AceQL HTTP 2.1 - Quick Start Guide

- Server Side Settings
  - Create the kawansoft_example database
  - Linux/Unix Installation & Server Startup
    - Download & installation
    - Update the PATH (Optional)
    - Testing AceQL HTTP Web server installation
    - Configure JDBC parameters in aceql-server.properties file
    - Add your JDBC driver to AceQL installation
    - Start the AceQL HTTP Web Server
  - Windows Installation & Server Startup
    - Add your JDBC driver to AceQL installation
    - Configure JDBC parameters in aceql-server.properties file
    - Start the AceQL Web Server
- Client Side
  - cURL
  - C# Client SDK
  - Java Client SDK
  - Python Client SDK
- From now on...

Server Side Settings

Create the kawansoft_example database

Download the database `kawansoft_example` schema corresponding to your database engine:

- kawansoft_example_mysq.txt
- kawansoft_example_postgresql.txt
- kawansoft_example_ms_sql_serverl.txt
- kawansoft_example_oracle_database.txt

For other databases engines, just tailor the file as indicated: kawansoft_example_other_databases.txt

Then launch the script that will create the tables in a database

Linux/Unix Installation & Server Startup

AceQL requires the installation of Java version 7, 8 or 9.

Download & installation

Open a terminal and download with `Wget`

If you use an Open Source database:

```
$ wget https://www.aceql.com/rest/soft/2.1/download/aceql-http-2.1.run
```

If you use a commercial database:

```
$ wget https://www.aceql.com/rest/soft/2.1/download/aceql-http-pro-2.1.run
```

You can get an AceQL Professional trial license key here: https://www.aceql.com/trial.

If you get a certificate error message, do one of the following:

1. If the problem is that a known root CA is missing and when you are using Ubuntu or Debian, then you can solve the problem with this one line: `sudo apt-getinstall ca-certificates`. Then retry the `Wget` call.
2. Retry the `Wget` call with `--no-check-certificate` at end of command line. Then check the PGP signature of the downloaded file using the corresponding `.asc` signature file available on download page using the PGP hyperlink.

In following lines we will assume that the Open Source edition is chosen. (Operating mode is the same for Pro edition).

```
chmod +x aceql-http-2.1.run
./aceql-http-2.1.run
```

This will create the `aceql-http-2.1` folder.

The full path to the `aceql-http-2.1` installation folder will be surnamed `ACEQL_HOME` in following text.

Example: if you run `aceql-http-2.1.run` from `/home/mike`, then software is installed in

`/home/mike/aceql-http-2.1` which is the value of `ACEQL_HOME`.

Update the PATH (Optional)

Open a shell session and make sure java binary is in the PATH by typing `Java -version` on the command line.

Add java to your PATH if the command does not display Java version.

Add to your PATH the path to the bin directory of `aceql-http-2.1` installation:

```
$ PATH=$PATH:/path/to/aceql-http-2.1/bin/;export PATH
```

Testing AceQL HTTP Web server installation

Call the `aceql-server` script to display the AceQL version:

```
$ aceql-server -version
```

It will display a line with all version info, like:

```
AceQL HTTP Open Source v2.1 - 18-jun-2018
```

Or:

```
AceQL HTTP Professional v2.1 - 18-jun-2018
```

Configure JDBC parameters in aceql-server.properties file

The AceQL HTTP Web Server configuration is done in property file whose surname is `aceql-server.properties`.

AceQL uses the default file `ACEQL_HOME/conf/aceql-server.properties`

Edit `ACEQL_HOME/conf/aceql-server.properties` and go to the `"Tomcat JDBC Connection Pool Section"`.

Set the database name to the `databases` property:

```
# Database names separated by commas
databases = kawansoft_example
```

Change the 4 JDBC properties values accordingly to your JDBC Driver, your database URL, and your database username/password. Each property name must be prefixed with the database name and a dot:

Example for PostgreSQL:

```
# PostgreSQL example
kawansoft_example.driverClassName= org.postgresql.Driver
kawansoft_example.url=jdbc:postgresql://localhost:5432/kawansoft_example
kawansoft_example.username=user1
kawansoft_example.password=password1
```

Add your JDBC driver to AceQL installation

Drop you JDBC driver jar into `ACEQL_HOME/lib-jdbc` directory.

