CUBRID ADO.NET Data Provider

Tutorial – Part II

December 2011- 1st release

January 2013–Updated for 9.1.0 release

This is the 2nd part of the **CUBRID ADO.NET Data Provider**.In this part, we will show you how to use someCUBRID specific ADO.NET extensions.

**Note**: If you haven’t read yet the 1st part, we strongly recommend you to do it, before proceeding with this 2ndpart<http://www.cubrid.org/wiki_apis/entry/cubrid-ado-net-tutorials>).

# CUBRID Metadata support

The CUBRID .NET 9.1.0 Driver adds support for databases metadata. Most of these methods are implemented in the **CUBRIDSchemaProvider**class:

publicDataTableGetDatabases(string[] filters)

publicDataTableGetTables(string[] filters)

publicDataTableGetViews(string[] filters)

publicDataTableGetColumns(string[] filters)

publicDataTableGetIndexes(string[] filters)

publicDataTableGetIndexColumns(string[] filters)

publicDataTableGetExportedKeys(string[] filters)

publicDataTableGetCrossReferenceKeys(string[] filters)

publicDataTableGetForeignKeys(string[] filters)

publicDataTableGetUsers(string[] filters)

publicDataTableGetProcedures(string[] filters)

publicstaticDataTableGetDataTypes()

publicstaticDataTableGetReservedWords()

publicstaticString[] GetNumericFunctions()

publicstaticString[] GetStringFunctions()

publicDataTable GetSchema(string collection, string[] filters)

Let’s see some examples of getting metadata information from a CUBRID connection.

## Get the list of Tables in the current database:

CUBRIDSchemaProvider schema = newCUBRIDSchemaProvider(conn);

DataTabledt = schema.GetTables(newstring[] { "%" });

Debug.Assert(dt.Columns.Count == 3);

Debug.Assert(dt.Rows.Count == 10);

Debug.Assert(dt.Rows[0][0].ToString() == "demodb");

Debug.Assert(dt.Rows[0][1].ToString() == "demodb");

Debug.Assert(dt.Rows[0][2].ToString() == "stadium");

## Get the list of Foreign Keys in a table:

CUBRIDSchemaProvider schema = newCUBRIDSchemaProvider(conn);

DataTabledt = schema.GetForeignKeys(newstring[] { "game" });

Debug.Assert(dt.Columns.Count == 9);

Debug.Assert(dt.Rows.Count == 2);

Debug.Assert(dt.Rows[0][0].ToString() == "athlete");

Debug.Assert(dt.Rows[0][1].ToString() == "code");

Debug.Assert(dt.Rows[0][2].ToString() == "game");

Debug.Assert(dt.Rows[0][3].ToString() == "athlete\_code");

Debug.Assert(dt.Rows[0][4].ToString() == "1");

Debug.Assert(dt.Rows[0][5].ToString() == "1");

Debug.Assert(dt.Rows[0][6].ToString() == "1");

Debug.Assert(dt.Rows[0][7].ToString() == "fk\_game\_athlete\_code");

Debug.Assert(dt.Rows[0][8].ToString() == "pk\_athlete\_code");

## Get the list of Indexes in a table:

CUBRIDSchemaProvider schema = newCUBRIDSchemaProvider(conn);

DataTabledt = schema.GetIndexes(newstring[] { "game" });

Debug.Assert(dt.Columns.Count == 9);

Debug.Assert(dt.Rows.Count == 5);

Debug.Assert(dt.Rows[3][2].ToString() == "pk\_game\_host\_year\_event\_code\_athlete\_code"); //Index name

Debug.Assert(dt.Rows[3][4].ToString() == "True"); //Is it a PK?

Remember, there is a lot more ☺– please check the data provider documentation (<todo – link to .chm help file>).

# CUBRID OIDs

**First of all, what is an OID…?**

A **CUBRID OID**(Object Identifier) is an object identifier represented by physical location information such as the volume number, page number and slot number. By using such OIDs, CUBRID manages the reference relationships of objects and searches, saves or deletes them. When an OID is used, accessibility is improved because the object in the heap file can be directly accessed without referring to the table (see <http://www.cubrid.org/questions/235437>).

As the concept of OID is specific only to CUBRID, the CUBRID .NET Provider implements custom extensions to deal with. For example:

CUBRIDOidoid = newCUBRIDOid("@620|1|0");

Debug.Assert(oid.Page() == 620);

Debug.Assert(oid.Slot() == 1);

Debug.Assert(oid.Volume() == 0);

or:

string sql = "select \* from nation limit 1";

using (CUBRIDCommandcmd = newCUBRIDCommand(sql, conn))

{

using (CUBRIDDataReader reader = (CUBRIDDataReader)cmd.ExecuteReader())

{

reader.Read();

CUBRIDOidoid = reader.GetOid();

}

}

# CUBRID Collections

**Collections** are a specific CUBRID data type(s). If you are not familiar with them, you can read more here: <http://www.cubrid.org/manual/840/en/Collection%20Types>.

