES\_NAME** – The \$DEFAULT\_RES\_NAME is the name of the view and is extracted dynamically from the resource being cached.

**\$DEFAULT\_RES\_PATH** – The \$DEFAULT\_RES\_PATH is the resource path and is extracted dynamically from the resource being cached.

**\$DEFAULT\_RES\_PARENT\_PATH** – The \$DEFAULT\_PARENT\_PATH is the parent resource path and is extracted dynamically from the resource being cached.

**\$DEFAULT\_TRIGGER\_SUFFIX** – The \$DEFAULT\_TRIGGER\_SUFFIX is the trigger suffix defined in the CONSTANTS as DefaultTriggerNameSuffix.

- o. **PURGE\_LOCATION:** NULL or the path to the trigger. If entry is not present or null, the trigger gets created in the default /Scripts/CacheTriggers folder.

```
<attribute><name>PURGE_LOCATION</name><value>$DEFAULT_PARENT_PATH/$DEFAULT_RES_NAME$DEFAULT_TRIGGER_SUFFIX</value></attribute>
```

Allowed Variables:

**\$DEFAULT\_RES\_NAME** – The \$DEFAULT\_RES\_NAME is the name of the view and is extracted dynamically from the resource being cached.

**\$DEFAULT\_RES\_PATH** – The \$DEFAULT\_RES\_PATH is the resource path and is extracted dynamically from the resource being cached.

**\$DEFAULT\_RES\_PARENT\_PATH** – The \$DEFAULT\_PARENT\_PATH is the parent resource path and is extracted dynamically from the resource being cached.

**\$DEFAULT\_TRIGGER\_SUFFIX** – The \$DEFAULT\_TRIGGER\_SUFFIX is the trigger suffix defined in the CONSTANTS as DefaultTriggerNameSuffix.

**Reset Maintenance Level****3) Reset maintenance level to force a full refresh**

- a. Reset the maintenance level back to null and reload the DV cache with the maintenance level.

**Validation****4) Validation**

- a. Validate database table and index name length for max allowed
- b. Alleviate naming collisions. Validate the cache or staging view is a dependent of the target DB cache or staging table to ensure that the table will not be dropped/created by mistake if the table name actually belongs to a different DV resource.

**Ability to Initialize Framework**

- 1) Rebind packaged queries after installation using the RebindPackagedQueries script.

- 2) Manage [drop/create] audit log correlation database sequence id  
CACHE\_FRAMEWORK\_SEQ
  - a. CreateDBSequence
  - b. DropDBSequence
- 3) Manage [drop/create] AUDIT\_LOG\_SEQ\_table
  - a. Create\_AUDIT\_LOG\_SEQ
  - b. Drop\_AUDIT\_LOG\_SEQ
- 4) Manage [drop/create] AUDIT\_LOG table
  - a. Create\_AUDIT\_LOG
  - b. Drop\_AUDIT\_LOG
- 5) Manage [drop/create] CACHING\_DATA table
  - a. Create\_CACHING\_DATA
  - b. Drop\_CACHING\_DATA
  - c. Insert\_CACHING\_DATA

### Ability to De-configure a Cache

The Cache Framework provides the ability to de-configure the cache resource in two different ways:

1. De-configure a single cache resource by providing the path.
2. De-configure all caches in CACHING\_DATA by leaving path null.

The following provides a detailed background on what the de-configure operation will execute:

- 1) Disable primary cache view (disable cache)
- 2) If Staging table used
  - a. Destroy staging view
  - b. Destroy DV staging table
  - c. Drop staging table in database
  - d. Delete the staging table cache\_status row
- 3) Destroy DV cache load or callback scripts. Optional to destroy custom callback implementation scripts.
  - a. Destroy first call back script [Initial Load or Pre-Callback and Pre-Callback Implementation (optional)]
  - b. Destroy second call back script [Initial Load or Pre-Callback and Pre-Callback Implementation (optional)]

- 4) Destroy DV cache table
- 5) Drop cache tables in database
- 6) De-configure primary cache view
  - a. Destroy cache configuration
  - b. Delete the cache table cache\_status row
- 7) Destroy the cache refresh trigger if it exists
- 8) Destroy the cache purge trigger if it exists
- 9) Delete AUDIT\_LOG\_SEQ entry
- 10) Verify the De-configuration

## 8 Cache Framework Deployment

The following is a discussion of how deployment works with the Cache Framework.

### Overview of Deployment

This discussion puts the deployment of cache objects into perspective of the bigger picture. When deploying cache objects, the developer must consider several factors.

1. The type of cache object [full or increment] being deployed.
  - a. When a full cache object is being deployed, there are no issues with deploying an object even if that object has been deployed before. If something happens to the cache\_status for that cache, it will be marked as invalid and will be fully refreshed on the next access to that cache object.
  - b. When an incremental cache object is being deployed for the first time there are no issues as the first time there is no data in the target cache. However, care must be taken not to deploy an incremental object again as it may result in marking the cache\_status to invalid thus causing a complete refresh of that cache.
2. How the DV objects get instantiated in the target server.
  - a. Packaged Import scripts is one methodology.
  - b. If running DV 7.0 or higher, Deployment Manager is another methodology.
  - c. AS Assets PDTTool is another methodology.
3. Methodology importing objects. This can be done via package import or version control deployment using PDTTool.
  - a. Ideally, all objects listed below would be packaged together into a single .car file for import unless PDTTool version control is being used in which case, the deployment plan should list the resources in the order shown below:
    - i. “CachingData” procedure will be used to setup the Cache Framework metadata. It is the only mechanism that is used to pass the intelligence about what is cached to the target server.
    - ii. Associated DV cache procedures scripts.
    - iii. Associate DV cache triggers.
    - iv. Associated DV staging views.
    - v. Actual DV view that is the primary cache target view.

4. How the database objects get created in the target cache database server.
  - a. The DeployCache procedure gets execute after import of the resources which creates any necessary tables and introspects them into DV.
5. How the cache refresh is performed.
  - a. The Cache Framework RefreshCache procedure can be executed from Studio or externally by a JDBC/ODBC tool.

## How to Deploy Cache Objects

How deployment works.

### DeployCache Procedure

This procedure is the main driver script for deploying cache views and tables. The script has the following input:

`inResourcePath` - VARCHAR(4096), - The specific view path to be configured or leave null to configure all views in CachingData (procedure) / CACHING\_DATA (table).

- 3) This path acts as a filter for what is already configured in CACHING\_DATA so the view must have already been configured in CACHING\_DATA.
- 4) Use a wildcard "%" at the end of the path to signify any text matching the text preceding the wildcard symbol "%".
- 5) If a container path is passed in with no wildcard "%", then the wildcard will be added automatically so that only view paths within this path range are deployed if applicable.

The script performs the following:

1. Execute the CachingData\_Insert procedure to insert CachingData() rows into CACHING\_DATA table.
  - a. The assumption is that with the deployment of a new table, the CACHING\_DATA metadata has not been inserted into the table and must be done prior to executing any cache configuration operations.
  - b. Note: It is required to deploy the CachingData procedure each time there is a cached view deployed.
2. Determine if the cache needs to be deployed based on whether cache tables exist in TDV and the generated view DDL columns and types matches the cache table columns and types excluding the cachekey column if it exists. If the this is a pre-existing cached view and the tables exist and they match the view signature then no action is taken.
3. If deploy cache then perform the following
  - a. Disable the cache views according to CACHING\_DATA query.
  - b. Invoke ConfigureCacheScript to create the tables but don't configure the views.



- i. The following dynamic SQL is used to override the “DROP\_RECREATE\_VIEW\_YN” by defaulting it to “N” so that the views do not get configured at the time of deployment no matter how that flag is set within the “CACHING\_DATA” procedure. Also note that the actual table path is substituted for the variable \$CachingDataTablePath at the time of deployment. Finally, the value of “DROP\_RECREATE\_TABLE\_YN” is read from the CACHING\_DATA table as per how it was set in the “CachingData” procedure. If it is set to “N” then no tables are created in the target cache database.

```
SELECT {option DISABLE_PLAN_CACHE, NO_DATA_CACHE}
 RUN_YN,
 "N" DROP_RECREATE_VIEW_YN,
 DROP_RECREATE_TABLE_YN,
 CACHE_TYPE,
 RESOURCE_PATH,
 ORGANIZATION,
 APPLICATION_NAME,
 CONSTANTS_PATH,
 CAST(ATTRIBUTES AS LONGVARCHAR) ATTRIBUTES
FROM $CachingDataTablePath
```

- c. Update impacted resources.
- d. Enable the cache views according to CACHING\_DATA query.

The script has the following output:

|                                               |              |                                                                                                                                           |
|-----------------------------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| errStatus                                     | VARCHAR,     | -- Status=SUCCESS, WARNING or FAIL                                                                                                        |
| errMessage                                    | LONGVARCHAR, | -- Status message upon FAIL otherwise NULL                                                                                                |
| warningMessage                                | LONGVARCHAR, | -- Warning messages.                                                                                                                      |
| numTablesCreated                              | INTEGER,     | -- The number of tables created by executing the DDL.                                                                                     |
| tableCreatedList                              | LONGVARCHAR, | -- Comma separated list of database tables                                                                                                |
| created by executing the DDL or NULL if none. |              |                                                                                                                                           |
| numViewsDeployed                              | INTEGER,     | -- The number of views that were deployed or NULL if none.                                                                                |
| viewsDeployedList                             | LONGVARCHAR, | -- Comma separated list of views that were                                                                                                |
| deployed or NULL if none.                     |              |                                                                                                                                           |
| numViewsConforming                            | INTEGER,     | -- The number of views that were not deployed because they are conforming to the existing table structure and do not need to be deployed. |

viewsConformingList LONGVARCHAR -- Comma separated list of views that are conforming to the table structure and don't need to be deployed.

## Sample PDTool Deployment Plan

This is a sample deployment plan which shows the order in which resources are checked out from VCS and imported into the target server. It also shows how to invoke the DeployCache procedure.

# -----

### # VCS Module: Cache Configuration Objects

# -----

```
PASS TRUE ExecuteAction vcsCheckout2 $SERVERID $VCONN
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle_Application/DataScripts/CachingData
"PROCEDURE" HEAD "$MODULE_HOME/VCSModule.xml" "$MODULE_HOME/servers.xml"
```

# -----

### # VCS Module: DV Database Objects

# -----

```
PASS TRUE ExecuteAction vcsCheckout2 $SERVERID $VCONN
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle_SampleSOR/SOR/SAMPLE_DS/CIS_CACHE/
V_Test_Datatypes_MixedCase "TABLE" HEAD "$MODULE_HOME/VCSModule.xml"
"$MODULE_HOME/servers.xml"
```

```
PASS TRUE ExecuteAction vcsCheckout2 $SERVERID $VCONN
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle_SampleSOR/SOR/SAMPLE_DS/CIS_CACHE/
V_Test_Datatypes_MixedCase_STG "TABLE" HEAD "$MODULE_HOME/VCSModule.xml"
"$MODULE_HOME/servers.xml"
```

# -----

### # VCS Module: DV Procedure Script Objects

# -----

```
PASS TRUE ExecuteAction vcsCheckout2 $SERVERID $VCONN
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle_SampleViews/Scripts/CacheLoadScripts/V_Test_Datatypes_MixedCase_CacheDeltaLoad
"PROCEDURE" HEAD "$MODULE_HOME/VCSModule.xml"
"$MODULE_HOME/servers.xml"
```

```
PASS TRUE ExecuteAction vcsCheckout2 $SERVERID $VCONN
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle_SampleViews/Scripts/CacheLoadScripts/V_Test_Datatypes_MixedCase_CacheInitialLoad
"PROCEDURE" HEAD "$MODULE_HOME/VCSModule.xml"
"$MODULE_HOME/servers.xml"
```

# -----

### # VCS Module: DV Staging Objects

# -----

```
PASS TRUE ExecuteAction vcsCheckout2 $SERVERID $VCONN
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle_SampleViews/Scripts/CacheStagingViews/V_Test_Datatypes_MixedCase_STG
"TABLE" HEAD "$MODULE_HOME/VCSModule.xml"
"$MODULE_HOME/servers.xml"
```

# -----

#### # VCS Module: DV Primary Cache View

# -----

```
PASS TRUE ExecuteAction vcsCheckout2 $SERVERID $VCONN
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle_SampleViews/Views/V_Test_Datatypes_Mixe
dCase "TABLE" HEAD "$MODULE_HOME/VCSModule.xml" "$MODULE_HOME/servers.xml"
```

# -----

#### # Resource Module: Deploy Cache Database Objects

# -----

- PASS TRUE ExecuteAction executeProcedure \$SERVERID  
**CacheManagement.CacheFramework.DeployCache ASAssets**  
 "\$MODULE\_HOME/servers.xml" "'0','0',  
 '/shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle\_Application/Constants/Constant  
 s',  
 '/shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle\_SampleViews/Views/V\_Test\_Da  
 tatypes\_MixedCase'" true

## 9 Cache Framework Implementation

The following is a discussion of how the Cache Framework has been implemented.

### Full Caching Methodology

At its most basic, the full caching solution involves the following steps:

- Gather all the current rows from the system of record.
- Insert the gathered rows into the cache table.
- If native load is available for the cache target and configured in DV then use it otherwise use JDBC inserts.

### Incremental Caching Methodology

At its most basic, the incremental caching solution involves the following steps:

- This only handles new record inserts since the last refresh.
- Gather the current rows that have been inserted since the last full or incremental refresh.
- Insert the gathered rows into the cache table.

