SafePeak Configuration Guide

Prerequisite installation steps:

The following steps are pretty simple and are achieved usually without further guidance:

1) Install SafePeak on the machine;
2) Your SQL Instance was "created" in SafePeak, forming a proxy bridge to your SQL Server;
3) SafePeak completes the initial Metadata learning, showing in the Dashboard: Cache=Enable;
4) Application has been connected to SafePeak and some SQL traffic was executed;

SafePeak configuration tools, options and process steps:

SafePeak configuration is about application-database performance and it should be done in several cycles. Perform the below steps 1-2-3 several times, until the desired results are reached. Usually an efficient and effective the configuration process requires between 2 to 4 such cycles. The steps are:

1. Run the application traffic (via SafePeak) – Test application performance;
2. Review and analyze received results, what is cached, what needs configuration etc.;
3. Configure;

Configuration guiding tools:

SafePeak comes with a set of tools that will guide you thru configuration and enable you focus the efforts only on the right objects, understanding their performance level without cache, cache state, level of cache effectiveness, investigate exceptions, and even dig deeper into profiling your sql traffic.

The review and analysis of received results is done using three available tools:

1. SafePeak dashboard – main configuration is driven by SafePeak SQL Analytics data;
2. Log files – to verify no exceptions that require to be handled, like the Server.0.log file;
3. Access to raw sql traffic information and safepeak configuration, using (if needed) SafePeak's internal repository MySQL database;

Configuration tools:

Several configuration tools available for users:

1. **SafePeak Dashboard** – the main configuration tool (in most cases the only required tool);
2. **Special settings in the *safepeak.xml* file** – Settings like <IgnoreTsqlDdl>, <IgnoreParserError>
3. **Cache Warm-up utility: the *SchedulerSerialization.xml* and *SafePeakInstanceWarmUp.xml*** – An advanced additional utility enabling to "push" certain queries based on list in an xml file, ensuring the data will be in cache before. Activation via SchedulerSerialization.xml and configuration using the SafePeakInstanceWarmUp.xml;

Configuration steps and process:

Configuration process and steps:

#	Step	
1	Eliminate all "Cache=Disable" events	Configure the Dynamic Objects and Set to ignore unneeded events
2	Activate inactive patterns	Manually activate maximum SQL Patterns set as Auto Caching=No.
3	Optimize non-cacheable patterns	Review maximum of non-cacheable stored procedures, and where optimize their settings in SafePeak (some can be cacheable).
4	Define semi-dynamic caching patterns	Improve Cache Hits of certain SQL Patterns by reducing their sensitivity to tables' updates.
5	Consider using Caching warm-up tool	Improve Cache Hits by of certain SQL Queries by configuring a list of queries that will be frequently loaded into cache.

Steps 1+2 are crucial to configure in every SafePeak deployment.

Step 3 can make a difference in some installations, especially where the traffic analysis shows higher WRITEs percent of traffic than expected, and investigation shows that some READ-In-Concept stored procedures are set as Non-Cacheable in SafePeak.

Step 4 can make a difference in some installations, and is relatively easy to configure.

Step 5 is designed for special cases, where caching desired improvement for a specific small-medium list of slow running queries. More manual configuration is not applied in most cases.

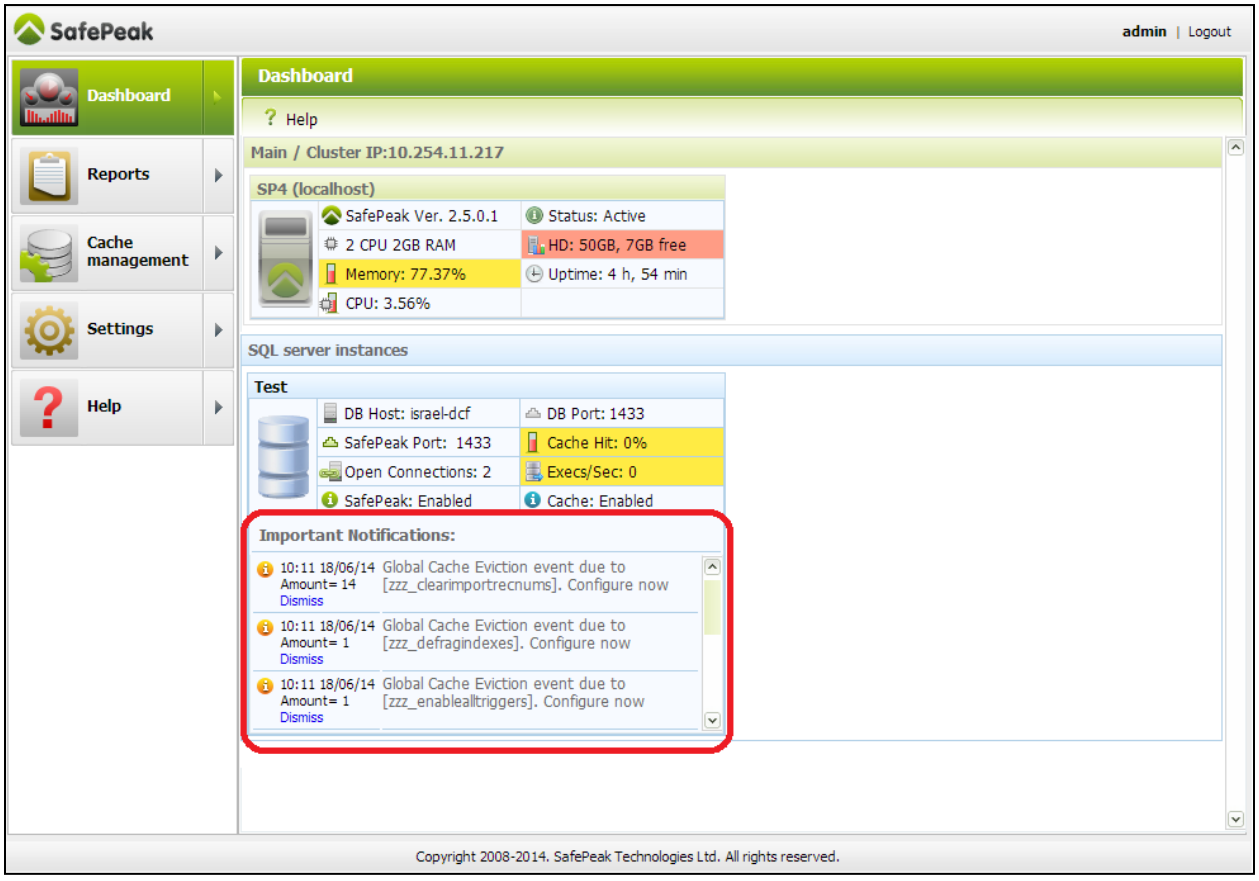
Step 1 – Eliminate "Cache=Disable" events

Configure everything that is needed to ensure that Cache=Disable events (as shown on ***Dashboard***) will not happen (causing global cache eviction and global cache lock), a critical configuration part. Once defined, these settings can be saved and exported to another environment.

These events are causing SafePeak CORE processing engine to "play it safe": **a)** A global cache eviction (at some cases); **b)** A Cache temporary Disable/Lock; **c)** Metadata Learning service to rescan the database schema for any changes; **d)** Resume caching when finished. During this process, the queries are not delayed and returned to application, but there is no caching.

Step 1.A – Configure all activated dynamic objects

Generate some application traffic, enter all main screens. During this process you may notice in SafePeak Dashboard screen: Cache=Disable. Wait 10-15 minutes for statistics calculation. The *Dashboard Screen → Important Notifications* will show a list of objects (usually stored procedures) that are causing Cache=Disable events and require configuration.



These are mostly Stored Procedures ("procs") or User Functions that SafePeak's Metadata Learning couldn't fully understand and signed in safest data integrity mode: On every occurrence: Evict all cache + Lock the cache and rescan metadata for schema changes. Since most such cases happen in complex code of Dynamic SQL parsing, these objects are called in SafePeak: "Dynamic Objects".