Because **collections** are not common to any database, but only to CUBRID, the support for them is implemented in some specific **CUBRIDConnection**methods:

publicvoidAddElementToSet(CUBRIDOid oid, StringattributeName, Object value)

publicvoidDropElementInSet(CUBRIDOid oid, StringattributeName, Object value)

publicvoidUpdateElementInSequence(CUBRIDOid oid, StringattributeName, int index, Object value)

publicvoidInsertElementInSequence(CUBRIDOid oid, StringattributeName, int index, Object value)

publicvoidDropElementInSequence(CUBRIDOid oid, StringattributeName, int index)

publicintGetCollectionSize(CUBRIDOid oid, StringattributeName)

Herebelow are two examples of using these CUBRID extensions.

## Reading values from a Collection data type:

using (CUBRIDCommandcmd = newCUBRIDCommand("SELECT \* FROM t", conn))

{

using (DbDataReader reader = cmd.ExecuteReader())

{

while (reader.Read())

{

object[] o = (object[])reader[0];

for (inti = 0; i<SeqSize; i++)

{

/…

}

}

}

}

## Updating a Collection data type:

conn.InsertElementInSequence(oid, attributeName, 5, value);

SeqSize = conn.GetCollectionSize(oid, attributeName);

using (CUBRIDCommandcmd = newCUBRIDCommand("SELECT \* FROM t", conn))

{

using (DbDataReader reader = cmd.ExecuteReader())

{

while (reader.Read())

{

int[] expected = { 7, 1, 2, 3, 7, 4, 5, 6 };

object[] o = (object[])reader[0];

/…

}

}

}

conn.DropElementInSequence(oid, attributeName, 5);

SeqSize = conn.GetCollectionSize(oid, attributeName);

# CUBRID LOBs

Starting from **8.4**release(<http://www.cubrid.org/release_history>), CUBRID deprecated the **GLO** data type, and added support for **LOB** data types:

* **BLOB**: Binary (<http://www.cubrid.org/manual/840/en/BLOB|CLOB>)
* **CLOB**: Character LOB (<http://www.cubrid.org/manual/840/en/BLOB|CLOB>)

These are specific CUBRID data type, and if you need to deal with such data types, you need to learn and use the CUBRID ADO.NET Provider specific extensions.

Here are some basic source code examples to get you start up fast.

## Reading BLOB data:

CUBRIDCommandcmd = newCUBRIDCommand(sql, conn);

DbDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

CUBRIDBlobbImage = (CUBRIDBlob)reader[0];

byte[] bytes = newbyte[(int)bImage.BlobLength];

bytes = bImage.getBytes(1, (int)bImage.BlobLength);

//…

}

## Updating CLOB data:

string sql = "UPDATE t SET c = ?";

CUBRIDCommandcmd = newCUBRIDCommand(sql, conn);

CUBRIDClobClob = newCUBRIDClob(conn);

str = conn.ConnectionString; //Use the ConnectionString for testing

Clob.setString(1, str);

CUBRIDParameterparam = newCUBRIDParameter();

param.ParameterName = "?";

param.CUBRIDDataType = CUBRIDDataType.CCI\_U\_TYPE\_CLOB;

param.Value = Clob;

cmd.Parameters.Add(param);

cmd.ExecuteNonQuery();

# Generic development notes

When developing.NET data provider applications, we recommend looking at the following resources:

* Follow the recommendations and best practices regarding developing secure .NET applications; you can find more information and help here (ADO.NET): <http://msdn.microsoft.com/en-us/library/hdb58b2f%28VS.80%29.aspx> and here (OLE DB) <http://msdn.microsoft.com/en-us/library/ms723115%28VS.85%29.aspx>
* For more open source examples on using ADO.NET, look here: <http://adonetsamples.codeplex.com/>
* A useful collection of resources regarding ADO.NET development can be found here: <http://blogs.msdn.com/cfs-file.ashx/__key/CommunityServer-Blogs-Components-WeblogFiles/00-00-00-48-03/0714.ADO.NET-Developer-Guidance-Map-_2D00_-v1.pdf>

|  |
| --- |
| A great source of information on how to write code is the existing Test Cases Suite for the driver. It contains 70+ automated test cases, covering many usage scenarios and showing how you can use the CUBRID specific extensions. This code resource can be found in the ***TestCases.cs***file (<http://svn.cubrid.org/cubridapis/adodotnet/branches/RB-9.1.0/Data/Source/Unit.TestCases/>). |

# Links & Resources

|  |  |
| --- | --- |
| Driver Wiki page | <http://www.cubrid.org/wiki_apis/entry/cubrid-ado-net-driver> |
| CUBRID Tutorials | <http://www.cubrid.org/wiki_apis/entry/cubrid-ado-net-tutorials> |
| MSDN ADO.NET resources | <http://msdn.microsoft.com/en-us/library/h43ks021%28v=vs.71%29.aspx> |
| SVN project location | <http://svn.cubrid.org/cubridapis/adodotnet/branches/RB-9.1.0/> |
| JIRA Issue tracker | <http://jira.cubrid.org/browse/APIS/component/10404> |
| Tutorial Source Code | You will find in the archive the full source code for the tutorial application. |

This concludes the**CUBRID ADO.NET Data Provider**tutorial.

Remember, please let us know your feedback – we highly appreciate your feedback - and remember to periodically check the CUBRID web site ([www.cubrid.org/tutorials](http://www.cubrid.org/tutorials))for moreCUBRID tutorials and resources.

**Thank you!**

The CUBRID Apps&Tools Team