Start the AceQL HTTP Web Server

We will use for our example the port 9090. You may use any port if 9090 is not free.

```
$ aceql-server -start -host localhost –port 9090
```

The console will display the properties used, test that the Connection is established on the server side and tell if everything is OK:

```
[ACEQL HTTP START] Starting AceQL HTTP Web Server...
[ACEQL HTTP START] AceQL HTTP Open Source v2.1 – 18-jun-2018
[ACEQL HTTP START] Using properties file:
[ACEQL HTTP START]  -> /home/mike/aceql-http-2.1/conf/aceql-server.properties
[ACEQL HTTP START] Setting System Properties:
[ACEQL HTTP START] Setting Default Connector attribute values:
[ACEQL HTTP START] Setting Context attribute values:
[ACEQL HTTP START] Setting Tomcat JDBC Pool attributes for kawansoft_example database:
[ACEQL HTTP START]  -> driverClassName =org.postgresql.Driver
[ACEQL HTTP START]  -> url =jdbc:postgresql://localhost:5432/kawansoft_example
[ACEQL HTTP START]  -> username = user1
[ACEQL HTTP START]  -> password = ********
[ACEQL HTTP START] Testing DataSource.getConnection() for kawansoft_example database:
[ACEQL HTTP START]  -> Connection OK!
[ACEQL HTTP START] kawansoft_example Configurators:
[ACEQL HTTP START]  -> databaseConfiguratorClassName:
[ACEQL HTTP START]     org.kawanfw.sql.api.server.DefaultDatabaseConfigurator
[ACEQL HTTP START] Configurators Status: OK.
[ACEQL HTTP START] URL for client side: http://localhost:9090/aceql
[ACEQL HTTP START] AceQL HTTP Web Server OK. Running on port 9090.
```

Don't take care of INFO warnings displays.

If everything is OK, last line will display:

```
[ACEQL HTTP START] AceQL HTTP Web Server OK. Running on port 9090.
```

If any, configuration errors are displayed with the tag

```
[ACEQL HTTP START FAILURE][USER CONFIGURATION]
```

We are now ready to send SQL requests from client side!

Windows Installation & Server Startup

AceQL requires the installation of Java version 7, 8 or 9. (64-bit only).

