Composite Information Server
Technical Note

Caching Views and Procedures in CIS

COMPOSITE SOFTWARE

Contents

Overview

This document describes the caching features of Composite Server 4.5 in detail from a technical perspective.

Resources for Caching

The following tabular resources can be cached:

- SQL View
- Physical Tables that have been introspected

The following procedural resources can be cached:

- Java Procedure
- Packaged Query Procedure
- Parameterized SQL Procedure
- Physical Stored Procedures that are Introspected
- SQL Script Procedure
- Transformation Procedures – Basic, Streaming, XSLT, and XQuery
- Web Service Operation

The following resources cannot be cached without being wrapped in one of the resources itemized above:

- Procedures with no outputs – There is no data to cache in this case.
- XML Files that have been introspected
- System Tables
- Non-data sources such as Folders, Definition Sets, and so on.

Configuring and Enabling a Cache

When a View or Procedure is selected for being cached, the cache needs to be configured and enabled. This section describes how to perform these tasks via the Studio. For details on how to do these tasks programmatically, see the section “Using the API.”

To configure caching for one of the supported resources, open the resource in Studio and select the **Caching** tab in the resource editor. You will see a button labeled **Create Cache**. Pressing this button will configure the resource for caching (as of the next save operation), but leave the caching in a disabled state.

The enabled/disabled state can be toggled using the **Enable** checkbox on the **Caching** panel. When configured and enabled, the cache will be activated. When disabled, the cache

configuration and any data in the cache will be preserved but any use of the resource will execute as normal instead of using the cached data.

To de-configure caching, press the **Destroy Cache** button on the upper right corner of the **Caching** panel. This action will erase the configuration and clear the cached data.

There is an important distinction between a cache being "**configured/de-configured**" and "**enabled/disabled**."

- Any cache that is configured will be tracked on the **Cached Resources** panel in the Manager and have data that is cached, preserved, and tracked. A cache that is "de-configured" is not tracked any longer and the data in the cache is cleared.
- This is distinct from being "enabled/disabled," which is a simple switch on whether or not to use the cache right now.

Table and Procedure Caching

This section discusses how data from tables, views, and procedures are cached.

View Caching

The caching of a View is a straightforward copy of all data returned from executing the View.

Procedure Caching

The caching mechanism for a Procedure varies depending upon whether the Procedure has input and/or output parameters.

The caching of a Procedure uses one storage table for each output cursor and an additional storage table for any scalar outputs. For example, a procedure with two `INTEGER` outputs and two `CURSOR` outputs would use three tables—one for the pair of scalars, one for the first cursor, and one for the second cursor.

If the Procedure has input parameters, the cached results are tracked separately for each unique set of input values. For example, a procedure with an `INTEGER` input parameter would have separate results cached for inputs of 1, 3, and 5. A procedure with two `INTEGER` input parameters would have separate results cached for inputs (1,1), (1,2), and (2,5). Each unique set of such input parameter values is called a ***variant***.

In order to track variants, the input parameter values are converted to a string to compare for uniqueness. If this string exceeds the size supported by the system, the procedure call will not be cached. Instead, it will be directly executed as if caching was disabled. The limit for this value in the current release is 255 characters.

The maximum number of variants to cache at one time is configured on the **Cache** tab. The default value is 32. When the system exceeds this number of variants, the loading of a new variant will cause the least recently used variant in the cache to be discarded.

Behavior in Composite Studio

Executing a View in Composite Studio will read from the cache if the View is saved and caching is enabled. When showing the execution plan, the cache data table will be shown in the plan. "Cache Data Tables" are described in the section "Understanding the Tables."

If the View is edited and not saved, the View will be run anonymously and the caching behaviors of the View will be ignored.

Executing a Procedure in the Studio will read from the cache for all Procedures except SQL Scripts. Studio always runs SQL Script procedures in an anonymous way that bypasses the caching behaviors. To test the caching behavior of a SQL Script, it must be saved and must be called from another Procedure or View.

Automatic and Data Source Caching

This section discusses how cached data are stored.

Automatic Caching

When caching is first configured, it defaults to using the **Automatic** mode. In the **Automatic** mode, the storage data source and storage tables for the cached data are chosen automatically for you.

The storage tables are automatically created, dropped, and maintained as the signature or existence of your Views and Procedures change.

Automatic caching makes use of a file-based data source located at: `/lib/sources/cacheDataSource.` The actual data is stored in files under `{InstallDir}/tmp/cache`. For details on file-based data sources, see the section "**The File-Cache Data Source**."

The cache storage table names are automatically generated to be unique using names like `view12345` and `proc54321p2`. The table names chosen by the system are displayed on the **Caching** panel for the resource. Also, the annotation on each of the generated tables includes information on which View or Procedure is being cached to that table.

These tables are accessible directly, but it is not generally recommended that these tables be used directly. Having them exposed makes it possible, however, to view the contents of these tables when developing.

When the server is running in a cluster and the cache is in **Automatic** mode, each server will keep a separate copy of cached data on its local file system. No sharing of data is performed.

Data Source Caching

As an alternative to the **Automatic** mode of caching, cache data can be stored in a relational database. To use this kind of storage, select the **User Specified** option in the **Caching** panel and either type the path of the data source or use the **Browse** button to select one.

When the server is running in a cluster and the cache is in **Data Source** mode, all servers in the cluster will access the same data source for cache information. In addition, the servers will cooperate to ensure that a minimum number of refreshes occur. For example, if two servers both

need to refresh the data in a cache, only one of them will perform the refresh, then both servers will use the updated data.

Status and Tracking Tables

Before a data source can be used for caching, it must have two types of tables set on it. These are the Status tables and Tracking tables. Only one pair of tables is required for the data source to support any number of Views and Procedures being cached to the data source.

If the table you need does not already exist, you can create them. Open the data source in Studio and select the **Info** tab. The **Caching** box in the **Info** panel will show two input fields, where the table paths can be entered, and a **Browse** button for each input field. Use the **Browse** button and select a container, which can be the data source node itself or a folder in the data source, type in a table name, and press the **Create** button. This command will display the DDL to create the table. This DDL can be copied out and run in a database-specific tool or the **Execute** button may be pressed.

To configure existing storage tables, open the data source in Studio and select the Info tab. In the Caching box, type the table paths, or use the Browse button to choose a table.

The colors red and black indicate valid choices and tables can be created or re-created as supported by the data source.

The paths to the tables are **case sensitive**. Be aware that when creating a table in DDL the database may not preserve the case of your entry and so when the data source and text entry are updated after running the DDL, the case of the created table name may not match what was originally entered.

Storage Tables

Once a data source is chosen for storing cached-data, each of the required storage tables needs to be specified.

In the **Caching** panel for a resource, one field will be present for each result cursor. For a Table or View one table is needed, and this table name is displayed in the field labeled result. Procedures may need more than one output table.