### Hybrid Incremental Caching Methodology

At its most basic, the hybrid incremental caching solution involves the following steps:

- This only handles new record inserts since the last refresh.
- For first time, initial load performs a native load (if configured) into a staging table and once complete, perform a local database insert/select from staging into the cache table.
- For delta loads, gather the current rows that have been inserted since the last full or incremental refresh.
- Insert the gathered rows into the cache table.

### Hybrid Incremental Merge Caching Methodology

At its most basic, the incremental merge caching solution involves the following steps:

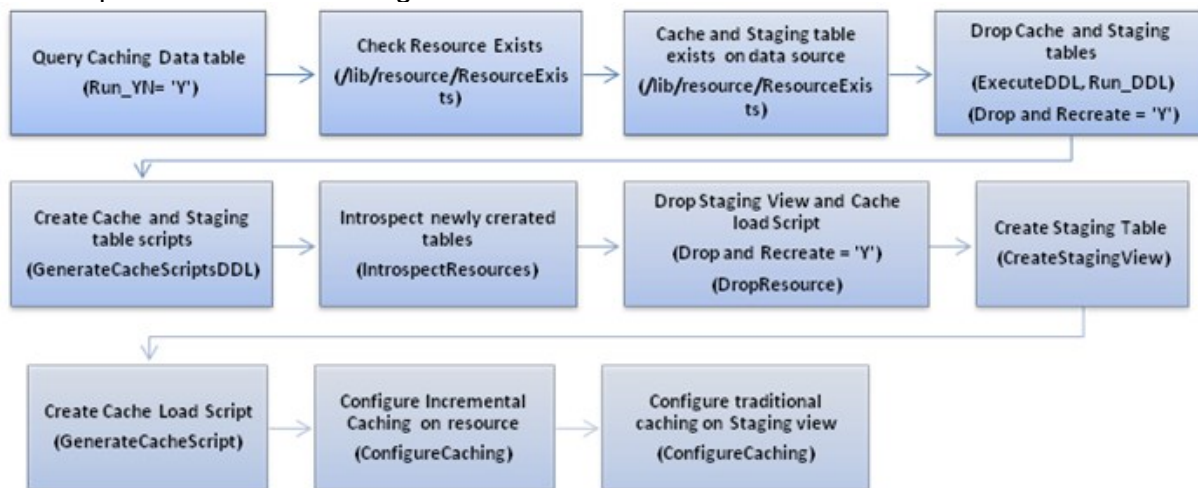
- This only handles inserts, updates and deletes since the last refresh.
- For first time, initial load perform a native load (if configured) into a staging table and once complete, perform a local database insert/select from staging into the cache table.

- For delta loads, gather the rows that have been inserted, updated, and deleted since the last full or incremental refresh.
- Insert the gathered rows into a staging table.
- Apply the changed rows to the existing cache data using the caching database's native `MERGE` command. This executes as a single ACID compliant transaction.

## Scripts

This folder contains the generic scripts that get invoked by the application template scripts.

The chart below details what the main script `<CF_FOLDER>/Scripts/ConfigureCacheScript` does when it performs a cache configuration:



The target data source for caching should be pre-configured for caching. This means that cache status and cache tracking table must be created and introspected. For best results, the case sensitivity and trailing space setting should match the settings of the view that is cached.

For each of the views that need to be cached, the “ConfigureCache” script iterates through the list and performs following steps:

1. Check if the target data source is configured for caching
2. Call `/lib/resource/ResourceExists` API and check if the view exists.
3. If drop and recreate database tables is Y (`DROP_RECREATE_TABLE_YN=Y`)
  - a. Call `GetResource` API function with view name as parameter to get columns and data type information and create a DDL string for creating a cache table.
  - b. When cache type is incremental, calls `GetResource` API function with view name as parameter to get columns and data type information and create a DDL string for creating a staging table.
  - c. Drop caching tables on data source

- d. Drop staging table on the data source if cache type is Hybrid incremental (HS, HN, HM).
  - e. Create the cache table by executing the DDL statement created in step 3a.
  - f. For incremental caching, create a staging table creating by executing the DDL statement generated in step 3b.
  - g. Introspect the data source to bring in the cache and staging target table.
- 4. If drop and recreate DV resources is Y (DROP\_RECREATE\_VIEW\_YN=Y)
  - a. Drop Staging view resource if drop resources flag is set to Y
  - b. Create a staging view as copy of source view in predefined folder when Hybrid incremental (HS, HN, HM, MT1, MT2, MT4).
  - c. Drop caching scripts if exist
  - d. Generate caching scripts
  - e. Configure caching parameters for the view. This step updates the following on the view
    - i. Target database
    - ii. Target table
    - iii. Refresh schedule configure as per attributes
    - iv. Update incremental scripts
  - f. Configure caching parameters for staging view using default caching mechanism and update the following information
    - i. Target database
    - ii. Target table for staging view cache
    - iii. Manual refresh schedule

**NOTE: The caching framework described in above section only configures a resource for caching. The framework does not start a refresh of data.**

### Cache Framework Application Template Folder

This folder contains the “interface” scripts used to initialize, manage, configure or deconfigure caching for a view. The following sections describe the sub-folders of <CF\_Folder>/ApplicationTemplate[Oracle|SqlServer|Netezza|MemSql|Postgres]. This folder will be referred to as “**ApplicationTemplate**”.

### Constants Folder

#### Constants()

Location: ApplicationTemplate/Constants/Constants()

This procedure contains a number of constants and other resources that are used by a number of caching resources. Update the `ApplicationBasePathConstant` variable with the correct value for `<CF_Folder>`.

## **DataScripts Folder**

### **CachingData()**

Location: `ApplicationTemplate/DataScripts/CachingData()`

This procedure is used to hold data for the caching framework. This script returns a cursor output which is used by `ConfigureCache` to orchestrate the scripting. This procedure is the primary mechanism used when deploying a cache resource from one environment to another. This resource should always be deployed along with the cached view and any other dependent resources except the TDV cache tables which get created at the time of deployment. The data populated in this script contains the following structure and correlates one for one with the `CACHING_DATA` database table:

| Column Name            | Required | Column Description                                                                                                                                                                                                                                                                                            |
|------------------------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RUN_YN                 | Y        | Possible values are 'Y' or 'N'.<br>Identifies rows for which caching script should be executed.                                                                                                                                                                                                               |
| DROP_RECREATE_VIEW_YN  | Y        | Possible values 'Y' or 'N'<br>If 'Y' is provided, the script will drop and recreate all DV caching framework related objects. If 'N' is provided, existing objects will not be dropped, but ignored.                                                                                                          |
| DROP_RECREATE_TABLE_YN | Y        | Possible values 'Y' or 'N'<br>If 'Y' is provided, the script will drop and recreate all database caching and staging tables on the cache target data source. This value is independent of the Drop and Recreate views.<br>If 'N' is provided, existing objects will not be dropped, but ignored.              |
| CACHE_TYPE             | Y        | The type of cache to configure.<br>FS=Full Single<br>FM=Full Multi-table<br>IN=Incremental<br>HS=Hybrid with Delta Staging table<br>HN=Hybrid with Delta No Staging Table<br>MT1=Merge Type 1 (Oracle and SQL Server)<br>MT2=Merge Type 2 (Oracle and SQL Server)<br>MT4=Merge Type 4 (Oracle and SQL Server) |

|                  |   |                                                                                                                                       |
|------------------|---|---------------------------------------------------------------------------------------------------------------------------------------|
| RESOURCE_PATH    | Y | Full path of the view being cached.                                                                                                   |
| ORGANIZATION     | Y | Organization name is used to identify the high-level usage such as "Mortgage". It is used as a filter for AUDIT_LOG and CACHING_DATA. |
| APPLICATION_NAME | Y | Application name is used to identify the subject area usage. It is used as a filter for AUDIT_LOG and CACHING_DATA.                   |
| CONSTANTS_PATH   | Y | The full DV path to the application constants file. This provides the context for all cache framework operations.                     |
| ATTRIBUTES       | N | Provides an XML configuration of attributes to configure various features. This value can be NULL. Name/Value pair attributes.        |

### **CachingData\_Display()**

Location: ApplicationTemplate/DataScripts/CachingData\_Display()

This procedure is used to display the contents of CACHING\_DATA table within the context of this "ORGANIZATION" and "APPLICATION\_NAME".

### **CachingData\_Insert()**

Location: ApplicationTemplate/DataScripts/CachingData\_Insert()

This script provides a way of inserting rows from the CachingData() procedure into the CACHING\_DATA table. The procedure uses a delete filter on "ORGANIZATION" and "APPLICATION\_NAME" when deleting records prior to insert.

### **Execute Folder**

#### **AuditLogDisplay()**

Location: ApplicationTemplate/Execute/AuditLogDisplay()

This procedure is used to dynamically display rows in the AUDIT\_LOG table. The input parameters are qualified by operators such as EQUAL "=" or LIKE "like". If the input parameter does not have a qualifier then it is assumed to use EQUAL "=" when constructing the where clause. If any input parameters are left NULL/blank they are not used to construct the where clause. The following are a list of input parameters that may be used:

- SEQUENCE\_NUM – The sequence number that is used to correlate across a group of like messages.
- RESOURCE\_NAME\_EQUAL – A resource name equivalent.
- RESOURCE\_NAME\_LIKE – A resource name like using a wildcard at the end of the string.



- `NOTIFICATION_TYPE` – The notification type [ERROR, AUDIT, INFO, DEBUG].
- `ORIG_USER_NAME` – The user@domain that originated the process.

## ConfigureCache()

Location: ApplicationTemplate/Execute/ConfigureCache()

This procedure is the main driver script and will do the following when configuring a cache view.

- Query CachingData driver table for all records with `RUN_YN = 'Y'`
- Create cache table on data source
- Create stage cache table on data source
- Introspect both tables
- Create staging view resource as a copy of primary view
- Create cache load script
- Configure cache on primary view
- Configure cache on staging view (default caching method)
- Drop and recreate cache table, staging table, cache scripts, stage view as well as trigger

The following are a list of input parameters that may be used:

- `inResourcePath` – The specific view path to be configured or leave null to configure all views in CachingData (procedure) / CACHING\_DATA (table). This path acts as a filter for what is already configured in CACHING\_DATA so the view must have already been configured in CACHING\_DATA. Use a wildcard "%" at the end of the path to signify any text matching the text preceding the wildcard symbol "%".

## ConfigureCache\_Multi\_Table()

Location: ApplicationTemplate/Execute/ConfigureCache\_Multi\_Table()

This procedure is the main driver script and will do the following when configuring a multi-table cache view.

It is provided for “ease of use”. It automatically updates the CachingData procedure and table so that the developer does not have to manually edit the script and insert the data into the table. The default mode is to configure multi-table caching with a pre-callback and post-callback procedure integrated into the cache view. It is configured to automatically drop and create indexes on the cache table during a cache refresh.

- Automatically update the CACHING\_DATA table and then regenerate the CachingData driver procedure.
- Create three cache tables in the data source.
- Introspect all necessary tables.

- Configure caching on primary view.
- Configure pre-callback and post-callback procedures.
- Drop and recreate cache table
  - a. Except when MemSql column store indexes are used.
- Create indexes if configured on the target view.

The following are a list of input parameters that may be used:

- `inResourcePath` – The full path to the view to be configured for multi-table caching.
- `tableConstraints` – Table Constraint(s) that can be used to qualify the table. Null if no table constraints. Database specific syntax that succeeds the column list and within the right parenthesis. CREATE TABLE "tablename" ( {ColumnList}, {TableConstraints} ) {TableOptions}
  - a. Separators: Each column in the table constraints list must contain a valid begin and end separator.
    - Oracle, Netezza, Postgres begin=" end="
    - SqlServer begin=[ end=]
    - MemSql begin=` end=`
    - MemSql: Format: key(<sortkeylist>) using clustered columnstore, shard key(<shardlist>) Example: key(`K1`,`K2`) using clustered columnstore, shard key(`S1`,`S2`)
- `tableOptions` – Table option(s) that can be used to qualify how the table is created. Null if no table options. Database specific syntax that succeeds the column list and is outside the right parenthesis. CREATE TABLE "tablename" ( {ColumnList}, {TableConstraints} ) {TableOptions}
  - a. Separators: Each column in the table option list must contain a valid begin and end separator.
    - Oracle, Netezza, Postgres begin=" end="
    - SqlServer begin=[ end=]
    - MemSql begin=` end=`
    - Netezza: Format: DISTRIBUTE ON (<keylist>) Example: DISTRIBUTE ON ("C1", "C2")
    - MemSql: Example: AUTOSTATS\_ENABLED
    - Azure Synapse Analytics (SQL Server DW): WITH (DISTRIBUTION = hash("id"), CLUSTERED COLUMNSTORE INDEX)
- `additionalAttributes` – Additional XML attributes to specify the cache definition.
  - a. Example: trigger attributes.
    - IF the keyword: CACHE\_DEFAULT\_TRIGGER is used then a cache trigger will be created using default attributes. Modify /DataScripts/Caching/Data and <ViewName>\_CacheTrigger with more accurate schedule information.

- IF the keyword: PURGE\_DEFAULT\_TRIGGER is used then a purge trigger will be created using default attributes. Modify /DataScripts/Caching/Data and <ViewName>\_PurgeTrigger with more accurate schedule information and purge column in the procedure attributes section.
- IF XML <attribute> is used then it will be taken as a literal string and used. If any syntax is wrong, it may throw an exception.

### **ConfigureCache\_Multi\_ColumnStore() [deprecated]**

Location: MemSql\_ApplicationTemplate/Execute/ConfigureCache\_Multi-ColumnStore()

[DEPRECATED]. Use. ConfigureCache\_Multi\_Table to achieve the same capability.

This MemSql only procedure is the main driver script and will do the following when configuring a cache view with a MemSql columnstore.