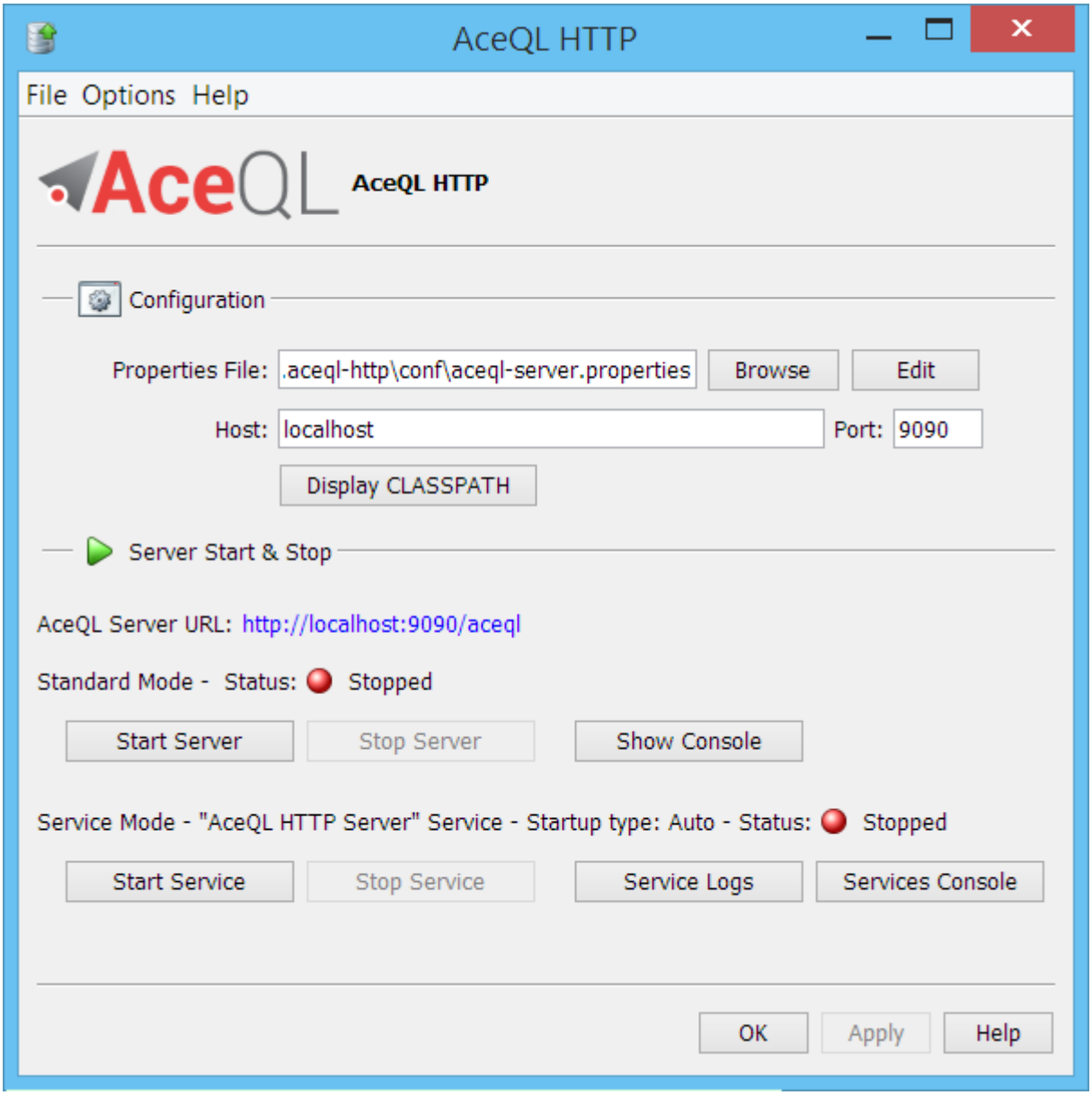
Because the software installs and runs a Windows Service, you must be logged as a Windows Administrator to install AceQL.

Download either:

- AceQL-http-2.1-Setup-x64.exe if you use an Open Source database.
- AceQL-HTTP-Pro-2.1-Setup-x64.exe if you use a commercial database. You can get an AceQL Professional trial license key here: https://www.aceql.com/trial.

Run the installer.

It will run AceQL at end of installation and display the Window:



N.B: Because of a bug in early versions of Java 9 on Windows, the interface will appear "ugly" or "blurred" on Java 9 if you have increased Windows Screen Resolution Options to 125% or 150%. See https://goo.gl/PAVvrd for more info. Set back Windows Screen Resolution to 100% for clean display.

Add your JDBC driver to AceQL installation

A recent PostgreSQL JDBC driver is pre-installed. (We could not pre-install other vendor drivers due to license restrictions). Skip this paragraph if you are a PostgreSQL user.

In order to install your JDBC Driver:

1. Quit completely AceQL with `Ctrl-Q` (or `File` menu & `Quit` item).
2. Add your JDBC Driver to your `CLASSPATH` or copy it to the `\lib-jdbc` subdirectory of the main installation directory.
3. Restart AceQL. Then verify that your JDBC Driver is in your current `CLASSPATH` with the `Display CLASSPATH` button.

Configure JDBC parameters in aceql-server.properties file

The AceQL HTTP Web Server configuration is done in property file whose surname is `aceqlserver.properties`.

A default prefilled `aceql-server.properties` file is set on in the interface

Click `Edit` button to change `aceql-server.properties` content and go to the:

`"Tomcat JDBC Connection Pool Section"`.

Set the database name to the databases property:

```
# Databasenames separated by commas
databases = kawansoft_example
```

Change the 4 JDBC properties values accordingly to your JDBC Driver, your database URL, and your database username/password. Each property name must be prefixed with the database name and a dot.

Example for PostgreSQL:

```
# PostgreSQL example
kawansoft_example.driverClassName= org.postgresql.Driver
kawansoft_example.url=jdbc:postgresql://localhost:5432/kawansoft_example
kawansoft_example.username=user1
kawansoft_example.password=password1
```

Leave the default **localhost** value for the Host field.

You may set any port value for the `Port` field if port 9090 is not free on your machine.