Either type the path of the table you want to use, or use the **Browse** button to select one.

When using the **Browse** functionality, tables are highlighted in red if they are not compatible with the data you need to cache. Selecting one will display a note detailing why the table is not compatible. Tables shown in black are compatible and can be used successfully.

If you select an existing table that is not compatible, a button to **Recreate** the table will also be shown. Selecting this button will display the database specific DDL that needs to be executed to recreate the table with a compatible schema for storage. This DDL can be copied out and run in a database specific tool, or the **Execute** button can be pressed to attempt and execute the DDL using the data source's configured login information. This attempt may or may not succeed depending on the permissions the data source's login has within the database. Be aware that "recreate" is accomplished by `DROP` and then `CREATE` actions so any data in the existing table is lost.

If the table you need does not already exist, you can create it as described in the preceding section.

In either case, the execution of DDL will be followed automatically by the equivalent of an **Add/Remove Resources** operation performed from the Studio. The newly created or re-created table will be updated, but other resources in the same folder in the data source may also be updated.

The paths to the data source and table are **case sensitive**. Be aware that when creating a table in DDL the database may not preserve the case of your entry and so when the data source and text entry are updated after running the DDL, the case of the created table name may not match what was originally entered.

Not all data sources that can be used for cache storage might support the generation or execution of DDL statements. In some cases, DDL may be presented but use of an external tool may be required. In other cases, even the display of DDL may not be available. Currently supported data sources are:

- DB2 v7 / v8
- Microsoft SQL Server 2000 / 2005
- MySQL 4.0 / 4.1
- Oracle 8i / 9i / 10g
- Sybase 11 / 12
- Teradata
- File-Cache (see information later in this document for details)

Privileges

In order for someone to use the View or Procedure that is being cached to a data source, that user must have proper privileges granted to access both the Status Table and data tables.

Users that need to read from the cache should be granted the `SELECT` privilege on these tables and `READ` privilege on folders above these tables.

The user that owns the resource being cached is the user identity that cache refresh and clear operations will be run with. That user needs the `READ`, `SELECT`, `INSERT`, `UPDATE`, and `DELETE` privileges on the Status Table, Tracking Table, and the data tables.

Privileges are managed automatically when using the **Automatic** mode of caching.

The “File-Cache” Data Source

The **Automatic** mode of caching makes use of a pre-created instance of the File-Cache data source type that cannot be re-configured.

Additional instances of this data source can be created using the **New->Data Source** menu option just like for any other data source. These additional instances can have the storage directory set up to store cache data files anywhere on the file system. These additional instances will not be managed the way the **Automatic** caches are. All table management is handled exactly like it is for any other data source.

The File-Cache data source uses a directory for each Table in it, and a file in that directory for storing the data. The data files are binary encoded for best performance with the server.

The directory tree of files is specifically designed to be free of server-instance specific data. This means it is possible to copy this directory to a new machine, re-introspect the data source on that machine, and have the data appear. This allows for the design of portable cached data sets.

The use of the File-Cache data source or other file-based data sources is not recommended when in a cluster. The caching system assumes that data source caches are available to all servers and this is not true for file caches. The use of network mounted files is also not recommended because the file data sources do not support file locking.

Managing Cache Data

Terminology

The term "**refreshing**" or "**loading**" a cache means to read all the data from the source table or procedure and create a new set of data in the storage table(s). This data set replaces the previous data set if the refresh succeeds. The term "refresh" is not meant to imply any sort of incremental updating behavior. Incremental updates are discussed in the section "Incremental Caching".

The term "**clearing**" a cache means to remove all data from the storage table(s) so there is no longer any data in the cache.

Scheduled Refresh of Cached Data

The **Caching** panel in the Studio offers the option to schedule a refresh of the data. This schedule can be set to any number of seconds, minutes, hours, days, or weeks. (You can also refresh the cached-data programmatically. See the section "Using the API.")

Each time a refresh completes, it starts a timer and performs the refresh when the timer completes. This timer persists across server restarts. If the server is down when the timer should have completed, the refresh will be performed shortly after the server starts.

A start time can be specified if the refreshing should not begin before a given time in the future. This defaults to the current date and time.

The start time can be used to place the recurring refresh at a given time of the day. For example, if the start time is 2am tonight and the interval is 1 day, it will refresh every day at around 2am.

If the refresh interval is shorter than the duration of a refresh, any scheduled refresh attempts while a refresh is still running will be ignored.

Any time the start time or interval is altered using Studio or any API, the schedule is reset. If the start time is in the past, this can cause an immediate refresh. For example, changing the interval from 4 hours to 2 hours will cause an immediate refresh if the start time is in the past. Disabling and then re-enabling the cache will also reset the schedule with the same behavior.

Expiration of Cached-Data

Any resource can have an expiration period applied to it, for example, "1 hour". After this specified period, the data is considered "expired" and will not be used. Any new query attempting to access the data will discard the old data and trigger a refresh.

The expiration period applies from the end of a successful refresh.

For tables, the expiration period applies to the whole table. For procedures, each input variant's data has its expiration tracked separately.

In addition to having the data discarded if an attempt is made to access it, a background task will look for expired data and proactively delete it. This background task will check every 60 seconds or the shortest expiration period assigned to any enabled cache, whichever is larger. For example, if the shortest period is "1 hour", then it will run once an hour. If the shortest period is "10 seconds", then it will run once every 60 seconds. If no enabled cache has an expiration period, then this task will not run.

Rules for Clearing Cache

The Studio offers three options for clearing a cache.

when user clears it manually – This option means that the cache clears only when cleared explicitly through Studio or an API, or when the cache expires.

when refresh begins – This option automatically clears the cache before starting a refresh. The effect of this is that any client attempting to read from the cached data will not see the previously cached data and will wait for the new data.

when refresh fails – This option automatically clears the cache if a refresh fails. The effect of this option is to allow access to previously cached data during a refresh, but to either provide updated data or to end up with a cleared cache when the refresh completes with success or failure respectively.

Triggers

Triggers that Operate on Caches

Trigger resources can be used to cause caches to refresh on a scheduled basis. You can use triggers to create richer refresh schedules than the built-in scheduling feature on the **Caching** panel for resources.

Triggers can also be used to cause refreshes to occur in other circumstances. For example, a user-defined event may be used to cause a refresh.

Triggers that Detect Cache Changes

There are two System Events related to caching that Triggers can be made to listen for: `cacheRefreshSuccess` and `cacheRefreshFail`. A Trigger can run logic as a result of either of these events.

Refer to the product documentation for additional details on Triggers (*User's Guide*, chapter *Triggers*).

Transaction Result Caching

There is a related feature in the product called “transactional result caching” that is available for Procedures. This feature is enabled by checking a box on the “Info” panel of any procedure.

When enabled, the first time the procedure is run during any transaction will capture the results in memory (using a disk-backed temporary store if enough memory is not available), and additional calls to the procedure with the same input parameters will return the same data.

