Documentation

Advanced Services Assets

Pull-Based Incremental Caching

Data Virtualization Business Unit Advanced Services

March 2013

TABLE OF CONTENTS

DOCUMENT CONTROL

Version History

Version	Date	Author	Description
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

Related Documents

Document	Date	Author
6.2 SP1 Users Guide (or later)		

Data Virtualization Business Unit (DVBU) Products Referenced

DVBU Product Name	Version
Composite Information Server 6.2	6.2.1 or later
AS DVBU Utilities	2014Q2

INTRODUCTION

Purpose

With the release of CIS 6.2 SP1, CIS now features a framework to support the development of custom pull-based incremental caching solutions. This document describes such a solution developed in the field by the AS team. At this time it only supports incremental caching of resources of views or tables (procedure output is not supported.)

Audience

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

- Architects.
- Developers.
- CIS Administrators.

Caching Overview

Caching in CIS 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 CIS 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.)

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 CIS are described in Appendix F of the CIS 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 CIS 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.

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.

INSTALLATION

Requirements for Installation

- Composite Information Server 6.2 with Service Pack 1 or higher installed.
- AS Development Utilities version 2014Q2 installed.
- A caching database must be created, introspected, and configured for caching. Currently only Oracle is supported as a caching data source "out of the box". MySQL and SQL Server support is planned for a future release. However, other databases can be supported by engaging an AS engineer.

Installation Steps

- Create or identify a folder to house the incremental caching code. By default, the solution expects to be installed in the `/shared/ASAssets` folder, but other locations can be used. When the incremental caching solution is imported, the structure `/shared/ASAssets/CacheManagement/IncrementalCache` is created. In the rest of this document `<IC_Folder>` will be used to reference this location.
- Import the `CacheManagement_YYYYMMDD.car` file into the CIS instance into `/shared/ASAssets` (or other desired location.)
- Edit the `<IC_Folder>/Configuration/Constants` procedure and look for the declaration of the `BASE_FOLDER` constant. This variable is used to represent the `<IC_Folder>` location. Edit the default value if it is not correct for your installation.
- If using Oracle as a caching database, rebind the following packaged queries and point them to your caching database (for details on rebinding a packaged query, see the "Rebinding a Procedure" section under "Using Stored Procedures" in the CIS User's Guide documentation.)
    - `<IC_Folder>/Configuration/support/pq_Oracle_CreateTableAs`
    - `<IC_Folder>/Configuration/support/pq_Oracle_DropTable`
    - `<IC_Folder>/IncrementalCaching/common/support/pq_Oracle_ApplyIncrUpdates`
    - `<IC_Folder>/IncrementalCaching/common/support/pq_Oracle_TruncateTable`
- If not using Oracle as a caching database, please contact your account manager to engage the services of an AS engineer who may be able to update the solution to use your caching database type.

CONFIGURING INCREMENTAL CACHING

The following is a discussion on how to configure a view for incremental caching.

Prerequisites

There are some prerequisites that must be met before configuring incremental caching can begin:

- The view to be cached must have caching already defined. It must have a caching database configured and must also have a cache data table configured. Only Single Table caching is supported with this incremental caching solution. The cache data does not need to be initially loaded, however. (While the Composite Studio development environment makes setting up caching fairly straightforward, these internal configuration mechanisms have not yet been exposed to CIS's scripting language so setting up the caching configuration must be done manually.)

- It is strongly advised that the cache data table 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.

- 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.

- Note that while partitioning is optional, if deleted rows need to be included as part of the incremental update, then partitioning is NOT optional. As a workaround for this, set the partition period in months to something like 12,000 to incrementally cache all the data.

Create a configuration script

The `<IC_Folder>/examples` folder contains a sample configuration script called `orcl_conf_incr_orders` that can be used as an example for configuring a view for incremental caching. The easiest starting point is to copy this script to a different folder and edit the contents.

The actual configuration script that your newly copied configuration script will call, `<IC_Folder>/Configuration/configureIncrementalCache()`, accepts 9 arguments:

- `inResourcePath` – The full path to the view to be configured for incremental caching.