Start the AceQL Web Server

Click on `Start Server`. This will open a console.

The console will display the properties used, test that the `Connection` is established on the server side and tell if everything is OK:

```
[ACEQL HTTP START] Starting AceQL HTTP Web Server...
[ACEQL HTTP START] AceQL HTTP Open Source v2.1 - 18-jun-2018
[ACEQL HTTP START] Using properties file:
[ACEQL HTTP START]  -> c:\.aceql-http\conf\aceql-server.properties
[ACEQL HTTP START] Setting System Properties:
[ACEQL HTTP START] Setting Default Connector attribute values:
[ACEQL HTTP START] Setting Context attribute values:
[ACEQL HTTP START] Setting Tomcat JDBC Pool attributes forkawansoft_example database:
[ACEQL HTTP START]  -> driverClassName = org.postgresql.Driver
[ACEQL HTTP START]  -> url = jdbc:postgresql://localhost:5432/kawansoft_example
[ACEQL HTTP START]  -> username = user1
[ACEQL HTTP START]  -> password = ********
[ACEQL HTTP START] Testing DataSource.getConnection() forkawansoft_example database:
[ACEQL HTTP START]  -> Connection OK!
[ACEQL HTTP START] kawansoft_example Configurators:

[ACEQL HTTP START]  -> databaseConfiguratorClassName:
```

```
[ACEQL HTTP START]     org.kawanfw.sql.api.server.DefaultDatabaseConfigurator
[ACEQL HTTP START] Configurators Status: OK.
[ACEQL HTTP START] URL for client side:http://localhost:9090/aceql
[ACEQL HTTP START] AceQL HTTP Web Server OK. Running on port 9090.
```

If everything is OK, last line will display:

```
[ACEQL HTTP START] AceQL HTTP Server OK. Running on port 9090.
```

If any, configuration errors are displayed with the tag:

```
[ACEQL HTTP START FAILURE][ USER CONFIGURATION]
```

We are now ready to send SQL requests from client side!

Client Side

AceQL can be accessed from client side:

- **Using any command line tool to make HTTP calls**. We will provide examples with cURL.
- **Using the C# Client SDK with C#** SQL regular syntax, same as with SQL Server Client classes. The C# Client SDK wraps all http communications aspects. Jump to C# Client SDK.
- **Using the Java Client SDK** that allows regular JDBC calls and wraps all HTTP communications aspects. Jump to Java Client SDK.
- **Using the Python Client SDK** that allows regular DB API 2.0 SQL calls and wraps all HTTP communications aspects. Jump to Python Client SDK.
- **Using the Swift Client SDK**. Read the Swift Client SDK User Guide.
- Using any other language that supports HTTP GET & POST calls.

cURL

So via cURL we connect to a database `kawansoft_example` with the identifiers `(MyUsername, MySecret)`:

```
$ curl --data-urlencode "password=MySecret" \
 http://localhost:9090/aceql/database/kawansoft_example/username/MyUsername/connect
```

le/username/MyUsername/connect

The command returns a JSON stream with a unique session identifier and a connection identifier:

```
{
   "status":"OK",
   "session_id":"mn7andp2tt049iaeaskr28j9ch"
   "connection_id":"1628176139"
}
```

On the server side, a JDBC connection is extracted from the connection pool created by the server at startup. The connection will remain ours during the session.

We will use the session identifier to authenticate all subsequent calls.

We insert a customer into the database:

```
$ curl --data-urlencode \
 "sql=insert into customer values (1,'Sir', 'Doe', 'John', '1 Madison Ave', 'New York', 'NY
10010', NULL)" \
 http://localhost:9090/aceql/session/mn7andp2tt049iaeaskr28j9ch/execute_update
```

Which returns:

```
{
   "status":"OK",
   "row_count":1
}
```

We view the inserted customer:

```
$ curl \
 --data-urlencode "sql=select * from customer" --data"pretty_printing=true" \
 http://localhost:9090/aceql/session/mn7andp2tt049iaeaskr28j9ch/execute_query
```