This feature is not the same type of caching that is used throughout this document. No refresh, clear, status, or tracking features are available.

The Transaction Result Caching feature may be useful as an alternative to full procedure result caching if the primary need is for transaction isolation instead of storing results for repeated access between transactions.

Importing From Release 3.7 or Earlier

This section describes how to transfer cached data from release 3.7 or earlier of Composite Server to the current version of Composite Server.

“File” Caching to “Automatic” Caching

When importing `CAR` files from the 3.7 release or earlier, all caching to File Caches will be converted into **Automatic** caching.

Cache refresh schedules will be preserved. Expiration will be set to “**never**” and the Clear Rule to “**when user clears it manually**.”

“Database” Caching to “Data Source” Caching

When importing `CAR` Files from the 3.7 release or earlier, caching to Databases will be converted into Data Source caching using the same storage table.

Cache refresh schedules will be preserved. Expiration will be set to “**never**” and the Clear Rule to “**when user clears it manually**”.

After import, each Database (now Data Source) cache will require manual definition of the Status and Tracking tables for the data source. See the section “**Data Source Status and Tracking Tables**” section for details on these settings.

In addition, each storage table must have a column named “**cachekey**” of type `INTEGER` added as the first column of the table. This can be done using the database administration tools, or Studio can be used to re-create the table as described in the section “**Data Source Caching”**.

Cache Lifecycle

This section describes the details of the cache lifecycle. It is intended for those that want to understand the timing and states in close detail.

Throughout this description, the term “cache” should be treated as applying to the whole cache for a View or to the cache data for a single input variant for a Procedure unless otherwise specified.

Enabling and Disabling

When a cache is disabled, all cache settings are ignored. The View or Procedure is used as if caching did not exist.

When a cache is enabled, use of a View will instead make use of the cache storage table to find the data, and use of a Procedure will make use of one or more cache storage tables to find the data.

Toggling between the enabled and disabled states will not cause refreshing or clearing of the data. It will also not reset the expiration date for the data.

Loading and Refreshing

A cache can be in a not-leaded state or loaded state.

Not Loaded State

Initially, a cache starts in a “not loaded” state.

Forced Loading of a Cache

A cache can be forced to load in four ways:

- An explicit Refresh Now action is requested using Studio
- A scheduled refresh
- Calling the system built-in procedure `/lib/resource/RefreshResourceCache()`
- Executing the published `RefreshResourceCache` Web services operation

All of the above are non-blocking actions that cause a background task to begin a refresh action if a refresh is not already in progress. If a refresh is already in progress, the action will simply return immediately.

For procedures, a forced refresh will refresh all variants that are currently cached. If no variants are cached, then no action is taken.

On Demand Loading of a Cache

In addition, a refresh may be caused indirectly if any attempt to read from a cache that is not loaded.

When this happens, it will start a background task to refresh the cache if a refresh task is not already running, then block until the refresh completes. If the refresh completes successfully, the query will unblock and use the data. If the refresh fails, the blocked query will fail providing the refresh failure’s information as the reason for the failure.

For procedures, only the one variant being accessed will be refreshed. This is different from a forced refresh, which refreshes all variants.

Reading from a Loaded Cache

Any attempt to read from a cache that is already loaded will return the current data, even if a refresh is currently in progress.

The Refresh Task

The background task that performs the refresh always runs as the user that owns the View or Procedure that is being cached. This is true regardless of which user requests or causes the refresh task to start.

Interaction with "Pass Through" Data Sources

The cache system does not support "pass through" data source configurations. The refresh action is always performed as a single user and is always performed without that user's credentials. Because of this, all data sources underlying the View or Procedure being cached must have specific credentials coded on them or the refresh action will fail.

If "pass through" is enabled on a data source that is under a cached view, the cache will no longer be able to refresh and whatever data is in the cache will be used unless the "clear on refresh failure" clear rule is also used to force a clear.

Clearing the Cache

This section discusses the ways to mark a cache for clearing.

A cache can be manually cleared in three ways:

- An explicit Clear Now action is requested using Studio.
- Calling the system built-in procedure `/lib/resource/ClearResourceCache()`
- Executing the published `ClearResourceCache` Web services operation

Cached rows are marked for clearing as a background task. A new request for that data will detect data marked as stale and fetch fresh data from the source(s). A forced clear marks all cached variants as stale data and readies them for clearing. It does not cancel any in-progress refresh actions. Those refresh actions will continue to execute and when successful they will load the cache as expected.

Other Dynamic Causes for a Cache to Clear

A cache may also be cleared in the following cases:

- An attempt to read from the cache while the cache contains expired data.

  The test for expired data is performed prior to any read and causes any expired data to be immediately cleared; then since the cache is not loaded it will cause a refresh as described in the **Loading and Refreshing** section.

- An attempt to refresh the cache while it contains expired data.

  The test for expired data is performed prior to starting the refresh and the clear is performed before returning from the refresh action.

- An attempt to refresh a cache using the "**when refresh begins**" clear option is selected.

  When this clear rule is used, the cache is cleared prior to returning from the refresh action.

- An attempt is made to load a procedure variant that is not currently loaded and the maximum number of variants is already loaded

  This causes the least recently used variant to be cleared immediately, then for the processing to continue as described in the **Loading and Refreshing** section.

- A cache refresh fails and the "**when refresh fails**" clear option is selected.

  When a refresh fails, instead of making new data available, the refresh task will clear out existing data.

In all the above situations, when it says that a cache is cleared immediately, it means that the data is made unavailable immediately by altering the Status Table and internal tracking information of the server, but a background task is used to actually clear the data from the data tables.

The Data Clear Task

The background task that performs the data clear always runs as the user that owns the View or Procedure that is being cached. This is true regardless of which user requests or causes the clear task to start.

Cache Status

The cache status is reported in Studio on the **Caching** panel for the resource and on the **Cached Resources** console in the Manager.

- A cache begins in the "**NOT LOADED**" state. This means no data is loaded.
- A cache that has been loaded successfully is in the "**OK**" state.
- A cache that fails to load successfully is in either the "**DOWN**" or "**STALE**" state. The **DOWN** state indicates that the cache is not loaded and the most recent refresh failed. The **STALE** state indicates that the cache is loaded with valid data, but that the most recent refresh failed. When a cache is **STALE**, reads against the cache can succeed.
- A cache that cannot operate due to a configuration error is in the "**CONFIG ERROR**" state.
- A cache that is disabled is in the "**DISABLED**" state.

Cache Events

The server event log is set to log all cache events by default. These events can be seen on the **Events** console in the Manager. They can also be found in the events log files and in the SYS_EVENTS system table.

The reported events are:

- cacheEnable
- cacheDisable
- cacheClear
- cacheRefreshStart
- cacheRefreshEnd
- cacheRefreshFail

The logging levels for these events can be set in the **Configuration** panel as with all other supported events. Refer to the product documentation for additional details.