- `inPkColCSV` – A comma separated list of the names of the "primary keys" of the view at `inResourcePath`. These don't need to be actual primary keys, just the combination of columns that uniquely identify each row. If a column name has non alpha-numeric characters in it, it should be qualified with double quotes, i.e. "MY%COLUMN$NAME".

- `inLastUpdatedColCSV` – A comma separated list of the names of the "last updated" column(s) that indicate when a row was last updated.
- `inCacheTableCacheKeyCol` – The column name for the cache key in the cache data table (in the cache database.)
- `inPartitionPeriodMonths` – The number of months of data to cache using cache partitioning. Set to `NULL` if partitioning is not required.
- `inActivityDateCol` – The name of the column that indicates when the row's activity occurred. This is different than the "last updated" column. For example, in a view that shows equipment pressure tests, a sales order may have occurred on a particular date (activity date), but someone may have needed to update the data in the row some number of days after the order occurred (last updated date.) Set to `NULL` if partitioning is not required.
- `incrType` – The method to use to collect updated rows. The procedure `<IC_Folder>/Configuration/Constants` has three constants defined that can be used as values for this input:
    - `INCR_TYPE_NONE` – Useful for setting up partitioned caching where incremental updates are not required. CIS will always perform full cache refreshes in this case.
    - `INCR_TYPE_LAST_UPDATED` – CIS will determine what has changed based on the column(s) defined in the `inLastUpdatedColCSV` input.
    - `INCR_TYPE_DELTA` – CIS will use a "delta" table to determine what has changed.
- `incrConfList` – A VECTOR (single dimensional array) of options for each of the incremental update collection methods. The options for each method is listed below:
    - `INCR_TYPE_NONE`
        - N/A
    - `INCR_TYPE_LAST_UPDATED`
        - `DeletedRowTablePath` – The path to the table/view containing information about rows that were deleted from `inResourcePath`. Set to NULL or comment out if deleted row information is not required/available.
        - `DeletedRowTablePkCSV` – Similar to `inPkColCSV`, this is a comma separated list of the names of the "primary keys" of the deleted row

table. The column names do not need to be the same as the ones in `inPkColCSV`, but the column(s) they represent need to appear in the EXACT SAME ORDER. For instance, if `inResourcePath` has primary keys named `UWI` and `Range` and the corresponding primary keys for the deleted row table are `UniqueWellId` and `LandRange`, the two primary key lists should be in the exact same order. NOTE: If the deleted row table has other columns in its primary key (such as deleted date) LEAVE THEM OUT. Set to `NULL` or comment out if deleted row information is not required/available.

        - `DeletedDateCol` – The column in the deleted row table that indicates when the row was deleted. Similar to a "last updated" column described above. Set to `NULL` or comment out if deleted row information is not required/available.

        - `DeletedActivityDateCol` – The column that indicates when the deleted row's activity occurred (similar to `inActivityDateCol`.) Set to `NULL` or comment out if deleted row information is not required/available.

    - `INCR_TYPE_DELTA`

        - `DeltaTablePath` – The path to the table/view containing the changes to be applied. It requires that all the columns be named the same as the columns in `inResourcePath` above with the addition of a column that indicates the last operation performed (see `DeltaOperationCol` below.)

        - `DeltaOperationCol` – The name of the column containing the last operation performed on the row.

        - `DeltaInsertOperation` – The value used in `DeltaOperationCol` to indicate the record was inserted.

        - `DeltaUpdateOperation` – The value used in `DeltaOperationCol` to indicate the record was updated.

        - `DeltaDeleteOperation` – The value used in `DeltaOperationCol` to indicate the record was deleted.

- `applyType` – The method to use to apply the updated rows to the existing cache data. This parameter is no longer used since only a single application type is supported. It is maintained as an input for backwards compatibility. The procedure `<IC_Folder>/Configuration/Constants` has one constant defined that should be used as a value for this input:

- `APPLY_TYPE_UPDATE_IN_PLACE` – Applies updates to the existing cache data.

Your newly copied script will have variables declared for these values for ease of use. This is not required, however, as the required values can be plugged directly into a call to `configureIncrementalCache()`.

NOTE: The generated scripts qualify the names of columns, views, and tables with double quotes to prevent referencing errors so the letter case of all the names configured in the above settings must be EXACT.