This returns the JSON stream:

```
{
    "status":"OK",
    "query_rows":[
      {
          "row_1":[
              {
                  "customer_id":1
              },
              {
                  "customer_title":"Sir "
              },
              {
                  "fname":"John"
              },
              {
                  "lname":"Doe"
              },
              {
                  "addressline":"1600 Pennsylvania Ave NW"
              },
              {
                  "town":"Washington"

              },
```

```
            {
                "zipcode":"DC 20500  "
            },
            {
                "phone":"NULL"
            }
        ]
    }
  ],
  "row_count":1
}
```

Let's update our customer using a prepared statement:

```
$ curl --data"prepared_statement=true" \
 --data "param_type_1=VARCHAR&param_value_1=Jim" \
 --data "param_type_2=INTEGER&param_value_2=1"\
 --data-urlencode"sql=update customer set fname=? where customer_id=?" \
 http://localhost:9090/aceql/session/mn7andp2tt049iaeaskr28j9ch/\
 execute_update
```

Which returns:

```
{
   "status":"OK",
   "row_count":1
}
```

And now we query back our customer, but without pretty printing and ask to GZIP the result:

```
$ curl \
 --data"pretty_printing=true&gzip_result=true" \
 --data-urlencode \
 "sql=select customer_id, customer_title,fname from customer" \
 http://localhost:9090/aceql/session/mn7andp2tt049iaeaskr28j9ch/\
 execute_query>result.gzip
```

And we end with a clean close of our session:

```
$ curl \
 http://localhost:9090/aceql/session/mn7andp2tt049iaeaskr28j9ch/disconnect
```

On the server side, the authentication info is purged and the JDBC connection is released in the pool. (A server thread regularly releases phantom connections that were not closed from the client side.)

From now, you can read the API User Guide to learn how to:

- Query or modify the `Connection` properties.
- Create SQL transactions.

- Insert Blobs in the database.
- Retrieve Blobs from the database.

C# Client SDK

1. Create the "AceQL.MyRemoteConnection" Windows Classic Desktop Console App in Visual Studio.
2. Install the AceQL.Client package with NuGet.
3. Download this C# source file: MyRemoteConnection.cs. Then insert it in your project.
4. The connection to the remote database is created using `AceQLConnection` class and passing the URL of the AceQL Servlet Manager of your configuration:

```
    /// <summary>
    /// RemoteConnection Quick Start client example.
    /// Creates a Connection to a remote database and open it.
    /// </summary>
    /// <returns>The connection to the remote database</returns>
    /// <exception cref="AceQLException">If any Exception occurs.</exception>
    public static async Task<AceQLConnection> ConnectionBuilderAsync()
    {
        // Port number is the port number used to start the Web Server:
        string server = "https://www.aceql.com:9443/aceql";
        string database = "kawansoft_example";

        string connectionString = $"Server={server}; Database={database}";

        // (username, password) for authentication on server side.
        // No authentication will be done for our Quick Start:
        string username = "MyUsername";
        char[] password = { 'M', 'y', 'S', 'e', 'c', 'r', 'e', 't' };

        AceQLConnection connection = new AceQLConnection(connectionString)
        {
            Credential = new AceQLCredential(username, password)
        };

        // Opens the connection with the remote database.
        // On the server side, a JDBC connection is extracted from the connection
        // pool created by the server at startup. The connection will remain ours
        // during the session.
        await connection.OpenAsync();

        return connection;
    }
```

