Managing database connections

MySQL (https://cloud.google.com/sql/docs/mysql/manage-connections) | PostgreSQL (https://cloud.google.com/sql/docs/postgres/manage-connections) | **SQL Server**

Beta

This feature is in a pre-release state and might change or have limited support. For more information, see the product launch stages (https://cloud.google.com/products/#product-launch-stages).

This page provides best practices and language-specific code samples to help you create applications that use Cloud SQL database connections effectively.

These samples are excerpts from a complete App Engine application available to you on GitHub. Learn more (#app-links).

Connection pools

A connection pool is a cache of database connections that are shared and reused to improve connection latency and performance. When your application needs a database connection, it borrows one from its pool temporarily; when the application is finished with the connection, it returns the connection to the pool, where it can be reused the next time the application needs a database connection.

C#

cloud-sql/sql-server/Startup.cs (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/master/cloud-sql/sql-server/Startup.cs)

GLECLOUDPLATFORM/DOTNET-DOCS-SAMPLES/BLOB/MASTER/CLOUD-SQL/SQL-SERVER/STARTUP.CS)

```
var connectionString = new SqlConnectionStringBuilder(
    Configuration["CloudSql:ConnectionString"])
```

```
// ConnectionString set in appsetings.json formatted as:
// "User Id=sqlserver;Password=;Server=cloudsql;Database=votes;"
{
    // Connecting to a local proxy that does not support ssl.
    Encrypt = false,
};
connectionString.Pooling = true;
// ...
DbConnection connection =
    new SqlConnection(connectionString.ConnectionString);
```

Opening and closing connections

When you use a connection pool, you must open and close connections properly, so that your connections are always returned to the pool when you are done with them. Unreturned or "leaked" connections are not reused, which wastes resources and can cause performance bottlenecks for your application.

C#

cloud-sql/sql-server/Controllers/HomeController.cs (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/master/cloud-sql/sql-server/Controllers/HomeController.cs)

NET-DOCS-SAMPLES/BLOB/MASTER/CLOUD-SQL/SQL-SERVER/CONTROLLERS/HOMECONTROLLER.CS)

```
insertTimestamp = DateTime.Now;
try
{
    // Insert a vote for SPACE or TAB with a timestamp.
    using (var insertVoteCommand = _connection.CreateCommand())
    {
        insertVoteCommand.CommandText =
            @"INSERT INTO votes (candidate, time_cast) VALUES (@candidate, @time_c
        var candidate = insertVoteCommand.CreateParameter();
        candidate.ParameterName = "@candidate";
        candidate.DbType = DbType.String;
        candidate.Value = team;
        insertVoteCommand.Parameters.Add(candidate);
        var timeCast = insertVoteCommand.CreateParameter();
```

```
        timeCast.ParameterName = "@time_cast";
        timeCast.DbType = DbType.DateTime;
        timeCast.Value = insertTimestamp;
        insertVoteCommand.Parameters.Add(timeCast);
        await insertVoteCommand.ExecuteNonQueryAsync();
    }
    return Content($"Vote successfully cast for '{team}' at time {insertTimestamp}
}
catch (Exception ex)
{
    // If something goes wrong, handle the error in this
    // section. This might involve retrying or adjusting
    // parameters depending on the situation.
    return StatusCode((int)HttpStatusCode.InternalServerError, ex);
}
```

Connection count

Every database connection uses client and server-side resources. In addition, Cloud SQL imposes overall connection limits that cannot be exceeded. Creating and using fewer connections reduces overhead and helps you stay under the connection limit.

C#

cloud-sql/sql-server/Startup.cs (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/master/cloud-sql/sql-server/Startup.cs)

GLECLOUDPLATFORM/DOTNET-DOCS-SAMPLES/BLOB/MASTER/CLOUD-SQL/SQL-SERVER/STARTUP.CS)

```
// MaximumPoolSize sets maximum number of connections allowed in the pool.
connectionString.MaxPoolSize = 5;
// MinimumPoolSize sets the minimum number of connections in the pool.
connectionString.MinPoolSize = 0;
```

Exponential backoff

If your application attempts to connect to the database and does not succeed, the database could be temporarily unavailable. In this case, sending too many simultaneous connection requests might waste additional database resources and increase the time needed to recover. Using exponential backoff prevents your application from sending an unhealthy number of connection requests when it can't connect to the database.

C#

cloud-sql/sql-server/Startup.cs (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/master/cloud-sql/sql-server/Startup.cs)

GLECLOUDPLATFORM/DOTNET-DOCS-SAMPLES/BLOB/MASTER/CLOUD-SQL/SQL-SERVER/STARTUP.CS)

```
var connection = Policy
    .HandleResult<DbConnection>(conn => conn.State != ConnectionState.Open)
    .WaitAndRetry(new[]
    {
        TimeSpan.FromSeconds(1),
        TimeSpan.FromSeconds(2),
        TimeSpan.FromSeconds(5)
    }, (result, timeSpan, retryCount, context) =>
    {
        // Log any warnings here.
    })
    .Execute(() => NewSqlServerConnection());
```

Connection timeout

There are many reasons why a connection attempt might not succeed. Network communication is never guaranteed, and the database might be temporarily unable to respond. Your application should handle broken or unsuccessful connections gracefully.

C#

cloud-sql/sql-server/Startup.cs (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/master/cloud-sql/sql-server/Startup.cs)

GLECLOUDPLATFORM/DOTNET-DOCS-SAMPLES/BLOB/MASTER/CLOUD-SQL/SQL-SERVER/STARTUP.CS)

```
// ConnectionTimeout sets the time to wait (in seconds) while
// trying to establish a connection before terminating the attempt.
connectionString.ConnectTimeout = 15;
```

Connection duration

Limiting a connection's lifetime can help prevent abandoned connections from accumulating. You can use the connection pool to limit your connection lifetimes.

C#

cloud-sql/sql-server/Startup.cs (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/master/cloud-sql/sql-server/Startup.cs)

GLECLOUDPLATFORM/DOTNET-DOCS-SAMPLES/BLOB/MASTER/CLOUD-SQL/SQL-SERVER/STARTUP.CS)

```
// ADO.NET connection pooler removes a connection
// from the pool after it's been idle for approximately
// 4-8 minutes, or if the pooler detects that the
// connection with the server no longer exists.
```

View the complete application

To see the complete application, click the link below.

C#

View the complete application (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/master/cloud-sql/sql-server/README.md)
for the C# programming language.

What's next

- Learn more about Private IP (https://cloud.google.com/sql/docs/sqlserver/private-ip).
- Learn about quotas and limits (https://cloud.google.com/sql/docs/sqlserver/quotas) for Cloud SQL and App Engine.
- Learn about best practices (https://cloud.google.com/sql/docs/sqlserver/best-practices) for working with Cloud SQL.

Last updated December 5, 2019.