Execute the configuration script

When the configuration script is executed, it will call `configureIncrementalCache()`. This, in turn, will perform a number of configuration checks to make sure everything is in place to configure the requested view for incremental caching. If any of these checks fail, execution will stop before any changes to CIS are made. If the checks pass, then the script will describe in the console window all the operations that it is performing and should complete the configuration successfully.

Deconfiguring incremental caching

The script `deconfigureIncrementalCache()` can be used to undo the incremental cache configuration and remove all the objects created during the incremental cache configuration steps. It takes a single input argument, which is the path to the original view that was incrementally cached.

INCREMENTAL CACHING IMPLEMENTATION

The following is a discussion of how incremental caching has been implemented.

Incremental Caching Methodology

At it's most basic, the incremental caching solution involves the following steps:

- 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.

Incremental Caching Folder and Common Resources

The following sections describe the sub-folders of `<IC_Folder>` and other resources:

CacheScripts

This folder contains the scripts each individual incrementally cached view uses to update its cached data. These are organized into sub-folders by view path.

Configuration

This folder contains the scripts used to configure or deconfigure incremental caching for a view.

Configuration/Constants()

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

Configuration/configureIncrementalCache()

This procedure takes a number of configuration inputs and creates the procedures and views necessary for incrementally caching a view or table. It will also perform whatever steps are needed to change the cache refresh from a full to an incremental refresh configuration in the repository.

Configuration/deconfigureIncrementalCache()

This procedure reverses the operations performed by the `configureIncrementalCache()` script for the same set of inputs.

Configuration/support

This folder contains a number of supporting scripts that assist with the configuration of incremental caching.

Common

This folder contains a number of stored procedures that perform several common functions that are utilized by the various incremental caching procedures.

Common/support

This folder contains a number of supporting scripts that assist with the execution of incremental caching.

Partitions

For those cached views that need to partition their data, the partitions folder contains the cached view's original query view, the cached partition view and the on-demand (or "live") view. These are also organized into sub-folders by view name.

Incremental Cache Scripts

Each view that is incrementally cached gets a set of scripts found in a folder structure named `<IC_Folder>/CacheScripts/<view path>`.

Definitions()

This script contains a number of public constants that are used by the other scripts in this folder. Various row types, partitioning configurations, and view locations are described here.

fullCacheRefresh()

This script performs the initial full cache refresh of the data so that subsequent refreshes can incrementally update it. It is the script that is plugged into the **Initialize the cache using** field in the resource's Cache tab.

incrementalRefresh()

This is the script that orchestrates the incremental refresh of a view. It is the script that is plugged into the **Refresh the cache using** field in the resource's Cache tab. It performs some basic checking such as verifying configurations then goes through a two-stage process to incrementally update the cache data:

- Collect the rows of data that are new or updated and the rows that need to be deleted (either because partitioning is configured, the partition date has been updated, and older data needs to be removed or because a deleted row table is configured and new deleted rows were found) into a staging table

- Apply the changes in a single transactional update (apply all the inserts, updates, and deletes and only when all are successful is a commit issued.)

collectIncrement()

