Connection Pooling and User-based Security Models

Posted At : March 14, 2005 4:04 PM | Posted By : Steven Erat
Related Categories: Java, ColdFusion

This blog entry will attempt to explain the effect on datasource connection pooling in ColdFusion MX when using unique username and password combinations passed through the CFQuery tag, often a requirement for databases configured for User-based Security. In this case the username/password combination often comes directly from end user logins on the application.

Normal Usage and Starting a Connection Pool
To provide a little background of normal circumstances, most ColdFusion users configure JDBC datasources to use datasource connection pooling by enabling the check box for "Maintain Connections" in the datasource detail page within the ColdFusion Administrator. This page accepts a single username/password combination when setting up the datasource. Connection pooling will cause new database connections created for that datasource to be added to a datasource connection pool for reuse later by another request.

Under normal circumstances a page request may contain a query where the datasource name is specified in the datasource attribute in the CFQUERY tag, but the username and password are not. In this case the login credentials for the query are taken from the common datasource properties in the datasource definition, and used for the initial database login for a new connection. Assuming this was the first request for a connection to that datasource, the page request would have created a new datasource connection, and if any other CFQUERY tags existed on the page for the same datasource, then the request would hold on to that new connection for use in the latter queries. At the end of the page request, that new connection will be left open, but returned to a connection pool instead of being immediately closed. If a new page request is made requiring a query to the same datasource, again where the username and password are taken from the CFAdmin datasource properties, the request will attempt to check out an existing connection from the connection pool. That second request will run all the queries on the page for that datasource/username/password combination, and then return the datasource connection back to the pool at the end of the request.

Growing the Connection Pool
Now imagine that after those first two requests have completed, there are two additional page requests having the same queries that are run at the same time in ColdFusion server. Assuming they run concurrently, they will both attempt to check out the single connection in the pool created during the first request earlier. One of these new requests will succeed and it will get the connection from the pool. The last page request will then check the connection pool and see that the existing connection in the pool is currently checked out or busy. Since there are no more connections in the pool and assuming that there is no limit to the number of connections in that pool, the last request will create a new datasource connection and use it for all queries during that request having the same datasource/username/password combination. When these two last page requests are completed, they will release the datasource connections and return two connections to the pool. The competition for datasource connections caused one connection to be reused and one new connection to be added to the pool.

Limiting Connection Pool Size
In such manner, the datasource connection pool will be grown. If a limit on the number of connections is established in the datasource detail page by enabling "Enable a Limit of N Connections", then the pool could grow until it reaches that limit, assuming that there continues to be a demand for the datasource connections and that competition exists where sometimes new connections are created when all other connections in the pool are checked out and busy. Once the pool size has reached its limit, then new requests for a connection from the pool must wait since the current ones are busy and new ones aren't permitted. That request will wait for an available connection from the pool until it reaches the "Login Timeout" setting in the datasource detail page where the page request will end with a timeout error.

Pruning the Connection Pool
This maxed out connection pool will shrink according to the Timeout and Interval properties in the datasource definition in the CF Admin. The Timeout setting in the datasource detail page (not to be confused with the datasource Login Timeout) reflects how old a used or idle connection must be before ColdFusion can garbage collect it and remove it from the pool, and the default setting is 20 minutes. The frequency that ColdFusion checks this is set by the Interval value, where the default value is 7 minutes. The threads that actually do the collecting are often referred to as skimmer threads because the skim the aged datasources off the pool. In the default, skimmer threads will check every 7 minutes for idle dsn connections that have been unused for 20 minutes or more. Each time it checks, all matching datasource connections will be collected, closed, and removed from the pool. This is different from earlier ColdFusion MX 6.0 settings that could be configured with the "shrink-by" value in jrun-resources.xml. The CFMX 6.1 and 7 "shrink-by" value is ~~effectively infinite since all matching idle connections will be removed~~ is fixed at 5 connections removed per interval, although the literal shrink-by setting is not accessible or configurable through any XML config file. In this way, the dsn connection pool size is regulated and pruned when needed.

Using a Different Username/Password for the Same Datasource
Confusion arises when unique username and password combinations are passed directly in the CFQuery tag rather than relying upon those in the CF Admin datasource detail page. In this case, a unique username and password combination in CFQUERY will cause a new connection to be created and added to the common dsn connection pool for that datasource name. The datasource connection pool will then contain a pool of connections having at least 2 sets of username/password combinations.