Configure All Dynamic Objects shown in the Notifications area and then "dismiss" all of them. Learn how to it here: How to Configure Dynamic Objects causing "Cache= Disable/Lock".

Repeat this process twice, to ensure all such objects were configured.

Step 1.B – Ensure no other *global cache disable* events happen

Generate application traffic again. Monitor Dashboard "Cache=Enabled" mode in Dashboard screen. Open in Notepad++ the server log file: \SafePeak\Instances\[Your Instance]\Logs\ Server.0.log. Go to the bottom of the file and search up for these words: "***global cache disable***". Other than stored procedures that are set as dynamic objects, there are two possible reasons:

a. T-SQL DDL commands
b. T-SQL query that SafePeak parser failed to understand

This is a rare case, as the T-SQL DDLs are usually sent by DBAs or by rare application event (like upgrade deployment). However, if the events are very frequent – consider configuring SafePeak to ignore it:

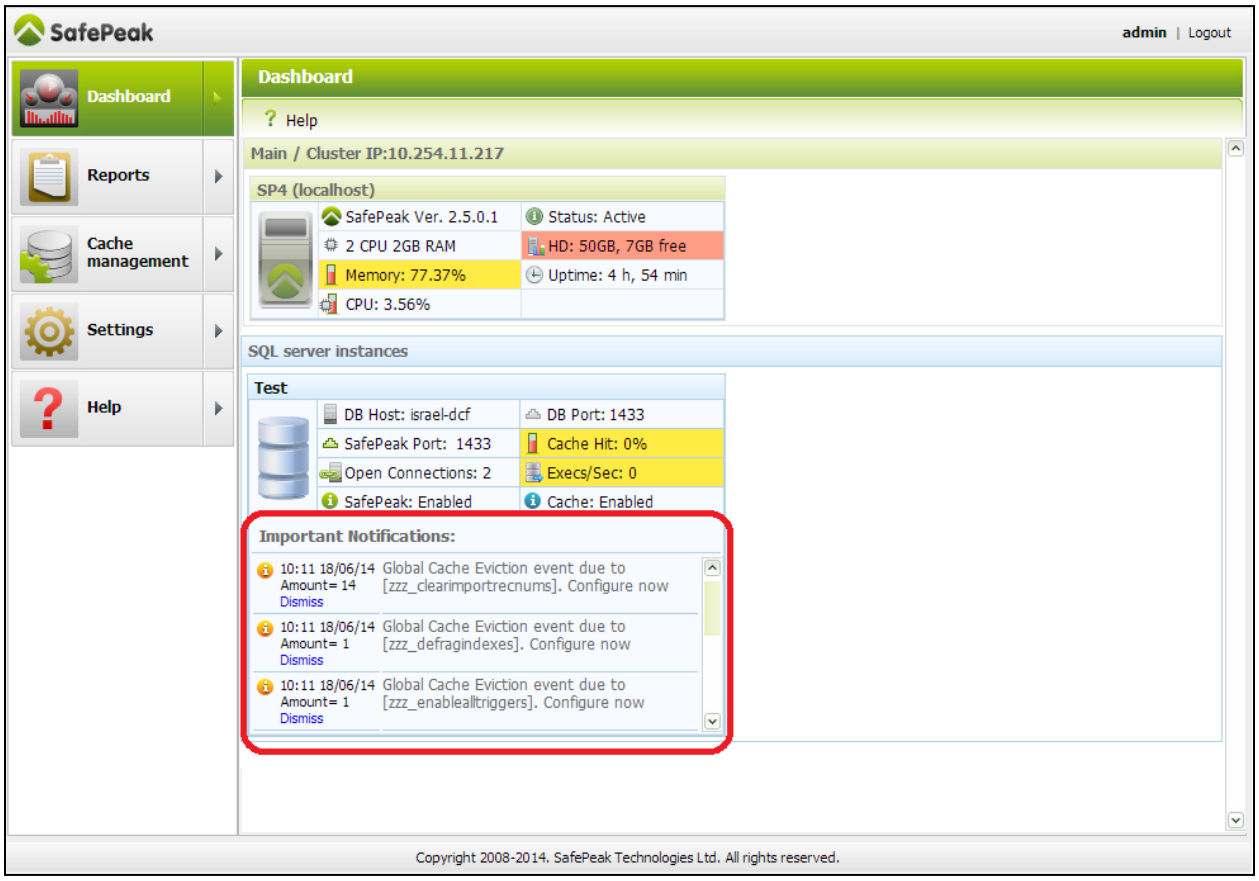
1. In the following file: \SafePeak\Instances\[Your Instance]\Safepeak.xml Set elements <IgnoreTsqlDdl> and < IgnoreParserError> to ***"on"***:
   <IgnoreTsqlDdl>***on***</IgnoreTsqlDdl>
   <IgnoreParserError>***off***</IgnoreParserError>

2. Refresh the loaded settings of [SafePeak Core Service *your-instance-name*]. Two possible ways:
   a. **By restart of the windows services (in this order):**
      i. [SafePeak Network Proxy Service *your-instance-name*]
      ii. [SafePeak Core Service *your-instance-name*]

   b. **By HTTP API command (*ServerManagementPort* = SafePeak instance port + 102)**: http://localhost:[ServerManagementPort]/reload.xml?key=1234&q=setting

How to Configure Dynamic Objects causing "Cache= Disable/Lock"

The first and most critical step is: **Configure all objects that will pop-up in the Dashboard → Notifications area.**