5. Build and run. It will insert a new `customer` and a new `orderlog`:

```
    /// <summary>
    /// Example of 2 INSERT in the same transaction.
    /// </summary>
    /// <param name="customerId">The customer ID.</param>
```

```
    /// <param name="itemId">the item ID.</param>
    /// <exception cref="AceQLException">If any Exception occurs.</exception>
    public async Task InsertCustomerAndOrderLogAsync(int customerId, int itemId)
    {
        // Create a transaction
        AceQLTransaction transaction = await connection.BeginTransactionAsync();

        string sql = "insert into customer values " + "" +
            "(@parm1, @parm2, @parm3, @parm4, @parm5, @parm6, @parm7, @parm8)";

        AceQLCommand command = new AceQLCommand(sql, connection);
        try
        {
            command.Parameters.AddWithValue("@parm1", customerId);
            command.Parameters.AddWithValue("@parm2", "Sir");
            command.Parameters.AddWithValue("@parm3", "Doe");
            command.Parameters.AddWithValue("@parm4", "John");
            // Alternate syntax
            command.Parameters.Add(new AceQLParameter("@parm5", "1 Madison Ave"));
            command.Parameters.AddWithValue("@parm6", "New York");
            command.Parameters.AddWithValue("@parm7", "NY 10010");
            command.Parameters.Add(new AceQLParameter("@parm8", AceQLNullType.VARCHAR));

            await command.ExecuteNonQueryAsync();

            sql = "insert into orderlog values " +
                        "(@customer_id, @item_id, @description, " +
                            "@item_cost, @date_placed, @date_shipped, " +
                            "@jpeg_image, @is_delivered, @quantity)";

            command = new AceQLCommand(sql, connection);

            command.Parameters.AddWithValue("@customer_id", customerId);
            command.Parameters.AddWithValue("@item_id", itemId);
            command.Parameters.AddWithValue("@description", "Item Description");
            command.Parameters.AddWithValue("@item_cost", 99D);
            command.Parameters.AddWithValue("@date_placed", DateTime.Now);
            command.Parameters.AddWithValue("@date_shipped", DateTime.Now);
            // No blob in our Quick start
            command.Parameters.Add(new AceQLParameter("@jpeg_image",
                AceQLNullType.BLOB));
            command.Parameters.AddWithValue("@is_delivered", 1);
            command.Parameters.AddWithValue("@quantity", 1);

            await command.ExecuteNonQueryAsync();
            await transaction.CommitAsync();
        }
        catch (Exception e)
        {
            await transaction.RollbackAsync();
            throw e;
        }

    }
```

The `SelectCustomerAndOrderLogAsync()` method of MyRemoteConnection.cs displays back the inserted values.

From now on, you can read the C# Client SDK User Guide.

Java Client SDK

1. Maven:

```
<groupId>com.aceql</groupId>
<artifactId>aceql-http-client-sdk</artifactId>
<version>1.0</version>
```

2. If you don't use Maven: the aceql-http-client-all-1.0.1.jar file contains the SDK with all dependencies.
3. Create an `org.kawanfw.sql.api.client.quickstart` package in your IDE.
4. Download this Java source file: MyRemoteConnection.java. Then insert it in the package.
5. The connection to the remote database is created using `AceQLConnection` class and passing the URL of the AceQL Servlet Manager of your configuration:

```
/**
 * Remote Connection Quick Start client example. Creates a Connection to a
 * remote database.
 *
 * @return the Connection to the remote database
 * @throws SQLException
 *             if a database access error occurs
 */
public static Connection remoteConnectionBuilder() throws SQLException {

  // The URL of the AceQL Server servlet
  // Port number is the port number used to start the Web Server:
  String url = "http://localhost:9090/aceql";

  // The remote database to use:
  String database = "kawansoft_example";

  // (username, password) for authentication on server side.
  // No authentication will be done for our Quick Start:
  String username = "MyUsername";
  char[] password = { 'M', 'y', 'S', 'e', 'c', 'r', 'e', 't' };

  // Attempt to establish a connection to the remote database:
  Connection connection = new AceQLConnection(url, database, username,
      password);

  return connection;
}
```