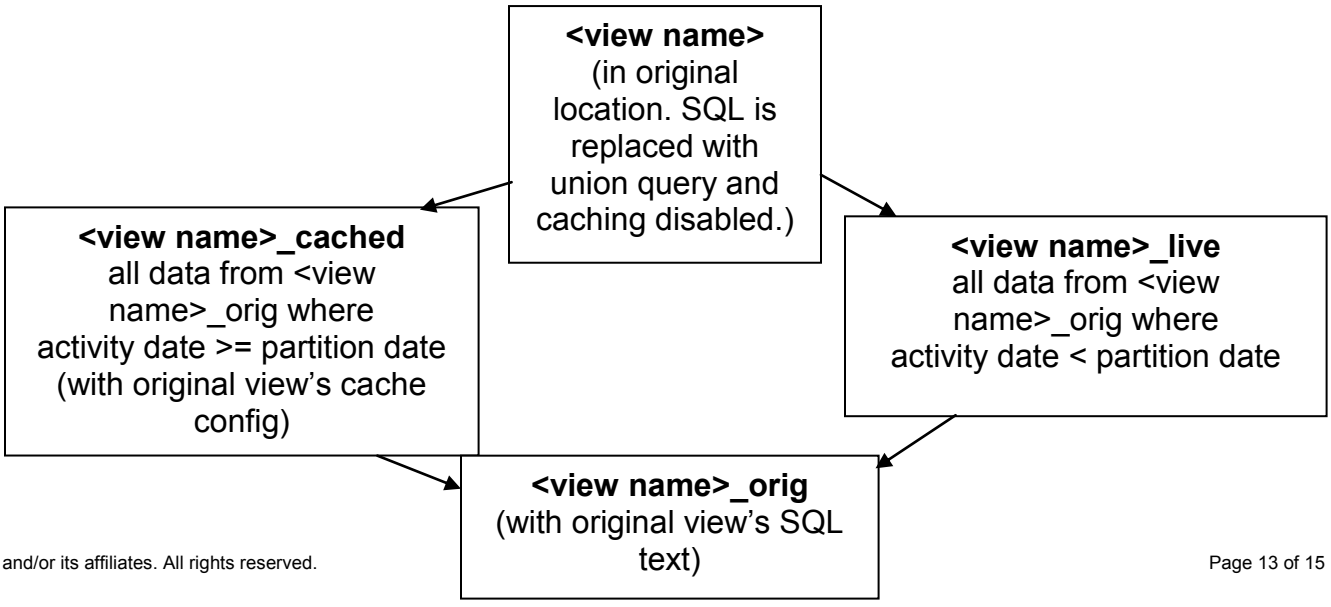
This script performs the first phase of the incremental update: gathering the changed rows. It gathers a set of new or updated rows (where the SOR last update date > `incrementalmaintenancelevel` from the cache database's `cache_status` table) and, if partitioning and/or a delete table is configured, it also gathers the rows to be deleted. These update rows are stored in a staging table in the cache database.

applyIncrement()

This script performs the second phase of the incremental update: applying the changes to the cache data. It uses a single transaction to apply all the inserts, updates, and deletes such that if any update fails, all can be rolled back. After all the updates are complete a commit is issued. (NOTE: This is accomplished using Oracle's `MERGE` command. If the cache database is something other than Oracle, please contact your AS consultant to see how this script can be modified to accommodate it.)

Partitioning Views

If a view is configured for partitioned incremental caching, a set of views will be created for it in `<IC_Folder>/Partitions/<view path>`. The SQL text of the original view will be copied to a view called `<view name>_orig` in this folder. The caching configuration will be moved to a view called `<view_name>_cached`, also in the same folder, which will only query the part of the original query that needs to be cached (data where the activity date is greater than or equal to the partition date.) A third view is created in the partition folder that contains the live portion of the partition (data where the activity date is less than the partition date.) Finally, the original view's SQL text will be replaced with a SQL query that unions the two halves of the partition together and its caching configuration will be disabled (but not deconfigured.) The new configuration will look like this:



<view_name>_cached

This view contains the SQL code for obtaining the cached portion of the partition. It inherits the caching configuration of the original view (the data for this cached view is placed in the data table configured in the original view. The original view's caching definition is disabled after configuration.) The partition date is automatically updated by its individual `IncrementalRefresh()` script.

<view_name>_live

This view contains the code for obtaining the uncached (or "live") portion of the partition. The partition date is automatically updated by its individual `IncrementalRefresh()` script.

<view_name>_orig

This view contains the SQL code from the original view to be cached. The two views above select from it to obtain their respective portions of the partitioned data (using a where clause that specifies a partition date.) The original view's code is replaced with a `UNION`'ed select of the two views above.

Performance Note

Because the partition date is hard coded in the union view and again in the partition views, CIS is smart enough to prune one side of the union or the other if the where clause of a user's query permits it. For example, if a user issues a query with a where clause condition that indicates only cached data is needed, CIS will prune the union and the fetch to the live data from its execution plan and only issue a fetch to the cached data.

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **www.cisco.com/go/offices.**

Printed in USA CXX-XXXXXX-XX 10/11