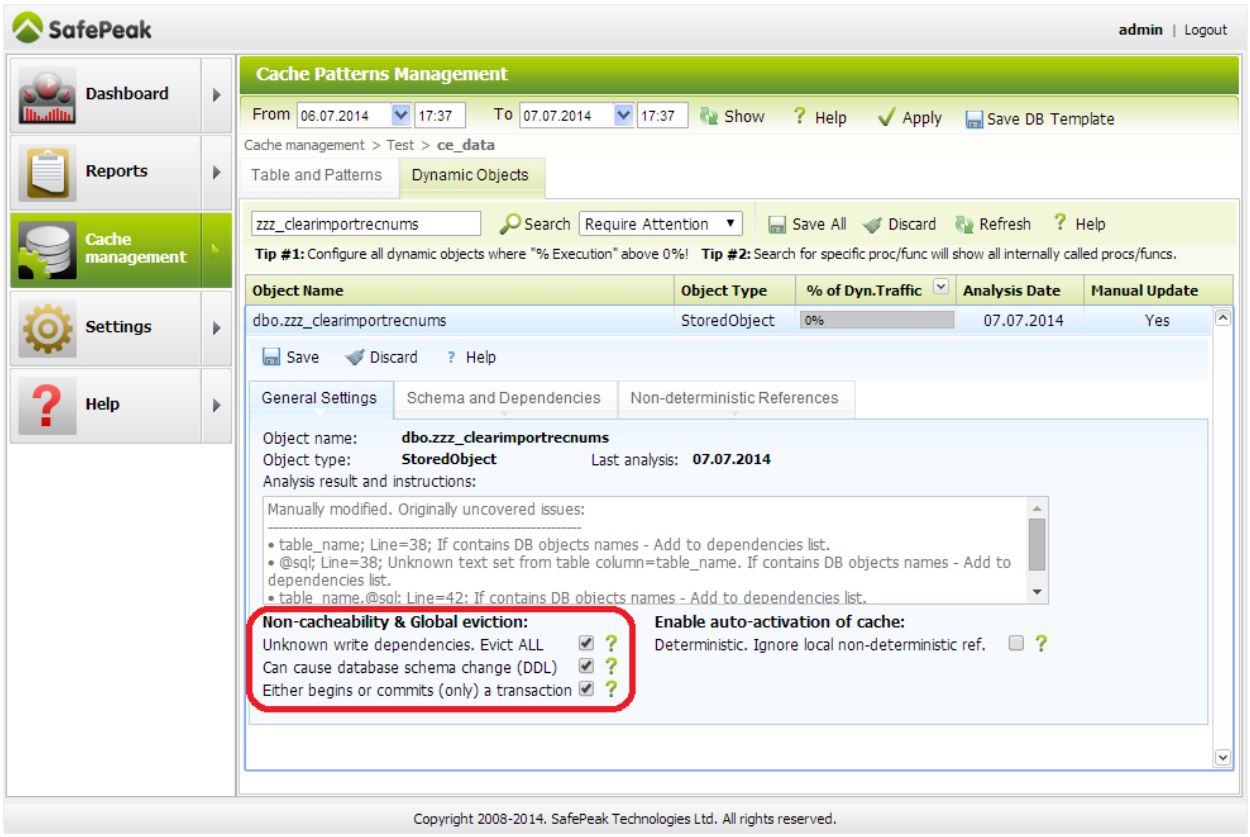


These are mostly Stored Procedures ("procs") or User Functions that SafePeak's Metadata Learning couldn't fully understand and signed in safest data integrity mode: On every occurrence Evict all cache + Lock the cache and rescan metadata for schema changes. Since most such cases happen in complex code of Dynamic SQL parsing, these objects are called in SafePeak: "Dynamic Objects".

SafePeak takes most cautious approach regarding data integrity of Dynamic Objects: SafePeak decides that there is a chance that there is a DML command (write) to an unknown table and therefore to ensure full data integrity SafePeak cleans ALL Cache on every object call, directly or indirectly (in case another procedure calls this problematic one). Also, if SafePeak decides there is a chance for DDL command inside (create/alter/drop), it will issue a rescan of the metadata. During rescan SafePeak will lock the cache (usually for several seconds) during which the application will work, but no cache will happen. In the example above, the object [***zzz_clearimportrecnums***] was called 14 times each time causing full cache eviction and cache lock, which of course severely reduces cache effectiveness.

Configuring a dynamic object

By clicking the notification line in the dashboard the user is forwarded to the Dynamic Objects screen:



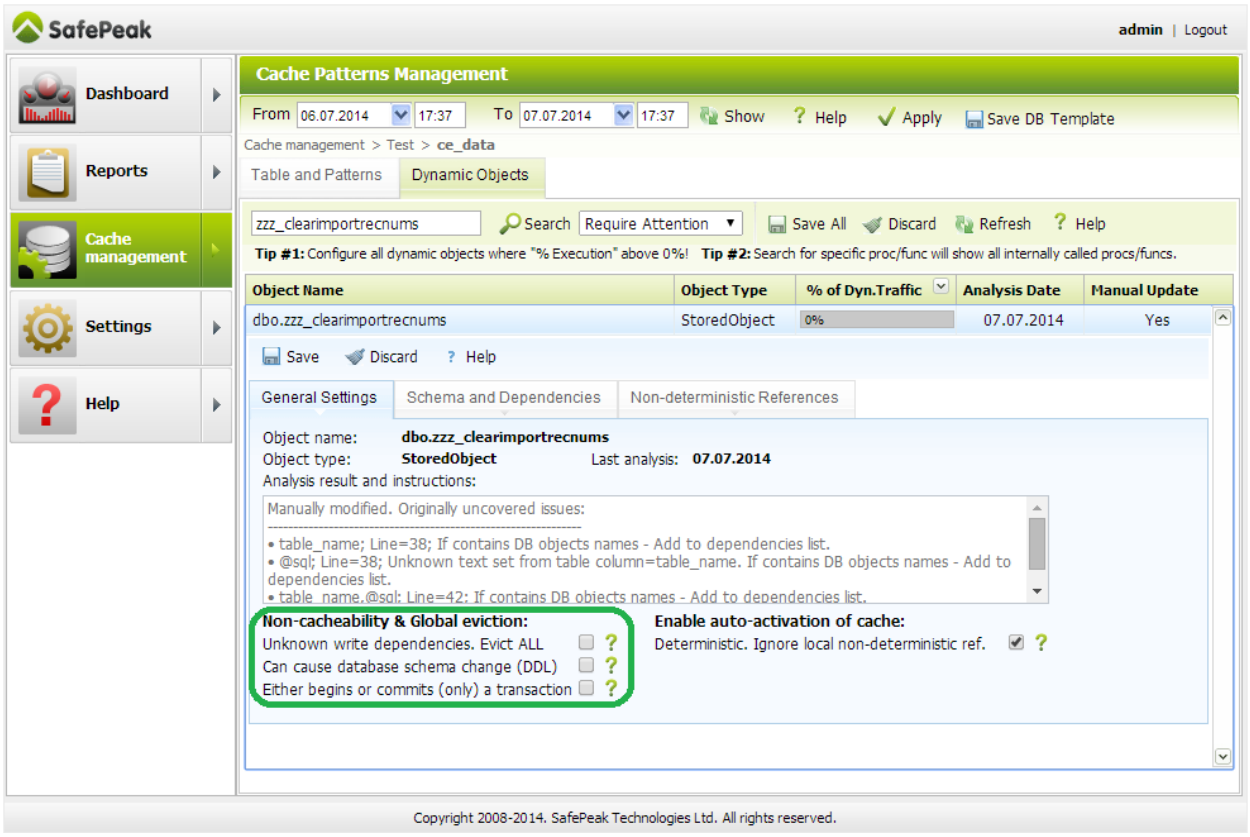
Main reasons for an object to become "dynamic object":

a) **Complex Dynamic SQL:** The object contains code that dynamically creates and executes SQL statements. SafePeak parses the dynamic SQL code, however there are cases where not all information is available and SafePeak requires manual completion of information;
b) **Encrypted objects:** SafePeak does not analyze encrypted objects;
c) **CLR objects:** SafePeak does not analyze CLR objects;
d) **SQL parsing failure:** Parsing SQL syntax issue.

Message example: *table_name,@sql; Line=42; If contains DB objects names - Add to dependencies list.*
Meaning: On line #42 there are two local variables / parameters, [*table_name*] and [*@sql*] that are used in Dynamic SQL construction. Three options are possible:

1) **Precaution due to textual parameter used:** A constructed dynamic SQL uses a text parameter. SafePeak cannot know its potential content: Only a text "filter" in a where clause; or a full sql query can be passed to the procedure and used inside the dynamic SQL. If it's the latter: Add (or just verify) the relevant objects as Dependencies in the Schema and Dependencies tab.
2) **Tables/Views are being READ /WRITE inside the dynamic SQL, or user functions and stored procedures are being EXECUTED, but SafePeak could not figure out which.** Add these objects Dependencies list as READ / WRITE / EXECUTE (accordingly).

Optimal configuration – Removing the Global Eviction/DDL settings



Main goal is to uncheck all "Non-cacheability & Global eviction" checkboxes (and instead add or verify the required Read/Write/Execute dependencies). The checkboxes meanings are:

- **Evict ALL** – Evicts all server cache on every object call.
- **DDL** – Disable Cache to rescan the Metadata for database schema change
- **Transaction** – A rare case, where one procedure begins a transaction and another closes it.

The "Deterministic-Ignore local non-deterministic references" checkbox

A database object (such as view, stored procedure or user defined function) can have references to system **non-deterministic functions** (we'll call them **NDFs**), such as GETDATE, @@ERROR, USER_NAME. The NDFs can return each time a different result. For example, GETDATE() returns a date-time that includes also the quickly changing milliseconds. Objects and SQL Patterns with non-deterministic references are not activated automatically. The list of them can be seen in the "*Non-deterministic References*" tab.

In many cases it is right to choose to ignore the NDFs and activate caching. For example:

- ***"select datepart(day, getdate())"*** *or* ***"SELECT all customers that today is their birthday"*** – Both queries can be cached since the result remains the same until midnight. Caching of these SQL Patterns should be manually enabled, with a setting a scheduled cache eviction at "00:00".

