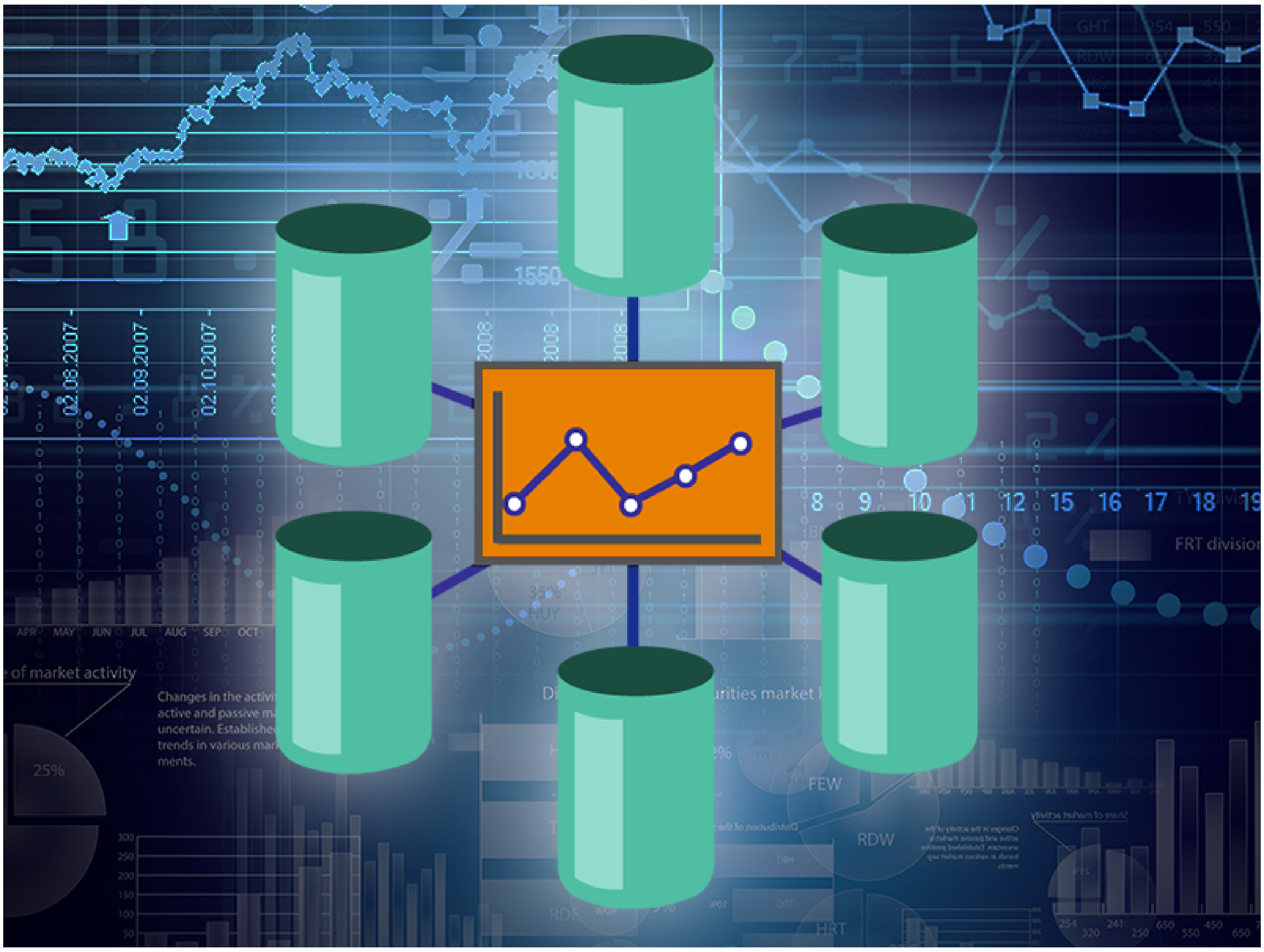
Article
Alessandro Marin · May 7, 2018 5m read

DeepSee: Databases, Namespaces, and Mappings - Part 5 of 5

The following post concludes the series with a list of all databases seen in the example for the fully flexible architecture.



Databases and mappings

The databases described below should be defined for each namespace, except for the application code (which in our example is stored in the APP-CODE database) that needs to be shared between namespaces. All namespaces where the DeepSee implementation runs should use global mappings so that globals are stored in and read from the correct database.

Database 1: DeepSee Cache

This database should store all DeepSee cache, that is the ^DeepSee.Cache.* and ^DeepSee.JoinIndex globals

(Note: this page in the documentation lists more globals as DeepSee cache but the ^DeepSee.Cache.* globals are by far the most important).

It is highly recommended map out the DeepSee cache globals to a dedicated database. The DeepSee cache globals should never be journaled otherwise DeepSee will perform poorly and journal files might be huge.

Map the ^DeepSee.Cache.* and ^DeepSee.JoinIndex globals to this Database. Optionally, also map to this Database the ^DeepSee.LastQuery and ^DeepSee.QueryLog globals, which are the globals storing a log for all executed MDX queries.