Configuration Changes

Changing a cache's configuration to point to a different storage data source or changing a storage data source to use a different Status table will cause the server to discard all in-memory information about the data being cached and for this information to be read from the Status table again. This re-read is performed as a background task, so there is a brief period prior to this re-read when accessing cache data may encounter errors.

Changing a cache's configuration to use different data storage tables will not cause any in-memory information to be updated. The cache will use the new tables assuming data is stored under the proper cache key. Performing an explicit Clear or Refresh action following this change is recommended.

Renaming a cached resource or any of its parent folders will cause the Status table to be updated with the new name and path of the resource. This update is performed as a background task, so there is a brief period prior to this update when accessing cache data may encounter errors.

Changing a cached View or a Procedure's signature for a cache that is using **Automatic** caching will cause the old data table to be dropped and recreated. An explicit clear action is performed automatically. This re-create and clear is performed as a background task, so there is a brief period prior to this task completing when accessing cache data may encounter errors.

Changing a cached View or a Procedure's signature for a cache that is using **Data Source** caching will not change the schema of the data storage tables. As a result, the View or Procedure is likely to have a configuration error. The error state will remain until the tables are either re-created using Studio or are altered using external database tools and then re-introspected using Studio.

Using Studio to execute DDL to re-create a table will delete any data in the cache table. Performing an explicit Clear or Refresh action following this change is recommended.

Using the API

The caching system can be operated on programmatically.

Procedures

The following procedures are available for use from SQL Scripts, Java Procedures, or can be published for use from JDBC clients:

- `/lib/resource/ClearResourceCache(path,type)`
- `/lib/resource/RefreshResourceCache(path,type)`
- `/lib/resource/UpdateResourceCacheEnabled(path,type,isEnabled)`
- `/lib/resource/CreateResourceCacheKey(path,type,cacheKeyOutput)`
- `/lib/resource/LoadResourceCacheStatus(path, type)`

Web Services

The following Web services are available for use from clients outside the server. They are under the `Composite Data Services/Web Services/system/admin` service in Studio:

- `resource/resourcePort/clearResourceCache`
- `resource/resourcePort/refreshResourceCache`
- `resource/resourcePort/getResourceCacheConfig`
- resource/resourcePort/setResourceCacheConfig

Understanding the Storage Tables

This section describes the tables for storing cached data.

The Data Source Status Table

The data source Status table is used to track which Views and Procedures currently have cached data stored, when they were last refreshed, and so on.

It is not legal in the current release for a single status table to be shared by two different Composite Servers. Each server must have its own table.

Reference Columns

The following columns are used to identify a particular set of cached data.

- *clusterid* - This column is NULL if the server it not a member of a cluster. It has the cluster's ID if the server is a member of a cluster.
- *serverid* – This column identifies which server is responsible for the entry in the table. A server's ID is constructed by appending its hostname, port, and a generated number. For example: myhost.mycompany.com-9400-12345
- *resourceid* – This column identifies which resource in the server is being cached. This is the path of the resource. For example, /shared/myfolder/myview.
- *parameters* - This column identifies which set of input parameters is being cached. This column is NULL for Views, and is always non-NULL for procedures. The value is a comma-separated list of the input parameters in a string form identical to the form that would be used to call the procedure from SQL Script. For example, if a procedure that has a VARCHAR and INTEGER input could have a "parameters" value of "'a string value',42".

Status Columns

The following columns are used to identify the status of a particular set of cached data:

- *status* – This column has the value 'A' if the row describes an Active (loaded) set of cache data. The value 'I' indicates an In Progress (refreshing) set of cache data and the value 'P' indicates a Probe by one server in a cluster against refresh attempts by other servers in the cluster. The value 'F' indicates a Failed set of cache data. The value 'C' indicates a Cleared set of cache data. The value 'K' is special and is described below.
- *cachekey* – This column holds the unique key that identifies a set of cache data. For an Active status row, this is the *cachekey* column value in the data table(s) for the active data. For an In Progress status row, this is the *cachekey* in use for refreshing.
- *starttime* – This column holds the time the data load (refresh) started.
- *finishtime* – This column is NULL while In Progress and is updated with the finish time when a load (refresh) ends in either success or failure.

- *cleartime* – This column is NULL at all times in the current release.
- *bytes* – This column is NULL while In Progress or after failure to load. When in an Active (loaded) state, this holds the approximate number of bytes in the cache.
- *message* – This column is NULL except for Failed rows, in which case this column holds the failure reason message.

The Key Row

There is one special row in the table with the status of 'K'. This row does not describe any cached data. Instead it is used to hold the next available *cachekey* value. The server updates this row as it uses *cachekey* values. Currently it increments it by 1,000 each time.

The Data Source Tracking Table

The data source Tracking table is used to track the Views and Procedures that are currently using the data source for caching, and what tables in the data source are in use. This is purely informational and is not used by the server at this time.

It is not legal in the current release for a single tracking table to be shared by two different Composite Servers. Each server must have its own table.

Reference Columns

The following columns are used to identify a particular set of cached data.

- *clusterid* - This column is NULL if the server it not a member of a cluster. It has the cluster's ID if the server is a member of a cluster.
- *serverid* – This column identifies which server is responsible for the entry in the table. A server's ID is constructed by appending its hostname, port, and a generated number. For example: `myhost.mycompany.com-9400-12345`
- *resourceid* – This column identifies which resource in the server is being cached. This is the path of the resource. For example: `/shared/myfolder/myview`

Tracking Columns

The following columns are used to identify the tables in use for of a particular set of cached data.

If a resource (such as a Procedure) requires more than one table for caching, one row will be entered in the Tracking table for each table it uses.

- *catalog* - This column identifies the catalog (if any) of a table in use.
- *schema* – This column identifies the schema (if any) of a table in use
- *table* – This column identifies the table name (if any) of a table in use
- *createtime* – This column is always NULL in the current release.

For data sources that do not support catalogs, the *catalog* column is always NULL. The same applies to the *schema* column.

Some database products refer to the schema concept as a "database" or by some other name used to segment tables into separate namespaces. For these database products, this name will be found in the *schema* column.

Cache Data Tables

Each cache data storage table contains one additional column in addition to the columns of data that are being stored. This *cachekey* column contains an integer that identifies which rows belong to which variant of the data. The data source Status table identifies which *cachekey* value is associated with current data or with specific parameter input variants for procedures.

Transaction Isolation

The *cachekey* value in the Status table 'A' row is known as the "active cache key".

If a transaction is making use of a cache with a given *cachekey* value and a refresh occurs, the newly loaded data will get a new *cachekey* value and will be made active when the refresh completes. The rows associated with the key values still in use by one or more transactions will not be cleared until they are no longer in use. This allows existing transactions to complete with consistent data for their entire run.

A background task is used to clear cache data that is no longer in use.

Caching in a Cluster

This section discusses how caching works in a cluster.