It is provided for “ease of use”. It automatically updates the CachingData procedure and table so that the developer does not have to manually edit the script and insert the data into the table. The default mode is to configure multi-table caching with a pre-callback and post-callback procedure integrated into the cache view. It is configured to automatically drop and create indexes on the cache table during a cache refresh.

- Automatically update the CACHING\_DATA table and then regenerate the CachingData driver procedure.
- Create multiple cache tables in the data source.
- Introspect all necessary tables.
- Configure caching on primary view.
- Drop and recreate cache table.

The following are a list of input parameters that may be used:

- `inResourcePath` – The full path to the view to be configured for multi-table caching.
- `keyNameList` – A comma-separated list of key names to create a column store from. Null if no column store is to be created.

### **ConfigureCacheAdhoc()**

Location: ApplicationTemplate/Execute/ConfigureCacheAdhoc()

This procedure provides a way of configuring a cache view in an adhoc way bypassing the CachingData() procedure. This script performs the same functions as "ConfigureCache" and is the main driver script and will do the following:

- Query CachingData driver table for all records with RUN\_YN = 'Y'
- Create caching table on data source
- Create staging cache table on data source

- Introspect both tables
- Create staging view resource as a copy of primary view
- Create cache load script
- Configure caching on primary view
- Configure caching on staging view (default caching method)
- Drop and recreate caching table, staging table, caching scripts, staging view as well as trigger

The following are a list of input parameters that may be used:

- `inResourcePath` – The specific view path to be configured or leave null to configure all views in CachingData (procedure) / CACHING\_DATA (table). This path acts as a filter for what is already configured in CACHING\_DATA so the view must have already been configured in CACHING\_DATA. Use a wildcard "%" at the end of the path to signify any text matching the text preceding the wildcard symbol "%".
- `inCacheType` – Valid types include the list below. Only specified when `inResourcePath` is specified.
  - FS=Full Single table cache, Optional Pre/Post callback scripts generated when using "ConfigureCache" procedure.
  - FM=Full Multi-table cache, Optional Pre/Post callback scripts generated when using "ConfigureCache" procedure.
  - IN=Incremental cache (no staging tables), Initial Load and Delta Load scripts generated.
  - HS=Hybrid with Delta Staging table, Initial and Delta use the same load script.
  - HN=Hybrid with Delta No Staging Table, Initial Load and Delta Load scripts generated.
  - MT1=Merge Type 1 using Deleted Table. Hybrid Merge Table with Initial and Delta staging table utilized and separate Initial and Delta load script.
  - MT2=Merge Type 2 using History table with Start and End Date (Bi-temporal)
  - MT4=Merge Type 4 using CDC style where there is an activity column specifying the action [I,U,D]. Hybrid Merge Column with Initial and Delta staging table utilized and separate Initial and Delta load script.
- `inDropRecreateViewYN` – Y/N. Drop and recreate the DV view resource. Only specified when `inResourcePath` is specified.
- `inDropRecreateTableYN` – Y/N. Drop and recreate the database table. Only specified when `inResourcePath` is specified.
- `inAttributes` – XML-based attributes if needed. Leave null if no attributes. Only specified when `inResourcePath` is specified.

## DeconfigureCache()

Location: ApplicationTemplate/Execute/DeconfigureCache()

This procedure is the main driver script that will de-configure the cache. It will do the following:

1. Disable primary cache view (disable cache)
2. If Staging table used
  - a. 2.a. Destroy staging view
  - b. 2.b. Destroy DV staging table
  - c. 2.c. Drop staging table in database
  - d. 2.d. Delete the staging table cache\_status row
3. Destroy DV cache load or callback scripts. Optional to destroy custom callback implementation scripts.
  - a. 3.a. Destroy first call back script [Initial Load or Pre-Callback and Pre-Callback Impl (optional)]
  - b. 3.b. Destroy second call back script [Initial Load or Pre-Callback and Pre-Callback Impl (optional)]
4. Destroy DV cache table
5. Drop cache tables in database
6. De-configure primary cache view
  - a. 6.a. Destroy cache configuration
  - b. 6.b. Delete the cache table cache\_status row
7. Destroy the cache refresh trigger if it exists
8. Destroy the cache purge trigger if it exists
9. Delete AUDIT\_LOG\_SEQ entry
10. Verify the De-configuration

The following are a list of input parameters that may be used:

- `inResourcePath` – The specific view path to be configured or leave null to configure all views in CachingData (procedure) / CACHING\_DATA (table). This path acts as a filter for what is already configured in CACHING\_DATA so the view must have already been configured in CACHING\_DATA. Use a wildcard "%" at the end of the path to signify any text matching the text preceding the wildcard symbol "%".

## DeployCache()

Location: ApplicationTemplate/Execute/DeployCache()

This procedure is the main driver script for deploying cache views and tables. The script performs the following:

1. Execute the `CachingData_Insert` procedure to insert `CachingData()` rows into `CACHING_DATA` table
2. Determine if the cache needs to be deployed based on whether cache tables exist in TDV and the generated view DDL columns and types matches the cache table columns and types excluding the cachekey column if it exists. If the this is a pre-existing cached view and the tables exist and they match the view signature then no action is taken.
3. If deploy cache then perform the following
  - a. Disable the cache views according to `CACHING_DATA` query
  - b. Invoke `ConfigureCacheScript` to create the tables but don't configure the views
  - c. Update impacted resources
  - d. Enable the cache views according to `CACHING_DATA` query

#### Input for `DeployCache`

`inResourcePath` VARCHAR(4096) - The specific view path to be configured or leave null to configure all views in `CachingData` (procedure) / `CACHING_DATA` (table).

- This path acts as a filter for what is already configured in `CACHING_DATA` so the view must have already been configured in `CACHING_DATA`.
- Use a wildcard "%" at the end of the path to signify any text matching the text preceding the wildcard symbol "%".
- If a container path is passed in with no wildcard "%", then the wildcard will be added automatically so that only view paths within this path range are deployed if applicable.

#### **MigrateCachedViews()**

Location: `ApplicationTemplate/Execute/MigrateCachedViews()`

This script provides a way to migrate views into the Cache Framework that are not under the control of the Cache Framework. Being under control of the Cache Framework means having a row in the `CACHING_DATA` table for each cached view and an equivalent row in the "CachingData" procedure.

Given a starting path, this procedure will interrogate each view and determine if it is currently in the Cache Framework or not. If not, it will add the view to the Cache Framework. Alternatively, the user may point directly to a cached view and only bring in that one view into the Cache Framework.

After completion of this procedure, if there are views that you do not want in the Cache Framework, then remove them from the "CachingData" procedure and execute "CachingData\_Insert" to remove and insert the remainder of the views in "CachingData".

**Input:**

`startingResourcePath`      VARCHAR(4096), - The starting CONTAINER path to search for cached views or a specific path to a cached view to bring into the Cache Framework.

**Output:**

```
result CURSOR (
 status VARCHAR, -- The status of what happend with the resource path.
 -- MIGRATED=The view was migrated into the Cache Framework.
 -- BYPASS=The view already exists and was bypassed.
 -- FAIL=A failure occurred.
 cacheType VARCHAR, -- The type of cache: FS=single, FM=multi-table, IN=incremental
 resourcePath VARCHAR(4096),-- The resource path being migrated.
 message LONGVARCHAR,-- Any messages regarding the resource path.
 actions LONGVARCHAR-- Any actions that must be taken post migration.
)
```

### **PurgeCacheData()**

Location: ApplicationTemplate/Execute/PurgeCacheData()

This procedure is a generic script that is invoked by the purge data trigger to purge data within a rolling period of time. The purge time is determined by the current timestamp - the purge period and compared to a timestamp column within the data. All rows are purged (deleted) that fall outside that rolling window.

- `inSequenceName` – The name of the sequence. Leave null when called manually and a sequence will be generated otherwise it is passed in from the invoking procedure.
- `inOrigUserName` – The original user@domain who started the process and transcends sessions. If not set it will be retrieved from the environment.
- `constantsPath` – Path to the application constants procedure.
- `purgeQuantity` – The quantity for the purgePeriod.
- `purgePeriod` – One of [HOUR, DAY, WEEK, MONTH, YEAR]. Note: MONTH is 31 days.
- `purgeColumnName` – The timestamp column used to compare with the purge period.
- `cacheViewPath` – The path to the DV cache view.

### **RefreshCache()**

Location: ApplicationTemplate/Execute/RegreshCache()

This procedure is used to kick off a cache refresh on a resource. Currently the default resource type is set to TABLE. The script takes one parameter "resourcePath" which is the full path to the view. The script parses the view path to extract the view name. The script returns SUCCESS or FAIL as output parameter depending if the cache refresh was successful or failure. Failure reason is written to the cache framework log "cfLog" which determines the log type based on how the "Constants" are configured. The log entry may go to the cs\_server.log file, Audit\_log, or printed to the command line. More than one option can be selected.

This script uses a blocking strategy; therefore, the script will wait till the refresh either completes or fails. The polling time is configurable and is currently set to 30 seconds to test whether the cache is completed or not.

### **Initialize Folder**

#### **\_Execute\_All\_Scripts()**

Location: ApplicationTemplate/Initialize/\_Execute\_All\_Scripts()

This procedure is an initialization script used drive the execution of all of the initialization scripts in this folder.

#### **Create\_AUDIT\_LOG()**

Location: ApplicationTemplate/Initialize/Create\_AUDIT\_LOG()

This procedure is an initialization script used to create the AUDIT\_LOG table. This script only needs to be run once. This script will automatically drop the AUDIT\_LOG table if it exists before creating it.

#### **Create\_AUDIT\_LOG\_SEQ()**

Location: ApplicationTemplate/Initialize/Create\_AUDIT\_SEQ()

This procedure is an initialization script used to create the AUDIT\_LOG\_SEQ table. This script only needs to be run once. This script will automatically drop the AUDIT\_LOG\_SEQ table if it exists before creating it.

#### **Create\_CACHING\_DATA()**

Location: ApplicationTemplate/Initialize/Create\_CACHING\_DATA()



This procedure is an initialization script used to create the CACHING\_DATA table. This script only needs to be run once. This script will automatically drop the CACHING\_DATA table if it exists before creating it.

### **CreateDBSequence()**

Location: ApplicationTemplate/Initialize/CreateDBSequence()

This procedure is an initialization script used to create the CACHE\_FRAMEWORK\_SEQ sequence. This script does *\*not\** drop the sequence first. The drop sequence must be invoked separately. The CACHE\_FRAMEWORK\_SEQ is used to correlate a unique number across DV sessions and transactions. It is used during all operations of the cache framework including Refresh, Configuration, De-configuration and other operations. It uses the AUDIT\_LOG\_SEQ table to store a correlation name and sequence number. For example, if a cache refresh was started by a user which then goes through the admin interface, the user name is preserved along with the sequence number even though control was transferred to another user. Additionally, the sequence number is preserved through the cache execution of either incremental load scripts or call back scripts.

### **Drop\_AUDIT\_LOG()**

Location: ApplicationTemplate/Initialize/Drop\_AUDIT\_LOG()

This procedure is an initialization script used to drop the AUDIT\_LOG table. This script also gets invoked by Create\_AUDIT\_LOG. This script will drop the AUDIT\_LOG table if it exists before creating it.

### **Drop\_AUDIT\_LOG\_SEQ()**

Location: ApplicationTemplate/Initialize/Drop\_AUDIT\_SEQ()

This procedure is an initialization script used to drop the AUDIT\_LOG\_SEQ table. This script also gets invoked by Create\_AUDIT\_LOG\_SEQ. This script will drop the AUDIT\_LOG\_SEQ table if it exists before creating it.

### **Drop\_CACHING\_DATA()**

Location: ApplicationTemplate/Initialize/Drop\_CACHING\_DATA()

This procedure is an initialization script used to drop the CACHING\_DATA table. This script also gets invoked by Create\_CACHING\_DATA. This script will drop the CACHING\_DATA table if it exists before creating it.

### **DropDBSequence()**

Location: ApplicationTemplate/Initialize/DropDBSequence()

This procedure is an initialization script used to drop the CACHE\_FRAMEWORK\_SEQ sequence.

### **RebindPackagedQueries()**

Location: ApplicationTemplate/Initialize/RebindPackagedQueries()

This procedure is an initialization script used to rebind the packaged queries from the temporary datasource to the one specified in the Constants procedure.

### **PackagedQueryDDLScripts Folder**

#### **TEMP\_DS**

Location: ApplicationTemplate/PackagedQueryDDLScript/TEMP\_DS()

This is the temporary data source used only at installation time when a rebind of the packaged queries is performed from the temp data source to the actual data source which is configured in the “constants” procedures.

#### **ExecuteDDL()**

Location: ApplicationTemplate/PackagedQueryDDLScript/ExecuteDDL()

This is a packaged query that is used to execute any DDL on the cache data source. It is rebound to the actual data source at the time of application configuration using the “/Initialize/RebindPackagedQueries” procedure.

The following are a list of input parameters that may be used:

- `inputSQL` – Take a SQL statement.

#### **ExecuteDDLSystem()**

Location: ApplicationTemplate/PackagedQueryDDLScript/ExecuteDDLSystem()

This is a packaged query that is used to execute any DDL on the system data source. The system data source is used to store the cache\_status, cache\_tracking and cache framework tables. It does not store cache tables. This is only required when the cache database is configured to use “other datasource”.

It is rebound to the actual data source at the time of application configuration using the “/Initialize/RebindPackagedQueries” procedure.

The following are a list of input parameters that may be used:

- `inputSQL` – Take a SQL statement.

