PostgreSQL Bulk Insert

Here’s a means to insert multiple records in a single insert call, which is about 40 times faster than performing an explicit insert for each row.

We will leverage the BeginBinaryImport method on the database connection class of the NPGSQL library.

To make this easy, we’ve created a wrapper call in our Postgres_DAL class of: OGA.Postgres.DAL.

The method to use, is called, PerformBulkBinaryImport.

NOTE: Since this method converts all strings to binary, there is no need to escape or sanitize any string data to prevent SQL injection attacks.
As well, this method also preserves format of any JSON structs, without having to escape quotes and apostrophes.

This method accepts two arguments:

- copyargs
- callback

CopyArgs Parameter

CopyArgs is a string that is formatted as a PostgreSQL COPY statement.
It includes the target table name, a list of columns (in parentheses), and how to receive the data.

For example, here’s the CopyArgs string for bulk inserting log messages into a table called, tbl_LogEntry”

```
string copyargs = "COPY public.\"tbl_LogEntry\" " +
                  "(\"time\", level, host, process, service, runtimeid, callsite, " +
                  "threadname, message, exc_type, exc_message, exc_stacktrace) " +
                  "FROM STDIN (FORMAT BINARY)";
```

The above string includes the target table, ‘public.tbl_LogEntry’.
It includes the ordered list of columns to expect.
And, it includes a FROM clause to say how to receive data (from STDIN and in binary format).

Callback Parameter

The second element is the callback.

This parameter must be set to a method, anonymous function, or lambda, that includes logic write the binary output.

Within that parameterized method, you will use the given ‘writer' object, to create rows, and add columns to each one (following the order in the CopyArgs parameter).

The callback should look like this:

```
var callback = Action<sdfsd> (writer) =>
{
  // Start a new row...
  writer.StartRow();
  writer.Write(etime, NpgsqlDbType.TimestampTz);
  writer.Write(m.host, NpgsqlDbType.Varchar);
  ...
  // Start a new row...
  writer.StartRow();
  writer.Write(etime, NpgsqlDbType.TimestampTz);
  writer.Write(m.host, NpgsqlDbType.Varchar);
}
```

Once you have created these two parameters, pass them to the PerformBulkBinaryImport method, and it will perform the bulk insert of your source data, as defined by your writer callback.

Here’s a working example of how to perform a bulk insert of multiple log entries into a log table, using the above method (taken from the PostGres Log Sink Manager Service in OGA.LogSink.Service):

```
// Declare the setup string for the binary importer...
string import_setup = $"COPY public.\"tbl_LogEntry\" (\"time\", level, host, process, service, runtimeid, callsite, threadname, message, exc_type, exc_message, exc_stacktrace) " +
                      "FROM STDIN (FORMAT BINARY)";

// Declare the row digester callback...
var res = this._dal.PerformBulkBinaryImport(import_setup, (writer) =>
{
    // Iterate entries, and digest each one...
    foreach (var m in msg)
    {
        try
        {
            if (m == null)
                continue;

            writer.StartRow();

            if(!DateTime.TryParse(m.time, out var etime))
            {
                etime = DateTime.Now;
            }

            // Set the UTC flag in the received timestamp...
            etime = DateTime.SpecifyKind(etime, DateTimeKind.Utc);

            writer.Write(etime, NpgsqlDbType.TimestampTz);

            writer.Write(m.level, NpgsqlDbType.Varchar);

            writer.Write(m.host, NpgsqlDbType.Varchar);
            writer.Write(m.process, NpgsqlDbType.Varchar);
            writer.Write(m.service, NpgsqlDbType.Varchar);

            writer.Write(m.RuntimeID, NpgsqlDbType.Varchar);
            writer.Write(m.callsite, NpgsqlDbType.Varchar);
            writer.Write(m.threadname, NpgsqlDbType.Varchar);

            writer.Write(m.message, NpgsqlDbType.Varchar);

            writer.Write(m.exception?.type ?? "", NpgsqlDbType.Varchar);
            writer.Write(m.exception?.message ?? "", NpgsqlDbType.Varchar);
            writer.Write(m.exception?.stacktrace ?? "", NpgsqlDbType.Varchar);
        }
        catch(Exception e)
        {
            int x = 0;
        }
    }
});
if(res != 1)
{
    // Failed to store log messages.
```

Revision #1
Created 7 June 2025 23:52:26 by glwhite
Updated 7 June 2025 23:54:55 by glwhite