Automatic versus Data Source Caching

When the server is running in a cluster and the cache is in **Automatic** mode, each server will keep a separate copy of cached data on its local file system. No sharing of data is performed.

When the server is running in a cluster and the cache is in **Data Source** mode, all servers in the cluster will access the same data source for cache information. In addition, the servers will cooperate to ensure that a minimum number of refreshes occur. For example, if two servers both need to refresh the data in a cache, only one of them will perform the refresh, then both servers will use the updated data.

The use of the File-Cache data source or other file-based data sources is not recommended when in a cluster. The caching system assumes that data source caches are available to all servers and this is not true for file caches. The use of network mounted files is also not recommended because the file data sources do not support file locking.

Cache Lifecycle in a Cluster

When in a cluster, the attempt to refresh a cache uses the Status table to determine if any other refreshes are in progress. If no others are in progress, the refresh will start and all other servers will be notified to re-read the Status table. If a refresh is already in progress, the server will contact the server that is currently refreshing and wait for that refresh to complete.

When in a cluster, the attempt to clear a cache will mark the cache data for clearing in the Status table. The background task to actually delete the data will contact other cluster members to determine what cache keys are no longer in use in order to safely remove only the rows that no server in the cluster is accessing. After the clear task completes, all other servers will be notified to re-read the Status table.

Scheduling Refreshes in a Cluster

A scheduled refresh in a cluster relies on the Trigger system cluster feature to ensure that triggers are processed only the appropriate number of times. Automatic mode cache schedules are configured to fire separately on each server. Data source mode cache schedules are configured to fire only once per cluster. Creation of Trigger resources, instead of using the schedule options provided on the Cache panel, can be used to control this behavior if a different behavior is desired.

Refer to the cluster documentation for additional details on the interaction of Triggers within a cluster.

Best Practices and Use Cases

This section discusses the best practices and use cases for using the caching feature.

File versus Database Caching

The main difference between file and database caching is that databases can perform result set filtering (`WHERE` clauses). This makes the file caching useful for small result sets, or for cases where the entire result set will be read all the time. Because databases caching allows filter conditions (`WHERE` clauses) to be executed in the data source, larger result sets that are frequently filtered can perform much better than with file caching.

Cache Indexing

The Composite Server does not index cache tables automatically. It is up the user to specify indexes according to the data result sets and usage patterns so that cache performance may be improved.

Note: Adding an index does add overhead to cache refreshes as insertions may take slightly longer depending on the database used, but the performance benefits of faster result set retrieval from an indexed cache may make that overhead worthwhile. Performance testing is recommended to ensure that goals are reached in

Indexing specific columns in the result set of a cached view will yield performance benefits when those columns are commonly used for filtering and viewing some subset of the data from the cache.

The first column of any index should be the cachekey column. The cachekey column is the only column automatically generated by Composite. Indexing the cachekey column will be useful in cases where many procedure cache variants are present.

Avoiding Stale Data

Cache data can become stale (out of date) when the data in the original system changes.

Scheduled Refreshing

One means of avoiding stale data is to schedule a refresh action to occur on a regular basis. Scheduled refreshes give control over when the cache data is loaded. It also results in the best performance for clients reading from the cache, because the refreshes occur as background tasks.

The down side of a scheduled refresh is that the cache is loaded regularly even if no clients are reading cache data.

Expiration Based Refreshing

Another means to avoid stale data is to give the data an expiration period. Expiration ensures that data is never more out of date than this period.

The down side of expiration is that cache refreshes may occur at any time and that the refresh blocks the client that detects that data is not loaded until the refresh completes. This can cause uneven response times for clients.

Avoiding Unnecessary Refreshing

If there is a way to find out if any data has changed in the original data source, a Procedure can be scheduled to run on a regular basis that tests if data has changed. If data has changed, it can call `RefreshResourceCache()` to force a refresh. If not, it can return without doing so. This has the advantages of a scheduled refresh, but avoids unnecessary work if no data has changed.

Caching to Avoid Load

A common use of caching is to avoid putting undue load on a corporate data system. By copying data into a cache, the cache absorbs the load. Both scheduled refreshes and expiration can be used to control how often the data is read from the original source.

Using Cache Windows

When caching to avoid load, it is sometimes desirable to only operate the cache during specific hours. This can be accomplished by enabling and disabling the cache on a schedule.

To do this, create a Trigger resource that calls the `UpdateResourceCacheEnabled()` procedure on a schedule to disable the cache at the appropriate times. Then create a second Trigger resource that enables the cache at the appropriate times. The pair of triggers will work together to enable and disable the cache over time.

Caching for Performance

Introducing a cache will not always improve performance. In some cases, the overhead of reading from the cache may be larger than the overhead of accessing the original data source.

When performance is the goal of using a cache, testing should be performed to ensure that this goal is reached. The proper choice of file versus database caching and use of indexes in a database (as described earlier) may be required to achieve this goal.

Handling Failures

A cache refresh may fail due to an outage or change to the original data source. The following are some options for handling such a failure.

Client Errors or Stale Data

The default behavior after a refresh failure is to return an error to the client if no data is loaded, or to return the currently loaded (but stale) data to the client.

To make the client always receive errors following a refresh failure, the cache Clear Rule can be set to “when refresh fails”. This will cause the cache to be cleared and to report errors on all read attempts following the failure.

To make the client tolerate some amount of stale data, both expiration and scheduled refreshing can be combined. For example, if the cache is set to refresh every 4 hours and the expiration

period is set to 24 hours, then the data will be allowed to go stale for up to 6 failed refreshes before it gets cleared and starts reporting errors.

Responding to Failures

A Trigger resource can be created that subscribes to cache refresh failures and sends an e-mail or executes a Procedure with appropriate error handling logic. This can be used to alert someone that there is a problem or to possibly perform a corrective task.

Incremental Caching

For some situations with very large data sets, it may be impractical to completely refresh the cache data each time. It may be desirable to instead only update the actual cache data that changed.

Incremental caching is not a built-in feature of the product, but is rather something that can be accomplished through the use of the API and additional programmed logic. The following sections describe some options, but there are other ways to accomplish incremental caching.

Determining What Changed

Incremental caching requires that there be some way to query the original data source as to what rows were inserted, updated, and deleted. It also requires that the rows in the data have a primary key that can uniquely identify each row.

The means of determining what rows were inserted, updated, and deleted will vary depending on the source. Some tables include a timestamp column for when the row of data was changed. Other tables may use sequence numbering that can be tracked to determine changes. In more extreme cases, database triggers may be used to detect changes and copy relevant data to a staging table.

In any case, the determination of what changed is not a built-in feature of the Composite Server.

Option #1: Transactional Database

For this option, a Procedure is written in Java or SQL Script and scheduled to run on a regular basis. This procedure performs some logic to identify what rows have changed.

Then, the server ID is acquired using “GetProperty(‘SERVER_ID’)”.

