# **ADO.NET - Design and Implementation**

## **Why ADO.NET**

* Ado.NET provider will help with easy ways to communicate with splicemachine using .NET framework.
* This enhanced way makes .NET users feel comfortable since the language used is C#.
* Please refer to the unit test file in the [link](https://drive.google.com/file/d/1BneVfobmHtp4iJtwc_-adK9yxCFPT4l8/view?usp=sharing) or copy paste the url to the browser and download to learn quickly how to use the provider.
* The unit test file will have methods helping in usage of the provider.

## **Establishing the Connection**

* SpliceDbConnection class is the connection class where we specify the connection properties like host, userid, password and port.
* Connection string will be in the format: “uid=admin;pwd=password;host=localhost;port=1527;”.
* Most used members of connection object are as follows:

1. Open()
2. Close()
3. Commit()
4. Rollback()
5. IsAutoCommit(Bool)

* Open() and Close() methods will be used to open and close the connections.(Ref. CODE SAMPLE #0001)
* Commit() and Rollback() will be used for committing or rolling back a transaction.(Ref. CODE SAMPLE #0002,#0002#A)
* IsAutoCommit is by default set to true and the commit is handled automatically while executing the queries. When it is set to false the commit() should be called to commit a transaction at known scope.(Ref. CODE SAMPLE #0002)

**CODE SAMPLE #0001:**

using (var connection = new SpliceDbConnection("uid=splice;pwd=admin;host=localhost;port=1527"))

{

connection.Open();

using (var command = connection.CreateCommand())

{

command.CommandText = "CREATE TABLE TESTTABLE(a int)";

var count = command.ExecuteNonQuery();

}

connection.Close();

}

**CODE SAMPLE #0002:**

using (var connection = new SpliceDbConnection("uid=splice;pwd=admin;host=localhost;port=1527"))

{

connection.Open();

connection.IsAutoCommit = false;

using (var command = connection.CreateCommand())

{

command.CommandText = "CREATE TABLE TESTTABLE(a int)";

var count = command.ExecuteNonQuery();

connection.Commit();

}

connection.Close();

}

**CODE SAMPLE #0002#A:**

using (var connection = new SpliceDbConnection("uid=splice;pwd=admin;host=localhost;port=1527"))

{

const string QryCreateTable = "CREATE TABLE TESTTABLE(COL1 BIGINT,COL2 INT,COL3 DECIMAL(16,5))";

const string QryInsertValue = "INSERT INTO TESTTABLE VALUES(?,?,?)";

connection.Open();

using (var command = connection.CreateCommand())

{

connection.IsAutoCommit = false;

command.CommandText = QryCreateTable;

var result1 = command.ExecuteNonQuery();

command.CommandText = QryInsertValue;

command.Parameters.Add(new SpliceDbParameter() { Value = 456});

command.Parameters.Add(new SpliceDbParameter() { Value = 789});

command.Parameters.Add(new SpliceDbParameter() { Value = 32.21 });

var result2 = command.ExecuteNonQuery();

connection.Rollback();

}

}

## **Creating a command**

* Creating a command can be done using SpliceDbCommand class.
* Most used members of the class are below:

1. CommandText(Ref. CODE SAMPLE #0002)
2. ExecuteNonQuery()(Ref. CODE SAMPLE #0002)
3. ExecuteReader()
4. Parameters

* CommandText is the string property which will be assigned with a query.
* ExecuteNonQuery will return the number of rows getting affected in the db after the query execution.
* ExecuteReader will be used to fetch the data from Db.
* Parameters will be used to add parameters to a command.

## **Create/Drop Tables/Sequences/Triggers:**

* Any DDL query can be executed using ExecuteNonQuery() method to create or drop a table or schema or sequences or triggers.
* Refer SAMPLE **#0002#A** for a sample program to create a table. Similar programs can be used to do any operations mentioned above.

## **Insert into Table with Parameters**

* SpliceDbParameter class can be used to add parameters to a command.(Ref. CODE SAMPLE #0003)
* Parameters inside the query should be indicated with a ‘?’ symbol.