**GetIntResult()**

Location: ApplicationTemplate/PackagedQueryDDLScript/GetIntResult()

This is a packaged query that is used to execute any DML query that returns an integer value on the cache data source. For example SELECT COUNT(\*) FROM x. It is rebound to the actual data source at the time of application configuration using the “/Initialize/RebindPackagedQueries” procedure.

The following are a list of input parameters that may be used:

- `selectCountSQLStatement` – Take a Select Count(\*) SQL statement.

**GetIntResultSystem()**

Location: ApplicationTemplate/PackagedQueryDDLScript/GetIntResultSystem()

This is a packaged query that is used to execute any DML query that returns an integer value on the system data source. The system data source is used to store the cache\_status, cache\_tracking and cache framework tables. It does not store cache tables. This is only required when the cache database is configured to use “other datasource”.

For example SELECT COUNT(\*) FROM x. It is rebound to the actual data source at the time of application configuration using the “/Initialize/RebindPackagedQueries” procedure.

The following are a list of input parameters that may be used:

- `selectCountSQLStatement` – Take a Select Count(\*) SQL statement.

**ShowIndexes()**

Location: MemSql\_ApplicationTemplate/PackagedQueryDDLScript/ShowIndexes()

This is a packaged query that is used by MemSql only to get/show indexes on a table.

**ShowTables()**

Location: MemSql\_ApplicationTemplate/PackagedQueryDDLScript/ShowTables()

This is a packaged query that is used by MemSql only to get/show tables in a schema.

**Published Procedures**

The published procedures are those that can be invoked externally by a 3<sup>rd</sup> party JDBC or ODBC tool.

**ClearPlanCache()**

Location: /databases/ASAssets/CacheManagement/CacheFramework/ClearPlanCache()

This procedure is designed to clear the plan cache. This script uses the admin interface providing elevated privileges to perform this action.

### **ClearRepositoryCache()**

Location: /databases/ASAssets/CacheManagement/CacheFramework/ClearRepositoryCache()

This procedure is designed to clear the repository cache. This script uses the admin interface providing elevated privileges to perform this action.

### **ConfigureCache()**

Location: /databases/ASAssets/CacheManagement/CacheFramework/ConfigureCache()

This procedure uses the admin interface providing elevated privileges to perform this action. Refer back to [ApplicationTemplate/Execute/ConfigureCache](#) for more details.

### **ConfigureCache\_Multi\_Table()**

Location:  
/databases/ASAssets/CacheManagement/CacheFramework/ConfigureCache\_Multi\_Table()

This procedure uses the admin interface providing elevated privileges to perform this action. Refer back to [ApplicationTemplate/Execute/ConfigureCache\\_Multi\\_Table](#) for more details.

### **ConfigureCacheAdhoc()**

Location: /databases/ASAssets/CacheManagement/CacheFramework/ConfigureCacheAdhoc()

This procedure provides a way of configuring a cache view in an adhoc way bypassing the CachingData() procedure. This script uses the admin interface providing elevated privileges to perform this action. Refer back to [ApplicationTemplate/Execute/ConfigureCacheAdhoc](#) for more details.

### **CopyPrivilegesFromParent()**

Location:  
/databases/ASAssets/CacheManagement/CacheFramework/CopyPrivilegesFromParent()

This procedure is designed to push privileges to a newly create resource and makes is same as the parent folder in which the resource exists. This script uses the admin interface providing elevated privileges to perform this action.

The script takes a resource name with path and resource type as input parameter and returns SUCCESS or FAIL as output message. Additional logging is done in AUDIT\_LOG table as well as

in cs\_server.log file. This script requires the user executing the script to have GRANT privileges on the source folder.

### **DeconfigureCache()**

Location: /databases/ASAssets/CacheManagement/CacheFramework/DeconfigureCache()

This procedure is used to de-configure the cache. Refer back to [ApplicationTemplate/Execute/DeconfigureCache](#) for more details.

### **IntrospectTables()**

Location: /databases/ASAssets/CacheManagement/CacheFramework/IntrospectTables()

This procedure is used to provide a consistent and generic interface for introspecting database tables. This script uses the admin interface providing elevated privileges to perform this action.

### **RefreshCache()**

Location: /databases/ASAssets/CacheManagement/CacheFramework/RefreshCache()

This procedure is used to kick off a cache refresh on a resource. Currently the default resource type is set to TABLE. This script uses the admin interface providing elevated privileges to perform this action. Refer back to [ApplicationTemplate/Execute/RefreshCache](#) for more details.

### **UpdateResourceOwner()**

Location: /databases/ASAssets/CacheManagement/CacheFramework/UpdateResourceOwner()

This procedure is used to change resource ownership of a resource. This script can be used to change ownership of a single resource or a folder with multiple resources. By default the ownership is pushed recursively to all the resources in a folder. This script uses the admin interface providing elevated privileges to perform this action.

Executing the library function used in this script requires ACCESS TOOLS, READ ALL RESOURCES, READ ALL USERS role, therefore ideally, this script should be executed by an Admin user.

## **Admin Interface Procedures**

The admin interface procedures serve the purpose of allowing a user to execute a procedure with elevated rights by redirecting the procedure through an admin interface DV internal data source where the user has admin rights.

### **AdminInterface/ClearPlanCache()**

Location: <CF\_FOLDER>/Scripts/AdminInterface/ClearPlanCache()

This procedure is designed to clear the plan cache. This script uses the admin interface providing elevated privileges to perform this action.

### **AdminInterface/ClearRepositoryCache()**

Location: <CF\_FOLDER>/Scripts/AdminInterface/ClearRepositoryCache()

This procedure is designed to clear the repository cache. This script uses the admin interface providing elevated privileges to perform this action.

### **AdminInterface/ConfigureCache()**

Location: <CF\_FOLDER>/Scripts/AdminInterface/ConfigureCache()

This procedure uses the admin interface providing elevated privileges to perform this action. Refer back to [ApplicationTemplate/Execute/ConfigureCache](#) for more details.

### **AdminInterface/ConfigureCache\_Multi\_Table()**

Location: <CF\_FOLDER>/Scripts/AdminInterface/ConfigureCache\_Multi\_Table()

This procedure uses the admin interface providing elevated privileges to perform this action. Refer back to [ApplicationTemplate/Execute/ConfigureCache\\_Multi\\_Table](#) for more details.

### **AdminInterface/ConfigureCacheAdhoc()**

Location: <CF\_FOLDER>/Scripts/AdminInterface/ConfigureCacheAdhoc()

This procedure provides a way of configuring a cache view in an adhoc way bypassing the CachingData() procedure. This script uses the admin interface providing elevated privileges to perform this action. Refer back to [ApplicationTemplate/Execute/ConfigureCacheAdhoc](#) for more details.

### **AdminInterface/CopyPrivilegesFromParent()**

Location: <CF\_FOLDER>/Scripts/AdminInterface/CopyPrivilegesFromParent()

This procedure is designed to push privileges to a newly create resource and makes is same as the parent folder in which the resource exists. This script uses the admin interface providing elevated privileges to perform this action.

The script takes a resource name with path and resource type as input parameter and returns SUCCESS or FAIL as output message. Additional logging is done in AUDIT\_LOG table as well as in cs\_server.log file. This script requires the user executing the script to have GRANT privileges on the source folder.

**AdminInterface/DeconfigureCache()**

Location: <CF\_FOLDER>/Scripts/AdminInterface/DeconfigureCache()

This procedure is used to de-configure the cache. Refer back to [ApplicationTemplate/Execute/DeconfigureCache](#) for more details.

**AdminInterface/IntrospectTables()**

Location: <CF\_FOLDER>/Scripts/AdminInterface/IntrospectTables()

This procedure is used to provide a consistent and generic interface for introspecting database tables. This script uses the admin interface providing elevated privileges to perform this action.

**AdminInterface/RefreshCache()**

Location: <CF\_FOLDER>/Scripts/AdminInterface/RefreshCache()

This procedure is used to kick off a cache refresh on a resource. Currently the default resource type is set to TABLE. This script uses the admin interface providing elevated privileges to perform this action. Refer back to [ApplicationTemplate/Execute/RefreshCache](#) for more details.

**AdminInterface/UpdateResourceOwner()**

Location: <CF\_FOLDER>/Scripts/AdminInterface/UpdateResourceOwner()

This procedure is used to change resource ownership of a resource. This script can be used to change ownership of a single resource or a folder with multiple resources. By default, the ownership is pushed recursively to all the resources in a folder. This script uses the admin interface providing elevated privileges to perform this action.

Executing the library function used in this script requires ACCESS TOOLS, READ ALL RESOURCES, READ ALL USERS role, therefore ideally, this script should be executed by an Admin user.

## 10 Appendix A – Configure Cache Framework Sample

The following is a discussion of how to augment the Cache Framework.

### Configuring the Cache Framework Sample

Configuring the Cache Framework sample is a one-time action that should be done by a DV architect with overall knowledge of the application and cache database target. There is a sample for each cache database type.

### Background Information

- 1) Samples include Application Template, Views and System of Record (SOR):
- 2) Purpose – The purpose of the sample is two-fold.
  - a. It provides a readymade data source that can be used for quickly learning and demonstrating the Cache Framework capabilities.
  - b. It serves as a base-line for testing the Cache Framework and verifying supported data types.
- 3) Assumptions: The Cache Framework has been installed at this location  
 <CS\_Folder>:  
       /shared/ASAssets/CacheManagement/CacheFrameworkSample
- 4) [Configure Oracle Sample](#)
  - a. <CS\_Folder>/Oracle\_Application – Cache Framework template for Oracle.
  - b. <CS\_Folder>/Oracle\_SampleViews – Sample views to configure caching with.
  - c. <CS\_Folder>/Oracle\_SampleSOR – Sample system of record database and scripts.
- 5) [Configure SQL Server Sample](#)
  - a. <CS\_Folder>/SqlServer\_Application – Cache Framework template for SQL Server.
  - b. <CS\_Folder>/SqlServer\_SampleViews – Sample views to configure caching with.
  - c. <CS\_Folder>/SqlServer\_SampleSOR – Sample system of record database and scripts.
- 6) [Configure Netezza Sample](#)
  - a. <CS\_Folder>/Netezza\_Application – Cache Framework template for Netezza.
  - b. <CS\_Folder>/Netezza\_SampleViews – Sample views to configure caching with.
  - c. <CS\_Folder>/Netezza\_SampleSOR – Sample system of record database and scripts.
- 7) [Configure MemSql Sample](#)
  - a. <CS\_Folder>/MemSql\_Application – Cache Framework template for MemSql.
  - b. <CS\_Folder>/MemSql\_SampleViews – Sample views to configure caching with.
  - c. <CS\_Folder>/MemSql\_SampleSOR – Sample system of record database and scripts.



## 8) [Configure Postgres Sample](#)

- a. <CS\_Folder>/Postgres\_Application – Cache Framework template for Postgres.
- b. <CS\_Folder>/Postgres\_SampleViews – Sample views to configure caching with.
- c. <CS\_Folder>/Postgres\_SampleSOR – Sample system of record database and scripts.

### Configure Oracle Sample:

#### 1) Application

- a. Location: <CS\_Folder>/Oracle\_Application
- b. Provides a cache target configured sample. It originated from <CF\_Folder>/ApplicationTemplateOracle. It points to the Oracle SOR for a cache target data source.

#### 2) Views

- a. Location: <CS\_Folder>/Oracle\_SampleViews
- b. Provides a sample view for each cache type. The CachingData procedure is already configured for this sample for the default location of the Cache Framework Sample. There is also a view that exercises the different data types. These views provide a baseline for testing and examples.

#### 3) SOR

- a. Location: <CS\_Folder>/Oracle\_SampleSOR
- b. The SAMPLE\_DS serves as both a system of record and the cache target database to simplify the connectivity for this sample.

#### 4) **Configure Oracle system of record samples**

- a. Select an Oracle database for creating a sample system of record
  - i. Create a user/schema with the name “CIS\_CACHE”
  - ii. Grant the user the ability to do the following:

– Provide password and tablespace details

```
CREATE USER CIS_CACHE IDENTIFIED BY <password> default tablespace ts_users temporary
tablespace temp;
```

```
GRANT CREATE SESSION TO CIS_CACHE;
```

```
GRANT CREATE SEQUENCE TO CIS_CACHE;
```

```
GRANT CREATE TABLE TO CIS_CACHE;
```

```
GRANT CREATE ANY INDEX TO CIS_CACHE;
```

```
GRANT CREATE TRIGGER TO CIS_CACHE;
```

```
GRANT DROP ANY SEQUENCE TO CIS_CACHE;
```

```
GRANT DROP ANY TABLE TO CIS_CACHE;
```

```
GRANT DROP ANY INDEX TO CIS_CACHE;
```

```
GRANT DROP ANY TRIGGER TO CIS_CACHE;
```

```
GRANT ALTER SESSION TO CIS_CACHE;
```

```

COMMIT;
-- Provide password and Database SID
CONNECT CIS_CACHE/<password>@<DB>;
ALTER SESSION SET NLS_LANGUAGE=American;
ALTER SESSION SET NLS_TERRITORY=America;
COMMIT;

```

- iii. Note: if a different schema is provided then the scripts will need to be rebound to that database and user once all of the tables are created and introspected.
- b. Open the SAMPLE\_DS data source connection.
  - i. Edit the connection details to point to the new
  - ii. Click "Test Connection" to verify that the connection is working.
- c. Execute create\_TEST\_INCR to create the following:
  - i. TEST\_INCR
  - ii. TEST\_INCR\_DEL
  - iii. TEST\_INCR\_HIST
  - iv. TEST\_INCR\_TRIGGER
  - v. TEST\_INCR\_DEL\_TRIGGER
- d. Re-introspect the SAMPLE\_DS data source.
- e. Validate the SOR tables.
  - i. Right click on ../SOR/TEST\_INCR and "Execute" to insure no error is thrown.
  - ii. Repeat for TEST\_INCR\_DEL
  - iii. Repeat for TEST\_INCR\_HIST
- f. Insert/Update/Delete rows as needed to test various cache types:
  - i. insert\_TEST\_INCR
  - ii. update\_TEST\_INCR
  - iii. deletePK\_TEST\_INCR

## 5) Configure Oracle Application Template

- a. RebindPackagedQueries
  - i. Location:  
<CS\_Folder>/Oracle\_Application/Initialize/RebindPackagedQueries
  - ii. Execute to rebind the two packaged queries [ExecutedDDL and GetIntResult] to the SAMPLE\_DS data source. There are no input

parameters. The information for rebinding is contained within the Constants procedure.

b. Initialize database tables

- i. Location: <CS\_Folder>/Oracle\_Application/Initialize
- ii. **CreateDBSequence** : Create DB sequence  
CACHE\_FRAMEWORK\_SEQ. Does not drop automatically.
- iii. **Create\_AUDIT\_LOG** : Will automatically drop if exists.
- iv. **Create\_AUDIT\_LOG\_SEQ** : Will automatically drop if exists.
- v. **Create\_CACHING\_DATA** : Will automatically drop if exists.

c. Execute CachingData\_Insert procedure

- i. Location: <CS\_Folder>/Oracle\_Application/DataScripts
- ii. **CachingData\_Insert(1)** – Execute to load rows into  
CACHING\_DATA table. Provide a 1 to delete existing rows before  
inserting.