Then, the data source Status Table is queried to find the currently active *cachekey*. This is in the row with the appropriate server ID, resource path, and *status* column with value ‘A’.

Then, the appropriate INSERT, UPDATE, and DELETE operations can be performed on the data table using the *cachekey* as a filter to avoid updating any other keys.

This approach to incremental refresh only works if the data source offers transaction isolation. It relies on having none of the data changes becoming visible to other already open transactions. If the data source does not isolate transactions, then the next option is required.

Option #2: Non-Transactional Updates

For this option, a Procedure is written in Java or SQL Script and scheduled to run on a regular basis. This procedure performs some logic to identify what rows have changed.

Then, "CreateResourceCacheKey()" is called to generate a new *cachekey* value.

Then, the server ID is acquired using "GetProperty('SERVER_ID')".

Then, INSERT a new row into the Status Table with the server ID, resource path, *cachekey*, the status set to 'I', and the *starttime* set to the current time to indicate an in-progress refresh. This should be performed on an independent transaction so it can be committed immediately.

Then, "LoadResourceCacheStatus()" is called to inform the server of the in-progress refresh.

Then perform INSERTs and other operations to create new rows in the storage table as appropriate, making sure all such rows have the new *cachekey* value. This should be performed on an independent transaction so it can be committed when done.

Then, UPDATE all rows in the Status Table with the server ID and resource path that have status 'A' to have status 'C'. This marks the previous active cache data for clearing. Then UPDATE the row into the Status Table with the serverID, resource path, and *cachekey* to set the status set to 'A' and *finishtime* to the current time. This will indicate that this cache key is the new active one. This should be performed on an independent transaction so it can be committed immediately.

Then, "LoadResourceCacheStatus()" is called to inform the server of the newly active data.

Since this approach does require all new rows to be created in the cache data table, it is more expensive than the first option, but since the rows can be copied from the previously active data, it is possible to make this perform reasonably well.

Database Specifics

Each database has suggested and allowed native data types for storing cache data. They are described in the following tables:

In the following tables:

- The **Composite Data Type** column shows each data type that may appear as a projection from a View or Procedure output parameter.
- The **Preferred Native Type** is the type that will be suggested in the DDL when using the feature to create or recreate tables from Studio.
- The **Other Allowed Native Types** column shows other types in the database that can be used as alternatives to the preferred type. A trailing '+' on a number entry means "or higher".

Table 1: File Cache

Composite Data Type	Preferred Native Type	Other Allowed Native Types
BIT	BIT	DECIMAL(1+,0), Larger integer type
TINYINT	TINYINT	DECIMAL(3+,0), Larger integer type, VARCHAR(20+)
SMALLINT	SMALLINT	DECIMAL(5+0), Larger integer type, VARCHAR(20+)
INTEGER	INTEGER	DECIMAL(10+,0), Larger integer type, VARCHAR(20+)
BIGINT	BIGINT	DECIMAL(19+,0), Larger integer type, VARCHAR(20+)
FLOAT	FLOAT	DOUBLE
DOUBLE	DOUBLE	VARCHAR(24+)
NUMERIC(p,g)	NUMERIC(p,q)	DECIMAL(p+,q+), VARCHAR(p+3+), CLOB, Integer type with enough resolution
DECIMAL(p,q)	DECIMAL(p,q)	DECIMAL(p+,q+), VARCHAR(p+3+), CLOB, Integer type with enough resolution
CHAR(n)	CHAR(n)	CHAR(n+), CLOB
VARCHAR(n)	VARCHAR(n)	(VARCHAR(n+), CLOB
CLOB	CLOB	
BINARY(n)	BINARY(n)	BINARY(n+), BLOB
VARBINARY(n)	VARBINARY(n)	VARBINARY(n+), BLOB
BLOB	BLOB	
DATE	DATE	VARCHAR(10+)
TIME	TIME	VARCHAR(15+)
TIMESTAMP	TIMESTAMP	
BOOLEAN	BOOLEAN	BIT, TINYINT, SMALLINT, INTEGER, BIGINT
XML	CLOB	VARCHAR(*) [Clips data if column is too small)
OTHER	(not cacheable)	

Table 2: IBM DB2 v7 / v8

Composite Data Type	Preferred Native Type	Other Allowed Native Types
BIT	SMALLINT	DECIMAL(1+,0), Larger integer type
TINYINT	SMALLINT	DECIMAL(3+,0), Larger integer type, VARCHAR(20+)
SMALLINT	SMALLINT	DECIMAL(5+,0), Larger integer type, VARCHAR(20+)
INTEGER	INTEGER	DECIMAL(10+,0), Larger integer type, VARCHAR(20+)
BIGINT	BIGINT	DECIMAL(19+,0), Larger integer type, VARCHAR(20+)
FLOAT	DOUBLE	VARCHAR(24+)
DOUBLE	DOUBLE	VARCHAR(24+)
NUMERIC(p,q)	DECMAL(p,q), CLOB [if p > 31]	DECIMAL(p+,q+), VARCHAR(p+3+), GRAPHIC(p+3+), CLOB
DECIMAL(p,q)	DECIMAL(p,q) CLOB [if p > 31]	DECIMAL(p+,q+), VARCHAR(p+3+), VARGRAPHIC(p+3+), CLOB, LONG_VARGRAPHIC
CHAR(n)	CHAR(n) CLOB[if n>254]	CHAR(n+), GRAPHIC(n+), VARCHAR(n+), VARGRAPHIC(n+), CLOB
VARCHAR(n)	VARCHAR(n) CLOB[if n>254]	VARCHAR(n+), VARGRAPHIC(n+), CLOB, LONG_VARGRAPHIC
CLOB	CLOB	LONG_VARGRAPHIC
BINARY(n)	BLOB	
VARBINARY(n)	BLOB	
BLOB	BLOB	
DATE	DATE	VARCHAR(10+)
TIME	TIME	VARCHAR(15+)
TIMESTAMP	TIMESTAMP	VARCHAR(26+)
BOOLEAN	SMALLINT	INTEGER, BIGINT
XML	CLOB	VARCHAR(*) [Clips data if column is too small], VARGRAPHIC(*), LONG_VARGRAPHIC
OTHER	(not cacheable)	

