As part of the 6.0 release of WordPress, the new performance team has been hard at work, to improve the performance of term queries. There are many term queries on the average page load and by improving these, it improves the performance of WordPress in general.

Improvement to term query caching.

Queries run by WP_Term_Query have been cached since [version number]. The way these caches are primed and handled has been improved in WordPress 6.0.

Removing to cache limit

Prior to WordPress 6.0, term query caches were limited to a 24 hour period for those using persistent object caching. This limitation is now removed, so if caches are not invalidated, it means that term query should cache much longer. For inactive sites or say overnight, caches should remain primed which should improve site performance.

Term query cache only caches the term ID

Term query caches have been changed instead of caching the whole term object to now only cache the term IDs. This means that the value stored in cache will be much smaller (in terms of memory) and will not fill up memory in session or persistent object cache. Once all the IDs for the terms are loaded, the _prime_term_cache function is called. This loads into memory terms that are not already in cache. If the term is already in memory, then it is not loaded again, which is a performance benefit. On an average page load, a term may be requested multiple times, like the case of a tag archive page. Early in the page load, get_term_by, will prime the term cache and all other calls, such as get_the_terms, will have the term already primed in page memory. This results, in most cases in fewer and smaller queries that are run against the term table.

Improved term query cache key generation

Previously, similar term queries that have similar cache keys would result in basically the same query being run twice on a single page load. As now all queries only get the term ID and store it in cache (see above) the cache is the exact same. For example, you create a call to get_terms where you request all categories and return only the field slug. If you do the same query and request only the name, then this would hit the same cache. Another improvement is the handling of parameters that can be passed to WP_Term_Query that can be ambiguous. Fields like slug that can be either a string or an array and now converted to always be an array, meaning that the likelihood of reusing a cache is higher as the cache key is the same, no matter which type of parameter that is passed.

Improve performance for navigation menu items

Convert wp_get_nav_menu_items to use a taxonomy query

Replace usage of get_objects_in_term function with a simple taxonomy query. This replacement changes from making two queries to get the menu items, converting them to use one simple query. This saves one query for each menu requested and adds consistency.

Prime all term and posts caches in wp_get_nav_menu_items

The wp_get_nav_menu_items function now calls _prime_term_cache and _prime_post_cache for all objects linked to menu items. If a menu contains a list of categories and pages, all the related objects are now primed in two cache calls (one for terms and one for posts). This will result in many fewer requests to the database and cache.

Convert term_exists to use get_terms

In [commit_number] the function term_exists was converted to use get_terms (WP_Term_Query) internally replacing raw uncached database queries. This function was one of the last places to perform raw queries to the terms table in the database. Using the get_terms function has a number of key benefits, including:

- Consistency with other core functions like get_term_by
- The ability to filter the results.
- Results of get_terms are cached.

Term_exists, is designed for backend use and is mostly used in core functions designed to write data to the term table. However, term_exists can and is used by some theme and plugin developers. This results in raw uncachable and unfilterable queries to be run on the front end of your site.

Now that term_exists is cached, custom import / migration tools may need to check if they correctly cache invalidation terms. If your importation code is using core functions like wp_insert_term, then there is no need to do anything, as core does its own cache invalidation for you. However, if you are writing data to the term table manually, then you may need to call the clean_term_cache function.

For those that need to ensure that term_exists is getting an uncached result, there are two ways in which a developer can do this.

1. Using the new `term_exists_default_query_args filter.`

   Using the new filter like so

   ```
   $callback = function ( $args ) {
       $args['cache_domain'] = microtime();

};
add_filter( 'term_exists_default_query_args',  $callback );
$check = term_exists( 123, 'category' );
remove_filter( 'term_exists_default_query_args',  $callback );
```

2. Using `wp_suspend_cache_invalidation`

   ```
   wp_suspend_cache_invalidation( true );
   $check = term_exists( 123, 'category' );
   wp_suspend_cache_invalidation( false );
   ```

Add a limit to taxonomy queries

The WP_Tax_Query class is used in WP_Query to limit queries by term. Under the hood, WP_Tax_Query uses WP_Term_Query which means that WP_Tax_Query will get the benefits of cache improvements, documented above. The WP_Tax_Query run, which transforms term slugs / names into term IDs to be queried, now has a limit added to it. This query limit improves the performance of the query. But it also improves the likelihood of an existing cache query to continue to exist in the object cache. For example, a standard tag archive calls get_term_by and primes the cache. Meaning, by the time it gets to the WP_Tax_Query, that query is being loaded from cache. This removes one query per tag archive page.