Ignoring the NDFs and activating cache can be done at the SQL Object level (like for a stored procedure) and at the SQL Pattern level.

The checkbox in this screen enables to ignore the locally called NDFs. Three states are possible:

- No NDFs present locally or in internal objects – Checked + Read-only. Caching of SQL Patterns with calling/executing this object are automatically activated;
- **NDFs are present in internal objects (like called procedures/functions)** – the non-deterministic setting is inherited. State = Unchecked + Read-only. Caching is not activated automatically;
- **NDFs are present and only in the schema of this object** – Default state = Unchecked, Manual override is possible. Overriding manually tells SafePeak to ignore the locally called NDFs and activate caching for the relevant SQL Patterns;

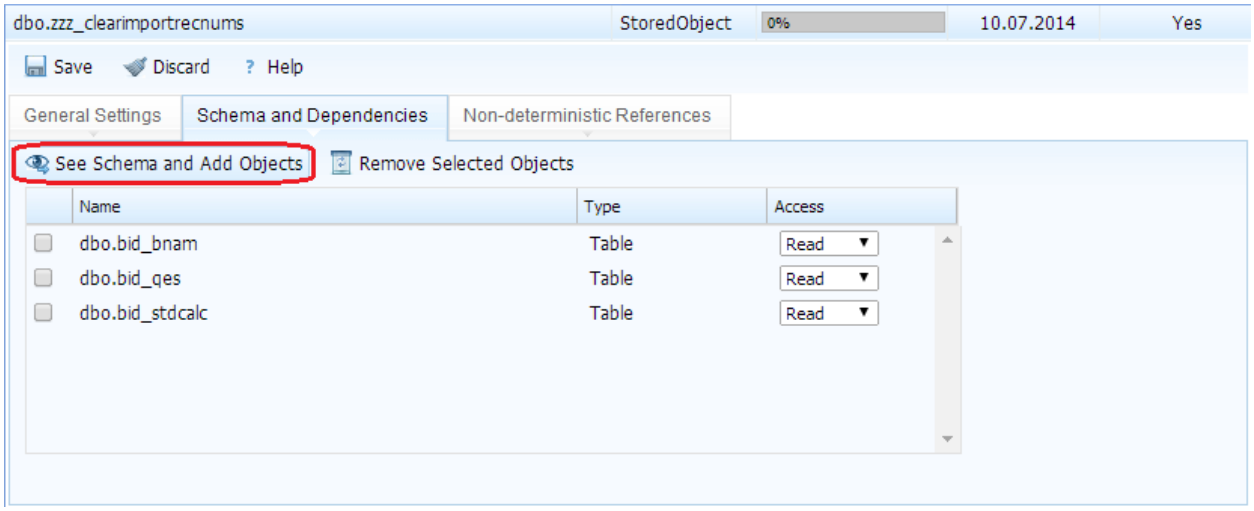
Dependencies and what they mean: Read / Write / Execute

An object (like "*get_CustomerOrders*") that has these "READ" dependencies: "Customers" and "Orders": then the object performs only READs (so it can be used for cache), but the cache pattern (and cached data) is sensitive to write to "Customers" and "Orders" tables.

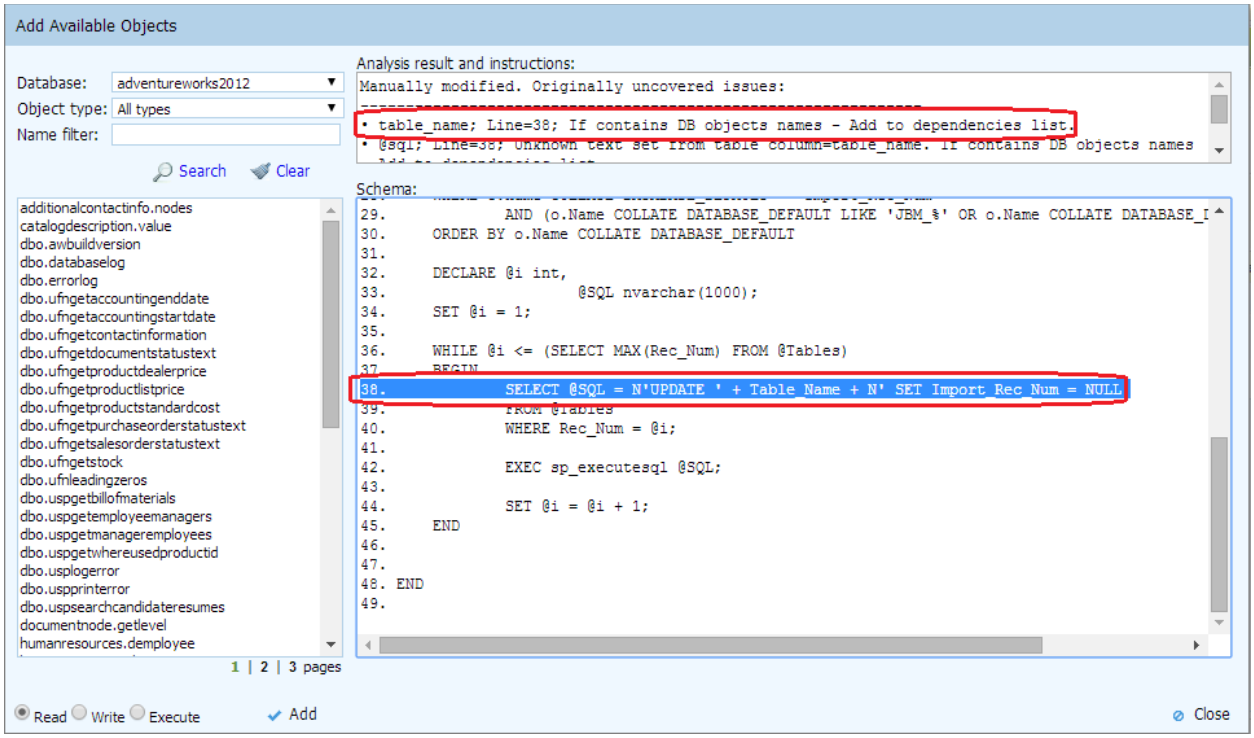
An object (like "*update_CustomerDetails*") that has "WRITE" dependencies to "Customers" table: such object cannot be cached and every time it is called by the application it will evict all cache items that have "Customers" read dependency.

"EXECUTE" dependency simply inherits the dependencies of the inherited object.

Managing dependencies is done in the "Schema and Dependencies tab.



To add dependencies – Click "*See Schema and Add Objects*":

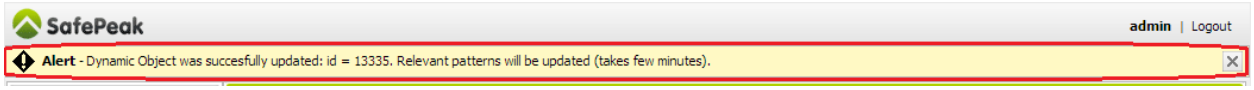


In the above example you are required to specify which Tables are going to be updated on line 38.

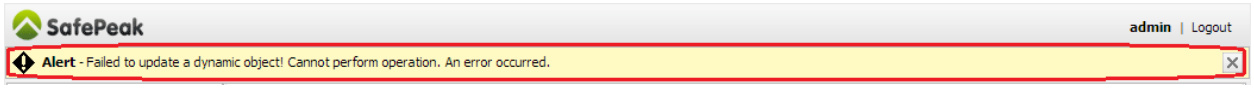
1. Find these tables in list on the right;
2. Since it is an UPDATE command – Choose "*Write*" radio button;
3. Click "Add";
4. Close when finished.