Table 3: Microsoft SQL Server 2000 / 2005

Composite Data Type	Preferred Native Type	Other Allowed Native Types
BIT	BIT	DECIMAL(1+,0), Larger integer type
TINYINT	SMALLINT	DECIMAL(3+,0), Larger integer type, VARCHAR(20+), NVARCHAR(20+)
SMALLINT	SMALLINT	DECIMAL(5+,0), Larger integer type, VARCHAR(20+), NVARCHAR(20+)
INTEGER	INTEGER	DECIMAL(10+,0), Larger integer type, VARCHAR(20+), NVARCHAR(20+)
BIGINT	BIGINT	DECIMAL(19+,0), Larger integer type, VARCHAR(20+), NVARCHAR(20+)
FLOAT	REAL	FLOAT, VARCHAR(24+)
DOUBLE	FLOAT	VARCHAR(24+)
NUMERIC(p,q)	DECMAL(p,q), TEXT(if p >38)	DECIMAL(p+,q+), VARCHAR(p+3+), NVARCHAR(p+3+), TEXT, NTEXT
DECIMAL(p,q)	DECIMAL(p,q) TEXT(if p>38)	DECIMAL(p+,q+), VARCHAR(p+3+), NVARCHAR(n+), TEXT, NTEXT
CHAR(n)	CHAR(n) TEXT(if p>38)	CHAR(n+), NCHAR(n+), VARCHAR(n+), NVARCHAR(n+), TEXT, NTEXT
VARCHAR(n)	VARCHAR(n) TEXT ((if n > 255)	VARCHAR(n+), NVARCHAR(n+), TEXT, NTEXT
CLOB	TEXT	NTEXT
BINARY(n)	BINARY(n) IMAGE (if n> 255)	BINARY(n+), IMAGE
VARBINARY(n)	VARBINARY(n) IMAGE (if n> 255)	VARBINARY(n+), IMAGE
BLOB	IMAGE	
DATE	VARCHAR(10)	VARCHAR(10+)
TIME	VARCHAR(15)	VARCHAR(15+)
TIMESTAMP	DATETIME	
BOOLEAN	BIT	TINYINT, SMALLINT, INTEGER, BIGINT
XML	TEXT	VARCHAR(*) [Clips data if column is too small], TEXT
OTHER	(not cacheable)	

Notes

- SQL Server's page size limits the number of bytes that can be stored directly in a column. This means it is possible when executing DDL to get an error that the resulting table requires a row size greater than this limit. The solution is to either raise the page size for the database, or to use indirect storage types such as `TEXT` and `IMAGE`. Composite chooses `TEXT` and `IMAGE` types if a value requires more than 255 bytes of storage for this reason, although SQL Server does allow `VARCHAR` and `VARBINARY` up to 8,000 bytes. Hand tuning of the data types used in a table can be used to improve storage efficiency.
- SQL Server `TINYINT` is range 0 to 255 and Composite `TINYINT` is -128 to 127, so these types are not compatible.
- SQL Server `DATETIME` has accuracy only to within 3.33ms, so some rounding error may occur in the milliseconds.

Table 4: MySQL 4.0 and 4.1

Composite Data Type	Preferred Native Type	Other Allowed Native Types
BIT	BIT	DECIMAL(1+, 0), Larger integer type
TINYINT	TINYINT	DECIMAL(3+, 0), Larger integer type, VARCHAR(20+)
SMALLINT	SMALLINT	DECIMAL(5+, 0), Larger integer type, VARCHAR(20+)
INTEGER	INTEGER	DECIMAL(10+, 0), Larger integer type, VARCHAR(20+)
BIGINT	BIGINT	DECIMAL(19+, 0), Larger integer type, VARCHAR(20+)
FLOAT	FLOAT	VARCHAR(24+)
DOUBLE	DOUBLE	VARCHAR(24+)
NUMERIC(p,q)	NUMERIC(p,q) TEXT (if p>30)	DECIMAL(p+,q+), VARCHAR(p+3+), TINYTEXT, MEDIUMTEXT ,LONGTEXT, Integer type with enough resolution
DECIMAL(p,q)	DECIMAL(p,q) TEXT (if p > 30)	DECIMAL(p+,q+), VARCHAR(p+3+), TINYTEXT, MEDIUMTEXT, LONGTEXT, Integer type with enough resolution
CHAR(n)	CHAR(n) LONGTEXT (if n> 255)	CHAR(n+), TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT
VARCHAR(n)	VARCHAR(n) LONGTEXT (if n > 255)	VARCHAR(n+), TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT
CLOB	CLOB	
BINARY(n)	BLOB LONGBLOB (if n > 255)	TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB
VARBINARY(n)	BLOB LONGBLOB (if n > 255)	TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB
BLOB	LONGBLOB	
DATE	DATE	VARCHAR(10+)
TIME	TIME	VARCHAR(15+)
TIMESTAMP	DATETIME	TIMESTAMP
BOOLEAN	BIT	BIT, BOOL
XML	LONGTEXT	VARCHAR(*), TINYINT, TEXT, MEDIUMTEXT [Clips data if col too small)

Notes

- MySQL removes trailing spaces from strings stored in a `VARCHAR` column and trailing 0x20 bytes from a VARBINARY column.
- MySQL truncates millisecond data from `TIME`, `DATETIME`, and `TIMESTAMP` columns.
- MySQL changes any `NULL` stored in a `TIMESTAMP` column into the current date. Use `DATETIME` to preserve `NULL` values.
- Composite creates tables using the "`utf8`" character set to properly handle international characters. You can create the tables using other character sets based on your performance and character set needs.
- Small variations in the least significant digits may be encountered when storing `FLOAT` and `DOUBLE` values due to the way the driver handles and database stores such data.

Table 5: Oracle 8i, 9i, 10g

Composite Data Type	Preferred Data Type	Other Allowed Native Types
BIT	NUMBER(1, 0)	NUMBER(1+, 0)
TINYINT	NUMBER(3, 0)	NUMBER(3+, 0), VARCHAR(20+), NVARCHAR(20+)
SMALLINT	NUMBER(5, 0)	NUMBER(5+, 0), VARCHAR(20+), NVARCHAR(20+)
INTEGER	NUMBER(10, 0)	NUMBER(10+, 0), VARCHAR(20+), NVARCHAR(20+)
BIGINT	NUMBER(19, 0)	NUMBER(19+, 0), VARCHAR(20+), NVARCHAR(20+)
FLOAT	FLOAT	VARCHAR(24+), FLOAT, BINARY_FLOAT, BINARY_DOUBLE
DOUBLE	VARCHAR(24)	VARCHAR(24+), FLOAT, BINARY_DOUBLE
NUMERIC(p,q)	NUMBER(p,q) CLOB [if p> 38]	NUMBER(p+,q+), VARCHAR2(p+ 3+), NVARCHAR2(p+3+), CLOB
DECIMAL(p,q)	NUMBER(p,q) CLOB [if p> 38]	NUMBER(p+,q+), VARCHAR2(p+ 3+), NVARCHAR2(p+3+), CLOB
CHAR(n)	CHAR(n) CLOB [if n > 2000]	CHAR(n+), VARCHAR2(n+), NVARCHAR2(n+), CLOB
VARCHAR(n)	VARCHAR(n) CLOB [if n > 4000]	VARCHAR2(n+), NVARCHAR2(n+), CLOB
CLOB	CLOB	
BINARY(n)	RAW(n) BLOB [if n> 255]	RAW(n+), BLOB
VARBINARY(n)	RAW(n) BLOB [if n> 255]	RAW(n+), BLOB
BLOB	BLOB	
DATE	VARCHAR2(10)	VARCHAR2(10+), NVARCHAR2(10+)
TIME	VARCHAR2(15)	VARCHAR2(15+), NVARCHAR2(15+)
TIMESTAMP	TIMESTAMP(9) [FOR 9i AND 10g) DATE [for 8i]	
BOOLEAN	NUMBER(1,0)	NUMBER(1+,0)
XML	CLOB	VARCHAR(*), NVARCHAR2(*) [Clips