6. Compile and run from your IDE the `MyRemoteConnection` class. It will insert a new `customer` and a new `orderlog`:

```
/**
 * Example of 2 INSERT in the same transaction.
 *
 * @param customerId
 *            the Customer Id
 * @param itemId
 *            the Item Id
 *
 * @throws SQLException
 */
public void insertCustomerAndOrderLog(int customerId, int itemId)
    throws SQLException {

  connection.setAutoCommit(false);

  try {
      // Create a Customer
      String sql = "INSERT INTO CUSTOMER VALUES ( ?, ?, ?, ?, ?, ?, ?, ? )";
      PreparedStatement prepStatement = connection.prepareStatement(sql);

      int i = 1;
      prepStatement.setInt(i++, customerId);
      prepStatement.setString(i++, "Sir");
      prepStatement.setString(i++, "Doe");
      prepStatement.setString(i++, "John");
      prepStatement.setString(i++, "1 Madison Ave");
      prepStatement.setString(i++, "New York");
      prepStatement.setString(i++, "NY 10010");
      prepStatement.setString(i++, null);

      prepStatement.executeUpdate();
      prepStatement.close();

      // Uncomment following line to test transaction behavior
      // if (true) throw new SQLException("Exception thrown.");

      // Create an Order for this Customer
      sql = "INSERT INTO ORDERLOG VALUES ( ?, ?, ?, ?, ?, ?, ?, ?, ? )";

      // Create a new Prepared Statement
      prepStatement = connection.prepareStatement(sql);

      i = 1;
      long now = new java.util.Date().getTime();

      prepStatement.setInt(i++, customerId);
      prepStatement.setInt(i++, itemId);
      prepStatement.setString(i++, "Item Description");
      prepStatement.setBigDecimal(i++, new BigDecimal("99.99"));
      prepStatement.setDate(i++, new java.sql.Date(now));
```

```
            prepStatement.setTimestamp(i++, new Timestamp(now));
            // No Blob in this example.
            prepStatement.setBinaryStream(i++, null);
            prepStatement.setInt(i++, 1);
            prepStatement.setInt(i++, 2);

            prepStatement.executeUpdate();
            prepStatement.close();

            System.out.println("Insert done!");
        } catch (SQLException e) {
            e.printStackTrace();
            connection.rollback();
            throw e;
        } finally {
            connection.setAutoCommit(true);
        }
    }
```

The `selectCustomerAndOrderLog` method of MyRemoteConnection.java displays back the inserted values.

From now on, you can read the Java Client SDK User Guide or run through the SDK Javadoc.

Python Client SDK

The aceql module supports Python 2.6–2.7 & 3.4–3.7.

1. Create a new project with your favorite IDE.
2. Install aceql module:

```
$ pip install aceql
```

3. Download this Python class: my_remote_connection.py
4. The connection to the remote database is created using a DB API 2.0 `Connection` class and passing the URL of the AceQL Servlet Manager of your configuration:

```
    @staticmethod
    def remote_connection_builder():
        """Remote Connection Quick Start client example.

        Creates a Connection to a remote database.
        """

        # The URL of the AceQL Server servlet
        # Port number is the port number used to start the Web Server:
        url = "http://localhost:9090/aceql"

        # The remote database to use:
        database = "kawansoft_example"

        # (username, password) for authentication on server side.
```

```
        # No authentication will be done for our Quick Start:
        username = "MyUsername"
        password = "myPassword"

        # Attempt to establish a connection to the remote database.
        # On the server side, a JDBC connection is extracted from the
        # connection pool created by the server at startup.
        # The connection will remain ours during the session.
        connection = aceql.connect(url, database, username, password)
        return connection
```

5. Run from your IDE the `my_remote_connection.py` module. It will insert a new `customer` and a new `orderlog`:

```
    def insert_customer_and_order_log(self, customer_id, item_id):
        """Example of 2 INSERT in the same transaction
           using a customer id and an item id.
        """

        self.connection.set_auto_commit(False)
        cursor = self.connection.cursor()

        try:
            # Create a Customer
            sql = "insert into customer values (?, ?, ?, ?, ?, ?, ?, ?)"
            params = (customer_id, 'Sir', 'John', 'Smith', '1 Madison Ave',
                      'New York', 'NY 10010', '+1 212-586-7000')
            cursor.execute(sql, params)

            # Create an Order for this Customer
            sql = "insert into orderlog values ( ?, ?, ?, ?, ?, ?, ?, ?, ? )"

            the_datetime = datetime.now()
            the_date = the_datetime.date()

            # (None, SqlNullType.BLOB) means to set the jpeg_image BLOB
            # column to NULL on server:
            params = (customer_id, item_id, "Item Description", 9999,
                      the_date, the_datetime, (None, SqlNullType.BLOB), 1, 2)
            cursor.execute(sql, params)

            self.connection.commit()
        except Error as e:
            print(e)
            self.connection.rollback()
            raise e
        finally:
            self.connection.set_auto_commit(True)
            cursor.close()
```

The `select_customer_and_orderlog` method of [my_remote_connection.py] displays back the inserted values.

From now on, you can read the Python Client SDK User Guide.

From now on...

You can read the Server User Guide. You will learn:

- How to create a Connection Pool.
- How to create a strong authentication on the server for your legitimate users.
- How to secure accesses to your SQL databases.