Saving the object

After finishing object configuration click the "***Save***" button. The following message will appear at the top of the screen:



In case you see "Save failed" message – just retry again:



Automated settings update of the relevant SQL Patterns

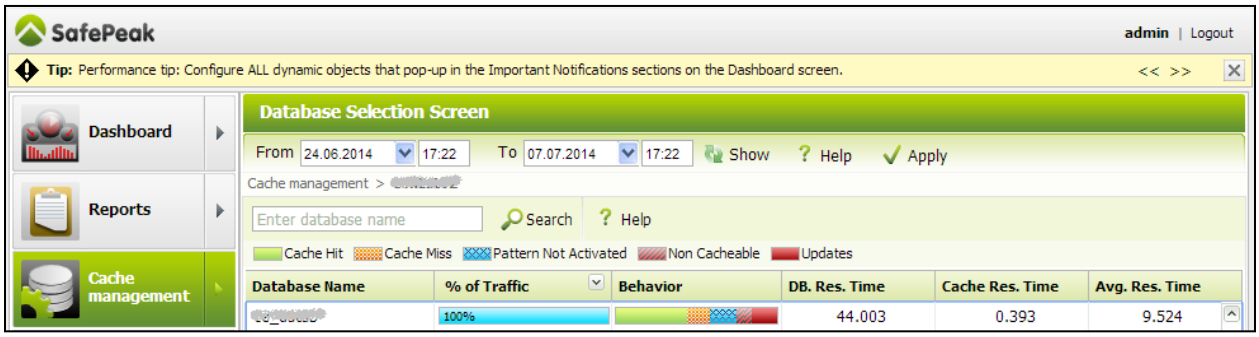
The patterns created for this object will receive your updates. Previously Non-Cacheable stored procedures, due to the Non-cacheability checkboxes or due to previously defined "Write" Dependencies, but were removed or converted to "Read" type – it is possible that a new pattern will need to be created, which will happen few minutes after your application will send such query again.

Steps 2-4: Cache Management of SQL Patterns

Cache Management screens – general review

The cache management screens are the main operation and configuration area. The section combines both your databases schematic representation and SQL Analytics regarding its usage and of course management of SafePeak caching rules. The SQL Analytics data represents ALL queries (not just samples), while the statistics are summarized every 15 minutes for every quarter of an hour.

Enter the Cache Management screen. Every screen in this section allows you to both see the database structure and understand how it is being utilized by the application traffic:



- ***% of Traffic*** – how much this object is being used
- ***Behavior*** – The colored behavior immediately shows the level of caching and read-write usage*:*
  Cache hits, *Cache Misses*, *Unmatched queries*, *Not-Activated Patterns*, *Updates (writes)*
  - *Cache Hits* – Queries were found in cache.
  - *Cache Misses* – Queries were not found in cache, data fetched from the database and placed into cache
  - *Unmatched Queries* – A first time SafePeak see a query so a pattern does not exist yet
  - *Non-Activated Patterns* – Queries have patterns, but were not activated for cache. Two possible reasons:
    - The pattern is Inactive and you can activate it;
    - There was a cache disable/lock during execution of that query (see section Eliminate "Cache=Disable" events)
  - *Updates (writes)* – Write type queries and stored procedures;

  Placing a mouse over the Behavior bar opens a Pop-up with detailed information, how many exactly queries were called of every type;

- ***Database Res. Time*** – Average speed (in milliseconds) for all queries sent to the Database;
- ***Caching Res. Time*** – Average speed (in milliseconds) for all queries returned from Cache;
- ***Avg. Res. Time*** – Average speed (in milliseconds) for All queries, both Database and Cache;

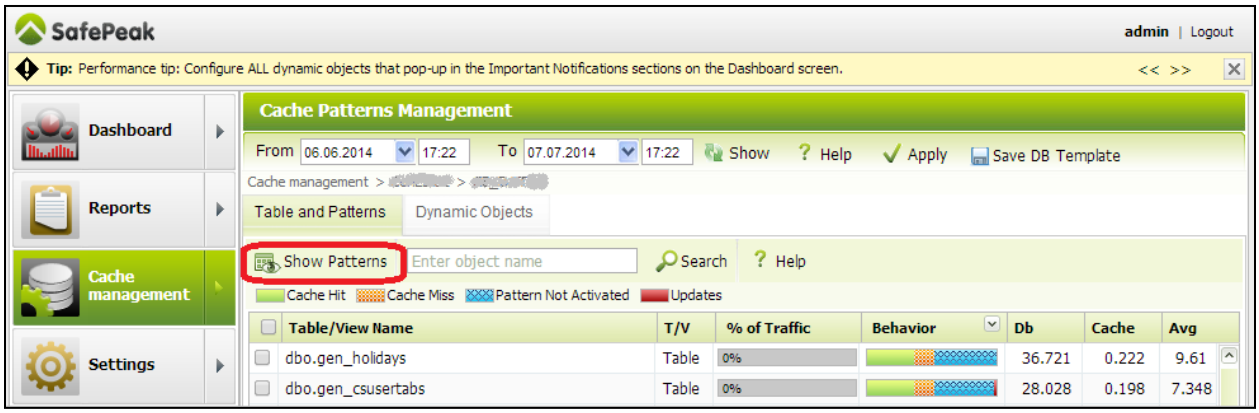
In all the next screens the default data view is the summary of last 24 hours. You can apply different ***From-To*** timeframe on the top and clicking the ***Show*** button. Additionally you can use Text and Select filters to find the objects that interest you and sort data by every column.

A tip: two most interesting ways to look at data are: (a) ***Sort by % of Traffic*** – to deal with most active queries; (b) ***Sort by DB Response time*** – to deal with most slow queries).

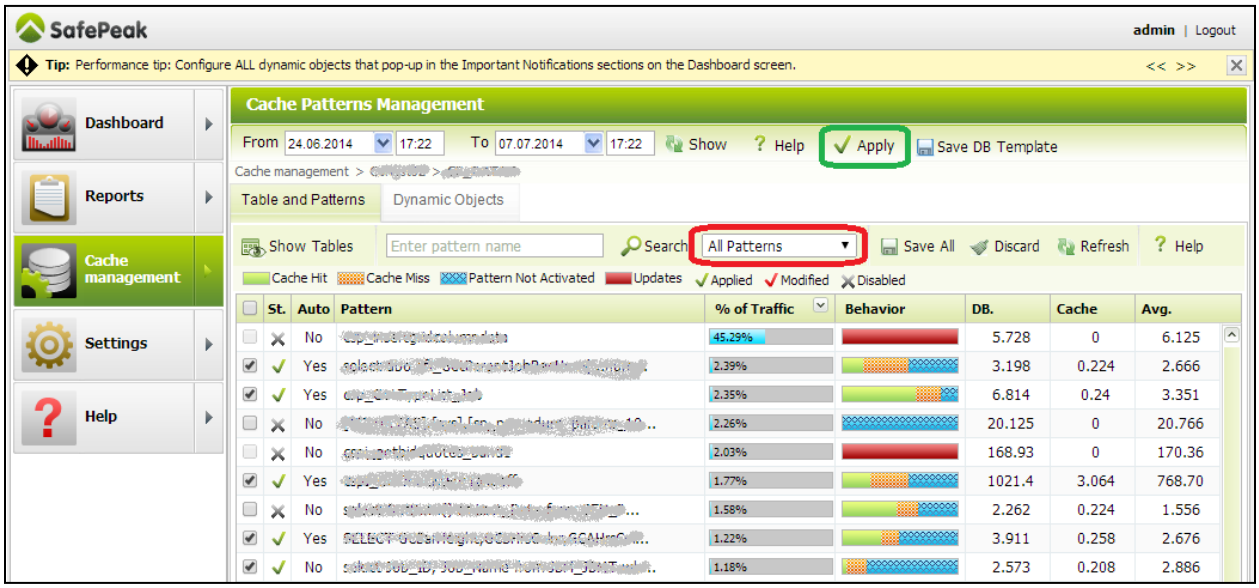
Tables and Views screen

Enter the Cache Management, Click on your Instance, Click on the database that interests you. You will arrive to the "Tables and Patterns" tab, where you will see the Tables and Views of the chosen database, together with their Usage Statistics.