## 6) Cache Configuration Generation

a. **ConfigureCache**

- i. Location: <CS\_Folder>/Oracle\_Application/Execute/ConfigureCache
- ii. Configure view cache, scripts, schedule and database tables.
- iii. Input: The path to view to be configured:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle\_SampleViews/Views/V\_TEST\_INCR\_HYB\_N

b. **ConfigureCache\_Multi\_Table**

- i. Location:  
<CS\_Folder>/Oracle\_Application/Execute/ConfigureCache\_Multi\_Table
- ii. Configure CachingData procedure, view cache, scripts and database tables.
- iii. Input: The path to view to be configured:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle\_SampleViews/Views/V\_TEST\_FULL\_MULTI

c. **RefreshCache**

- i. Location: <CS\_Folder>/Oracle\_Application/Execute/RefreshCache
- ii. Refresh the cache.
- iii. Input: The path to view to be configured:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle\_SampleViews/Views/V\_TEST\_INCR\_HYB\_N

**d. AuditLogDisplay**

- i. Report on the outcome of the cache or configuration.
- ii. Location: <CS\_Folder>/Oracle\_Application/Execute/AuditLogDisplay
- iii. Input: leave all blank to select all.

**e. DeconfigureCache**

- i. De-configure cache, scripts and database tables.
- ii. Location: <CS\_Folder>/Oracle\_Application/Execute/DeconfigureCache
- iii. Input: The path to view to be configured:  
         /shared/ASAssets/CacheManagement/CacheFrameworkSample/Oracle\_SampleViews/Views/V\_TEST\_INCR\_HYB\_N

**Configure SQL Server Sample:****1) Application**

- a. Location: <CS\_Folder>/SqlServer\_Application
- b. Provides a cache target configured sample. It originated from <CF\_Folder>/ApplicationTemplateSqlServer. It points to the Sql Server SOR for a cache target data source.

**2) Views**

- a. Location: <CS\_Folder>/SqlServer\_SampleViews
- b. Provides a sample view for each cache type. The CachingData procedure is already configured for this sample for the default location of the Cache Framework Sample. There is also a view that exercises the different data types. These views provide a baseline for testing and examples.

**3) SOR**

- a. Location: <CS\_Folder>/SqlServer\_sampleSOR
- b. The SAMPLE\_DS serves as both a system or record and the cache target database to simplify the connectivity for this sample.

**4) Configure SQL Server system of record samples**

- a. Select a SQL Server database for creating a sample system of record
  - i. Create a catalog with the name "CIS\_CACHE" and schema "dbo". Create a user CIS\_CACHE.
  - ii. Grant the user the ability to do the following:
 

```
USE [CIS_CACHE];
GRANT CREATE SEQUENCE ON SCHEMA::dbo TO [CIS_CACHE];
GRANT CREATE TABLE TO [CIS_CACHE];
GRANT ALTER ON SCHEMA::dbo TO [CIS_CACHE];
```

GO;

- iii. Note: if a different catalog and schema is provided then the scripts will need to be rebound to that database and user once all of the tables are created and introspected.
- b. Open the SAMPLE\_DS data source connection.
  - i. Click “Test Connection” to verify that the connection is working.
- c. Create database objects
  - i. SQL Server 2012 - Execute /SQL2012/create\_TEST\_INCR to create the following:
    - 1. TEST\_INCR
    - 2. TEST\_INCR\_DEL
    - 3. TEST\_INCR\_HIST
    - 4. TEST\_INCR\_TRIGGER
    - 5. TEST\_INCR\_DEL\_TRIGGER
  - ii. SQL Server 2008 - Execute /SQL2008/create\_TEST\_INCR\_2008 to create the following:
    - 1. TEST\_INCR
    - 2. TEST\_INCR\_DEL
    - 3. TEST\_INCR\_HIST
- d. Re-introspect the SAMPLE\_DS data source.
- e. Validate the SOR tables.
  - i. Right click on ../SOR/TEST\_INCR and "Execute" to insure no error is thrown.
  - ii. Repeat for TEST\_INCR\_DEL
  - iii. Repeat for TEST\_INCR\_HIST
- f. Insert/Update/Delete rows as needed using various scripts:
  - i. SQL Server 2012
    - 1. /SQL2012/insert\_TEST\_INCR
    - 2. /SQL2012/update\_TEST\_INCR
    - 3. /SQL2012/deletePK\_TEST\_INCR
  - ii. SQL Server 2008
    - 1. /SQL2008/insert\_TEST\_INCR\_2008
    - 2. /SQL2008/update\_TEST\_INCR\_2008

### 3. /SQL2008/deletePK\_TEST\_INCR\_2008

## 5) Configure SQL Server Application Template

### a. RebindPackagedQueries

- i. Location:  
`<CS_Folder>/SqlServer_Application/Initialize/RebindPackagedQueries`
- ii. Execute to rebind the two packaged queries [ExecuteDDL and GetIntResult] to the SAMPLE\_DS data source. There are no input parameters. The information for rebinding is contained within the Constants procedure.

### b. Initialize database tables

- i. Location: `<CS_Folder>/SqlServer_Application/Initialize`
- ii. **CreateDBSequence** : Create DB sequence CACHE\_FRAMEWORK\_SEQ. Does not drop automatically.
- iii. **Create\_AUDIT\_LOG** : Will automatically drop if exists.
- iv. **Create\_AUDIT\_LOG\_SEQ** : Will automatically drop if exists.
- v. **Create\_CACHING\_DATA** : Will automatically drop if exists.
- vi. Execute CachingData\_Insert procedure
  1. Location: `<CS_Folder>/SqlServer_Application/DataScripts`
  2. **CachingData\_Insert(1)** – Execute to load rows into CACHING\_DATA table. Provide a 1 to delete existing rows before inserting.

## 6) Cache Configuration Generation

### a. ConfigureCache

- i. Location: `<CS_Folder>/SqlServer_Application/Execute/ConfigureCache`
- ii. Configure cache, scripts, schedule and database tables.
- iii. Input: The path to view to be configured:  
`/shared/ASAssets/CacheManagement/CacheFrameworkSample/SqlServer_SampleViews/Views/V_TEST_INCR_HYB_N`

### b. ConfigureCache\_Multi\_Table

- i. Location:  
`<CS_Folder>/SqlServer_Application/Execute/ConfigureCache_Multi_Table`
- ii. Configure CachingData procedure, view cache, scripts and database tables.
- iii. Input: The path to view to be configured:

/shared/ASAssets/CacheManagement/CacheFrameworkSample/SqlServer\_SampleViews/Views/V\_TEST\_FULL\_MULTI

**c. RefreshCache**

- i. Location: <CS\_Folder>/SqlServer\_Application/Execute/RefreshCache
- ii. Refresh the cache.
- iii. Input: The path to view to be configured:

/shared/ASAssets/CacheManagement/CacheFrameworkSample/SqlServer\_SampleViews/Views/V\_TEST\_INCR\_HYB\_N

**d. AuditLogDisplay**

- i. Location: <CS\_Folder>/SqlServer\_Application/Execute/AuditLogDisplay
- ii. Report on the outcome of the cache or configuration.
- iii. Input: leave all blank to select all.

**e. DeconfigureCache**

- i. Location:  
<CS\_Folder>/SqlServer\_Application/Execute/DeconfigureCache
- ii. De-configure cache, scripts and database tables.
- iii. Input: The path to view to be configured:

/shared/ASAssets/CacheManagement/CacheFrameworkSample/SqlServer\_SampleViews/Views/V\_TEST\_INCR\_HYB\_N

**Configure Netezza Sample:**

**1) Application**

- a. Location: <CS\_Folder>/Netezza\_Application
- b. Provides a cache target configured sample. It originated from <CF\_Folder>/ApplicationTemplateNetezza. It points to the Netezza SOR for a cache target data source.

**2) Views**

- a. Location: <CS\_Folder>/Netezza\_SampleViews
- b. Provides a sample view for each cache type. The CachingData procedure is already configured for this sample for the default location of the Cache Framework Sample. There is also a view that exercises the different data types. These views provide a baseline for testing and examples.

**3) SOR**

- a. Location: <CS\_Folder>/Netezza\_SampleSOR
- b. The SAMPLE\_DS serves as both a system or record and the cache target database to simplify the connectivity for this sample.

#### 4) Configure Netezza system of record samples

- a. Select a Netezza database for creating a sample system of record
  - i. Create a database in the Netezza data source called "CIS\_CACHE" owned by the user ADMIN.
  - ii. Grant the user the ability to do the following:
 

```
GRANT LIST ON ADMIN TO ADMIN ;
GRANT CREATE SEQUENCE TO ADMIN ;
GRANT CREATE TABLE TO ADMIN ;
GRANT DROP ON SEQUENCE TO ADMIN ;
GRANT DROP ON TABLE TO ADMIN ;
COMMIT;
```
  - iii. Note: if a different database and user is provided then the scripts will need to be rebound to that database and user once all of the tables are created and introspected.
- b. Open the SAMPLE\_DS data source connection.
  - i. Edit the connection details to point to the new
  - ii. Click "Test Connection" to verify that the connection is working.
- c. Execute create\_TEST\_INCR to create the following:
  - i. TEST\_INCR
  - ii. TEST\_INCR\_DEL
  - iii. TEST\_INCR\_HIST
- d. Re-introspect the SAMPLE\_DS data source.
- e. Validate the SOR tables.
  - i. Right click on ../SOR/TEST\_INCR and "Execute" to insure no error is thrown.
  - ii. Repeat for TEST\_INCR\_DEL
  - iii. Repeat for TEST\_INCR\_HIST
- f. Insert/Update/Delete rows as needed to test various cache types:
  - i. insert\_TEST\_INCR
  - ii. update\_TEST\_INCR
  - iii. deletePK\_TEST\_INCR

#### 5) Configure Netezza Application Template

- a. RebindPackagedQueries
  - i. Location:
 

```
<CS_Folder>/Netezza_Application/Initialize/RebindPackagedQueries
```



- ii. Execute to rebind the two packaged queries [ExecutedDDL and GetIntResult] to the SAMPLE\_DS data source. There are no input parameters. The information for rebinding is contained within the Constants procedure.
- b. Initialize database tables
  - i. Location: <CS\_Folder>/Netezza\_Application/Initialize
  - ii. **CreatedBSequence** : Create DB sequence CACHE\_FRAMEWORK\_SEQ. Does not drop automatically.
  - iii. **Create\_AUDIT\_LOG** : Will automatically drop if exists.
  - iv. **Create\_AUDIT\_LOG\_SEQ** : Will automatically drop if exists.
  - v. **Create\_CACHING\_DATA** : Will automatically drop if exists.
- c. Execute CachingData\_Insert procedure
  - i. Location: <CS\_Folder>/Netezza\_Application/DataScripts
  - ii. **CachingData\_Insert(1)** – Execute to load rows into CACHING\_DATA table. Provide a 1 to delete existing rows before inserting.

## 6) Cache Configuration Generation

### a. ConfigureCache

- i. Location: <CS\_Folder>/Netezza\_Application/Execute/ConfigureCache
- ii. Configure cache, scripts, schedule and database tables.
- iii. Input: The path to view to be configured:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Netezza\_Sample Views/Views/V\_TEST\_INCR\_HYB\_N

### b. ConfigureCache\_Multi\_Table

- i. Location:  
<CS\_Folder>/Netezza\_Application/Execute/ConfigureCache\_Multi\_Table
- ii. Configure CachingData procedure, view cache, scripts and database tables.
- iii. Input: The path to view to be configured:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Netezza\_Sample Views/Views/V\_TEST\_FULL\_MULTI

### c. RefreshCache

- i. Location: <CS\_Folder>/Netezza\_Application/Execute/RefreshCache
- ii. Refresh the cache.
- iii. Input: The path to view to be configured:

/shared/ASAssets/CacheManagement/CacheFrameworkSample/Netezza\_Sample  
Views/Views/V\_TEST\_INCR\_HYB\_N

**d. AuditLogDisplay**

- i. Location: <CS\_Folder>/Netezza\_Application/Execute/AuditLogDisplay
- ii. Report on the outcome of the cache or configuration.
- iii. Input: leave all blank to select all.