**CODE SAMPLE #0003:**

using (var connection = new SpliceDbConnection("uid=splice;pwd=admin;host=localhost;port=1527"))

{

connection.Open();

connection.IsAutoCommit = false;

using (var command = connection.CreateCommand())

{

command.CommandText = "INSERT INTO TESTTABLE VALUES(?)";

var parameter1 = new SpliceDbParameter();

parameter1.Value = 1;

command.Parameters.Add(parameter1);

var count = command.ExecuteNonQuery();

connection.Commit();

}

connection.Close();

}

**Data Retrieval**

* Data retrieval is possible using two ways,

1. ExecuteReader()(Ref. CODE SAMPLE #0004)
2. SpliceDbAdapter class(Ref. CODE SAMPLE #0005)

* ExecuteReader() will fetch the data from the table every time we call up the read() method.
* SpliceDbAdapter will fetch the entire data from the table and populate it into a DataTable.
* Stored Procedures/Functions can be invoked only through ExecuteReader(). SpliceDbAdapter does not support StoredProcedures/Functions.

**CODE SAMPLE #0004:**

using (var connection = new SpliceDbConnection("uid=splice;pwd=admin;host=localhost;port=1527"))

{

connection.Open();

using (var command = connection.CreateCommand())

{

command.CommandText = "SELECT \* FROM TESTTABLE";

var dataReader = command.ExecuteReader();

while(dataReader.read())

{

//Every row data will be present in the dataReader object. Column index can be indicated inside square brackets.

var data1 = dataReader[0];

}

}

connection.Close();

}

**CODE SAMPLE #0005:**

using (var connection = new SpliceDbConnection("uid=splice;pwd=admin;host=localhost;port=1527"))

{

connection.Open();

using (var command = connection.CreateCommand())

{

command.CommandText = "SELECT \* FROM TESTTABLE";

DataTable myData = new DataTable();

SpliceDbAdapter adapter = new SpliceDbAdapter();

adapter.SelectCommand = command;

adapter.Fill(myData);

//myData will have the rows and columns of the result set

}

connection.Close();

}

**Procedure/Functions execution**

* Procedure/Function can be called/invoked simply using the ExecuteNonQuery() method.
* In case of the data retrieval through function/procedure we can use a data reader.
* Procedures can also be equipped with parameters as like we do in an insert statement.
* Refer CODE SAMPLE #0006 for the data retrieval through procedures/functions.

**CODE SAMPLE #0006:**

using (var connection = new SpliceDbConnection("uid=splice;pwd=admin;host=localhost;port=1527"))

{

connection.Open();

using (var command = connection.CreateCommand())

{

command.CommandText = "CALL SYSCS\_UTIL.SYSCS\_GET\_ALL\_PROPERTIES()";

var dataReader = command.ExecuteReader();

while(dataReader.read())

{

//Every row data will be present in dataReader object. Column index can be indicated inside square brackets.

var data1 = dataReader[0];

}

}

connection.Close();

}

## **SpliceMachine DRDA**

* The drda layer has the core connectivity layer through which the splice machine gets connected and worked.
* This layer is responsible for connection, query execution and result set handling.
* On top of SIMBA this layer will communicate with db and handle the request and responses at the backend.
* Apart from BLOB and CLOB all data types are supported in this layer.
* It has methods to execute the query as an immediate or prepared statement.
* It also includes the parameter get and set extensions.

## **DRDA Read Write Flow**

* To know the core methods of reading and writing data to a database, the important classes involving read write operations are,
* DrdaColumnExtensions.cs
* DrdaStreamReaderExtensions.cs
* DrdaStreamWriterExtensions.cs

## **DrdaColumnExtensions.cs**

* This file has two methods WriteColumnValue and ReadColumnValue for writing and reading values.

## **DrdaStreamWriterExtensions.cs**

* This file has extension methods to load the stream with bytes values.

## **SpliceMachine Provider**

* Provider is the layer on top of DRDA. This layer will have reference to SIMBA dlls.
* This layer will simplify the development of ADO.NET with use of DRDA in the backend.
* This layer will be the main communication layer that the ADO.NET user will be using. It has classes that the user will be using directly(Ex. SpliceConnection,SpliceCommand,etc).

## 

## **Development tries made for BLOB and CLOB**

* There are two problems dealing with BLOB and CLOB one is writing large BLOB values and CLOB values. Another one is reading BLOB and CLOB values.
* The core problem of writing large blob and clob values is with writing length.
* The length calculation for larger blob and clob has some difficulties.
* The file location of the length calculation is **Parameters => CompositeCommand.cs**
* The core problem with reading BLOB and CLOB is the development part for reading blob and clob is missing in DRDA. This may be yet to be developed. In comparison with the codepoints that are sent to the server from JDBC, there were several codepoints that were sent in JDBC whereas it was missing in ADO.NET. Need to analyse the missing codepoints and add the same in DRDA.