The checkbox near every table and view enables to filter the next screen, SQL Patterns, showing only patterns that depend in read or write way on the checked tables and views, useful when focusing analysis on sql queries accessing a certain table, like: *show me all queries that access Customers table.*



Opening the SQL Patterns screen - Click on "*Show Patterns*".



A standard application usually creates between few hundreds to few thousands of SQL patterns. Each SQL pattern is a normalized query representation and is used for defining caching rules for such queries, as well as sharing with the SafePeak user the relevant SQL Analytics information.

The checkbox near the Pattern represents the current Active/Inactive state, while the second X or V tells you how it is set in the production state. Activating the patterns requires two steps:

1. **Enable via checkbox** – Just click on the checkbox. No need to open it and clicking save;
2. **Apply all changes** – After you finished with all your changes, click on the ***Apply*** on the top.

Step 2 – Activate the Inactive Patterns

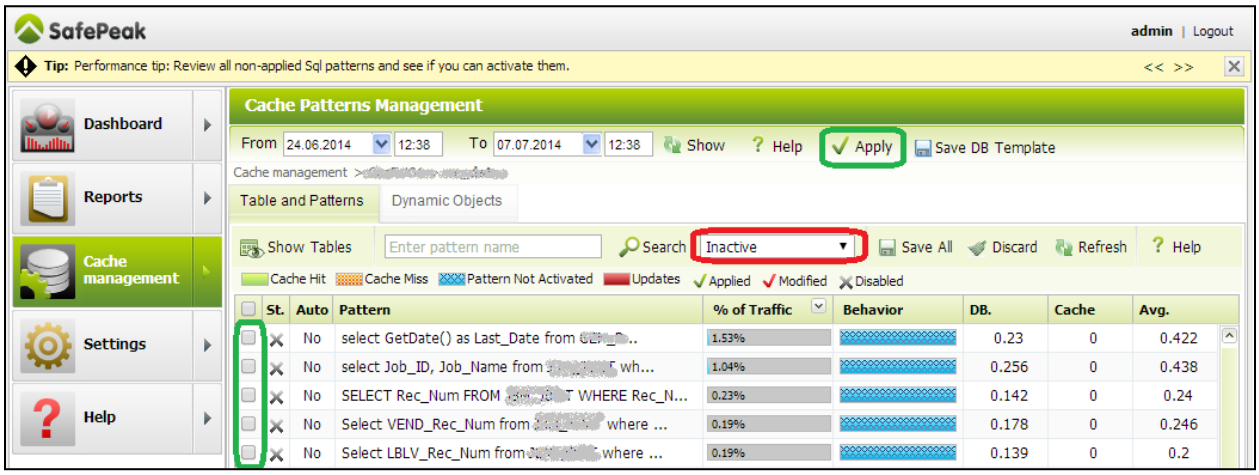
After configuring in step 1 all critical dynamic objects, it's time to review the ***Inactive SQL Patterns***, understand the reason for inactivation and activate as many as possible – for best performance.

A quick look at the **Behavior** column (in all caching and reports screens) shows in blue color how much traffic consist of SQL queries / patterns that require manual activation.

Review the Inactive patterns and activate as many as you can to get high cache hit ratio.

SQL Patterns screen – The "Inactive" queries

Choosing "***Inactive***" filter in the SQL Patterns screen will show only list of Cache=Inactive SQL Patterns:



Usually reason for pattern inactivation is a reference to Non-Deterministic Function (NDFs). Depending on the SQL logic, some patterns should be simply activated (ignoring the NDFs), others to be activated but with specific cache settings, for others caching isn't possible (with unique result-set for each call).

Review carefully all (focus on most frequent and the slowest) patterns and consider cache activation.

How to activate pattern for caching?

Simply enable the checkbox near the pattern row (like the checkboxes in the green ellipse in the above screenshot example). No need to open and save from inside – the internal Save button is only for if change caching configuration of the SQL pattern.

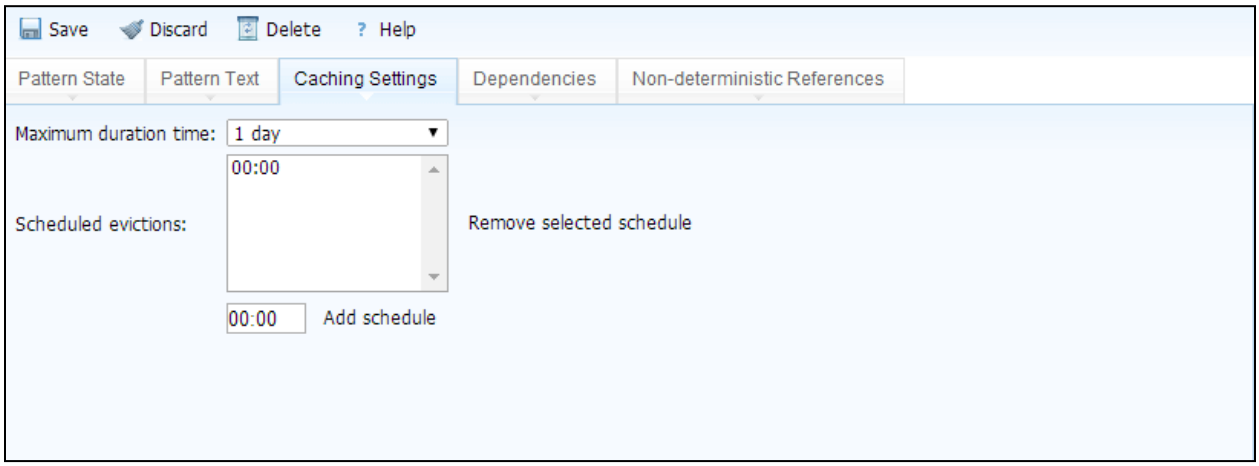
When you are finished, click Apply.

Configuration and activation of classic inactive patterns – containing a GetDate ()

The following example shows a list of customers that today is their birthday.

```
SELECT * FROM Customers
WHERE DATEPART(d, DateOfBirth) = DATEPART(d, GETDATE())  AND DATEPART(m, DateOfBirth) = DATEPART(m, GETDATE())
```

Because the query has a Non-Deterministic reference (*getdate*) it will not be automatically activated. However, the result for such query remains the same until the end of the current day. Therefore it can be cached for 1 day, with a Scheduled automated eviction from cache at the end of the day (00:00).



At the end of the review and activation process – click Apply.

Step 3 – Optimize Non-Cacheable Patterns

After you configured in step 1 all critical dynamic objects, in step 2 activated as many as possible inactive patterns, it's time to review the ***Non-cacheable patterns***.

The non-cacheable patterns can be:

1. SQL Queries that contain WRITE commands (like DMLs, DDLs),
2. SQL Queries that contain special SQL Hints that SafePeak sees by default as non-cacheable, or
3. Stored Procedures that SafePeak identified as ones having WRITE commands inside.