**e. DeconfigureCache**

- i. Location: <CS\_Folder>/Netezza\_Application/Execute/DeconfigureCache
- ii. De-configure cache, scripts and database tables.
- iii. Input: The path to view to be configured:

/shared/ASAssets/CacheManagement/CacheFrameworkSample/Netezza\_Sample  
Views/Views/V\_TEST\_INCR\_HYB\_N

**Configure MemSql Sample:**

**1) Application**

- a. Location: <CS\_Folder>/MemSql\_Application
- b. Provides a cache target configured sample. It originated from <CF\_Folder>/ApplicationTemplateMemSql. It points to the MemSql SOR for a cache target data source.

**2) Views**

- a. Location: <CS\_Folder>/MemSql\_SampleViews
- b. Provides a sample view for each cache type. The CachingData procedure is already configured for this sample for the default location of the Cache Framework Sample. There is also a view that exercises the different data types. These views provide a baseline for testing and examples.

**3) SOR**

- a. Location: <CS\_Folder>/MemSql\_SampleSOR
- b. The SAMPLE\_DS serves as both a system or record and the cache target database to simplify the connectivity for this sample.

**4) Configure MemSql system of record samples**

- a. Select an MemSql database for creating a sample system of record
  - i. Create a user/schema with the name "ciscache"
  - ii. Grant the user the ability to do the following:

GRANT CREATE SEQUENCE TO ciscache ;

GRANT CREATE TABLE TO ciscache ;

GRANT CREATE TRIGGER TO ciscache ;

```

GRANT CREATE SESSION TO ciscache ;
GRANT CREATE ANY INDEX TO ciscache ;
GRANT DROP ANY SEQUENCE TO ciscache ;
GRANT DROP ANY TABLE TO ciscache ;
GRANT DROP ANY TRIGGER TO ciscache ;
GRANT DROP ANY INDEX TO ciscache ;
GRANT ALTER SESSION TO ciscache ;
COMMIT;

```

- iii. Note: if a different schema is provided then the scripts will need to be rebound to that database and user once all of the tables are created and introspected.
- b. Open the SAMPLE\_DS data source connection.
  - i. Edit the connection details to point to the new
  - ii. Click "Test Connection" to verify that the connection is working.
- c. Execute create\_TEST\_INCR to create the following:
  - i. TEST\_INCR
- d. Re-introspect the SAMPLE\_DS data source.
- e. Validate the SOR tables.
  - i. Right click on ../SOR/TEST\_INCR and "Execute" to insure no error is thrown.
- f. Insert/Update/Delete rows as needed to test various cache types:
  - i. insert\_TEST\_INCR
  - ii. update\_TEST\_INCR
  - iii. deletePK\_TEST\_INCR

## 5) Configure MemSql Application Template

- a. RebindPackagedQueries
  - i. Location:  
`<CS_Folder>/MemSql_Application/Initialize/RebindPackagedQueries`
  - ii. Execute to rebind the two packaged queries [ExecuteDDL and GetIntResult] to the SAMPLE\_DS data source. There are no input parameters. The information for rebinding is contained within the Constants procedure.
- b. Initialize database tables
  - i. Location: `<CS_Folder>/MemSql_Application/Initialize`
  - ii. **CreateDBSequence** : Create DB sequence  
 CACHE\_FRAMEWORK\_SEQ. Does not drop automatically.

- iii. **Create\_AUDIT\_LOG** : Will automatically drop if exists.
- iv. **Create\_AUDIT\_LOG\_SEQ** : Will automatically drop if exists.
- v. **Create\_CACHING\_DATA** : Will automatically drop if exists.

c. Execute CachingData\_Insert procedure

- i. Location: <CS\_Folder>/MemSql\_Application/DataScripts
- ii. **CachingData\_Insert(1)** – Execute to load rows into CACHING\_DATA table. Provide a 1 to delete existing rows before inserting.

## 6) Cache Configuration Generation

### a. ConfigureCache

- i. Location: <CS\_Folder>/MemSql\_Application/Execute/ConfigureCache
- ii. Configure view cache, scripts, schedule and database tables.
- iii. Input: The path to view to be configured:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample/MemSql\_Sample Views/Views/V\_TEST\_FULL\_SINGLE

### b. ConfigureCache\_Multi\_Table

- i. Location:  
<CS\_Folder>/MemSql\_Application/Execute/ConfigureCache\_Multi\_Table
- ii. Configure CachingData procedure, view cache, scripts and database tables.
- iii. Input: The path to view to be configured:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample/MemSql\_Sample Views/Views/V\_TEST\_FULL\_MULTI

### c. RefreshCache

- i. Location: <CS\_Folder>/MemSql\_Application/Execute/RefreshCache
- ii. Refresh the cache.
- iii. Input: The path to view to be configured:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample/MemSql\_Sample Views/Views/V\_TEST\_FULL\_SINGLE

### d. AuditLogDisplay

- i. Report on the outcome of the cache or configuration.
- ii. Location: <CS\_Folder>/MemSql\_Application/Execute/AuditLogDisplay
- iii. Input: leave all blank to select all.

### e. DeconfigureCache

- i. De-configure cache, scripts and database tables.
- ii. Location: <CS\_Folder>/MemSql\_Application/Execute/DeconfigureCache
- iii. Input: The path to view to be configured:  
 /shared/ASAssets/CacheManagement/CacheFrameworkSample/MemSql\_Sample  
 Views/Views/V\_TEST\_FULL\_SINGLE

### Configure Postgres Sample:

#### 1) Application

- a. Location: <CS\_Folder>/Postgres\_Application
- b. Provides a cache target configured sample. It originated from <CF\_Folder>/ApplicationTemplatePostgres. It points to the Postgres SOR for a cache target data source.

#### 2) Views

- a. Location: <CS\_Folder>/Postgres\_SampleViews
- b. Provides a sample view for each cache type. The CachingData procedure is already configured for this sample for the default location of the Cache Framework Sample. There is also a view that exercises the different data types. These views provide a baseline for testing and examples.

#### 3) SOR

- a. Location: <CS\_Folder>/Postgres\_SampleSOR
- b. The SAMPLE\_DS serves as both a system or record and the cache target database to simplify the connectivity for this sample.

#### 4) Configure Postgres system of record samples

- a. Select an Postgres database for creating a sample system of record
  - i. Create a user/schema with the name “ciscache”
  - ii. Grant the user the ability to do the following:

```
GRANT CREATE SEQUENCE TO ciscache ;
GRANT CREATE TABLE TO ciscache ;
GRANT CREATE TRIGGER TO ciscache ;
GRANT CREATE SESSION TO ciscache ;
GRANT CREATE ANY INDEX TO ciscache ;
GRANT DROP ANY SEQUENCE TO ciscache ;
GRANT DROP ANY TABLE TO ciscache ;
GRANT DROP ANY TRIGGER TO ciscache ;
GRANT DROP ANY INDEX TO ciscache ;
GRANT ALTER SESSION TO ciscache ;
COMMIT;
```

- iii. Note: if a different schema is provided then the scripts will need to be rebound to that database and user once all of the tables are created and introspected.
- b. Open the SAMPLE\_DS data source connection.
  - i. Edit the connection details to point to the new
  - ii. Click “Test Connection” to verify that the connection is working.
- c. Execute create\_TEST\_INCR to create the following:
  - i. TEST\_INCR
- d. Re-introspect the SAMPLE\_DS data source.
- e. Validate the SOR tables.
  - i. Right click on ../SOR/TEST\_INCR and "Execute" to insure no error is thrown.
- f. Insert/Update/Delete rows as needed to test various cache types:
  - i. insert\_TEST\_INCR
  - ii. update\_TEST\_INCR
  - iii. deletePK\_TEST\_INCR

## 5) Configure Oracle Application Template

- a. RebindPackagedQueries
  - i. Location:
    - <CS\_Folder>/Postgres\_Application/Initialize/RebindPackagedQueries
  - ii. Execute to rebind the two packaged queries [ExecutedDDL and GetIntResult] to the SAMPLE\_DS data source. There are no input parameters. The information for rebinding is contained within the Constants procedure.
- b. Initialize database tables
  - i. Location: <CS\_Folder>/Postgres\_Application/Initialize
  - ii. **CreateDBSequence** : Create DB sequence  
CACHE\_FRAMEWORK\_SEQ. Does not drop automatically.
  - iii. **Create\_AUDIT\_LOG** : Will automatically drop if exists.
  - iv. **Create\_AUDIT\_LOG\_SEQ** : Will automatically drop if exists.
  - v. **Create\_CACHING\_DATA** : Will automatically drop if exists.
- c. Execute CachingData\_Insert procedure
  - i. Location: <CS\_Folder>/Postgres\_Application/DataScripts

- ii. **CachingData\_Insert(1)** – Execute to load rows into CACHING\_DATA table. Provide a 1 to delete existing rows before inserting.

## 6) Cache Configuration Generation

### a. ConfigureCache

- i. Location: <CS\_Folder>/Postgres\_Application/Execute/ConfigureCache
- ii. Configure view cache, scripts, schedule and database tables.
- iii. Input: The path to view to be configured:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Postgres\_Sample Views/Views/V\_TEST\_FULL\_SINGLE

### b. ConfigureCache\_Multi\_Table

- i. Location:  
<CS\_Folder>/Postgres\_Application/Execute/ConfigureCache\_Multi\_Table
- ii. Configure CachingData procedure, view cache, scripts and database tables.
- iii. Input: The path to view to be configured:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Postgres\_Sample Views/Views/V\_TEST\_FULL\_MULTI

### c. RefreshCache

- i. Location: <CS\_Folder>/Postgres\_Application/Execute/RefreshCache
- ii. Refresh the cache.
- iii. Input: The path to view to be configured:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Postgres\_Sample Views/Views/V\_TEST\_FULL\_SINGLE

### d. AuditLogDisplay

- i. Report on the outcome of the cache or configuration.
- ii. Location: <CS\_Folder>/Postgres\_Application/Execute/AuditLogDisplay
- iii. Input: leave all blank to select all.

### e. DeconfigureCache

- i. De-configure cache, scripts and database tables.
- ii. Location:  
<CS\_Folder>/Postgres\_Application/Execute/DeconfigureCache
- iii. Input: The path to view to be configured:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample/Postgres\_Sample Views/Views/V\_TEST\_FULL\_SINGLE

## 11 Appendix B – Merge Cache Types

The following is a discussion of the different Merge cache types that are supported.

### Merge Cache Types

Adding a new cache type involves the following:

#### Merge and Slowly Changing Dimensions <sup>1</sup>

**Dimension** is a term in data management and data warehousing. It's the logical groupings of data such as geographical location, customer or product information. With **Slowly Changing Dimensions (SCDs)** data changes slowly, rather than changing on a time-based, regular schedule.

For example, you may have a dimension in your database that tracks the sales records of your company's salespeople. Creating sales reports seems simple enough, until a salesperson is transferred from one regional office to another. How do you record such a change in your sales dimension?

You could calculate the sum or average of each salesperson's sales, but if you use that to compare the performance of salespeople, that might give misleading information. If the salesperson was transferred and used to work in a hot market where sales were easy, and now works in a market where sales are infrequent, his/her totals will look much stronger than the other salespeople in their new region. Or you could create a second salesperson record and treat the transferred person as a new sales person, but that creates problems.

Dealing with these issues involves SCD management methodologies referred to as Type 0 through 6. Type 6 SCDs are also sometimes called Hybrid SCDs.

#### Type 0

The **Type 0** method is passive. It manages dimensional changes and no action is performed. Values remain as they were at the time the dimension record was first inserted. In certain circumstances history is preserved with a Type 0. High order types are employed to guarantee the preservation of history whereas Type 0 provides the least or no control.

The most common types are I, II, and III.

#### Type 1

This methodology overwrites old with new data, and therefore does not track historical data. Example of a supplier table:

<sup>1</sup> See [Wikipedia Slowly Changing Dimensions example](#) by Bruce Ottman and Ralph Kimball



| Supplier_Key | Supplier_Code | Supplier_Name  | Supplier_State |
|--------------|---------------|----------------|----------------|
| 123          | ABC           | Acme Supply Co | CA             |

In the above example, Supplier\_Code is the natural key and Supplier\_Key is a surrogate key. Technically, the surrogate key is not necessary, since the row will be unique by the natural key (Supplier\_Code). However, to optimize performance on joins use integer rather than character keys. If the supplier relocates the headquarters to Illinois the record would be overwritten:

| Supplier_Key | Supplier_Code | Supplier_Name  | Supplier_State |
|--------------|---------------|----------------|----------------|
| 123          | ABC           | Acme Supply Co | IL             |

The disadvantage of the Type I method is that there is no history in the data warehouse. It has the advantage however that it's easy to maintain.

If you have calculated an aggregate table summarizing facts by state, it will need to be recalculated when the Supplier\_State is changed.

## Type 2

This method tracks historical data by creating multiple records for a given natural key in the dimensional tables with separate surrogate keys and/or different version numbers. Unlimited history is preserved for each insert.

For example, if the supplier relocates to Illinois the version numbers will be incremented sequentially:

| Supplier_Key | Supplier_Code | Supplier_Name  | Supplier_State | Version |
|--------------|---------------|----------------|----------------|---------|
| 123          | ABC           | Acme Supply Co | CA             | 0       |
| 124          | ABC           | Acme Supply Co | IL             | 1       |

Another method is to add 'effective date' columns.

| Supplier_Key | Supplier_Code | Supplier_Name  | Supplier_State | Start_Date  | End_Date    |
|--------------|---------------|----------------|----------------|-------------|-------------|
| 123          | ABC           | Acme Supply Co | CA             | 01-Jan-2000 | 21-Dec-2004 |
| 124          | ABC           | Acme Supply Co | IL             | 22-Dec-2004 |             |

The null End\_Date in row two indicates the current tuple version. In some cases, a standardized surrogate high date (e.g. 9999-12-31) may be used as an end date, so that the field can be included in an index, and so that null-value substitution is not required when querying.

