PowerShell LiteDB

Disclaimer

This SOFTWARE PRODUCT is provided by El Condor "as is" and "with all faults." El Condor makes no representations or warranties of any kind concerning the safety, suitability, lack of viruses, inaccuracies, typographical errors, or other harmful components of this SOFTWARE PRODUCT. There are inherent dangers in the use of any software, and you are solely responsible for determining whether this SOFTWARE PRODUCT is compatible with your equipment and other software installed on your equipment. You are also solely responsible for the protection of your equipment and backup of your data, and El Condor will not be liable for any damages you may suffer in connection with using, modifying, or distributing this SOFTWARE PRODUCT.

You can use this **SOFTWARE PRODUCT freely, if you would you can credit me in program comment:**

El Condor – CONDOR INFORMATIQUE – Turin

Comments, suggestions and criticisms are welcomed: mail to rossati@libero.it

Conventions

Commands syntax, instructions in programming language and examples are with font `COURIER NEW`. The optional parties of syntactic explanation are contained between `[square parentheses]`, alternatives are separated by `|` and the variable parties are in *`italics`*.

Contents table

1 LiteDB

LiteDB is a NoSQL database composed by a single DLL; LiteDB is similar to MongoDB.
LiteDB stores data as documents, which are JSON-like objects containing key-value pairs. Each document stores both its data and its structure[1]. The stored documents are called collections id est a group of related documents that have a set of shared fields; collections are analogous to the tables of relational databases.

2 Using LiteDB with Powershell

The use of LiteDB with Powershell is simple and it is based on tell Powershell where find the `LiteDB.dll` by means of `Add-Type` cmdlet like:

```
add-type -path "C:\Program Files\PowerShell\7\LiteDB.dll"
```

The `LiteDB.dll` exposes methods for manage the Data Base; this documentation is related to the use of methods to operate on the Data Base through SQL commands processed by the Powershell module `liteDB.psm1`.

2.1 The `liteDB.psm1` module

The module exposes the function `LiteDB`: `function liteDB($parm,$db = $null) {...` where `$parm` is the command and `$db` is the handle of the Data Base.

2.1.1 Open Data Base

This command must be before the other commands.

`$parm`	`Open` *`DataBase`*
`$db`	not necessary
return:	the handle to Data Base
Examples	`$db = liteDB "Open Quotes.db" # (Windows)` `$db = liteDB "Open powershell/Quotes.db" # (VM Ubuntu)`*

* I don't know why in Linux the default folder is `home/`*`user`* and not `home/`*`user`*`/powershell` that contains the Data Base and where the scripts are executed.

2.1.2 List of field

`$parm`	`List` *`FieldName`* `FROM` *`collection`*
`$db`	the handle received from the `Open` command.
return	ordered array of the field

Example	`$sql = "LIST Author FROM Quotes"`
	`$a = liteDB $sql $db`
	`Foreach ($i in $a) {`
	`  [Console]::WriteLine($i)`
	`}`

Note:	This command is translated to the SQL statement: `Select @key AS` *`FieldName`* `FROM` *`collection`* `GROUP BY` *`FieldName`*

2.1.3 Insert a row

`$parm`	`INSERT INTO` *`collection`* `VALUES (`*`json`*`)`
`$db`	the handle received from the `Open` command.
return	object `BsonDataReader`, the property current contains the number of row affected.
example	`$values = @{Author = "Unknown"; Where ="Where"; Quote = '{0:yyyyMMdd_HHmmss_ffff}' -f (Get-Date)}`

[1] Quoted from official site: https://www.litedb.org/docs/data-structure/.

```
$sql = "INSERT INTO Quotes VALUES " + ($values | ConvertTo-Json)
$p = liteDB $sql $db
```

2.1.4 Delete row(s)

`$parm`	`DELETE` *`collection`* `[WHERE ...]`
`$db`	the handle received from the `Open` command.
return	object `BsonDataReader`, the property `current` contains the number of row affected.
example	`$sql = "DELETE Quotes WHERE Author = 'Unknown'"` `$p = liteDB $sql $db`
note	`DELETE` command doesn't contains `FROM` keyword.

2.1.5 Update row(s)

`$parm`	`UPDATE` *`collection`* `SET` *`fieldName`* `=` *`value`*`[,`*`fieldName`* `=` *`value`*`,...]` `[WHERE ...]`
`$db`	the handle received from the `Open` command.
return	object `BsonDataReader`, the property `current` contains the number of row affected.
example	`$sql = "UPDATE QUotes SET TimeStamp = '" + '{0:yyyyMMdd_HHmmss_ffff}' -f (Get-Date) + "' WHERE Author LIKE '%Smith%'"` `$p = liteDB $sql $db`
note	the *`fieldName`* may be missing in the modified row and is then added to the row itself.

2.1.6 Select

`$parm`	`SELECT` *`field`*`[,`*`field`*`] ... FROM` *`collection`* `[WHERE ...] [ORDER BY ...]` `{LIMIT` *`n`*`]`
`$db`	the handle received from the `Open` command.
return	Array of rows as hash data
example	`"SELECT _id, TimeStamp, Author, Quote FROM Quotes WHERE Author LIKE '%Smith%' LIMIT 3"` `$p = liteDB $sql $db`

2.1.7 Close database

`$parm`	`Close`
`$db`	the handle received from the `Open` command.

3 Technical notes

```
PS C:\Sviluppo\PowerShell> $reader | get-member

   TypeName: LiteDB.BsonDataReader

Name        MemberType            Definition
----        ----------            ----------
Dispose     Method                void Dispose(), void
IDisposable.Dispose()
Equals      Method                bool Equals(System.Object obj)
GetHashCode Method                int GetHashCode()
GetType     Method                type GetType()
Read        Method                bool Read(), bool
IBsonDataReader.Read()
ToString    Method                string ToString()
Item        ParameterizedProperty LiteDB.BsonValue Item(string
field) {get;}
Collection  Property              string Collection {get;}
Current     Property              LiteDB.BsonValue Current {get;}
HasValues   Property              bool HasValues {get;}

PS C:\Sviluppo\PowerShell> $reader.Collection
Quotes
```

4 History

0.1.0		First release