cubeSQL

ReadMe

© 2005-2011 SQLabs, All rights reserved.

cubeSQL is a fully featured and high performance relational database management system (RDBMS) built on top of the sqlite database engine. It's incredible fast, has a small footprint and it is very scalable. It can runs on Windows, Linux and MacOS X in both 32bit or 64bit versions.

Some features includes:

- Multi-core and multiprocessor aware.
- Strong AES encryption (128, 192 and 256 bit).
- Supports unlimited connections (For each supported operating system, cubeSQL uses a state of the art event API, kqueue on Mac OS X, epool on Linux and I/O Completion Ports on Windows).
- Full ACID (Atomic, Consistent, Isolated, Durable) compliant.
- Platform independent storage engine.
- Full support of triggers and transactions.
- Journal engine for crash recovery.
- Supports databases of 2 terabytes.
- Supports sqlite 3 databases.
- Automatic logging.
- Automatic compression.
- Multiversion concurrency control (MVCC).
- Plugins for extending the SQL language and the custom commands supported by the server.
- Restore and backup support.
- Mac OS X, Windows and Linux support.
- Native 32bit and 64bit supports.

... and much more

cubeSQL can be access by:

- REALbasic
- PHP
- C/C++/ObjC (whit the C SDK)
- DLL
- ODBC (available soon)
- any JSON client

Read the 5 Minutes Guide in order to be able to use cubeSQL is just few minutes and then read the Admin Manual in order to fully understand how to administer and register your server, finally read the Language Reference in order to know all the custom commands recognized by the server.

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.

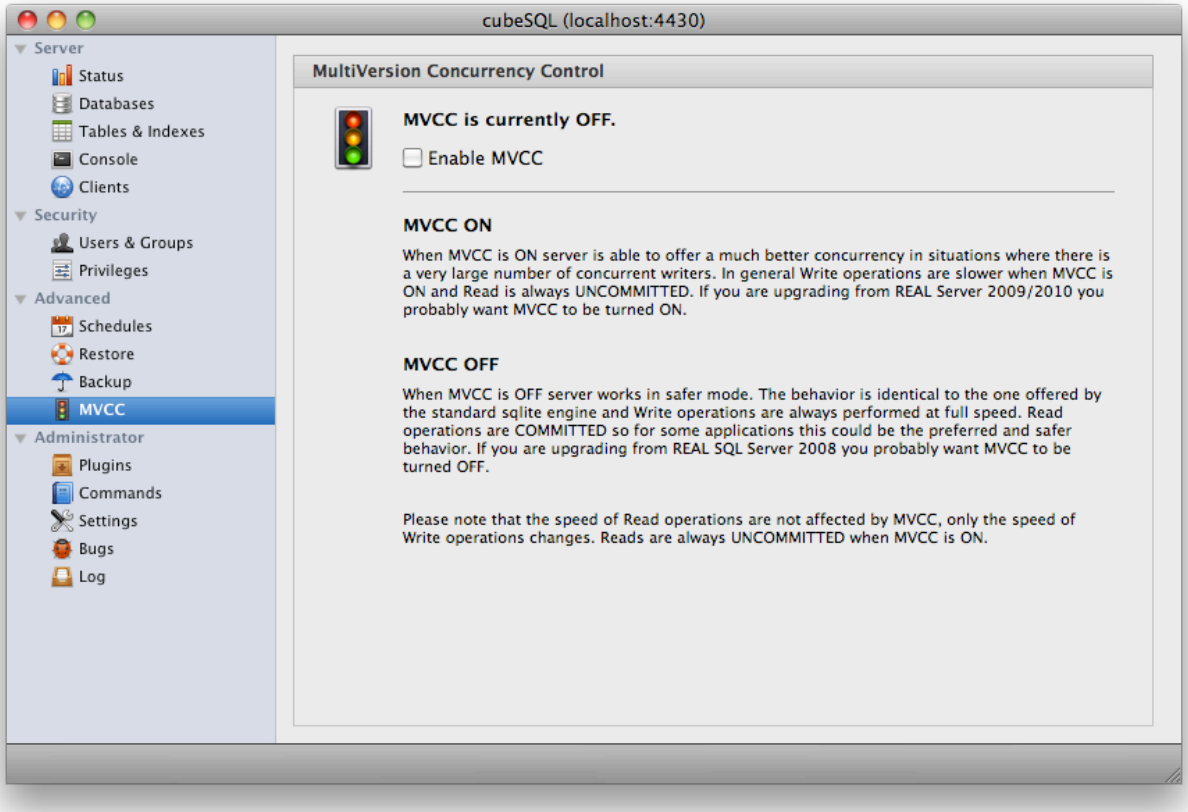
There are several MVCC related custom commands, one to query for its status:

```
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"
}
```