Transactions that reference a particular surrogate key (Supplier\_Key) are then permanently bound to the time slices defined by that row of the slowly changing dimension table. An aggregate table summarizing facts by state continues to reflect the historical state, i.e. the state the supplier was in at the time of the transaction; no update is needed.

If there are retrospective changes made to the contents of the dimension, or if new attributes are added to the dimension (for example a Sales\_Rep column) which have different effective dates from those already defined, then this can result in the existing transactions needing to be updated to reflect the new situation. This can be an expensive database operation, so Type 2 SCDs are not a good choice if the dimensional model is subject to change.

### Type 3

This method tracks changes using separate columns and preserves limited history. The Type 3 preserves limited history as it is limited to the number of columns designated for storing historical data. The original table structure in Type 1 and Type 2 is the same but Type III adds additional columns. In the following example, an additional column has been added to the table to record the supplier's original state - only the previous history is stored.

| Supplier_Key | Supplier_Code | Supplier_Name  | Original_Supplier_State | Effective_Date | Current_Supplier_State |
|--------------|---------------|----------------|-------------------------|----------------|------------------------|
| 123          | ABC           | Acme Supply Co | CA                      | 22-Dec-2004    | IL                     |

This record contains a column for the original state and current state—cannot track the changes if the supplier relocates a second time.

One variation of this is to create the field Previous\_Supplier\_State instead of Original\_Supplier\_State which would track only the most recent historical change.

### Type 4 (CDC style)

The **Type 4** method is usually referred to as using "history tables", where one table keeps the current data and an additional table is used to keep a record of some or all changes. Both the surrogate keys are referenced in the Fact table to enhance query performance.

For the above example the original table name is **Supplier** and the history table is **Supplier\_History**.

Supplier:

| Supplier_key | Supplier_Code | Supplier_Name  | Supplier_State |
|--------------|---------------|----------------|----------------|
| 123          | ABC           | Acme Supply Co | IL             |

Supplier\_History:

| Supplier_key | Supplier_Code | Supplier_Name  | Supplier_State | Create_Date |
|--------------|---------------|----------------|----------------|-------------|
| 123          | ABC           | Acme Supply Co | CA             | 22-Dec-2004 |

This method resembles how database audit tables and **change data capture** techniques function.

### Merge Type 1 – Cache Framework Example

- 1) This methodology overwrites old with new data, and therefore does not track historical data. The implementation shown below does incorporate a start date which can be used to find changes to the “AS-IS” data. History is not maintained with a type 1 table.
- 2) Uses a “delete” history table to capture deletes only. One possible mechanism for capturing deletes is a trigger.
- 3) The Cache Framework will key off of both the system of record and history table for changes. The delta load scripts will look at the SDT timestamp field for changes greater than the last incremental cache refresh maintenance date to pull inserts or updates. The delta load script will look at the delete history table for deletes where the SDT timestamp field is greater than the cache refresh maintenance date.
- 4) The following is a description of the interaction between a system of record table and a “delete” history table.
- 5) SEQ = table sequence, K1 = primary key column 1, K2 = primary key column 2, SDT = Start Date

System of Record Table:

| SEQ | K1 | K2  | C3    | C4   | SDT                 |
|-----|----|-----|-------|------|---------------------|
| 4   | 11 | id1 | Desc1 | 1.00 | 2014-11-16 14:00:00 |
| 2   | 12 | id2 | Desc2 | 2.00 | 2014-11-17 14:00:00 |
| 3   | 13 | id3 | Desc3 | 3.00 | 2014-11-18 14:00:00 |
| 4   | 14 | id4 | Desc4 | 4.00 | 2014-11-19 14:00:00 |

DELETE TRIGGER

DELETE TRIGGER

Delete Table:

| SEQ | K1 | K2  | C3    | C4   | SDT                 |
|-----|----|-----|-------|------|---------------------|
| 3   | 13 | id3 | Desc3 | 3.00 | 2014-11-20 10:00:00 |
| 1   | 11 | id1 | Desc1 | 1.00 | 2014-11-20 11:00:00 |

System of Record Table (Post Delete Operation):

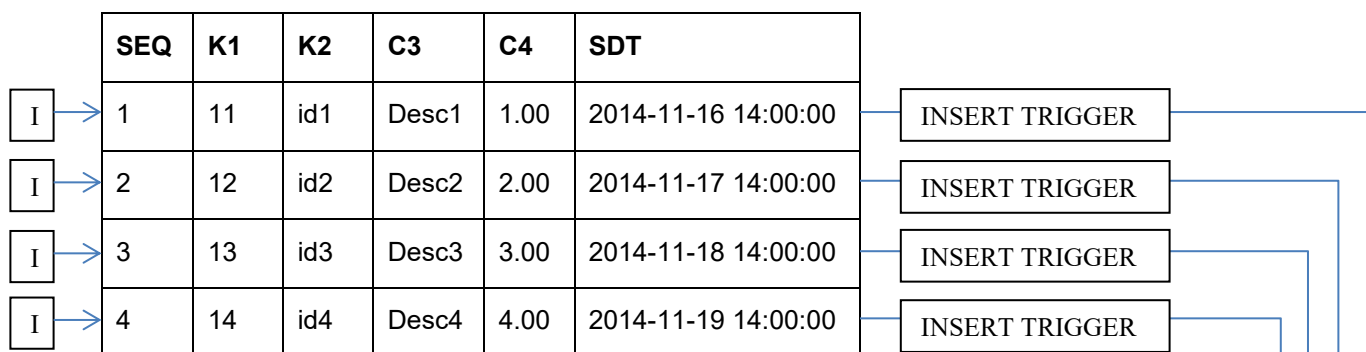
| SEQ | K1 | K2 | C3 | C4 | SDT |
|-----|----|----|----|----|-----|
|-----|----|----|----|----|-----|

|   |    |     |       |      |                     |
|---|----|-----|-------|------|---------------------|
| 2 | 12 | id2 | Desc2 | 2.00 | 2014-11-17 14:00:00 |
| 4 | 14 | id4 | Desc4 | 4.00 | 2014-11-19 14:00:00 |

### Merge Type 2 – Cache Framework Example

- 1) Uses a history table to capture inserts, updates and deletes. One possible mechanism for capturing inserts, updates and deletes is a trigger.
- 2) The Cache Framework will key off of both the system of record and history table for changes. The delta load scripts will look at the SDT timestamp field for changes greater than the last incremental cache refresh maintenance date to pull inserts or updates. The delta load script will look at the history table for deletes where the SDT timestamp field is greater than the cache refresh maintenance date and the EDT timestamp is not equal to the surrogate '9999-12-31 00:00:00' and the record does not exist in the system of record table.
- 3) The following is a description of the interaction between a system of record table and a history table.
- 4) SEQ = table sequence, K1 = primary key column 1, K2 = primary key column 2, SDT = Start Date, EDT = End Date [a surrogate key value of 9999-12-31 may be used to denote an end date so that the EDT value is not NULL when the record is not closed out.]

System of Record Table (Initial Insert Operations):



|   |    |     |       |      |                     |                     |
|---|----|-----|-------|------|---------------------|---------------------|
| 4 | 14 | id4 | Desc4 | 4.00 | 2014-11-19 14:00:00 | 1999-12-31 00:00:00 |
|---|----|-----|-------|------|---------------------|---------------------|

System of Record Table (Update and Delete Operation):

| SEQ   | K1        | K2         | C3           | C4          | SDT                        |                |
|-------|-----------|------------|--------------|-------------|----------------------------|----------------|
| 1     | 11        | id1        | Desc1        | 1.00        | 2014-11-16 14:00:00        |                |
| U → 2 | 12        | id2        | <b>D2.1</b>  | <b>2.01</b> | <b>2014-11-20 20:00:00</b> | UPDATE TRIGGER |
| 3     | 13        | id3        | Desc3        | 3.00        | 2014-11-18 14:00:00        |                |
| D → 4 | <b>14</b> | <b>id4</b> | <b>Desc4</b> | <b>4.00</b> | <b>2014-11-19 14:00:00</b> | DELETE TRIGGER |

History Table (Post SOR Update and Delete):

| SEQ      | K1        | K2         | C3           | C4          | SDT                        | EDT                        |
|----------|-----------|------------|--------------|-------------|----------------------------|----------------------------|
| 1        | 11        | id1        | Desc1        | 1.00        | 2014-11-16 14:00:00        | 1999-12-31 00:00:00        |
| 2        | <b>12</b> | <b>id2</b> | Desc2        | 2.00        | 2014-11-17 14:00:00        | <b>2014-11-20 20:00:00</b> |
| 3        | 13        | id3        | Desc3        | 3.00        | 2014-11-18 14:00:00        | 1999-12-31 00:00:00        |
| 4        | 14        | id4        | Desc4        | 4.00        | 2014-11-19 14:00:00        | <b>2014-11-20 10:00:00</b> |
| <b>5</b> | <b>14</b> | <b>id4</b> | <b>Desc4</b> | <b>4.00</b> | <b>2014-11-20 10:00:00</b> | <b>2014-11-20 10:00:00</b> |
| <b>6</b> | <b>12</b> | <b>id2</b> | <b>D2.1</b>  | <b>2.01</b> | <b>2014-11-20 20:00:00</b> | <b>1999-12-31 00:00:00</b> |

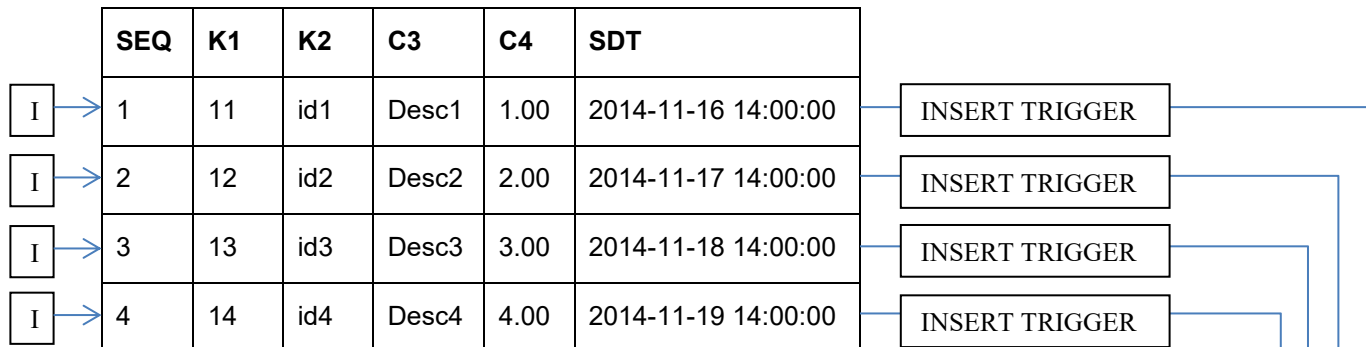
System of Record Table (Post Delete and Update Operation):

| SEQ | K1 | K2  | C3    | C4   | SDT                 |
|-----|----|-----|-------|------|---------------------|
| 1   | 11 | id1 | Desc1 | 1.00 | 2014-11-16 14:00:00 |
| 2   | 12 | id2 | D2.1  | 2.01 | 2014-11-20 20:00:00 |
| 3   | 13 | id3 | Desc3 | 3.00 | 2014-11-18 14:00:00 |

### Merge Type 4 – Cache Framework Example

- 1) Also known as Change Data Capture (CDC) style of merge where it tracks changes based on an activity flag either in the table or through an external CDC process that acts upon the system of record and history table to keep both in synch.
- 2) The Cache Framework will key off of both the system of record and history table for changes. The delta load scripts will look at the SDT timestamp field for changes greater than the last incremental cache refresh maintenance date to pull inserts or updates. The delta load script will look at the history table for deletes where the SDT timestamp field is greater than the cache refresh maintenance date and ACTIVITY='D'.
- 3) Uses a history table to capture inserts, updates and deletes. One possible mechanism for capturing inserts, updates and deletes is a trigger.
- 4) The following is a description of the interaction between a system of record table and a history table.
- 5) SEQ = table sequence, K1 = primary key column 1, K2 = primary key column 2, SDT = Start Date, ACTIVITY = CDC flag such as I,U,D for insert, update, delete operation.

