AceQL

AceQL HTTP v1.0

Quick Start Guide

June 30, 2017

KawanSoft SAS, a French S.A.S corporation with assets of € 2,000
R.C.S. registration Paris B 534 990 981
56, boulevard Pereire - 75017 Paris
Tel: +33 (0)1 77 69 59 58 - Fax: +33 (0)1 72 74 95 39 - Web: www.kawansoft.com

Contents

1 Server Side Settings

1.1 Create the kawansoft_example database

Download the example database schema: kawansoft_example.txt. Change the types of the `date_shipped` and `jpeg_image` columns accordingly to your database engine. Then launch the script that will create the tables in a database.

(The database default name used in our quick start is `kawansoft_example` but you can create the tables in any database).

1.2 Installation

Download AceQL binary installation file.
(Choose Open Source version if you use an Open Source database, choose Professional version for commercial database.)

Unzip/untar the binary file in a directory.

1.2.1 Setting ACEQL_HOME environment variable

Set the `ACEQL_HOME` environment variable value to the installation directory.

Windows example
If you unzip `aceql-http-1.0-bin.zip` in `c:\Users\Mike`:

```
c:>set ACEQL_HOME="c:\Users\Mike\aceql-http-1.0-bin"
```

The quotation marks are optional and required only if ACEQL_HOME path contains spaces (ex: `c:\Users\Mike Smith\aceql-4.0-bin`).

Linux example:
If you untar `aceql-http-1.0-bin.tar.gz` in `/home/mike`:

```
$ ACEQL_HOME=/home/mike/aceql-http-1.0-bin;export ACEQL_HOME
```

1.2.1.1 Linux/Unix script settings

Set exec permissions to `ACEQL_HOME/bin/aceql-server.sh` script:

```
$ chmod ug+x $ACEQL_HOME/bin/aceql-server.sh
```

Add a symbolic link in order to call the script without the `.sh` extension:

```
$ ln -s $ACEQL_HOME/bin/aceql-server.sh $ACEQL_HOME/bin/aceql-server
```

1.2.2 Update the PATH

Open a shell session and make sure `java` binary on Linux/Unix or `java.exe` on Windows is in the `PATH` by typing `Java -version` on the command line.
Add `java` or `java.exe` to your `PATH` if the command does not display Java version.

Add to your `PATH` the `bin` directory of AceQL installation:

Windows

```
C:>SET PATH=%PATH%;%ACEQL_HOME%\bin
```

Unix/Linux

```
$ PATH=$PATH:$ACEQL_HOME/bin/;export PATH
```

1.2.3 Testing server installation

Open a shell session and make sure `java` binary on Linux/Unix or `java.exe` on Windows is in the `PATH` by typing `Java -version` on the command line.
Add `java` or `java.exe` to your `PATH` if the command does not display Java version.

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

Windows

```
c:>aceql-server -version
```

Unix/Linux

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

Note
Except in code examples, we will use from now on the sign / to designate indifferently Windows or Unix/Linux directories.

1.3 Configure JDBC parameters in aceql-server.properties file

The AceQL 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".
Change the 4 properties values accordingly to your JDBC Driver, your database URL, and your database username/password:

Example for PostgreSQL:

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

1.4 Add your JDBC driver to AceQL installation

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

1.5 Start the AceQL Web Server

We will use for our example the port 9090. You may use any port if 9090 is not free.
The AceQL Web server is started with a script file (Windows .bat and Unix/Linux Bash).

1.5.1 Windows

```
c:>aceql-server -start -host localhost -port 9090
```

1.5.2 Linux/Unix

```
$ 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 v1.0 - 27-jun-2017
[ACEQL HTTP START] Using properties file:
[ACEQL HTTP START]  -> /home/mike/aceql-http-1.0-bin/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.
[ACEQL HTTP START] To close normally: java org.kawanfw.sql.WebServer -stop -port 9090
[ACEQL HTTP START] From command line, use [Ctrl]+[C] to abort abruptly
```

Don’t take care of INFO warnings displays.

If any, configuration errors are displayed with the tag [KAWANSOFT FRAMEWORK - USER CONFIGURATION FAILURE].
We are now ready to send SQL requests from client side!

2 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 *2.2 C# Client SDK: using SQL*).
- **Using the Java Client SDK** that allows regular JDBC calls and wraps all http communications aspects. (Jump to *2.3 Java Client SDK: using JDBC*).
- Using any other language that supports HTTP GET & POST calls.

2.1 cURL

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

```
curl \
 http://localhost:9090/aceql/database/kawansoft_example/username/\
MyUsername/connect?password=MySecret
```

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

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

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
```

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.

2.2 C# Client SDK: using SQL

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 `ServerSqlManager` Servlet of your configuration :

5) Build and run. It will insert a new Customer and a new Orderlog:

```
/// <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()
{
    string server = "http://localhost:9090/aceql";
    string database = "kawansoft_example";

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

    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
    await connection.OpenAsync();

    return connection;
}
```

```
    /// <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.

2.3 Java Client SDK: using JDBC

1. Maven:

```
<groupId>com.aceql</groupId>
<artifactId> aceql-http-client-sdk</artifactId>
<version>1.0-beta.3</version>
```

2. Create an `org.kawanfw.sql.api.client.quickstart` package in your IDE.

3. Download this java source file: MyRemoteConnection.java. Then insert it in the package.

4. The Connection to the remote database is created using AceQLConnection class and passing the URL of the `ServerSqlManager` Servlet 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;
}
```

5. 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 if a database access error occurs
     */
    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();

            // 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 Client Java SDK User Guide or run through the SDK Javadoc.

3 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.
