cubeSQL

REAL Studio Plugin

CubeSQL REAL Studio plugin inherits directly from the Database class so all the methods and properties of the Database class can be used directly from within the CubeSQL plugin. More informations about the Database class can be found online at:
http://docs.realsoftware.com/index.php/Database_Class

Standard Prepare method is not supported by CubeSQL plugin and it has been replaced by the VMPrepare method. Additional properties and methods are listed below.

CubeSQLPlugin module:

- SSLLibrary As FolderItem
- CryptoLibrary As FolderItem
- Version As String
- kAESNONE (Integer Constant)
- kAES128 (Integer Constant)
- kAES192 (Integer Constant)
- kAES256 (Integer Constant)
- kSSL (Integer Constant)

CubeSQL Class additional properties:

- ServerVersion As String
- Port As Integer
- Timeout As Integer
- AutoCommit As Integer
- Encryption As Integer
- EndChunk As Boolean
- UseREALServerProtocol As Boolean
- IsEndChunk As Boolean
- PingFrequency As Integer
- SSLCertificate As FolderItem

CubeSQL Class Additional methods:

- Connect() as Boolean
- IsConnected() as Boolean
- LastRowID() as Int64
- SendChunk(chunk as String)
- ReceiveChunk() as String
- SendEndChunk()
- SendAbortChunk()
- Ping() as Boolean
- VMPrepare(sql as String) as CubeSQLVM

- TableName(rs As RecordSet) As String

CubeSQLVM methods:

- BindInt(index As Integer, value As Integer)
- BindDouble(index As Integer, value As Double)
- BindText(index As Integer, value As String)
- BindBlob(index As Integer, value As String)
- BindInt64(index As Integer, value As Int64)
- BindNull(index As Integer)
- BindZeroBlob(index As Integer, length As Integer)
- VMExecute()
- VMSelect() As RecordSet

Please take a look at the examples contained in the REAL Studio folder (inside the Clients folder) in order to better understand how to correctly use the classes, properties and the plugin.

To use the new plugin just instantiate it with:
Dim db **as New** CubeSQLServer

When you install the cubeSQL plug-in in REALbasic's plugins folder, the cubeSQLServer class is added to REALbasic's framework. With it, you can develop custom cubeSQL applications. The cubeSQLServerPlugin requires REALbasic 2005 Professional, Release 1 or later.

Please report your feedback, your impressions or your bug reports directly to me at:
marco@sqlabs.com

MVCC

In order to increase concurrency cubeSQL supports MultiVersion Concurrency Control (MVCC), a standard technique for avoiding conflicts between reads and writes of the same object. MVCC guarantees that each transaction sees a consistent view of the database.

MVCC is a fairly common technique in database implementation and all the modern DBMS adopt an implementation of the MVCC algorithm developed by Jim Starkey while he was working at DEC (he is the database architect who developed InterBase, the first relational database to support multi-versioning and he is currently working for MySQL AB).

Implements a complete MVCC algorithm inside a database adds a big overhead so a lot of DBMS uses a sort of relaxed version in order to achieve better performance. cubeSQL uses an optimistic variant of MVCC that provides execution time consistency. (PostgreSQL uses locks -- a pessimistic scheme.) Instead of raising a ReadConflictError to signal a consistency problem, cubeSQL automatically reads non-current data that provides consistency, in other words when MVCC is ON read operations are always UNCOMMITTED.

When MVCC is ON server is able to offer a much better concurrency in situations where there is a very large number of concurrent writers. In general Write operations are slower when MVCC is ON and Read is always UNCOMMITTED. If you are upgrading from REAL Server 2009/2010 you probably want MVCC to be turned ON.

When MVCC is OFF server works in safer mode. The behavior is identical to the one offered by the standard sqlite engine and Write operations are always performed at full speed. Read operations are COMMITTED so for some applications this could be the preferred and safer behavior. If you are upgrading from REAL SQL Server 2008 you probably want MVCC to be turned OFF.