Composite Data Type	Preferred Data Type	Other Allowed Native Types
		data if column is too small]
OTHER	(not cacheable)	

Notes

- Oracle changes any empty string stored in a VARCHAR2 or NVARCHAR2 column to NULL. This can have the effect of altering empty string data stored in such columns.
- Oracle FLOAT columns have a maximum of 126 digits, equivalent to a floating point number with exponent “E125”. Composite FLOAT values have a maximum of E38 and DOUBLE values have a maximum of “E308”. This is why VARCHAR is used to store Composite DOUBLE values by default. The FLOAT type may be used if your values fit within that range.

Table 6: Sybase

Composite Data Type	Preferred Data Type	Other Allowed Native Types
BIT	BIT	DECIMAL(1+,0), Larger integer type
TINYINT	SMALLINT	DECIMAL(3+,0), Larger integer type, VARCHAR(20+), NVARCHAR(20+)
SMALLINT	SMALLINT	DECIMAL(5+,0), Larger integer type, VARCHAR(20+), NVARCHAR(20+)
INTEGER	INT	DECIMAL(10+,0), VARCHAR(20+), NVARCHAR(20+)
BIGINT	DECIMAL(19,0)	DECIMAL(19+,0)
FLOAT	REAL	FLOAT, VARCHAR(24+)
DOUBLE	FLOAT	VARCHAR(24+)
NUMERIC(p,q)	DECIMAL(p,q) TEXT [if p>38]	DECIMAL(p+,q+), VARCHAR(p+3+), NVARCHAR(p+3+), TEXT
DECIMAL(p,q)	DECIMAL(p,q) TEXT [if p>38]	DECIMAL(p+,q+), VARCHAR(p+3+), NVARCHAR(p+3+), TEXT
CHAR(n)	CHAR(n) TEXT [if n>255]	CHAR(n+), NCHAR(n+), VARCHAR(n+), NVARCHAR(n+), TEXT
VARCHAR(n)	VARCHAR(n) TEXT [if n>255]	VARCHAR(n+), NVARCHAR(n+), TEXT
CLOB	TEXT	NTEXT
BINARY(n)	BINARY(n) IMAGE [if n>255]	BINARY(n+), IMAGE
VARBINARY(n)	VARBINARY(n) IMAGE [if n>255]	VARBINARY(n+), IMAGE
BLOB	IMAGE	
DATE	VARCHAR(10)	VARCHAR(10+)
TIME	VARCHAR(15)	VARCHAR(15+)
TIMESTAMP	VARCHAR(26)	DATETIME, VARCHAR(26+)
BOOLEAN	BIT	TINYINT, SMALLINT, INTEGER
XML	TEXT	VARCHAR(*) [Clips data if column too small]
OTHER	(not cacheable)	

Notes

- Sybase's page size limits the number of bytes that can be stored directly in a column. This means it is possible when executing `DDL` to get an error that the resulting table requires a row size greater than this limit. The solution is to either raise the page size for the database, or to use indirect storage types such as `TEXT`. Composite chooses `TEXT` types if a value requires more than 255 bytes of storage for this reason, although Sybase does allow `VARCHAR` and `VARBINARY` with larger size. Hand tuning of the data types used in a table can be used to improve storage efficiency.
- Sybase `DATETIME` has accuracy only to within `3.33ms`, so some rounding error may occur in the milliseconds.

Table 7: Teradata

Composite Data Type	Preferred Data Type	Other Allowed Native Types
BIT	BYTEINT	DECIMAL(1+,0), Larger integer type
TINYINT	BYTEINT	DECIMAL(3+,0), Larger integer type, VARCHAR(20+), VARGRAPHIC(20+)
SMALLINT	SMALLINT	DECIMAL(5+,0), Larger integer type, VARCHAR(20+), VARGRAPHIC(20+)
INTEGER	INTEGER	DECIMAL(10+,0), VARCHAR(20+), GRAPHIC(20+), VARGRAPHIC(20+)
BIGINT	CHAR(20)	DECIMAL(19+,0), VARCHAR(20+)
FLOAT	FLOAT	VARCHAR(24+)
DOUBLE	FLOAT	VARCHAR(24+)
NUMERIC(p,q)	DECIMAL(p,q) CLOB [if p>18]	DECIMAL(p+,q+), VARCHAR(p+3+), GRAPHIC(p+3+), CLOB
DECIMAL(p,q)	DECIMAL(p,q) CLOB [if p>18]	DECIMAL(p+,q+), VARCHAR(p+3+), GRAPHIC(p+3+), CLOB
CHAR(n)	CHAR(n) CLOB [if n>32,000]	CHAR(n+), GRAPHIC(n+), VARCHAR(n+), CLOB
VARCHAR(n)	VARCHAR(n) CLOB [if n>32,000]	VARCHAR(n+), VARGRAPHIC(n+)
CLOB	CLOB	
BINARY(n)	BYTE(n) BLOB [if n>32,000]	BYTE(n+)
VARBINARY(n)	VARBYTE(n) BLOB [if n>32,000]	VARBYTE(n+)
BLOB	BLOB	
DATE	DATE	VARCHAR(10+)
TIME	VARCHAR(15)	VARCHAR(15+)
TIMESTAMP	TIMESTAMP	VARCHAR(26+)
BOOLEAN	BYTEINT	SMALLINT, INTEGER
XML	CLOB	VARCHAR(*) [Clips data if column too small]
OTHER	(not cacheable)	

ABOUT COMPOSITE SOFTWARE

Composite is the SOA data services leader. Composite data services are the fastest path for delivering existing data to your next-generation applications.

With Composite's award-winning Enterprise Information Integration (EII) technology, high performance information server, easy-to-use data services development suite, and pre-built applications services, you can build applications faster without worrying about data location and complexity. Find, access, integrate, and deliver on-demand information from critical and disparate sources, including SAP, Siebel, and Oracle applications located across the enterprise.

Composite leverages existing data assets and reusable data services across both SOA and legacy environments, simplifying your SOA migration and improving your business responsiveness while lowering costs and SOA risks.

Composite's products are proven at over 100 large enterprises, including five of the top six US investment banks, and embedded by Cognos, Informatica, BMC, and others.

For more information, please visit www.compositesw.com.

Headquarters
2655 Campus Drive, Suite 200
San Mateo, CA 94403

support@compositesw.com