Managing database connections

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

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.

PYTHON JAVA NODE.JS C#

cloud-sql/mysql/sqlalchemy/main.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/cloud-sql/mysql/sqlalchemy/main.py)

)UDPLATFORM/PYTHON-DOCS-SAMPLES/BLOB/MASTER/CLOUD-SQL/MYSQL/SQLALCHEMY/MAIN.PY)

```
# The SQLAlchemy engine will help manage interactions, including automatically
# managing a pool of connections to your database
db = sqlalchemy.create_engine(
    # Equivalent URL:
    # mysql+pymysql://<db_user>:<db_pass>@/<db_name>?unix_socket=/cloudsql/<cloud_
    sqlalchemy.engine.url.URL(
        drivername="mysql+pymysql",
        username=db_user,
```

```
        password=db_pass,
        database=db_name,
        query={"unix_socket": "/cloudsql/{}".format(cloud_sql_connection_name)},
    ),
    # ... Specify additional properties here.
    # ...
)
```

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.

PYTHON JAVA NODE.JS C#

cloud-sql/mysql/sqlalchemy/main.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/cloud-sql/mysql/sqlalchemy/main.py)

)UDPLATFORM/PYTHON-DOCS-SAMPLES/BLOB/MASTER/CLOUD-SQL/MYSQL/SQLALCHEMY/MAIN.PY)

```
# Preparing a statement before hand can help protect against injections.
stmt = sqlalchemy.text(
    "INSERT INTO votes (time_cast, candidate)" " VALUES (:time_cast, :candidate)"
)
try:
    # Using a with statement ensures that the connection is always released
    # back into the pool at the end of statement (even if an error occurs)
    with db.connect() as conn:
        conn.execute(stmt, time_cast=time_cast, candidate=team)
except Exception as e:
    # If something goes wrong, handle the error in this section. This might
    # involve retrying or adjusting parameters depending on the situation.
    # ...
```

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.

PYTHON JAVA NODE.JS C#

cloud-sql/mysql/sqlalchemy/main.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/cloud-sql/mysql/sqlalchemy/main.py)

)UDPLATFORM/PYTHON-DOCS-SAMPLES/BLOB/MASTER/CLOUD-SQL/MYSQL/SQLALCHEMY/MAIN.PY)

```
# Pool size is the maximum number of permanent connections to keep.
pool_size=5,
# Temporarily exceeds the set pool_size if no connections are available.
max_overflow=2,
# The total number of concurrent connections for your application will be
# a total of pool_size and max_overflow.
```

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.

PYTHON JAVA NODE.JS C#

cloud-sql/mysql/sqlalchemy/main.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/cloud-sql/mysql/sqlalchemy/main.py)

)UDPLATFORM/PYTHON-DOCS-SAMPLES/BLOB/MASTER/CLOUD-SQL/MYSQL/SQLALCHEMY/MAIN.PY)

```
# SQLAlchemy automatically uses delays between failed connection attempts,
# but provides no arguments for configuration.
```

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.

PYTHON JAVA NODE.JS C#

cloud-sql/mysql/sqlalchemy/main.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/cloud-sql/mysql/sqlalchemy/main.py)

)UDPLATFORM/PYTHON-DOCS-SAMPLES/BLOB/MASTER/CLOUD-SQL/MYSQL/SQLALCHEMY/MAIN.PY)

```
# 'pool_timeout' is the maximum number of seconds to wait when retrieving a
# new connection from the pool. After the specified amount of time, an
# exception will be thrown.
pool_timeout=30,  # 30 seconds
```

Killing a connection

A mysql user with the `PROCESS` privilege in Cloud SQL is able to execute a `KILL` statement on a connection of any other mysql user (except Cloud SQL administrative users).

You can list the connections to an instance using the mysql client and executing the `SHOW PROCESSLIST` command. Use the `Id` to kill the connection. For example:

```
mysql> SHOW PROCESSLIST;
mysql> KILL 6;
```

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.

PYTHON JAVA NODE.JS C#

cloud-sql/mysql/sqlalchemy/main.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/cloud-sql/mysql/sqlalchemy/main.py)

)UDPLATFORM/PYTHON-DOCS-SAMPLES/BLOB/MASTER/CLOUD-SQL/MYSQL/SQLALCHEMY/MAIN.PY)

```
# 'pool_recycle' is the maximum number of seconds a connection can persist.
# Connections that live longer than the specified amount of time will be
# reestablished
pool_recycle=1800,  # 30 minutes
```

View the complete application

To see the complete application, click the link below.

PYTHON JAVA NODE.JS C#

View the complete application (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/cloud-sql/mysql/sqlalchemy/README.md) for the Python programming language.

What's next

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

- Learn more about connecting from an external application (https://cloud.google.com/sql/docs/mysql/connect-external-app).

Last updated January 8, 2020.