Database 2: Implementation and Settings

This database contains the ^DeepSee.* globals, which include most of the DeepSee implementation. This database includes all DeepSee cube or subject area definitions as well as information for many functionalities such as Cube Manager (^DeepSee.CubeManager*), cubes definitions and settings (^DeepSee.Cubes, ^DeepSee.Dimensions), DeepSee items (^DeepSee.Folder*, ^DeepSee.FolderItem*), pivot variables (^DeepSee.Variables), term lists (DeepSee.TermList), user settings (^DeepSee.DashboardSettings), DeepSee overrides (^DeepSee.Overrides), and others.

It is recommended to store these functionalities in a separate Read-Write database to be able to journal and regularly back this database up. By doing so it will possible to restore all definitions, settings, and user data in case of a disruptive event.

Map all the remaining ^DeepSee* globals to this Database.

Database 3: DeepSee Updates

To keep the cubes updated with the source tables DeepSee uses the ^OBJ.DSTIME and ^DeepSee.Update globals. On the Production Database, store the ^OBJ.DSTIME global in this database and mirror it to the Analytics server. If the system is running an adhoc or a recent versions of Caché where ^DeepSee.Update is used (normally available from Caché 2016.1.2), also store ^DeepSee.Update on this database. In this case, the database on the Analytics server storing ^OBJ.DSTIME must be Read-Write to be able to purge ^OBJ.DSTIME after it is copied to ^DeepSee.Update. Notice that using this database is necessary when the database hosting data (APP-DATA in our examples) is Read-only, otherwise it would be impossible to purge ^OBJ.DSTIME.

Journaling should be enabled on the Production server. Map ^OBJ.DSTIME and the ^DeepSee.Update to this Database.

Database 4: Fact Tables

DeepSee cubes are based on a source class but populate and use fact and dimension tables. These tables contain information for each record built in the cube and are used by DeepSee at runtime.

The choice of defining a dedicated database for fact, dimension tables, and indices is often taken to apply a journaling setting different than other databases. Please read the note below on building cubes when journaling is enabled. Another reason to map fact, dimension tables, and indices to a separate database might be to be able to define a non-default block size (for example 16K block instead of the default 8K). Using a different block size can help performance of MDX queries.

The fact and dimension tables are stored in the ^DeepSee.Fact* and ^DeepSee.Dimension* globals. DeepSee indices are stored in ^DeepSee.Index, and ^DeepSee.JoinIndex global is used when cubes define relationships. Map these globals to this database.

Database 5: DeepSee Indices

DeepSee indices are the indices for the cube ’ s Fact Table.

The reason to store DeepSee indices in a separate database is the possible big size of the ^DeepSee.Index global. Having a different journaling settings and defining a non-default block size could facilitate recovery and help performance.

Journaling is optional: choose the same setting as in the previous database.

Map the ^DeepSee.Index global to this Database.

Note on journaling and building cubes

Users should be aware that building cubes deletes and recreates the cubes ’ fact and index tables. This means that when journaling is enabled the SETs and KILLs of globals such as ^DeepSee.Fact*, ^DeepSee.Index are logged in journal files. As a result, rebuilding cubes might lead to a huge amount of entries in the journal files and possible problems with disk space.

It is recommended to map fact tables and indices to one or two separate databases (Databases 4 and 5 above).

For the Fact and Indices databases journaling is optional and depends on the business needs. It might preferable to disable journaling when cubes are relatively small and fast to build, or cubes are scheduled to rebuild periodically.

Enable journaling on this database when cubes are relatively big and it takes too long to rebuild them. The ideal case to keep journaling on is when cubes are in a stable state and only get periodically synchronized, but not built. One way to safely build cubes is to temporarily disable journaling for the Fact and Indices databases (Databases 4 and 5, respectively).

Summary

Database	Globals to map	Function	Settings
1 - Source data		Get data from Production	Mirrored from Production. Share with all namespaces
2 - Source code		Split code from data	Share with all namespaces
3 - DeepSee Cache	^DeepSee.Cache.* ^DeepSee.JoinIndex ^DeepSee.LastQuery ^DeepSee.QueryLog	Allow journaling other databases while keeping the DeepSee cache non-journaled	Disable journaling
4 - Implementation and Settings	^DeepSee.*	Allows restoring the DeepSee implementation and user settings	Enable journaling, backup regularly
5 - DeepSee Updates	^OBJ.DSTIME ^DeepSee.Update	Allows keeping cubes current	Mirrored from Production. Keep Read-Write
6 - Fact Tables	^DeepSee.Dimension* ^DeepSee.Fact	Be able to journal or not Block size can be changed	Journaling is optional

	^DeepSee.JoinIndex		
7 - DeepSee Indices	^DeepSee.Index	Define this Database if cubes are big and you need better performance in queries/builds, otherwise store with Fact Tables (Database 5)	Journaling as in Fact Tables database

Conclusions

This series outlines best practices related to databases and mappings that you should consider for a Business Intelligence implementation using Caché and DeepSee. It is certainly possible to successfully deploy DeepSee implementations using a smaller number of Databases than the ones recommended in this series, but this might expose the implementation to limitations.

#Analytics #Beginner #Databases #Deployment #Mapping #Tutorial #InterSystems IRIS BI (DeepSee)

Source URL:https://community.intersystems.com/post/deepsee-databases-namespaces-and-mappings-part-5-5