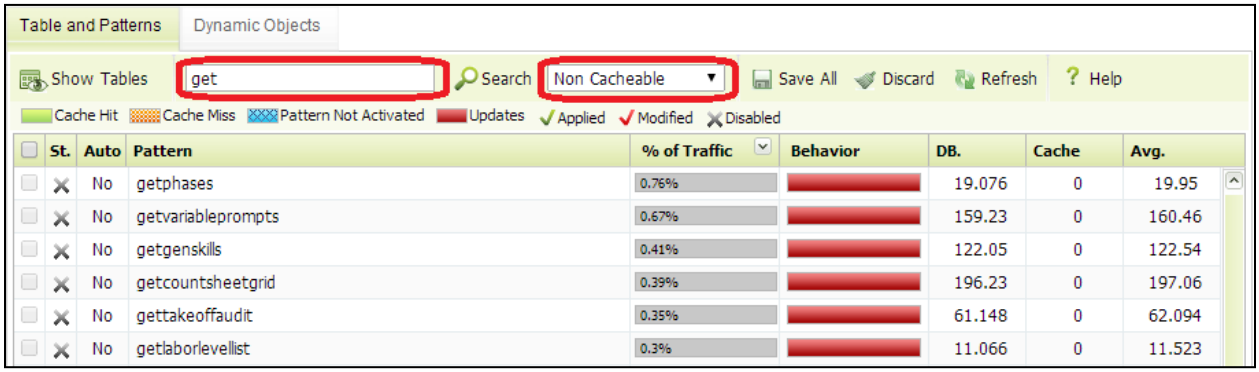
It is important to review these SQL Patterns, as some can be re-configured for cache optimization. In some cases its possible and the right optimization decision to turn a WRITE Stored Procedure into a READ one. A procedure that for has a non-critical WRITE Dependency setting is affecting the performance in two ways:

a. Since they are not cached, SafePeak does not improve (*directly) their performance.
b. Furthermore, on every call of the stored procedure SafePeak will evict cache items based on the Write Dependencies of the procedure.

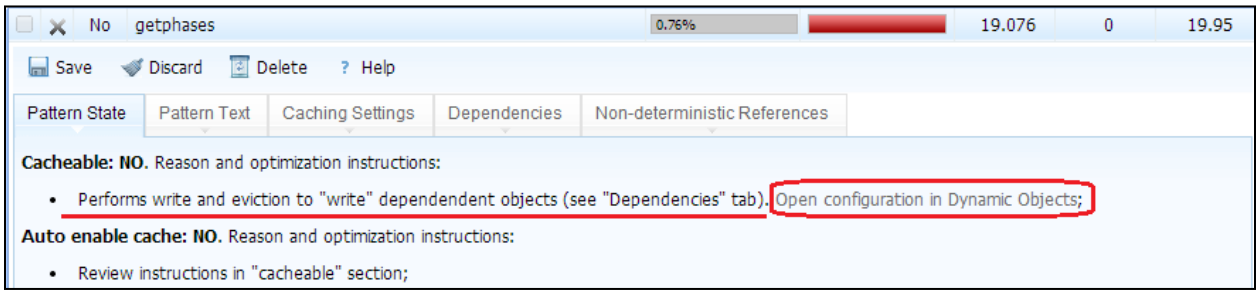
* Performance of non-cached queries is getting an indirect improvement, since with SafePeak there is less load, locks, blocks etc on the SQL Server.

Example – Search for Non-Cacheable stored procedures with "get" in the text:

Review your non-cacheable patterns and see if some have in similar concept. For example, SELECTS with special session settings or hints, or instead stored procedures that you think have a "READ" nature.



Opening a pattern shows reasons why cache was not possible and provides tuning options.



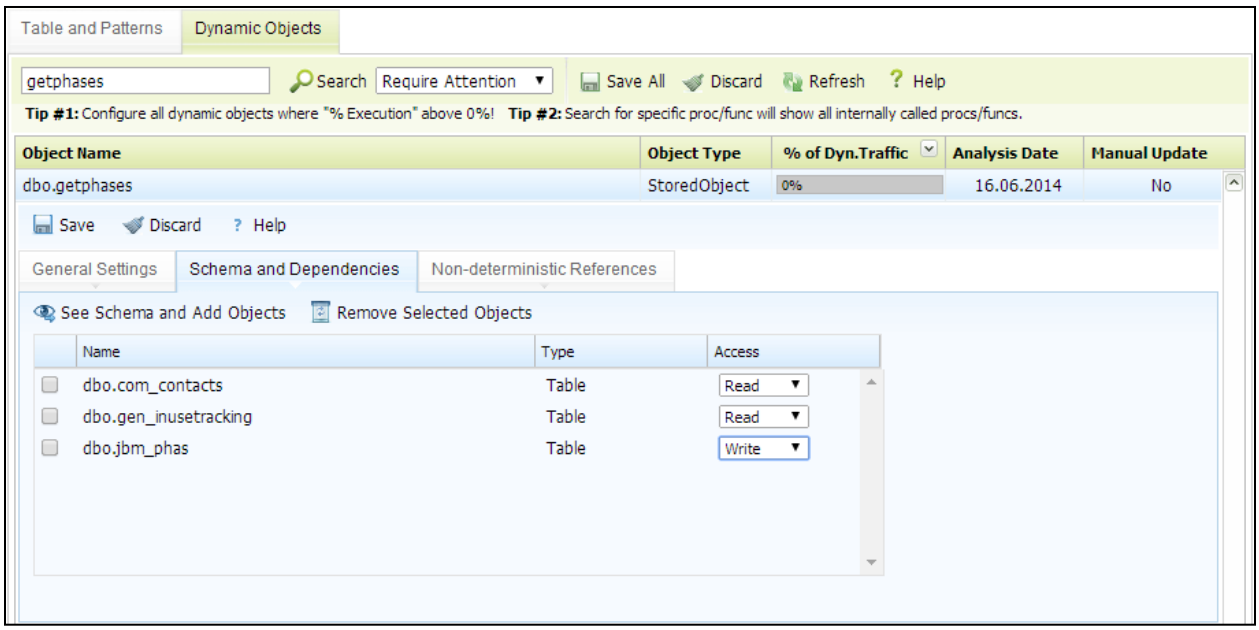
To review a procedure details and possibly convert it to a cacheable procedure is done by clicking the "*Open configuration in Dynamic Objects*" link. See section below: "Identifying ignorable writes".

Another possible reason for a READ pattern with SELECT Query to become non-cacheable is due to certain SQL Hints. In some of specific cases this message appears: "*Make pattern cacheable. Conversion to cacheable pattern is possible*". Clicking on the link enables switching the pattern to cacheable state.

Object configuration optimization – Identifying ignorable writes

Reviewing and overriding settings of stored procedures, defining the kind of Dependencies it has, is performed inside the Dynamic Objects screen. Searching for an object using object name enables to reach any type of object and override its configuration (procedure/user-function/trigger/view).

For example, "`getPhases`" stored procedure that we identified in the Non-cacheable Patterns list and decided to investigate further for possible conversion to a cacheable state, shows these dependencies:



Of course the WRITE dependency to table `Jbm_Phas` is the one that prevents us from caching. If we suspect that this object is focused on Reads and we want to cache it, we can check the object definition (in "***See Schema and Add Objects***" or in SSMS) and understand why there is a WRITE. Example:

```
CREATE PROCEDURE [dbo].[getPhases]
        @userId int,
        @PrepData int
AS
        -- CHECK if data preparation is required during the first execution time:
        IF EXISTS (        SELECT 1 FROM Jbm_Phas Phases WHERE Phases.UserId=@userId and
Phases.IsDataPrepared<1)
                BEGIN
                        -- Prepare the data in the Jbm_Phas table
                        UPDATE Jbm_Phas
                        SET Jbm_Phas.PrepData=@PrepData, Jbm_Phas.IsDataPrepared=1
                        WHERE Phases.UserId=@userId;
                END
        -- RUN the main report joining several tables
        SELECT *
        FROM Jbm_Phas Phases
        INNER JOIN gen_inusertracking Iut ON Iut.userId=Phases.UserId
        INNER JOIN com_Contacts Con ON Con.ContactId=Iut.ContactId
        WHERE Phases.UserId=@userId
GO
```

In the beginning this procedure prepares some data in the `Jbm_Phas` table but only for the first time it is executed (there comes the WRITE), and in the following calls it will be just READING data.

Since this procedure depends on `Jbm_Phas` table (and updates itself), we can perform a change to `Jbm_Phas` Dependency to **READ** access. The `getPhases` will become cacheable, but sensitive to data changes in `Jbm_Phas` table, with auto eviction of `getPhases` from cache. After change, `getPhases` will not be in cache and will be sent to the Database, which will make the necessary data preparations. The returned result will be inserted into the SafePeak in-memory cache and for executions 2^{nd} -3^{rd} -...-10^{th} the data will be returned from cache.