An Example of User-based Security
As an example, lets say the CF Admin datasource is called Bedrock, and the dsn detail has the user Fred configured as the username. Then imagine that there are pages having CFQuery tags for datasource="Bedrock" but not having the username or password attributes specified, and imagine that there are one or more other pages having a CFQuery tag with the user Barney.

If some page requests have already been made causing the Bedrock datasource connection pool to have a couple Fred connections, the first execution of a CFQuery tag with username="Barney" will request to checkout a Bedrock connection where the username is Barney. No matter how many Fred connections are in the Bedrock pool,

only Barney connections can be returned to the query with Barney as the user. Since the Bedrock pool only contains Fred connections, the query for user Barney will not get to reuse an existing connection at first, behaving as though it were using its own Barney pool as a subset of the Bedrock pool. The CFQuery tag will open a new datasource connection for the Barney user and at page end it will return the connection to the Bedrock pool. The next request for a datasource connection from the Bedrock pool by a Barney query will get to reuse the existing Barney connection.

User-based Security
The lesson here is that one connection pool can exist per datasource definition in the CF Admin where that pool may contain unique subsets of connections based on username/password combinations which are passed in the CFQuery tag in the code. One common reason for doing this is when the database is configured for User-based Security, and end user login credentials may be passed directly through as the query login credentials.

One Datasource Setting Recommendation
In this scenario, one recommended configuration would be **to not enable a limit on the connection pool** size since every user's connection will get added to the pool and usable only by that user, not anyone else, so the list of open database connections could be as large as the list of currently logged on end users. To prune the connection pool size back down as quickly as possible, set the dsn Timeout low, perhaps just 2 or 3 minutes, and set the Interval at 1. This way the skimmer thread will clean the pool of 5 idle connections every minute, removing those connections not active for 2 or 3 minutes.

Using connection pooling without a pool size limit might be a good choice if you have lots of code pages (50-100+) containing user-based query credentials, to save all the overhead of having to establish a connection each time. The pool size could grow quite large in this case, but connection reuse will help speed up the user experience.

An Exception to the Recommendation
However, if you were maintaining a connection pool with user-based seucrity model and you did want **to enable a connection pool limit** (which you really don't), you wouldn't want to enforce a limit smaller than the number of users that might be logged on. If a limit on the Bedrock pool existed and set to 4 for example, and was currently filled with connections having usernames of Fred, Wilma, Barney, Betty, then a request for a connection having username Dino would have to wait for one of the others to timeout and get collected even if there were no actively used connections. Dino couldn't reuse Fred or Barney's connections. He(?) would need his own connection from the Bedrock pool.

An Alternate Recommendation for Datasource Settings Configuration
Another possible configuration useful with User-based Security models would be **to not enable connection pooling** at all by unchecking the setting for "Maintain Connections" in the dsn detail page. This would cause every page request having a unique set of credentials to open a datasource connection upon the first query execution, reuse that connection for the duration of the page on other queries having the same datasource name and login user. At page end, the connection will be closed. This will cause increased overhead from opening and closing connections often, but is useful when having to limit the total number of connections open to the database in a User-based

Security model. Since connection pooling isn't used, there's no need to worry about the Timeout and Interval settings.

This alternative might be a good choice for an application having not many code pages (say < 50) with user-based security with the query credentials. New connections are opened and closed for every user for every page request having such a query, causing some overhead, and that's why you wouldn't want to do this if you have a large number of pages having these user-based queries. This configuration would be helpful if you were concerned about limiting the overall number of connections between the ColdFusion server and the database.

How Does the Client Variable Storage Datasource Fit In?
Remember that when using a single datasource for the sole purpose of client variable storage, and using Client Management in the CF Application Framework, each page request within the application will have one connection open to the client variable datasource independent of the other application datasources. If the option to disable Client variable updates for Last Visited and Hitcount is set, then a client variable datasource will be opened only if any client variable data has been changed or when pre-existing client variable data is referenced. A client variable dataource relies upon a single login credential, so connection pooling could be set and a limit could be enabled on the pool size where the limit is roughly equal to the CFAdmin Simultaneous Requests value, plus a couple more to be safe.