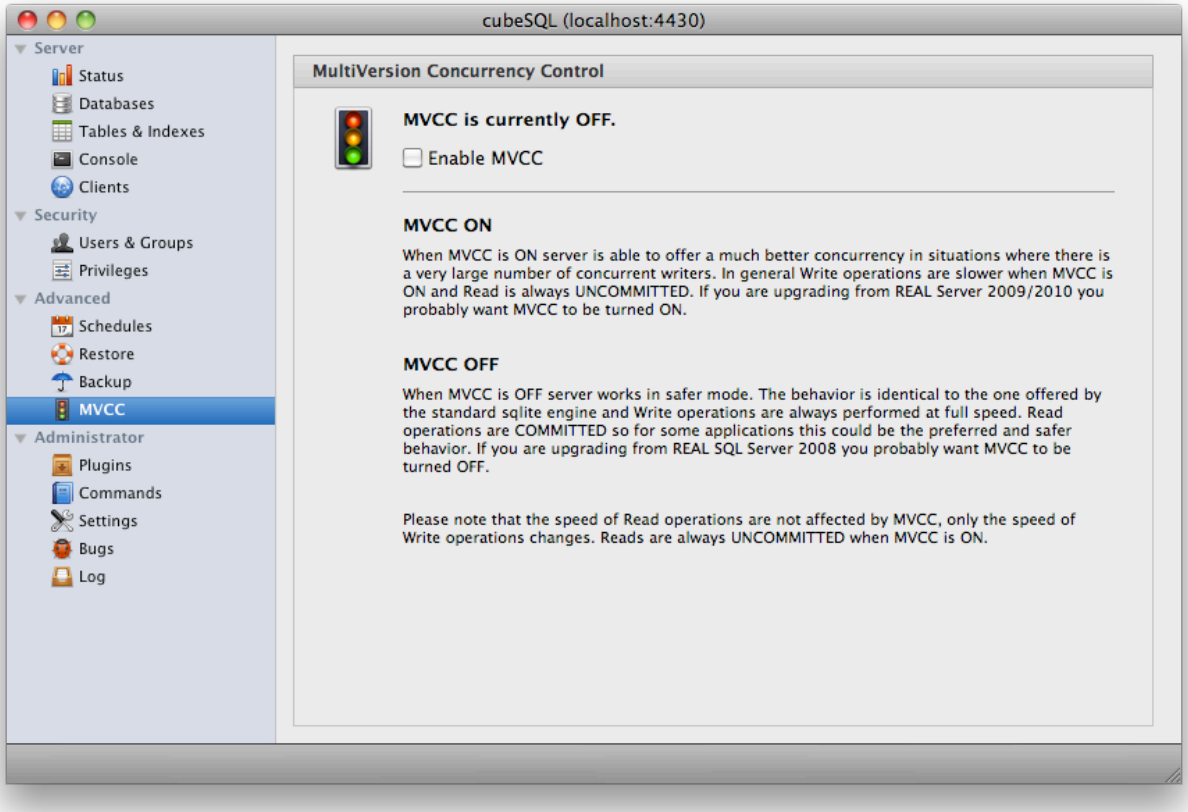
There are several MVCC related custom commands, one to query for its status:

```
SHOW MVCC
```

and two to enable/disable it:

```
ENABLE MVCC
DISABLE MVCC
```

You can run these commands from REAL Studio, or using the C SDK or PHP or any other supported client. An easy way to play with the MVCC is to just use the graphical interface provided by the Admin application:



REALbasic example
This example enabled MVCC on the server.

```
db = New CubeSQLServer
db.Host = "localhost"
db.port = 4430
db.UserName = "admin"
db.Password = "admin"

If (db.connect = false) then
    MsgBox "Connect error: " + db.ErrorMessage
    return
end if

db.SQLExecute("ENABLE MVCC;")
If db.error then MsgBox "An error occurred: " + db.ErrorMessage
```

JSON

In order to try to supports as much heterogeneous clients as possible cubeSQL fully supports the JSON open standard protocol. JSON is a lightwave text based protocol and is built-into any major language (like PHP, Ruby, LiveCode and so on).

For a complete and working JSON implementation we strongly suggest you to take a look at the cubeSQLServer.php class.

For basic operations are supported by our JSON implementation, connect, execute, select and disconnect.

CONNECT

```
{
     "command":"CONNECT",
     "username":"VAR1",
     "password":"VAR2",
     "randpool":"12345"
}
```

This is the first command that is required in order to open a JSON connection with the server.

randpool is any random integer array

username is encoded into VAR1:
VAR1 is SHA1_HEX(randpool + username)
+ is the string concatenation symbol
SHA1_HEX is SHA1 computed in hex mode

password in encoded into VAR2:
VAR2 is SHA1_HEX(randpool + BASE64(SHA1(SHA1(password))))
+ is the string concatenation symbol
SHA1_HEX is SHA1 computed in hex mode

In case of error any JSON command returns:

```
{
     "errorCode":"7047",
     "errorMsg":"An error occurred..."
}
```

In case of a successful execution errorCode is set to 0.

EXECUTE

This command executes an sql statement on the server:

```
{
     "command":"EXECUTE",
     "sql":"UPDATE foo SET col1='test';"
}
```

SELECT

This command executes an sql query on the server and returns a cursor using the JSON protocol:

```
{
     "command":"SELECT",
     "sql":"SELECT * FROM foo;"
}
```

DISCONNECT

This command close current connection with the server:

```
{
     "command":"DISCONNECT"
}
```