System of Record Table (Initial Insert Operations):



History Table (Post SOR Insert):

| SEQ | K1 | K2  | C3    | C4   | SDT                 | ACTIVITY |
|-----|----|-----|-------|------|---------------------|----------|
| 1   | 11 | id1 | Desc1 | 1.00 | 2014-11-16 14:00:00 | I        |
| 2   | 12 | id2 | Desc2 | 2.00 | 2014-11-17 14:00:00 | I        |
| 3   | 13 | id3 | Desc3 | 3.00 | 2014-11-18 14:00:00 | I        |
| 4   | 14 | id4 | Desc4 | 4.00 | 2014-11-19 14:00:00 | I        |

System of Record Table (Update and Delete Operation):

| SEQ   | K1        | K2         | C3           | C4          | SDT                        |                |
|-------|-----------|------------|--------------|-------------|----------------------------|----------------|
| 1     | 11        | id1        | Desc1        | 1.00        | 2014-11-16 14:00:00        |                |
| U → 2 | 12        | id2        | <b>D2.1</b>  | <b>2.01</b> | <b>2014-11-20 20:00:00</b> | UPDATE TRIGGER |
| 3     | 13        | id3        | Desc3        | 3.00        | 2014-11-18 14:00:00        |                |
| D → 4 | <b>14</b> | <b>id4</b> | <b>Desc4</b> | <b>4.00</b> | <b>2014-11-19 14:00:00</b> | DELETE TRIGGER |

History Table (Post SOR Update and Delete):

| SEQ      | K1        | K2         | C3           | C4          | SDT                        | ACTIVITY |
|----------|-----------|------------|--------------|-------------|----------------------------|----------|
| 1        | 11        | id1        | Desc1        | 1.00        | 2014-11-16 14:00:00        | I        |
| 2        | 12        | id2        | Desc2        | 2.00        | 2014-11-17 14:00:00        | I        |
| 3        | 13        | id3        | Desc3        | 3.00        | 2014-11-18 14:00:00        | I        |
| 4        | 14        | id4        | Desc4        | 4.00        | 2014-11-19 14:00:00        | I        |
| <b>5</b> | <b>14</b> | <b>id4</b> | <b>Desc4</b> | <b>4.00</b> | <b>2014-11-20 10:00:00</b> | <b>D</b> |
| <b>6</b> | <b>12</b> | <b>id2</b> | <b>D2.1</b>  | <b>2.01</b> | <b>2014-11-20 20:00:00</b> | <b>U</b> |

System of Record Table (Post Delete and Update Operation):

| SEQ | K1 | K2  | C3    | C4   | SDT                 |
|-----|----|-----|-------|------|---------------------|
| 1   | 11 | id1 | Desc1 | 1.00 | 2014-11-16 14:00:00 |
| 2   | 12 | id2 | D2.1  | 2.01 | 2014-11-20 20:00:00 |
| 3   | 13 | id3 | Desc3 | 3.00 | 2014-11-18 14:00:00 |

## 12 Appendix C – Adding a New Cache Type

The following is a discussion of how to augment the Cache Framework.

### Adding a New Cache Type to the Source Code

Adding a new cache type involves the following:

- 1) To add a new cache type, modify these procedures and create an implementation procedure
  - a. Create a new implementation procedure that follows the pattern of one like
    - i. GenerateCacheScriptsFullCallback
    - ii. GenerateCacheScriptsHybridDeltaNoStage
    - iii. GenerateCacheScriptsHybridDeltaStage
    - iv. GenerateCacheScriptsIncremental
    - v. GenerateCacheScriptsHybridDeltaMergeType1
    - vi. GenerateCacheScriptsHybridDeltaMergeType2
    - vii. GenerateCacheScriptsHybridDeltaMergeType4
  - b. List of procedures to modify
    - i. CommonTypes.ValidCacheTypesAll
    - ii. ConfigureCacheScript
    - iii. DeleteReloadCache
    - iv. UpdateCacheConfiguration
    - v. PublishedImpl/DeployCache
    - vi. HelperScripts/ValidateTableDependency
  - c. List of Template procedures to modify
    - i. ApplicationTemplateMemSql/DataScripts/CachingData - example
    - ii. ApplicationTemplateMemSql/Execute/ConfigureCacheAdhoc - comments
    - iii. ApplicationTemplateOracle/DataScripts/CachingData - example
    - iv. ApplicationTemplateOracle/Execute/ConfigureCacheAdhoc - comments
    - v. ApplicationTemplateNetezza/DataScripts/CachingData - example
    - vi. ApplicationTemplateNetezza/Execute/ConfigureCacheAdhoc - comments
    - vii. ApplicationTemplatePostgres/DataScripts/CachingData - example



- viii. ApplicationTemplatePostgres/Execute/ConfigureCacheAdhoc - comments
- ix. ApplicationTemplateSqlServer/DataScripts/CachingData - example
- x. ApplicationTemplateSqlServer/Execute/ConfigureCacheAdhoc - comments

## 13 Appendix D – Adding a New Cache Data Source Type

The following is a discussion of how to augment the Cache Framework when adding a new cache target data source.

### Adding a New Cache Data Source to the Source Code

Adding a new cache type involves the following:

Assumptions:

- The Cache Framework has been installed at this location <CF\_Folder>:  
/shared/ASAssets/CacheManagement/CacheFramework
- The Cache Framework Sample has been installed at this location <CS\_Folder>:  
/shared/ASAssets/CacheManagement/CacheFrameworkSample

**Step 1.** Determine what features are supported by the database so they can be mapped to capabilities in the CommonTypes

- 1) Map TDV data types to native data types

Example: Oracle is shown as an example.

All DV types that are supported are listed in the first column "DV\_TYPE".

The mapping exercise is to determine what the "NATIVE\_TYPE" is by doing internet research and creating a sample cache table using V\_TEST\_DATATYPES. The DV adapter for the given data source will generate the data types. This will provide a way to determine what the mapping should be. Pay attention to what the min and max precision should be for CHAR, VARCHAR, NUMERIC, DECIMAL, BINARY and VARBINARY. Just because a data base supports one precision, does not mean that the DV data source adapter will support that. For example, it seems to be a general rule that DV supports NUMERIC/DECIMAL(1 through 38). Beyond that, DV will use a CLOB field.

The USE\_PRECISION columns instructs "GenerateColumnMappings" to use the given precision from the DV view. Y=use precision. N=do not use precision. For example VARCHAR(255) would map to VARCHAR2(255). VARCHAR(4001) would map to CLOB with no precision.

SET OracleDataTypeMapping =

| --DV_TYPE                                                                                                 | NATIVE_TYPE  | MIN_PRECISION | MAX_PRECISION | USE_PRECISION |
|-----------------------------------------------------------------------------------------------------------|--------------|---------------|---------------|---------------|
| -- String types                                                                                           |              |               |               |               |
| 'CHAR                                                                                                     | CHAR         | 1             | 2000          | Y             |
| CHAR                                                                                                      | CLOB         | 2001          | NULL          | N             |
| VARCHAR                                                                                                   | VARCHAR2     | 1             | 4000          | Y             |
| VARCHAR                                                                                                   | CLOB         | 4001          | NULL          | N             |
| LONGVARCHAR                                                                                               | CLOB         | NULL          | NULL          | N             |
| XML                                                                                                       | CLOB         | NULL          | NULL          | N             |
| CLOB                                                                                                      | CLOB         | NULL          | NULL          | N             |
| -- Boolean types                                                                                          |              |               |               |               |
| -- NOTE: Even though BOOLEAN to NUMBER(1,0) is supported by the adapter, it is not supported for caching. |              |               |               |               |
| BOOLEAN                                                                                                   | NUMBER(1,0)  | NULL          | NULL          | N             |
| -- Integer types                                                                                          |              |               |               |               |
| BIT                                                                                                       | NUMBER(1,0)  | NULL          | NULL          | N             |
| TINYINT                                                                                                   | NUMBER(3,0)  | NULL          | NULL          | N             |
| SMALLINT                                                                                                  | NUMBER(5,0)  | NULL          | NULL          | N             |
| INTEGER                                                                                                   | NUMBER(10,0) | NULL          | NULL          | N             |
| BIGINT                                                                                                    | NUMBER(19,0) | NULL          | NULL          | N             |
| -- Decimal/Numeric types                                                                                  |              |               |               |               |
| DECIMAL                                                                                                   | NUMBER       | 1             | 38            | Y             |
| DECIMAL                                                                                                   | CLOB         | 39            | NULL          | N             |
| NUMERIC                                                                                                   | NUMBER       | 1             | 38            | Y             |

|                 |               |      |      |     |
|-----------------|---------------|------|------|-----|
| NUMERIC         | CLOB          | 39   | NULL | N   |
| DOUBLE          | BINARY_DOUBLE | NULL | NULL | N   |
| FLOAT           | BINARY_FLOAT  | NULL | NULL | N   |
| REAL            | BINARY_FLOAT  | NULL | NULL | N   |
| -- Date types   |               |      |      |     |
| DATE            | DATE          | NULL | NULL | N   |
| TIMESTAMP       | TIMESTAMP(9)  | NULL | NULL | N   |
| TIME            | VARCHAR2(15)  | NULL | NULL | N   |
| INTERVAL DAY    | VARCHAR2(30)  | NULL | NULL | N   |
| INTERVAL MONTH  | VARCHAR2(9)   | NULL | NULL | N   |
| INTERVAL YEAR   | VARCHAR2(9)   | NULL | NULL | N   |
| -- Binary types |               |      |      |     |
| BLOB            | BLOB          | NULL | NULL | N   |
| BINARY          | BLOB          | 1    | NULL | N   |
| VARBINARY       | RAW(1024)     | NULL | NULL | N'; |

- 2) Syntax for Table exists
- 3) Syntax for Index exists
- 4) Syntax for Create table
- 5) Syntax for Drop table
- 6) Syntax for Create index
- 7) Syntax for Create unique index
- 8) Syntax for Drop index
- 9) Syntax for Create sequence
- 10) Syntax for Drop sequence
- 11) Syntax for Get sequence
- 12) Syntax for Execute table statistics
- 13) Object begin and end separator such as " or `
- 14) Maximum object length including 2 characters for a separator
- 15) Syntax for Truncate table
- 16) Syntax for Merge operation
- 17) Example of a Merge operation
- 18) Syntax to convert string to timestamp - used in  
`<CF_Folder>/Scripts/PublishedImpl/PurgeCacheData`

**Step 2.** The purpose of this procedure is to provide guidance what to modify when porting to a new cache database.

- 1) Modify the following code resources
  1.
    - a. `<CF_Folder>/Scripts/CommonTypes`
    - b. `<CF_Folder>/Scripts/ResetIncrMaintenanceLevel`
    - c. `<CF_Folder>/Scripts/HelperScripts/CreateIndex`
    - d. `<CF_Folder>/Scripts/HelperScripts/DropIndex`
    - e. `<CF_Folder>/Scripts/HelperScripts/DropTable`
    - f. `<CF_Folder>/Scripts/HelperScripts/ExecuteTableStatistics`
    - g. `<CF_Folder>/Scripts/HelperScripts/GetDBResourceValue`  
 [used to test for database objects. Logic depends on syntax of  
`<databasetype>ObjectTableExists` and `<databasetype>ObjectIndexExists`]
- 2) Copy template `<CF_Folder>/ApplicationTemplateOracle` to  
`<CF_Folder>/ApplicationTemplateNewDBType`
  - a. `<CF_Folder>/ApplicationTemplateNewDBType/Constants/Constants`
  - b. `<CF_Folder>/ApplicationTemplateNewDBType/DataScripts/CachingData`

- 3) Copy sample <CF\_Folder>Sample/Oracle\_Application to <CF\_Folder>Sample/NewDBType\_SampleSOR
  - a. Recreate the TEMP\_DS datasource in the type of the cache database.
  - b. <CF\_Folder>Sample/NewDBType\_Application/Constants/Constants
  - c. <CF\_Folder>Sample/NewDBType\_Application/DataScripts/CachingData
- 4) Copy <CF\_Folder>Sample/Oracle\_SampleSOR to <CF\_Folder>Sample/NewDBType\_SampleSOR
  - a. Recreate the SAMPLE\_DS datasource in the type of the cache database.
  - b. <CF\_Folder>Sample/NewDBType\_SampleSOR/SOR/create\_TEST\_INCR
  - c. <CF\_Folder>Sample/NewDBType\_SampleSOR/SOR/insert\_TEST\_INCR
- 5) Copy <CF\_Folder>Sample/Oracle\_SampleViews to <CF\_Folder>Sample/NewDBType\_SampleViews and modify the source path
  - a. <CF\_Folder>Sample/NewDBType\_SampleViews/Views/V\_TEST\_DATATYPES
  - b. <CF\_Folder>Sample/NewDBType\_SampleViews/Views/V\_TEST\_FULL\_MULTI
  - c. <CF\_Folder>Sample/NewDBType\_SampleViews/Views/V\_TEST\_FULL\_SINGLE
  - d. <CF\_Folder>Sample/NewDBType\_SampleViews/Views/V\_TEST\_INCR

-- Copy examples only if the cache data base supports merge otherwise they cannot be used.

  - e. <CF\_Folder>Sample/NewDBType\_SampleViews/Views/V\_TEST\_INCR\_HYB\_MT1
  - f. <CF\_Folder>Sample/NewDBType\_SampleViews/Views/V\_TEST\_INCR\_HYB\_MT2
  - g. <CF\_Folder>Sample/NewDBType\_SampleViews/Views/V\_TEST\_INCR\_HYB\_MT4
  - h. <CF\_Folder>Sample/NewDBType\_SampleViews/Views/V\_TEST\_INCR\_HYB\_N
  - i. <CF\_Folder>Sample/NewDBType\_SampleViews/Views/V\_TEST\_INCR\_HYB\_S
- 6) Test the CommonTypes data mapping using the sample V\_TEST\_DATATYPES
  - a. After manually creating the cache, excute the DDL and copy the syntax of the create. Compare the DDL with the CommonTypes data mapping. Whatever is being generated by the adapter for that database should match up with what was configured for the CommonTypes data mapping. Adjust the data mapping accordingly. Look for the min/max precision for CHAR, VARCHAR, DECIMAL, NUMERIC, BINARY and VARBINARY. Test the min/max based on what the database precisions are defined but be prepared to change them based on what the adapter supports.
  - b. After the cache table has been created and introspected, execute the following procedure which should return a 0, SUCCESS if all of the view generated columns match the actual native types coming from the cached view. If a 1, SUCCESS is returned, then the there was a mismatch on the column types. Adjust the CommonTypes data mapping according to the native types.

/shared/ASAssets/CacheManagement/CacheFramework/Scripts/HelperScripts/DeployCacheVerification

  - c. It is very important that the DeployCacheVerification returns a 0 (match) so that "DeployCache" will work correctly and not indiscriminately recreate cache tables since it invokes DeployCacheVerification to determine whether to create/recreate the tables or not.