Step 4 – Defining semi-dynamic caching patterns

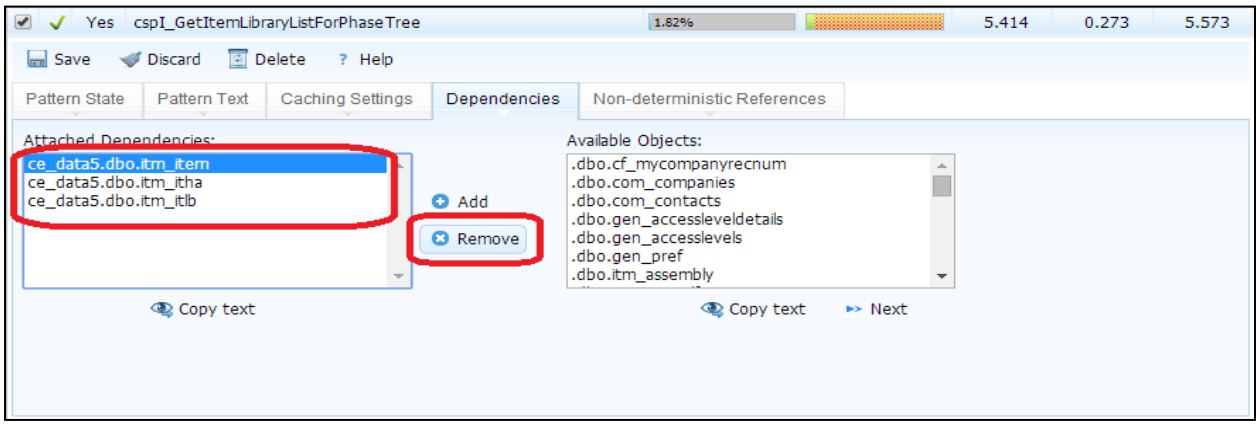
Go back to Active Patterns screen. Look for activated SQL Patterns, with high traffic (or slow performance), that have low level of cache hit ratio. Look for SQL Patterns the Behavior column value, where the orange part takes the majority of the width. Example:

The pattern above is activated 1.82% of all calls to the current database, but Cache Hit % is very low (green part). In many cases when the administrator is closely familiar with the application he can decide to improve pattern cache hit ratio by reducing pattern sensitivity to updates and limiting cache time.

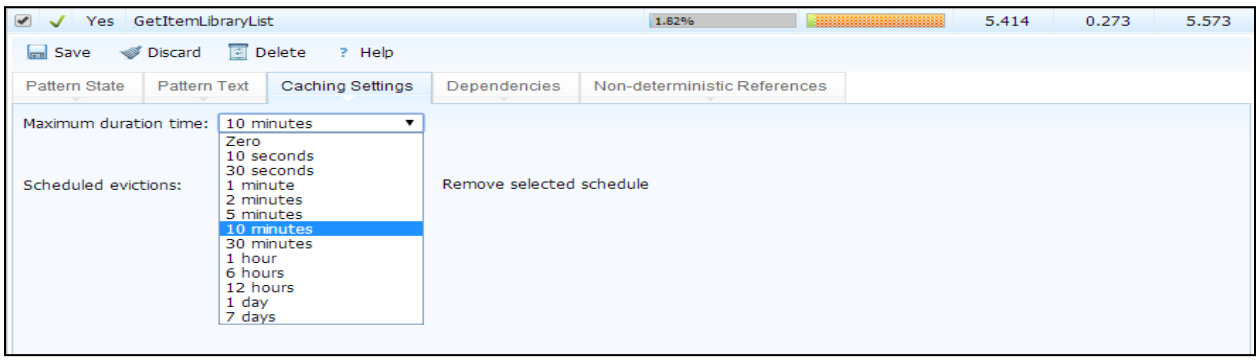
The above `GetItemLibraryList` procedure is called many times, but is not hitting cache most of the time probably due to frequent WRITEs to its dependent tables. If the administrator knows the nature of the application screen (or webpage) calling this query and the nature of the query, he can decide that it may be OK to return a slightly out-dated result.

For example, the "News Feed" in Facebook can be cached for few minutes. Even if the data changes, it is OK that the user will receive updates only after few minutes, no one will notice. A classic Static Cache.

By default SafePeak implements a fully Dynamic Caching, sensitive to writes for every pattern. You can reduce pattern sensitivity to writes by ***removing*** its dependencies to certain or all tables:



And then configuring the cache "maximum duration time" (time to live) to X minutes or hours:



Step 5 – Using the Caching Warm-up utility

Part of SafePeak package is a Caching Warm-up utility. To use and activate the Cache Warm-up utility a manual activation and configuration is required. The utility is a separate library that enables frequent scheduled executions of list of SQL queries and Stored Procedures calls based on XML file. The goal of this utility is to give another tool that will "pre-fetch" certain important queries into SafePeak Cache so when the real users will request them it will be already in-memory.

Caching Warm-up: Configuring queries list

Using Notepad++ (or similar tool), open the Cache Warm-up configuration file:

\Instances\[Your_Instance_Name SafePeakInstanceWarmUp.xml

The configuration consists of 3 sections:

1) **General configuration**: including ThreadsNumber, CommandTimeOut, ConnectionString/s
2) **Sql batch statements/queries:** SqlStatements
3) **Stored Procedures statement list:** to be executed in RPC way, the list in SqlRPCStatements

1. General configuration

Configure the general settings. Notice that the **Data Source / Server** are the local SafePeak instance, which is running on the same machine therefore it will be=**localhost,[SafePeak_Port]**.

2. Configuring the SqlStatements

Setting up the Sql-Statements is very simple. For every database (or different statement) define list of queries required to be executed:

```
<Statement>
    <Database>master</Database>
    <SessionSettings/>
    <Provider>SQLClient</Provider>
  - <SqlCommands>
        <Command>Select * from T1 where col1=1;</Command>
        <Command>Select * from T1 where col1=2;</Command>
        <Command>Select * from T2;</Command>
        <Command>exec searchItems @Category=1, @max_price=0, @items_per_page=10, @page_number=1, @searchFilter=N'samsung'</Command>
    </SqlCommands>
</Statement>
```

The SQL Statements can be stored procedures calls that are executed as batch queries

Notice that the queries settings are case-sensitive, every extra Tab, Space or Enter counts. Best strategy is open the SQL Pattern in SafePeak Dashboard, copy it and then place the values (numbers/strings) instead of the normalized variables definition in the pattern.

3. Configuring the SqlRPCStatements

Configuring the SQL RPC Statements, stored procedures with specific parameters values, can be more complicated. There are two ways to define the calls – chose what is more convenient for you.

1. **A easier way:** In section SqlRPCCommands define Command's (params come without spaces!):
   exec searchItems @Category=1,@items_per_page=10,@page_number=1,@searchFilter=N'samsung'
   The query is case-sensitive and the order of parameters-with-values matters. This definition process was designed for easy copy of procedures calls from a SQL Server Profiler recording.

2. **A more complex way:** Separately defining Procedure Name (SqlCommand), Ordered List of Parameters (ParamNames), Parameters' Types (ParamTypes), A separation character (ParamsValuesSeperationChar) and A sets of Parameters-Values (ParamValuesSet).

Caching Warm-up: Activation

Configure using Notepad++ (or similar tool) the Scheduler configuration file (in SafePeak directory):

\Instances\[Your_Instance_Name SchedulerSerialization.xml

Find Section: <activityType>**CallSafePeakLibrary**</activityType>

Change Active state to true: <Active>**true**</Active>

Save the file.

Open Windows Services, find service "***SafePeak Scheduler [Your_Instance_Name]***" and restart it.

To check log of the executions, review the **logInstanceWarmUp.log** and the **Scheduler.log** files in "***Instance**[Your_Instance_Name]**Logs***".